

# RAZVOJ MOBILNIH APLIKACIJA

PRIRUČNIK ZA 4. RAZRED GIMNAZIJE

AUTORI:

ZLATKO STAPIĆ, IVAN ŠVOGOR i DAVOR FODREK

VARAŽDIN, 2016.

Priručnik je izrađen u sklopu projekta „HEUREKA – spoznajom do uspjeha“ kojeg je financirala Europska unija.



Projekt je financirala Europska unija u 100%-om iznosu iz Europskog socijalnog fonda kroz Operativni program „Razvoj ljudskih potencija 2007.-2013., poziv na dostavu projektnih prijedloga HR.3.1.20 Promocija kvalitete i unaprjeđenje sustava odgoja i obrazovanja na srednjoškolskoj razini.

Sadržaj ove publikacije / emitiranog materijala isključiva je odgovornost Srednje škole Ivanec

**SREDNJA ŠKOLA IVANEC – nositelj projekta**

Ravnateljica: mr.sc. Lidija Kozina dipl.oec

Eugena Kumičića 7, 42 240 Ivanec

Telefon: 042 782 344; Faks: 042 781 512

E-mail: [info@ss-ivanec.hr](mailto:info@ss-ivanec.hr)

Web: <http://www.ss-ivanec.hr/>

**SREDNJA ŠKOLA MATE BLAŽINE LABIN – partner na projektu**

Ravnatelj: Čedomir Ružić, prof.

Rudarska 4, 52 220 Labin

Telefon: 052 856 277; Faks: 052 855 329

E-mail: [ssmb@ss-mblazine-labin.skole.hr](mailto:ssmb@ss-mblazine-labin.skole.hr)

Web: <http://www.ssmb.hr>

**Posredničko tijelo razine 1**

Ministarstvo znanosti, obrazovanja i sporta

Ulica Donje Svetice 38, 10000 Zagreb

E-mail: [esf@mzos.hr](mailto:esf@mzos.hr)

Web: <http://public.mzos.hr>

**Posredničko tijelo razine 2**

Agencija za odgoj i strukovno obrazovanje i obrazovanje odraslih, Organizacijska jedinica za upravljanje strukturnim instrumentima

Radnička cesta 37b, 10000 Zagreb

E-mail: [defco@asoo.hr](mailto:defco@asoo.hr)

Web: <http://www.asoo.hr/defco>

**Za više informacija o EU fondovima u RH:**

[www.mrrfeu.hr](http://www.mrrfeu.hr), [www.strukturnifondovi.hr](http://www.strukturnifondovi.hr)

**Autori:** Zlatko Stapić, Ivan Švogor i Davor Fodrek

**Nakladnik:** Sveučilište u Zagrebu, Fakultet organizacije i informatike  
Pavlinska 2, 42000 Varaždin

**Za nakladnika:** Prof.dr.sc. Neven Vrčec, dekan

ISBN 978-953-6071-54-8

CIP zapis je dostupan u računalnome katalogu Nacionalne i sveučilišne knjižnice u Zagrebu pod brojem 000942866.

## PREDGOVOR

Trenutno svjedočimo sve češćoj primjeni pametnih, ugrađenih i povezanih uređaja, integraciji interneta stvari i interneta svega u mobilne aplikacije, primjeni virtualne, ali i proširene stvarnosti, te “oživljavanju” različitih predmeta koji su sada u stanju “razgovarati” s vašim mobitelom, kao što su na primjer igračke za djecu, sportska oprema, ali i automobili, stambeni prostori i slično. Spomenute tehnologije toliko su integrirane u našu svakodnevicu da su postale neizostavnim dijelom obavljanja svakodnevnih životnih aktivnosti poput plaćanja računa, mrežnoga pretraživanja, razmjene informacija ili razonode.

Industrija informacijsko-komunikacijskih tehnologija (IKT) trenutno je jedna od najbrže rastućih industrija u Hrvatskoj. Ova industrija, kojoj pripada i razvoj mobilnih proizvoda, od 2013. godine, a posebno tijekom 2015. godine, bilježi ubrzan oporavak u smislu broja zaposlenih i ukupnog udjela u izvozu proizvoda i usluga. Istraživanje potreba poslodavaca na lokalnoj razini također je pokazalo da postoji nedostatak stručnjaka sa znanjima u navedenom području te postoji stalna potreba za kadrom sa suvremenim IT kompetencijama u području mobilnih tehnologija i razvoja softvera.

Stoga, ovaj priručnik za fakultativni predmet *Razvoj mobilnih aplikacija*, ima za cilj omogućiti učenicima 4. razreda opće gimnazije uvid u ovo izazovno područje te im pružiti temelje i motivirati ih za daljnje usavršavanje. Specifični ciljevi priručnika su omogućiti učenicima usvajanje temeljnih znanja o osmišljavanju programskoga proizvoda i njegove arhitekture, objektno orijentiranom pristupu razvoja aplikacija, stjecanje vještina za rad u integriranom razvojnom okruženju, upoznavanje elemenata korisničkoga sučelja Android aplikacije kao i korištenje web servisa i biblioteka treće strane u razvoju aplikacija. Ovaj priručnik naglasak stavlja na razvoj mobilnog softvera te ne prikazuje koncepte koji su karakteristični za razvoj drugih programskih proizvoda.

Konačno, cilj priručnika je utvrditi temelje u domeni programske logike i razumijevanja složenih problema koje predstavlja okruženje u kojima se rabi mobilna aplikacija, kao i mogućnost samostalnoga djelovanja i rješavanja izazova u nepredviđenim okolnostima. Poznavanjem temeljnih informatičkih koncepata kao što su programiranje, algoritmi ili strukture podataka, učenik ne ostaje samo korisnik informacijsko-komunikacijske tehnologije već i njezin stvaratelj.

Priručnik je podijeljen u dvije osnovne cjeline. Prva cjelina donosi poglavlja koja se odnose na teorijske koncepte razvoja mobilnih aplikacija, a druga cjelina prikazuje cjelokupni proces razvoja mobilne aplikacije *Memento* vodeći učenika kroz cjelokupni proces *korak po korak*.

*Autori*



## SADRŽAJ PRIRUČNIKA

Predgovor .....	A
Sadržaj priručnika .....	1
1 Uvod u mobilni razvoj .....	5
1.1 Razrada projektne ideje .....	6
1.1.1 Odabir domene razvoja .....	6
1.1.2 Analiza navika potrošača i promjena navika .....	6
1.1.3 Analiza tehnoloških trendova i novih tehnologija .....	7
1.1.4 Odabir segmenta ciljanih korisnika .....	7
1.1.5 Definiranje potencijalnih ideja .....	7
1.1.6 Analiza konkurentskih proizvoda .....	8
1.1.7 Pronalazak vodeće funkcionalnosti .....	8
1.1.8 Konačni odabir projektne ideje .....	9
1.2 Proces razvoja mobilnih aplikacija .....	10
1.2.1 Faze procesa razvoja .....	10
1.2.2 Klasifikacija metodika razvoja.....	11
1.2.3 Klasifikacija pristupa razvoju .....	12
1.3 Scrum proces razvoja .....	14
1.3.1 Osnovni Scrum koncepti.....	15
1.3.2 Prioritetna lista funkcionalnosti .....	15
1.3.3 Sprint .....	16
1.3.4 Planiranje sprinta.....	17
1.3.5 Dnevni Scrum.....	18
1.3.6 Ažuriranje liste zadataka i grafa odrađenog posla .....	18
1.3.7 Sprint retrospektiva.....	19
1.4 Projektni tim i uloge .....	20
1.4.1 Voditelj proizvoda.....	21
1.4.2 Dizajner korisničkog sučelja.....	21
1.4.3 Razvojni inženjer.....	22
1.4.4 Tester.....	23
1.4.5 Ostale uloge.....	23
1.5 Pitanja za provjeru znanja .....	26
1.6 Resursi za samostalan rad .....	26
2 Objektno orijentirani pristup razvoju .....	27
2.1 Uvod u objektno orijentirano programiranje.....	28

2.2	Java programski jezik.....	28
2.3	Tipovi podataka .....	29
2.3.1	Deklaracija, inicijalizacija i definicija varijabli .....	29
2.3.2	Numerički tipovi podataka .....	30
2.3.3	Operacije nad numeričkim tipovima .....	32
2.3.4	Evaluiranje izraza .....	32
2.3.5	Prelijevanje vrijednosti .....	33
2.3.6	Pretvorba tipova podataka.....	33
2.3.7	Znakovni tipovi podataka .....	33
2.4	Logičke strukture .....	34
2.4.1	Logika vođena događajima.....	34
2.4.2	Slijed .....	36
2.4.3	Selekcija .....	37
2.4.4	Iteracija.....	40
2.5	Metode i svojstva .....	42
2.6	Objektno orijentirano programiranje.....	44
2.6.1	Ciljevi OOP-a .....	44
2.6.2	Koncept objekta i razumijevanje OOP-a.....	45
2.6.3	Učahurivanje.....	46
2.6.4	Nasljeđivanje .....	47
2.6.5	Polimorfizam .....	49
2.6.6	Sučelje.....	51
2.7	Pitanja za provjeru znanja .....	54
2.8	Resursi za samostalan rad .....	54
3	Razvoj Android aplikacija .....	55
3.1	Integrirano razvojno okruženje .....	56
3.1.1	Klasifikacija razvojnih okruženja.....	56
3.1.2	Android Studio.....	57
3.1.3	Kreiranje novog projekta .....	59
3.2	Android SDK.....	61
3.2.1	Aplikacijski stog .....	61
3.2.2	Struktura SDK-a .....	62
3.2.3	Instaliranje Android API-a.....	63
3.3	Programska logika Android aplikacije .....	64
3.3.1	Android aktivnosti .....	65
3.3.2	Resursi .....	67
3.3.3	Kvalifikatori resursa.....	68

3.3.4	Android Manifest.....	69
3.3.5	Gradle .....	70
3.3.6	Namjera (Intent).....	71
3.4	Alati Android Studija .....	72
3.4.1	Alati za razvoj.....	72
3.4.2	Alati za otkrivanje pogrešaka .....	74
3.4.3	Ostali važni alati.....	75
3.5	Pitanja za provjeru znanja .....	77
3.6	Resursi za samostalan rad .....	77
4	Rad s podacima .....	79
4.1	Pohrana podataka .....	80
4.1.1	Mogućnosti pohrane podataka .....	81
4.1.2	Ključ-vrijednost parovi.....	81
4.1.3	Korištenje parova ključ-vrijednost putem programskog kôda .....	81
4.1.4	Korištenje parova ključ-vrijednost prema preddefiniranim pravilima .....	83
4.2	Zapis podataka u datoteke .....	85
4.2.1	Primjer zapisivanja tekstualne datoteke na internu memoriju.....	87
4.2.2	Čitanje sadržaja datoteka .....	88
4.2.3	Primjer zapisivanja podataka na vanjsku memoriju.....	89
4.2.4	Brisanje datoteka.....	91
4.3	Rad sa mobilnom bazom podataka .....	92
4.3.1	Analiza potrebe zapisa podataka za CookBook mobilnu aplikaciju .....	92
4.3.2	Nativni pristup .....	95
4.3.3	ORM – Active Android .....	99
4.3.4	Kreiranje baze podataka putem Active Android biblioteke .....	100
4.4	Rad s web servisima .....	106
4.4.1	SOAP .....	107
4.4.2	REST .....	107
4.4.3	Korištenje REST servisa kod Androida .....	108
4.4.4	Retrofit i GSON .....	109
4.5	Pitanja za provjeru znanja .....	117
4.6	Resursi za samostalan rad .....	117
5	Primjer razvoja mobilne aplikacije.....	119
5.1	Memento.....	120
5.2	Kreiranje projekta.....	121
5.3	Izrada pogleda .....	124

5.3.1	Priprema dizajna glavne aktivnosti za tabove .....	124
5.3.2	Uvoz potrebnih ikona u projekt.....	127
5.3.3	Programsko dodavanje tabova i ikona .....	129
5.3.4	RecyclerView, prvi dio – priprema .....	130
5.3.5	Postavke jezika i eksternalizacija teksta .....	135
5.3.6	RecyclerView, drugi dio – adapteri.....	138
5.4	Izrada entitetnih klasa .....	142
5.5	Unos i prikaz podataka – rad s dijalozima i fragmentima pogleda .....	148
5.6	Mobilna baza podataka .....	154
5.6.1	Unos, pohrana i prikaz podataka.....	161
5.7	Korištenje web servisa.....	169
5.7.1	Priprema za ispis podataka.....	170
5.7.2	Implementacija infrastrukture za rad sa web servisima .....	174
5.8	Pitanja za provjeru znanja .....	180
5.9	Resursi za samostalan rad .....	180
	Rječnik stručnih pojmova .....	183
	Popis kratica i akronima .....	187
	Korištena literatura .....	189



# 1 UVOD U MOBILNI RAZVOJ

*Razvoj mobilnog softvera (eng. mobile software development) čini složen skup aktivnosti koje provode članovi projektnog tima u svrhu osmišljavanja, izrade, puštanja u rad te održavanja programskog proizvoda. Stoga, uvodno poglavlje ovog priručnika ima za cilj predstaviti cjelokupnu sliku mobilnog razvoja kroz prikaz osnovnih pojmova kroz svaku od faza razvoja. Proces razvoja mobilnih aplikacija, promatran van konteksta poslovnog sustava, započinje fazom razrade projektne ideje i definiranjem funkcionalnosti budućeg programskog proizvoda, nakon koje slijedi dizajn korisničkog sučelja (izgleda) proizvoda, dizajn arhitekture i strukture proizvoda, izrada (programiranje), testiranje i konačno puštanje u rad i održavanje. U ovom poglavlju ukratko su opisani svi navedeni koraci te uloge članova tima.*

## SADRŽAJ POGLAVLJA

Razrada projektne ideje .....	6
Proces razvoja mobilnih aplikacija .....	10
Scrum proces razvoja .....	14
Projektni tim i uloge .....	20
Dodatni resursi .....	26

## 1.1 Razrada projektne ideje

Otvaranje mobilnih platformi, dostupnost materijala za samo-učenje, jednostavna distribucija kreiranog softvera, kao i vrlo veliki broj mobilnih korisnika, kroz posljednjih 7 godina kreirali su izuzetno povoljnu klimu i uključivanje velikog broja tvrtki i razvojnih inženjera u utrku za kreiranjem različitog mobilnog sadržaja, a u svrhu osvajanja što većeg tržišnog udjela. Već u prvom kvartalu 2016. godine na Google Play<sup>1</sup> trgovini se nalazilo više od 2 milijuna<sup>2</sup> aplikacija za Android uređaje. Zbog toga je kvalitetna projektna ideja kao i kvalitetna razrada projektne ideje ključna za uspjeh programskog proizvoda na ovako dinamičnom i turbulentnom tržištu.

Dva su osnovna pristupa generiranju projektne ideje - nesistematski i sistematski pristup a brojni su kriteriji koji mogu utjecati na odabir jednog ili drugog pristupa.

- nesistematski pristup se temelji na brzom definiranju projektnih zadataka bez prethodnog istraživanja i segmentacije tržišta ili pozicioniranja u tržišnu nišu. Ovakav pristup karakterističan je za pojedince koji „imaju genijalnu ideju“ i žele ju što prije spremi za tržište. Naravno, korištenje nesistematskog pristupa nije preporučljivo, osim u slučaju kada se projektna ideja ne generira u svrhu ostvarivanja profita već u druge svrhe kao što su na primjer učenje ili razvoj za vlastite potrebe.
- sistematski pristup, s druge strane, podrazumijeva provođenje aktivnosti segmentacije tržišta i ciljanih korisnika u svrhu pronalaska tržišne niše koja još nije pokrivena i zasićena konkurentskim proizvodima. Ovaj pristup svakako ovisi i o domeni poslovanja tvrtke ili želji pojedinca da se bavi određenim područjem. Neke od tržišnih niša koje još uvijek nisu zasićene su na primjer zdravlje, ekologija, energetska učinkovitost, poljoprivreda i slično. Sistematski pristup generiranju projektne ideje uključuje provedbu aktivnosti opisanih u sljedećim poglavljima.

### 1.1.1 Odabir domene razvoja

Obično se prije samog procesa generiranja projektne ideje tvrtka već odlučila za razvoj mobilne aplikacije ili mobilne igre. U ovom poglavlju, a i u ostalim poglavljima priručnika, promatrat ćemo samo aktivnosti koje se odnose na razvoj mobilnih aplikacija, te razvoj mobilnih igara neće biti u našem fokusu. Ukoliko tvrtka već nije specijalizirana za razvoj mobilnih aplikacija u specifičnom području (domeni), onda je prva odluka koju treba donijeti upravo tržišno pozicioniranje i odabir domene razvoja. Pozicioniranje u bilo koju domenu zahtjeva posjedovanje domenskog znanja, što ovisno o ekspertizi uposlenika i suradnika tvrtke, može biti prednost ili nedostatak. Po rezultatima anketiranja<sup>3</sup> razvojnih inženjera i tvrtki može se zaključiti da je izrada kvalitetnih rješenja u jednoj domeni bolja strategija od pokrivanja različitih domena različitim proizvodima. Neke od tržišnih niša koje još uvijek nisu zasićene mogu biti *zdravlje*, *ekologija*, *energetska učinkovitost*, *poljoprivreda* i slično. Rezultat ove aktivnosti je definirana domena razvoja mobilnog proizvoda.

### 1.1.2 Analiza navika potrošača i promjena navika

Proizvodi koji mijenjaju navike korisnika mogu naići na otpor njihovom prihvaćanju kod značajnog broja populacije. S druge strane, upravo takvi proizvodi koji nude nove mogućnosti postaju privlačni

<sup>1</sup> <https://play.google.com/>

<sup>2</sup> Izvor: Statista.com. Pristupano u lipnju 2016. na <http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/>

<sup>3</sup> Izvor: Research2Guidance – App Strategy Advisory & Market Research. <http://research2guidance.com/>

mlađim korisnicima koji lakše prihvaćaju promjene. Pravovremeno prepoznavanje novog trenda u navikama korisnika može donijeti značajnu konkurentsku prednost. Ponekad je u provedbu ove aktivnosti potrebno uključiti i vanjske stručnjake u području marketinga ili psihologije koji su specijalizirani za ovakve analize. Rezultat ove aktivnosti daje temelje za generiranje inovativnih ideja koje služe postojećim navikama ili pak ideja koje imaju za cilj kreirati nove trendove u odabranoj domeni razvoja.

### 1.1.3 Analiza tehnoloških trendova i novih tehnologija

Tehnološki ciklus, to jest vrijeme od pojave, preko prihvaćanja i konačno zasićenja pojedinom tehnologijom, u posljednjih 5 godina je višestruko skraćen. Iako brza tehnološka evolucija ima i nedostatke, ona donosi i brojne prednosti, a u kontekstu razvoja projektne ideje najvažnije je spomenuti, da pojava nove tehnologije „resetira“ tržište te ponovno otvara nove mogućnosti u svim domenama. Trenutno se nalazimo na početku životnog ciklusa više novih tehnologija od kojih valja izdvojiti proširenu stvarnost (eng. Augmented reality), Internet stvari (eng. Internet of things – IoT) i Internet svega (eng. Internet of Everything – IoE).

*Proširena stvarnost* predstavlja tehnološku inovaciju koja se odnosi na korištenje nosivih i mobilnih uređaja koji korisniku dodatnim informacijama (tekst, slika, zvuk, vibracija i slično) dopunjuju informacije koje prima iz realnog svijeta. Tako na primjer, dok promatrate turističku znamenitost nekoga grada, gledajući kroz naočale s proširenom stvarnošću (kroz koje znamenitost vidite vlastitim očima), vidite i virtualnog turističkog vodiča koji stoji pored znamenitosti i pojašnjava ju iako ga fizički tu nema.

S druge strane, *Internet stvari* predstavlja mrežu sačinjenu od fizičkih objekata (stvari, uređaja) stalno spojenih na Internet s kojeg mogu primiti i slati podatke. Najčešće se koristi kako bi se opisalo skup fizičkih uređaja koji imaju mogućnost prikupljati podatke (pomoću senzora) i slati ih u centralnu bazu ili pak prikazati rezultat njihove obrade. Tako na primjer, svaki puta kada se približite semaforu na kojem je trenutno crveno svjetlo, vaš pametni sat na ruci može uključiti vibraciju kako bi vas upozorio da stanete. U ovom primjeru, sat zna smjer vašeg kretanja, komunicira sa semaforom, obrađuje informacije o trenutno uključenim svjetlosnim signalima i po potrebi obavještava korisnika.

Za razliku od Interneta stvari, *Internet svega* predstavlja mrežu sačinjenu od fizičkih i nefizičkih objekata (stvari, uređaja, usluga, ljudi, računalnih agenata, ugrađenih uređaja....) stalno spojenih na Internet s kojeg mogu primiti i slati podatke. Ostale karakteristike su slične konceptu Internet stvari.

Rezultat aktivnosti analize tehnoloških trendova i novih inovacija daje uvid u nove tehnologije te popis onih novih tehnologija oko kojih se može graditi daljnje generiranje projektnih ideja u odabranoj domeni razvoja.

### 1.1.4 Odabir segmenta ciljanih korisnika

Ponekad je segment korisnika za koje želimo razviti aplikaciju ovisan o prethodno odabranoj domeni, a ponekad je potrebna dodatna analiza i odluka. Za pojedine skupine korisnika na tržištu još uvijek ne postoje kvalitetna rješenja koja zadovoljavaju njihove potrebe. Razmislite o mobilnim aplikacijama koje bi bila namijenjene isključivo *umirovljenicima, osobama s invaliditetom, vojnicima, liječnicima, djeci, adolescentima, samohranim roditeljima, policajcima* ili drugima. Rezultat ove aktivnosti je definiran segment korisnika za koje razvijamo projektno rješenje.

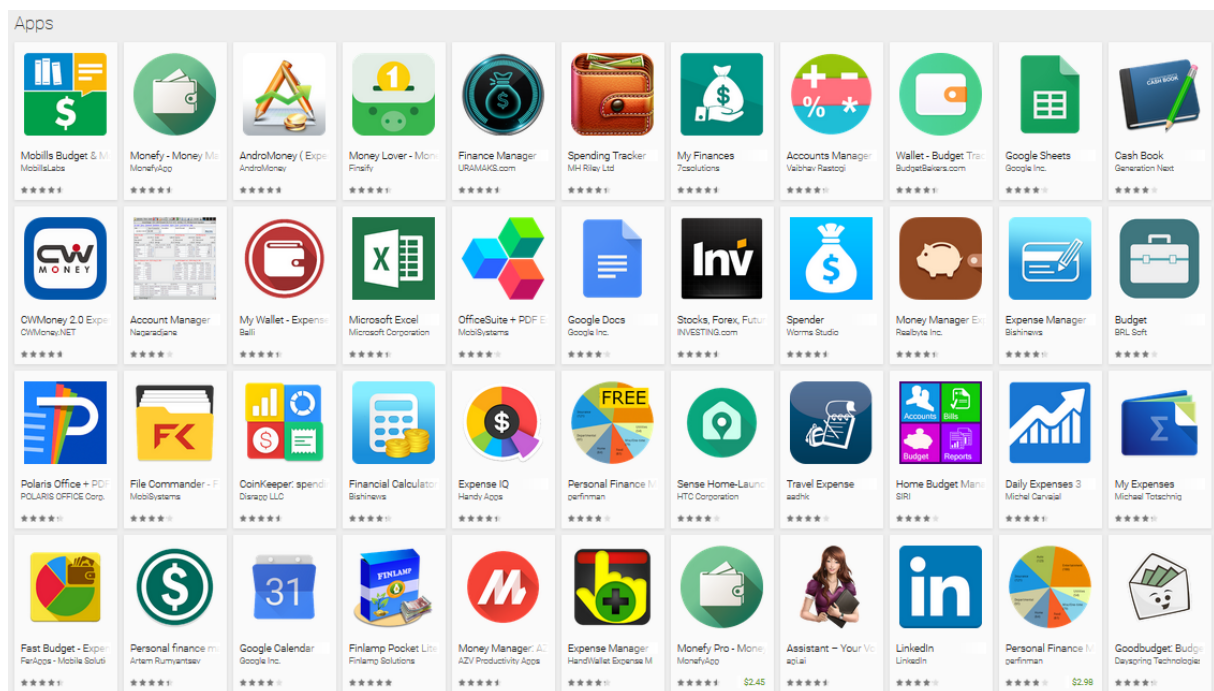
### 1.1.5 Definiranje potencijalnih ideja

Uzimajući u obzir utvrđene trendove, tehnologije, domenu razvoja i ciljne korisnike u ovom koraku je potrebno provesti aktivnost generiranja potencijalnih ideja. Svaka ideja se temelji na jednoj

ključnoj funkcionalnosti (eng. key feature) oko koje se definiraju pomoćne i dodatne funkcionalnosti. Postoje različite tehnike generiranja ideja, od koji su najpoznatije tehnike *oluje mozgova* (eng. Brain Storming). Tehnike iz obitelji oluje mozgova se temelje na brzom kreiranju velikog broja ideja od strane svih članova projektnog tima, a nakon toga se vrši selekcija i odabir uz moguće dopune i izmjene originalno kreiranih ideja. Rezultat ove aktivnosti je kreirana lista potencijalnih ideja za razvoj mobilnog proizvoda pri čemu je za svaku ideju definirana i ključna funkcionalnost.

### 1.1.6 Analiza konkurentskih proizvoda

Za svaku ideju koja je prošla proces selekcije potrebno je napraviti analizu konkurentskih proizvoda na tržištu te utvrditi njihove nedostatke, kritike korisnika i mogućnosti kreiranja boljeg i kvalitetnijeg rješenja. Analizom tržišta za, na primjer, ideju kreiranja mobilne aplikacije za evidenciju osobnih financija, lako se može utvrditi da trenutno postoji više od 30 postojećih rješenja s korisničkom bazom od ukupno više od stotinu milijuna korisnika (Slika 1). Međutim, detaljnijom analizom, također se može utvrditi da većina tih rješenja ne koristi nove tehnologije, nema kvalitetno riješenu sinkronizaciju između članova obitelji, nema višerazinsku klasifikaciju kategorija potrošnje te da svakako postoji mogućnost njihovog poboljšanja. Rezultat ove aktivnosti je popis svih prednosti i nedostataka konkurentskih proizvoda za svaku projektnu ideju koja je u užem izboru.



Slika 1. Dio rezultata pretraživanja postojećih aplikacija na temu osobnih financija

### 1.1.7 Pronalazak vodeće funkcionalnosti

Nastavno na aktivnost analize konkurentskih proizvoda, tijekom ove aktivnosti potrebno je definirati sve moguće funkcionalnosti promatranih projektnih ideja. Ključne funkcionalnosti i dio dodatnih funkcionalnosti su već definirani u prethodnom koraku, ali sada, nakon što imamo uvid u ostale proizvode na tržištu, potrebno je zaokružiti projektnu ideju popisom svih funkcionalnosti i uključenih tehnologija. Poseban naglasak treba staviti na definiranje vodeće (najvažnije) funkcionalnosti (eng. Kill feature), to jest funkcionalnosti koja čini razliku između novog proizvoda i ostalih proizvoda na tržištu. Primjer popisa funkcionalnosti prikazan je u tablici (Tablica 1).

Tablica 1. Definiranje funkcionalnosti

KONKURENTSKI PROIZVODI	NOVI PROIZVOD
<ul style="list-style-type: none"> <li>○ evidencija prihoda</li> <li>○ evidencija rashoda</li> <li>○ kategorizacija prihoda</li> <li>○ kategorizacija rashoda</li> <li>○ planiranje potrošnje po kategorijama</li> <li>○ ponavljajuće transakcije</li> <li>○ obavijesti o dospjelim obvezama</li> <li>○ izvješća</li> <li>○ sinkronizacija s više uređaja</li> </ul>	<ul style="list-style-type: none"> <li>✓ evidencija prihoda</li> <li>✓ evidencija rashoda</li> <li>✓ kategorizacija prihoda</li> <li>✓ kategorizacija rashoda</li> <li>✓ planiranje potrošnje po kategorijama</li> <li>✓ ponavljajuće transakcije</li> <li>✓ obavijesti o dospjelim obvezama</li> <li>✓ izvješća</li> <li>✓ sinkronizacija s više uređaja</li> <li>+ višerazinska kategorizacija prihoda</li> <li>+ višerazinska kategorizacija rashoda</li> <li>+ rotacijski grafovi</li> <li>+ analiza trendova</li> <li>+ inteligentna analiza moguće uštede</li> <li>+ optičko prepoznavanje (OCR) računa</li> <li>+ NFC sinkronizacija</li> <li>+ ...</li> </ul>

U navedenom primjeru možemo vidjeti kako je projektni tim odlučio zadržati funkcionalnosti koje postoje u konkurentskim proizvodima, ali je također definirao i set novih funkcionalnosti koje postojeći proizvodi ne sadržavaju. Vodeća funkcionalnost, koja bi trebala činiti najveću razliku na tržištu, te oko koje će se graditi i većina buduće marketinške kampanje i promocije proizvoda je mogućnost *inteligentne analize moguće uštede* kod korisnika. Na sličan način potrebno je definirati funkcionalnosti i za ostale projektne ideje.

#### 1.1.8 Konačni odabir projektne ideje

Uzimajući u obzir rezultate svih prethodnih koraka projektni tim donosi konačnu odluku te odabire proizvod s kojim će krenuti u proces implementacije. Neki od važnih faktora koje tijekom konačnog odabira treba uzeti u obzir su:

- ✓ znanje i iskustvo projektnog tima u odabranoj domeni i s odabranim tehnologijama
- ✓ planirano vrijeme do objave proizvoda na tržište (eng. Planned time to market)
- ✓ potrebni ljudski, materijalni i novčani resursi
- ✓ potencijalnu zaradu od projektne ideje
- ✓ zasićenost tržišta i očekivani trendovi u ponašanju korisnika
- ✓ broj tehnoloških oštrica u promatranom proizvodu

S posebnom pozornošću treba promotriti *tehnološke oštrice*. Pojam predstavlja elemente projektnog rješenja u kojima projektni tim nema iskustva, koji su potpuno novi na tržištu, neispitani i rizični, ili jednostavno dijelove programskog proizvoda koje je s trenutnom razinom tehnologije teško riješiti. Za primjer, *tehnološka oštrica* može biti zahtjev da se u realnom vremenu sinkroniziraju podaci između više mobilnih uređaja. Ovaj zahtjev uključuje integraciju više različitih tehnologija i svakom

projektnom timu predstavlja izazov koji se još uvijek ne može riješiti bez kašnjenja od jedne do nekoliko sekundi.

Tek nakon analize svih ovih faktora projektni menadžment ili projektni tim mogu odabrati ideju koja mora biti **motivirajuća za projektni tim** ali pri tome i **izvediva u planiranom vremenu i s planiranim resursima** i u konačnici **interesantna i korisna korisnicima**. Koristeći neku od tehnika oluje mozgova projektni tim može za odabranu projektnu ideju definirati naziv proizvoda. S konačno odabranom idejom projektni tim započinje proces razvoja programskog proizvoda provodeći aktivnosti koje su opisane u sljedećem poglavlju.

## 1.2 Proces razvoja mobilnih aplikacija

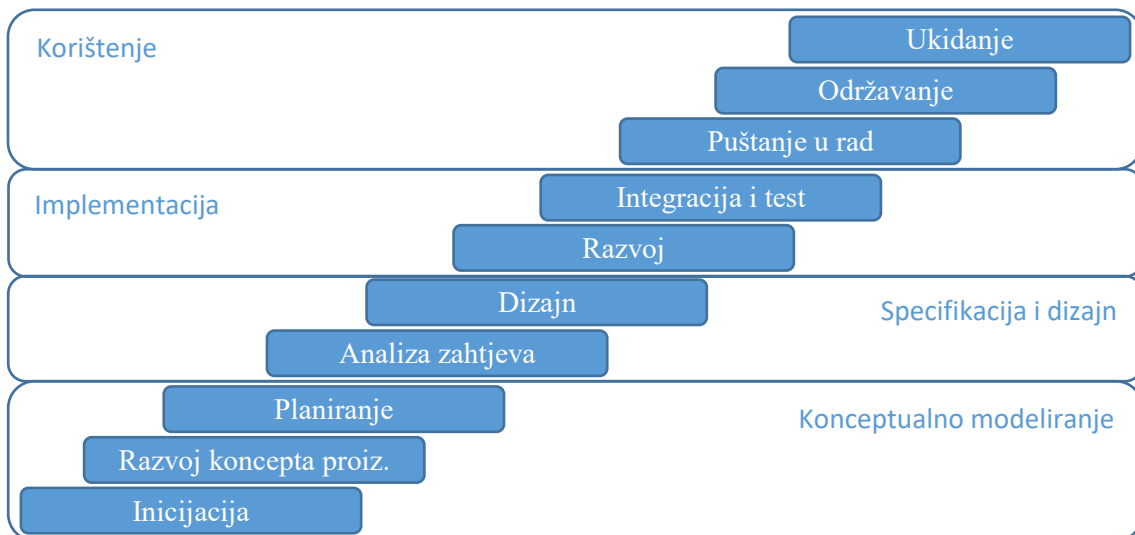
Razvoj mobilnog softvera (eng. Mobile software development) čini složen skup aktivnosti koje provode članovi projektnog tima u svrhu osmišljavanja, izrade, puštanja u rad te održavanja programskog proizvoda. Proces razvoja mobilnih aplikacija, promatran van konteksta poslovnog sustava, započinje fazom razrade projektne ideje i definiranjem funkcionalnosti budućeg programskog proizvoda, što je opisano u prethodnom poglavlju, a nakon koje slijedi dizajn korisničkog sučelja (izgleda) proizvoda, dizajn arhitekture i strukture proizvoda, izrada (programiranje), testiranje i konačno puštanje u rad i održavanje.

Osnovne definicije koje treba poznavati kako bi se razumio proces razvoja uključuju sljedeće pojmove:

- ✓ Proces razvoja programskog proizvoda – je niz aktivnosti koji pretvara korisničke zahtjeve u programski proizvod [1]. U procesu razvoja korisničke potrebe se pretvaraju u strukturirane korisničke zahtjeve; zahtjevi u jasnu specifikaciju dizajna, koji se potom implementira u programski kôd; programski kôd se testira, te potom isporučuje u obliku programskog proizvoda i pušta u uporabu.
- ✓ Metodika razvoja programskog proizvoda – predstavlja sistematiziran pristup u provedbi procesa razvoja. Prema SWEBOOK-u, odabir prikladne metodike može imati značajan utjecaj na uspjeh cjelokupnog projekta razvoja [2]. Metodiku razvoja programskog proizvoda možemo definirati kao okvir po kojem se strukturira, planira i kontrolira proces razvoja programskog proizvoda, a koji uključuje unaprijed definirane elemente isporuke i ostale artefakte koji su kreirani od strane projektnog tima u svrhu razvoja i održavanja proizvoda [3]. Projektni timovi mogu imati različite pristupe u provedbi okvira definiranog metodikom razvoja.

### 1.2.1 Faze procesa razvoja

Kako bi smo lakše razumjeli sljedeća poglavlja priručnika, u ovom poglavlju prikazat ćemo generički pristup u razvoju programskih proizvoda kroz prikaz osnovnih faza u procesu razvoja. Prema Elliotu (2004) najstariji formalizirani pristup razvoju je SDLC pristup (eng. Systems Development Life Cycle) [4]. Spomenuti pristup je bio fazno orijentiran i sadržavao je sljedeće osnovne faze:



*Slika 2. Pojednostavljenje SDLC modela*

Faza *inicijacije* predstavlja aktivnosti inkubacije i promišljanja projektne ideje. Tijekom inicijacije uspostavljaju se prvi kontakti između investitora i projektnog tima. Tijekom faze *razvoja koncepta proizvoda* provodi se analiza tržišta i budućem mobilnom proizvodu se definiraju osnovne *funkcionalnosti*. Ova faza je detaljno opisana u prethodnom poglavlju. Faza *planiranja* se odnosi na aktivnosti projektnog menadžmenta, a faza *analize zahtjeva* predstavlja detaljnu analizu svih korisničkih zahtjeva iz kojih se radi arhitekturni i strukturni *dizajn* programskog proizvoda. Slijede faze *razvoja* u kojoj se razvijaju moduli programskog proizvoda, *integracije* u kojoj se moduli spajaju te *testiranja* u kojoj se testira gotov proizvod. U konačnici proizvod se održava objavom novih verzija te na kraju uklanja s tržišta.

Sve navedene SDLC faze mogu se grupirati u četiri osnovne faze koju se primjenjive i na razvoj mobilnih aplikacija. Daljnje analize u ovom priručniku ćemo temeljiti kroz četiri osnovne faze razvoja softvera: *konceptualno modeliranje* – faza osmišljavanja ideje i projekta razvoja programskog proizvoda, *specifikacija i dizajn* – faza detaljne specifikacije funkcionalnosti i izgleda programskog proizvoda, *implementacija* – faza izrade i testiranja programskog proizvoda, *korištenje* – faza isporuke, korištenja i održavanja programskog proizvoda.

### 1.2.2 Klasifikacija metodika razvoja

Iako je prva verzija SWEBOK standarda [5] definirala tri osnovne skupine konceptata za sistematizaciju metodika razvoja, u novoj verziji, koja je još uvijek u procesu izrade [2] sve metodike možemo promatrati kroz koncepte koji ih svrstavaju u jednu ili više sljedećih skupina: metodike temeljene na iskustvu, formalne metodike, metodike prototipiranja i agilne metodike razvoja.

- ✓ Heurističke metodike – metodike softverskog inženjerstva temeljena na iskustvu. Često se koriste u praksi, a dijelimo ih na metodike temeljene na strukturnoj analizi i dizajnu, metodike temeljene na podatkovnom modeliranju, te metodike temeljene na objektno orijentiranom dizajnu i analizi.
- ✓ Formalne metodike – metodike softverskog inženjerstva temeljene na rigidnim matematičkim notacijama i jezicima. Koriste se za razvoj visoko zahtjevnih sustava te nisu često primjenjivane u mobilnom razvoju.
- ✓ Metodike prototipiranja – metodike temeljene na pristupu kreiranja nedovršenih proizvoda ili proizvoda s minimalnom funkcionalnošću, najčešće u svrhu isprobavanja tehnoloških



oštrica ili implementacije zahtjeva. Temeljem kreiranog prototipa može se dobiti povratna informacija na kvalitetu ispunjenosti zahtjeva, korisničkog sučelja, dizajna i/ili drugih teško razumljivih aspekata proizvoda. Obično, prototip ne postaje konačni proizvod bez značajnih dorada.

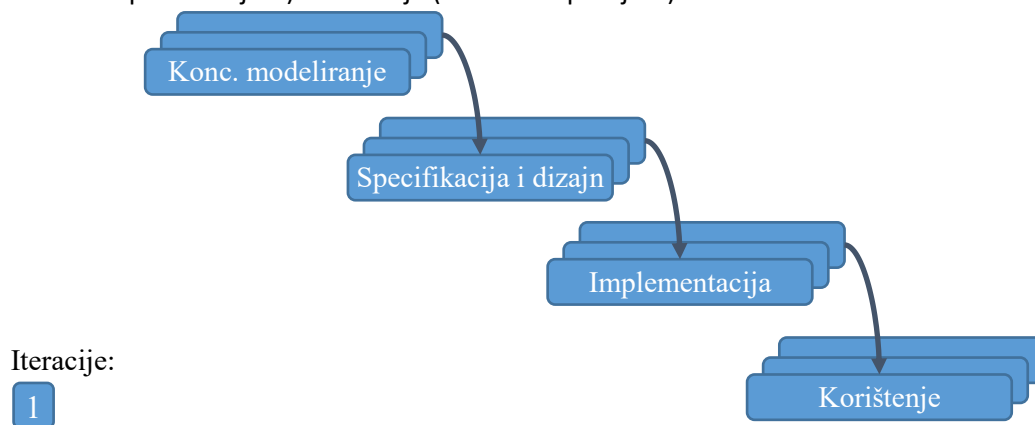
- ✓ Agilne metodike – metodike razvoja nastale 1990-tih godina iz potrebe smanjenja velikog dodatnog posla povezanog s provedbom zahtjevnih i detaljnih metodika razvoja. Agilne metodike su jednostavne metodike koje se temelje na kratkim iterativnim razvojnim ciklusima, timovima koji se samo-organiziraju, jednostavnom dizajnu, učestalom refaktoriranju kôda, razvoju vođenom testiranjem, uključenosti korisnika te naglasku na kreiranju verzije proizvoda koja ima nove funkcionalnosti spremne za isporuku na kraju svakog razvojnog ciklusa.

Spomenute skupine nisu disjunktne, što znači da ista metodika razvoja, može pripadati jednoj ili više skupina. Tako na primjer, objektno orijentirana heuristička metodika može biti agilna, formalna ili se pak temeljiti na prototipiranju.

### 1.2.3 Klasifikacija pristupa razvoju

Pristupi razvoju mogu biti grupirani temeljem više različitih kriterija. U razvoju mobilnih programskih proizvoda posebno su zanimljive klasifikacije temeljem *razvojnog ciklusa* i temeljem *osnovnog modela*. Tako, uzimajući u obzir razvojni ciklus, to jest provedbu faza razvoja i njihovog redoslijeda pristupi razvoju mogu biti:

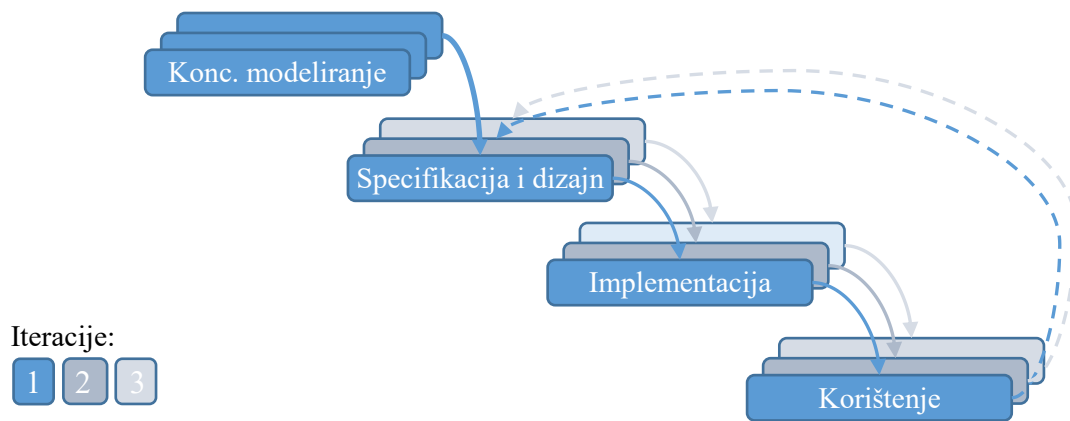
- ✓ fazno orijentiran pristup – pristup u kojem je svaka faza provedena samo jednom tijekom cjelokupnog procesa razvoja. Prije prelaska na sljedeću fazu provodi se verifikacija (provjera u skladu sa specifikacijom) i validacija (korisnička provjera) kreiranih rezultata trenutne faze.



Slika 3. Fazno orijentirani pristup

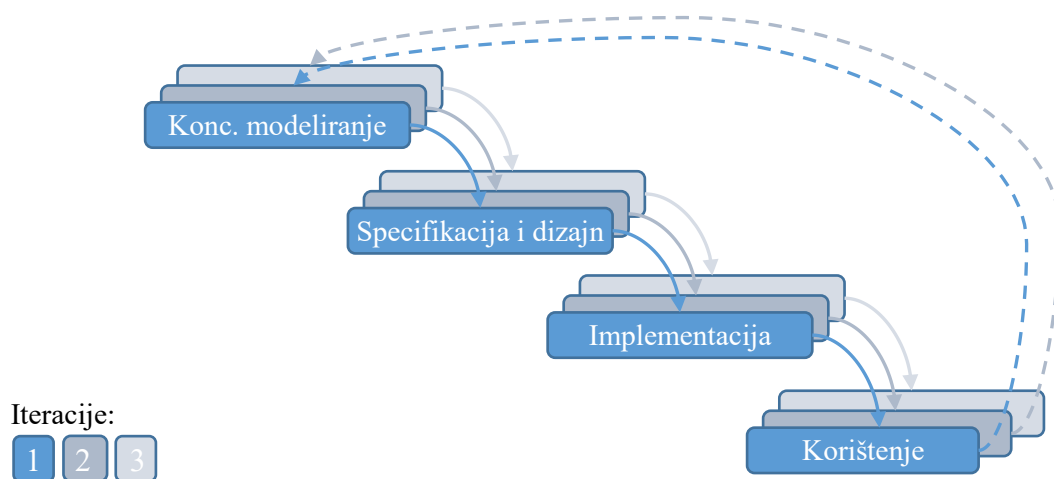
- ✓ djelomično inkrementalni pristup - pristup u kojem se nekoliko faza razvoja provodi više puta - inkrementalno, a set početnih faza se obično provodi samo jedanput kao u fazno orijentiranom pristupu. Postoji više varijanti ovog modela u kojima variraju faze koje se ponavljaju.





Slika 4. Djelomično inkrementalni pristup

- ✓ inkrementalni pristup – pristup u kojem se cjelokupna funkcionalnost mobilnog programskog proizvoda kreira i isporučuje iterativno i inkrementalno. Kreirani modeli i funkcionalnost evoluiraju i poboljšani su svakim inkrementom.



Slika 5. Inkrementalni pristup

S druge strane, promatrajući osnovni model koji se koristi kako bi se definirao proizvod, pristupi razvoju mogu biti: *funkcijsko (procesno) orijentirani pristupi* koji definiraju da je specifikacija funkcionalnosti temeljni model, *podatkovno orijentirani pristupi* koji definiraju da je model podataka temeljni model, te *objektno orijentirani pristupi* koji promatraju objekte i modele objekata kao temeljne modele. Objektni model u osnovi objedinjuje i procesni/funkcijski i podatkovni model u objekt, koji pak mogu biti korišteni kako bi se kreirali statički i dinamički modeli sustava.

Zbog specifičnosti mobilnog razvoja, razmjerno mali broj tvrtki još uvijek koristi klasični (tradicionalni, najčešće fazni) pristup razvoju mobilnih programskih proizvoda, a velika većina tvrtki koristi agilni razvoj (najčešće inkrementalni) koji ima naglasak na konačnom proizvodu, zadovoljnom korisniku i fleksibilnosti, pri tome stavljajući u drugi plan stroge ugovore i projektne planove, opširnu dokumentaciju i tromo odgovaranje na zahtjeve za promjenama.

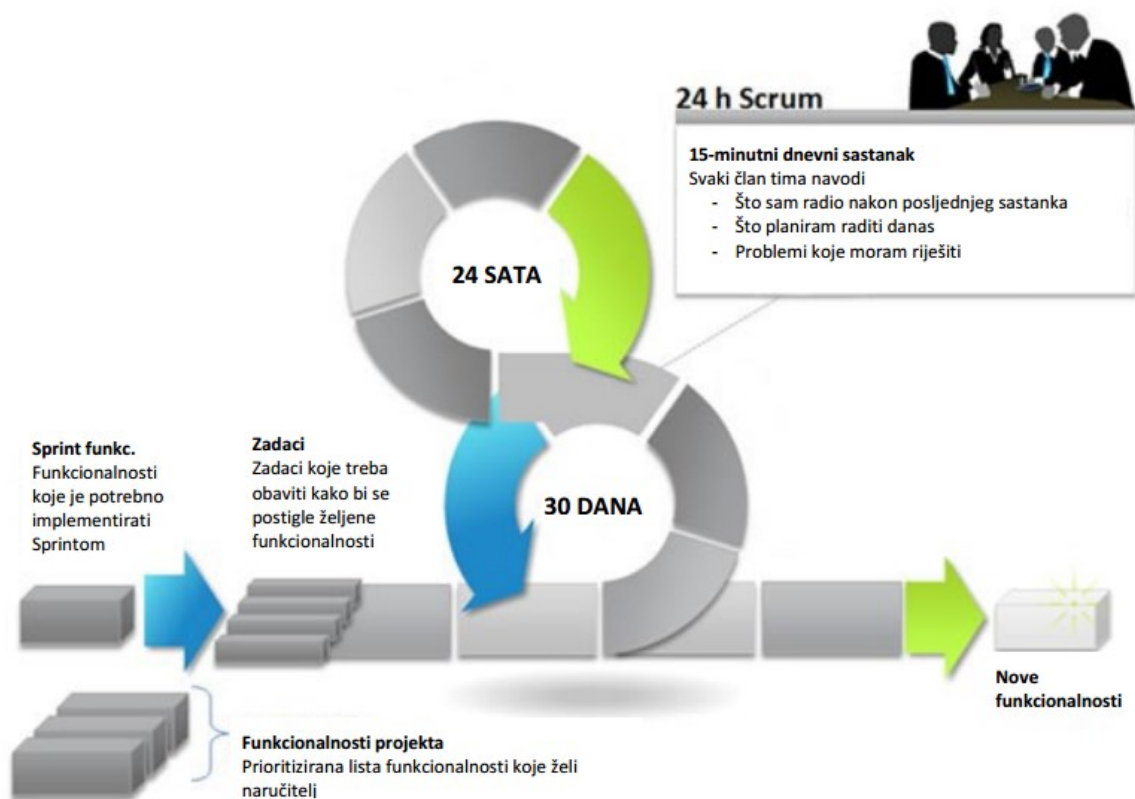
U praksi najčešće korišten predstavnik agilnih metodika razvoja je Scrum metodika. U sljedećem poglavlju prikazane su smjernice Scruma koji definira kako organizirati projektni tim i kako iterativno provesti proces razvoja mobilnog programskog proizvoda.

### 1.3 Scrum proces razvoja

Scrum je agilna metodika razvoja programskih proizvoda koju su kreirali Ken Schwaber i Jeff Sutherland [6]. Riječ je o definiranom pristupu koji sadrži mali set praksi i preddefiniranih uloga i kao takav je postao de-facto standard kod agilnog razvoja. Budući da sadrži jako malo praksi, može ga se primijeniti na različite situacije, ali zbog toga Scrum najčešće nije dostatan, te projektni timovi i tvrtke ponekad primjenjuju i prakse „preuzete“ iz drugih agilnih metodika, kao što je na primjer Ekstremno programiranje. Za detalje o ekstremnom programiranju možete konzultirati [7].

Web mjesto [Scrum.org](http://Scrum.org) pruža različite mogućnosti učenja Scrum procesa. Također, originalni vodič „The Scrum Guide“ [6] je ažuriran svake godine, i može ga se koristiti za početak učenja Scruma. Konačno, već spomenuti priručnik Jeffa Sutherlanda [8] također sadrži sve što je potrebno znati kako bi se počelo s primjenjivanjem Scrum praksi. Osim navedenih, postoji jako puno drugih materijala, knjiga, članaka, vodiča, prezentacija i priručnika koje možete konzultirati za proširenje znanja.

Poglavlja koja slijede temeljena su i imaju preuzete materijale iz Priručnika Jeffa Sutherlanda [8], osim ako je to navedeno eksplicitnom referencom na drugi izvor.



Slika 6. Scrum proces

### 1.3.1 Osnovni Scrum koncepti

Scrum je iterativni i inkrementalni okvir za razvoj programskih proizvoda. Timovi koji koriste Scrum navode značajna poboljšanja u produktivnosti i motivaciji. Jedna od razloga tome je i činjenica da Scrum poznaje samo tri „uloge“ članova projektnog tima:  *vlasnik proizvoda* (eng. Product Owner),  *Scrum majstor* (eng. Scrum Master) i  *projektni tim*. Vlasnik proizvoda može biti investitor, naručitelj ili voditelj proizvoda (eng. Product Manager). Scrum majstor je odgovoran za Scrum te obavlja sve aktivnosti potrebne da bi Scrum funkcionirao. Tu ulogu obično obavlja projektni menadžer ili voditelj tima, međutim važno je spomenuti da on u Scrumu nema nadređenu ulogu u odnosu na ostatak projektnog tima, već ima ulogu uklanjanja prepreka koje usporavaju ili zaustavljaju projekt, ili pak nedozvoljavaju provedbu Scrum praksi. Projektni tim ima 5 do 10 članova, koji pokrivaju različita područja (interdisciplinaran je) uključujući razvojne inženjere, dizajnere korisničkog sučelja, testere i druge [9].

Projektne funkcionalnosti su definirane kroz listu zadataka to jest željenih  *funkcionalnosti projekta* (eng. Project Backlog). Ova lista ne treba nužno biti detaljna niti u potpunosti opisana, već se temelji na željama korisnika ili naručitelja. Listu kreira  *vlasnik proizvoda* koji ujedno definira prioritete zadataka, to jest sortira funkcionalnosti od onih koje su mu najvažnije do onih koje su najmanje važne i mogu biti implementirane na kraju procesa [9].

Scrum, kako je prikazano (Slika 6), cjelokupni proces razvoja dijeli u razvojne cikluse koji se nazivaju **Sprintovi** (eng. Sprint). Te iteracije su kraće od mjesec dana, a obično se mjere u tjednima. Sprintovi se izvršavaju jedan iza drugoga, te svaki sprint ima fiksno trajanje – što znači da završavaju na planirani datum, bez obzira jesu li svi zadaci ispunjeni ili ne. Trajanje sprinta se nikad ne produžuje.

Na početku svakog sprinta, interdisciplinarni tim odabire funkcionalnosti (korisničke zahtjeve) iz prioritizirane liste svih zadataka projekta. Tim namjerava završiti sve odabrane funkcionalnosti do kraja sprinta i dat će sve od sebe kako bi to i postigao. Odabrani zadaci, to jest postavljeni cilj, se tijekom sprinta ne mijenjaju, a svakoga dana tim se okuplja na kratkom dnevnom sastanku na kojem se može donijeti novi plan ili reorganizirati tim kako bi se povećala vjerojatnost uspješnog završetka sprinta. Na kraju svakog sprinta naručitelju se prezentiraju nove funkcionalnosti, nakon čega slijedi „privatni“ retrospektivni sastanak kako bi se analizirao sprint, izvukle pouke i povratne informacije u svrhu poboljšanja sljedećih sprintova.

Scrum stavlja naglasak na proizvod koji na kraju sprinta ima dovršene funkcionalnosti. U smislu mobilnog razvoja dovršena funkcionalnost mora biti:

- ✓ integrirana u postojeće rješenje
- ✓ potpuno testirana
- ✓ potencijalno isporučiva.

### 1.3.2 Prioritetna lista funkcionalnosti

Prioritetna lista funkcionalnosti (eng. Product Backlog) predstavlja popis svih željenih funkcionalnosti programskog proizvoda sortiranih sukladno prioritetima isporuke, kako ih vidi vlasnik proizvoda. Važno je naglasiti da ova lista funkcionalnosti evoluirala tijekom provedbe projekta, pri čemu vlasnik može dodavati, brisati ili drugačije sortirati stavke. Također, projektni tim nakon svakog Sprinta dopunjava listu novom procjenom potrebnog napora za implementaciju pojedinih zahtjeva. Uvijek postoji samo jedna lista željenih funkcionalnosti što tjera vlasnika proizvoda da definira prioritete promatrajući sve funkcionalnosti mobilnog programskog proizvoda.

Item	Details (wiki URL)	Priority	Estimate of Value	Initial Estimate of Effort	New Estimates of Effort Remaining as of Sprint...					
					1	2	3	4	5	6
As a buyer, I want to place a book in a shopping cart (see UI sketches on wiki page)	...	1	7	5						
As a buyer, I want to remove a book in a shopping cart	...	2	6	2						
Improve transaction processing performance (see target performance metrics on wiki)	...	3	6	13						
Investigate solutions for speeding up credit card validation (see target performance metrics on wiki)	...	4	6	20						
Upgrade all servers to Apache 2.2.3	...	5	5	13						
Diagnose and fix the order processing script errors (bugzilla ID 14823)	...	6	2	3						
As a shopper, I want to create and save a wish list	...	7	7	40						
As a shopper, I want to add or delete items on my wish list	...	8	4	20						

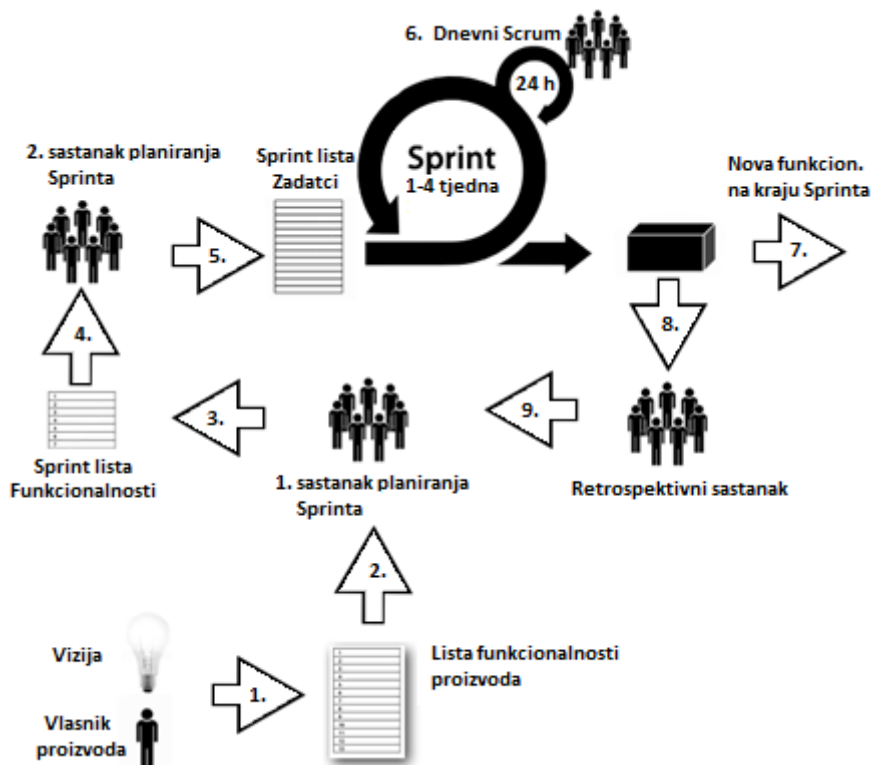
Slika 7. Lista funkcionalnosti proizvoda (eng. Product Backlog)

Scrum ne propisuje formu niti način procjene preostalog napora u listi funkcionalnosti, ali je uobičajeno koristiti relativne procjene izražene u „poenima“ u odnosu na apsolutne procjene kao što su na primjer „čovjek mjeseci“. Osnovni razlog tome je što tijekom vremena „poeni“ mogu rasti (ukoliko tim procijeni da treba više napora za dovršetak funkcionalnosti) ili padati.

Stavke prioritetne liste zadataka mogu značajno varirati u veličini ili potrebnom naporu. Pri tome se zahtjevnije stavke razbijaju na jednostavnije tijekom 2. sastanka projektnog tima za Sprint. Također se, tijekom istog sastanka, izrazito jednostavne funkcionalnosti mogu okupiti u jedan zadatak.

### 1.3.3 Sprint

Scrum strukturira razvoj programskog proizvoda u iteracije koje se nazivaju sprintovi. Te iteracije su tipičnog trajanja od 1 do 4 tjedna. Sprintovi imaju fiksno trajanje, nikad se ne produžuju, te završavaju na planirani datum neovisno o uspješnosti ispunjenja uključenih zadataka.



Slika 8. Detaljan Scrum proces [8]

### 1.3.4 Planiranje sprinta

Kako je prikazano na slici (Slika 8), aktivnosti planiranja sprinta provode se na početku svake iteracije, ali prije početka samog sprinta. Vlasnik proizvoda i Scrum tim (uz pomoć Scrum majstora) tijekom prvog (1.) sastanka prolaze kroz cjelokupnu prioritiziranu listu funkcionalnosti proizvoda, analiziraju ciljeve i kontekst stavki liste, definiraju konačnu željenu funkcionalnost, te Scrum tim odabire one stavke liste koje, sukladno željama vlasnika, ali i tehničkim mogućnostima koje ovise o izrađenoj infrastrukturi proizvoda, može implementirati do kraja sprinta. Popis odabranih funkcionalnosti za novi sprint se naziva lista zadataka sprinta (eng. Sprint Backlog). Tijekom ovog prvog sastanka, vlasnik proizvoda ima ključnu ulogu, a cilj sastanka je u potpunosti shvatiti želje vlasnika proizvoda.

Drugi sastanak tima (Slika 8) fokusira se na detaljno planiranje svih zadataka koje je potrebno obaviti u svrhu implementacije svih funkcionalnosti uključenih u sprint. Pri tome se detalji upisuju u kreiranu listu zadataka sprinta (Slika 10). Ovaj sastanak ne bi trebao trajati duže od jednog sata za svaki tjedan planiranog trajanja sprinta. Drugi važan zadatak ovog sastanka je definiranje vremenskog okvira i kapaciteta projektnog tima (Slika 9).

<b>Sprint Length</b>	2 weeks
<b>Workdays during Sprint</b>	8 days

Team Member	Available Days During Sprint*	Available Hours per Day	Total Available Hours
Tracy	8	4	32
Sanjay	7	5	35
Phillip	8	4	32
Jing	6	5	30

*Slika 9. Definiranje vremenskih kapaciteta*

U ovoj fazi je moguće uključiti u listu dodatne funkcionalnosti ili čak maknuti neke od odabranih funkcionalnosti s glavne liste, ukoliko se utvrdi da tim ima višak kapaciteta ili pak u drugom slučaju nema dovoljno kapaciteta za provedbu svih potrebnih zadataka. Popis detaljnih zadataka, ali ne i popis funkcionalnosti, može biti izmijenjen tijekom same provedbe sprinta, to jest čak i kada članovi tima započnu implementaciju. Redoslijed implementacije funkcionalnosti i provedbe definiranih zadataka nije bitan, već se tim prilagođava specifičnim okolnostima i pokušava identificirati međudnose funkcionalnosti, a sve u cilju maksimiziranja produktivnosti i kvalitete.

Product Backlog Item	Sprint Task	Volunteer	Initial Estimate of Effort	New Estimates of Effort Remaining as of Day...					
				1	2	3	4	5	6
As a buyer, I want to place a book in a shopping cart	modify database		5						
	create webpage (UI)		8						
	create webpage (Javascript logic)		13						
	write automated acceptance tests		13						
	update buyer help webpage		3						
...									
Improve transaction processing performance	merge DCP code and complete layer-level tests		5						
	complete machine order for pRank		8						
	change DCP and reader to use pRank http API		13						

*Slika 10. Primjer liste zadataka sprinta*

Konačno, tijekom ovog sastanka članovi tima volonterski odabiru zadatke koje preuzimaju na sebe. Zbog predanosti Scrumu, ne može se dogoditi da neki od zadataka ostane ne odabran od nekog člana tima.

### 1.3.5 Dnevni Scrum

Nakon početka sprinta, tim provodi još jednu od ključnih Scrum praksi: dnevni Scrum, to jest dnevne kratke sastanke. Riječ je o 15-minutnim sastancima koji se održavaju svakog radnog dana u unaprijed dogovoreno vrijeme. Sastancima prisustvuju svi članovi tima i prezentiraju informacije potrebne kako bi se utvrdio napredak projekta. Nova procjena preostalog napora po zadacima se upisuje u Sprint listu zadataka (Slika 11). Ako je potrebno, tim može organizirati sastanak (odmah nakon dnevnog Scruma) i napraviti izmjene u planu sprinta ili realocirati dodijeljene zadatke članovima tima.

Product Backlog Item	Sprint Task	Volunteer	Initial Estimate of Effort	New Estimates of Effort Remaining at end of Day...					
				1	2	3	4	5	6
As a buyer, I want to place a book in a shopping cart	modify database	Sanjay	5	4	3	0	0	0	
	create webpage (UI)	Jing	3	3	3	2	0	0	
	create webpage (Javascript logic)	Tracy & Sam	2	2	2	2	1	0	
	write automated acceptance tests	Sarah	5	5	5	5	5	0	
	update buyer help webpage	Sanjay & Jing	3	3	3	3	3	0	
...									
Improve transaction processing performance	merge DCP code and complete layer-level tests		5	5	5	5	5	5	
	complete machine order for pRank		3	3	8	8	8	8	
	change DCP and reader to use pRank http API		5	5	5	5	5	5	
...									
		<b>Total (person hours)</b>	50	49	48	44	43	34	

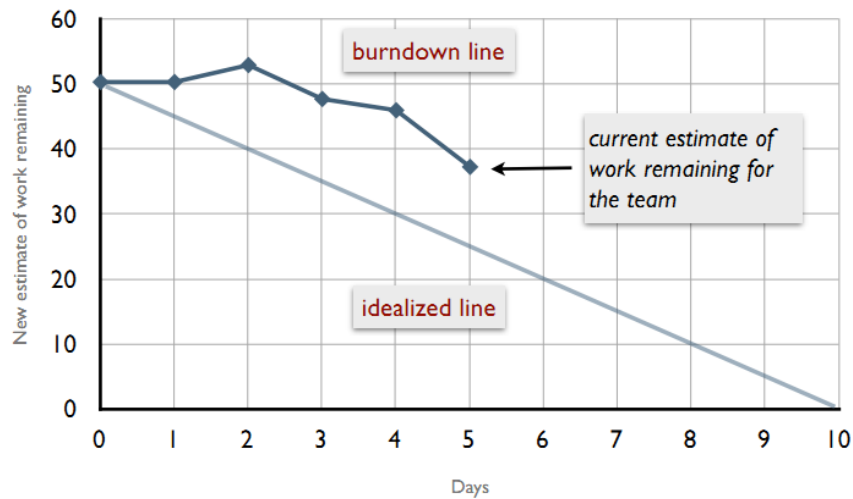
*Slika 11. Dnevno ažuriranje Sprint liste zadataka*

Kako bi dnevni Scrum sastanci bili što kraći, preporuka je da članovi tima ostanu stajati, a tijekom 15-minuta cijeli tim, vrlo kratko, prezentira tri (i samo tri) teme: (1) Što su uspjeli napraviti od jučerašnjeg sastanka; (2) Što planiraju napraviti tijekom dana; i (3) imaju li kakve probleme i prepreke u realizaciji. Tijekom dnevnog Scruma nema diskusije. Naravno, važno je naglasiti da ovo nije jedina komunikacija između članova tima (koji međusobno komuniciraju tijekom cijelog radnog dana, ali sa specifičnim temama vezanim uz izvršenje zadataka), niti se radi o izvješću voditelju projekta (koji niti ne postoji u Scrumu), već se radi o načinu razmjene informacija između članova tima, te po potrebi prilagođavanju tima novim okolnostima. Kako je već spomenuto, ako se pojavi potreba za diskusijom, tim organizira sastanak odmah nakon dnevnog Scruma.

### 1.3.6 Ažuriranje liste zadataka i grafa odrađenog posla

Kako se lista zadataka, a posebno preostalog napora za njihovo izvršenje, ažurira svaki dan, projektni tim također vodi evidenciju ukupno odrađenog/preostalog posla koja se prikazuje pomoću posebnog grafičkog prikaza (eng. Burndown chart). Ovaj graf (Slika 12) na dnevnoj razini prikazuje novu procjenu preostalog napora do završetka svih zadataka. Idealno, riječ je o konstantno silaznom trendu koji je na putu da do kraja sprinta dosegne „nulu“ to jest da prikaže da su zadaci implementirani te da nema potrebe za novim naporom. Ako linija trenda ne slijedi planiranu putanju, Scrum majstor bi trebao poduzeti mjere (promijeniti pristup, ukloniti prepreke, zatražiti pomoć drugih stručnjaka, smanjiti obim posla..) na vrijeme (što je prije moguće) kako ne bi dopustio da Sprint završi neuspjehom.





Slika 12. Grafički prikaz preostalog posla

### 1.3.7 Sprint retrospektiva

Po završetku sprinta i prezentaciji implementiranih funkcionalnosti korisniku, Scrum tim provodi sastanak na kojem se zajedno sa svim zainteresiranim stranama analizira i diskutira provedeni sprint, te se dogovara daljnja strategija razvoja. Na sastanku mogu prisustvovati vlasnik proizvoda, Scrum majstor, članovi tima, korisnici, stručnjaci i bilo tko drugi zainteresiran za projekt i/ili njegove rezultate (eng. *Stakeholders*).

Odmah nakon ovog sastanka, Scrum tim se okuplja na retrospektivni sastanak kako bi se prodiskutiralo ono što jest i što nije funkcioniralo tijekom samog sprinta, te kako bi projektni tim izvukao pouku za sljedeće iteracije, te dogovorio promjene koje će pokušati uvesti.

Item	Details (wiki URL)	Priority	Estimate of Value	Initial Estimate of Effort	New Estimates of Effort Remaining at end of Sprint...					
					1	2	3	4	5	6
As a buyer, I want to place a book in a shopping cart (see UI sketches on wiki page)	...	1	7	5	0	0	0			
As a buyer, I want to remove a book in a shopping cart	...	2	6	2	0	0	0			
Improve transaction processing performance (see target performance metrics on wiki)	...	3	6	13	13	0	0			
Investigate solutions for speeding up credit card validation (see target performance metrics on wiki)	...	4	6	20	20	20	0			
Upgrade all servers to Apache 2.2.3	...	5	5	13	13	13	13			
Diagnose and fix the order processing script errors (bugzilla ID 14823)	...	6	2	3	3	3	3			
As a shopper, I want to create and save a wish list	...	7	7	40	40	40	40			
As a shopper, I want to to add or delete items on my wish list	...	8	4	20	20	20	20			
...	...			...	...					
<b>Total</b>				<b>537</b>	<b>580</b>	<b>570</b>	<b>500</b>			

Slika 13. Ažurirana lista funkcionalnosti i prioriteta

Kako su u ovom trenutku neke stavke prioritete liste već dovršene, potrebno je evidentirati nove procjene za dovršenje preostalih stavki (Slika 13). Neki timovi imaju i grafički prikaz preostalog posla na cijelom projektu, koji je svojom svrhom i formom identičan spomenutom grafu preostalih zadataka sprinta. Slijedom analize sprinta i retrospektive, vlasnik proizvoda može ažurirati i listu zadataka s dodavanjem, promjenom i/ili brisanjem postojećih funkcionalnosti. Budući da su svi

spremnim za početak novog ciklusa razvoja nema potrebe za bilo kakvim čekanjem između dvaju sprintova, te planiranje novog sprinta može početi već sljedeće radno jutro.

## 1.4 Projektni tim i uloge

Kao i u svakoj drugoj industriji, tako i u industriji razvoja mobilnih aplikacija postoji široki spektar uloga, odgovornosti i ekspertize koju treba projektni tim kako bi kvalitetno osmislio i izradio programski proizvod. U prethodnom poglavlju smo opisali Scrum proces, ali projektni tim smo promatrali kao „crnu kutiju“ te nismo navodili detalje o ekspertizi i ulogama članova tima. U ovom poglavlju, pokušat ćemo prikazati projektni tim i njegove uloge u uobičajenim okolnostima razvoja, te ćemo navesti i ostale uloge koje mogu biti neophodne u implementaciji specifičnih projekata. Popis poslova projektnog tima iznesenog u ovom poglavlju dat će vam smjernice o mogućim specijalizacijama u vlastitom profesionalnom razvoju.

Većina projektnih timova sastavljena je od dva do četiri člana koji obavljaju sve projektne aktivnosti. Minimalno, projektni tim mora sadržavati razvojnog inženjera, dizajnera korisničkog sučelja i voditelja proizvoda. Naravno, ponekad sve te aktivnosti obavlja jedna osoba, ali tada ne možemo govoriti o timu.

Stoga, kako je prikazano u tablici ispod (Tablica 2), sve uloge u projektnom timu možemo podijeliti u dvije skupine: *osnovne uloge* i *ostale uloge*. Osnovne uloge su *voditelj projekta*, *dizajner korisničkog sučelja*, *razvojni inženjer* (popularno zvani programer) te *tester* odnosno osoba zadužena za osiguranje kvalitete. Ostale uloge se javljaju u većim timovima, u dobro organiziranim tvrtkama, te ovise o veličini i tipu projekta. Od ostalih uloga valja izdvojiti *voditelja projekta*, *arhitekta*, *voditelja razvoja*, *marketinškog stručnjaka*, *administratora baze podataka*, *programera pozadinskih servisa* te naravno *stručnjaka iz domene* kojom se bavi mobilna aplikacija.

**Tablica 2. Uloge članova projektnog tima**

Osnovne uloge	Ostale uloge, ovisno o veličini i tipu projekta
<ul style="list-style-type: none"> <li>✓ <b>Voditelj proizvoda</b> (eng. Product manager)</li> <li>✓ <b>Dizajner korisničkog sučelja</b> (eng. Interface Designer)</li> <li>✓ <b>Razvojni inženjer</b> (eng. Developer)</li> <li>✓ <b>Provjera kvalitete, Tester</b> (eng. Quality Assessor, Tester)</li> </ul>	<ul style="list-style-type: none"> <li>✓ <b>Voditelj projekta</b> (eng. Project Manager)</li> <li>✓ <b>Arhitekt rješenja</b> (eng. Solution Architect)</li> <li>✓ Analitičar funkcionalnosti (eng. Functional Analyst)</li> <li>✓ Projektant portabilnosti (eng. Portability Planner)</li> <li>✓ Upravitelj razvoja (eng. Development Manager)</li> <li>✓ <b>Voditelj razvoja</b> (eng. Development Lead)</li> <li>✓ Upravitelj verzijama (eng. Release Manager)</li> <li>✓ Voditelj kvalitete (eng. Quality Lead)</li> <li>✓ Projektant implementacije (eng. Deployment Planner)</li> <li>✓ Implementator (eng. Deployment Publisher)</li> <li>✓ <b>Marketinški stručnjak</b> (eng. Marketing Expert)</li> <li>✓ Analitičar potreba obuke (eng. Training requirement analyser)</li> <li>✓ Trener (eng. Trainer)</li> <li>✓ Arhitekt baze podataka (eng. Database Architect)</li> <li>✓ Programer baze podataka (eng. Database Programmer)</li> <li>✓ <b>Administrator baze podataka</b> (eng. Database Administrator)</li> <li>✓ Priprema podataka (eng. Data Populator)</li> <li>✓ Arhitekt pozadinskih servisa (eng. Backend Service Architect)</li> <li>✓ <b>Razvojni inženjer pozad. servisa</b> (eng. Backend Developer)</li> <li>✓ <b>Domenski stručnjak</b> (eng. Subject Matter Expert)</li> </ul>

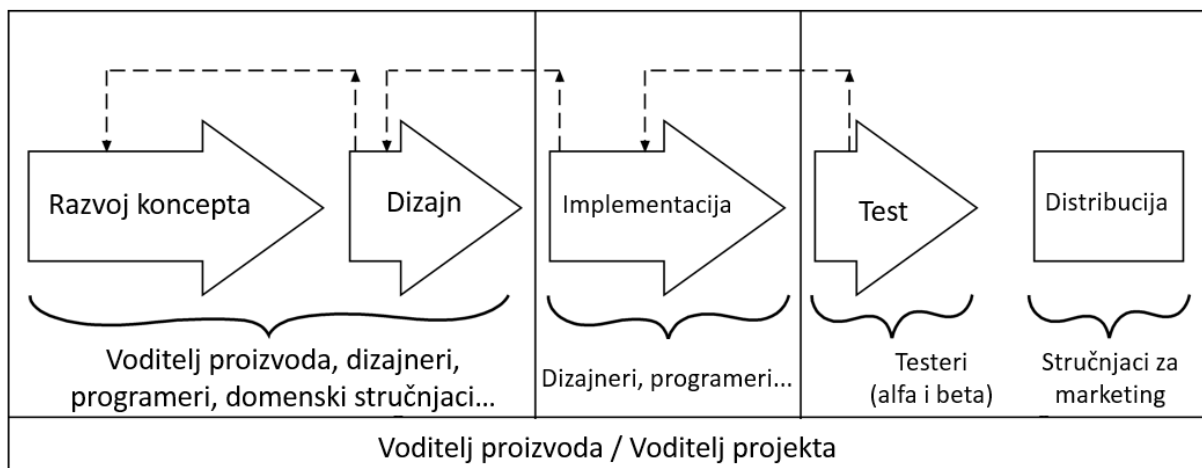


U malim timovima, članovi koji imaju osnovne uloge zastupljeni su tijekom cjelokupnog procesa razvoja, međutim u pojedinim fazama se stavlja naglasak na jednu ili drugu ulogu (Slika 14). Voditelj proizvoda, iako prisutan u svim fazama razvoja, vrlo važnu ulogu ima tijekom faze konceptualnog modeliranja, pa i tijekom faze dizajna, kako bi se osiguralo da vizija koju ima o programskom proizvodu bude jasno prenesena svim članovima tima te pretočena u nedvosmislene specifikacije. S druge strane, marketinški stručnjak, iako bitan i u fazi razvoja ideje, posebnu ulogu ima u fazi pripreme distribucije i distribucije samog proizvoda.

Detaljan opis svih osnovnih uloga i kratak opis važnih ostalih uloga dan je u sljedećim poglavljima.

### 1.4.1 Voditelj proizvoda

U užem smislu riječi, *voditelj proizvoda* (eng. Product manager) je u smislu provedbe projektnih aktivnosti najodgovornija osoba u projektnom timu koja pretvara viziju u konkretan mobilni proizvod. U Scrum timovima, voditelj proizvoda je zapravo *vlasnik proizvoda*, to jest osoba koja je odgovorna za (ne)uspjeh projekta, povratak investiranih sredstava, brzo postizanje točke pokrića i zarade. U malim tvrtkama, voditelj proizvoda ima ulogu direktora (eng. Chief Executive Officer – CEO). Važna uloga voditelja proizvoda je poznavanje svih aspekata projekta te rad na uklanjanju prepreka i definiranju strategije postizanja zadanih ciljeva. Voditelj proizvoda bi trebao imati znanje i ekspertizu iz različitih područja, a neka od njih su vođenje, osnove razvoja mobilnih proizvoda, kratkoročno i dugoročno planiranje, korisničko iskustvo, upravljanje klijentima pa čak i marketing.

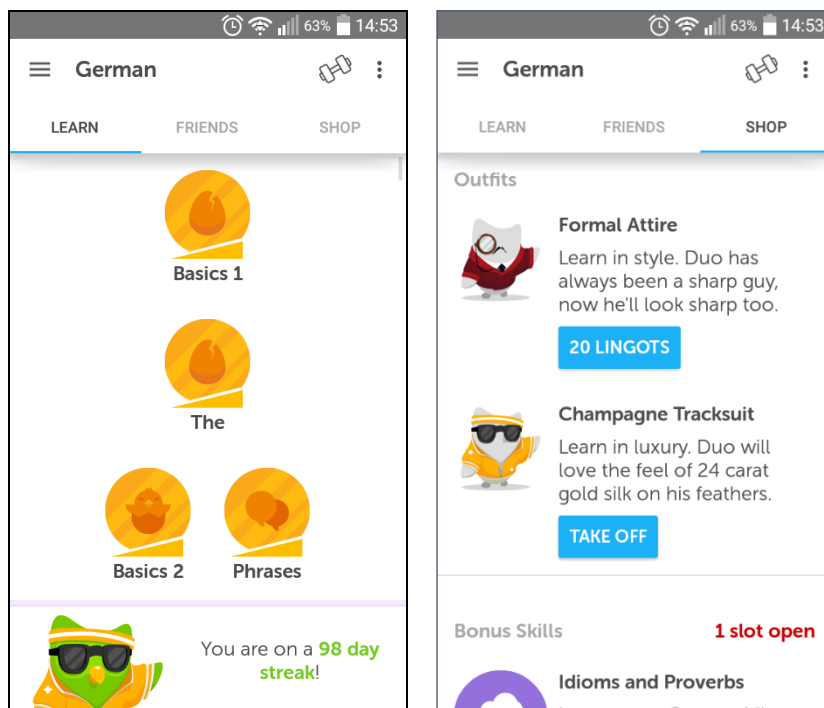


Slika 14. Fokus na ulogama po fazama razvoja

### 1.4.2 Dizajner korisničkog sučelja

Dva su osnovna zadatka *dizajnera korisničkog sučelja* (eng. Designer). On vodi računa o izradi jasnog i jednostavnog korisničkog sučelja (eng. User Interface – UI) u skladu s preporukama i trendovima u odabranoj mobilnoj platformi te pazi na postizanje visoke razine zadovoljstva u korisničkom iskustvu (eng. User Experience – UX). Proizvod koji ima lijepo i privlačno korisničko sučelje (UI) ne mora nužno biti dobar u kontekstu korisničkog iskustva (UX). Naime, za dobro korisničko iskustvo, proizvod mora biti intuitivan, lako navigabilan, prikladno dizajniran za ciljane korisnike, te mora omogućiti korisniku da u svakom trenutku zna gdje se nalazi, kako se može vratiti na početni zaslon te gdje će ga svaka akcija dovesti.

Primjer nedostatka u navigaciji možemo pronaći u najpopularnijoj svjetskoj mobilnoj aplikaciji za učenje stranih jezika – Duolingo<sup>4</sup>. Aplikacija je preuzeta više od 50 milijuna puta, te ima mobilnu inačicu za sve popularne mobilne platforme, kao i web inačicu. Dizajner korisničkog sučelja je u ovoj aplikaciji napravio odličan posao u smislu UI-a, primijenio je posljednje trendove dizajna Android aplikacija, kao što je *Material* dizajn (uključujući primjenu koncepata kao što su prijelazi, animacije, osvjetljenje, sjene i drugo) te je osnovnu navigaciju osmislio kroz primjenu klasične *ViewPager*<sup>5</sup> kontrole koja omogućuje tranziciju između tri osnovna pogleda: *Learn*, *Friends* i *Shop* ( ).



Slika 15. Duolingo design

Međutim, stranicu za učenje korisnici intuitivno smatraju osnovnom početnom stranicom aplikacije, te nakon prelaska na pomoćne stranice za pregled napretka u učenju prijatelja (*Friends*) ili za kupovinu unutaraplikacijskih proizvoda (*Shop*), intuitivno očekuju da funkcionalnost hardverske tipke za povratak u natrag (eng. *Back*) ponovno prikaže početnu stranicu, što nije slučaj. Naprotiv, aplikacija se zatvara. Naravno, ovaj mali nedostatak ne utječe značajno na korisničko iskustvo, ali može biti pokazatelj o kakvim sve detaljima dizajner korisničkog sučelja treba paziti.

U konačnici, uloga dizajnera u tvrtki može biti i pomoć u kreiranju branda od mobilnog proizvoda ili od cijele tvrtke, kreirajući logo, dizajn web stranice, promotivnih materijala i filmova te reklama, što sve obično radi u suradnji s marketinškim stručnjakom.

### 1.4.3 Razvojni inženjer

Popularan kolokvijalni naziv za *razvojnog inženjera* je programer – što je doslovan prijevod s engleskog jezika (eng. *Programmer*) koji se toliko uvriježio u svakodnevni jezik da ćemo ga i mi koristiti u ovom kratkom poglavlju. Programeri su zaduženi za razvoj i implementaciju mobilnog programskog proizvoda koristeći dogovorene razvojne okoline i programske jezike. Za razliku od programera pozadinskih servisa koji moraju imati znanje u web tehnologijama, programeri Android aplikacija najčešće pišu programski kôd u specifičnom razvojnom okruženju, Android Studiju, te

<sup>4</sup> <https://www.duolingo.com/>

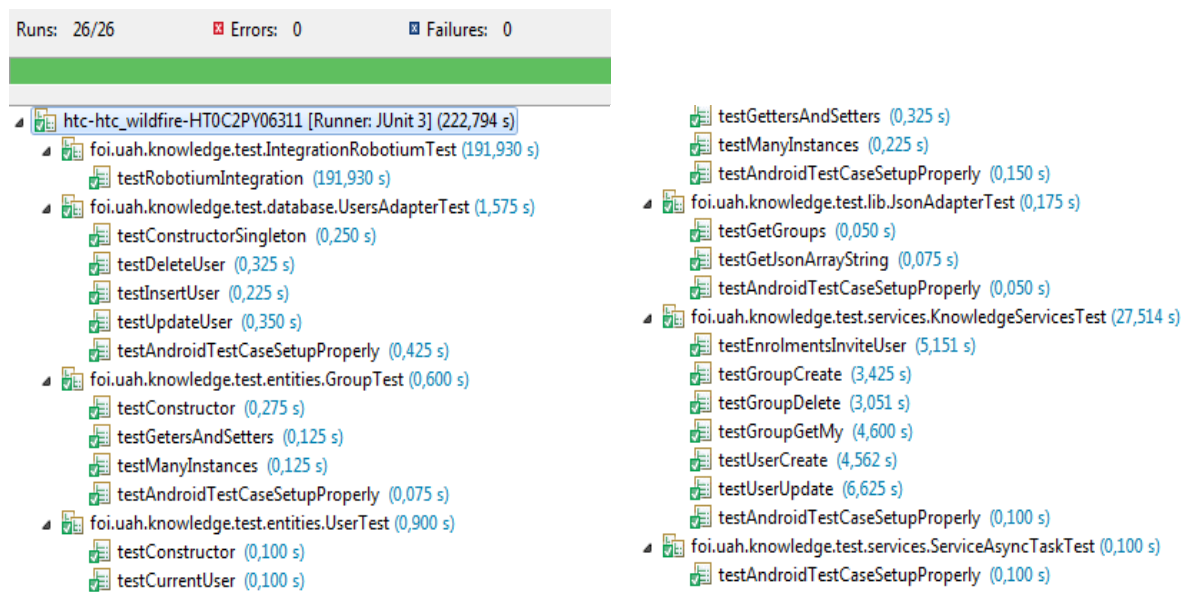
<sup>5</sup> Ova kontrola i ostale kontrole korisničkog sučelja opisane su u posebnom poglavlju ovoga priručnika.

koriste Java programski jezik. Velik dio ovog priručnika stavlja naglasak upravo na ove dvije tehnologije.

Programeri su odgovorni za implementaciju dogovorene arhitekture, te za implementaciju i integraciju pojedinačnih modula u cjelokupno rješenje. Nerijetko, pojedini moduli moraju komunicirati sa drugim sustavima, te ulogu njihovog spajanja također imaju programeri. Iako su sve uloge važne, programeri najčešće doprinose cjelokupnom rješenju s najviše uložene posla i truda.

#### 1.4.4 Tester

Sistematsko testiranje je jedna od najčešće zapostavljenih, a važnih aktivnosti. Zabludom, projektni timovi smatraju da je dovoljno testiranje koje provode razvojni inženjeri tijekom samog pisanja programskog kôda. Takav pristup najčešće dovodi do pogrešaka koje otkrivaju korisnici, što utječe na lošu reputaciju programskog proizvoda.



Slika 16. Primjer rezultata automatiziranog testiranja

S druge strane, *testeri* (eng. Tester) provode sistematsko testiranje programskog kôda, a potom i programskog proizvoda provodeći niz automatiziranih, polu-automatiziranih i manualnih testova. Neki od značajnih testova koji se provode tijekom faze razvoja su jedinični testovi (Slika 16), funkcionalni testovi, integracijski testovi, sistemski testovi, stres testovi te testovi prihvatljivosti. U Androidu postoji više gotovih okvira pomoću kojih se mogu pisati automatizirani ili polu-automatizirani testovi a neki od njih su Android Test Framework, Robotium i drugi.

Dizajn, pisanje i provođenje ovih testova te osiguranje kvalitete programskog proizvoda (eng. Quality Assurance – QA) osnovna su zadaća testera.

#### 1.4.5 Ostale uloge

U ovisnosti o veličini i vrsti projekta, pri razvoju mobilnih programskih proizvoda javljaju se i druge bitne uloge sa specifičnim zadacima i znanjima. Detaljan popis svih uloga dan je u već spomenutoj tablici (Tablica 2), a podebljano su prikazane važnije uloge prisutne u svakoj ozbiljnoj tvrtki koja se bavi razvojem mobilnog softvera. Osim osnovnih uloga pobrojanih u prethodnom poglavlju, ovdje prikazujemo druge važne uloge članova projektnog tima.

Za vježbu, razvrstajte sve uloge članova projektnog tima pobrojanih u spomenutoj tablici u sljedeća područja koja pokrivaju: *projektni menadžment, dizajn programskog proizvoda, razvoj programskog*

*proizvoda, testiranje, objava/isporuka, marketing, obuka korisnika, upravljanje podacima, razvoj pozadinskih servisa, poznavanje domene.*

**Voditelj projekta** (eng. Project Manager) – Voditelj projekta za razliku od voditelja proizvoda ima nadređenu ulogu u projektnom timu te je direktno odgovoran za planiranje, provedbu i izvješćivanje o projektnim aktivnostima. Voditelj projekta koristi alate i tehnike projektnog menadžmenta u obavljanju svojih aktivnosti te kako bi u definiranom vremenu, s dostupnim resursima implementirao željene funkcionalnosti. Po potrebi vrši restrukturiranje projektnog tima te promjenu uloga članova tima.

**Arhitekt rješenja** (eng. Solution Architect) – Arhitekt rješenja ima odgovornost u osmišljavanju i definiranju arhitekture programskog proizvoda. Arhitekt treba imati dosta iskustva u razvoju, treba moći dalekosežno sagledati posljedice primjene određene tehnologije, razvojnog stila ili pak arhitekturnog ili strukturnog dizajna. Arhitekt definira module programskog proizvoda i njihove veze.

**Voditelj razvoja** (eng. Development Lead) – U timovima u kojima ima više razvojnih inženjera (programera), voditelj razvoja upravlja, nadgleda i kontrolira njihov rad te im daje zadatke. Voditelj razvoja također mora odlučiti o pristupu i tehnologiji koja će biti korištena u rješenju određenog problema, te je odgovoran za osmišljavanje cjelokupne programske infrastrukture – skeleta na koji se dodaju moduli i funkcionalnosti. Voditelj razvoja nadgleda kvalitetu programskog kôda, vrši reviziju kôda te pazi da nekvalitetan, nedovoljno testiran, nekomentiran i nerefaktoriran kôd ne uđe u repozitorij kao konačno rješenje za neku funkcionalnost.

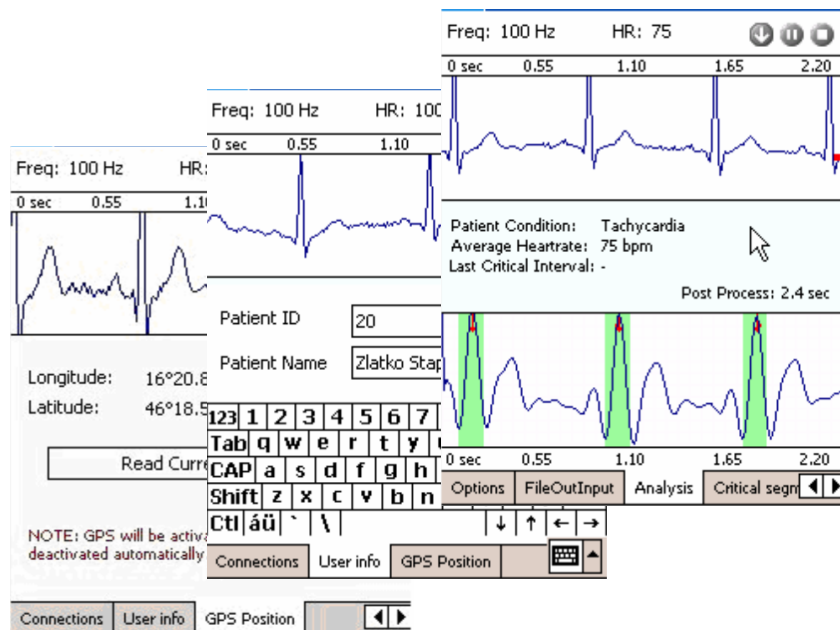
**Stručnjak za marketing** (eng. Marketing Expert) – Marketinški stručnjak ima potpuno drugačije obrazovanje od ostatka projektnog tima, često je vanjski suradnik u timu, te je važno da s ostatkom tima radi u sinergiji i stalnoj komunikaciji. Marketinški stručnjak definira, u okviru budžeta, strategiju prodora na tržište i privlačenja što većeg broja korisnika. On definira vrijeme kada će se na tržište izaći sa informacijama o proizvodu, demo verzijama proizvoda, marketinškom kampanjom ili sa proizvodom u cjelini. Stručnjak za marketing usko surađuje sa dizajnerima koji izrađuju ili sudjeluju u izgradnji marketinških i promidžbenih materijala. Konačno, marketinški stručnjak pomaže izraditi brand od mobilnog proizvoda ili tvrtke.

**Administrator baze podatka** (eng. Database Administrator) – Većina mobilnih programskih proizvoda generira, prikuplja, obrađuje i pohranjuje podatke. Arhitekturna rješenja najčešće zahtijevaju centraliziranu pohranu tih podataka u bazu podataka koja se nalazi na udaljenom računalu ili u oblaku (eng. Cloud). U pojedinim aplikacijama upravljanje podacima je jedan od najzahtjevnijih izazova projektnom timu kako bi osigurali dostupnost podataka u realnom vremenu i sa što manje vremena čekanja korisnika na podatke. U takvim sustavima, uloge kao što su administrator baze podataka, arhitekt baze i/ili skladišta podataka, pa i osobe koje se bave unosom i obradom podataka postaju jako važne. Osnovni zadaci administratora baze podataka uključuju odabir optimalnog sustava za upravljanje bazom podataka, definiranje i implementacija modela podataka, osiguranje sigurnosnih pohrana podataka, optimizacija sustava i slično.

**Razvojni inženjer pozadinskih servisa** (eng. Backend Developer) – U kontekstu rada s podacima opisanog kod prethodne uloge, razmjena podataka između mobilnih aplikacija i pozadinskih sustava, ostvaruje se pomoću web servisa. Web servisi su web aplikacije bez korisničkog sučelja, dizajnirane i izrađene isključivo u svrhu komunikacije s mobilnim aplikacijama. Web servisi rade na web serverima, te prema zahtjevu uslužuju mobilne aplikacije podacima ili prikupljaju podatke od mobilnih aplikacija. Programeri web servisa moraju imati znanja u korištenju web tehnologija i web programskih jezika. U

kontekstu razvoja za Android, web servisi mogu biti pisani u Javi, PHP-u, ASP.Net-u ili bilo kojem drugom jeziku podržanom od strane web servera.

**Domenski stručnjak** (eng. Subject Matter Expert) – Ovisno o tipu i svrsi mobilne aplikacije, projektni tim obično nema dovoljno znanja iz specifičnih domena u kojima razvija mobilnu aplikaciju. Za primjer možemo uzeti mobilnu aplikaciju koja se bavi nadgledanjem i analizom rada srca. Specifična potrebna znanja za komunikaciju sa uređajem za snimanje rada srca, prihvatanje EKG podataka, prikaz podataka na ekranu, obradu signala i slično, znanja su koja projektni tim vjerojatno posjeduje.



*Slika 17. Isječak iz mobilnog programskog proizvoda Smart.ECG<sup>6</sup>*

Međutim, analizu EKG signala i prepoznavanje anomalija u radu srca kao što su tahikardija, bradikardija, predinfarktno stanje, hipertrofije i druge, domena su u kojoj projektni tim ne može imati znanje te treba mentorstvo i vodstvo kako bi implementirali rješenje koje je u stanju prepoznati takve nedostatke. Drugim riječima, domenski stručnjak obično dio svog domenskog znanja prenosi projektnom timu koji ga onda ugrađuju u samo rješenje. Domenski stručnjaci su vanjski suradnici na projektu, a dobra suradnja između njih i projektnog tima obično postaju kritičan faktor u uspjehu cjelokupnog projekta.

<sup>6</sup> Projekt Smart.ECG, FOI Varaždin, 2005-2010. Autori: Velić, Stapić, Novak, Padavić. Stručnjak iz domene: Car S., kardiolog.

## 1.5 Pitanja za provjeru znanja

1. Pojasnite koncept „*razrada projektne ideje*“. Koji je osnovni cilj provedbe aktivnosti vezanih uz razradu projektne ideje?
2. Koje su prednosti sistematskog pristupa razradi projektne ideje u odnosu na nesistematski pristup?
3. Navedite i pojasnite aktivnosti koje provodi projektni tim tijekom procesa razrade projektne ideje.
4. Koji su ključni parametri važni kod pozicioniranja u domenu razvoja?
5. Pojasnite na koji način promjena navika korisnika može biti prilika ali i prijetnja projektnom timu.
6. Navedite nekoliko novih tehnologija koje su trenutno u nastajanju i navedite nekoliko mogućih primjena tih tehnologija.
7. Koja je osnovna razlika pojmova *proširena stvarnost* i *virtualna stvarnost*?
8. Razmislite i navedite nekoliko područja primjene tehnologija *interneta stvari* i *interneta svega*. Na koji način s navedenim tehnologijama možemo unaprijediti ta područja?
9. Odaberite nekoliko kriterija po kojima se mogu segmentirati korisnici, te navedite primjere segmentacije po tim kriterijima.
10. Koje tehnike možemo koristiti prilikom provede aktivnosti definiranja projektnih ideja?
11. Pojasnite pojam „*vodeća funkcionalnost*“ i značaj postojanja takve funkcionalnosti u odnosu na konkurentne proizvode.
12. Navedite ključne čimbenike koje treba uzeti u obzir kod konačnog odabira projektne ideje.
13. Koje su osnovne faze procesa razvoja i koji su rezultati tih faza?
14. Pojasnite razliku između agilnih metodika razvoja i klasičnih metodika u koje spadaju heurističke metodike, formalne metodike i metodike prototipiranja.
15. Koje su specifičnosti djelomično inkrementalnog pristupa provedbi metodika razvoja?
16. Definirajte Scrum proces razvoja, uloge članova Scrum tima, te osnovne korake u Scrum procesu.
17. Pojasnite pojam *sprint* kao jedan od osnovnih Scrum koncepata.
18. Koji se artefakti (dokumenti, alati...) kreiraju i koriste tijekom Scrum procesa?
19. Navedite i pojasnite način nastajanja i svrhu postojanja *grafa preostalog posla*.
20. Navedite i pojasnite osnovne uloge članova projektnog tima?

## 1.6 Resursi za samostalan rad<sup>7</sup>

- ✓ **Langer M. Arthur.** A Guide to software development: designing and managing the life cycle. New York, NY: Springer Berlin Heidelberg; 2016.
- ✓ **Murch Richard.** The Software Development Lifecycle - A Complete Guide: Richard Murch; 2012.
- ✓ **Sommerville Ian.** Software Engineering. 9. izdanje. Boston: Pearson; 2011.
- ✓ **Schwaber Ken, Sutherland Jeff.** The Scrum Guide - The definitive guide to Scrum: The rules of the game. Scrum.org; 2011.

<sup>7</sup> Zbog nedostataka literature na hrvatskom jeziku, donosimo popis izvora na engleskom jeziku.

## 2 OBJEKTNO ORIJENTIRANI PRISTUP RAZVOJU

*Objektno orijentirana paradigma predstavlja način dizajna i razvoja programskih proizvoda korištenjem objekata kao osnovnih gradbenih elemenata. Objekti predstavljaju apstrakciju entiteta iz stvarnosti, a opisuju se pomoću klasa objekata. U odnosu na strukturni dizajn i strukturno programiranje, objektno orijentirani dizajn i objektno orijentirano programiranje zahtijeva od razvojnih inženjera drugačiji način razmišljanja što ponekad predstavlja problem. Stoga, u ovom poglavlju želimo prikazati osnovne objektno orijentirane koncepte i principe koristeći programski jezik Javu. Kako bi smo mogli razumjeti ove složene koncepte, u prvim poglavljima predstavljamo osnovne Jave koncepte, a potom i složene koncepte povezane s objektno orijentiranim programiranjem.*

### SADRŽAJ POGLAVLJA

Uvod u objektno orijentirano programiranje.....	28
Java programski jezik.....	28
Tipovi podataka i operacija .....	29
Logičke strukture.....	33
Metode i svojstva .....	42
Objektno orijentirano programiranje.....	44
Dodatni resursi .....	54



## 2.1 Uvod u objektno orijentirano programiranje

Objektno orijentirana paradigma (OOP) je prisutna u svijetu programiranja već skoro pedeset godina. Prvi programski jezik koji je bio objektno orijentirano koncipiran je bio SmallTalk. Prvi programski jezik koji u cijelosti podržava objektno orijentirano programiranje za Windows platformu je Microsoft Visual C++. Upravo je ovaj jezik bio dugo godina prvi izbor svima koji su željeli slijediti objektno orijentirane principe, ali budući nije jednostavan pa ni elegantan programski jezik zamijenjen je programskim jezicima koji su dizajnirani i građeni kao objektno orijentirani programski jezici. Jedan od takvih jezika je i Java koji koristimo u razvoju Android mobilnih aplikacija.

Objektno orijentirani pristup predstavlja potpuno drugačiju logiku dizajniranja programskog proizvoda od tradicionalne strukturne logike. U objektno orijentiranom dizajnu primjena objektno orijentiranih principa i koncepata je nužna i neizostavna. Uz to, osim što treba znati primijeniti koncepte OOP-a, treba razumjeti zašto ih uopće primjenjivati. Uvođenjem OOP riješena su tri velika problema strukturnih programskih jezika, a time i proizvoda koji su pisani tim jezicima. OOP donosi mogućnost *modularne* izrade programa, *ponovne iskoristivosti* postojećeg kôda te jednostavne nadogradnje već postojećeg programskog kôda.

Cijela objektno orijentirana paradigma se temelji na pretpostavci da sam sustav i osoba koja piše programski kôd dijele zajedničko okruženje, te da su svi programi definirani kao proširenje ovom okruženju. Kao primjer možemo navesti način gradnje kuće. Ako kuću gradite u C-u, onda je za nju potrebno opisati svaku pa i najmanju česticu, svaki atom, jer je C atomarni programski jezik. Objektno orijentirani jezici dopuštaju definiranje vlastitih objekata, kao što su recimo gradbeni elementi kuće ili pak cijeli zidovi ili katovi, a potom i izgradnju kuće spajajući ove gradbene elemente. Kako bi smo razumjeli sintaksu i način kreiranja spomenutih „gradbenih elemenata“ mobilnog programskog proizvoda – to jest objekata, u sljedećim poglavljima ćemo prvo predstaviti programski jezik Javu, a potom ćemo pokazati njezina OOP svojstva.

## 2.2 Java programski jezik

Zbog složenost programskog jezika Java, ovo poglavlje predstavlja samo kratki uvod u osnovne koncepte programiranja u Javi. Stoga, svakako preporučamo učenje ovog programskog jezika pomoću sveobuhvatnijih priručnika ili knjiga. Materijali prikazani u ovom poglavlju temeljeni su na knjizi Y. Daniela Lianga [10].

Java je programski jezik opće namjene. Inicijalno je kreirana 1991. godine u svrhu programiranja elektroničkih čipova ugrađenih u kućanske aparate i uređaje opće namjene. Već 1995. Java dobija današnje ime nakon što je redizajnirana za razvoj web aplikacija. Od 2010. godine Java je u vlasništvu tvrtke Oracle koji nastavlja graditi na viziji da bude više-platfomski programski jezik. To je rezultiralo da Java danas postane programski jezik koji se koristi za izradu softvera ugrađenog u uređaje, aplikacija za web, mobilnih aplikacija, te samostalnih desktop aplikacija. Također, programi pisani u Javi mogu se izvršavati na većini operacijskih sustavima.

Java je, kako navode njezini dizajneri, programski jezik koji se temelji na jednostavnosti, orijentiranosti na objekte, distribuiranosti, robusnosti, sigurnosti, neovisnosti o arhitekturi, portabilnosti, visokoj učinkovitosti, višedretvenosti i dinamičnosti.

Otvorenost Jave, njezinu dostupnost i veliku popularnost prepoznali su i kreatori Androida, te su ju prigrlili kao osnovni programski jezik za kreiranje Android aplikacija. Stoga u ovom poglavlju

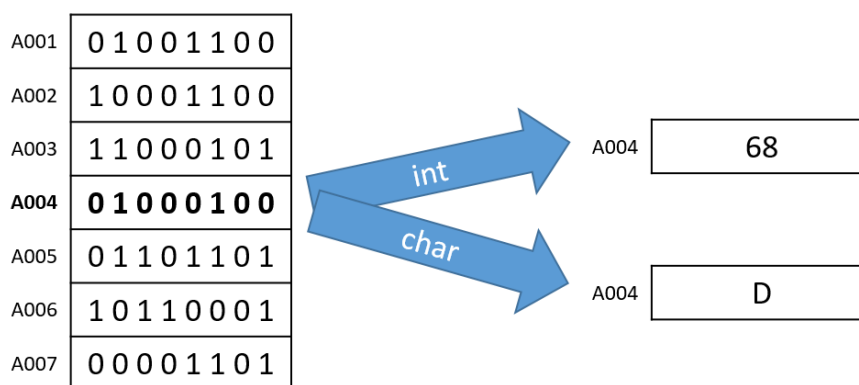


donosimo osnove programiranja u Javi kao uvodu u objektno orijentiranu paradigmu i u razvoj Android aplikacija.

## 2.3 Tipovi podataka

Tip podataka predstavlja način na koji će podaci iz računalnog programa biti pohranjeni u memoriju računala, to jest način na koji će se podaci tumačiti pri čitanju iz memorije. Promotrite sljedeći scenarij:

Podaci se u memoriju računala zapisuju u obliku binarnog zapisa, grupirani u memorijske riječi (bajtove). Zbog jednostavnosti, pretpostavimo da se svaki broj i svaki znak zapisuju u jednu memorijsku riječ, dakle jedan bajt koji je dužine 8 bitova.



Slika 18. Interpretacija memorijskog zapisa ovisna o tipu podataka

Promatrajući primjer memorijskog zapisa prikazan na slici iznad (Slika 18) vidimo da se na memorijskoj lokaciji s adresom A004, nalazi zapis bitova „01000100“. Međutim, računalni program će pri izvršavanju promatrani zapis interpretirati ovisno o deklariranom tipu podatka, te ako je varijabla kojoj je dodijeljena ta adresa deklarirana kao cijeli broj (int) onda će njena vrijednost biti protumačena kao cijeli broj 68. S druge strane, ako je varijabla na toj adresi deklarirana kao znak (char) onda će njena vrijednost biti protumačena kao znak D.

### 2.3.1 Deklaracija, inicijalizacija i definicija varijabli

Kako bi smo podatke mogli pohraniti na jednoznačno mjesto u memoriju, te im kasnije opet pristupiti bez da pamtimo adresu gdje su pohranjeni, u Javi koristimo posebne identifikatore koji se nazivaju *varijable*. Varijable se koriste u svrhu označavanja (imenovanja) memorijskih lokacija s promjenjivim (varijabilnim) podacima. Varijabla za pohranu cijelog broja može biti ovo:

```
int vrijednost = 68;
```

Iznad prikazana linija kôda sadrži potpunu **definiciju** varijable, što znači da je u jednoj liniji kôda **deklarirana** varijabla naziva *vrijednost* i tipa *int*, te joj je odmah **inicijalizirana** vrijednost na 68. Ovaj zapis (Slika 19) ujedno predstavlja osnovnu sintaksu deklariranja i inicijaliziranja varijabli u Javi.



konverzije i zapisa brojeva može se koristiti i neki od dostupnih online kalkulatora, kao što je na primjer Schmidtov kalkulator [13].

### 2.3.3 Operacije nad numeričkim tipovima

Postoji pet osnovnih matematičkih operacija koje se mogu provoditi nad numeričkim varijablama. Popis operacija i znakovnih operatora koje ih predstavljaju, te prioritet izvođenja operacija prikazan je u tablici ispod (Tablica 4).

Tablica 4. Prioritet operacija

Operacija	Operand	Prioritet	Primjer	Rezultat
<b>Zbrajanje</b>	+	2	72 + 18	90
<b>Oduzimanje</b>	-	2	72.0 – 18.5	53.5
<b>Množenje</b>	*	1	72 * 18	1296
<b>Dijeljenje</b>	/	1	18.0 / 72.0	0.25
<b>Ostatak pri dijeljenju</b>	%	1	73 % 18	1

Ispod se nalazi isječak programskog kôda koji prikazuje osnovni rad s varijablama i numeričkim tipovima podataka u Javi.

```
int brojKrugova = 10;
float radijus;
double površina;

radijus = 3.5f;
površina = Math.pow(radijus, 2) * Math.PI * brojKrugova;

System.out.println("Ukupna površina " + brojKrugova + " krugova je: " + površina);
```

U prikazanom primjeru su korišteni numerički tipovi podataka za pohranu *cijelog broja* krugova za koje se računa površina, zatim *realnog 32-bitnog broja tipa float* za pohranu radijusa kruga, te ukupnog rezultata kao *realnog 64-bitnog broja tipa double*. Također, korišteni su i koncepti definicije varijable *brojKrugova*, zatim odvojene *deklaracije* i *inicijalizacije* podacima varijabli *radijus* i *površina*, te koncepti korištenja preddefiniranih paketa za matematičke operacije (konstanta *PI* je definirana u Javi u paketu *Math*) i ispisa u log (pomoću paketa *System*). Rezultat izvođenja programa vidljiv je u Log panelu:

```
Ukupna površina 10 krugova je: 384.84510006474966
```

### 2.3.4 Evaluiranje izraza

Redoslijed izvođenja operatora u Javi jednak je aritmetičkim pravilima izvođenja operatora. Ako izraz sadrži zagrade, onda se prvo evaluiraju dijelovi izraza u zagradama, a potom ostatak izraza. Kod primjene operatora, prvo se evaluiraju operacije množenja, dijeljenja i ostatka, a potom operacije zbrajanja i oduzimanja. Ukoliko izraz ima više operacija istog ranga, one se evaluiraju redoslijedom kojim su napisani, to jest s lijeva na desno.

Tablica 5. Redoslijed evaluiranja izraza

Redoslijed evaluiranja izraza u Javi
8.5 + 4 * 12 – 6 / (1 + 1 * 2) % 3
8.5 + 4 * 12 – 6 / (1 + 2) % 3
8.5 + 4 * 12 – 6 / 3 % 3
8.5 + 48 – 6 / 3 % 3
8.5 + 48 – 2 % 3
8.5 + 48 – 2
56.5 – 2

54.5

 $8.5 + 4 * 12 - 6 / (1 + 1 * 2) \% 3 = 54.5$ 

### 2.3.5 Preljevanje vrijednosti

Posebnu pozornost u radu s numeričkim tipovima podatka treba obratiti na takozvano preljevanje vrijednosti (eng. overflow). Naime, do preljevanja vrijednosti dolazi kada se u varijablu pokušava upisati numerička vrijednost izvan opsega koji može stati u definirani tip podataka. Također do preljevanja vrijednosti može doći kao rezultat provođenja matematičke operacije nad postojećom vrijednosti. Pogledajte sljedeći primjer:

```
int vrijednost = 2147483647 + 1;
//stvarna vrijednost zapisana u varijablu će biti -2147483648
```

Program u kojem se može dogoditi preljevanje vrijednosti imat će logičke pogreške koje je teško pronaći u fazi testiranja.

### 2.3.6 Pretvorba tipova podataka

Java podržava automatsku pretvorbu tipova podataka, koja će se u različitim scenarijima izvršiti sa ili bez gubitka informacija. Svaka numerička vrijednost se može upisati u varijablu čiji tip podržava širi opseg vrijednosti. Takva operacija se naziva *proširenje tipa podataka*. S druge strane, ukoliko želite upisati vrijednost u varijablu koja ima manji opseg vrijednosti, to jest ukoliko želite *suziti tip podataka*, onda morate koristiti eksplicitnu pretvorbu tipa podataka.

Pretvorba tipa podataka (eng. type casting) je posebna operacija koja pretvara podatak iz jednog tipa u drugi tip. Sintaksa pretvorbe je jednostavna – potrebno je upisati željeni (odredišni) tip u zagradu ispred naziva varijable ili vrijednosti koju treba pretvoriti. Promotrite sljedeće primjere i rezultate:

```
System.out.println((int) 8.6f);           //8
System.out.println((int) 4.2);           //4
System.out.println((double) 8.6f);       //8.600000381469727
System.out.println((double) 1 / 2);      //0.5
System.out.println((double) (1 / 2));    //0
```

U prvom primjeru, kod pretvorbe *float* u *int* vrijednost, zbog sužavanja tipa, to jest pretvorbe u tip s manjim opsegom, Java će odbaciti realni dio broja i zadržat će cijeli dio broja. Slično je i u drugom primjeru u kojem se pretvara *double* vrijednost u *int* vrijednost. Primijetite da pretvorba nije isto što i zaokruživanje, jer bi zaokruživanje broja 8.6 vratilo vrijednost 9. U trećem primjeru pretvaramo *float* u *double* vrijednost. Zbog karakteristika i razlika u *float* i *double* zapisu, konvertirani broj nikad ne može biti identičan originalnom broju. Ipak razlike u pretvorbi su zanemarive. U prethodnjem primjeru, operand 1 se pretvara u *double* vrijednost, zatim se provodi operacija dijeljenja nad vrijednostima tipa *double* i tipa *int*, pri čemu Java prednost daje *double* vrijednosti, što znači da će i rezultat biti *double* vrijednost. Razmislite zašto je rezultat 0 u posljednjem primjeru?

### 2.3.7 Znakovni tipovi podataka

Poseban tip podataka *char* predstavlja tip podataka u koji se može pohraniti jedan jedini znak, dok se niz znakova pohranjuje u složeni tip podataka koji se naziva *String*. Vraćajući se na sliku s početka ovog poglavlja (Slika 18), primijetit ćemo da se znak i broj upisuju u memoriju na sličan način – kao niz jedinica i nula. Pri tome, dogovor oko načina pretvorbe numeričkih vrijednosti je definiran standardom, a dogovor oko pretvorbe znakova zovemo *shemom kodiranja*. Java koristi *Unicode* shemu kodiranja, koja je zamišljena da u 16-bitova ima mogućnost pohraniti sve moguće znakove, te

tako omogući maksimalnu portabilnost. Međutim, pokazalo se da 65.536 različitih kombinacija nije dovoljno za pohranu svih znakova, te se shema morala naknadno proširiti pomoćnim znakovima.

Slijedi nekoliko primjera upisa znakovnih vrijednosti u *char* i *String* varijable.

```
char slovo = 'Z';
char znakBroja = '4';
char drugiZnak = '.';
char jednostrukiNavodnik = '\\';
String rijec = "razvoj";
String malaRijec = "i";
String navodnik = "\"";
```

Primijetite, znakovi se upisuju u varijablu tipa *char* pomoću jednostrukih navodnika. Znakovni nizovi se upisuju u varijable tipa *String* pomoću dvostrukih navodnika. Znakovi i znakovni nizovi mogu sadržavati bilo koji znak uključujući slova, interpunkcijske znakove, posebne znakove u različitim jezicima, simbole i slično. Ukoliko želite promatrati specijalne znakove koji u Javi imaju sintaksnu ulogu, onda ih morate prefiksirati specifični znakom *backslash* (\). Tako u posljednjem primjeru za varijable tipa *char* i posljednjem primjeru za varijable tipa *String* primjenjujemo *backslash* kako bi smo označili kompajleru da znak koji slijedi ne promatra kao dio sintakse Java jezika, već kao vrijednost.

Razmislite kako bismo ispisali jedan znak *backslash*, a kako dva *backslasha* uzastopno?

Na znakove se mogu primijeniti i numerički operatori. Promotrite sljedeće primjere i njihove rezultate:

```
int i = '2' + '3';           //50 + 51 = 101
int j = 2 + 'a';           //2 + 97 = 99
char c = (char)99;        //c
char d = 'd' + 'e';       //É
```

Java u navedenim primjerima konvertira operande u cijele brojeve, provodi operaciju, te potom vrijednost upiše u cjelobrojnu varijablu (kod prva dva primjera) ili ju konvertira natrag u znakovnu vrijednost (kod druga dva primjera). Ovakve operacije su izrazito rizične jer mogu rezultirati sa neočekivanim ili nemogućim vrijednostima, te ih treba izbjegavati.

Za razliku od *char* tipom podataka, nad *String* tipom podataka se može provoditi samo operacija konkatenacije (spajanja znakovnih nizova) za koju se koristi operator „+“. Pogledajmo primjer.

```
String ime = "Razvoj";
String prezime = "Aplikacija";
System.out.println(ime + " " + prezime); //Razvoj Aplikacija
```

Operator konkatenacije spaja znakovne nizove u novi znakovni niz čija je dužina jednaka zbroju dužina spojenih nizova. Sve ostale operacije nad znakovnim nizovima, kao što su uspoređivanje, pretraživanje, dijeljenje i slično provode se pomoću specifičnih metoda dostupnih u *String* klasi podataka.

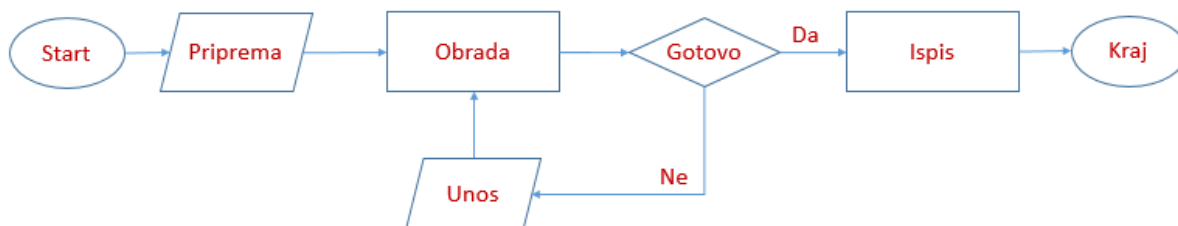
## 2.4 Logičke strukture

### 2.4.1 Logika vođena događajima

Moderni programski jezici spadaju u skupinu programskih jezika u kojima je osnovna programska logika vođena događajima (eng. Event driven). To znači da se programski proizvodi grade na način da

neprestano osluškiju događaje, te izvršavanjem programskog kôda odgovaraju na njih to jest upravljaju njima (eng. Event handling). Takva je i Java za Android.

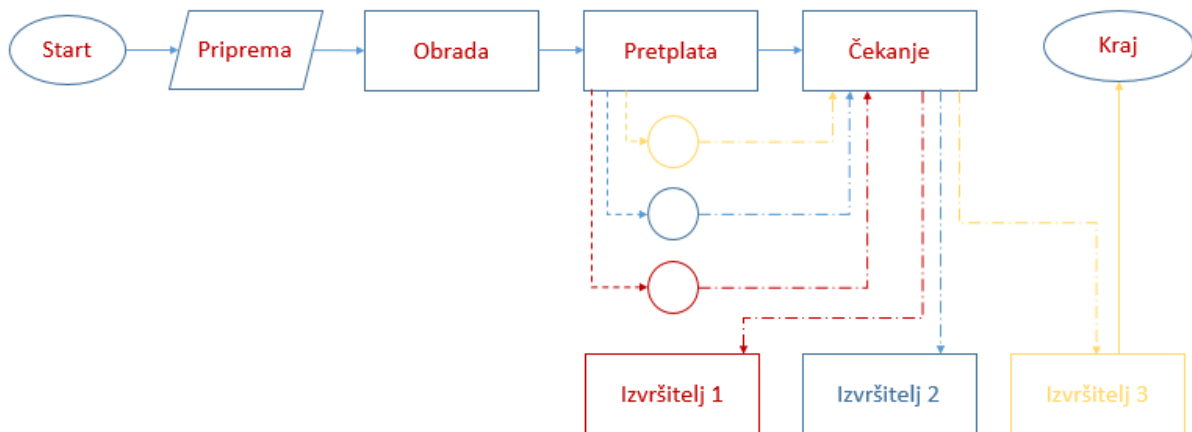
Za usporedbu, u strukturnom programiranju, programski proizvod bi započeo od glavne funkcije, te bi se bez prestanka, uz povremeno čekanje na korisnički unos, slijedno izvršavao do kraja (Slika 20). S druge strane, Android aplikacije (pisane u Javi) započinju kôdom glavne dretve u kojem se, između ostalog, odradi priprema prikaza korisničkog sučelja i podataka, kreira se vlastite događaje i postavi se slušatelje na događaje (eng. Event listener) i nakon toga glavna dretva čeka dok se ne dogodi događaj koji treba obraditi (Slika 21).



*Slika 20. Slijedno izvođenje programa s povratnom vezom*

U pristupu programiranju vođenom događajima, događaje podižu (eng. Trigger) operacijski sustav, aplikacija i korisnici. Neki od događaja koje operacijski sustav šalje svim aplikacijama su: promjena statusa baterije, promjene vrijednosti očitavanja na sensorima, prosljeđivanje poruka drugih aplikacija i slično. Aplikativni događaji mogu biti podignuti po završetku akcije, promjeni vrijednosti u varijabli, dobivanju odgovora od web servisa i slično. Korisnički pokrenuti događaji su događaji kojima se obavještava aplikacija da je korisnik izvršio nekakvu radnju kao što je pritisak (eng. Tap) ili dugi pritisak na elemente korisničkog sučelja, korisničke geste na zaslonu ili korisnikom sučelju, korisnički pritisak na hardverske tipke i slično.

Na slici ispod prikazan je slijed izvođenja programa vođenog događajima. Na tom primjeru možemo vidjeti da glavna dretva provodi pripremu podataka i osnovnu obradu, te potom pretplatu (isprekidani tokovi) na tri događaja (tri kružića) pri čemu pripremi i izvršitelje. Različite boje događaja i povezanih elemenata sugeriraju da izvršitelji nisu kompatibilni te se strogo zna koji izvršitelj može odgovoriti na koji događaj. Po nastanku događaja, bilo korisničkom ili akcijom sustava (isprekidani tokovi s točkom), glavna dretva se aktivira i izvršava odgovarajući kôd. U uobičajenom primjeru aplikacije za Android jedna takva akcija po svom završetku zatvara program.



Slika 21. Izvođenje programa s pretplatom na događaje

Na isti događaj može se izvršiti više zainteresiranih izvršitelja unutar jedne ili različitih aplikacija. Redoslijed izvršavanja više izvršitelja koji oslušuju isti događaj se provodi istim redoslijedom kojim su izvršitelji pretplaćeni na događaj. Međutim, dobra je praksa izbjegavati graditi programsku logiku koja se temelji na redoslijedu provođenja izvršitelja budući da razvojni inženjer teško može upravljati spomenutim redoslijedom.

Osim promjena na logici provođenja cijelog programa, u izvršavanju logičkih cjelina programskog kôda, nema suštinskih promjena u odnosu na strukturno izvršavanje programa. Stoga, sve logičke strukture koje se pojavljuju u izvršavanju programa možemo svrstati u *slijed*, *selekciju* ili *iteraciju*.

#### 2.4.2 Slijed

Slijed (eng. Sequence) predstavlja logičku strukturu koja osigurava izvršavanje linija kôda u istom slijedu u kojem su pisane. Slijed je ujedno i osnovna programska struktura koja jamči razvojnom inženjeru dosljednu provedbu kreiranog programskog kôda koji će se izvršiti uvijek na isti način bez bilo kakvih odstupanja u konačnom rezultatu.

Slijed može biti narušen uporabom skokova ili paralelnim izvršavanjem programskog kôda. Uporabe skokova su stručne zajednice odbačene prije dvadesetak godina, te se uobičajeno smatra da uporaba skokova predstavlja ili dizajnerske propuste ili propuste u znanju samih razvojnih inženjera. S druge strane, paralelno izvršavanje programskog kôda je ponekad potrebno ili čak neizbježno i ono se postiže uporabom više *dretvi* (eng. Thread). Sinkronizacija i razmjena podataka između dretvi su teme koje ćemo samo djelomično obraditi u praktičnom dijelu priručnika, a zbog njihove složenosti preporučamo konzultiranje sveobuhvatne literature.

```
float stanjeRacuna = 5600.50f;
float iznosTransakcije = 125.40f;
float prethodnoStanje = stanjeRacuna;
stanjeRacuna -= iznosTransakcije;
System.out.println("Prethodno stanje računa: \t" + prethodnoStanje);
System.out.println("Iznos transakcije: \t\t" + iznosTransakcije);
System.out.println("Trenutno stanje računa: \t" + stanjeRacuna);
```



```

I/System.out: Prethodno stanje računa: 5600.5
I/System.out: Iznos transakcije: 125.4
I/System.out: Trenutno stanje računa: 5475.1

```

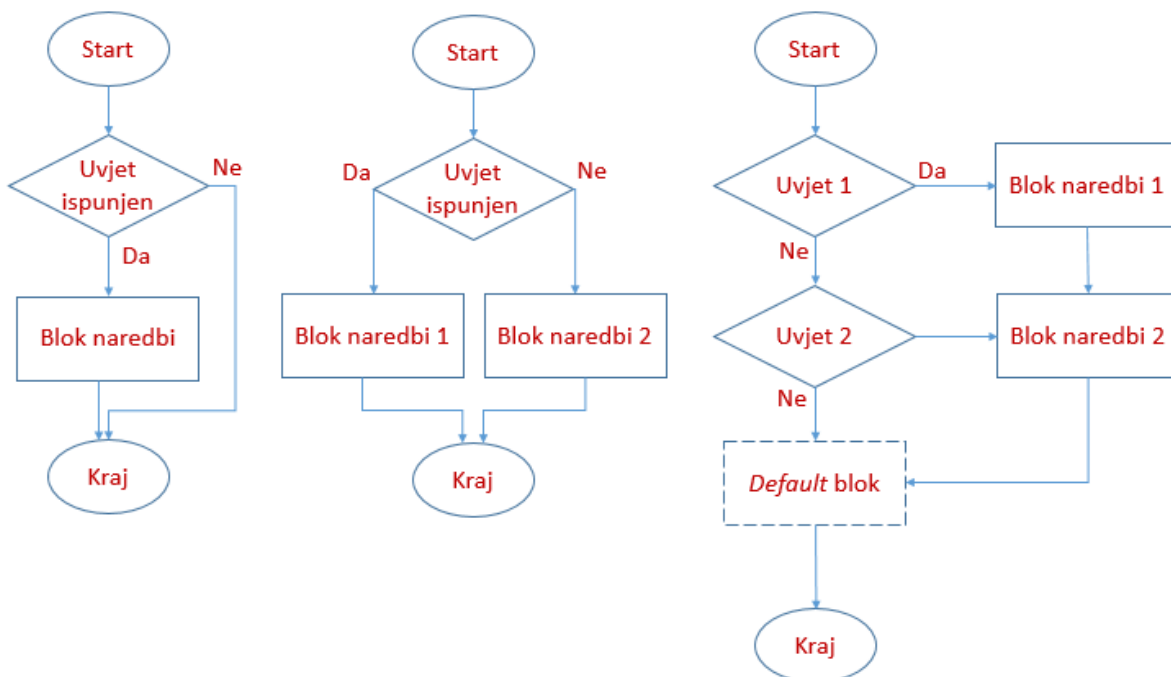
Slika 22. Rezultat provedbe slijednog programskog kôda

Slijedno provođenje prethodno prikazano programskog kôda osigurat će da se vrijednost stanja računa nakon definicije na iznos od 5600.50 prvo upiše u varijablu `prethodnoStanje`, a potom izmijeni. Pri ispisu vidimo da je spomenuta varijabla `prethodnoStanje` sačuvala inicijalnu vrijednost stanja računa.

### 2.4.3 Selekcija

Selekcija (eng. Selection) predstavlja logičku strukturu odabira provedbe jednog između više mogućih programskih blokova. Mogućnosti se nazivaju grane. Odabir se vrši temeljem provjere postavljenih logičkih uvjeta. Kod ispravno definirane selekcije u jednom prolazu se može izvršiti maksimalno jedna, a minimalno niti jedna grana. Drugim riječima, selekcija predstavlja mogućnost programskog proizvoda da odluči koje će naredbe izvršiti.

Naredbe kojima se postiže odabir u Javi su *if naredba*, *if... else naredba* i *switch naredba*. Shematski prikaz spomenutih naredbi dan je na slici ispod (Slika 23).



Slika 23. Logičke strukture u implementaciji selekcije

#### 2.4.3.1 Logički izrazi

Kako bi smo razumjeli provjeru uvjeta u naredbama selekcije, važno je razumjeti logičke izraze u Javi i njihovu evaluaciju. Naime, rezultat evaluacije logičkih izraza može biti *točno* ili *pogrešno* (eng. True; False).

Vrijednost *točno* – *netočno* se pohranjuje pomoću specifičnog tipa podataka koji se naziva *Boolean*. Stoga, rezultat evaluacije svakog logičkog izraza može biti pohranjen u *boolean* varijablu te će imati vrijednost *true* ili *false*.

Logički izrazi se kreiraju i provjeravaju uz korištenje operatora usporedbe (eng. *Comparison Operators*). Java poznaje operatore usporedbe prikazane u tablici ispod (Tablica 6). Operatori usporedbe podrazumijevaju postojanje operanda s lijeve i desne strane operatora.

*Tablica 6. Operatori usporedbe*

Operator	Značenje
==	Operator jednakosti
!=	Operator nejednakosti
<, <=, >, >=	Operatori usporedbe veličine

Za kreiranje i provjeru složenih logičkih izraza Java koristi Booleovu algebru, to jest logički izrazi se mogu slagati u složene izraze uz pomoć Booleovih logičkih operatora. Svi logički operatori, osim operatora negacije, podrazumijevaju da se s lijeve i desne strane nalazi Bool vrijednost, dakle ili točno ili netočno. Operator negacije se provodi nad jednim operandom. Booleovi logički operatori navedeni su u tablici ispod (Tablica 7).

*Tablica 7. Logički (Booleanovi) operatori*

Logički operatori	Značenje / Notacija
!	Negacija NOT
&&	Konjunkcija AND
	Disjunkcija OR
^	Isključiva disjunkcija XOR

Za razumijevanje evaluacije složenih logičkih izraza potrebno je poznavati *tablicu istinitosti* pri primjeni logičkih operatora na boolean vrijednosti (Tablica 8).

*Tablica 8. Tablica evaluacije logičkih izraza*

P1	Operator	P2	Rezultat
true	!		false
false	!		true
true	&&	true	true
true	&&	false	false
false	&&	true	false
false	&&	false	false
true		true	true
true		false	true
false		true	true
false		false	false
true	^	true	false
true	^	false	true
false	^	true	true
false	^	false	false

Primjer i vrjednovanja složenog logičkog izraza koji provjerava je li 2016 godina prijestupna godina prikazan je ispod:

`(2016 % 400 == 0) || ((2016 % 4 == 0) && (2016 % 100 != 0))`

Nakon provedbe matematičkih operacija o kojima smo govorili u prethodnom poglavlju, slijedi logičko vrjednovanje istinitosti izraza (Tablica 9). Rezultat pokazuje da je 2016. godina uistinu prijestupna godina.

Tablica 9. Redoslijed vrjednovanja složenog izraza

Evaluiranje složenog logičkog izraza u Javi
<code>(2016 % 400 == 0)    ((2016 % 4 == 0) &amp;&amp; (2016 % 100 != 0))</code>
<code>(16 == 0)    ((0 == 0) &amp;&amp; (16 != 0))</code>
<code>false    (true &amp;&amp; true)</code>
<code>false    true</code>
<code>true</code>
<code>(2016 % 400 == 0)    ((2016 % 4 == 0) &amp;&amp; (2016 % 100 != 0)) = true</code>

#### 2.4.3.2 Primjeri selekcije

Najjednostavnija implementacija selekcije je pomoću *if naredbe*. Kako je prikazano na prvoj logičkoj strukturi danoj u slici iznad (Slika 23), rezultat vrjednovanja uvjeta može ali i ne mora biti provedba programskog bloka. U primjeru ispod, rezultat vrjednovanja vraća *true* što znači da će se programski blok izvršiti i u log sustava bit će ispisana rečenica „**Godina 2016 je prijestupna!**“.

```
int godina = 2016;
if ((godina % 400 == 0) || ((godina % 4 == 0) && (godina % 100 != 0))) {
    System.out.println("Godina " + godina + " je prijestupna!");
}
```

Programski blok se može sastojati od nijedne, jedne ili više linija kôda. Ako je programski blok prazan ili ako se sastoji od više linija kôda, onda blok mora biti jasno ograđen vitičastim zagradama koje predstavljaju početak '{' i kraj '}' programskog bloka. Ako se blok sastoji od samo jedne linije kôda, zagrade nisu obvezne, ali je preporučeno pisati ih.

U drugom primjeru, godina koja se provjerava je rezultat prethodne obrade (ne mora uvijek biti ista) te je evaluacija izraza neizvjesna. Međutim, u svakom slučaju izraz će ispisati jednu od dvije moguće poruke, budući da kod *if...else naredbe* jedan programski blok mora biti izvršen. To je grafički prikazano u drugoj logičkoj strukturi danoj na slici (Slika 23).

```
int godina = izracunataGodina;
if ((godina % 400 == 0) || ((godina % 4 == 0) && (godina % 100 != 0))) {
    System.out.println("Godina " + godina + " je prijestupna!");
}
else {
    System.out.println("Godina " + godina + " nije prijestupna!");
}
```

U trećem primjeru se koristi *switch naredba* koja pojednostavljuje pisanje složenog logičkog *if...else* izraza koji može imati više mogućih scenarija izvođenja. Ako analiziramo treću logičku strukturu selekcije (Slika 23), možemo primijetiti nekoliko bitnih stvari:

- Ako provjera prvog izraza vrati *true* Java će izvršiti Blok naredbi 1, ali i sve ostale blokove naredbi. Kako bi smo to izbjegli, moramo koristiti *break naredbu* na kraju prvog bloka naredbi.
- Može, ali i ne mora, postojati osnovni (eng. default) blok naredbi koji se izvršava ukoliko niti jedan uvjet nije zadovoljen.
- Logički izraz kod *switch naredbe* uvijek se odnosi na primjenu operatora uspoređivanja „==“.

Promotrimo primjer:

Pretpostavimo da računalni program za određivanje prijestupne godine može korisniku vratiti razlog (cijeli broj) zašto neka godina nije prijestupna. Analizom logičkog izraza koji određuje prijestupnost godine možemo utvrditi da godina nije prijestupna ako:

- razlog 1: Godina nije djeljiva sa 4.
- razlog 2: Godina je djeljiva sa 4 i 100, ali nije djeljiva sa 400.
- u svim drugim slučajevima: Godina je prijestupna!

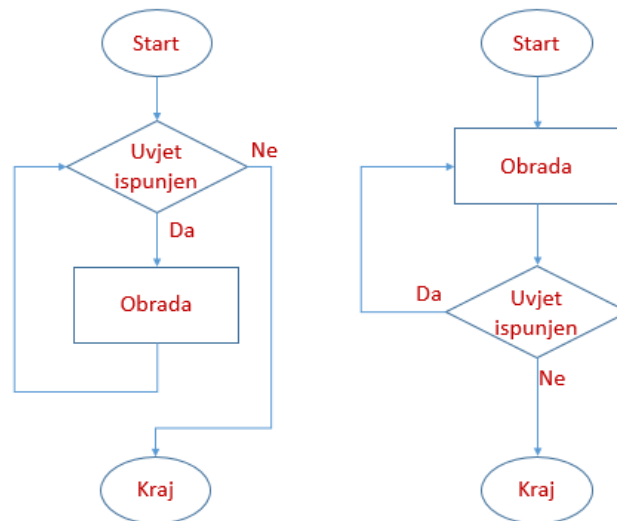
Stoga, korištenjem switch naredbe korisniku možemo ispisati poruke rezultata analize kako slijedi.

```
switch (razlog) {
    case 1: {
        System.out.println("Godina nije djeljiva sa 4!");
        break;
    }
    case 2: {
        System.out.println("Godina je djeljiva sa 100, ali nije djeljiva sa 400!");
        break;
    }
    default: {
        System.out.println("Pa nema razloga. Godina je prijestupna!");
    }
}
```

#### 2.4.4 Iteracija

Iteracija (eng. Iteration) predstavlja logičku struktura koja omogućava višestruko izvršavanje jedne ili više linija programskog kôda. Stoga, iteracija predstavlja prvi i osnovni alat povećanja razine ponovne iskoristivosti kôda, što je cilj svih razvojnih inženjera. Osnovna struktura kojom provodimo iterativnost je *petlja* (eng. Loop).

Programski jezik Java poznaje više naredbi kojima se postiže iterativnost, međutim sve ih dijelimo u dvije skupine. U prvu skupinu spadaju naredbe koje uvjet provođenja ponovljivog bloka kôda ispituju na početku, to jest prije svakog izvođenja. Takve naredbe su *while petlja* i *for petlja*. U drugu skupinu spadaju naredbe koje uvjet provođenja ispituju na kraju ponovljivog bloka, to jest nakon što je blok linija kôda već izveden. Takva naredba je *do...while petlja*. Pogledajte sliku ispod (Slika 24).



Slika 24. Logičke strukture u implementaciji iteracije

Najčešće korištene petlje iz prve skupine su petlje *while* i *for*. Budući da se uvjet ispitivanja provođenja programskog bloka (omeđenog vitičastim zagradama) provjerava na početku, rezultat provođenja ovih logičkih struktura može biti da se programski blok ne izvrši niti jednom, da se izvrši jednom, da se izvrši više puta, pa čak i da se izvrši beskonačno puno puta (što ako nije kontrolirano predstavlja programsku pogrešku). Primjer kôda sa istim rezultatom napisan u *while* i *for* sintaksi nalazi se ispod.

```
int pocetna = 1990;
int zavrсна = 2020;
int trenutna = pocetna;
while (trenutna <= zavrсна) {
    if ((trenutna % 400 == 0) || ((trenutna % 4 == 0) && (trenutna % 100 != 0)))
        System.out.println(trenutna);
    trenutna++;
}
```

```
int pocetna = 1990;
int zavrсна = 2024;
for (int trenutna = pocetna; trenutna < zavrсна; trenutna++){
    if ((trenutna % 400 == 0) || ((trenutna % 4 == 0) && (trenutna % 100 != 0))){
        System.out.println(trenutna);
    }
}
```

Razmislite što radi prikazani kôd, te koje rezultate vraća verzija pisana s *while* petljom, a koje rezultate vraća verzija pisana sa *for* petljom. Pojasnite algoritam te matematičke i logičke operatore.

Najčešće korištena petlja iz druge skupine je *do...while* petlja. Ova petlja provodi provjeru uvjeta tek na kraju bloka kôda, te se kod primjene ove petlje ne može dogoditi da će programski blok biti ne izveden. Isti primjer programskog kôda koji ispisuje sve prijestupne godine u zadanom intervalu dan je u sintagmi *do...while* petlje ispod.

```
int pocetna = 1990;
int zavrсна = 2020;
int trenutna = pocetna;
```

```
do{
    if((trenutna % 400 == 0) || ((trenutna % 4 == 0) && (trenutna % 100 != 0))){
        System.out.println(trenutna);
    }
    trenutna++;
}
while(trenutna <= zavrсна);
```

```
I/System.out: 1992
I/System.out: 1996
I/System.out: 2000
I/System.out: 2004
I/System.out: 2008
I/System.out: 2012
I/System.out: 2016
I/System.out: 2020
```

Slika 25. Rezultat provedbe primjera s iteracijama

## 2.5 Metode i svojstva

Drugi važan iskorak u povećanju ponovne iskoristivosti pri pisanju računalnog kôda nastao je kreiranjem koncepta *metode* (funkcije, operacije). Metoda predstavlja programski blok koji se može sastojati od niti jedne, jedne ili više linija kôda čije izvršenje možemo pozvati po potrebi, te opcionalno proslijediti joj određene vrijednosti ili primiti odgovor.

```
private int ProvjeriPrijestupnostGodine(int godina)
{
    /*programski kôd*/
    return rezultat;
}
```

Iznad prikazani primjer sadrži potpunu **definiciju** metode, što znači da je ovim kôdom **deklarirana** metoda naziva *ProvjeriPrijestupnostGodine*, tipa *boolean*, kojoj je proslijeđen jedan **parametar**, te joj je odmah **inicijalizirano** tijelo, to jest programski kôd koji se izvršava ovom metodom. Sve metode koje nisu tipa *void* moraju završiti naredbom vraćanja vrijednosti. Opis definicije metode dan je i na slici ispod (Slika 26).

vidljivost	tip metode	naziv metode	parametri
private	int	ProvjeriPrijestupnostGodine	(int godina)
/*programski kod*/ return rezultat;			
tijelo metode			

Slika 26. Osnovna sintaksa definiranja varijabli u Javi

Primjer definiranja i korištenja metode naveden je u primjeru ispod. Prvi isječak programskog kôda sastavni dio je neke složenije strukture, te u **slijedu izvođenja operacija** sadrži i **definiranje varijabli**, to jest granica ispitivanja prijestupnosti godine. U primjeru su granice definirane kao *pocetnaGodina* i *zavrснаGodina*.

Primjenjujući **while** iterator programskim kôdom se promatra svaka godina između definiranih granica, uključujući i granice, te se u svakom koraku iterativnog bloka **poziva metoda *ProvjeriPrijestupnostGodine***. Metoda prima jedan parametra – promatranu godinu, a rezultat obrade vraća u obliku cijelog broja. Prihvaćeni rezultat se pohranjuje u privremenu varijablu *rezultatObrade* te se isti prosljeđuje drugoj metodi – *IspisiPojasnjenje*, koja ispisuje rezultat u jeziku razumljivom korisniku.

```
int pocetnaGodina = 2198;
int zavrснаGodina = 2204;
int promatranaGodina = pocetnaGodina;
while (promatranaGodina <= zavrснаGodina) {
    int rezultatObrade = ProvjeriPrijestupnostGodine(promatranaGodina);
    IspisiPojasnjenje(promatranaGodina, rezultatObrade);
    promatranaGodina++;
}
```

Metoda *ProvjeriPrijestupnostGodine* prima godinu kao parametra, te provjerava zadovoljava li godina uvjete prijestupnosti. Rezultat obrade može biti:

- 0 – godina je prijestupna
- 1 – godina nije djeljiva sa 4
- 2 – godina je djeljiva sa 4 i 100, ali nije sa 400

U ovoj se metodi primjenjuje **selekcija** te **operatori usporedbe** i **logički operatori** o kojima je također bilo govora u prethodnim poglavljima. Konačno u metodi *IspisiPojasnjenje*, koja prima dva različita parametra, se koristi **switch selekcija** te se korisniku ispisuju razumljive poruke.

```
private int ProvjeriPrijestupnostGodine(int godina) {
    int rezultat;
    if ((godina % 400 == 0) || ((godina % 4 == 0) && (godina % 100 != 0))) {
        rezultat = 0;
    }
    else {
        if ((godina % 4) != 0) {
            rezultat = 1;
        }
        else{
            rezultat = 2;
        }
    }
    return rezultat;
}

private void IspisiPojasnjenje(int godina, int rezultatObrade) {
    switch (rezultatObrade) {
        case 1: {
            System.out.println("Godina " + godina + " nije djeljiva sa 4!");
            break;
        }
        case 2: {
            System.out.println(
                "Godina " + godina + " je djeljiva sa 4 i 100, ali nije sa 400!");
            break;
        }
        default: {
            System.out.println(
                "Pa nema razloga. Godina " + godina + " je prijestupna!");
        }
    }
}
```

```
}
}
```

Dvije prikazane metode imaju različite povratne tipove, ali i različit broj i tipove parametara koje primaju. Ovdje je važno navesti da tip *void* predstavlja odsutnost povratne vrijednosti, zbog čega druga metoda niti nema *return* naredbu. Rezultat provedbe ovog programskog kôda prikazan je u slici ispod ().

```
I/System.out: Godina 2198 nije dijeljiva sa 4!
I/System.out: Godina 2199 nije dijeljiva sa 4!
I/System.out: Godina 2200 je djeljiva sa 4 i 100, ali nije djeljiva sa 400!
I/System.out: Godina 2201 nije dijeljiva sa 4!
I/System.out: Godina 2202 nije dijeljiva sa 4!
I/System.out: Godina 2203 nije dijeljiva sa 4!
I/System.out: Pa nema razloga. Godina 2204 je prijestupna!
```

*Slika 27. Rezultat provedbe primjera s metodama*

Pregled osnovnih koncepata programiranja u Javi predstavljen ovim poglavljem sve koncepte prezentira do razine primjenjivosti, ali ne ulazi u detaljne specifikacije, varijacije ili iznimke koje se mogu pojaviti pri klasičnom razvoju. Stoga, u svrhu boljeg razumijevanja Java programskog jezika i njegovih potpunih mogućnosti, svakako preporučamo konzultiranje sveobuhvatnije literature.

## 2.6 Objektno orijentirano programiranje

### 2.6.1 Ciljevi OOP-a

Bjarne Stroustrup, otac C++ programskog jezika, jednom je prilikom rekao da mali program može biti napisan u bilo kojem jeziku i bilo kako. Ako ne odustaješ suviše lako, na kraju ćeš sigurno učiniti da radi. Ali veliki program, to je drugačija priča. Ako ne koristiš tehnike dobrog programiranja, nove pogreške će se javljati jednako brzo kako ćeš uklanjati već postojeće pogreške [14].

Stroustrup je svakako bio u pravu, a možemo reći da je razlog za ovo velika međuovisnost različitih dijelova programa. Svaka promjena u jednom dijelu programa utječe na ostatak napisanog kôda i ukoliko se koristi strukturno ili proceduralno programiranje ovu međuovisnost je nemoguće u potpunosti kontrolirati. Rješenje za ovaj problem je kreiranje malih, u potpunosti neovisnih, dijelova kôda koji sa ostatkom aplikacije komuniciraju preko definiranih sučelja. Ovi dijelovi kôda nazivaju se moduli, a rješenje spomenutom problemu *modularnost*. Drugi veliki problem koji se nametnuo je bila nemogućnost *ponovne iskoristivosti* napisanog programskog kôda jer su dijelovi programskog kôda bili previše isprepleteni vezama koje se nije smjelo prekinuti, ili pak takva infrastruktura nije bila prisutna u određenoj aplikaciji. Na poslijetku, treći problem je bio *nemogućnost nadogradnje* aplikacija i svaka pa i najmanja promjena na postojećoj aplikaciji implicirala je mnogo nepotrebnih i suvišnih promjena ili pak pisanje cijele aplikacije iz početka.

Objektno orijentirana paradigma je zamišljena upravo na način da riješi sve spomenute probleme; da maksimizira modularnost, mogućnost ponovne iskoristivosti i mogućnost nadogradnje programskog kôda. Upravo zbog toga ovo poglavlje donosi koncepte objektno orijentiranog pristupa koji je danas najrašireniji pristup razvoju programskih proizvoda uopće.

Koncepti objektno orijentiranog programiranja (OOP-a) su razvijeni na način da primjena ovog pristupa pojednostavljuje razvoj i testiranje iznimno velikih i kompliciranih sustava. Kao neke od odlika objektno orijentiranog programiranja spomenut ćemo *apstrakciju podataka*, *učahurivanje*,

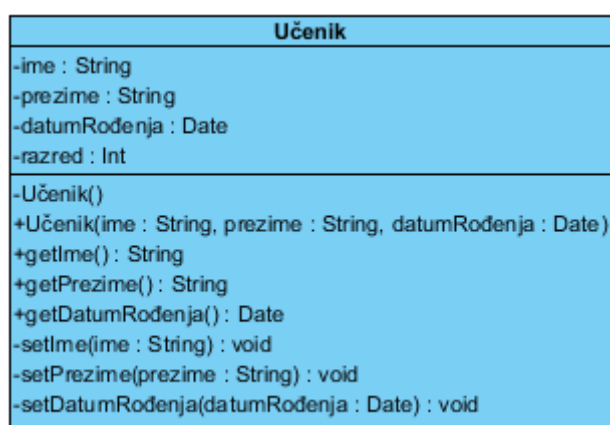


*nasljeđivanje* i *polimorfizam* kao i koncepte koji su korišteni da bi se spomenuti principi mogli ostvariti.

Programski jezik Java je u potpunosti objektno orijentirani programski jezik, dizajnirana upravo za razvoj objektno orijentiranih aplikacija i kao takva je odlična podloga za učenje i prihvaćanje OO strategije u razvoju mobilnih aplikacija.

## 2.6.2 Koncept objekta i razumijevanje OOP-a

Da bi u potpunosti iskoristili mogućnosti OOP-a, važnije je razumjeti metodologiju izgradnje programa koja se sada mijenja iz samih osnova, nego samu sirovu sintaksu nekog jezika koji sada ima značajku da osigura okruženje koje je potrebno OO programima da se izvršavaju. U ovom okruženju, najvažnija značajka je pojam *objekta*, koji objedinjava *podatke*, *metode* koje operiraju nad ovim podacima, te *načine komunikacije* s drugim objektima. Pokušajte prepoznati koncepte podataka i operacija promatrajući definiciju objekta pomoću jezika za modeliranje UML<sup>8</sup> ().



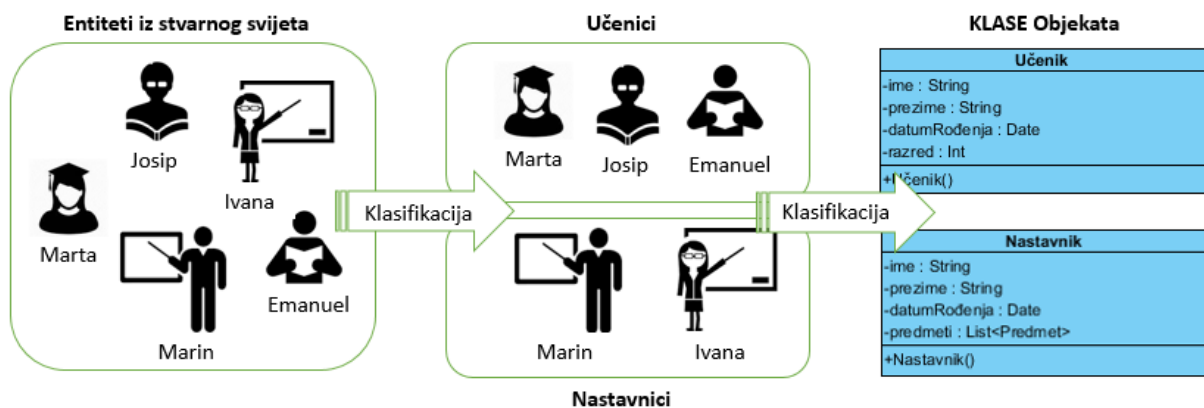
Slika 28. Uobičajen grafički prikaz objekta klase pomoću UML sintakse

U tradicionalnom programskom jeziku potrebno je specificirati svakom sastavnom dijelu proizvoda kojeg razvijamo *što raditi i kako to napraviti*. U OO sustavima možemo pretpostaviti da *objekt* već zna kako obaviti određene stvari i kod njegove primjene, potrebno je samo navesti *što treba raditi*. Ako pak i moramo programirati i dio „kako nešto napraviti“, onda to radimo samo jedanput, tu metodu pridružujemo željenom objektu, a potom ju samo koristimo. Navedena razlika najčešće se navodi i kao ključna razlika između treće i četvrte generacije programskih jezika.

Objekti koje sadrže programski sustavi četvrte generacije su programska analogija objekata iz realnog svijeta. Ključna razlika između objekata u OO sustavu i bilo koje komponente napisane u neobjektno orijentiranom jeziku je u tome što objekt po svojoj definiciji sadrži podatke i neophodni set metoda koje su odgovorne za manipuliranje tim podacima. S druge strane, u neobjektno orijentiranim jezicima programeri su odgovorni za eksplicitno povezivanje funkcija s podacima nad kojim one operiraju.

Dakle, prilikom definicije *objekta*, razmišljamo o cijeloj *klasi objekata*, a svojstva i metode koje opisuju jedinke te klase specificiramo u oblik atributa i metoda. Upravo zbog toga, prilikom definiranja, entitete iz stvarnog svijeta uvijek promatramo kao klasu istih objekata koje opisuju njihovim značajkama (atributima) te im dodjeljujemo radnje (metode). Primjer ovog procesa identificiranja klasa objekata je dan na slici ispod (Slika 29).

<sup>8</sup> UML – Standardni jezik za modeliranje (eng. Unified Modeling Language). Za detalje pogledajte <http://www.uml.org/>



Slika 29. Identificiranje klasa objekata

Tijekom cijelog procesa izgradnje programa, sličnosti između objekata od kojih se izgrađuje sustav moraju biti prepoznate i iskorištene. Pretpostavimo da mobilni programski proizvod koji izgrađujemo treba imati tri posebna odjeljka evidencije. Evidenciju o inventaru, zatim evidenciju prodaje i na posljetku evidenciju zaposlenika. Razlažući zahtjeve za izgradnju ovih evidencija, uvidjet ćemo da postoje određene komponente u svakoj evidenciji koje dijele zajednički koncept. Sustav za potporu inventara sadrži i podatke o dobavljačima, sustav za potporu prodaji sadrži i podatke o kupcima a sustav za evidenciju osoblja sadrži podatke o zaposlenima. Treba primijetiti da svaki od ovih objekata (dobavljač, kupac, zaposlenik) predstavlja osobu i da sve osobe sadrže neke zajedničke podatke (na primjer imaju ime i adresu). Upravo zbog toga, u sustavu mora postojati objekt većeg stupnja apstrakcije, koji bi definirao zajedničke karakteristike svakog od ovih elemenata.

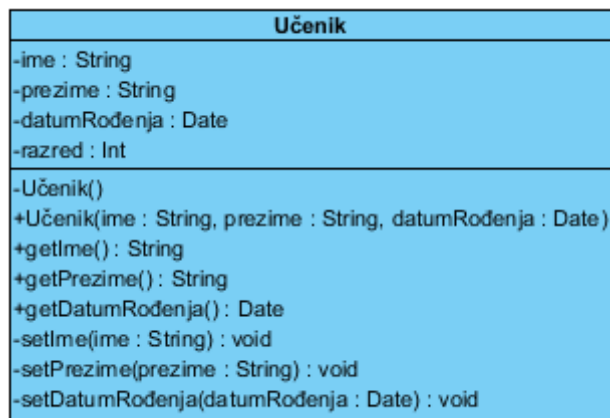
Svaki objekt je definiran *razredom* ili *klasom*, koju zapravo možemo predstaviti kao predložak (eng. template) za kreiranje specifičnih objekata koji se ponašaju na sličan način. Klase možemo grupirati u hijerarhiju, na čijoj će se najvišoj razini nalaziti klase čije su karakteristike sastavni dio klasa na nižim razinama. S druge strane, na najnižoj razini će se naći klasa sa specifičnim karakteristikama primjenjivim samo u njoj. Za primjer možemo navesti selekciju životinja. Promatrat ćemo mesojede, ali pri tome ne znamo radi li se o vukovima ili lavovima. Zapravo tu se radi o osobinama koje vukovi i lavovi imaju slične, na primjer da jedu meso. Ako se spustimo na razinu vuka ili lava, onda ćemo vidjeti da su ove dvije životinje uistinu različite. I upravo nam entitet *mesojeda* kao klasa najviše razine omogućava da na sličan način, bazirano na zajedničkim karakteristikama opišemo i vukove i lavove. Definirajući nove klase, mi ne moramo opisivati svaki aspekt objekta koji opisujemo, dovoljno je navesti samo razlike od već definiranih klasa.

Osnovno pravilo koje treba slijediti pri objektno orijentiranom dizajnu je da treba ignorirati posebne slučajeve tako dugo dok ih se može ignorirati [15, p. 10]. Naime, na početku procesa dizajna prvi korak treba biti potraga za sličnostima koje objekti imaju. Uz ovo, OO sustavi omogućavaju dizajnerima da izbjegnu specijalizaciju sve dok se ne izgradi generalni kostur, to jest struktura programa. Ovaj proces se naziva **generalizacija**, a tek kad je ona napravljena, prelazi se na **specijalizaciju**, koja je posljednji korak. Naime, specijalizaciju je moguće činiti onoliko dugo dok se ne iscrpi budžet naručitelja programa ili se ne zadovolje svi njegovi specifični uvjeti. Pri tome je važno spomenuti da poslije kvalitetno urađene generalizacije, može se konstantno specijalizirati sustav, a da se ne utječe na ništa što je već prije bilo napravljeno.

### 2.6.3 Učahurivanje

Učahurivanje (engl. Encapsulation) je prvi važan koncept OOP-a. Učahurivanje predstavlja uključivanje programskog kôda (metoda) i podataka (atributa, svojstava) u isti objekt. Učahurivanje

omogućuje smještanje podataka u onaj objekt koji sadrži i definiciju upravljanja tim podacima, te također omogućuje kontrolirani pristup do podataka kojima drugi objekti mogu pristupiti isključivo kroz eksplicitno definirana javna svojstva i/ili metode. Stoga, svi drugi objekti programskog proizvoda koji trebaju određene specifične podatke, dohvaćaju ih kroz objekt koji ih u sebi sadrži. Prednost učajurivanja je da samo jedan objekt mora znati kako se nositi sa posebnostima vlastitih podataka, a drugim objektima to uopće nije važno. Drugim riječima, učajurivanje omogućuje prikrivanje načina implementacije određene funkcionalnosti koja je poznata samo objektu u kojem je sadržana.

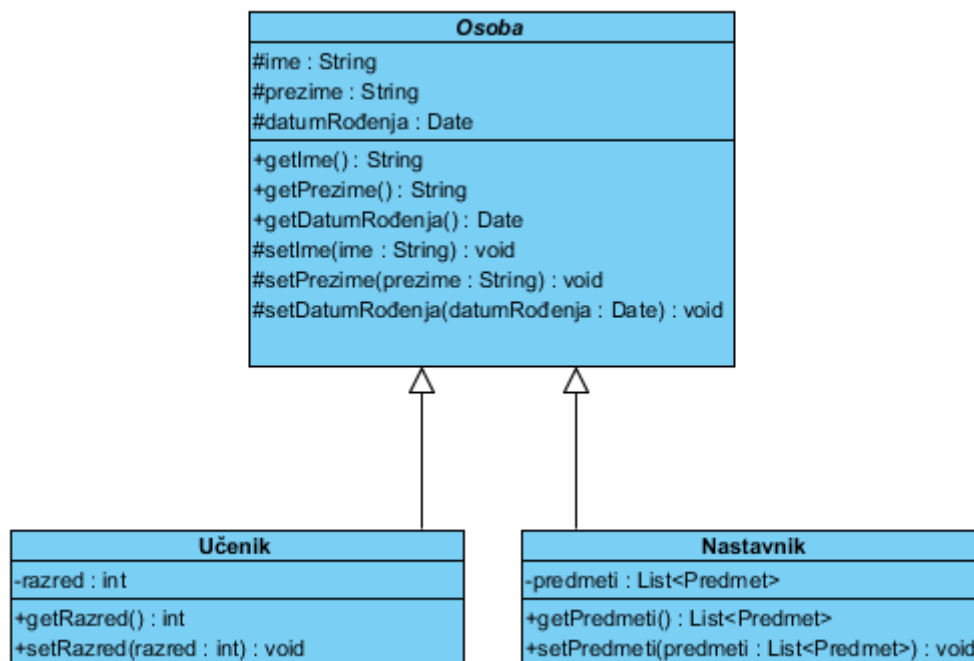


Slika 30. Učajurivanje

Na već prikazanom primjeru klase *Učenik* možemo vidjeti da su podaci kao što su *ime*, *prezime*, *datumRođenja* i *razred* skriveni sadržani u objektu koji ima metode za manipulaciju tim podacima. Budući da su spomenuti podaci privatni, ne može im se pristupiti osim kroz javno dostupne metode kao što su na primjer *getIme()*. Također možemo vidjeti da su pojedine metode također privatne (skrivena) što znači da nisu dostupne u sučelju ovog objekta prema drugim objektima, te ih drugi objektni ne mogu koristiti, niti mogu znati da one postoje.

#### 2.6.4 Nasljeđivanje

Govorili smo o hijerarhiji klasa i mogućnostima koje pruža OO pristup da više klasa niže razine koristi definirane elemente u klasi više razine. Ovaj koncept se naziva *nasljeđivanje* (eng. Inheritance). Izvedena klasa koristi podatkovne i funkcionalne mogućnosti od osnovne klase da bi učinila određene operacije. Primjer može biti generalizacija podataka iz klasa *Učenik* i *Nastavnik* u klasu *Osoba* (Slika 31). Naš definirani entitet *Osoba* omogućava svakoj od dvije specijalizirane klase da koriste podatke te metode kojima se manipulira tim podacima sadržane u klasi *Osoba*. Neposredno nakon definiranja klase *Učenik* izvedene iz klase *Osoba*, novo definirana klasa odmah može koristiti spomenute atribute i metode. Kažemo da je klasa *Učenik* naslijedila ove mogućnosti od klase *Osoba*, to jest naslijedila ih je od njoj nadređene klase.



Slika 31. Nasljeđivanje

Specijalizirana klasa može imati i vlastite atribute i metode, ali nasljeđivanje je izričito jednosmjerno i ovi atributi i metode ne mogu biti korišteni od strane nadređene klase.

Također, u primjeru iznad možemo vidjeti da je klasa *Osoba* apstraktna klasa, što znači da, budući da ne definira entitete iz stvarnog svijeta u potpunosti, ne može imati instance objekata. Svaki objekt instanciran u sustavu, ako se vodimo primjerom sa slike (Slika 29) mora biti ili učenik ili nastavnik, ali ne samo osoba.

Konačno, budući da se u Javi privatni atributi i privatne metode ne nasljeđuju, ukoliko im se treba eksplicitno pristupiti iz podklase, onda se moraju proglasiti javnima (svima dostupnima) ili zaštićenima (eng. Protected), što znači da su dostupni samo u klasama iz istog paketa. Na dijagramu, identifikator # predstavlja zaštićene elemente klase.

Definicija apstraktne klase može izgledati kao na primjeru ispod. Obratite pozornost na ključne riječi kojima se postiže željena funkcionalnost: *package*, *import*, *public*, *abstract*, *class*, *protected* i *this*, te na tipove podataka korištene u primjeru: *Date*, *String* i *void*.

```

package hr.foi.prirucnik.nasljedjivanje;

import java.util.Date;

public abstract class Osoba {
    protected String ime;
    protected String prezime;
    protected Date datumRodjenja;

    public String getIme() { return ime; }
    public String getPrezime() { return prezime; }
    public Date getDatumRodjenja() { return datumRodjenja; }

    protected void setIme(String ime)
    { this.ime = ime; }

    protected void setPrezime(String prezime)
    
```

```

    { this.prezime = prezime; }

    protected void setDatumRodjenja(Date datumRodjenja)
    { this.datumRodjenja = datumRodjenja; }
}

```

Primjer klase *Učenik* koja nasljeđuje klasu *Osoba* može izgledati kako je prikazano ispod. Ključna riječ na nasljeđivanje je *extends*, što u doslovnom prijevodu znači da klasa *Učenik* nadograđuje klasu *Osoba*.

```

package hr.foi.prirucnik.nasljedjivanje;

public class Ucenik extends Osoba {
    private int razred;

    public int getRazred() { return razred; }
    public void setRazred(int razred)
    { this.razred = razred; }
}

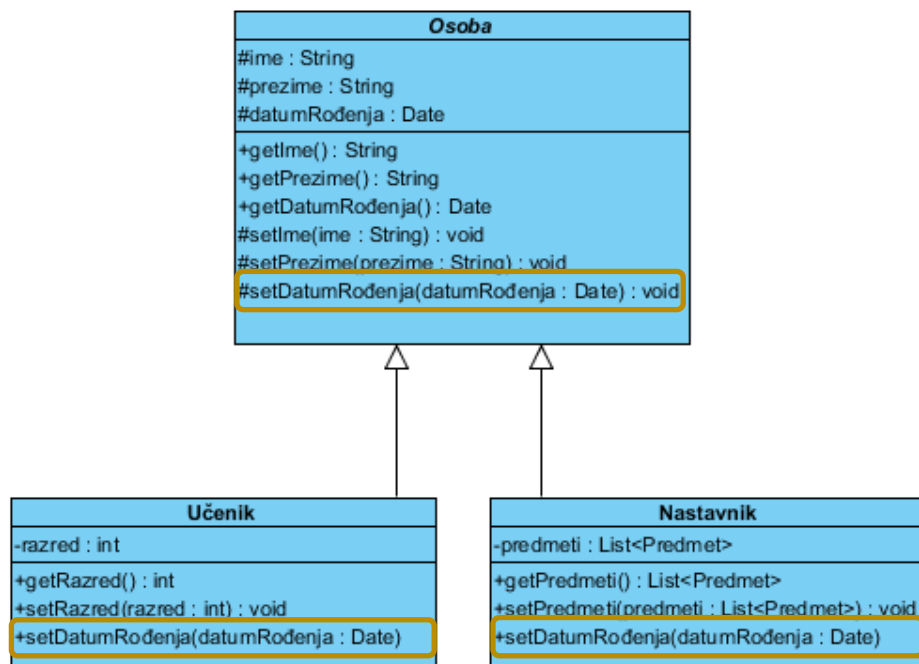
```

Iz gornjeg primjera se može vidjeti razmjerno malo kôda za opis klase *Učenik*, iako objekt klase *Učenik* ima sve mogućnosti sadržane u klasi *Osoba* uz dodatno svojstvo *razred* i metode za njegovo čitanje i pisanje.

### 2.6.5 Polimorfizam

Polimorfizam (engl. Polymorphism) je treće važno svojstvo, bez kojeg objektni pristup ne bi mogao postojati. Polimorfizam zapravo omogućava definiranje operacija koje su ovisne o tipu. Već smo prije rekli kako se svakom članu osnovne klase može pristupiti u izvedenoj klasi. To zapravo omogućava očuvanje integriteta objekta, ali to ne znači da metode iz izvedene klase moraju isto raditi što i metode u osnovnoj klasi, iako su naslijeđene. Naime, u objektno orijentiranom pristupu moguće je svaku naslijeđenu metodu na više načina preoblikovati, da u karakterističnoj situaciji radi upravo ono što mi želimo, pa čak ako se to razlikuje od onog što smo naslijedili. Mogućnost korištenja metoda osnovne klase, ali i promijenjenih metoda u izvedenoj klasi se naziva polimorfizam. Drugim riječima možemo reći da isti objekt može imati više oblika, a na programeru je da koristi onaj oblik koji mu je u karakterističnom trenutku potreban.

Polimorfizam možemo pojasniti i na našem primjeru klasa *Osoba*, *Učenik* i *Nastavnik*. Iako obje specifične klase nasljeđuju metodu *setDatumRođenja*, zbog specifičnosti logike provjere ispravnosti datuma rođenja kod učenika i provjere datuma rođenja kod nastavnika, ove dvije klase mogu preoblikovati / premostiti (eng. Override) spomenutu metodu te omogućiti specifičnu vlastitu implementaciju.



Slika 32. Polimorfizam

Datum rođenja učenika, u trenutku upisa mlađeg od 6 godina vjerojatno nije ispravan, dok bi s druge strane datum rođenja nastavnika, u trenutku upisa mlađeg od 18 godina također bio neispravan. Primjer ispod pokazuje preoblikovanu metodu `setDatumRođenja` koja umjesto da datum jednostavno upiše u lokalnu varijablu (kako je naslijeđeno od klase `Osoba`) pruža vlastitu implementaciju te uz pomoć jedne privatne metode vrši dodatne logičke provjere. Ovdje je potrebno primijetiti korištenje ključne riječi `Override` koja anotira (`@`) metodu te tako kompajleru daje do znanja da zanemari naslijeđenu metodu i u prijevodu koristi ovu preoblikovanu metodu. Preoblikovana metoda uvijek mora imati isto ime kao i naslijeđena metoda, a u Javi se prije svake metode koju se preoblikuje dodaje anotacija preoblikovanja (`@Override`).

```

package hr.foi.prirucnik.polimorfizam;

import java.util.Date;

public class Ucenik extends Osoba {
    private int razred;

    public int getRazred() { return razred; }
    public void setRazred(int razred)
    { this.razred = razred; }

    @Override
    public void setDatumRodjenja(Date datumRodjenja)
    {
        //učenik ne može biti mlađi od 6 godina
        if (dohvatiStarost(datumRodjenja) < 6)
        {
            throw new IllegalArgumentException
                ("Datum rođenja učenika nije točan!");
        }
        else
        {
            this.datumRodjenja = datumRodjenja;
        }
    }
}
    
```

```

private int dohvatiStarost(Date datum)
{
    //dio kôda irelevantan u ovom kontekstu
}
}

```

Međutim, polimorfizam se ogleda tek kroz kontekst korištenja klase *Učenik* u nekoj trećoj klasi. Tada, klasa *Učenik* po potrebi može poprimiti oblik klase *Osoba* ili oblik klase *Učenik*, pri čemu će u jednom slučaju prikazati svojstva i metode dostupne u klasi *Osoba*, a u drugom slučaju svojstva i metode dostupne u klasi *Učenik*. Ovdje je važno spomenuti da će preoblikovana metoda uvijek biti izvršena, pa čak i kad se učenik predstavlja kao *Osoba*. Ako želimo pristupiti metodi iz nadklase onda moramo koristiti ključnu riječ *super*. Isječak iz kôda koji prikazuje polimorfizam pokazan je u nastavku.

```

Calendar cal = Calendar.getInstance();
cal.set(1996, 10, 17);
Date datumRodjenja = cal.getTime();

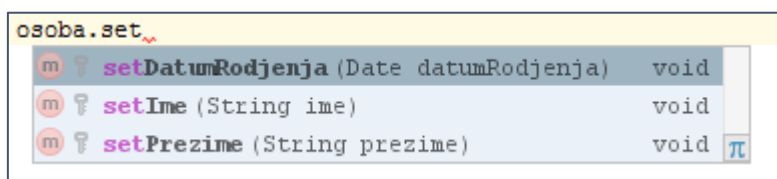
Ucenik ucenik = new Ucenik();
ucenik.setIme("Marko");
ucenik.setDatumRodjenja(datumRodjenja);
ucenik.setRazred(1);

Osoba osoba = ucenik;
osoba.setIme("Marko");
osoba.setDatumRodjenja(datumRodjenja);

```

U prethodnom isječku programskog kôda, prvi segment se odnosi na upis datuma u objekt tipa *Date*. Zbog odbacivanja većine mogućnosti zastarjelog paketa *java.util.Date*, u novijim verzijama se alternativno može koristiti paket *java.util.Calendar*.

Promatrajući objekt *ucenik* u obliku tipa *Ucenik*, primijetimo da su dostupna sva naslijeđena, ali i dodatna specifična svojstva i metode. Međutim, promatrajući ga u obliku tipa *Osoba* možemo vidjeti da za korištenje više nema specifičnih svojstava i metoda kao što je npr. metoda *setRazred()*.



Slika 33. Sučelje objekta kroz oblik nadklase

U Java programskom jeziku svaka klasa nasljeđuje klasu *Object*, bilo indirektno kroz neku drugu naslijeđenu klasu, ili direktno ako klasa eksplicitno ne nasljeđuje niti jednu drugu klasu. To znači da se objekt bilo koje korisnički definirane klase može u svakom trenutku preoblikovati u tip *Object* te pri tome koristiti osnovna svojstva i metode naslijeđene od klase *Object*.

### 2.6.6 Sučelje

Pri razvoju mobilnih aplikacija za Android, izuzetno je važno razumjeti i znati primijeniti pojam *sučelja* (eng. Interface), te stoga smatramo važnim predstaviti sučelja kao jedan od specifičnih koncepata objektno orijentirane paradigme.

Sučelje predstavlja poseban stereotip klase koja sadrži javnu definiciju, ali ne i implementaciju metoda. Za usporedbu, već smo spomenuli neke od stereotipa klase. Za vježbu, samostalno istražite i proučite specifičnosti osnovnih stereotipa: apstraktna klasa, konkretna klasa, struktura, enumeracija.

```
package hr.foi.prirucnik.sucelja;

import java.util.Date;

public interface IOsoba {
    public String getIme();
    public String getPrezime();
    public Date getDatumRodjenja();

    public void setIme(String ime);
    public void setPrezime(String prezime);
    public void setDatumRodjenja(Date datumRodjenja);
}
```

Druge klase mogu implementirati (to jest realizirati) sučelje, ali ga ne mogu naslijediti. Svaka klasa koja implementira sučelje ugovorno jamči da će sadržavati sve javne metode i druge konstrukte koji su definirani tim sučeljem.

```
package hr.foi.prirucnik.sucelja;

import java.util.Date;

public abstract class Osoba implements IOsoba {
    protected String ime;
    protected String prezime;
    protected Date datumRodjenja;

    public String getIme() { return ime; }
    public String getPrezime() { return prezime; }
    public Date getDatumRodjenja() { return datumRodjenja; }

    protected void setIme(String ime)
    { this.ime = ime; }

    protected void setPrezime(String prezime)
    { this.prezime = prezime; }

    protected void setDatumRodjenja(Date datumRodjenja)
    { this.datumRodjenja = datumRodjenja; }
}
```

Klasa se ugovorno obvezuje na implementaciju sučelja pomoću ključne riječi *implements*. Kako je prikazano u prethodnom kôdu, klasa može učahuriti i druge podatkovne konstrukte te metode za pristup i manipulaciju istima, ali pri tome ne smije narušiti potpunu implementaciju ugovornog sučelja.

Drugim riječima, mogućnost korištenja sučelja još je jedna od blagodati polimorfizma i objektno orijentirane paradigme. Primjer višeobličja objekta *ucenik* možemo vidjeti u primjeru ispod.

```
Ucenik ucenik = new Ucenik();
ucenik.setIme("Marko");
ucenik.setDatumRodjenja(datumRodjenja);
ucenik.setRazred(1);

IOsoba osoba = ucenik;
```



```
osoba.setIme("Marko");  
osoba.setDatumRodjenja(datumRodjenja);  
osoba.setRazred(1);
```

Koncept sučelja se u razvoju mobilnih aplikacija za Android koristi u različite svrhe, ali najčešća uporaba je u svrhu postizanja *modularnosti* te implementacije koncepta *događaja*. Ako se dva ili više modula obvežu implementirati unaprijed definirano sučelje, onda je za aplikaciju potpuno svejedno koja se od tih implementacija koristi, budući da obje imaju iste unaprijed definirane i u sučelju opisane funkcionalnosti. Naravno, u kontekstu izrade modula, dobar arhitekturni dizajn znači da se sva komunikacija s modulom vrši isključivo kroz sučelje. S druge strane, kod implementacije koncepta događaja, sučelje pomaže da se osigura da svi zainteresirani objekti ugovorno jamče da će sadržavati metodu koju se pozove kada se događaj dogodi.

Jedan od načina razumijevanja sučelja (eng. Interface) je pomoću fizičkih sučelja na vašem osobnom računalu. Vrlo dobar primjer je USB sučelje koje definira vezu između različitih vanjskih uređaja i vašeg računala. USB port nema vlastitu funkcionalnost, ali zato opisuje protokol razgovora kojeg svaka vanjska jedinica koja se želi spojiti na računalo mora zadovoljiti. Pomoću istog USB sučelja i protokola, vaše računalo razgovara s mišem, tipkovnicom, printerom i slično.

## 2.7 Pitanja za provjeru znanja

1. Koje su karakteristike i dobre osobine *Java* programskog jezika?
2. Što definira i čemu služi „*tip podataka*“?
3. Navedite i pojasnite osnovnu sintaksu definiranja varijabli u Javi.
4. Koji su osnovni tipovi podataka?
5. Koja je razlika između jednostavnih (osnovnih) i složenih (izvedenih) tipova podataka?
6. Pojasnite pojam *prioritet operanda* i njegove implikacije pri primjeni.
7. Kako se implementira i čemu služi pretvorba tipova podataka?
8. Navedite specifičnosti znakovnih tipova podataka u Javi?
9. Koje su karakteristike, prednosti i nedostaci logike vođene događajima?
10. Koje su osnovne logičke strukture koje se pojavljuju pri izvršavanju programa? Pojasnite ih.
11. U kojim slučajevima se preporuča koristiti selekciju tipa *switch*?
12. Koji su mogući rezultati evaluacije jednostavnih, a koji složenih logičkih izraza?
13. Koja je razlika između logičkih operatora i operatora usporedbe u Javi?
14. Kreirajte matematičko-logički izraz koji provjerava parnost broja?
15. Koja je osnovna razlika između *while* i *do-while* petlje?
16. Napišite i testirajte programski kôd koji ispisuje sve parne brojeve u nizu od 84 do 231.
17. Koja je uloga primjene metoda pri razvoju programskog proizvoda?
18. Izmijenite programski kôd iz 16. kako bi se provjera parnosti vršila pomoću metode.
19. Pojasnite pojam „*objektno orijentirano*“.
20. Koja je osnovna razlika između objektno orijentiranog i strukturnog pristupa razvoju?
21. Koji su osnovni ciljevi primjene objektno orijentiranog programiranja?
22. Pojasnite pojmove *objekta* i *klase objekata*. Čemu služi klasifikacija?
23. Koji su osnovni koncepti objektno orijentiranog programiranja?
24. Pojasnite razliku između *učahurivanja* i *nasljeđivanja*. Čemu služi prvi, a čemu drugi koncept?
25. Kreirajte model klasa *Prijevozno sredstvo – Automobil – Autobus*, te na njemu pojasnite pojam *polimorfizam*?
26. Čemu služe sučelja u Javi?

## 2.8 Resursi za samostalan rad

- ✓ **Fain Yakov.** Programiranje Java. Wrox/IT Expert; 2015.
- ✓ **Liang Y. Daniel.** *Introduction to JAVA Programming - Comprehensive Version*, 9. izdanje. Boston: Pearson, 2013.
- ✓ **Bergin J, Stehlik M, Roberts J, Pattis R.** Karel J Robot: a gentle introduction to the art of object-oriented programming in Java. Dreamsongs Press; 2013.
- ✓ **Wu C. Thomas.** An introduction to object-oriented programming with Java. 5. izdanje. Boston: McGraw Hill Higher Education; 2010.
- ✓ **Perry J. Steven.** *IBM Introduction to Java Programming*: <http://www.ibm.com/developerworks/java/tutorials/jintrotojava1/>, pristupano u srpnju, 2016.
- ✓ **Oracle Web.** *Object-Oriented Programming Concepts*, <https://docs.oracle.com/javase/tutorial/java/concepts/>, pristupano u srpnju 2016

### 3 RAZVOJ ANDROID APLIKACIJA

*Ovo poglavlje donosi sažet prikaz mogućnosti i osnove rada sa integriranim razvojnim okruženjem za razvoj aplikacija za Android – Android Studiom. Također, u ovom poglavlju se prikazuje osnovni set klasa za razvoj Android aplikacija (Android SDK), pojašnjava temeljni koncept aplikacije – aktivnost, pojašnjava programska logika i višeslojnost aplikacije i konačno vodi korisnika kroz proces kreiranja, kompiliranja, pokretanja i testiranja prvog projekta. Poseban naglasak je stavljen na odnos aktivnosti i pripadajućih resursa, te automatski kreirane i održavane klase R.*

#### SADRŽAJ POGLAVLJA

Integrirano razvojno okruženje .....	56
Android SDK.....	61
Programska logika Android aplikacije .....	64
Alati Android Studija .....	72
Dodatni resursi .....	77

## 3.1 Integrirano razvojno okruženje

### 3.1.1 Klasifikacija razvojnih okruženja

Razvoj Android aplikacija možemo provoditi koristeći različite alate i/ili razvojna okruženja. Sukladno temeljnim razlikama sve alate možemo klasificirati u tri osnovne skupine:

- Integrirana razvojna okruženja za razvoj Android aplikacija
- Razvojna okruženja za više-platfomski razvoj
- Generatori aplikacija

Integrirana razvojna okruženja za razvoj Android aplikacija (eng. Integrated Development Environment (IDE)) predstavljaju cjeloviti skup alata za provedbu svih aktivnosti u procesu razvoja, testiranja i objave Android aplikacije. U usporedbi s ostalim alatima, na tržištu se nalazi tek mali broj cjelovitih integriranih razvojnih okruženja dizajniranih specifično za Android razvoja. Prema [16], trenutno se mogu koristiti samo četiri razvojna okruženja (Tablica 10).

*Tablica 10. Android integrirana razvojna okruženja*

Naziv	Programski jezik
Android Studio	Java
Basic4Android	BASIC
IntelliJIDEA	Java
RFO Basic	BASIC

Na spomenutom popisu se više ne nalazi *Eclipse* razvojno okruženje koje je do 2015. godine bilo jedino službeno podržano od strane Googlea. Međutim, budući da se *Eclipse* (kao okolina korištena za različite svrhe) od početka koristio kao IDE „dok se ne kreira novi IDE isključivo za Android“, različiti proizvođači su pokušali kreirati vlastitu okolinu. Jedan od takvih je i IntelliJ, na kojem je nastao IntelliJIDEA, a potom i Android Studio kojeg Google preuzima kao novo službeno razvojno okruženje. Stoga, u ovom poglavlju ćemo se isključivo baviti opisom najvažnijeg predstavnika ove skupine – Android Studiom.

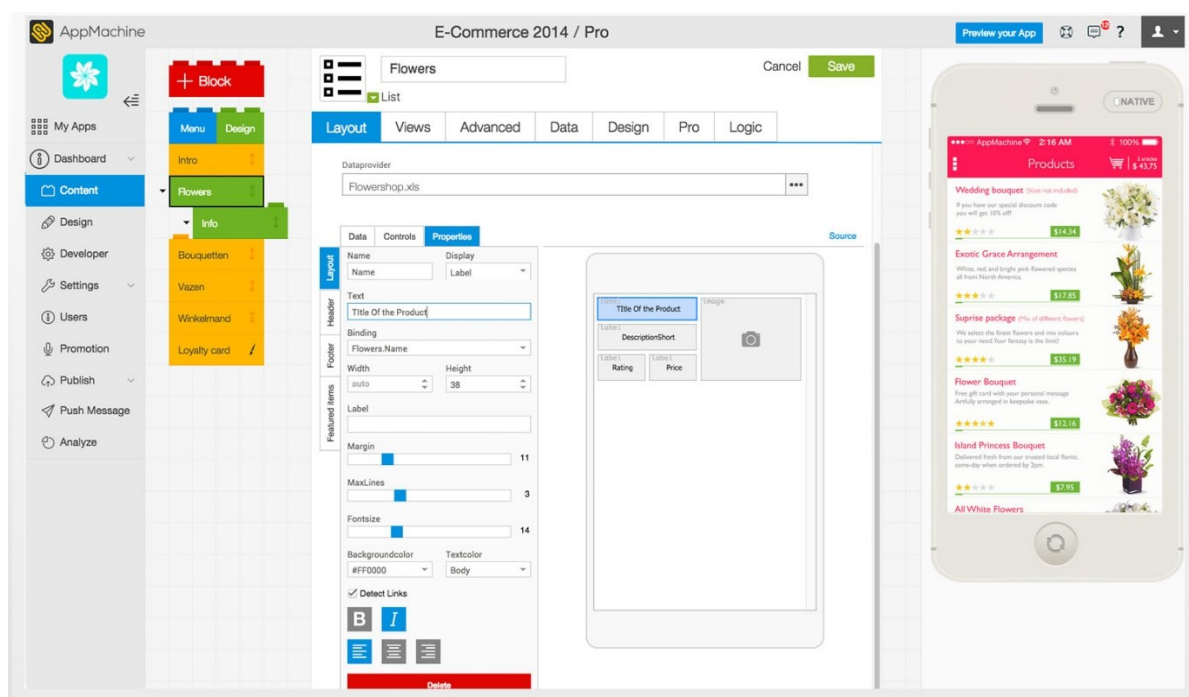
Razvojna okruženja za više-platfomski razvoj predstavljaju skupinu alata koji jedan programski kôd automatski transformiraju u jedan ili više ciljanih programskih kôdova. Razvojni inženjer može pisati programski kôd u specifičnom jeziku (najčešće ograničenom na domenu – DSL-u) ili u nekom od postojećih poznatih programskih jezika, a potom koristeći alate iz IDE-a za višeplatfomski razvoj automatski kreira programski kôd za Android i/ili neke druge mobilne platforme. Trenutno ima više od 150 različitih dostupnih sustava za više-platfomski razvoj, od kojih su neki spremni za komercijalnu uporabu. Sukladno [16], neki od najpoznatijih više-platfomskih rješenja, uključujući i programske jezike u kojima se programira osnovna aplikacija prikazani su u tablici ispod (Tablica 11).

*Tablica 11. Više-platfomska rješenja*

Naziv	Programski jezik
AIDE (Android IDE)	HTML5/C/C++
Application Craft	HTML5
Cordova	HTML5
Corona	Lua
Intel XDK	HTML5
Kivy	Python

MIT App Inventor	Blocks
Monkey X	BASIC
MonoGame	C#
MoSync	HTML5/C/C++
NS BASIC	BASIC
PhoneGap	HTML5
RAD Studio XE	Object Pascal, C++
RhoMobile Suite	Ruby
Telerik	HTML5
Titanium	JavaScript
Xamarin	C#

Generatori aplikacija predstavljaju alate za razvoj programskih proizvoda koji temeljem generičkog ulaza i/ili specifikacije mogu automatski kreirati Android aplikaciju. Razvojni inženjer ima malo ili nimalo kontrole nad programskim kôdom, a ponašanje aplikacije može izmijeniti isključivo izmjenom ulazne specifikacije u generator. Većina ovih alata su još uvijek u fazi prototipa te nisu komercijalno primjenjivi. Prema [16], primjeri generatora aplikacija su AppMakr<sup>9</sup>, GoodBarber<sup>10</sup>, AppyPie<sup>11</sup> i AppMachine<sup>12</sup>. Sučelje AppMachine generatora prikazano je na slici ispod (Slika 34).



Slika 34. Sučelje AppMachine generatora

### 3.1.2 Android Studio

Android Studio je trenutno jedino službeno podržano integrirano razvojno okruženje za razvoj Android aplikacija [17]. Sastoji se od niza alata koji služe za razvoj mobilnog programskog proizvoda,

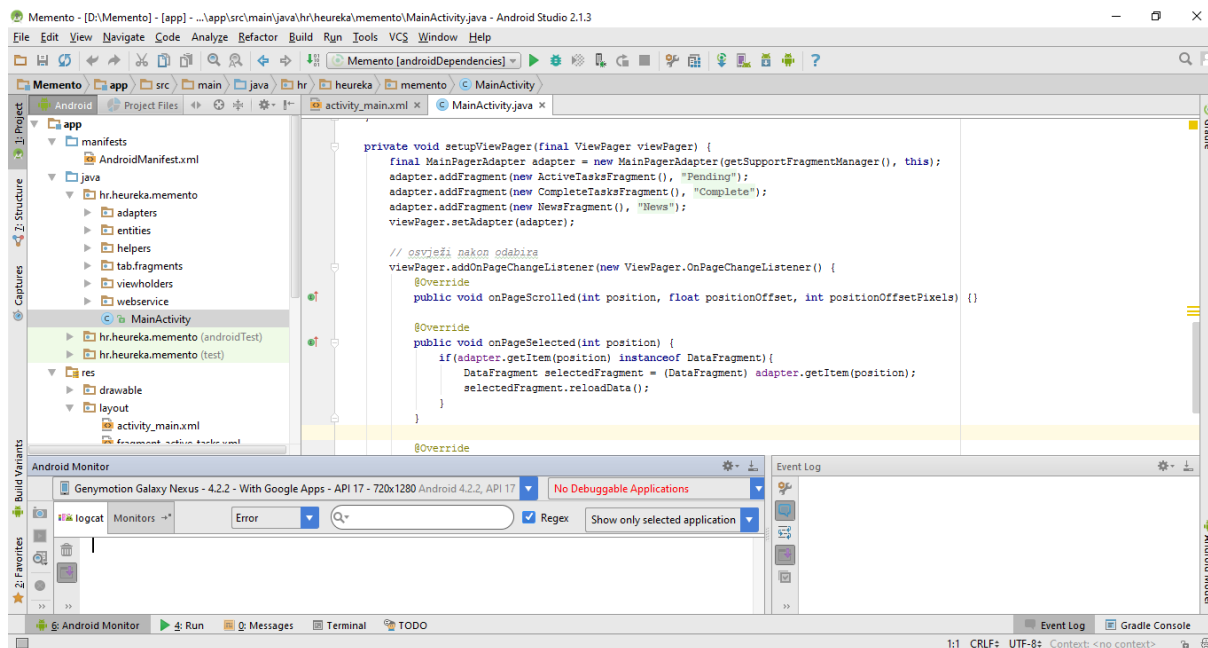
<sup>9</sup> <http://www.appmakr.com/>

<sup>10</sup> <https://www.goodbarber.com/>

<sup>11</sup> <http://www.appypie.com>

<sup>12</sup> <http://www.appmachine.com>

uključujući alate za dizajn korisničkog sučelja, pisanje programskog kôda, testiranje i otklanjanje pogrešaka, komunikaciju sa mobilnim uređajem, pripremu programskog proizvoda, pripremu aplikacije za objavu i slično. Kako bi ste instalirali Android Studio, posjetite službenu web stranicu i slijedite upute za preuzimanje i instaliranje razvojnog okruženja: <https://developer.android.com/studio/index.html>.



Slika 35. Osnovno sučelje Android Studio razvojnog okruženja

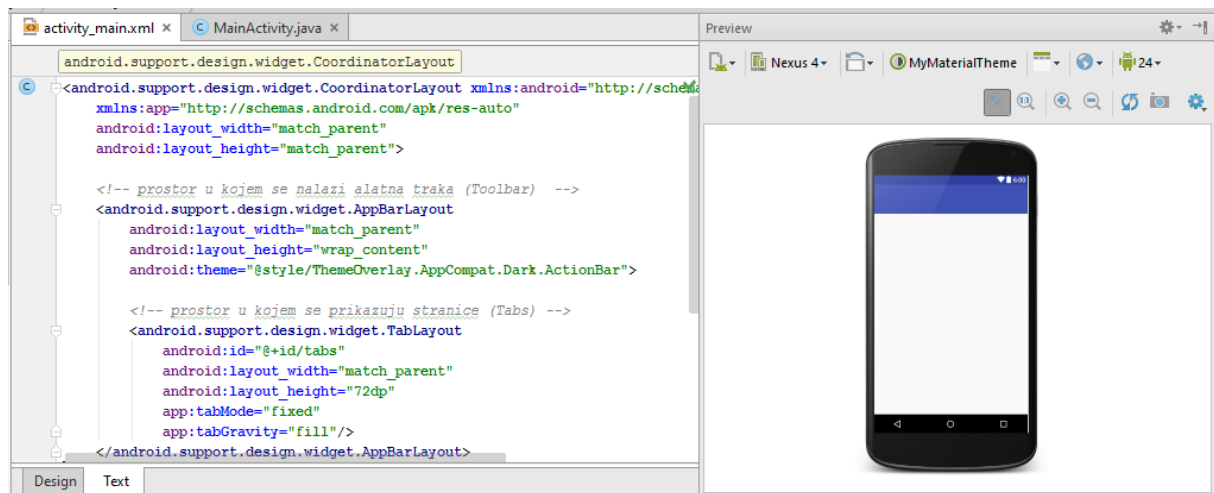
Pogled prikazan na slici iznad (Slika 35) predstavlja osnovni razvojni pogled unutar Android studio razvojnog okruženja, ali nakon otvaranja postojećeg ili kreiranja novog projekta. Pogled je podijeljen u pet osnovnih dijelova. U vrhu se nalazi traka s alatima, s lijeve strane je panel strukture projekta, desno se nalazi radna površina, a dolje su u osnovnom pogledu paneli za nadgledanje statusa uređaja (lijevo) i nadgledanje poruka o događajima (desno).

Korisnik može po potrebi razmješati alate po ekranu koristeći klasični „povuci i pusti“ princip, a važno je spomenuti da se osnovni pogled uvijek može vratiti odabirom opcije *Windows -> Restore Default Layout*.

Mnogi alati su dostupni na radnoj površini, ali su privremeno skriveni, te ih se može otvoriti klikom na listove (eng. Tab) s njihovim nazivima koji su raspoređeni sa svih strana razvojnog okruženja. Neki od skrivenih alata na prikazanoj slici su npr. *Android Model* i *Gradle Console* (dolje desno), *Run* ili *Favorites* (dolje lijevo), *Structure* (gore lijevo) i *Gradle* (gore desno). Konačno, ovo je samo dio alata budući da se mnogi od njih automatski pokreću u pozadini, ili su dostupni isključivo na zahtjev korisnika, ili se pak automatski prikazu u nekoj drugoj perspektivi kao što je perspektiva za otklanjanje pogrešaka.

Važno svojstvo panela radne površine (u kojem je na slici iznad prikazan programski kôd) jeste da se dvostrukim klikom na list naziva nekog dokumenta, taj dokument otvara preko cijelog ekrana. Za povratak na originalni pogled potrebno je ponovno dvostruko kliknuti na isti list.

Za kraj, neki od spomenutih panela imaju vlastitu hijerarhiju (pod listove) kojoj se pristupa klikom na naziv podlista prikazanog u promatranom panelu. Tako na primjer, promatrajući radni dokument *activity\_main.xml* otvoren u radnom panelu, u donjem lijevom kutu dokumenta (i panela) možemo vidjeti opcije prikaza u *design* ili *text* pogledu (Slika 36).



Slika 36. Prikaz radnog lista (panela) s podlistovima

Nakon otvaranja postojećeg ili kreiranja novog projekta s nastavnikom, pokušajte otvoriti spomenute alate te provjeriti koje vam sve opcije pružaju.

Prilikom razvoja primjera iz ovog udžbenika, nećemo koristiti sve Android studio alate, ali ćemo se svakako dotaknuti onih osnovnih.

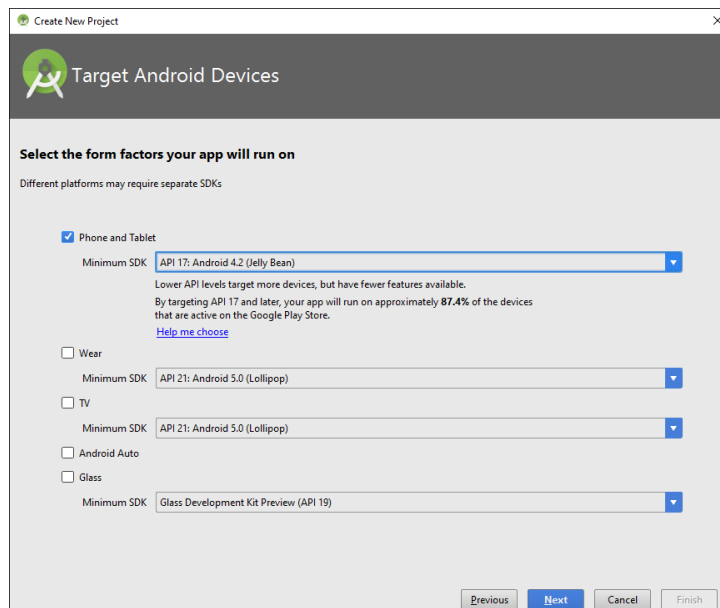
### 3.1.3 Kreiranje novog projekta

Proces kreiranja novog projekta u Android Studiju je vođen čarobnjakom za izradu projekata. Sve odluke koje korisnik mora donijeti tijekom ovog procesa mogu naknadno biti promijenjene. Proces kreiranja se sastoji od sljedećih osnovnih koraka:

1. Pokrenuti Android Studio. Pri prvom pokretanju niti jedan projekt neće biti automatski otvoren. Odaberi opciju „Start a new Android Studio project“.
2. Upisati osnovne podatke o projektu:
  - a. naziv aplikacije,
  - b. domenu iz koje se automatski kreira korijenski paket proizvoda, te
  - c. lokaciju projekta na disku.

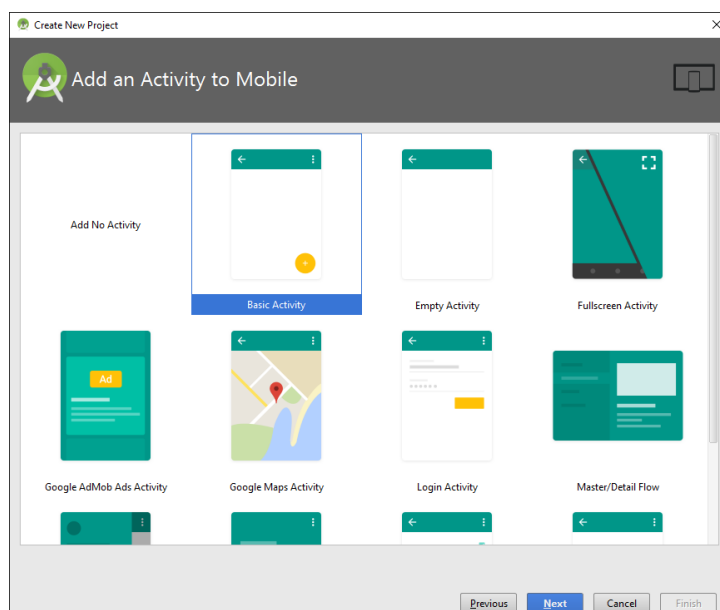
Korijenski paket je važan pojam koji predstavlja jedinstvenu identifikacijsku oznaku vaše aplikacije. Obično svi paketi/proizvodi neke tvrtke imaju isti korijen, ali različit naziv proizvoda. Korijen na primjer može sadržavati i oznaku države i naziv same tvrtke.

3. Odabir uključenih modula / ciljanih platformi i specifičnih verzija. Za odabir su dostupni, ali neobavezni razvojni okviri: *Mobilni uređaji*, *nosivi uređaji (wear)*, *TV*, *android auto* i *naočale (glass)*. Odabirom novije verzije razvojnog okruženja, razvojni inženjer ima dostupne nove mogućnosti, ali mora žrtvovati dio korisnika kod kojih se aplikacija pisana u novom razvojnom okruženju ne može izvršavati. S druge strane, odabirom starije verzije razvojnog okruženja imamo pristup većem broju korisnika, ali na uštrb novih mogućnosti. Tako na primjer, u trenutku pisanja ovog udžbenika, odabirom verzije API 19 (Android 4.4), dostupno je bilo 74% korisnika, za API 21 (Android 5) oko 40.5% korisnika, a za API 24 (Android 7) manje od 1% korisnika.



Slika 37. Odabir uključenih modula

4. *Odabir tipa početne aktivnosti* – Android Studio nudi više mogućnosti za odabir tipa početne aktivnosti, što ovisi o odabranoj minimalno podržanoj SDK verziji. Neke od mogućih tipova aktivnosti su *osnovna aktivnost*, *prazna aktivnost*, *aktivnost preko cijelog ekrana*, *aktivnost za prikaz reklama*, *aktivnost za prikaz mape*, *aktivnost za autentikaciju korisnika* i slično. Pojam *aktivnosti* će biti detaljno pojašnjen u sljedećim poglavljima.

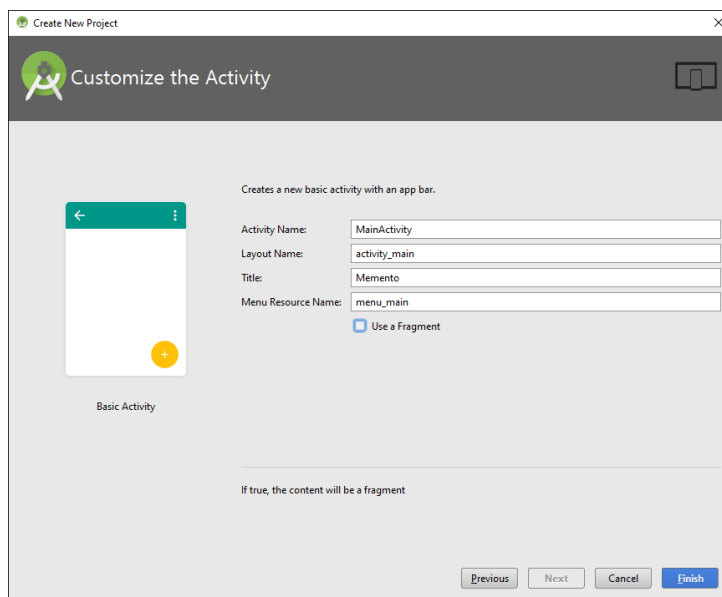


Slika 38. Odabir početne aktivnosti

5. *Imenovanje aktivnosti i resursa*. U ovisnosti o vrsti odabrane aktivnosti, u ovom koraku se definiraju imena uključenih dokumenata i resursa. Važno je primijetiti da se aktivnosti pišu u Java kôdu te se i imenuju sukladno pravilima imenovanja klasa o kojima smo govorili u poglavlju o osnovama objektno orijentiranog programiranja. S druge strane, resursi su xml dokumenti, te se imenuju malim slovima s riječima odvojenim donjom crtom ( `_` ) kako bi java



klase i XML resursi bili logički odvojeni i kako nazivi ne bi zbunjivali razvojnog inženjera pri programiranju.



*Slika 39. Imenovanje aktivnosti i resursa*

6. *Završavanje.* Klikom na *Finish* bit će kreiran projekt odabrane konfiguracije i otvorit će se osnovni Android Pogled pojašnjen u prethodnom primjeru.

## 3.2 Android SDK

Kako bi smo razumjeli ostale koncepte pojašnjene u ovom poglavlju, potrebno je pojasniti pojam *Android SDK* koji se odnosi na set klasa i osnovnih alata ciljano kreiranih za razvoj aplikacija za Android. Ovaj set klasa i pripadajućih alata je temeljen na Java setu klasa (Java SDK), te zajedno sa alatima dostupnim u Android Studiju čini minimum potreban za razvoj aplikacija za Android. Prvi set klasa za Android (API 1) objavljen je 2008. godine, a posljednji je API 24 koji se može pokretati samo na Android 7 operacijskom sustavu, te je objavljen u kolovozu 2016. godine.

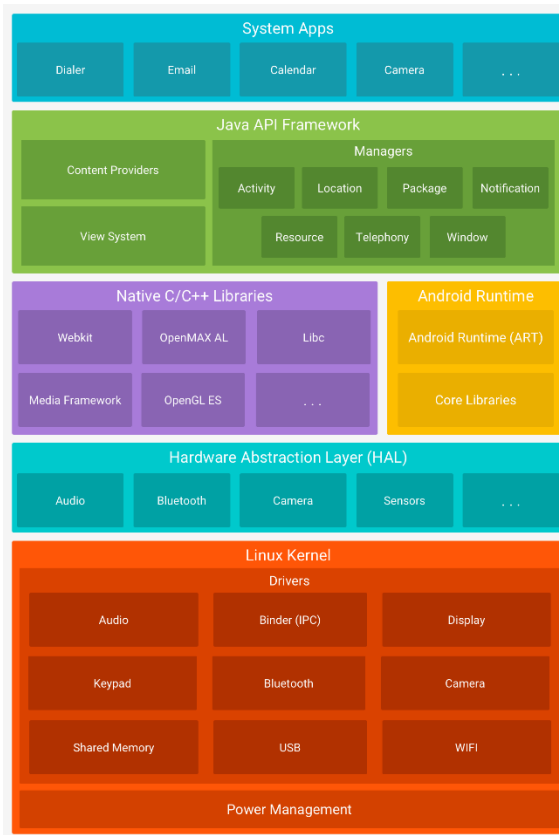
Brojne su novosti koje donosi API 24. U svrhu kratkog uvoda može se navesti da se od ove verzije mogu pokretati dvije aplikacije istovremeno (podjelom ekrana na dva dijela), da je u SDK uključen novi set klasa za rad s 3D grafikom (Vulkan API), optimizaciju potrošnje baterije, povećanu sigurnost, stabilnost, upravljanje memorijom i korisničko iskustvo.

### 3.2.1 Aplikacijski stog

Sukladno [18], Android softverski stog je kreiran na Linuxu te sadrži najvažnije komponente kako je prikazano na slici ispod (Slika 40). Na dnu stoga, u osnovi, nalazi se Linux jezgra (eng. Linux Kernel) koja je odgovorna za osnovni pristup hardverskim jedinicama i upravljanje napajanjem. Na drugoj razini se nalazi Androidov sloj za apstrakciju hardvera (eng. Hardware Abstraction Layer – HAL) koji se s jedne strane oslanja direktno na jezgru, a s druge strane kroz standardna sučelja omogućuje višim razinama korištenje spomenutih hardverskih komponenti.

Na trećoj razini se nalaze nativne Linux biblioteke pisane u C/C++ programskom jeziku koje se također mogu koristiti s viših slojeva, to jest iz sistemskih i korisničkih aplikacija. S druge strane se nalazi Android Runtime (ART) kao temeljni stroj za izvršavanje aplikacija koji izvršava svaku aplikaciju u vlastitom procesu s vlastitom ART instancom. ART je od verzije API 21 u potpunosti zamijenio stari

virtualni stroj (Dalvik) te novim mogućnostima kao što su kompiliranje u naprijed (eng. Ahead of time compilation) omogućio znatno brže instaliranje, pokretanje i izvršavanje aplikacije.



Slika 40. Android aplikacijski stog

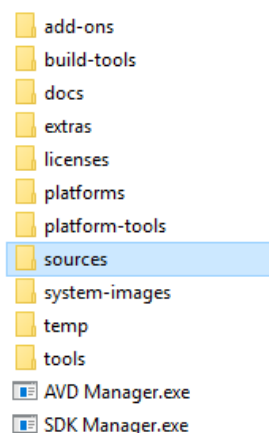
U konačnici se nalazi Java API razvojni okvir koji čini dostupnim sve funkcionalnosti Android OS-a uključujući upravitelje (eng. Managers), pružatelje sadržaja (eng. Content providers) i elemente korisničkog sučelja (eng. View System) za izgradnju sistemskih i korisničkih aplikacija. Povrh svega, korisničke aplikacije mogu pozivati i koristiti ugrađene sistemske aplikacije za ostvarenje željenih funkcionalnosti.

### 3.2.2 Struktura SDK-a

Osnovna struktura seta za razvoj aplikacija za Android (Android SDK) prikazana je na slici pored (Slika 41). SDK sadrži moguća proširenja, alate za kompiliranje, dokumentaciju, dodatne resurse, podatke o licencama, podatke o podržanim platformama, alate specifične za platforme, izvorne kôdove, slike sustava, ostale alate te dva vrlo važna upravitelja: upravitelj SDK-a i upravitelj virtualnih uređaja.

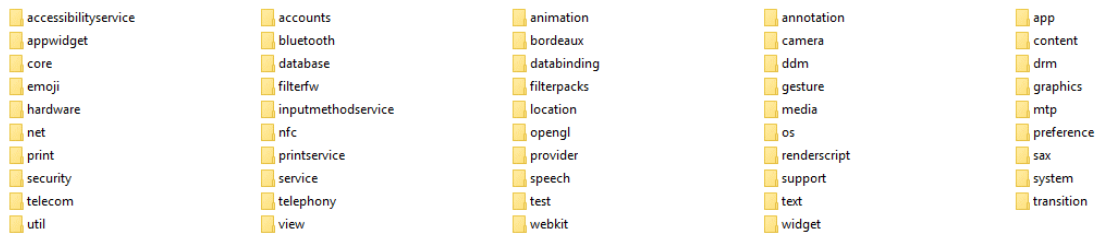
Pomoću SDK upravitelja korisnik može dodavati, brisati ili ažurirati različite verzije SDK-a ili samo elemente iz pojedinih verzija, dok pomoću AVD upravitelja korisnik može pokretati virtualne uređaje s različitim verzijama Androida, ovisno o tome što sve ima instalirano.

Jedan od alata, vrijedan posebne pozornosti, koji se nalazi u mapi *platform-tools* naziva se ADB. Riječ je o pozadinskom alatu kojeg pokreće Android Studio, a koji je odgovoran za omogućavanje komunikacije između Android Studija i virtualnog ili fizičkog uređaja na kojem se aplikacije mogu izvoditi i testirati.



Slika 41. Android SDK struktura

Konačno, od posebnog značaja za ovaj pregled je sadržaj mape *sources* koja sadrži sve osnovne (nativne) pakete koji se mogu koristiti pri razvoju aplikacija za Android. Popis paketa, bez detaljnog prikaza klasa unutar njih prikazan je na slici ispod (Slika 42).

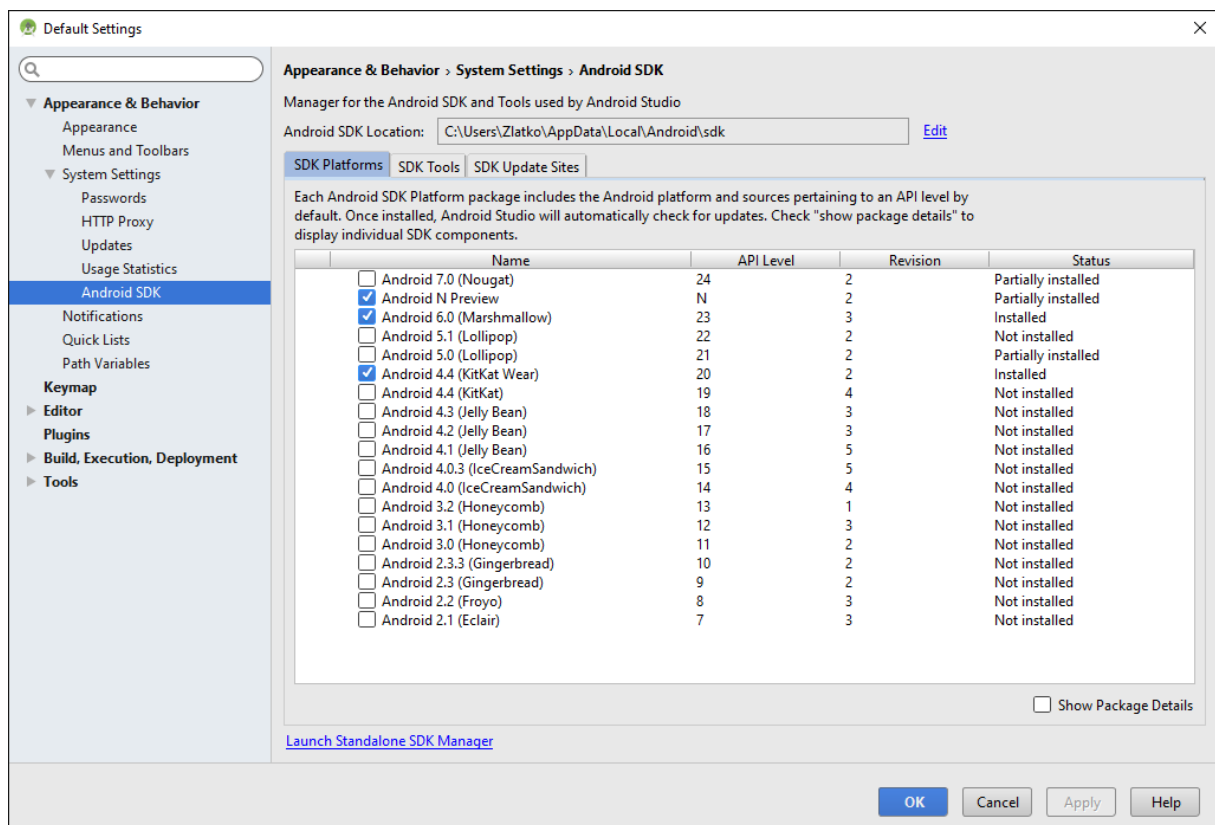


Slika 42. Dio osnovnih paketa dostupnih u Android SDK-u

Paketi koji se koriste vrlo često i neizostavni su u razvoju svake aplikacije su *util*, *view*, *support*, *text*, *widget*, *app*, *content*, *preference* i *system*. Naravno, ovaj popis paketa nije konačan, jer se u Setu također nalaze i *Java paketi*, *org paketi* i drugi.

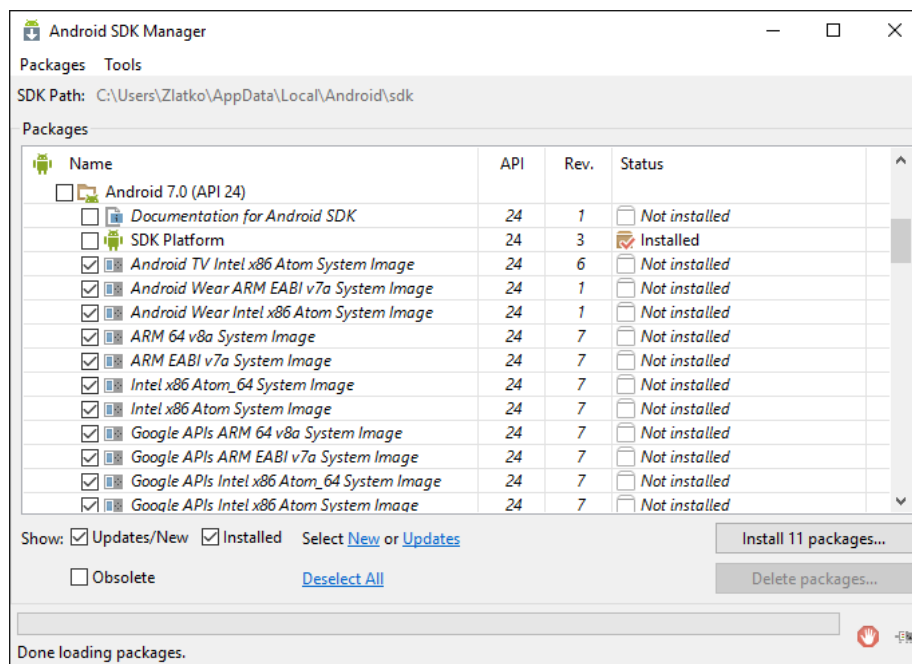
### 3.2.3 Instaliranje Android API-a

U svrhu dodavanja, promjene ili brisanja instaliranih SDK komponenti, potrebno je pokrenuti SDK manager. Isti se može pokrenuti kroz Android Studio (*Tools -> Android -> SDK Manager*), ili kao zaseban program iz mape kako je prethodno navedeno.



Slika 43. Android Studio SDK upravitelj

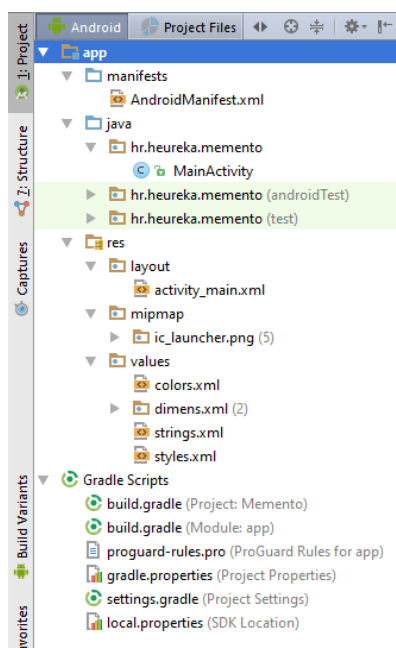
U svrhu jednostavnijeg i boljeg prikaza verzija predlažem korištenje samostalnog SDK upravitelja koji se može pokrenuti klikom na link u donjem lijevom uglu ekrana prikazano na slici iznad (Slika 43).



Slika 44. Android SDK upravitelj

Za potrebe vježbi prihvatite ažuriranje bilo kojeg od postojećih instaliranih paketa, te instalirajte ili dodajte posljednje verzije Android SDK Tools, SDK Platform Tools, SDK Build-tools iz **mape Tools**, Documentation for Android SDK, SDK Platform, Samples for SDK, ARM EABI v7a System Image iz **mape Android 7** (API 24) te Android Support Library, Google Play Services i Google USB Driver iz **mape Extras**. SDK manager je prikazan na slici iznad. Provjerite i ostale dostupne komponente te instalirajte po potrebi.

### 3.3 Programska logika Android aplikacije



Slika 45. Struktura android aplikacije

Razvoj aplikacija za Android je uvelike vođen zahtjevom za strogim odvajanjem korisničkog sučelja od programske logike. Taj zahtjev je toliko striktan da se u fazi razvoja programskog proizvoda koriste različiti jezici za opis dvaju spomenutih aplikativnih slojeva: Java za opis programske logike i XML za opis i dizajn sučelja.

Koristeći panel *Project* i pogled *Android* (Slika 45) koji služe za prikaz strukture projektnih dokumenata, promatranih kao elemente Android aplikacije, možemo vidjeti da su sve Java klase smještene u `java` čvor projektnog stabla, a svi opisi resursa su smješteni u `res` čvor projektnog stabla.

Osim navedenih u *manifest* čvoru je smješten osnovni aplikacijski proglas (eng. Manifest) koji definira sve zahtjeve aplikacije, a u potpuno odvojenoj cjelini pod nazivom *Gradle Scripts* su smještene sve konfiguracijske skripte potrebne za Gradle – automatski proces koji u realnom vremenu analizira i po potrebi kompilira programski kôd.

Detaljnijom analizom sadržaja `java` čvora, primijetit ćemo da

su sve programske klase smještene u osnovni aplikacijski paket, te dva pomoćna paketa koji služe za testiranje programskog proizvoda. Mi ćemo kroz ovaj priručnik koristiti isključivo osnovni aplikacijski paket `hr.heureka.memento` i u njemu ćemo kreirati vlastitu strukturu paketa. S druge strane, čvor `res` sadrži mapu `layout` u kojoj su smještene XML datoteke koje opisuju izgled aplikacije, mapu `mipmap` koja sadrži grafičke resurse te mapu `values` u kojoj su dokumenti koji sadrže vrijednosti u obliku aplikacijskih resursa.

### 3.3.1 Android aktivnosti

Aktivnost (eng. Activity) predstavlja aplikativnu dretvu koja se izvršava u aplikativnom kontekstu te kojoj je dana mogućnost da u vlastitom prozoru prikaže korisničko sučelje [19]. Aktivnost tipično predstavlja jednu korisničku funkcionalnost nad programskim proizvodom, te u interakciji s korisnikom i s operacijskim sustavom kontrolira elemente korisničkog sučelja.

```
package hr.heureka.memento;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

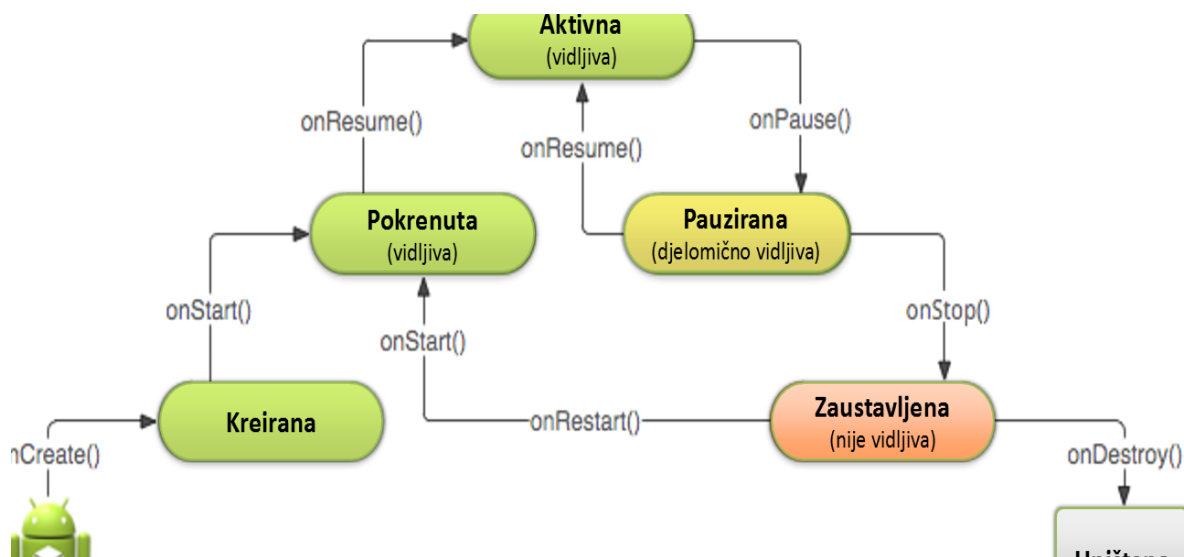
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

U prethodnom primjeru programskog kôda možemo vidjeti da klasa `MainActivity` nasljeđuje klasu `AppCompatActivity` (podklasa klase `Activity` iz Android razvojnog okruženja) koja omogućuje kompatibilnost sa svim verzijama operacijskog sustava koje smo pri kreiranju projekta označili kao ciljane verzije. Kako bi prevoditelj poznao sve klase koje se koriste u primjeru, potrebno je uvesti (eng. import) dvije nadklase iz paketa `support` i `os`.

Trenutno, unutar klase postoji implementacija samo jedne metode, koja je u ovom slučaju nadjačanje metode `onCreate` iz nadklase. Sukladno konvenciji pisanja naziva metoda u Androidu, svaka metoda koja odgovara na neki događaj ima prefiks *na* (eng. *on*). Tako metoda `onCreate` odgovara na događaj kreiranja aktivnosti, te se izvršava odmah nakon što je aktivnost kreirana.

Gornji kôd u načelu govori: „Kada kreiraš aktivnost dodjeli joj dizajn grafičkog sučelja iz `R.layout.activity_main`“. Kroz ovaj predmet promovirat ćemo nekoliko dobrih praksi programiranja. Uz navedenu ponovnu iskoristivost (engl. reusability), ovdje je prikazan primjer višeslojne arhitekture. Dizajn grafičkog sučelja (`/res/layout/activity_main.xml`), odvojen je od programskog kôda (`/src/hr.heureka.memento/MainActivity.java`).

Kako bi smo razumjeli ponašanje aktivnosti, sukladno [20] prikazat ćemo dijagram mogućih stanja aktivnosti i događaje koji iniciraju promjenu tih stanja (Slika 46).



Slika 46. Stanja aktivnosti u Android aplikaciji

Stoga, aktivnost može biti u jednom od sljedećih stanja:

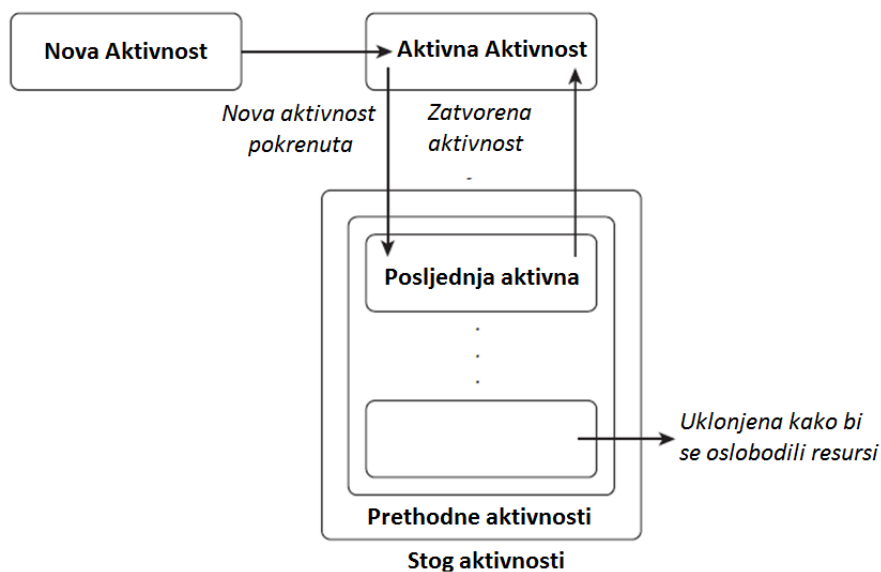
- ✓ **Aktivna** – kada je aktivnost na vrhu stoga, vidljiva, označena, čeka na korisnikovu interakciju ili prikazuje neke podatke. Aktivnosti niže u stogu su pauzirane ili zaustavljene.
- ✓ **Pauzirana** – kada vaša aktivnost jest vidljiva no nije u fokusu, ona se pauzira. Primjerice kada se otvori Da/Ne dijalog.
- ✓ **Zaustavljena** – Kada nije vidljiva, aktivnost u Androidu se zaustavlja. Ona ostaje u memoriji zajedno sa informacijama stanja te postaje kandidat za terminiranje. U ovom trenutku preporučljivo je pohraniti stanje. Kada je aktivnost zatvorena, ona postaje neaktivna.
- ✓ **Uništena** – nakon što je aktivnost zaustavljena, odnosno prije nego je pokrenuta, ona je neaktivna. Sve neaktivne aktivnosti su uklonjene iz aplikacijskog stoga.

Vidljivi životni ciklus aktivnosti je između `onStart` i `onStop` događaja. U tom periodu aktivnost je vidljiva korisniku.

- ✓ `onStop` - najčešće se koristi za zaustavljanje animacija, dretvi ili čitača senzora.
- ✓ `onStart` – nastavlja rad aktivnosti (ili `onRestart` koja ju pokreće iznova)

Aktivni životni ciklus aplikacije počinje sa `onResume` i završava sa `onPause` događajima. Aktivnost je aktivna kada je u fokusu odnosno korisnik je u interakciji s njom. Aktivno stanje aktivnosti završava s prikazom nove aktivnosti, gašenjem uređaja ili kada aktivnost izgubi fokus. Tipični primjer ove interakcije je pojava dijaloga. Odmah nakon `onPause` poziva se `onSaveInstanceState` koja daje mogućnost spremanja stanja i UI-a. Ti podaci prenose se preko `Bundle` klase. Često se stanje sprema kod `onPause` događaja jer poslije njega aktivnost može biti ugašena bez upozorenja.

Aplikacija za Android ne kontrolira životni tijek svojih procesa već to čini Android runtime, a stanje svake aktivnosti doprinosi u odluci određivanja prioriteta roditeljske aplikacije. Android runtime prema prioritetu odlučuje o prekidanju pojedinih aplikacija. Stanje svake aktivnosti određeno je stogom aktivnosti koji radi na LIFO (Posljednji ulazi prvi izlazi – eng. Last In First Out) principu. Kada se pokrene nova aktivnost, trenutni ekran ide na vrh stoga. Kada korisnik navigira korak natrag, grafičko sučelje trenutne aplikacije se gasi, aktivnost se gasi (ukoliko se ne radi o višedretvenim sustavima, procesima, prioritetnim aktivnostima itd.), izlazi iz stoga i prva slijedeća aktivnost dolazi na zaslon. Detalji ovakvog pristupa prikazani su na sljedećoj slici (Slika 47).



Slika 47. Stog aktivnosti

### 3.3.2 Resursi

Programski kôd prikazan na početku prethodnog poglavlja odmah pri kreiranju aktivnosti dohvaća identifikator resursa koji definira izgled korisničkog sučelja, te ga prosljeđuje metodi `setContentView` koja će ga prikazati korisniku. Riječ je, naravno, o sljedećoj liniji kôda:

```
setContentView(R.layout.activity_main);
```

Ovdje bi smo posebnu pozornost obratili na dvije stvari. Prvo, na način na koji su resursi (definirani pomoću XML-a) korišteni u programskoj logici koja se piše u Javi, a drugo na samu definiciju resursa u XML-u.

Ključnu ulogu u povezivanju resursa i programskog kôda ima **automatski generirana klasa R**, koja sadrži jedinstvene identifikatore svih resursa definiranih u sklopu projekta. Resursi su u toj klasi statički i organizirani su hijerarhijski sukladno mapama u koje su fizički smješteni. Svaki puta, kada korisnik promijeni resurse definirane u XML-u, ako je rezultirajući XML validan, Android Studio će automatski osvježiti klasu R te se odmah potom resursi mogu koristiti i u programskom kôdu.

Važno je naglasiti da pogreške u XML-u rezultiraju nemogućnošću kreiranja klase R, nakon čega će Android Studio javiti pogreške i u programskom kôdu gdje god se spomenuta statička klasa koristila. U tom slučaju nije potrebno ispravljati programski kôd, već pronaći i ukloniti pogreške koje onemogućuju kreiranje klase R.

Za kraj kratkog osvrt na klasu R, treba imati na umu da su svi identifikatori kreirani unutar klase R cjelobrojnog tipa podataka (`int`) te da su imenovani identično kao i resurs koji je korisnik definirao. Stoga, promotrimo XML opis resursa `activity_main` koji se koristi u prethodnom primjeru.

```

<android.support.design.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
android:layout_width="match_parent"
android:layout_height="match_parent">

    <!-- prostor u kojem se nalazi alatna traka (Toolbar) -->
    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar">

        <!-- prostor u kojem se prikazuju stranice (Tabs) -->
        <android.support.design.widget.TabLayout
            android:id="@+id/tabs"
            android:layout_width="match_parent"
            android:layout_height="72dp"
            app:tabMode="fixed"
            app:tabGravity="fill"/>
    </android.support.design.widget.AppBarLayout>

    <!-- prostor u kojem se prikazuju sadržaji pojedinih stranica -->
    <android.support.v4.view.ViewPager
        android:id="@+id/viewpager"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior" />
</android.support.design.widget.CoordinatorLayout>

```

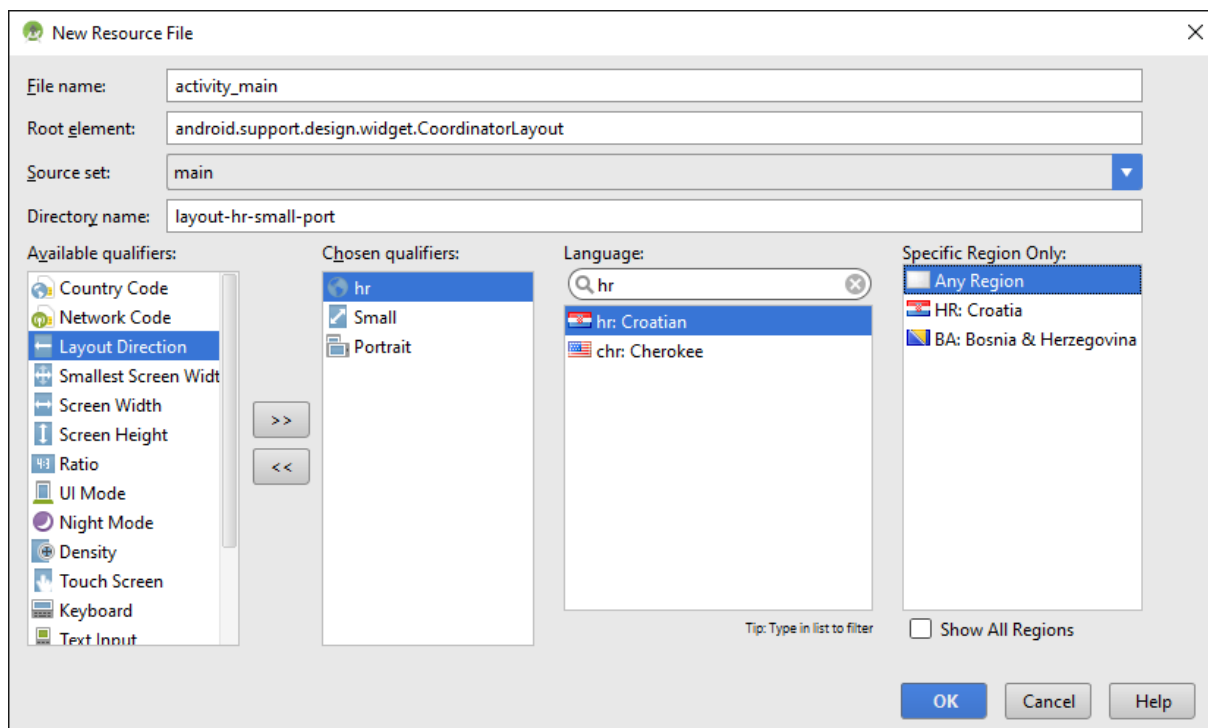
Budući da se dokument naziva `activity_main.xml` i nalazi se u mapi `layout`, klasa `R` sadrži identifikator toga resursa imenovan `R.layout.activity_main`. Međutim, `R` sadrži i sve druge resurse eksplicitno imenovane u ovom dokumentu. To su, primijetiti ćete, objekt `tabs` klase `TabLayout` i objekt `viewpager` klase `ViewPager`. Oba ova resursa imenovana su definiranjem `id` atributa dodjeljujući mu vrijednost `"@+id/naziv_resursa"`.

Stoga, u programskom kôdu, ove identifikatore možemo pronaći koristeći `R.id.naziv_resursa`.

### 3.3.3 Kvalifikatori resursa

Još jedna specifičnost razvoja za Android jest mogućnost korištenja *kvalifikatora resursa*. Kvalifikator je dodatna oznaka pridružena resursu koja taj resurs kvalificira za korištenje u specifičnom kontekstu na koji se taj kvalifikator odnosi. Drugim riječima, možemo kreirati dva resursa koji opisuju izgled `activity_main` pogleda. Međutim, jedan od njih možemo kvalificirati da bude korišten kada je uređaj u uspravnom položaju (eng. `Portrait`), a drugi da bude korišten kada je uređaj u horizontalnom položaju (eng. `Landscape`). Kako je prikazano na slici ispod (Slika 48), moguća je i kombinacija različitih kvalifikatora. Tako će iz primjera ispod, resurs koji se upravo kreira, biti korišten samo kada se aplikacija pokrene na uređaju s hrvatskim jezikom (`hr`), malih dimenzija ekrana (`small`) u uspravnom položaju (`portrait`).





Slika 48. Kvalificiranje resursa

Ovako kreirani resurs `activity_main` bit će smješten u mapu `layout_hr_small_port`. Međutim, prilikom pristupanja resursu iz programskog kôda, nije potrebno navoditi kvalifikatore. Dovoljno je pristupiti resursu `R.layout.activity_main`, a operacijski sustav će sam potražiti onu verziju resursa koja najbolje odgovara kontekstu u kojem se aplikacija nalazi. U najgorem slučaju, bit će korišten osnovni (eng. Default) resurs.

### 3.3.4 Android Manifest

Važan konstrukt svake aplikacije za Android je dokument proglašava zapisan u `AndroidManifest.xml` datoteci.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="hr.heureka.memento">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/MyMaterialTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Proglas je u novijim verzijama rasterećen elemenata koji su se odnosili na proces kompiliranja programskog proizvoda, te sadašnja verzija sadrži samo aplikativno bitne oznake. Tako se trenutno u proglasu nalazi naziv korijenskog paketa, te opis aplikacije koji sadrži aplikativne atribute kao što su ikona, naslov aplikacije, tema i slično. Također, u proglasu moraju biti pobrojani i svi zahtjevi aplikacije prema sustavu ili drugim aplikacijama. Tako ova aplikacija trenutno ne zahtijeva nikakva posebna dopuštenja od strane korisnika (kao npr. za pristup Internetu, osobnim podacima, porukama, kontaktima i slično).

Posebnu cjelinu u proglasu, unutar aplikacijske oznake (eng. Tag) čine oznake koje opisuju aplikacijske aktivnosti. Trenutno je prijavljeno korištenje samo jedne aktivnosti (`.MainActivity`), kojoj je korištenjem filtera namjera (eng. Intent filtera) na akciju zahtjeva sustava za pokretanjem glavne (eng. Main) aktivnosti i pokretanjem same aplikacije definirano da bude glava aplikacijska aktivnost, to jest aktivnost od koje će se aplikacija početi izvršavati.

Pogledajte sadržaj `AndroidManifest.xml` dokumenta kako bi ste pronašli informacije koje sadržava. Detaljno o proglasu aplikacija za Android možete pročitati u službenoj dokumentaciji [21].

### 3.3.5 Gradle

Gradle predstavlja dodatni alat (eng. Plugin) razvijen od treće strane (Gradle Inc.) koji se pri razvoju aplikacija za Android koristi za automatiziranje procesa izgradnje (eng. Build) programskog proizvoda (eng. Build automation) [22], [23]. Gradle u osnovi predstavlja poseban sustav koji u suradnji s Android studiom provodi zadane operacije nad programskim kôdom te automatski kreira izlazni dokument spreman za pokretanje na ART-u.

Gradle se također koristi u svrhu upravljanja paketima uključenim u programski proizvod to jest paketima o kojima programski proizvod ovisi (eng. Dependencies), ali i za kreiranje vlastite programibilne/promjenjive logike izgradnje programskog proizvoda [24], [25].

Gradle skripte se mogu definirati na razini cijelog projekta, ali i na razini svakog pojedinog aplikacijskog modula. Primjer ispod prikazuje sadržaj Gradle skripte za glavni aplikativni modul *app*.

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 24
    buildToolsVersion "24.0.0"

    defaultConfig {
        applicationId "hr.heureka.memento"
        minSdkVersion 17
        targetSdkVersion 24
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
}
```

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:24.0.0'
    // opcije dizajna
    compile 'com.android.support:design:24.0.0'
}
```

Iz priloženog dokumenta se može zaključiti da Gradle ovaj modul promatra kao aplikaciju za Android primjenom odgovarajućeg dodatka (eng. Plugin). Potom, slijedi popis atributa Android aplikacije, uključujući verzije Android SDK-a, konfiguraciju aplikacije i tip izgradnje aplikacije. U konačnici Gradle sadrži popis dodatnih modula koji uključuje sve `jar` datoteke u mapi `libs`, ali i dvije biblioteke eksplicitno navedene pomoću `compile` naredbe. Posljednja biblioteka `junit` se koristi samo kod kreiranja testne aplikacije.

Osim prikazanog dokumenta za Gradle, kroz panel *Project* i pogled *Android*, u grani *Gradle Scripts* možete provjeriti i analizirati i ostale Gradle skripte i postavke.

### 3.3.6 Namjera (Intent)

Namjera (eng. Intent) predstavlja poseban mehanizam prosljeđivanja poruka koji se može koristiti unutar i između aplikacija. *Namjera* deklarira namjeru da će aktivnost ili servis izvršiti neku radnju, odnosno za eksplicitno pokretanje aktivnosti ili servisa. *Namjerom* se može pristupiti interakciji s bilo kojom komponentom aplikacija instaliranih na Android uređaju. Time Android uređaj postaje platforma međusobno povezanih, neovisnih komponenti koje zajednički čine povezani sustav. Ako želite sami kreirati aktivnost koja može oslušivati namjere drugih aplikacija potrebno ju je proglasiti *slušateljem namjera* (eng. Intent Receiver). Međutim, ta funkcionalnost izlazi iz okvira ovog priručnika.

Za primjer korištenja klase `Intent`, prikazat ćemo programski kôd za pokretanje druge aktivnosti.

```
public void sendMessage(View view) {
    Intent intent = new Intent(this, NewActivity.class);
    EditText editText = (EditText) findViewById(R.id.edit_message);
    String message = editText.getText().toString();
    intent.putExtra("PORUKA", message);
    startActivity(intent);
}
```

Dakle, kreirali smo objekt namjere – `Intent`, u koji smo prosljedili podatke o trenutnom kontekstu (`this`) o aktivnosti koju želimo pokrenuti (`NewActivity.class`), te dodatno i poruku označenu ključnom riječi „PORUKA“ (`message`). Operacijski sustav će izvršiti namjeru čim se za to steknu uvjeti. U većinu slučajeva, uvjeti su ispunjeni odmah, međutim u izvanrednim situacijama operacijski sustav može privremeno ili potpuno odgoditi izvršenje namjere, na primjer ako se u istom trenutku dogodi telefonski poziv koji ima viši prioritet izvršenja.

### 3.4 Alati Android Studija

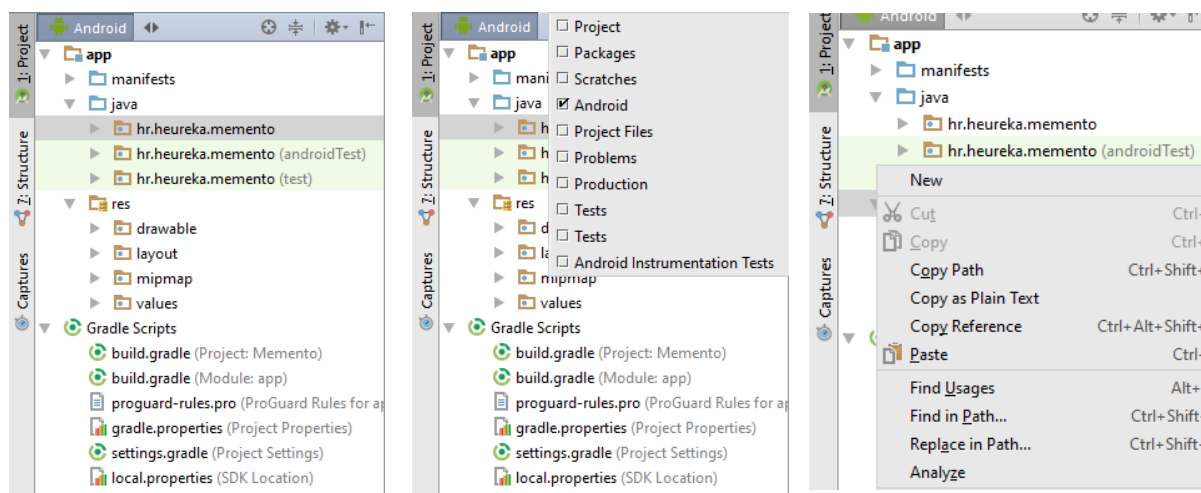
U perspektivi ovoga priručnika, te provedbe praktičnog dijela nastave razvoja mobilne aplikacije, sve alate u Android Studiju možemo podijeliti u sljedeće skupine:

- ✓ alati za razvoj
- ✓ alati za otkrivanje pogrešaka
- ✓ ostali važni alati

#### 3.4.1 Alati za razvoj

U fazi razvoja programskog proizvoda koristimo alate za pisanje programskog kôda u Javi, pregled i izmjenu strukture projekta i definiranje projektnih resursa.

Panel *Project* sa različitim pogledima koristimo u svrhu pregleda i izmjene strukture projekta, dodavanje modula, dokumenta i resursa. Rad s ovim alatom je intuitivan i svodi se na korištenje kontekstualnih izbornika (desni klik miša) otvorenih na odgovarajućim čvorovima prikazanim u strukturi.



Slika 49. Različiti pogledi Project panela

Slika 49 prikazuje alat za prikaz strukture projekta u Android pogledu (lijevo), opciju promjene pogleda (u sredini), te primjer kontekstualnog izbornika na čvoru `res` koji omogućuje naredbom `New` kreiranje novih izbornika (desno).

Alat za pisanje programskog kôda (Java editor) prikazan je u središnjem (radnom) panelu. Taj alat upotpunjuje pisanje programskog kôda s nekoliko važnih mogućnosti. Prva mogućnost je označavanje sintakse (eng. Syntax Highlighting) kojom alat prepoznaje ključne riječi programskog jezika Java, uključene klase, korištene varijable i slično. Također, alat prepoznajući programske konstrukte automatski formatira programski kôd i čini ga čitljivijim. Druga važna mogućnost ovog alata je *savjetnik* (eng. Intellisense). Savjetnik prikazuje razvojnom inženjeru mogućnosti korištenja programskih konstrukta u ovisnosti o kontekstu, ili bolje rečeno, o liniji kôda koja se u danom trenutku kreira. Primjer korištenja alata je prikazan na slici ispod (Slika 50).

```

MainActivity.java x
private void setupTabIcons() {
    tabLayout.getTabAt(0).setIcon(imageResIds[0]);
    tabLayout.getTabAt(1).setIcon(imageResIds[1]);
    tabLayout.getTabAt(2).setIcon(imageResIds[2]);
}

private void setupViewPager(final ViewPager viewPager) {
    final MainPagerAdapter adapter = new MainPagerAdapter(getSupportFragmentManager(), this);
    adapter.addFragment(new ActiveTasksFragment(), "Pending");
    adapter.addFragment(new CompleteTasksFragment(), "Complete");
    adapter.addFragment(new
viewPager.setAdapter(ada

// osvježi nakon odabira
viewPager.addOnPageChang
    @Override
    public void onPageSc

    @Override
    public void onPageSe
        if(adapter.getIt
            DataFragment
            selectedFragment.

    @Override
    public void onPageScrollStateChanged(int state) {}
};
}
}

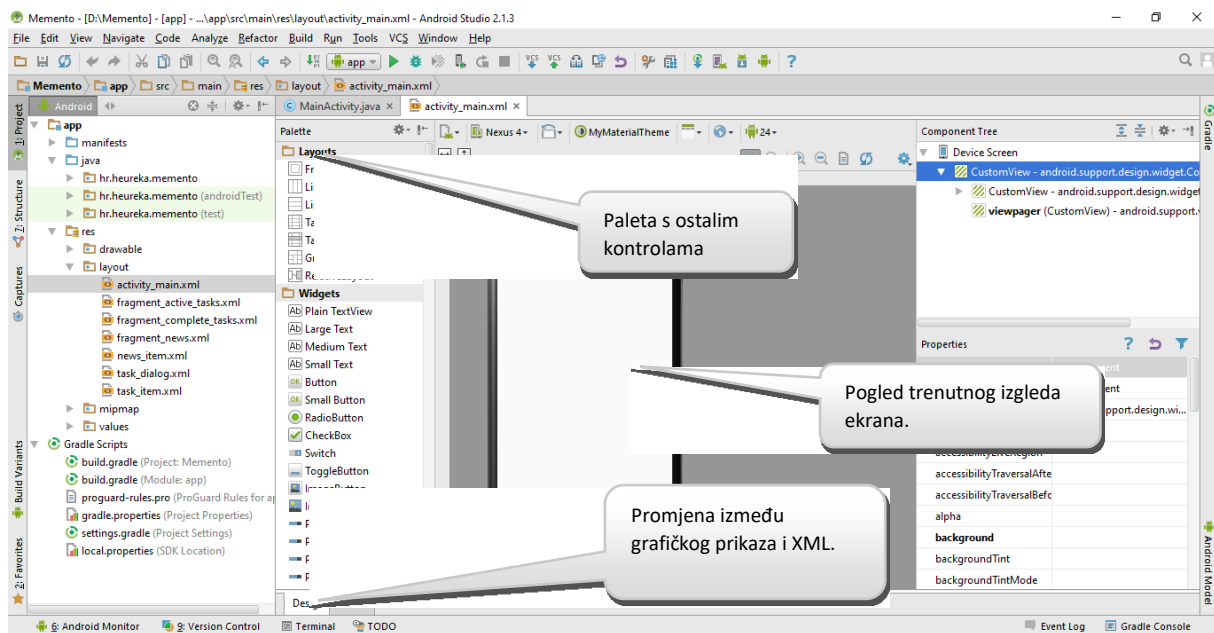
```

The screenshot shows the IDE interface with a context menu open over the `reloadData()` method call in the `setupViewPager` method. The menu items include `reloadData()` (void), `equals(Object obj)` (boolean), `hashCode()` (int), `toString()` (String), `getClass()` (Class<? extends DataFragment>), `notify()` (void), `notifyAll()` (void), `wait()` (void), `wait(long l, int i)` (void), and `wait(long millis)` (void). The `selectedFragment` property is highlighted in yellow.

Slika 50. Alat za pisanje programskog kôda

Posebno je važno spomenuti da alat za pisanje programskog kôda ima dosta dodatnih mogućnosti kojima se može pristupiti koristeći kontekstualni izbornik ili naredbe *Code*, *Analyze*, *Refactor* u glavnom aplikacijskom izborniku.

Drugi modus alata za razvoj odnosi se na skup alata koji omogućuju definiranje resursa. Pri tome skoro za svaki resurs postoji specifičan alat koji kroz grafičko sučelje omogućuje definiranje resursa. Zajednička karakteristika svih ovih alata je da imaju izlaz u obliku XML dokumenta koji opisuje svaki pojedini resurs.



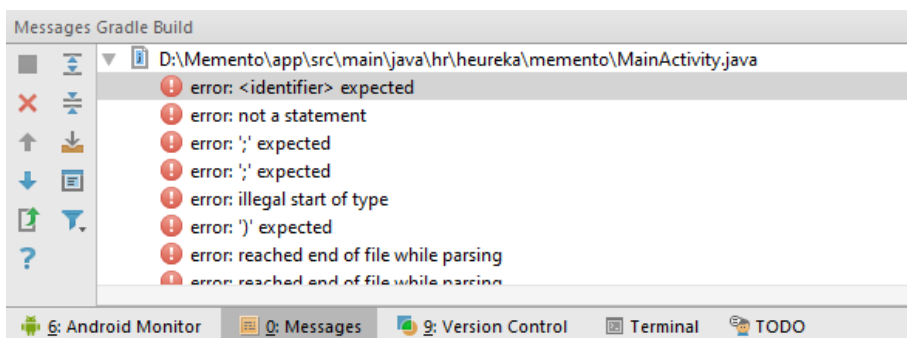
Slika 51. Alat za dizajn ekrana

Opis funkcionalnosti svih ovih alata nije tema ovog priručnika, ali budući da je rad u tim alatima intuitivan, već nakon nekoliko kreiranih primjera učenik i/ili inženjer će steći dovoljno iskustva kako bi mogao koristiti njihove mogućnosti.

### 3.4.2 Alati za otkrivanje pogrešaka

Pogreške se pri razvoju aplikacija za Android otklanjaju još tijekom razvoja, pri čemu se koriste upravo spomenuti alati za prikaz strukture (u kojem su podcrtani paketi, to jest klase i resursi) koji imaju sintaktičke pogreške.

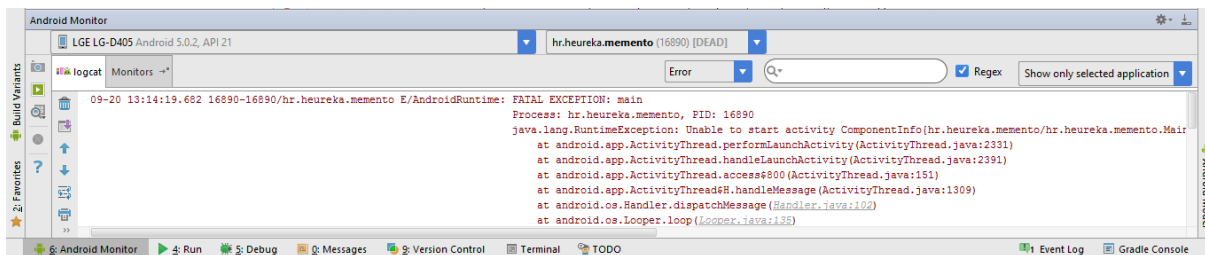
Posebna pomoć je alat *Messages* koji eksplicitno navodi uzroke pogrešaka i prijedloge rješenja istih (Slika 52). Iako su ove poruke u velikoj većini slučajeva korisne, ponekad mogu biti i pogrešne te mogu usmjeriti razvojne inženjere u pogrešnom smjeru.



Slika 52. Panel poruka

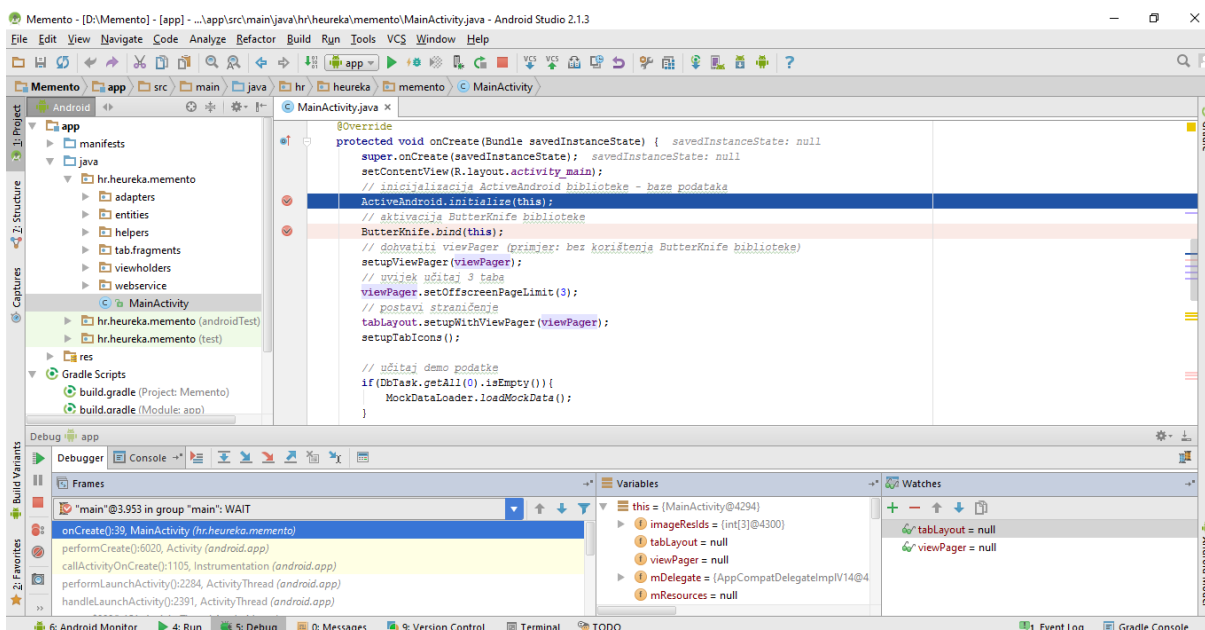
Spomenuti alat omogućuje i automatsko fokusiranje one linije kôda u kojoj je prepoznat problem jednostavnim dvostrukim klikom miša na odgovarajuću poruku. Za kraj, u ovom alatu se prikazuju i druge relevantne poruke, a ne samo poruke o pogreškama.

Posljednji set alata koji treba spomenuti u svrhu otkrivanja pogrešaka prilikom izvršavanja programskog proizvoda su alati *Android Monitor* i *Debug*. Pomoću *Android Monitor* alata može se vidjeti status izvršavanja aplikacije na uređaju te log poruke same aplikacije, sustava ili poruke o pogrešci kako je to prikazano na slici ispod (Slika 53).



Slika 53. Android Monitor alat

Debug alat surađuje sa alatom za pisanje programskog kôda pri čemu se izvršavanje aplikacije može pauzirati, izvršavati liniju-po-liniju, te uz pomoć pomoćnih panela dobiti uvid u stanje memorije, instancirane objekte i vrijednosti njihovih privatnih i javnih varijabli.

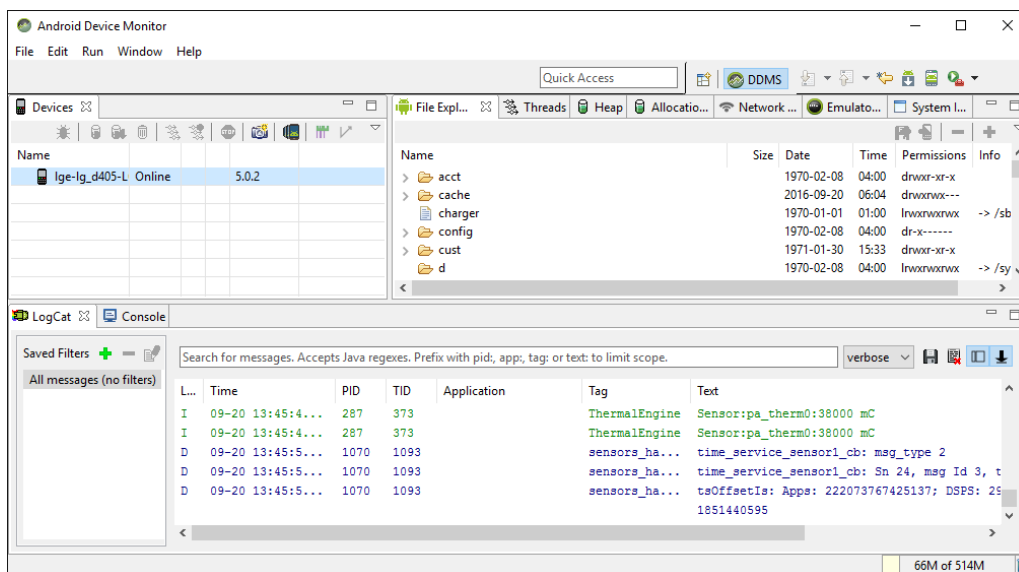


Slika 54. Alat Debug i pomoćni alati

Slika 54 prikazuje mogućnosti uvida u stanje memorije i pronalaska pogrešaka u realnom vremenu. U donjem dijelu slike je alat Debug sa stogom naredbi (Frames), lokalnim varijablama (Variables), te vlastitim definiranim pogledima na memoriju (Watches). Razvojni inženjer dodavanjem *točka prekida izvršenja kôda* (eng. Breakpoint) može označiti gdje želi zaustaviti izvršenje kôda, te pomoću kontekstualnog izbornika dodati *pogled na objekt ili varijablu* (eng. Watch). Točka prekida na slici iznad je kreirana na liniji kôda `ActiveAndroid.initialize(this);`

### 3.4.3 Ostali važni alati

Ostali važni alati Android Studija odnose se na infrastrukturne alate koji omogućuju ažuriranje samog Studija, upravljanje setom klasa za razvoj aplikacija za Android (SDK Manager), alatom za upravljanje virtualnim i povezanim uređajima (AVD Manager) te alat za upravljanje trenutno spojenim uređajem (Android Device monitor).



Slika 55. Android Device Monitor

Android Device Monitor (Slika 55) pokrećemo naredbom Tools -> Android -> Android Device Monitor. Pomoću ovog alata se može pristupiti dokumentima i podacima na uređaju, napraviti sliku ekrana s uređaja i slično.



### 3.5 Pitanja za provjeru znanja

1. Koja je razlika između integriranih razvojnih okruženja i generatora aplikacija?
2. U kojim slučajevima je prikladno koristiti generatora aplikacija?
3. Od kojih se sve alata sastoji Android studio?
4. Kreirajte različite vrste projekata za Android i proučite projektne materijale koji su automatski kreirani. Pokušajte prepoznati ugrađene elemente (klase, resurse, biblioteke) i njihove međusobne veze.
5. Kako imenujemo aktivnosti (i ostale klase), a kako resurse?
6. Pojasnite slojeve aplikacijskog stoga Android operacijskog sustava.
7. Čemu služi set za razvoj softvera (SDK) i koje su osobine Android SDK-a?
8. Koji su osnovni elementi aplikacije za Android? Pojasnite proglas, aktivnosti, resurse i Gradle skripte.
9. Što je *aktivnost* Android aplikacije? Koja sve stanja aktivnost može imati i koji se događaji okidaju pri prijelazu stanja?
10. Kako funkcionira stog aktivnosti? Kada se aktivnost stavlja, a kada skida sa stoga?
11. Čemu služe kvalifikatori resursa? Pri definiranju novog resursa analizirajte koji sve tipovi kvalifikatora postoje.
12. Čemu služi proglas (manifest) aplikacije za Android?
13. Kako funkcionira sustav izgradnje (eng. build) aplikacija za Android? Čemu služi Gradle?
14. Analizirajte alate Android Studija. Čemu služi alat Android monitor?
15. Koje alate koristimo za pisanje programskog kôda, a koje za otkrivanje pogrešaka?
16. Pojasnite kako se koristi *Watch* panel pri otkrivanju pogrešaka.
17. Koji su ostali važni alati integrirani u Android Studio?

### 3.6 Resursi za samostalan rad

- ✓ **Gargenta Marko.** Naučite Android. O'Reilly/IT Expert; 2011.
- ✓ **Gimson Matthew.** Android programming: complete introduction for beginners : step by step guide how to create your own Android app easy!, CreateSpace Independent Publishing Platform 2015.
- ✓ **Beer Paula, Simmons Carl.** Hello app inventor!: Android programming for kids and the rest of us. Shelter Island, NY: Manning Publications Co; 2015.
- ✓ **Gerber Adam, Craig Clifton.** Learn Android Studio: build Android apps quickly and effectively. Berkeley, CA: Apress; 2015.



## 4 RAD S PODACIMA

*Ovo poglavlje daje pregled i primjer rada s podacima kod razvoja mobilnih aplikacija za Android uređaje, prvenstveno mobilne telefone i tablete, a podijeljeno je u tri glavna dijela. Prvi dio poglavlja koncentrira se na pregled dostupnih mogućnosti za spremanje ne strukturiranih ili slabo strukturiranih podataka te rad sa datotekama. U drugom djelu fokus na rad sa mobilnom bazom podataka, a posebna pozornost dana je modeliranju podataka, te prijenosu modela podatka u programski kôd. Uz opis rada Android operacijskog sustava sa SQLite bazom podataka, detaljan primjer napravljen je korištenjem Active Android biblioteke koja značajno olakšava i ubrzava rad sa mobilnom bazom podataka. Treći dio ovog poglavlja odnosi se na rad s udaljenim podacima odnosno web servisima. Uz opis osnova koje su potrebne za razumijevanje načela rada i korištenja web servisa, primjer njihovog korištenja u razvoju Android mobilnih aplikacija prikazan je uz pomoć biblioteke Retrofit.*

### SADRŽAJ POGLAVLJA

Pohrana podataka .....	80
Zapis podataka u datoteke .....	85
Rad sa mobilnom bazom podataka .....	92
Rad s web servisima .....	106
Dodatni resursi .....	117

## 4.1 Pohrana podataka

*Angry Birds*, *Skype*, *Twitter*, samo su neke od mobilnih aplikacija koje su na Android uređajima na listi najinstaliranijih sa nešto više od sto milijuna jedinstvenih instalacija, dok su *YouTube* i *WhatsApp Messenger* primjer onih koje imaju više od milijardu jedinstvenih instalacija. Iako na prvi pogled aplikacije nemaju značajnijih karakteristika koje dijele, veže ih potreba za pohranom podataka. Danas, većina najkorištenijih mobilnih aplikacija na Android Play-u implementira neki oblik pohrane, prikaza ili obrade podataka, stoga je izuzetno važno razumjeti koje mogućnosti rada s podacima su dostupne razvojnim inženjerima za Android.

Kako navodi službena Android dokumentacija, većina ne-trivijalnih aplikacija ima potrebu pohrane korisničkih postavki, spremanja značajnije količine raznolikih datoteka ili korištenje neke vrste strukturiranog zapisa podataka. Osnovne mogućnosti pohrane podataka na Androidu su, a) ključ-vrijednost skupovi, b) datoteke i c) baza podataka.

*Ključ-vrijednost skupovi* (engl. *key-value pairs*) zapisuju se u obliku XML datoteke koja se sastoji od parova koji sadrže dvije karakteristike, ključ i vrijednost. Ključ je obično ime kojim se pristupa do neke vrijednosti. Primjer ovakvih skupova datoteka može se pronaći u `/res/values` svake Android mobilne aplikacije. Razmotrimo primjer `/res/values/dimens.xml` datoteke.

```
<resources>
  <!-- Default screen margins, per the Android Design guidelines. -->
  <dimen name="activity_horizontal_margin">16dp</dimen>
  <dimen name="activity_vertical_margin">16dp</dimen>
</resources>
```

U navedenom primjeru, *ključ* nazvan `name` sadržaja „`activity_horizontal_margin`“ dohvaća *vrijednost* `16dp`. Ključ-vrijednost skupovi obično se koristi za zapis manje količine podataka.

*Datoteke* se koriste za zapis većeg skupa podataka na datotečni sustav kojemu je pristup nalik onome korištenom kod klasičnih datotečnih sustava operacijskih sustava poput Linux-a, Windows-a, itd. Aplikacije koje koriste mogućnost rada s datotekama za vrijeme instalacije traže posebnu dozvolu kojom korisnik dozvoljava pristup datotečnom sustavu, bilo internom (na kojoj je instaliran Android operacijski sustav) ili eksternom (podatkovni dio memorijskog prostora, fizički uređaj poput micro-SD kartice ili simulirana micro-sd kartica).

Obzirom na pristup datotečnom prostoru, datoteke mogu biti *javne* ili *privatne*. Javne datoteke su one koje iako neka aplikacija kreira, omogućeno je drugim aplikacijama da ih koriste, a za vrijeme uklanjanja aplikacije koja ih je kreirala, one se ne brišu. Primjer takvih datoteka su fotografije ili datoteke preuzete s weba.

Privatne datoteke su one kojima je vlasnik isključivo aplikacija koja ih kreira i za vrijeme deinstalacije iste trebaju biti uklonjene. Android operacijski sustav svaku aplikaciju tretira kao korisnika, a kako Android pripada Linux obitelji operacijskih sustava, prema sustavu prava pristupa, druge aplikacije, odnosno korisnici, nemaju pristup privatnom direktoriju korisnika, odnosno aplikacije. No, ponekad, to je omogućeno ukoliko korisnik ima *root* prava koja omogućuju čitanje `/Data/Dana/` direktorija gdje se nalaze privatni podaci svake aplikacije. Po zadanim postavkama, uređaji su zaključani i ne daju tu mogućnost, no popularno nazvanim procesom „rootanja“ omogućuje korisniku da pristupi tim podacima. Čitatelju ostavljamo na razmišljanje koje su prednosti i koji su nedostaci za običnog korisnika, za stručnog korisnika i za malicioznog korisnika.

*Baza podataka* omogućuje strukturirani zapis, slično kao i kod privatnih datoteka, pristup bazi podataka u načelu je dozvoljen samo aplikaciji koja ju kreira i koristi. Takav zapisa podataka omogućuje formalnu specifikaciju entiteta, njihovi atributa i tipova, te složene odnose s drugim podacima. Kao zadani sustav za upravljanje bazom podataka Android koristi SQLite, no postoje i druga proširenja, primjerice BerkeleyDB, Couchbase Lite, Realm, itd.

#### 4.1.1 Mogućnosti pohrane podataka

Odabir načina perzistencija podataka, odnosno fizičke lokacije ovisi o vrsti mobilne aplikacije i potrebama korisnika. Obzirom na to, kod Androida postoje sljedeće mogućnosti zapisa podataka [26]:

- *Zajedničke postavke* (engl. *shared preferences*) – pohrana podataka pomoću klase `SharedPreferences` koja omogućuje pohranu ključ-vrijednost parova. Podržani tipovi podataka su `boolean`, `float`, `int`, `long` i `string`. Obično se koristi za pohranu postavki korisnika na razini aplikacije [27] [28].
- *Unutarnja memorija* – kako je ranije rečeno, datoteke pohranjene u internoj memoriji su privatne, i pohranjene na način da njima ne mogu pristupiti druge aplikacije, a direktno ni korisnik. Nakon deinstalacije aplikacije, datoteke s interne memorije se uklanjaju, a zapisane su na lokaciji `/Data/Data/paket/files`.
- *Vanjska memorija* – primarno se odnosi na memoriju čiju je primjenu moguće zabraniti operacijskom sustavu, odnosno memoriju koja nije nužno trajno prisutna, bila ona fizički ili softverski uklonjiva (engl. *unmount*). Datoteke zapisane na vanjsku memoriju su dostupne drugim aplikacijama i korisniku, a putem USB pristupa je moguća i njihova manipulacija. Za pristup tom mediju, mobilna aplikacija mora imati dozvolu.
- *SQLite database* – ranije spomenut strukturirani zapisa podataka.
- *Mrežno dostupna memorija* – kada je dozvoljen i omogućen pristup mreži, moguće je zapisivati i dohvaćati podatke na mrežno dostupnim servisima.

Naredna poglavlja demonstriraju pohranu podataka korištenjem ključ-vrijednost parova, datoteka i baze podataka. Kao demonstracijski primjer na kojima će biti pokazani navedeni koncepti, koristi se jednostavna aplikacija nazvana CookBook kroz koju su pokazani demonstracijski primjeri. CookBook je aplikacija koja omogućuje pohranu recepata i sastojaka istih.

#### 4.1.2 Ključ-vrijednost parovi

Raniji primjer XML-a koji prikazuje na koji način se zapisuju ključ-vrijednost parovi može se koristiti na dva načina. Prvi se odnosi na ručni (programskim kôdom) zapis i čitanje podataka, dok se drugi odnosi na polu-automatiziran pristup, pri čemu programer u XML datoteci definira koji podaci se zapisuju.

#### 4.1.3 Korištenje parova ključ-vrijednost putem programskog kôda

Klasa zajedničkih postavki, `SharedPreferences` omogućuje zapis i čitanje nekoliko prethodno definiranih tipova podataka. Tipičan primjer zapisa para ključ-vrijednosti definira se kao na primjeru:

```
SharedPreferences sharedPref = getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();

editor.putString("default_user_email", "ihorvat@cookbooker.demo");
editor.putInt("max_ingredients_per_page", 100);
```

```
editor.putBoolean("use_imperial_system", false);
editor.commit();
```

U prethodnom primjeru, najprije se putem `getPreferences` dohvaća trenutna instanca dijeljenih postavke koja se odnosi na kontekst trenutne aktivnosti. `MODE_PRIVATE` odnosi se na (zadani) način kreiranja datoteke, pri čemu je pristup istoj mogući samo putem aplikacije koja je kreira.

U sljedećoj liniji nad `SharedPreferences` objektom metodom `edit` dohvaća se sučelje koje omogućuje modifikaciju sadržaja tog objekta, odnosno modifikaciju povezane XML datoteke.

Slijedi zapis parova ključ-vrijednost u obliku teksta, cijelog broja i logičke vrijednosti, a na kraju se metodom `commit` zapisuju promjene.

Navigiranjem do aplikacije `CookBook` i izlistavanjem sadržaja direktorija rezultat je sljedeći:

```
1|root@vbox86p: /Data/Data/hr.heureka.cookbook # ls
cache
lib
root@vbox86p: /Data/Data/hr.heureka.cookbook #
```

No, nakon izvršavanja ranijeg kôda, pojaviti će se novi direktorij sa jednom datotekom i sadržajem prikazanom u sljedećem primjeru:

```
root@vbox86p: /Data/Data/hr.heureka.cookbook # ls
cache
lib
shared_prefs <-- NOVO
root@vbox86p: /Data/Data/hr.heureka.cookbook # cd shared_prefs
root@vbox86p: /Data/Data/hr.heureka.cookbook/shared_prefs # ls
MainActivity.xml
root@vbox86p: /Data/Data/hr.heureka.cookbook/shared_prefs # cat MainActivity.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="default_user_email">ihorvat@cookbooker.demo</string>
  <boolean name="use_imperial_system" value="false" />
  <int name="max_ingredients_per_page" value="100" />
</map>
root@vbox86p: /Data/Data/hr.heureka.cookbook/shared_prefs #
```

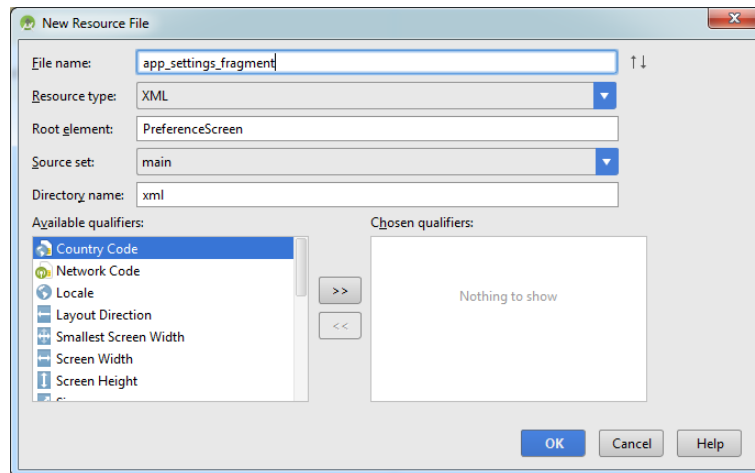
Datoteka je nazvana prema asociranoj aktivnosti a sadržaj je zapisan u XML formatu.

**ZADATAK:** *Samostalno promotrite zapis i usporedite ga s Java kôdom.*

#### 4.1.4 Korištenje parova ključ-vrijednost prema preddefiniranim pravilima

Korištenje parova ključ-vrijednost zadani je način zapisa podataka kod fragmenata sa preferencama mobilne aplikacije, `PreferenceFragment`. Za razliku od prethodnih dijeljenih preferenci, `PreferenceFragment` kreira XML datoteku sa zapisom preferenci korisnika na razini konteksta čitave mobilne aplikacije.

Kako bi se definirale vrijednosti koje će fragment sa preferencama mobilne aplikacije zapisivati, potrebno je dodati novu XML datoteku. Pritiskom na `/res` direktorij u strukturi projekta, odabrat desnim klikom opciju „New“, „Android resource file“, nakon čega će se pojaviti dijalog prikazan na slici (Slika 56).



Slika 56. Dodavanje nove XML datoteke za prikaz postavki aplikacije

Prikazani dijalog potrebno je popuniti prema tablici (Tablica 12).

Tablica 12. Postavke za dodavanje XML datoteke za postavke mobilne aplikacije.

File name	app_settings_fragment
Resource type	XML
Root element	PreferenceScreen
Source set	main
Directory name	xml

Nova datoteka prema zadanom nazivu kreira se u `/res/xml` direktoriju i potrebno je upisati sljedeći sadržaj:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
  <PreferenceCategory
    android:title="Main settings">

    <EditTextPreference
      android:key="default_user_email"
      android:title="Enter email"
      android:summary="The email of a default user"
      android:dialogTitle="Enter email" />

    <EditTextPreference
      android:key="max_ingredients_per_page"
      android:title="Page size"
      android:summary="Largest number of ingredients per page"
      android:dialogTitle="Enter page size" />
  </PreferenceCategory>
</PreferenceScreen>
```

```

<CheckBoxPreference
    android:defaultValue="false"
    android:key="use_imperial_system"
    android:title="Use imperial system"
    android:summary="The default is metric system" />

</PreferenceCategory>
</PreferenceScreen>

```

Kao što je vidljivo iz primjera, kreiran je pogled, odnosno ekran `PreferenceScreen` kojim će korisniku biti prikazane mogućnosti aplikacije, odnosno postavke može postaviti na željene vrijednosti. `PreferenceCategory` oznakom (*engl. tag*) određen je naziv kategorije. Iako je moguće imati više kategorija kako bi se korisnik lakše snalazio, u ovom primjeru definirana je samo jedna.

Svaki od elemenata ima tri ključna svojstva, ključ (`key`), naslov (`title`) i opis (`summary`). Na raspolaganju su sljedeći elementi:

- `CheckBoxPreference` – logičke vrijednosti, istina ili laž.
- `EditTextPreference` – tekstualne i brojčane vrijednosti. Prilikom unosa otvara se dialog prema definiranom naslov (`android:dialogTitle` svojstvo).
- `ListPreference` – omogućuje korisniku odabir jednog elementa iz liste definirane dodatnom XML datotekom, unutar `/res/values` direktorija. Prilikom unosa otvara se dialog, kao i za prethodnu oznaku.
- `MultiSelectListPreference` – jednako kao i prethodna oznaka, samo što omogućuje odabir više vrijednosti.
- `SwitchPreference` – jednako kao i `CheckBoxPreference` no drugačijeg grafičkog prikaza.

Kako bi se dodao navedeni tip fragmenta, potrebno je u projektu kreirati novu klasu, primjerice naziva `AppSettingsFragment` i proširiti je klasom `PreferenceFragment` te ga asociirati sa XML datotekom koja sadrži definiciju postavki (`addPreferenceFromResource`), kao što je prikazano u sljedećem kôdu:

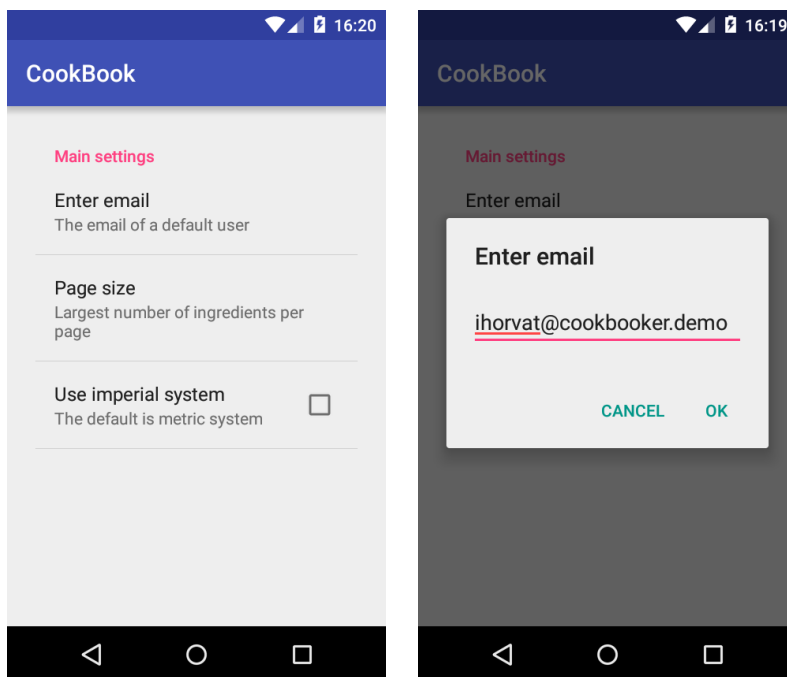
```

public class AppSettingsFragment extends PreferenceFragment {
    @Override
    public void onCreateView(View view, Bundle savedInstanceState) {
        super.onCreateView(view, savedInstanceState);
        addPreferencesFromResource(R.xml.app_settings_fragment);
    }
}

```

Rezultat navedenog kôda prikazan je na slici Slika 57.





*Slika 57. Fragment sa zadanim postavkama čije se vrijednosti zapisuju kao ključ-vrijednost parovi*

Također, nakon izvršavanja navedenog kôda, kreirati će se datoteka strukturiranog naziva prema sljedećem pravilu `korijenski.paket.nazivaplikacije_preferences.xml`. To moguće provjeriti putem `adb`<sup>13</sup> konzole, prikazanom u sljedećem primjeru.

```
root@vbox86p: /Data/Data/hr.heureka.cookbook/shared_prefs # ls
MainActivity.xml
hr.heureka.cookbook_preferences.xml

root@vbox86p: /Data/Data/hr.heureka.cookbook/shared_prefs # cat
hr.heureka.cookbook_preferences.xml

<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="default_user_email">ihorvat@cookbooker.demo</string>
  <boolean name="use_imperial_system" value="false" />
  <boolean name="checkbox_preference" value="true" />
</map>

root@vbox86p: /Data/Data/hr.heureka.cookbook/shared_prefs #
```

## 4.2 Zapis podataka u datoteke

Kod razvoja mobilnih aplikacija namijenjenih za Android zapis podataka u datoteke koristi se u slučaju kada je potrebno zapisati veću količinu podataka, primjerice slike, filmovi ili datoteke uredskih paketa (.docx, .xlsx, .odt, .pdf, itd.).

Svi Android uređaji imaju mogućnost zapisa datoteka na unutarnju ili vanjsku memoriju. Kod većine Android uređaja u današnje vrijeme oba tipa memorije odnose se na unutarnju memoriju koja je sastavni dio uređaja. No, prije nekoliko godina (a kod nekih uređaja i danas) vanjska memorija bila je

<sup>13</sup> ADB – Android debug bridge, program kojim se putem računala spaja na android uređaj, odnosno naredbeni redak operacijskog sustava [27].

prijenosni medij, poput micro SD kartica koja se po potrebi mogla izvaditi i mijenjati. Obje vrste memorije imaju specifične karakteristike prikazane u tablici (Tablica 13).

*Tablica 13. Usporedba unutarnje i vanjske memorije*

Unutarnja memorija	Vanjska memorija
Uvijek dostupna i nije uklonjiva.	Nije uvijek dostupna, moguće ju je fizički (USB ili micro SD kartica) ili softverski (unmount) ukloniti iz sustava.
Datoteke su dostupne samo aplikaciji koja im je vlasnik.	Datoteke su dostupne svim aplikacijama.
Deinstalacijom vezane aplikacije, uklanjaju se sve kreirane datoteke.	Deinstalacijom vezane aplikacije, kreirane datoteke se uklanjaju samo ako je eksplicitno naznačeno.

Sukladno preporuci službene dokumentacije za razvoj Android mobilnih aplikacija, unutarnja memorija koristi se u slučaju kada se želi osigurati da datotekama koje kreira aplikacija ne koristi nitko drugi osim nje. Vanjska memorija koristi se u slučaju kada nema takvih potreba za upravljanje pristupom datotekama ili u slučaju pohrane velikih datoteka. U načelu, vanjska memorija je značajno veća od unutarnje, a za njeno korištenje, mobilna aplikacija u `AndroidManifest.xml` datoteci mora od korisnika zatražiti posebne dozvole:

```
<!-- korištenje eksterne memorije za pisanje -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<!-- korištenje eksterne memorije za čitanje -->
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

#### 4.2.1 Primjer zapisivanja tekstualne datoteke na internu memoriju

Za zapis datoteka koristi se `java.io` biblioteka. Primjer koji slijedi prikazuje zapis datoteke naziva „CookBookFile“ sadržaja „This is a CookBook file!“ [26].

```
String filename = "CookBookFile";
String string = "This is a CookBook file! ";
FileOutputStream outputStream;

try {
    outputStream = openFileOutput(filename, Context.MODE_PRIVATE);
    outputStream.write(string.getBytes());
    outputStream.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

Klasa `FileOutputStream` koristi se za kreiranje datoteke i zapisivanje sadržaja u istu. Njena metoda `openFileOutput` kreira otvara ili kreira datoteku prema zadanom nazivu i prema zadanim pravima, u ovom slučaju `MODE_PRIVATE` što znači da će datoteci moći pristupati samo CookBook aplikacija.

Izvršavanjem navedenog kôda iz aplikacije adb konzolom moguće je izlistati sadržaj direktorija aplikacije CookBook:

```
root@vbox86p: /Data/Data/hr.heureka.cookbook # ls
cache
databases
lib
shared_prefs
root@vbox86p: /Data/Data/hr.heureka.cookbook # ls
cache
databases
files
lib
shared_prefs
root@vbox86p: /Data/Data/hr.heureka.cookbook # cd files
root@vbox86p: /Data/Data/hr.heureka.cookbook/files # ls
CookBookFile
root@vbox86p: /Data/Data/hr.heureka.cookbook/files # cat CookBookFile
This is a CookBook file!
root@vbox86p: /Data/Data/hr.heureka.cookbook/files #
```

Kreirao se novi direktorij naziva „files“ unutar kojeg je datoteka prema zadanom nazivu i sadržaju. Ako je nekome poznata točna putanja do CookBookFile datoteke, a nije korišten `MODE_PRIVATE` u tom slučaju pristup datoteci će biti moguć.

Važno je još ponoviti da će ovakav tip datoteke biti prisutan na Android uređaju sve dok je aplikacija CookBook instalirana. No, moguće je napraviti i datoteke koje su privremene te koje korisnik može ukloniti. Privremene datoteke zapisuju se pomoću `createTempFile` metode iz klase `File` sukladno sljedećem primjeru:

```
String content = "This is a CookBook cache file! ";
File file;
try {
    file = File.createTempFile("CookBookCache", null, getCacheDir());
    outputStream = new FileOutputStream(file);
    outputStream.write(content.getBytes());
    outputStream.close();
}
```

```

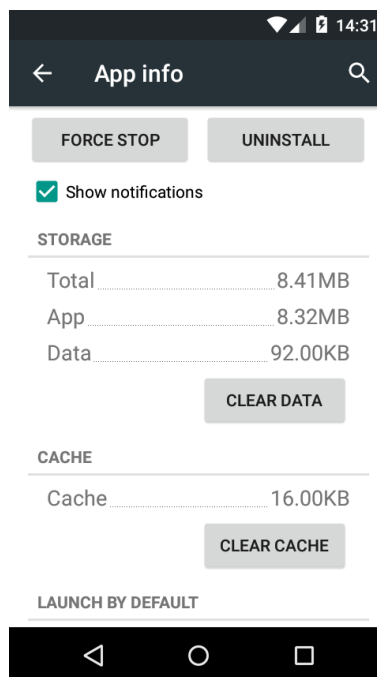
} catch (IOException e) {
    e.printStackTrace();
}
    
```

Izlistavanjem sadržaja direktorija CookBook mobilne aplikacije neposredno po izvođenju prethodnog kôda može se uočiti novi direktorij „cache“ koji sadrži privremenu datoteku prema danom imenu, no uz sufiks Unix vremenskog zapisa (engl. Unix timestamp):

```

root@vbox86p: /Data/Data/hr.heureka.cookbook # ls
cache
databases
files
lib
shared_prefs
root@vbox86p: /Data/Data/hr.heureka.cookbook # cd cache
root@vbox86p: /Data/Data/hr.heureka.cookbook/cache # ls
CookBookCache-1583310953.tmp
root@vbox86p: /Data/Data/hr.heureka.cookbook/cache # cat CookBookCache-
1583310953.tmp
This is a CookBook cache file!
root@vbox86p: /Data/Data/hr.heureka.cookbook/cache #
    
```

Privremene datoteke moguće je obrisati putem postavki Android uređaja, odnosno putem sustavskih postavki aplikacija, označiti željenu aplikaciju što čime se otvara ekran prikazan na slici Slika 58 i konačno pritiskom na tipku „Clear cache“.



Slika 58. Brisanje sadržaja direktorija s privremenim datotekama

#### 4.2.2 Čitanje sadržaja datoteka

Čitanje datoteke moguće je putem sljedećeg kôda:

```

try {
    InputStream inputStream = openFileInput("CookBookFile");

    InputStreamReader inputStreamReader = new InputStreamReader(inputStream);
    BufferedReader bufferedReader = new BufferedReader(inputStreamReader);
    
```

```

String line = "";
StringBuilder stringBuilder = new StringBuilder();

while ( (line = bufferedReader.readLine()) != null ) {
    stringBuilder.append(line);
}
inputStream.close();
System.out.println("Reading result: " + stringBuilder.toString());
} catch (FileNotFoundException e) {
    e.printStackTrace();
}
catch (IOException e) {
    e.printStackTrace();
}
}

```

Čitanje sadržaja datoteka vrši se pomoću klase `InputStreamReader`. Klasom `InputStream` otvara se datoteka, a zatim se čita redak po redak (`while` petlja u prethodnom primjeru). Sadržaj retka putem klase `StringBuilder` čita se u obliku teksta. Po završetku čitanja ulazni tok podataka (instanca `InputStream` klase) obavezno se zatvara, a u navedenom primjeru se pročitani sadržaj ispisuje u Logcat konzoli:

```

07-19 14:23:55.551 25216-25216/hr.heureka.cookbook I/System.out: Reading result:
This is a Cookbook file!

```

#### 4.2.3 Primjer zapisivanja podataka na vanjsku memoriju

Kako je moguće da vanjska memorija nije uvijek dostupna za korištenje, prije korištenja iste valjda provjeriti njenu prisutnost, što je vidljivo u sljedećem primjeru:

```

String state = Environment.getExternalStorageState();
if (Environment.MEDIA_MOUNTED.equals(state)) {
    try {
        File path =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS);
        File file = new File(path, "temp.txt");

        FileOutputStream outputStream = new FileOutputStream(file);
        outputStream.write("Hello world from temp! \n".getBytes());
        outputStream.flush();
        outputStream.close();

    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
    catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

Ono što je specifično kod zapisa datoteke u internu memoriju, uz samu provjeru dostupnosti memorije, je i lokacija pohrane koja se dobiva iz pobrojenja `Environment`. Sve lokacije pobrojane u tablici ispod (Tablica 14) su javno dostupne za korištenje.

*Tablica 14. Standardne lokacije za zapis datoteke na vanjskoj memoriji*

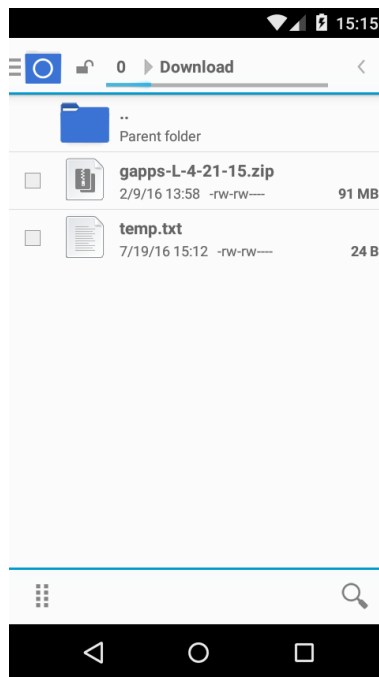
DIRECTORY_ALARMS	Direktorij za pohranu zvučnih datoteka koje se koriste za melodije alarmnih poruka.
DIRECTORY_DCIM	Direktorij za pohranu slika i videa, primarno dobivenih putem kamere Android uređaja.
DIRECTORY_DOCUMENTS	Direktorij pohranu datoteka korisnika.
DIRECTORY_DOWNLOADS	Direktorij za pohranu svih datoteka preuzetih putem mreže.
DIRECTORY_MOVIES	Direktorij za pohranu filmova.
DIRECTORY_MUSIC	Direktorij za pohranu glazbe.
DIRECTORY_NOTIFICATIONS	Direktorij za pohranu zvučnih datoteka koje se koriste kod dojava notifikacija.
DIRECTORY_PICTURES	Direktorij za pohranu slika.
DIRECTORY_PODCASTS	Direktorij za pohranu zvučnih datoteka vezanih za „podcast“.
DIRECTORY_RINGTONES	Direktorij za pohranu zvučnih datoteka koje se odnose na melodije poziva.

Sve navedene lokacije su samo konvencija, no korisnik za zapis pojedinih sadržaja može odabrati i druge lokacije. Izvršavanjem kôda iz prethodnog primjera korištenjem adb konzole moguće je izlistati sadržaj vanjske memorije te navigirati do kreirane datoteke u Download direktoriju. Također, iz popisa direktorija može se uočiti struktura direktorija koja odgovara pobrojenju Environment prikazanom u tablici Tablica 14.

```

root@vbox86p:/sdcard # ls
Alarms
Android
DCIM
Download
Movies
Music
Notifications
Pictures
Podcasts
Ringtones
storage
root@vbox86p:/sdcard # cd Download
root@vbox86p:/sdcard/Download # ls
gapps-L-4-21-15.zip
temp.txt
root@vbox86p:/sdcard/Download # cat temp.txt
Hello world from temp!
root@vbox86p:/sdcard/Download #
    
```

Isto tako, korištenjem mobilne aplikacije za pregled sadržaja datotečnog sustava Android uređaja (*File Manager*) može je navigirati do kreirane datoteke, što je prikazano na slici (Slika 59).



*Slika 59. Prikaz sadržaja direktorija Download na eksternoj memoriji putem File Manager-a*

#### 4.2.4 Brisanje datoteka

Vrlo jednostavnom metodom `delete`, nad instancom klase `File` brisanje datoteke vrši se sljedećom linijom kôda:

```
file.delete();
```

Navedena metoda ima logički povratni tip, stoga je moguće provjeriti uspješnost brisanja.

### 4.3 Rad sa mobilnom bazom podataka

Android omogućuje rad sa nekoliko različitih vrsta baza podataka, primjerice BerkeleyDB, Couchbase Lite, LevelDB, UnQLite i mnoge druge. No, zadana baza podataka koja je sastavni dio Android operacijskog sustava je SQLite, pa baš upravo iz tog razloga će se u narednim primjerima koristiti navedena baza.

*SQLite* je transakcijska SQL baza podataka koja ne koristi server i nije ju potrebno dodatno konfigurirati, a zapisana je na datotečnom sustavu u obliku datoteke [29]. U ovom poglavlju bit će razmotrena dva načina korištenja SQLite baze podataka:

- a) Nativni pristup, korištenjem dostupnih klasa i metoda iz Android operacijskog sustava, odnosno paketa `android.database.sqlite`.
- b) Pristup putem biblioteke za objektno-relacijsko mapiranje (*engl. ORM – object-relational mapping*).

No, prije daljnjeg rada sa bazama, promotrimo primjer aplikacije CookBook i potrebe za pohranom podataka.

#### 4.3.1 Analiza potrebe zapisa podataka za CookBook mobilnu aplikaciju

Kako je prije rečeno, jedini zahtjev u mobilnoj aplikaciji CookBook je zapisivati podatke o receptima. Promotrimo što to znači za razvojne inženjere mobilne aplikacije tako da prepoznamo entitete. Entitet se u ovom kontekstu na objekte realnog svijeta koji su opisani specifičnim skupom atributa i čine vlastitu odijeljenu cjelinu, a u bazi podataka predstavljaju jednu tablicu. U tom svjetlu, entitetne klase odnose su one klase koje programskim jezikom opisuju entitet, a namijenjene su radu sa objektima koji se čitaju ili pohranjuju u relacijsku bazu podataka. Modeliranje baza podataka tema za čitavi novi predmet, u ovom dijelu dajemo brzi pregled kroz najosnovnije elemente potrebne za izradu mobilne baze podataka.



Kako bismo odredili koje entitetne klase su potrebne za CookBook, promotrimo sljedeći recept:

#### Tijesto za pizzu

**Priprema:** Kvasac razmutiti u mlakoj vodi i dodati šećera te pustiti da stoji oko 5 minuta na mjestu bez protoka zraka. Brašnu dodati sol i dignuti kvasac te lagano umijesiti da se izmiješaju sastojci. Prekriti tijesto i ostaviti u frižideru preko noći. Drugi dan, tijesto položiti na brašno, te staviti brašno s gornje strane i kružnim pokretima ruku napraviti okruglo tijesto (ne koristiti valjak). Tijesto je spremno za dodavanje sastojaka.


#### Sastojci:

20g svježeg kvasca  
1 žličica šećera  
1 žličica soli  
200 ml mlake vode  
320g pšeničnog glatkog (visoko proteinskog) brašna

Svakako, uočljivo je da je prvi entitet sam recept (*Recipe*). Recept čine naziv, opis pripreme te niz sastojaka. Kako relacijske baze podataka dolaze iz relacijske teorije u matematici, prilikom dizajna relacijskog modela potrebno se držati određenih pravila i notacija. Dakle, prvi korak je napraviti shemu relacije, koja se sastoji od imena relacije i popisa imena atributa. Trenutno to je:

**Recipe** (id\_recipe, name, description, ingredient).

Primarni atribut, odnosno ključ je onaj koji jednoznačno definira određeni entitet, odnosno redak u tablici. Isto tako, prethodnu relacijsku shemu, može se prikazati pomoću relacijskog ERA modela koji prikazuje entitete, attribute i njihove veze, što je vidljivo na slici (Slika 60).

Recipe		
 id_recipe	int	NN (PK)
name	varchar(50)	
description	TEXT	
ingredient	TEXT	

Slika 60. ERA model sa samo jednim entitetom, *Recipe*

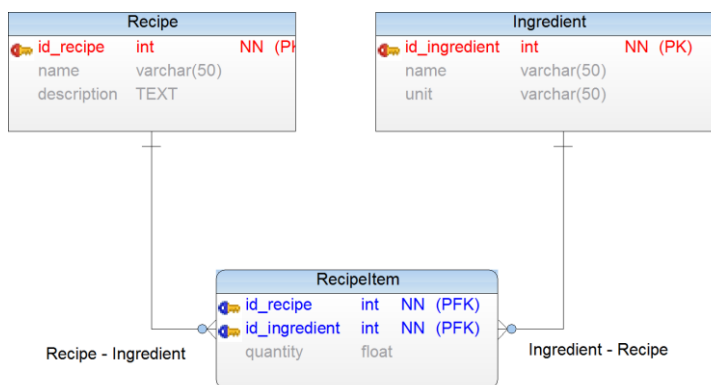
No trenutno, promotrimo li ERA model, može se uočiti da ukoliko imamo 2 recepta koja oboje primjerice trebaju brašno, svaki puta će ponovno trebati unositi brašno. Slično tome, teško će se raditi filtriranje podataka, primjerice ako želimo sve recepte koji kao sastojak imaju piletinu. Dakle, pojavljuju se problem u navedenom modelu, što svoj korijen ima u višestrukom ponavljanju podataka, odnosno redundanciji, koja je samo jedan od problema koji se mogu pojaviti. Mogući su i problemi s performansama baze podataka te integritetom podataka koji stvaraju anomalije kod kreiranja novih zapisa, brisanja ili njihovog ažuriranja.

Kako bi se navedeni problemi riješili, pristupa se postupku normalizacije baze podataka kojom tablice smještano u neku od *normalnih formi*. Postoji sedam normalnih formi i sve svoj korijen imaju u matematičkoj logici stoga je normalizacija formalno definiran proces. Za praktične svrhe, najčešće su dovoljne prve tri normalne forme. Za one koji žele znati svih sedam normalnih forma te vezanih pravila, upućujemo na literaturu vezanu uz teoriju baza podataka; [30] [31]. Jedno od pravila normalizacije jest da svi atributi u tablici moraju biti atomarni, odnosno ne rastavljivi. Takav primjer je atribut sastojak (*engl. ingredient*), odnosno jedan recept može imati puno sastojaka. Svaki sastojak ima ime i mjernu jedinicu, stoga bi relacijska shema sada bila:

**Recipe** (id\_recipe, name, description, ingredient),  
**Ingredient**(id\_ingredient, name, unit).

Isto tako, jedan sastojak može svakako biti na više recepata, pa je stoga brojnost veze između tablice Recipe i Ingredient *više naprama više*, odnosno *M:N*. To predstavlja problem jer više ne možemo jednoznačno identificirati koji sastojak pripada kojem receptu i obratno.

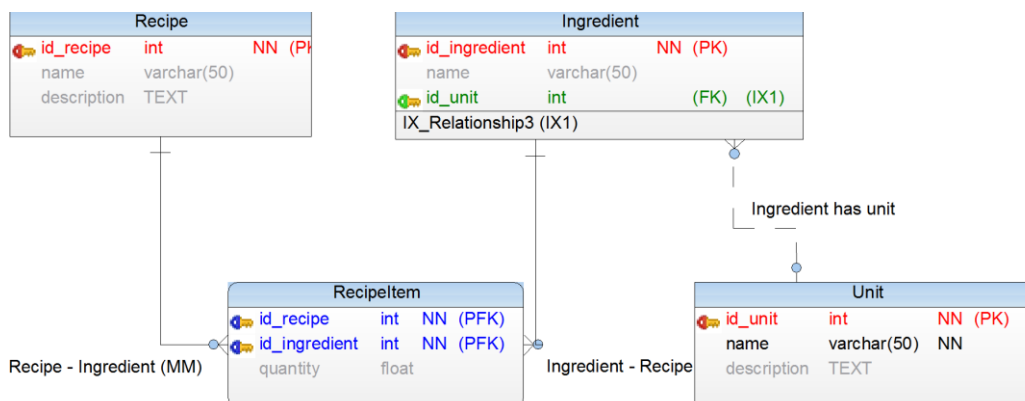
Taj problem rješava se uvođenjem *tablice slabih entiteta*, odnosno trećom tablicom prema kojoj obje navedene tablice imaju vezu više, dok ona prema obje tablice ima vezu jedan. Isto tako, ona ima *složeni ključ* sastavljen od primarnih ključeva tablica u odnosu M:N. Prikaz rješenja dan je slikom (Slika 61).



Slika 61. Uvođenje tablice slabih entiteta

U ovom slučaju, ako je potrebno dohvatiti sve sastojke nekog recepta, potrebno je tablicu RecipeItem filtrirati prema stranom ključu id\_recipe. Slično tome, želimo li dohvatiti sve recepte koji sadrže određeni sastojak, potrebno je napraviti filtriranje po ključu id\_ingredient. Kako tablica RecipeItem veže sastojak i recept, kao treći atribut dodana je i količina koja se odnosi na količinu s kojom neki sastojak ulazi u recept.

Nadalje, sličan problem kao i ranije postoji u tablici Ingredient, a odnosi se na mjernu jedinicu (*engl. unit*). Kako bi moglo doći do ponavljanja podataka koja je nepoželjna, atribut jedinice potrebno je staviti u novu tablicu.



Slika 62. Normalizirana baza podataka

Slika 62 prikazuje konačan ERA model za CookBook mobilnu aplikaciju. Atribut mjerne jedinice postao je tablica, a uz to tablici je dodan primarni ključ i novi atribut opisa.

U svrhu pojednostavljenja primjera CookBook, za implementaciju ćemo koristiti ERA model prikazan na slici Slika 61.

### 4.3.2 Nativni pristup

Za demonstraciju nativnog pristupa korištenju baze podataka kod Androida [32], u postojećem projektu CookBook mobilne aplikacije, kreirati novi paket naziva `dbnative`, koji će sadržavati sve potrebne klase za kreiranje baze podataka i rad sa tablicama.

Temeljna klasa za rad sa bazom podataka je `SQLiteOpenHelper` čijim nasljeđivanjem je potrebno implementirati dvije temeljne metode, `onCreate` i `onUpgrade`. Kako i ime sugerira, na događaj kreiranja baze (dakle baza podataka još ne postoji), unutar metode `onCreate` implementira se programski kôd i vezane SQL naredbe putem kojih se generira baza podataka. Kod događaja dogradnje (`onUpgrade`) baze vršimo korekciju (ako želimo mijenjati relacije), što se signalizira prema broju verzije trenutne baze podataka, koja je obavezan parametar kod metoda za pristup bazi podataka.

Prva klasa koju je time potrebno dodati u paket `dbnative` je `DbHelper`, a ona nasljeđuje temeljnu klasu za pristup bazi podataka, što je pokazano u sljedećem primjeru:

```
public class DbHelper extends SQLiteOpenHelper {
    public DbHelper(Context context, String name, SQLiteDatabase.CursorFactory
factory, int version) {
        super(context, name, factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        // implementacija kreiranja baze podataka
        db.execSQL("CREATE TABLE Recipe (id INTEGER PRIMARY KEY, name TEXT,
description TEXT);");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // implementacija ažuriranja baze podataka
    }
}
```

Nakon toga je potrebno kreirati entitetnu klasu koja će odgovarati tablici iz baze podataka. Za to je potrebno napraviti novi paket, nazvati ga `entities` a unutar istog kreirati prvu entitetnu klasu, `Recipe`:

```
public class Recipe {
    public String name;
    public String description;

    public Recipe(String name, String description) {
        this.name = name;
        this.description = description;
    }
}
```

Uz tu klasu, potrebna je i klasa pomoću koje će se pristupati bazi podataka, a prikazana je na sljedećem primjeru:

```
public class DataAdapter {
```

```

// naziv datoteke baze podataka i trenutna verzija
public static final String DATABASE_NAME = "database.db";
public static final int DATABASE_VERSION = 1;

private DBHelper dbHelper;
private SQLiteDatabase sqLiteDatabase;

private Context context;

public DataAdapter(Context context) {
    this.context = context;
}
// otvaranje veze prema bazi podataka za čitanje
public SQLiteDatabase openToRead() throws android.database.SQLException {
    dbHelper = new DBHelper(context, DATABASE_NAME, null,
DATABASE_VERSION);
    sqLiteDatabase = dbHelper.getReadableDatabase();
    return sqLiteDatabase;
}
// otvaranje veze prema bazi podataka za pisanje
public SQLiteDatabase openToWrite() throws android.database.SQLException {
    dbHelper = new DBHelper(context, DATABASE_NAME, null,
DATABASE_VERSION);
    sqLiteDatabase = dbHelper.getWritableDatabase();
    return sqLiteDatabase;
}
// zatvaranje veze prema bazi podataka
public void close() {
    sqLiteDatabase.close();
    dbHelper.close();
}
}

```

Može se uočiti da navedena klasa implementira metode za čitanje i pisanje u bazu podataka, a isto tako pohranjuje naziv datoteke baze podataka te njenu trenutnu verziju.

Zbog preglednijeg načina rada, svakom entitetu pridružiti će se odgovarajuća adapterska klasa koja će naslijediti `DataAdapter` klasu te je koristiti za pristup bazi na jedinstven način. Time se verzija i opći podaci o bazi podataka bit će na jednom mjestu.

Za tablicu `Recipe`, ta klasa će se stoga zvati `RecipeAdapter`, bit će smještena u paketu `dbnative`, a nasljeđuje ranije kreiranu klasu `DataAdapter`:

```

public class RecipeAdapter extends DataAdapter {

    private static final String TABLE = "Recipe";
    private static final String KEY_ID = "id";

    public RecipeAdapter(Context context) {
        super(context);
    }

    public long insertRecipe(Recipe recipe) {
        // dodavanje novog recepta
    }

    public List<Recipe> getAllRecipes() {
        // pregled svih recepata
        return null;
    }
}

```

U navedenom kôdu nedostaje implementacija metode za unos novog recepta i pregled svih, odnosno dohvaćanje liste svih trenutno unesenih recepata u mobilnoj bazi podataka. Kao što je vidljivo, tablica ima svoje ime, koje treba odgovarati imenu u bazi podataka, te oznaku primarnog ključa.

Podaci namijenjeni za zapis u bazi podataka pohranjuju se pomoću klase `ContentValues`, pa će stoga metoda `insertRecipe` biti:

```
public long insertRecipe(Recipe recipe){
    ContentValues contentValues = new ContentValues();

    contentValues.put("name", recipe.name);
    contentValues.put("description", recipe.description);

    SQLiteDatabase db = openToWrite();

    return db.insert(TABLE, null, contentValues);
}
```

Iz priloženog kôda uočljivo je da objekt klase `ContentValues` koristi zapise u obliku ključ-vrijednost koje potom zapisuje u odgovarajuću tablicu u bazi podataka. Iz super klase `DataAdapter` koristi se metoda koja otvara vezu prema bazi podataka namijenjenu za pisanje.

Kako bi se čitali podaci, potrebno je koristiti `Cursor` klasu koja nakon postavljenog SQL upita sadrži rezultate, kao što to prikazuje sljedeći primjer:

```
public List<Recipe> getAllRecipes(){
    List<Recipe> result = new ArrayList<Recipe>();

    String[] columns = new String[]{KEY_ID, "name", "description"};
    SQLiteDatabase db = openToRead();

    Cursor cursor = db.query(TABLE, columns, null, null, null, null, null);

    for(cursor.moveToFirst(); !(cursor.isAfterLast()); cursor.moveToNext()){
        String name = cursor.getString(cursor.getColumnIndex("name"));
        String description =
        cursor.getString(cursor.getColumnIndex("description"));
        Recipe recipe = new Recipe(name, description);
        result.add(recipe);
    }

    return result;
}
```

Iz primjera je vidljivo kako nakon postavljanja upita nad tablicom pomoću metode `query`, rezultat se pohranjuje u instancu klase `Cursor`. Nadalje, prolaskom kroz sve elemente tog objekta, čitaju se vrijednosti iz baze podataka prema nazivu atributa. U ovom slučaju, rezultat je lista recepata.

Sljedeći primjer pokazuje na koji način se koriste kreirane klase za nativni rad sa bazom podataka. Nakon kreiranja instance klase `RecipeAdapter`, korištenjem njene metode `insertRecipe`, kreiraju se recepti (odnosno instance klase `Recipe`). Po završetku korištenja baze podataka, potrebno je zatvoriti vezu.

Također, primjer pokazuje ispis trenutnog sadržaja tablice `Recipe` u konzolnoj liniji, pozivanjem metode `getAllRecipes`. Izvođenje navedenog kôda najbolje je isprobati unutar glavne aktivnosti.

```
// pohrana zapisa
RecipeAdapter recipeAdapter = new RecipeAdapter(this);
recipeAdapter.insertRecipe(new Recipe("Pizza", "Priprema: Kvasac razmutiti..."));
recipeAdapter.insertRecipe(new Recipe("Bolognese", "Priprema: nasjeckajte povrće..."));
recipeAdapter.close();

//čitanje zapisa
System.out.println("Current entries: \n");
for(Recipe recipe : recipeAdapter.getAllRecipes()){
    System.out.println("Name: " + recipe.name + "\n");
}
```

Izvršavanjem prethodnog primjera, u konzolnom retku Android Studia (Logcat) naći će se ispis svih recepata:

```
07-21 14:13:56.878 7487-7487/hr.heureka.cookbook I/System.out: Current entries:
07-21 14:13:56.882 7487-7487/hr.heureka.cookbook I/System.out: Name: Pizza
07-21 14:13:56.882 7487-7487/hr.heureka.cookbook I/System.out: Name: Bolognese
```

Isto tako, korištenjem adb konzole na Android uređaju, navigacijom do database direktorija, može se vidjeti i kreirana baza podataka: `database.db` i `database.db-journal`. `database.db` je SQLite datoteka, odnosno sama baza podataka, dok je `database.db-journal` datoteka je privremena pomoćna datoteka koja se koristi kod transakcija SQLite baze podataka.

```
root@vbox86p: /Data/Data/hr.heureka.cookbook # ls
cache
databases
files
lib
shared_prefs
root@vbox86p: /Data/Data/hr.heureka.cookbook # cd databases
root@vbox86p: /Data/Data/hr.heureka.cookbook/databases # ls
database.db
database.db-journal
root@vbox86p: /Data/Data/hr.heureka.cookbook/databases #
```

Osim toga, adb konzola omogućuje direktno spajanje na bazu podataka i korištenje SQL upita, naredbom `sqlite3 naziv-baze`, odnosno u ovom slučaju `sqlite3 database.db`:

```
root@vbox86p: /Data/Data/hr.heureka.cookbook/databases # ls
database.db
database.db-journal
root@vbox86p: /Data/Data/hr.heureka.cookbook/databases # sqlite3 database.db
SQLite version 3.8.6 2014-08-15 11:46:33
Enter ".help" for usage hints.
sqlite> .tables
Recipe          android_metadata
sqlite> select * from recipe;
1|Pizza|Priprema: Kvasac razmutiti...
2|Bolognese|Priprema: nasjeckajte povrće...
sqlite> select * from recipe where id = 1;
1|Pizza|Priprema: Kvasac razmutiti...
sqlite> .quit
root@vbox86p: /Data/Data/hr.heureka.cookbook/databases #
```

Primjer prikazuje spajanje na bazu podataka, izlistavanje svih trenutno dostupnih tablica naredbom `.tables` (`Recipe` i `android_metadata`, koja sadrži opće podatke o bazi, primjerice korišteni

jezik). Po spajanju na bazu, moguće je izvršavati i SQL upite što je prikazano upitom `SELECT`, a izlazak iz aplikacije `sqlite3` vrši se naredbom `.quit`. Svaki upit nad bazom podataka mora završavati sa znakom točka-zarez `;`.

Time je prikazan proces rada s bazom podataka korištenjem osnovnog (nativnog) pristupa. Navedeni odlomci ne prikazuju ovaj pristup u cijelosti jer nisu kreirane metode za nadogradnju strukture podataka ili ažuriranje podataka niti je implementiran rad sa svim tablicama. Analizom ovakvoga pristupa u radu s bazom podataka možemo uočiti da zahtjeva pisanje mnogo programskog kôda te da posljedično troši i dosta vremena razvojnog tima.

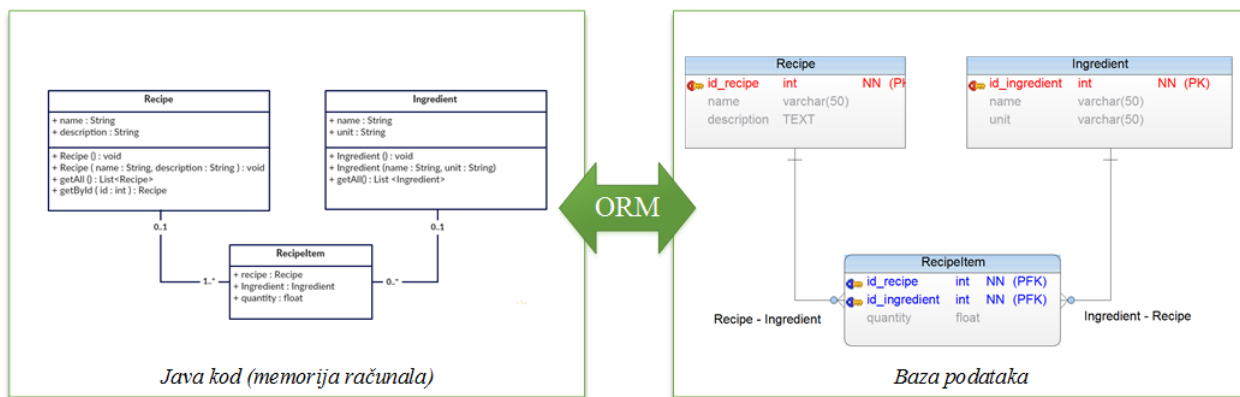
*Uz pristup kreiranju baze podataka iz programskog kôda, pomoću alata `DB Browser for SQLite` moguće je kreiranje baze podataka putem grafičkog alata, no taj dio nije pokriven ovim priručnikom stoga preporučamo za one koji žele znati više da posjete [sqlitebrowser.org](http://sqlitebrowser.org), preuzmu i isprobaju navedeni alat.*

Za pregledniju arhitekturu mobilne aplikacije, smanjenja potrebnog kôda za rad s bazom podataka i njeno učinkovitije korištenje obično se koriste sustavi za objektno-relacijsko mapiranje, ORM. Android ima više različitih biblioteka treće strane za ORM, primjerice Sugar ORM, Sprinkles, DB Flow, Active Android, itd. U sljedećem poglavlju, prikazuje se Active Android kroz implementaciju istog primjera. Active Android, odabran je jer je prilično rasprostranjen i jedan od najranijih Android ORM sustava, no svaki od prethodno nabrojanih ima svojih kvaliteta i valja ih razmotriti prilikom odluke o korištenju ORM-a.

#### 4.3.3 ORM – Active Android

*Objektno-relacijski mapper* je sustav koji omogućuje rad s bazom podataka korištenjem objektno-orijentirane paradigme, pri čemu se odvaja model klasa od sustava za upravljanje bazom podataka. U načelu, ORM je skup biblioteka koje omataju bazu podataka skupom klasa za jednostavniju primjenu i automatizaciju standardnih operacija (SQL upita) poput kreiranja baze podataka, čitanja, zapisivanja, pretraživanja i dr.

U prethodnom, nativnom pristupu, kreiranje baze podataka, tablica i postavljanje upita nad istima zahtijevalo je za svaku tablicu posebnu adaptersku klasu koja sadrži sve SQL upite potrebne za rad. Takvim pristupom, programski kôd je zavisan od baze podataka, jer iako je pristup i sam SQL standardiziran, uvijek postoje sitne razlike koje će izazvati pogreške u izvođenju, ukoliko se mijenja sam sustav za upravljanje bazom podataka. Kod ORM-a, njegova implementacija često sadrži mogućnost odabira više sustava za upravljanje bazom podataka, pa time on predstavlja most između programskog jezika, odnosno objekata iz kôda prema entitetima u bazi podataka. Primjenom ORM-a, zamjena baze podataka značila bi minimalne promjene u programskom kôdu (barem u teoriji, iako nije uvijek slučaj, a u softverskoj zajednici postoje brojne diskusije na navedenu temu). Slika 63 prikazuje kako ORM mapira tablice iz baze podataka u klase programskog kôda, i obratno.



Slika 63. Mapiranje klasa iz memorije u tablice u bazi podataka

ORM omogućuje dva osnovna pristupa razvoju podatkovnog sloja mobilne aplikacije:

- a) *Prvo kôd* – temeljem kreiranih entitetnih klasa u programskom kôdu, ORM generira bazu podataka. ORM sam brine o tipovima podataka i odnosima između tablica, pa čak veze više naprama više iz programskog kôda (gdje je to dozvoljeno), automatski rješava uvođenjem tablice slabih entiteta. No, trenutna implementacija Active Androida tu funkcionalnost još ne omogućuje, pa prema slici Slika 63, model klasa odgovara modelu podataka.
- b) *Prvo baza podataka* – ORM se povezuje na postojeću bazu podataka te generira klase kojima se pristupa do pojedinih tablica. Active Android u trenutnoj implementaciji nema ovu mogućnost, no mnogi drugi ORM-ovi za druge web ili stolne aplikacije imaju.

#### 4.3.4 Kreiranje baze podataka putem Active Android biblioteke

Za korištenje Active Android biblioteke u build.gradle (Project: Memento) treba dodati referencu na Maven repozitorij, na kojem se biblioteka nalazi:

```
allprojects {
    repositories {
        jcenter()
        // za ActiveAndroid
        mavenCentral()
        maven { url "https://oss.sonatype.org/content/repositories/snapshots/" }
    }
}
```

Nakon se u build.gradle (Module:app) dodaje referenca na samu Active Android biblioteku:

```
// active android
compile 'com.michaelpardo:activeandroid:3.1.0-SNAPSHOT'
```

I konačno, u AndroidManifest.xml dodaju se meta podaci o bazi podataka, odnosno ime i verzija. Kako je ranije već spomenuto, prilikom bilo kakve promjene nad strukturom baze podataka, verziju je potrebno ručno uvećati za jedan, kako bi se pokrenula onUpdate metoda. Neposredno nakon oznake koja se odnosi na glavu aktivnost, no prije završetka oznake aplikacije dodati:

```
<meta-data
    android:name="AA_DB_NAME"
```



```

    android:value="cookbook.db" />
<meta-data
    android:name="AA_DB_VERSION"
    android:value="1" />

```

Baza podataka je spremna za korištenje, što kod Active Androida znači da je potrebno kreirati entitetne klase koje će naslijediti Model klasu iz Active Androida, a *anotacijama*<sup>14</sup> @Table i @Column naznačiti u klasi što je tablica, atribut.

Za entitetne klase, kreirati novi paket, nazvati ga entities, a unutar navedenog treba kreirati tri klase; Recipe, Ingredient i RecipeItem. Klasa Recipe je:

```

@Table(name = "recipe")
public class Recipe extends Model {
    @Column(name = "name")
    public String name;
    @Column(name = "description")
    public String description;

    public Recipe() {};

    public Recipe(String name, String description) {
        this.name = name;
        this.description = description;
    }
}

```

Recipe nasljeđuje klasu Model čime dobiva karakteristike entiteta, odnosno nasljeđuje metode poput save i delete. Anotacijom Table dodanom neposredno prije definicije klase Recipe, daje se uputa Active Androidu da će klasa Recipe biti objektna reprezentacija tablice recipe u bazi podataka. Slično tome, anotacijama @Column definiraju se atributi entiteta.

Ono što nedostaje su metode koje bi omogućile dohvaćanje svih zapisa iz baze podataka ili određenog zapisa prema identifikatoru, što se može napraviti dodavanjem statičkih metoda:

```

public static List<Recipe> getAll() {
    return new Select().from(Recipe.class).orderBy("name DESC").execute();
}

public static Recipe getById(int id) {
    return new Select().from(Recipe.class).where("id = " + id).executeSingle();
}

```

Kao što je vidljivo, ne koristi se SQL već se koriste metode iz ORM-a, čime je kôd neovisan o sustavu za upravljanje bazom podataka.

Logika izrade klase Ingredient je identična:

```

@Table(name = "ingredient")
public class Ingredient extends Model {
    @Column(name = "name")
    public String name;
}

```

<sup>14</sup> Anotacije su meta-instrukcije koje se dodaju u Java kodu kako bi se pojedinim klasama, metodama ili atributima dodale (anotirale) dodatne karakteristike. Programski gledano, anotacija je programsko sučelje.

```

@Column(name = "unit")
public String unit;

public Ingredient();

public Ingredient(String name, String unit) {
    this.name = name;
    this.unit = unit;
}

// dohvati sve sastojke
public static List<Ingredient> getAll(){
    return new Select().from(Ingredient.class).orderBy("name DESC").execute();
}
}

```

Za razliku od entitetnih klasa `Recipe` i `Ingredient`, klasa `RecipeItem` kao atribute koristi baš te složene klase. Kako se radi sa ORM-om, atributi koji se odnose na druge entitetne klase se koriste baš kao i bilo koji drugi atributi, a Active Android će se pobrinuti da se strani ključevi dobro zapisuju.

```

@Table(name = "recipe_item")
public class RecipeItem extends Model {
    @Column(name = "recipe", onDelete = Column.ForeignKeyAction.CASCADE)
    public Recipe recipe;
    @Column(name = "ingredient", onDelete = Column.ForeignKeyAction.CASCADE)
    public Ingredient ingredient;
    @Column(name = "quantity")
    public float quantity;

    public RecipeItem() {}

    public RecipeItem(Recipe recipe, Ingredient ingredient, float quantity) {
        this.recipe = recipe;
        this.ingredient = ingredient;
        this.quantity = quantity;
    }

    // dohvati sve recepte
    public List<Recipe> getRecipes(){
        return getMany(Recipe.class, "RecipeItem");
    }

    // dohvati sastojke za recept
    public static List<RecipeItem> getIngredientsForRecipe(Recipe recipe){
        return new Select().from(RecipeItem.class).where(" Recipe = ? ",
recipe.getId()).execute();
    }

    // dohvati sve sastojke
    public List<Ingredient> getIngredients(){
        return getMany(Ingredient.class, "RecipeItem");
    }
}

```

Sljedeći primjer, kojeg je najbolje izvršiti u glavnoj aktivnosti, inicijalizira Active Android (`ActiveAndroid.initialize()` treba izvršiti samo jednom). Nakon toga u bazu podataka dodaje se pet sastojaka i dva recepta. Važno je uočiti da za to SQL uopće nije korišten, a metoda

save koja je naslijeđena iz klase Model Active Androida sadrži kôd za spremanje koji je automatski generiran.

```
// dodavanje sastojaka
Ingredient flour = new Ingredient("Flour", "g");
flour.save();
Ingredient sugar = new Ingredient("Sugar", "spoon");
sugar.save();
Ingredient salt = new Ingredient("Salt", "spoon");
salt.save();
Ingredient yeast = new Ingredient("Yeast", "g");
yeast.save();
Ingredient water = new Ingredient("Water", "ml");
water.save();

// dodavanje recepta
Recipe pizza = new Recipe("Pizza", "Priprema: Kvasac razmutiti...");
pizza.save();
Recipe bolognese = new Recipe("Bolognese", "Priprema: nasjeckajte povrće...");
bolognese.save();

//pridruživanje sastojaka receptu
RecipeItem recipeItemPizzaSalt = new RecipeItem(pizza, salt, 1f);
recipeItemPizzaSalt.save();
RecipeItem recipeItemPizzaSugar = new RecipeItem(pizza, sugar, 1f);
recipeItemPizzaSugar.save();
RecipeItem recipeItemPizzaFlour = new RecipeItem(pizza, flour, 320f);
recipeItemPizzaFlour.save();
RecipeItem recipeItemBologneseWater = new RecipeItem(bolognese, water, 300f);
recipeItemBologneseWater.save();
```

Također, kod dodavanja sastojaka receptu, može se uočiti kako se ovdje radi sa objektima. Dakle, RecipeItem u konstruktoru kao parametar prima objekt recepta i objekt sastojka koji se receptu pridružuje u određenoj količini.

Izvršavanjem navedenog kôda, pomoću adb konzole može se navigirati do novokreirane baze podataka, te postavljanjem upita vidjeti unesene podatke. Pretraživanjem se može vidjeti recept i njegovi sastojci ili obratno, sastojci koji se nalaze na pojedinom receptu.

Sljedećim kôdom, koji se može napisati odmah nakon kôda kojim se podaci unose u bazu podataka, ispisuje se sadržaj tablice RecipeItem, tako da se for petljom prolazi kroz sve recepte. Ista petlja ima ugniježđenu petlju koja zatim za dohvaćeni recept, dohvaća sastojke iz tablice RecipeItem:

```
for(Recipe recipe : Recipe.getAll()){
    System.out.println("Recipe: " + recipe.name);
    for(RecipeItem recipeItem : RecipeItem.getIngredientsForRecipe(recipe)){
        System.out.println("Ingredient: " + recipeItem.ingredient.name);
    }
}
```

U Logcat konzoli, među mnoštvom zapisa pronaći će se i ispis:

```
07-22 14:16:12.296 8498-8498/hr.heureka.cookbook I/System.out: Recipe: Pizza
07-22 14:16:12.299 8498-8498/hr.heureka.cookbook I/System.out: Ingredient: Sugar
07-22 14:16:12.299 8498-8498/hr.heureka.cookbook I/System.out: Ingredient: Salt
07-22 14:16:12.299 8498-8498/hr.heureka.cookbook I/System.out: Ingredient: Flour
07-22 14:16:12.299 8498-8498/hr.heureka.cookbook I/System.out: Recipe: Bolognese
07-22 14:16:12.302 8498-8498/hr.heureka.cookbook I/System.out: Ingredient: Water
```

Isto kao i ranije kod nativnog pristupa bazi podataka, korištenjem adb-a mogu se provjeriti upisani podaci, kao što je prikazano u sljedećem primjeru:

```
root@vbox86p: /Data/Data/hr.heureka.cookbook/databases # sqlite3 cookbook.db
SQLite version 3.8.6 2014-08-15 11:46:33
Enter ".help" for usage hints.
sqlite> .tables <- 1
android_metadata  ingredient          recipe          recipe_item
sqlite> select * from recipe;
1|Priprema: Kvasac razmutiti...|Pizza
2|Priprema: nasjeckajte povrće...|Bolognese
sqlite> select * from ingredient; <- 2
1|Flour|g
2|Sugar|spoon
3|Salt|spoon
4|Yeast|g
5|Water|ml
sqlite> select * from recipe_item where ingredient = 4; <- 3
sqlite> select * from recipe_item; <- 4
1|2|1.0|1
2|3|1.0|1
3|1|320.0|1
4|5|300.0|2
sqlite> select * from recipe_item where ingredient = 1; <- 5
3|1|320.0|1
sqlite> select * from recipe where id = 1; <- 6
1|Priprema: Kvasac razmutiti...|Pizza
sqlite> select * from recipe_item where ingredient = 5; <- 7
4|5|300.0|2
sqlite> select * from recipe where id = 2; <- 8
2|Priprema: nasjeckajte povrće...|Bolognese
sqlite> .quit <- 9
root@vbox86p: /Data/Data/hr.heureka.cookbook/databases #
```

Sqlite3 programom spajamo se na datotečnu bazu podataka `cookbook.db` koja se kreira uz već postojeću `database.db` korištenu kod primjera prethodnog poglavlja. Prema brojčanim komentarima (*crveno*), slijedi objašnjene za svaku izvršenu naredbu (pojedini upiti su ponovljeni, no sa drugačijim parametrima kako bi demonstrirao SQL i usporedili rezultati):

- 1.) *Naredba baze podataka za ispis svih tablica.*
- 2.) *SQL upit kojim se ispisuju čitav sadržaj tablice `Ingredient`. Kao što se vidi, svi sastojci upisani putem Java kôda su sadržani u tablici, bez da je ručnog pisanja SQL-a.*
- 3.) *SQL upit kojim se iz tablice koja sadrži sastojke pojedinih recepata ispisuju svi recepti koji koriste sastojak sa identifikatorom 4, odnosno ispis svih recepata i sastojaka koji koriste kvasac.*
- 4.) *SQL upit kojim se ispisuju čitav sadržaj tablice `RecipeItem`.*
- 5.) *SQL upit za ispis svih recepata i sastojaka koji koriste brašno (slično kao 3).*
- 6.) *SQL upit koji ispisuju recept sa identifikatorom 1 (bez sastojaka).*
- 7.) *SQL upit kojim se ispisuju svi recepti i sastojci koji koriste vodu.*
- 8.) *SQL upit koji ispisuju recept sa identifikatorom 2 (bez sastojaka).*
- 9.) *Naredba baze podataka za izlaz iz adb programa.*

Demonstracijom Active Androida trebalo bi uočiti razlike sa nativnim pristupom. Active Android pruža određenu fleksibilnosti i jednostavnost kod promjena u bazi podataka, a isto tako značajno olakšava rad sa podacima. Iako Active Android nije ORM u punom smislu, do određene mjere uklanja potrebu za primjenom SQL-a i smanjuje broj potrebnih izmjena u kôdu, u slučaju promjene strukture baze podataka.

## 4.4 Rad s web servisima

*Web servis* je naziv za standardiziranu uslugu razmjene podataka između klijenta i poslužitelja na webu. Kako se za prijenos podataka koristi HTTP, podaci koji se razmjenjuju između klijenta i poslužitelja formatirani su u jednom od dva proširiva tekstualna oblika, XML i JSON.

*XML* (engl. *extensible markup language*) je format za razmjenu podataka koji je napravljen sa svrhom da bude jednostavan, proširiv, samo-opisujući i čitljiv čovjeku i stroju [33]. XML pripada SGML skupini jezika koji je definiran standardom ISO 8879. Primjer sličnog, no strože definiranog jezika iz iste skupine je HTML, danas najrasprostranjeniji jezik za prikaz web stranica. Internet preglednici automatski prevode taj jezik u sadržaj razumljiv čovjeku, primjer poziva web servisa koji odgovara sa HTML-om je prikazan na slici (Slika 64 (a)). XML je vrlo sličan HTML-u, no slabije definiran, odnosno temeljni elementi XML-a, oznake i atributi, nisu definirani standardom, već posebnom XSD (engl. *XML schema definition*) datotekom, pisanom u standardiziranom XML formatu. Primjer strukture podataka formatirane u XML obliku prikazuje Slika 64 (b).

*JSON* (engl. *JavaScript Object Notation*) kao i XML je format za razmjenu podataka koji je napravljen s idejom da bude jednostavan za čitanje ljudima i računalima, a temelji se na programskom jeziku JavaScript [34]. Ideja vodilja kôd kreiranja JSON-a bila je smanjenje paketa u kojima se podaci šalju kako bi se ubrzao prijenos podataka. Također, format u kojem su podaci zapisani dozvoljava direktno vezanje na objekte u programskom kôdu. JSON se temelji na parovima ključ-vrijednost, a primjer odgovora web servisa prikazan je na slici (Slika 64 (c)).



(a)

```

{
  "coord": {
    "lon": 16.34,
    "lat": 46.3
  },
  "weather": [
    {
      "id": 803,
      "main": "Clouds",
      "description": "broken clouds",
      "icon": "04d"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 24.79,
    "pressure": 1018,
    "humidity": 69,
    "temp_min": 21.67,
    "temp_max": 28
  },
  "visibility": 10000,
  "wind": {
    "speed": 2.1,
    "deg": 10
  },
  "clouds": {
    "all": 75
  },
  "dt": 1469437172,
  "sys": {
    "type": 1,
    "id": 5883,
    "message": 0.0329,
    "country": "HR",
    "sunrise": 1469417322,
    "sunset": 1469471575
  },
  "id": 3188383,
  "name": "Varazdin",
  "cod": 200
}
                
```

(b)

```

<current>
  <city id="3188383" name="Varazdin">
    <coord lon="16.34" lat="46.3"/>
    <country>HR</country>
    <sun rise="2016-07-25T03:28:42" set="2016-07-25T18:32:55"/>
  </city>
  <temperature value="24" min="24" max="24" unit="metric"/>
  <humidity value="69" unit="%"/>
  <pressure value="1017" unit="hPa"/>
  <wind>
    <speed value="3.6" name="Gentle Breeze"/>
    <gusts/>
    <direction value="350" code="" name=""/>
  </wind>
  <clouds value="75" name="broken clouds"/>
  <visibility/>
  <precipitation value="0.155" mode="rain" unit="3h"/>
  <weather number="803" value="broken clouds" icon="04d"/>
  <lastupdate value="2016-07-25T08:00:00"/>
</current>
                
```

(c)

Slika 64. Primjeri odgovora web servisa u različitim formatima (a) HTML, (b) XML, (c) JSON

- a) I dok oba formata imaju svojih prednosti i nedostataka, danas se oba prilično intenzivno koriste kod web servisa, koji vrlo često imaju mogućnost prezentacije podataka u oba formata. Na taj način, sam klijent odabire format koji mu najviše odgovara. Iako ih postoje više, danas su najpopularnija dva pristupa konzumiranja web servisa, SOAP i REST. ).

#### 4.4.1 SOAP

*SOAP (engl. simple object access protocol)* je protokol koji služi za razmjenu podataka između web servisa, a koristi standardizirane poruke definirane XML-om [35]. Iako kao protokol implementacije najčešće koristi HTTP, SOAP je po definiciji neovisan o protokolima niže razine. Na taj način, SOAP nije ovisan o operacijskom sustavu, programskim jezicima i tehnologijama. Temeljni element SOAP-a je poruka koja se sastoji od tri osnovna dijela prikazana na slici (Slika 65).



Slika 65. Struktura SOAP poruke

- a) *SOAP omotnica* – je obavezni dio SOAP poruke koja definira njen početak i kraj, te identificira poruku.
- b) *SOAP zaglavlje* – je opcionalni dio SOAP poruke koji se koristi za definiranje opcionalnih atributa kod obrade poruke.
- c) *SOAP tijelo* - je obavezni dio SOAP poruke koji sadrži korisni sadržaj, odnosno sam podatak koji klijent i server razmjenjuju.

Uz navedeno, najvažnije karakteristike SOAP-a su:

- b) *Sigurnost*- uz sam SSL protokol (*engl. secure socked layer*), SOAP omogućuje uvođenje dodatnih elemenata zaštite (npr. WS-Security) čime postiže veća sigurnost i zbog čega je često korišten u transakcijama između velikih poduzeća.
- c) *Atomarne transakcije* – što znači da SOAP garantira sigurnu ispostavu poruke i ACID transakcije: A – atomarnost poruke (dostavlja sve podatke ili ništa), C – konzistenciju stanja servisa, I – izolacijsko svojstvo koje osigurava konzistenciju paralelne obrade i D – izdržljivost koje garantira postojanost obavljenih transakcija (bez obzira na vanjske elemente).

#### 4.4.2 REST

Za razliku od SOAP-a, *REST (engl. representational state transfer)* nije protokol već arhitekturni stil web servisa koji se temelji na HTTP protokolu vođen principima skalabilnosti, jednostavnosti, učinkovitosti, prenosivosti i pouzdanosti [36]. Nakon konzumacije REST servisa, on ne zadržava

stanje, što znači da poslužuje podatke klijentu, a potom komunikacija prestaje. Daljnju interakciju inicira klijent kroz univerzalne identifikatore resursa (engl. URI), putem jedne od pet dostupnih HTTP metoda: POST (kreiranje sadržaja), GET (čitanje sadržaja), PUT (modifikacija sadržaja<sup>15</sup>), PATCH (ažuriranje i modifikacija sadržaja), DELETE (brisanje sadržaja). Web servisi koji su građeni REST načelom, često imaju mogućnost prezentacije sadržaja u bilo kojem formatu (najčešće JSON i XML).

Kako REST koristi HTTP kao protokol prijenosa sadržaja, vrlo je jednostavan te ga je moguće konzumirati putem Internet preglednika, a razvoj klijenata i dokumentacije je značajnije jednostavnije. Česte odgovore koje traže klijenti moguće je čuvati u privremenoj memoriji što značajnije ubrzava rad i skalabilnost.

*Nećemo ulaziti u daljnju raspravu oko toga da li je bolji SOAP ili REST pristup korištenja web servisa i željeli bismo potaknuti čitatelja da sam istraži daljnje detalje, no vrijedi naglasiti da su oba pristupa u upotrebi, te da oba imaju prednosti i nedostatke. Odluku o korištenju pojedinog od navedenih pristupa treba donijeti prema argumentima prednosti i nedostataka za neku konkretnu primjenu.*

Kod razvoja Android mobilnih aplikacija, u ovom priručniku demonstrirati ćemo korištenje REST servisa, iz razloga što mobilne aplikacije gotovo u potpunosti koriste REST pristup. Tek kod nekih specifičnih slučajeva koristi se SOAP.

#### 4.4.3 Korištenje REST servisa kod Androida

Slično kao i kod razvoja mobilnih baza podataka, kod Androida postoje dva razvojna pristupa konzumiranja web servisa. Nativni, podržan samo bibliotekama dostupnima u Androidu, te automatiziran korištenjem bibliotekama treće strane.

Kod nativnog pristupa, potrebni su sljedeći elementi [27]:

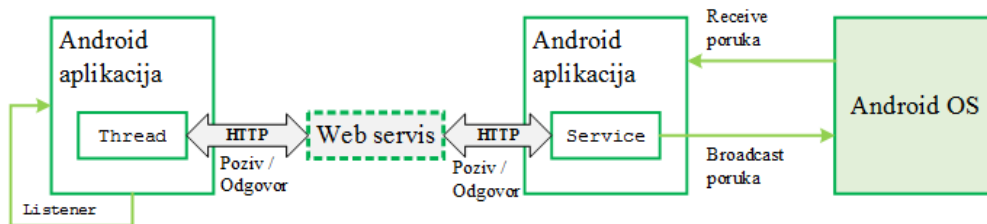
- a) Android servis ili dretva (nit) koji se koristi za slanje i primanje HTTP zahtjeva, tako da ne blokira aplikaciju za vrijeme slanja ili čekanja odgovora. Android ne dozvoljava izvođenje dugotrajnih operacija na glavnoj dretvi, npr. `onPostExecute` (izvršavanje HTTP POST metode).
- b) Klase i metode za slanje i primanje HTTP zahtjeva.
- c) Klase i metode za obradu primljenog sadržaja.

Slika 66 prikazuje pojednostavljeni prikaz događaja kod primjene osnovnog (nativnog) pristupa web servisima. Desni dio slike prikazuje slučaj u kojemu se koristi Android pozadinski servis, koji komunicira sa web servisom putem HTTP-a. Može se uočiti da nakon što stigne odgovor, Android servis aplikacije šalje `Broadcast` poruku na razini Android operacijskog sustava [27]. Ta poruka sadrži posebni identifikator koji definira tip usluge koji se traži. U ovom slučaju, ista Android aplikacija koja kreira putem servisa takvu poruku, jedina sadrži `Receiver` kojim se jedina i javlja na tu poruku. Nadalje, tu će poruku koristiti i dodatnim klasama obraditi njen sadržaj i dobiti odgovor iz od servisa.

Lijeva polovica slike prikazuje istu interakciju, no ovaj puta bez pozadinskog servisa nego sa dretvom. Naime, po kreiranju dretve koja će komunicirati sa web servisom, potrebno je kreirati i slušatelja (engl. `listener`) događaja dolaska podataka. Kada podaci stignu, taj će se događaj aktivirati i obraditi poruku. Ovom implementacijom izbjegnuto je korištenje Android OS-a u interakciji s web servisom.

<sup>15</sup> Poruka sadrži samo promjene





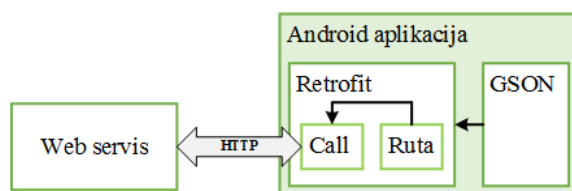
Slika 66. *Nativni pristup web servisima (lijevi dio putem dretve, desni dio putem klase Service)*

Zbog zavisnosti o servisu, formatu poruke, složenom kôdu i potrebom za poznavanjem detalja HTTP protokola, u ovom priručniku nećemo prikazivati detalje nativne implementacije korištenja web servisa. Umjesto toga, koristiti ćemo biblioteke treće strane koje će značajno olakšati primjenu web servisa, i to potrebu za servisom i dretvom, potrebu za obradama događaja HTTP protokola i poslužitelja te manualnu obradu dolazne poruke. Biblioteke koje će se koristiti su Retrofit, za rad sa servisima i GSON za obradu dolazne poruke.

#### 4.4.4 Retrofit i GSON

*Retrofit* je biblioteka treće strane koja kod rada s web servisima omogućuje da se prema metodama web servisa ponašamo kao i bilo kojim drugim Java metodama, kroz implementaciju sučelja koje definiraju rute (putanje) do metoda web servisa [37]. U kombinaciji sa *GSON* bibliotekom koja JSON objekte serijalizira u Java objekte i obratno, Retrofit značajno smanjuje količinu kôda potrebnog za rad sa web servisima, a isto tako samu organizaciju kôda čini preglednijom [38].

Slika 67 prikazuje kako metoda `Call` iz Retrofita poziva web servis temeljem definirane (ranije spomenute) rute. Po dobivanju odgovora, koristi GSON biblioteku koja mehanizmom anotiranja serijalizira JSON objekte u Java objekte temeljem klasa definiranih u projektu. Dakle, odgovori web servisa, trebaju biti opisani Java klasama koje sadrže samo atribute i konstruktor analogan atributima JSON odgovora od web servisa. Takve klase još se nazivaju *POJO* klase (*engl. plain old java object*).



Slika 67. *Shema primjene kombinacije biblioteka Retrofit i GSON za konzumiranje web servisa*

U narednom primjenu, napraviti ćemo poziv web servisa za dohvaćanje vremenske prognoze s otvorenog web servisa `OpenWeatherMap.org`. Servis prema danim podacima (ime mjesta, šifra mjesta, geografska visina i širina) dohvaća prognozu vremena.

Kao prvi korak prema dohvaćanju tih podataka, u Android projekt je potrebno dodati reference na Retrofit i GSON biblioteke u `build.gradle` (Module: app):

```
// retrofit, okhttp, gson dependencies
compile 'com.google.code.gson:gson:2.4'
compile 'com.squareup.retrofit:retrofit:2.0.0-beta2'
compile 'com.squareup.retrofit:converter-gson:2.0.0-beta2'
compile 'com.squareup.okhttp:okhttp:2.7.0'
```

Uz navedene biblioteke, uključena je i `okhttp` biblioteka, putem koje Retrofit više neće biti ovisan o lokalnom HTTP servisu Android operacijskog sustava, već će koristiti implementaciju treće strane.

Nakon toga, potrebno je analizirati odgovor web servisa, kako bi se mogle napraviti POJO klase. Puni opis aplikacijskih programskih sučelja za korištenje OpenWeatherMap web servisa dostupan je na stranici <http://openweathermap.org/api>, no za sad promotrimo sam poziv web servisa:

<http://api.openweathermap.org/data/2.5/weather?q=Varazdin&appid=SIFRA&units=metric>

Navedeni poziv sastoji se od elemenata prikazanih u tablici (Tablica 15). Uočite da je prvi argument metoda označen sa `?`, a svaki argument nakon toga je dodan sa znakom `&` u obliku parova ključ-vrijednost.

*Tablica 15. Elementi URI-a za poziv web servisa*

http	Protokol za transport podataka
api.openweathermap.org	Korijenska adresa do servisa (URL)
data	Putanja do web aplikacije koja implementira web servis
2.5	Verzija web aplikacije
weather	Naziv usluge (metode koja se koristi)
<b>Argumenti:</b>	
q	Upit lokacije za koju se dohvaća prognoza vremena
appid	Identifikator naše aplikacije (potrebno se registrirati za besplatno korištenje web servisa)
units	Jedinice, imperijalne ili metrički sustav.

Odgovor web servisa za zadani upit u JSON formatu je:

```
{
  "coord":{
    "lon":16.34,
    "lat":46.3
  },
  "weather":[
    {
      "id":802,
      "main":"Clouds",
      "description":"scattered clouds",
      "icon":"03d"
    }
  ],
  "base":"stations",
  "main":{
    "temp":24.55,
    "pressure":1017,
    "humidity":57,
```

```

    "temp_min":22.78,
    "temp_max":27
  },
  "visibility":10000,
  "wind":{
    "speed":2.1,
    "deg":60
  },
  "clouds":{
    "all":40
  },
  "dt":1469452471,
  "sys":{
    "type":1,
    "id":5883,
    "message":0.0276,
    "country":"HR",
    "sunrise":1469417335,
    "sunset":1469471562
  },
  "id":3188383,
  "name":"Varazdin",
  "cod":200

```

Za osnovno razumijevanje JSONa potrebno je znati:

- a) *JSON objekt je sadržan unutar vitičastih zagrada { ... },*
- b) *Polja su zapisana unutar uglatih zagrada [ ... ],*
- c) *Vrijednosti su zapisane u ključ-vrijednost parovima odvojenim dvotočjem i unutar navodnika "ključ": "vrijednost".*

Imajući to na umu, iz navedenog primjera odgovora web servisa, može se uočiti da se radi o jednom objektu sa 12 korijenskih atributa, redom: coord, weather, base, main, visibility, wind, clouds, dt, sys, id, name i cod. Nadalje, atribut coord kao vrijednost ima novi objekt koji ima dva svojstva, lat i lon. Objekt weather kao vrijednost ima polje sa samo jednim objektom, koji ima četiri svojstva id, main, description i icon. main kao vrijednost ima vrijednost sa pet atributa, wind sa dva itd.

Kako bi GSON mogao serijalizirati primljeni objekt u objekt Java klase, potrebno je napraviti istu klasnu strukturu, pri čemu klasa odgovora ima svih ranije dvanaest navedenih svojstava, svojstvo coord kao atribut sadrži novi objekt, weather polje objekata itd. U konačnici, bilo bi potrebno sedam klasa: Forecast (sam odgovor sa 12 atributa), Coord, Weather, Main, Wind, Clouds i Sys<sup>16</sup>. No, ukoliko nas ne zanima čitav odgovor, moguće je napraviti klase samo za attribute koji su nam od interesa. U ovom slučaju, nas zanimaju samo konkretni atributi vezani uz prognozu, odnosno atributi sadržani u svojstvu odgovora main. Stoga, bit će potrebne dvije POJO klase kojima ćemo dodijeliti prefiks Ws (webservice), WsForecast i WsMain. Te dvije klase moraju sadržavati sva svojstva, jednaka kao i njihovi ekvivalenti iz JSON odgovora.

<sup>16</sup> Kako su to vrlo često samo trivijalne POJO klase, neki alati omogućuju njihovo generiranje uz predočenje JSON odgovora.

U projektu, treba kreirati paket i nazvati ga `webservice`, a unutar njega još jedan paket `wsentities`, te kreirati klase `WsForecast` i `WsMain`, zajedno sa mutatorskim klasama prema JSON odgovoru<sup>17</sup>.

`WsMain` (nakon napisanih atributa, koristiti prečac za generiranje mutatorskih metoda „Getters and Setters“):

```
public class WsMain {
    public float temp;
    public float temp_min;
    public float temp_max;
    public float humidity;
    public float pressure;

    public float getTemp() {
        return temp;
    }

    public void setTemp(float temp) {
        this.temp = temp;
    }

    public float getPressure() {
        return pressure;
    }

    public void setPressure(float pressure) {
        this.pressure = pressure;
    }

    public float getTemp_min() {
        return temp_min;
    }

    public void setTemp_min(float temp_min) {
        this.temp_min = temp_min;
    }

    public float getTemp_max() {
        return temp_max;
    }

    public void setTemp_max(float temp_max) {
        this.temp_max = temp_max;
    }

    public float getHumidity() {
        return humidity;
    }

    public void setHumidity(float humidity) {
        this.humidity = humidity;
    }
}
```

<sup>17</sup> Konvencija nad konfiguracijom (engl. Convention over configuration), što znači da GSON prema konvenciji traži samo attribute koji postoje, dok druge zanemaruje.

WsForecast kod svojstva main kao tip podataka ima definirani objekt klase WsMain, što će GSON povezati sa odgovarajućim JSON objektom odgovora:

```
public class WsForecast {
    // convention over configuration
    public String name;
    public int id;
    public WsMain main;

    public WsMain getMainData() {
        return main;
    }

    public void setMainData(WsMain main) {
        this.main = main;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Sljedeći korak, u paketu webservice treba definirati programsko sučelje koje će sadržavati rute (putanje) do metoda web servisa zajedno sa njihovim parametrima. Trenutno, koristiti ćemo samo jednu metodu i to za dohvat ranije prikazanog JSON-a. Programsko sučelje nazvati WsCaller:

```
public interface WsCaller {
    @GET("/data/2.5/weather")
    Call<WsForecast> getForecast(@Query("q") String city, @Query("units") String
    units, @Query("appid") String appid);
}
```

U navedenom primjeru se vidi da će se u Java kôdu unutar Android aplikacije metoda za poziv web servisa zvati getForecast, a ista sadrži tri anotirana parametra. Anotacije parametara govore za koje argumente web servisa prikazanih u tablici (Tablica 15) će se parametri Java programskog sučelja vezati. Isto tako, čitava metoda anotirana je putanjom do glavne metode web servisa, zajedno sa definicijom HTTP metode koja će se koristiti kod slanja zahtjeva, u ovom slučaju to je GET. Metoda getForecast vraća Call objekt sa listom WsForecast objekata.

Preostala implementacija zahtjeva novu klasu čiji će se objekti koristiti za asinkroni, ne blokirajući poziv web servisa. Novu klasu nazvati `WeatherWebService` i smjestiti je unutar paketa `webservice`. Klasa će imati sljedeći sadržaj:

```
public class WeatherWebService {
    public static final String API_ID = "REGISTRACIJSKI_KLJUC";
    public static final String BASE_URL = "http://api.openweathermap.org/";

    private ArrayList<WsForecast> weatherForecast;

    public void getWeather(){
        weatherForecast = new ArrayList<WsForecast>();
    }
}
```

Metoda `getWeather` treba implementirati sam poziv web servisa i slušatelj događaja (*engl. listener*) koji će se aktivirati kada stigne odgovor i napraviti serijalizaciju odgovora u lokalne POJO klase.

Sadržaj metode `getWeather`:

```
public void getWeather(){
    weatherForecast = new ArrayList<WsForecast>();

    // kreiranje retrofit objekta i vezivanje za osnovni URL te GSON biblioteku
    Retrofit retrofit = new Retrofit.Builder()
        .baseUrl(BASE_URL)
        .addConverterFactory(GsonConverterFactory.create())
        .build();

    // generiranje metode web servisa prema WsCaller sučelju
    WsCaller serviceCaller = retrofit.create(WsCaller.class);

    // asinkroni poziv web servisa, zajedno sa parametrima metode
    Call<WsForecast> call = serviceCaller.getForecast("Varazdin,HR", "metric",
API_ID);

    // implementacija događaja nakon stizanja odgovora
    call.enqueue(new Callback<WsForecast>() {
        // ako je poziv uspješan
        @Override
        public void onResponse(Response<WsForecast> response, Retrofit retrofit) {
            if (response.isSuccess()) {
                System.out.println("Web service call successful!");
                // dohvati atribut "body" od klase odgovora, koji najčešće sadrži
JSON
                weatherForecast.add(response.body());
                // za 0-ti dohvaćeni element liste ispiši ime i temperaturu grada:
                System.out.println("Grad: " + weatherForecast.get(0).getName());
                System.out.println("Temperatura: " +
```

```

weatherForecast.get(0).getMainData().getTemp();

    }

}

// ako poziv nije uspješan
@Override
public void onFailure(Throwable t) {
    System.out.println("Web service call failed!");
}

});
}

```

Kao što je vidljivo u komentarima navedenog primjera, najprije se instancira objekt odgovora `weatherForecast` lista, nakon čega slijedi kreiranje Retrofit objekta kojemu se kao parametar proslijeđuje korijenski URL web servisa te GSON objekt za kasniju serijalizaciju JSON-a. Potom, Retrofit generira implementaciju metoda programskog sučelja `WsCaller` prema instrukcijama zapisanim u anotacijama. Na kraju, kreira se `call` objekt nad kojim se metodom `enqueue` asinkrono poziva web servis. Primjenom anonimne metode definira se pozivna metoda (engl. `callback`, metoda koja se poziva iniciranjem nekog događaja). U našem slučaju to je dolazak odgovora od web servisa, što za posljedicu može imati dva ishoda, uspjeh (metoda `onResponse`) ili pogrešku (`onFailure`). Također, metoda `onResponse`, provjerava da li je HTTP rezultirao uspjehom ili neuspjehom. Ako su podaci uspješno stigli, u objekt odgovora `weatherForecast` dodaje se tijelo dohvaćene HTTP poruke koja sadrži JSON. Konačno, nakon svega moguće je ispisati vrijednosti. U slučaju da poziv web servisa nije uspio, o tom će se ispisati poruka.

Prije nego se navedeni kôd može pokrenuti i testirati, potrebno je u `AndroidManifest.xml` dodati dozvolu korištenja Interneta, a na odgovarajućoj aktivnosti pokrenuti sam postupak dohvaćanja web servisa. Dozvola se dodaje prije oznake `application`:

```

<!-- korištenje Interneta -->
<uses-permission android:name="android.permission.INTERNET" />

```

A sam poziv servisa iz aktivnosti čine dvije, vrlo jednostavne linije kôda:

```

// poziv web servisa
WeatherWebService weatherWebService = new WeatherWebService();
weatherWebService.getWeather();

```

A to će u Logcat konzoli Android Studia rezultirati sa:

```

07-25 17:03:10.138 10364-10364/hr.heureka.cookbook I/System.out: Web service call
successful!
07-25 17:03:10.138 10364-10364/hr.heureka.cookbook I/System.out: Grad:Varazdin
07-25 17:03:10.138 10364-10364/hr.heureka.cookbook I/System.out: Temperatura: 26.0

```

Što se obzirom na temperaturu u uredu autora ovog priručnika čini kao točan odgovor za kraj mjeseca srpnja 😊.

*Naravno, trenutnu arhitekturu trebalo bi prilagoditi na način da se odgovor ispisuje na korisniku prihvatljiv način, no to je moguće primjenom znanja kroz ranije usvojena poglavlja.*



## 4.5 Pitanja za provjeru znanja

1. Koje mogućnosti za pohranu podataka nudi Android?
2. Što su ključ-vrijednost parovi?
3. Pojasnite razliku između unutarnje i vanjske memorije.
4. Koja je putanja u datotečnom sustavu Androida do datoteka neke aplikacije?
5. Što je Android Debug Bridge?
6. Čemu služe normalne forme?
7. Kako se rješava problem veze M:N?
8. Što je omogućuje ORM sustav?
9. Samostalno definirajte bazu podataka s minimalno dva glavna entiteta za neki vama dobro poznat sustav.
10. Na koji način Android pokreće ažuriranje strukture tablice u bazi podataka?
11. Gdje se u Android projektu postavlja naziv baze podataka?
12. Čemu služe web servisi?
13. U čemu biblioteka Retrofit olakšava rad s web servisima?
14. Što je GSON?
15. Pojasnite razliku između SOAP i REST web servisa.
16. Što su POJO klase i čemu služe?

## 4.6 Resursi za samostalan rad

- ✓ **Robert Manger:** Baze podataka - Drugo izdanje  
*Sveučilište u Zagrebu, Prirodoslovno matematički fakultet, Matematički odsjek, 2011*
- ✓ **Mladen Vedriš,** Uvod u baze podataka  
*Tečajevi srca, Sveučilište u Zagrebu, Sveučilišni računski centar*
- ✓ Ostale baze podataka, LevelDB, Couchbase Lite, BerkeleyDB, Realm
- ✓ Ostali ORM sustavi, SugarORM, Sprinkles, DBFlow
- ✓ Android aplikacije za učenje SQL-a: Practice and Learn SQL, Learn SQL  
<https://play.google.com/store/apps/details?id=com.knowledify.sqlush&hl=en>  
<https://play.google.com/store/apps/details?id=com.sololearn.sql&hl=en>



## 5 PRIMJER RAZVOJA MOBILNE APLIKACIJE

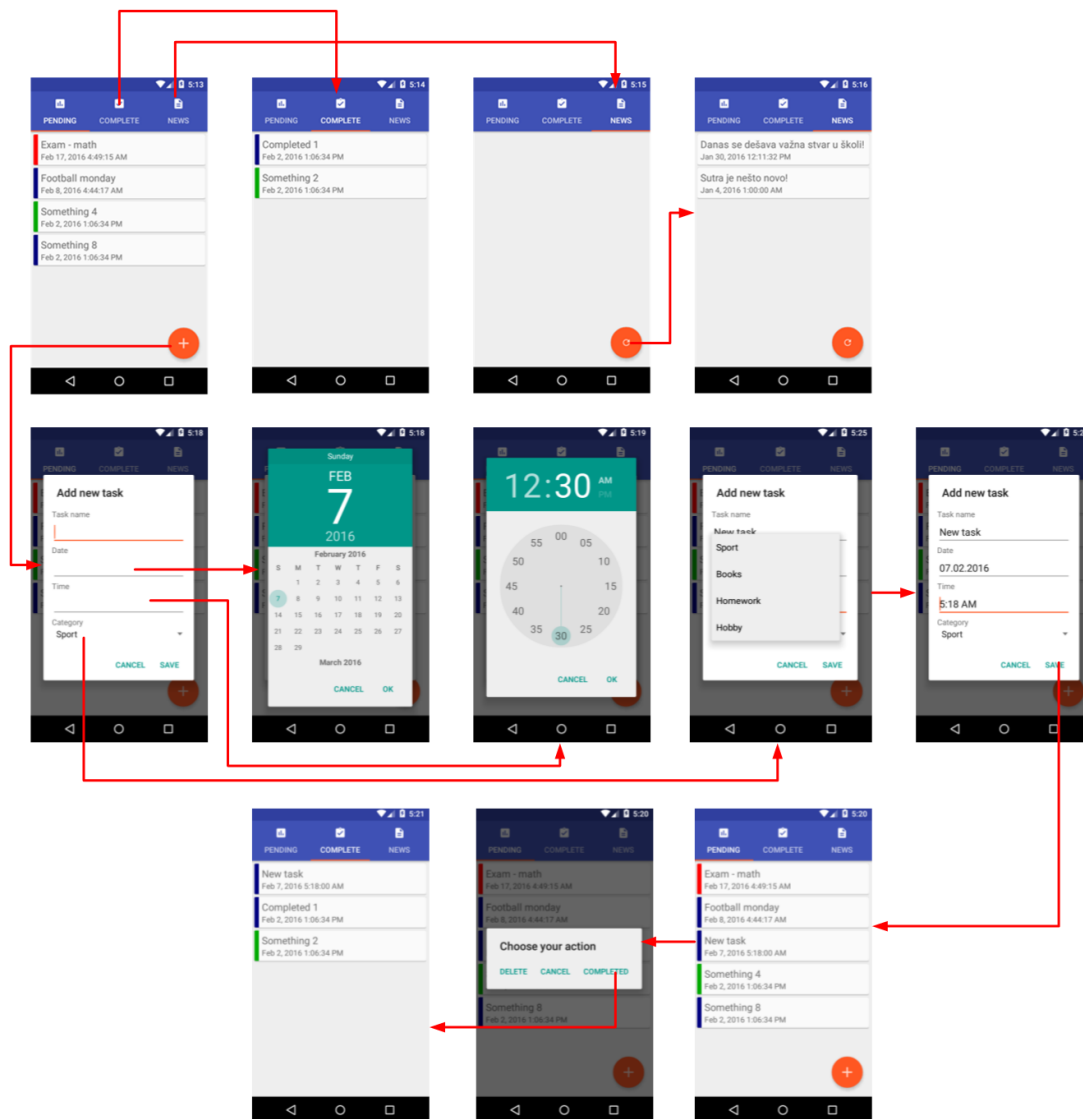
*Ovo poglavlje gradi na teorijskim osnovama prikazanim u svim prethodnim poglavljima, te na praktičan i razumljiv način, metodom direktnog usmjeravanja vodi kroz cjelokupni proces razvoja mobilne aplikacije Memento. Cilj poglavlja je kroz praktičan primjer prikazati koncepte razvoja mobilne aplikacije koja omogućuje korisniku evidenciju i kategorizaciju osobnih nastavnih, vannastavnih i privatnih obveza te pregled školskih vijesti koje se dohvaćaju sa mrežnog servisa. Poglavlje razrađuje ideju Memento aplikacije, prikazuje kreiranje projekta, izradu programske logike i pogleda, te rad s lokalnim i udaljenim podacima.*

### SADRŽAJ POGLAVLJA

Memento.....	120
Kreiranje projekta.....	121
Izrada pogleda .....	124
Izrada entitetnih klasa .....	142
Unos i prikaz podataka – rad s dijalozima i fragmentima pogleda .....	148
Mobilna baza podataka .....	154
Korištenje web servisa.....	169
Dodatni resursi .....	180

## 5.1 Memento

Pojam *memento* dolazi s latinskog jezika i znači prisjećati se, odnosno pamtiti što je prikladan naziv za mobilnu aplikaciju čiji se razvoj prati kroz ovo poglavlje. Naime, trenutno poglavlje daje potpuni pregled, od početka do kraja razvoja mobilne aplikacije koja omogućuje unos i pohranu osobnih zadataka prema različitim kategorijama te praćenje ispunjenosti istih. Nadalje, ona omogućuje svakom učeniku da za svoju školu dobiva trenutno aktualne novosti. Kroz ovaj proces razvoja biti će govora o kreiranju početnog projekta, izradi Android pogleda, izradi entitetnih klasa, unosu i prikazu podataka, radu s dijalogima te radu s mobilnom bazom podataka i web servisima. Slika 68 prikazuje izgled, odnosno navigacijske putanje kroz Memento.



Slika 68. Moguće navigacijske putanje kroz mobilnu aplikaciju Memento

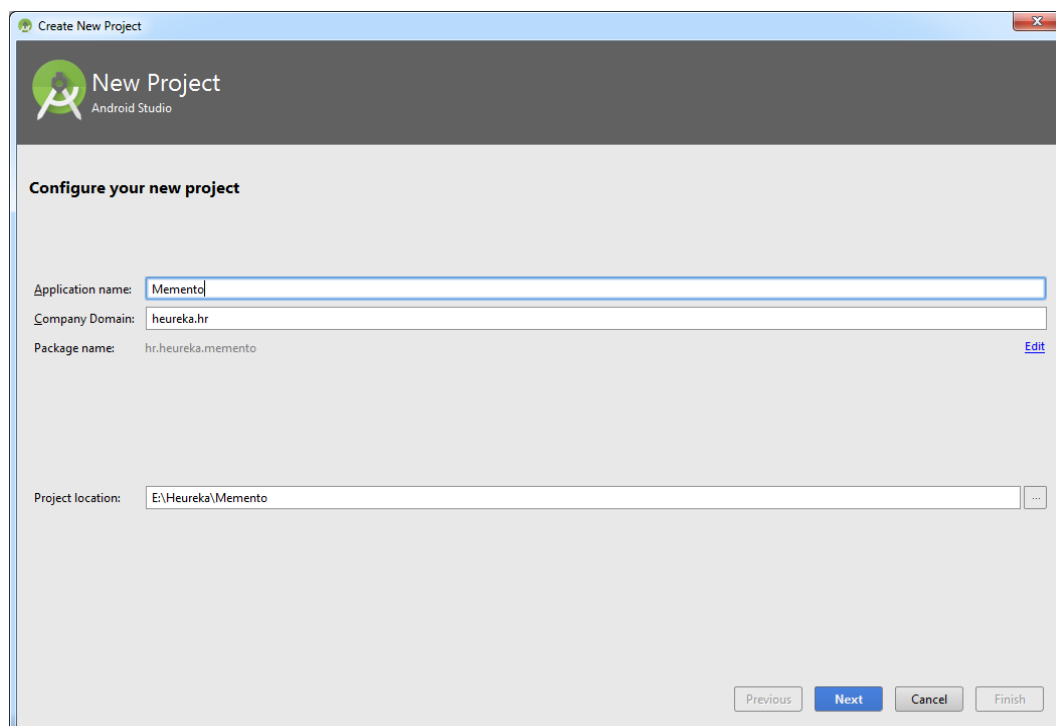
## 5.2 Kreiranje projekta

Prvi korak razvoja svake mobilne aplikacije je kreiranje projekta i odabir početnih postavki. Nakon pokretanja *Android Studija* potrebno je u „File“ odabrati opciju „New, New Project...“, nakon čega se inicijalizira dijalog koji korisnika vodi kroz kreiranje projekta.

Kao što se vidi na slici Slika 69, potrebno je navesti naziv aplikacije (engl. Application name), naziv tvrtke odnosno organizacije kojoj razvojni inženjer pripada (Company Domain), što ujedno definira i naziv korijenskog paketa mobilne aplikacije. Opcionalno, korisnik može odabrati lokaciju projekta na koju će se fizički smjestiti svi artefakti projekta (tekstualne datoteke, izvršne datoteke, slike itd.). Podatke je potrebno popuniti prema sadržaju tablice (Tablica 16), nakon čega treba odabrati opciju „Next“.

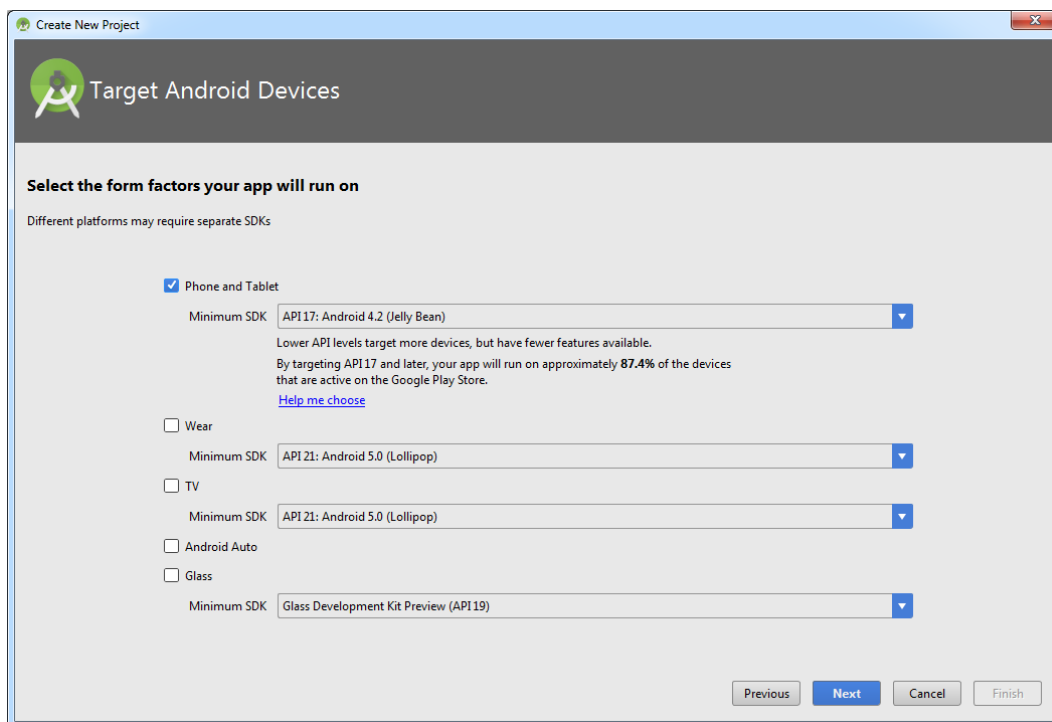
*Tablica 16. Početni podaci o projektu*

Koncept	Vrijednost
Application name:	Memento
Company domain:	heureka.hr
Project location:	<particija>:\Heureka\Memento



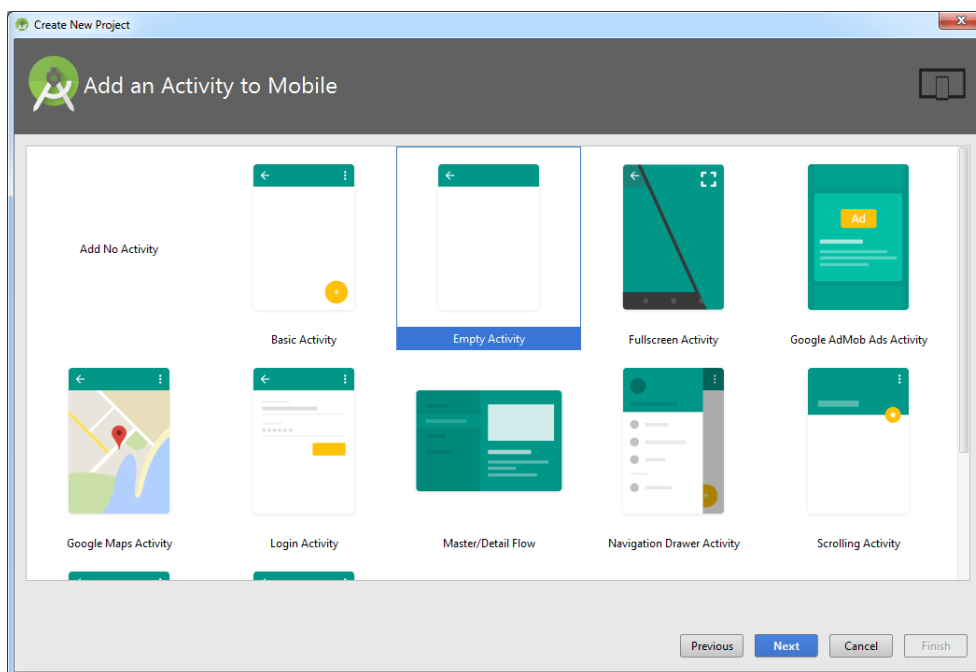
*Slika 69. Kreiranje projekta, dijalog za definiranje naziva projekta*

U sljedećem dijalogu odabire se uređaj za kojeg je mobilna aplikacija predviđena kao i najmanja verzija razvojnog paketa (engl. SDK – software development kit), što je prikazano na slici. Odabrati opciju „Phone and Tablet“, a za minimalnu verziju razvojnog paketa odabrati „API17: Android 4.2 (Jelly Bean)“ što bi prema prikazanom trebalo podržavati nešto više oko 87% uređaja na tržištu. Po uspješnom odabiru treba nastaviti s kreiranjem projekta pritiskom na „Next“.



Slika 70. Odabir razvojnog paketa

Sljedeći dijalog omogućuje korisniku odabir prve aktivnosti koje će kreirati AndroidStudio nakon završetka projekta kreiranja novog projekta. Među navedenim opcijama, kao na slici (Slika 71) odabrati opciju „Empty Activity“, što će kreirati praznu aktivnost bez bilo kakvih elemenata grafičkog korisničkog sučelja (engl. GUI, graphical user interface).

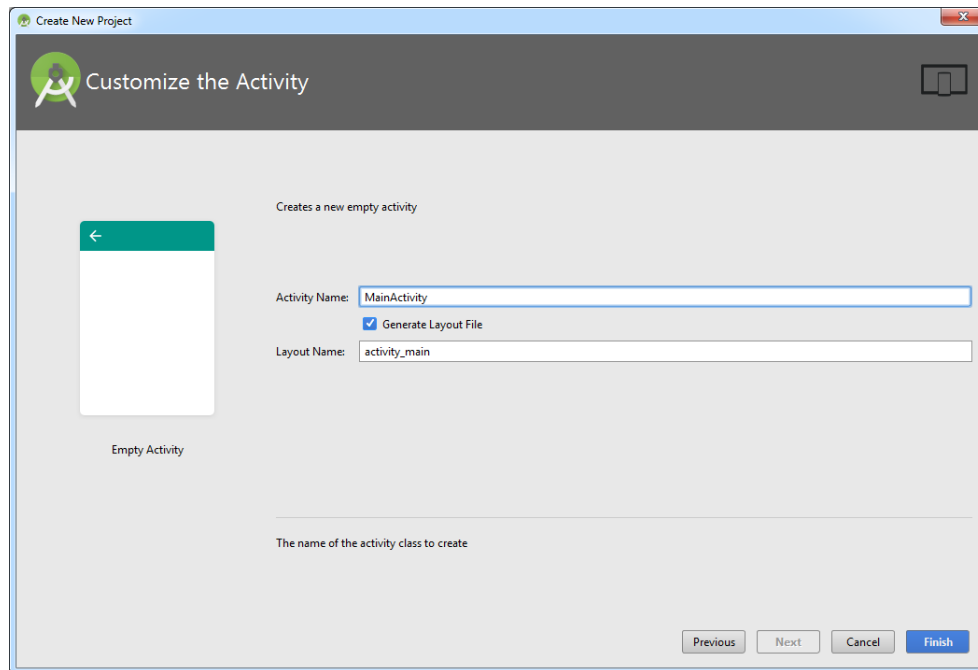


Slika 71. Odabir vrste početne aktivnosti

Posljednji dijalog u procesu kreiranja projekta traži korisnika unos naziva početne aktivnosti te pripadajućeg pogleda, odnosno grafičke reprezentacije. Prema tablici (Tablica 17) popuniti „Activity Name“ i „Layout Name“ što daje sadržaj prikazan na slici (Slika 72). Nakon unosa odabrati opciju „Finish“.

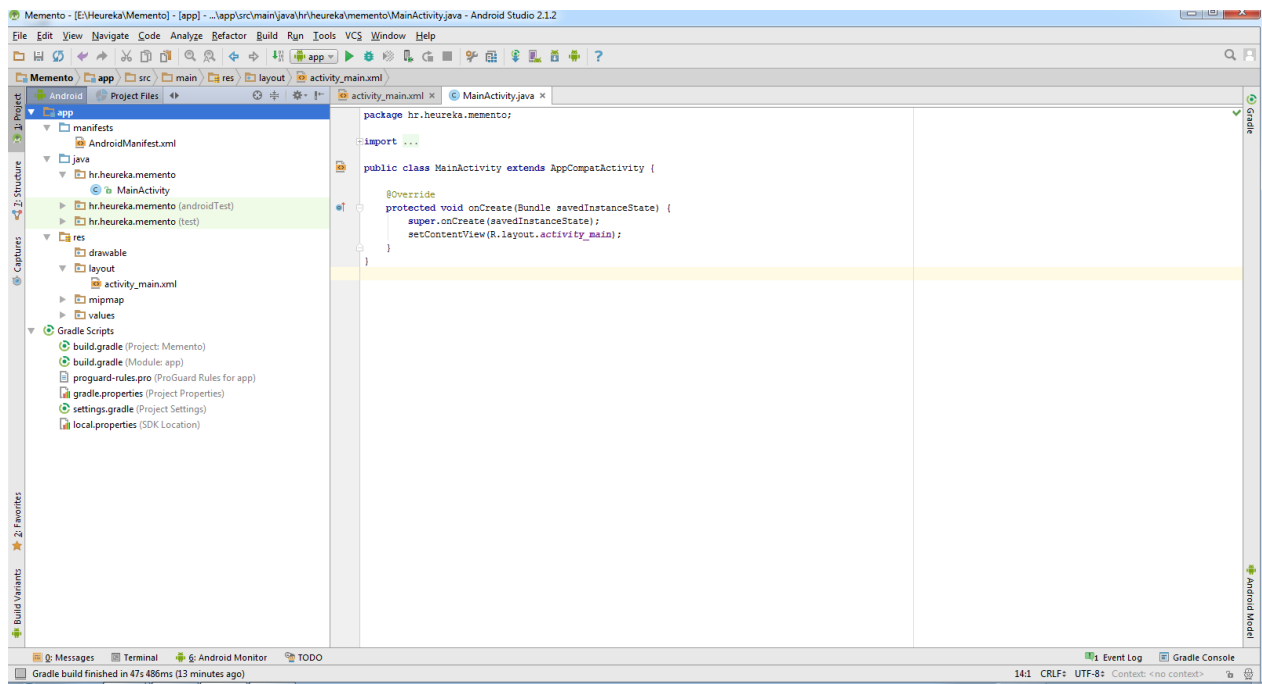
*Tablica 17. Kreiranje projekta, odabir imena glavne aktivnosti*

Vrsta	Naziv
Activity name:	MainActivity
Layout name:	activity_main



*Slika 72. Odabir naziva glavne aktivnosti*

Odabirom opcije „Finish“ proces kreiranja novog projekta je završen, a rezultat je vidljiv na slici (Slika 73), razvojno okruženje sa prikazom strukture projekta te kôdom početne aktivnosti *MainActivity.java*.



Slika 73. Uspješno kreiran projekt

## 5.3 Izrada pogleda

Kako će mobilna aplikacija, odnosno glavni pogled (activity\_main.xml) sadržavati gumb koji lebdi nad sadržajem fragmenta (*engl. FAB, floating action button*) korijenski grafički element pogleda treba biti CoordinatorLayout, koji omogućuje koordinaciju podelemenata specifičnog tipa, odnosno animiranih grafičkih elemenata iz “Material” specifikacije [39] [40].

### 5.3.1 Priprema dizajna glavne aktivnosti za tabove

Unutar prethodnog pogleda potrebno je dodati tri podelementa, AppBarLayout, TabLayout i ViewPager. AppBarLayout je pogled koji dopunjuje klasični horizontalni izbornik uobičajeno na vrhu Android mobilne aplikacije sa dodatnim funkcionalnostima poput animiranog pomicanja sadržaja [41]. Kako će Memento sadržavati 3 horizontalno postavljena fragmenta kojima se pristupa pomicanjem ulijevo ili udesno, koristi se TabLayout za prikaz naziva tabova<sup>18</sup> te pripadajućeg grafičkog elementa koji prikazuje trenutnu poziciju te ViewPager koji omogućuje horizontalne tranzicije među fragmentima (koji predstavljaju sadržaj tabova) [42] [43].

U build.gradle (Module:app) dodati zavisnosti vezane uz komponente Material dizajna, što je potrebno za izmjenu zadane alatne trake tako da umjesto naslova mobilne aplikacije samo tabove:

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.4.0'
    // opcije dizajna
    compile 'com.android.support:design:23.4.0'
}
```

<sup>18</sup> U ovom priručniku, umjesto riječi „kartica“ za prikaz horizontalno straničenog izbornika koristimo anglizam tab.



activity\_main.xml:

```
<android.support.design.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
android:layout_width="match_parent"
android:layout_height="match_parent">

<!-- prostor u kojem se nalazi alatna traka (Toolbar) -->
<android.support.design.widget.AppBarLayout
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar">

<!-- prostor u kojem se prikazuju stranice (Tabs) -->
<android.support.design.widget.TabLayout
android:id="@+id/tabs"
android:layout_width="match_parent"
android:layout_height="72dp"
app:tabMode="fixed"
app:tabGravity="fill"/>
</android.support.design.widget.AppBarLayout>

<!-- prostor u kojem se prikazuju sadržaji pojedinih stranica -->
<android.support.v4.view.ViewPager
android:id="@+id/viewpager"
android:layout_width="match_parent"
android:layout_height="match_parent"
app:layout_behavior="@string/appbar_scrolling_view_behavior" />
</android.support.design.widget.CoordinatorLayout>
```

Radi testiranja, u procesu razvoja preporučljivo je često (odnosno nakon dodavanja funkcionalnosti ili izmjena) pokrenuti aplikaciju da se vide promjene i eventualno uoče pogreške. Pokretanjem Mementa u trenutnom stadiju razvoja dobiva se rezultat prikazan na slici (Slika 74), pri čemu je tekst koji prikazuje korištene poglede označen crvenom bojom.



*Slika 74. Izgled glavne aktivnosti nakon dodavanja AppBarLayout-a, TabLayout-a i ViewPager-a*

Kako je vidljivo iz prethodne slike (Slika 74), još uvijek nije uklonjen naslov aplikacije. Da bi se to postiglo, potrebno je kreirati novi dizajn, koji se sastoji od *stila* i *boja*, definiranih redom u `styles.xml` i `colors.xml`.

U `styles.xml` definirati stil po imenu „MyMaterialTheme“, definirati da se ne prikazuje naslov te definirati osnovne boje u mobilnoj aplikaciji:

```
<resources>
    <style name="MyMaterialTheme" parent="MyMaterialTheme.Base"></style>
    <style name="MyMaterialTheme.Base"
parent="Theme.AppCompat.Light.DarkActionBar">
        <item name="windowNoTitle">true</item>
        <item name="windowActionBar">false</item>
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
        <item name="android:textColorPrimary">@color/textColorPrimary</item>
        <item name="android:textColorSecondary">@color/textColorSecondary</item>
    </style>
</resources>
```

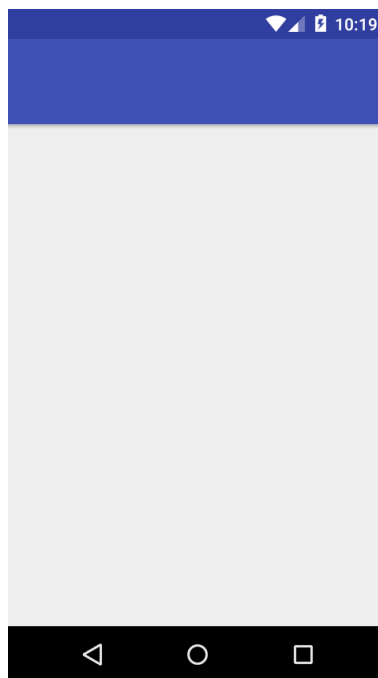
Dodati boje koje nedostaju u `colors.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF5722</color>
    <color name="textColorPrimary">#212121</color>
    <color name="textColorSecondary">#727272</color>
</resources>
```

A da bi promjene bile vidljive, u `AndroidManifest.xml` datoteci potrebno je specificirati korištenu temu izmjenom svojstva `android:theme` u:

```
android:theme="@style/MyMaterialTheme"
```

Ponovno pokretanje Mementa prikazano je na slici (Slika 75). Uočljivo je da je uklonjen naslov aplikacije, `TabLayout` nije popunjen jer još nema dodanih tabova, a sadržaj `ViewPager` je također prazan. Sljedeći korak je dodavanje tabova, njihovih naslova te ikona. Memento prikazuje tri taba, a) tab za prikaz unesenih zadataka koji nisu obavljeni, b) tab za prikaz uspješno obavljenih zadataka i c) tab za prikaz aktualnih novosti škole.



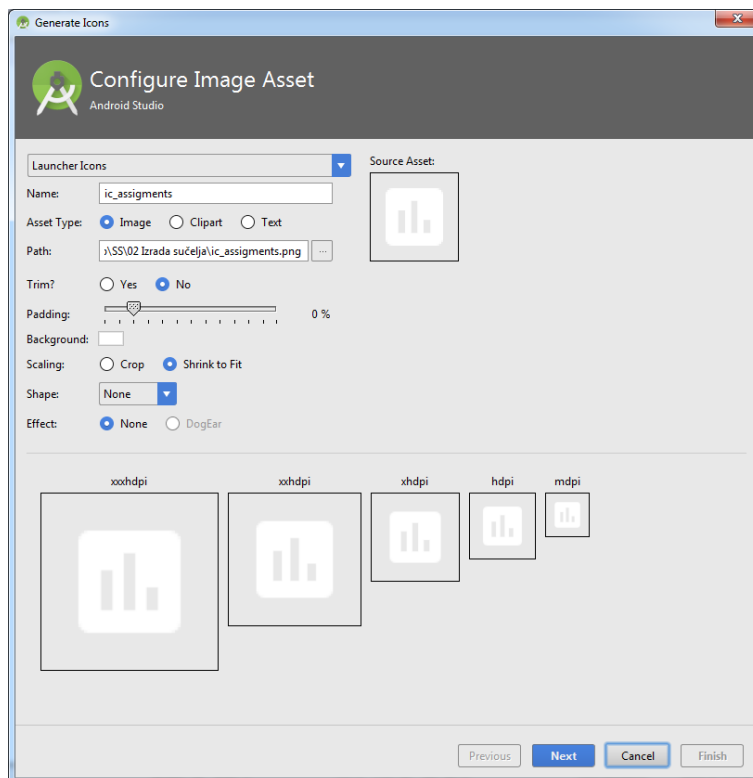
*Slika 75. Rezultat promjene teme dizajna*

### 5.3.2 Uvoz potrebnih ikona u projekt

Naslovi pojedinih tabova te njihovih ikona će biti postavljeni programski, što znači da je grafičke elemente potrebno dohvatiti u programskom kôdu, a ne u XML-u pogleda. Za pojednostavljenje korištenja grafičkih elemenata u kôdu u ovom projektu koristi se biblioteka treće strane, `ButterKnife` koju je potrebno dodati u projekt u `build.gradle (Module:app)`:

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.4.0'
    // opcije dizajna
    compile 'com.android.support:design:23.4.0'
    // ButterKnife
    compile 'com.jakewharton:butterknife:7.0.1'
}
```

Sljedeće što je potrebno jest u AndroidStudio projekt učitati ikone za prikaz na tabovima. Desnim klikom na `/res` projektnog stabla odabrati opciju „New, Image asset...” nakon čega se otvara dijalog prikazan na slici (Slika 76).



*Slika 76. Dodavanje vektorskog resursa (slike) u projekt*

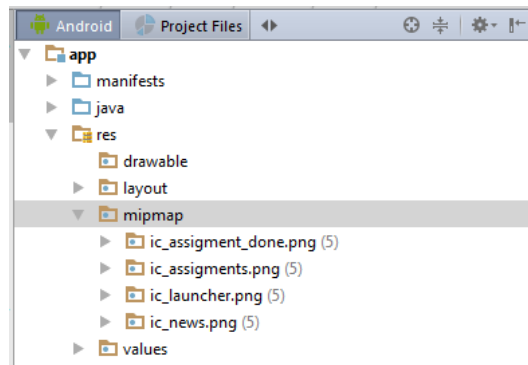
U prethodnom dijalogu, iz padajućeg izbornika treba odabrati „Launcher icons“, a ostatak popuniti prema tablici (Tablica 18).

*Tablica 18. Postavke za uvoz nove slike*

Koncept	Vrijednost
Name:	ic_assignments
Asset Type:	activity_main
Path:	Putanja do datoteke
Scaling:	Shrink to fit
Shape:	None
Effect:	None

Sve ostale postavke ostaviti na zadane vrijednosti te odabrati opciju „Next“ i nakon toga „Finish“. Unutar direktorija `res/mipmap` sada je vidljiva upravo dodana ikona.

Na jednaki način kao i za `ic_assignments` ikonu koja će biti vezana uz tab zadataka u tijeku, potrebno je dodati još dvije ikone, `ic_assignments_done` i `ic_news`, za završene zadatke te za novosti dohvaćene s web servisa što će biti vidljivo u stablu projekta kao što je prikazano na slici (Slika 77).



Slika 77. Prikaz dodanih ikona u strukturi projekta

### 5.3.3 Programsko dodavanje tabova i ikona

U MainActivity.java potrebno je dohvatiti TabLayout i ViewPager neposredno prije onCreate metode jednako kao i dohvatiti ikone iz repozitorija resursa projekta sljedećim kôdom (kombinacijom Alt+Enter dodati zavisnosti koje nedostaju):

```
@Bind(R.id.tabs)
TabLayout tabLayout;
@Bind(R.id.viewpager)
ViewPager viewPager;

private int[] imageResIds = {
    R.mipmap.ic_assignments,
    R.mipmap.ic_assignment_done,
    R.mipmap.ic_news,
};
```

Nakon onCreate metode dodati još dvije metode setupTabIcons i setupViewPager. Prva je zadužena za postavljanje ikona prema poziciji dodanih tabova dok je druga nešto kasnije služiti popunjavanju sadržaja pojedinih viewPager elemenata (sadržaj tabova):

```
private void setupTabIcons() {
    tabLayout.getTabAt(0).setIcon(imageResIds[0]);
    tabLayout.getTabAt(1).setIcon(imageResIds[1]);
    tabLayout.getTabAt(2).setIcon(imageResIds[2]);
}

private void setupViewPager(final ViewPager viewPager) {
    // kôd za prikaz pojedinih tabova
}
```

I konačno, u onCreate metodi potrebno je napraviti inicijalizaciju ButterKnife biblioteke, popuniti sadržaj tabova, asociirati ViewPager i TabLayout te definirati da za vrijeme izvođenja ViewPager u memoriji čuva 3 taba.

Unutar metode onCreate dodati:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // aktivacija ButterKnife biblioteke
```

```

ButterKnife.bind(this);
// postaviti sadržaj (popuniti viewPager)
setupViewPager(viewPager);
// uvijek učitaj 3 taba
viewPager.setOffscreenPageLimit(3);
// postavi straničenje
tabLayout.setupWithViewPager(viewPager);
setupTabIcons();
}

```

Trenutno nije moguće pokretati aplikaciju jer najprije treba kreirati pogled (View) za svaki Tab te implementirati podsustav za upravljanje Tabovima.

### 5.3.4 RecyclerView, prvi dio – priprema

RecyclerView pogled će se koristiti za prikaz sadržaja liste zadataka ili dohvaćenih vijesti. Taj pogled poseban je zbog toga što prikazuje samo ograničeni set velikog skupa podataka te samostalno radi kasno učitavanje vidljivih podataka (engl. Lazy Load) [44]. Vrlo često, zbog bolje preglednosti u kombinaciji sa RecyclerView-om koristi se i CardView, koji elementima liste dodaje pozadinu te mogućnost zaokruženih rubova [45].

Za navedene poglede potrebno je dodati zavisnosti u `build.gradle (Module:app)` koji će prikazivati liste aktivnosti:

```

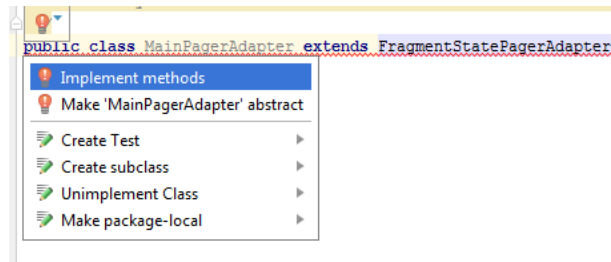
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.4.0'
    // opcije dizajna
    compile 'com.android.support:design:23.4.0'
    // butterknife
    compile 'com.jakewharton:butterknife:7.0.1'
    // recycler
    compile 'com.android.support:recyclerview-v7:23.4.0'
    compile 'com.android.support:cardview-v7:23.4.0'
}

```

Obzirom da ViewPager podatke prikazuje korištenjem Adapter-a, odnosno mehanizma kojim se podaci vežu za grafičke elemente pogleda, potrebno je dodati `FragmentStatePagerAdapter` klasu u projekt [46] [47]. Ta klasa asocira ViewPager sa Fragmentima čime jedan fragment odgovara jednoj stranici, odnosno tabu, a isto tako dinamički popunjava pojedine stranice sadržajem u vremenu izvođenja (također *LazyLoad*).

Ovakav adapter u Memento projektu naziva se `MainPagerAdapter`, te je za dodavanje istog potrebno napraviti novi paket desnim klikom na korijenski paket projekta i odabrati „New“, „Package“. Unutar navedenog paketa kreirati `MainPagerAdapter.java` klasu desnim klikom na novokreirani paket te odabirom „New“, „Java Class“.

Neposredno nakon toga, proširiti klasu super klasom `FragmentStatePagerAdapter` nakon čega će AndroidStudio ponuditi opciju auto-generiranja ne implementiranih metoda. Te metode, moguće je implementirati klikom na ikonu žarulje (vidi Slika 78) koja se pojavi na toj liniji kôda te odabrati opciju „Implement methods“ ili kraće kombinacijom tipaka `Alt+Enter`.



*Slika 78. Automatsko generiranje ne implementiranih naslijeđenih metoda*

Nakon te opcije, potrebno je implementirati dodane metode, `getItem`, `getCount`, `addFragment` i `getPageTitle`, te dvije liste. `mFragmentManager` koja će sadržavati listu `Fragment` (koje prikazuju sadržaj taba) te listu naslova tabova, `mFragmentTitleList`.

MainPagerAdapter.java:

```
public class MainPagerAdapter extends FragmentStatePagerAdapter {

    // popis svih fragmenata (tabova) i njihovaih naslova
    private final List<Fragment> mFragments = new ArrayList<>();
    private final List<String> mFragmentsTitles = new ArrayList<>();

    public MainPagerAdapter(FragmentManager manager, Context ctx) {
        super(manager);
    }

    @Override
    public Fragment getItem(int position) {
        return mFragments.get(position);
    }

    @Override
    public int getCount() {
        return mFragments.size();
    }

    // dodavanje novog taba u listu svih dostupnih
    public void addFragment(Fragment fragment, String title) {
        mFragments.add(fragment);
        mFragmentsTitles.add(title);
    }

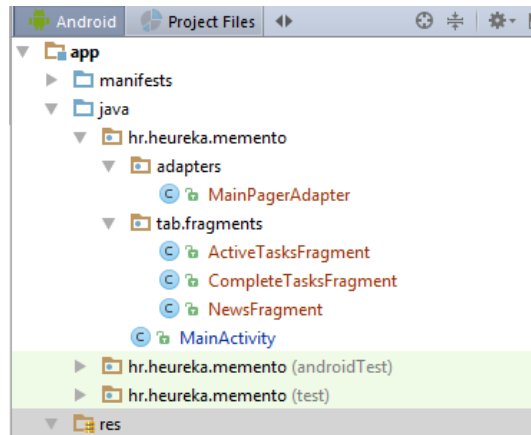
    // dohvaćanje naslova taba prema poziciji
    @Override
    public CharSequence getPageTitle(int position) {
        return mFragmentsTitles.get(position);
    }
}
```

Sljedeći korak je kreirati tri fragmenta za prikaz zadataka i vijesti te njihovih pripadajućih pogleda. Za grupiranje fragmenata koji će se prikazivati u tabovima kreirati novi paket naziva tab.fragments unutar glavnog paketa, a unutar istog kreirati tri Java klase proširene Fragment klasom (iz android.support.v4.app). Klase redom nazvati:

ActiveTasksFragment.java,

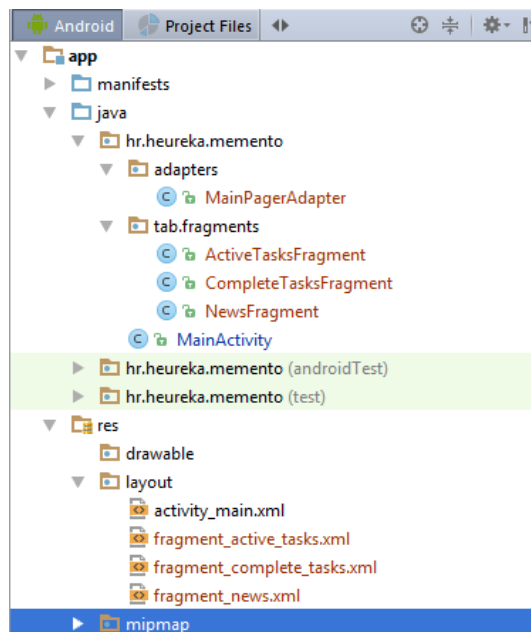
CompleteTasksFragment.java i NewsFragment.java, što daje rezultat prikazan na slici (Slika 79).





*Slika 79. Fragmenti mobilne aplikacije Memento*

Za svaki kreirani fragment, desnim klikom na paket /res „New“, „Resource File“ dodati pogled: `fragment_active_tasks.xml`, `fragment_complete_tasks.xml` i `fragment_news.xml`, što će dati rezultat prikazan na slici .



*Slika 80. Pogledi asocirani s ranije dodanim fragmentima*

Svakom od ranije kreiranih fragmenata, pridružiti odgovarajući pogled putem preopterećenja metode `onCreateView`. Za `ActiveTasksFragment` kôd je sljedeći:

```

public class ActiveTasksFragment extends Fragment {

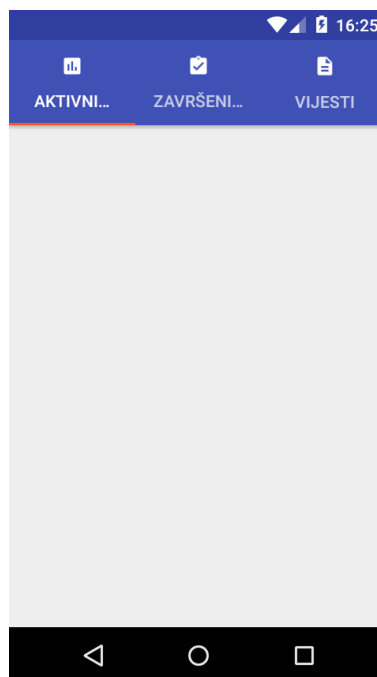
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_active_tasks, container,
false);
        return rootView;
    }
}
    
```

Isto je potrebno napraviti za preostala dva fragmenta, nakon čega je u MainActivity moguće dodati fragmente u ViewPager te inicijalizirati prikaz tabova. Za to je potrebno dopuniti ranije dodanu metodu setupViewPager:

```

private void setupViewPager(final ViewPager viewPager) {
    // kôd za prikaz pojedinih tabova
    final MainPagerAdapter adapter = new
MainPagerAdapter(getSupportFragmentManager(), this);
    adapter.addFragment(new ActiveTasksFragment(), "Aktivni zadaci");
    adapter.addFragment(new CompleteTasksFragment(), "Završeni zadaci");
    adapter.addFragment(new NewsFragment(), "Vijesti");
    viewPager.setAdapter(adapter);
}
    
```

Nakon čega je Memento moguće pokrenuti i vidjeti kreirane tabove što je prikazano na slici (Slika 81).



Slika 81. Memento nakon dodavanja tabova i odgovarajućih fragmenata

### 5.3.5 Postavke jezika i eksternalizacija tekstova

U kôdu metode `setUpViewPager` je uočljivo da tabovi koji su dodani u listu trenutno imaju statički definirane tekstove naslova, što nije dobar pristup programiranju jer se povećava broj točaka koje je u kasnijem održavanju potencijalno potrebno mijenjati. Primjerice, što kada bi Memento trebao podržavati nekoliko različitih svjetskih jezika, koji bi se tekst tada unosio?

Srećom, u razvoju Android aplikacija takav se problem rješava vrlo elegantno, korištenjem `strings.xml` datoteke koja se nalazi unutar `/res/values` paketa, a namijenjena je tome da sadrži sve tekstove koje aplikacija koristi. Dodatno, svi su tekstovi na jednom mjestu, jednostavno se dohvaćaju iz programskog kôda te ih je moguće prevoditi.

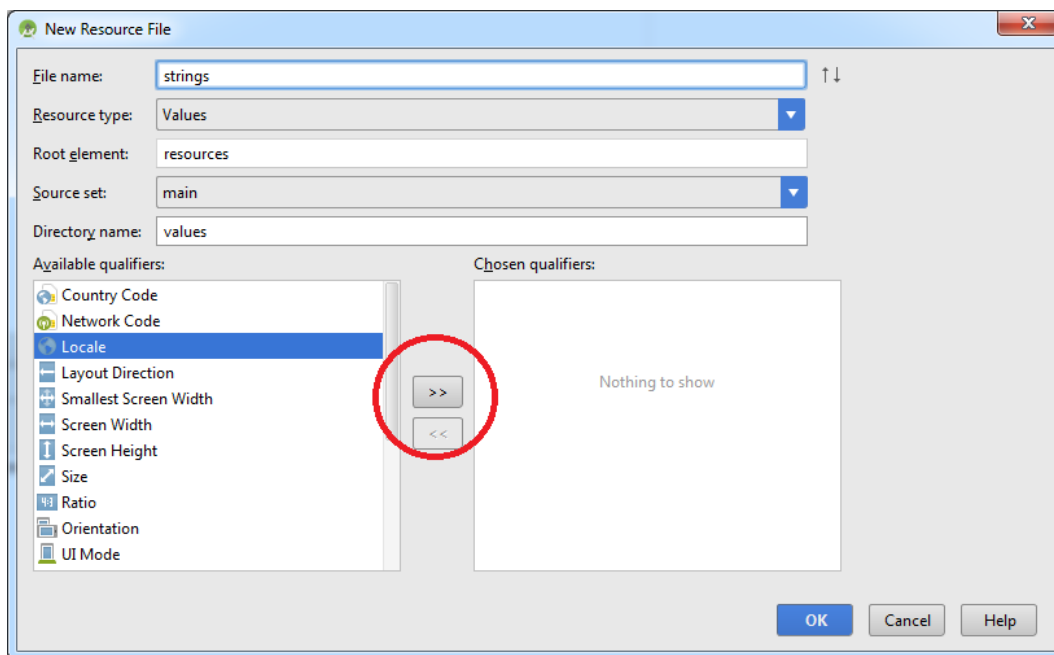
Potrebno je otvoriti navedenu datoteku i dodati sve tekstove koje Memento koristi, na engleskom jeziku:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">Memento</string>
  <string name="tab_title_in_progress">Pending</string>
  <string name="tab_title_complete">Complete</string>
  <string name="tab_title_news">News</string>
  <string name="dialog_title">Add new task</string>
  <string name="dialog_title_long">Choose your action</string>
  <string name="dialog_task_name">Task name</string>
  <string name="dialog_date">Date</string>
  <string name="dialog_time">Time</string>
  <string name="dialog_category">Category</string>
  <string name="save">Save</string>
  <string name="delete">Delete</string>
  <string name="completed">Completed</string>
  <string name="cancel">Cancel</string>
  <string name="loading">Loading</string>
  <string name="loading_message">Fetching data from web service..</string>
</resources>
```

Ovisno o postavkama Android uređaja, odnosno odabranom zadanom jeziku sustava, aplikacija može učitavati `strings.xml` datoteku prema „Locale“ kvalifikatoru. Pomoću kvalifikatora, u Android mobilnim aplikacijama specificira se pojedinačna konfiguracija različitih resursa (slika, tekstova itd.) [48].

Kako bi se dodao prijevod mobilne aplikacije na hrvatski jezik prema ranije definiranim tekstovima, desnim klikom na `/res` odabrati opciju „New“, „Android resource file“ nakon čega se otvara dijalog prikazan na slici (Slika 82).

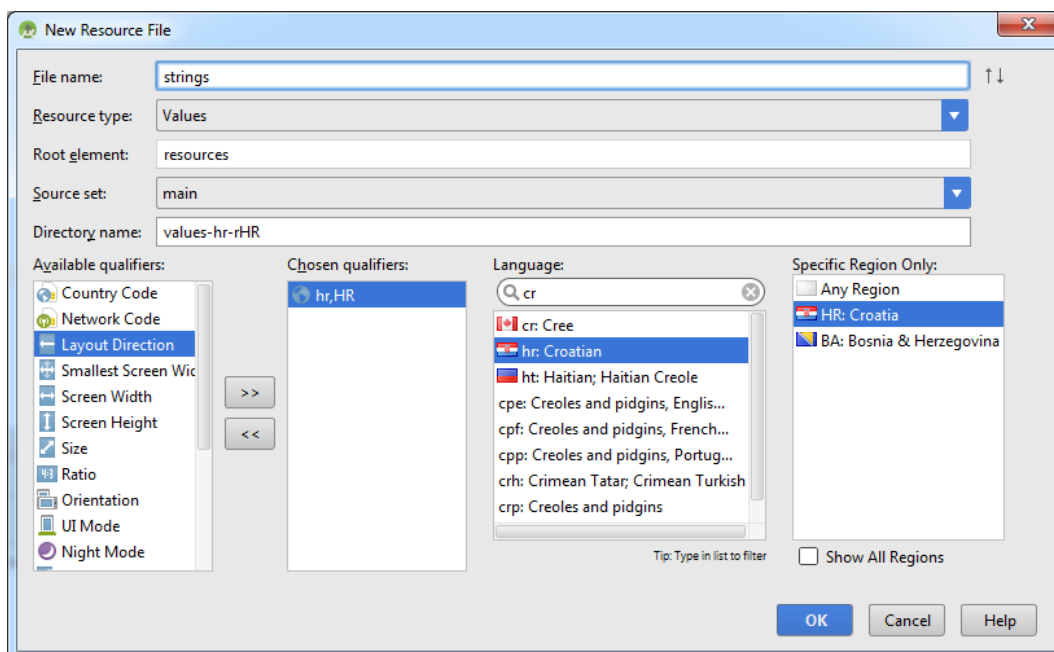
Pod opcijom „File name:“ potrebno je upisati `strings`, kao „Resource type:“ odabrati `Values`, a pod „Available qualifiers“ odabrati „Locale“ nakon čega treba pritisnuti gumb „>>“ (označen crvenom bojom na slici (Slika 82)).



Slika 82. Odabir kvalifikatora za prijevod mobilne aplikacije

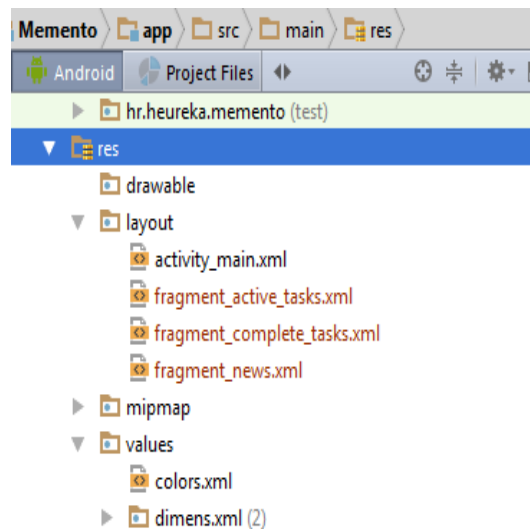
Nakon toga, pojavit će se dodatne opcije pod „Chosen qualifiers“ gdje treba odabrati „HR:Croatia“ što će otvoriti dodatne opcije „Language“ i „Specific Region Only“ prikazane na slici (Slika 83), prema kojoj je potrebno popuniti ostatak dijaloga i na kraju odabrati opciju OK.

Za dodavanje novog prijevoda datoteke `strings.xml` postupak je jednaki, samo je potrebno odabrati drugi jezik. U kôdu koji slijedi koristiti će se tekstovi iz datoteke `strings.xml`, a čiji je prijevod vidljiv nakon promjene jezika Android sustava na uređaju (najčešće u glavnom izborniku, pod sustavskim postavkama opcija „Language & input“).



Slika 83. Odabir jezika i regije za „Location“ kvalifikator

U strukturi projekta, datoteka `strings.xml` se sada prikazuje kao direktorij, gdje je glavna `strings.xml` datoteka za engleski jezik, a svaka druga koja je dodana putem opcije „Location“ kvalifikatora je označena sa skraćenim zapisom kvalifikatora u zagradama, što je prikazano na slici (Slika 84).



Slika 84. Uspješno dodani prijevod `strings.xml` datoteke za hrvatski jezik

Za prevođenje, potrebno je odabrati hrvatsku verziju `strings.xml` datoteke, u nju zalijepiti sadržaj zadane verzije datoteke i upisati prijevod. U Java kôdu tekstovi se učitavaju prema nazivu, dok je `strings.xml` datoteka automatski odabrana prema postavkama sustava.

`strings.xml` (hr):

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">Memento</string>
  <string name="tab_title_in_progress">U tijeku</string>
  <string name="tab_title_complete">Završeno</string>
  <string name="tab_title_news">Novosti</string>
  <string name="dialog_title">Dodaj novi zadatak</string>
  <string name="dialog_title_long">Odaberite radnju</string>
  <string name="dialog_task_name">Naziv</string>
  <string name="dialog_date">Datum</string>
  <string name="dialog_time">Vrijeme</string>
  <string name="dialog_category">Kategorija</string>
  <string name="save">Spremi</string>
  <string name="delete">Obriši</string>
  <string name="completed">Završeno</string>
  <string name="cancel">Otkazi</string>
  <string name="loading">Učitavam</string>
  <string name="loading_message">Dohvaćam podatke sa web servisa..</string>
</resources>
```

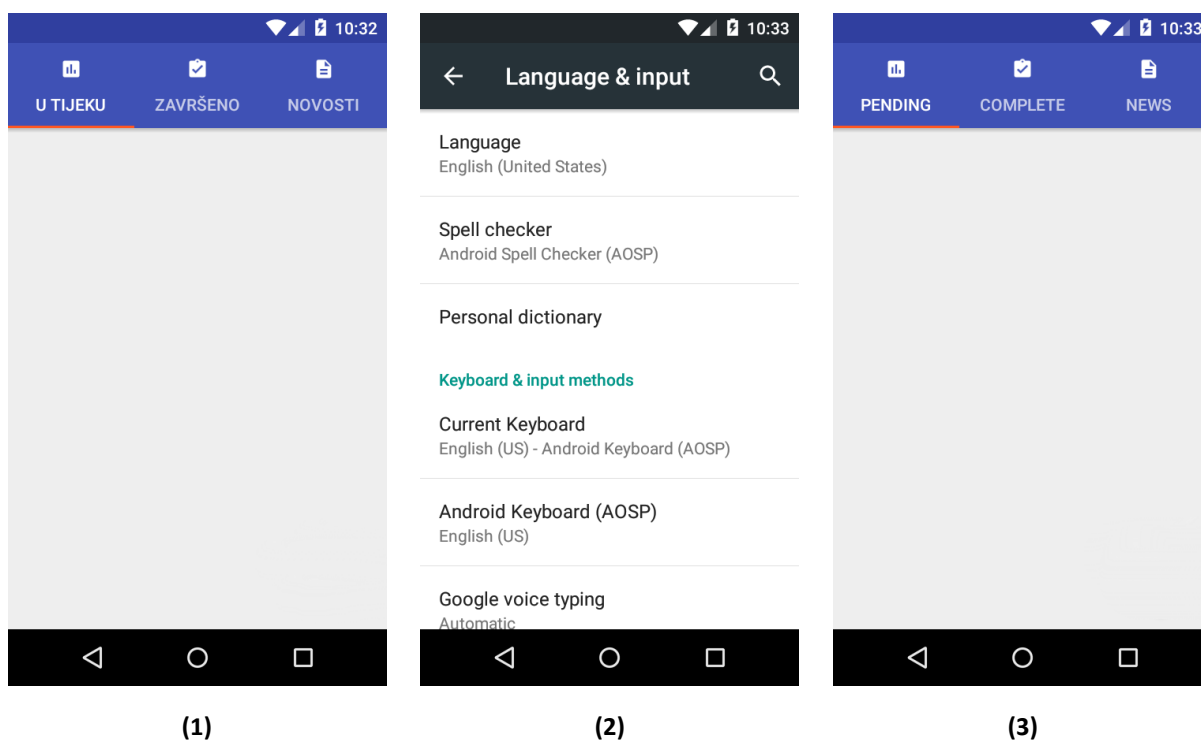
I konačno, da bi promjene bile uočljive, potrebno je u Java kôdu dinamički čitati tekstove. Trenutno, jedino mjesto na kojem je to moguće jest u `MainActivity.java` za naslove tabova:

```
private void setupViewPager(final ViewPager viewPager) {
  // kôd za prikaz pojedinih tabova
  final MainPagerAdapter adapter = new
  MainPagerAdapter(getSupportFragmentManager(), this);
  adapter.addFragment(new ActiveTasksFragment(),
```

```

getString(R.string.tab_title_in_progress));
    adapter.addFragment(new CompleteTasksFragment(),
getString(R.string.tab_title_complete));
    adapter.addFragment(new NewsFragment(), getString(R.string.tab_title_news));
    viewPager.setAdapter(adapter);
}
    
```

Aplikaciju je sada moguće pokrenuti, a promjene se mogu uočiti samo promjenom jezika Android sustava, kao što pokazuje slika (Slika 85). Prvi dio (1) pokazuje trenutni izgled Mementa u slučaju kada je hrvatski jezik zadani jezik Android sustava. U drugom dijelu slike (2) promijenjen zadani jezik je promijenjen u engleski, nakon čega je Memento ponovno pokrenut. Sada je vidljivo (3) da se koristi engleska verzija `strings.xml` datoteke.



*Slika 85. Promjena jezika Android sustava mijenja zadanu `strings.xml` datoteku*

### 5.3.6 RecyclerView, drugi dio – adapteri

Svaki pogled od tri ranije kreirana temeljna fragmenta sadržavat će listu objekata, odnosno popis zadataka ili popis vijesti. Kako bi se lista iz programskog kôda prikazala na listi grafičkog korisničkog sučelja, koristi se ranije spominjan RecyclerView kojeg je potrebno dodati u pogled `fragment_active_tasks.xml`:

```

<?xml version="1.0" encoding="utf-8"?>

<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    
```

```

<LinearLayout
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="5dp"
    android:id="@+id/activeTaskLinerLayout">

    <android.support.v7.widget.RecyclerView
        android:id="@+id/rv_active_tasks"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

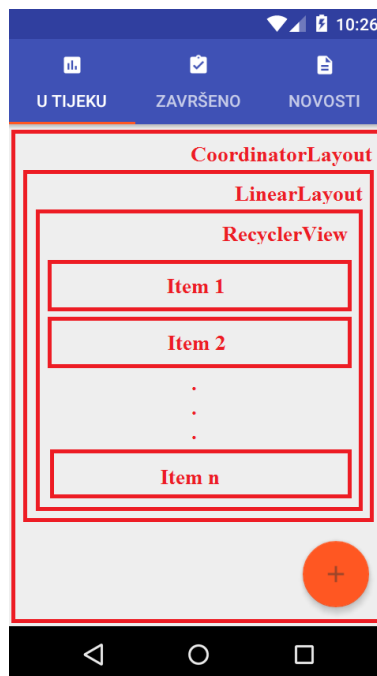
</LinearLayout>

<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="16dp"
    android:clickable="true"
    android:src="@drawable/ic_add"
    app:layout_anchor="@id/activeTaskLinerLayout"
    app:layout_anchorGravity="bottom|right|end"/>

</android.support.design.widget.CoordinatorLayout>

```

Obzirom da FAB koristi sliku, potrebno je dodati istu u `res/drawable` na sličan način kao i ranije, uz razliku da se ovaj put iz padajućeg izbornika odabere „Action bar and tab icons“. Nakon pokretanja aplikacije, FAB se nalazi u donjem desnom kutu, kako je prikazano na slici (Slika 86). Ista slika ujedno pokazuje grafičke elemente koji trenutno nisu vidljivi.



*Slika 86. Početni prikaz nakon dodavanja FAB-a i prikaz okvira pojedinih pogleda*

`CoordinatorLayout` se u ovom slučaju koristi za prikaz FAB-a i `LinearView` koji će sve elemente liste sortirati jedan do drugog, horizontalno ili vertikalno, ovisno o odabiru. Jedini podelement trenutno će biti `RecyclerView` koji će sadržavati listu zadataka Item 1, Item 2, ... , Item n (ili u slučaju za Novosti listu vijesti).

Jedan element liste, na slici (Slika 86) označen kao „Item“, potrebno je upisati pogledom u XML-u, te ga novim Adapter-om vezati za RecyclerView. Unutar /res/layout napraviti novu datoteku naziva task\_item.xml za opis izgleda jednog retka RecyclerView-a, odnosno zadatka:

```
<?xml version="1.0" encoding="utf-8" ?>
<android.support.v7.widget.CardView
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:id="@+id/cv"
android:layout_marginBottom="5dp"
android:clickable="true"
android:focusable="true"
android:foreground="?android:attr/selectableItemBackground">

<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <LinearLayout
        android:layout_width="8dp"
        android:layout_height="match_parent"
        android:id="@+id/lv_category"
        android:orientation="horizontal"
        android:background="@android:color/holo_blue_dark">
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:padding="5dp">

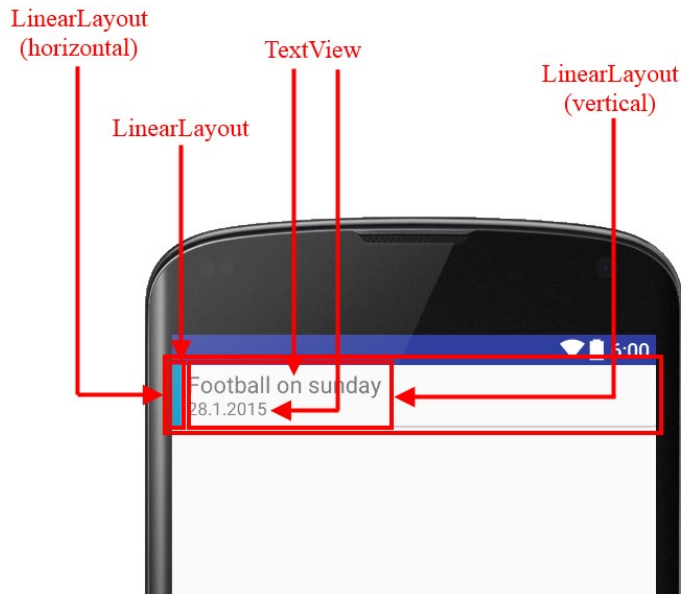
        <TextView
            android:id="@+id/tv_rv_name"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            tools:text="Football on Sunday"
            style="@style/Base.TextAppearance.AppCompat.Medium"/>

        <TextView
            android:id="@+id/tv_rv_deadline"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            tools:text="28.1.2015"
            style="@style/Base.TextAppearance.AppCompat.Small"/>

    </LinearLayout>
</LinearLayout>
</android.support.v7.widget.CardView>
```

Slika 87 vizualizira navedeni XML kôd s naglašenim korištenim grafičkim elementima, LinearLayout koji elemente smješta horizontalno, to su novi LinearLayout za prikaz boje odabrane kategorije te još jedan LinearLayout koji svoje elemente smješta vertikalno. Ti elementi su TextView pogledi koji služe prikazu naslova zadatka i vremenu kada se održava.





*Slika 87. Grafička reprezentacija XML kôda za opis jednog elementa RecyclerView liste za prikaz zadataka*

Posljednji korak prije nego što je moguće pokrenuti mobilnu aplikaciju je povezati RecyclerView iz pogleda sa listom u kôdu, odnosno kreirati RecyclerView.Adapter koji `task_item.xml` pridružuje k `fragment_active_tasks.xml`.

Unutar postojećeg paketa `adapters` dodati novu klasu i nazvati je `ActiveTasksRecyclerViewAdapter.java` koja služi povezivanju liste zadataka dohvaćenih iz lokalne baze podataka listi prikazanoj na ekranu. Uz to, treba napraviti novi paket naziva `viewholders` unutar kojeg je potrebno kreirati klasu `TaskViewHolder.java`. `ViewHolder` klasa predstavlja logiku upravljanja elementima liste, odnosno obrađuje događaje i radi sa grafičkom reprezentacijom jednog elementa RecyclerView-a [32], koji je u ovom slučaju `task_item.xml`.

Klasu `TaskViewHolder` proširiti klasom `RecyclerView.ViewHolder`:

```
class TaskViewHolder extends RecyclerView.ViewHolder{
    public TaskViewHolder(View itemView) {
        super(itemView);
    }
}
```

Nadalje, `ViewHolder` se obično veže za entitetne klase koje opisuju objekte prikazane u RecyclerView listi. Kako trenutno Memento nema nikakvih podataka ni takvih klasa, prije daljnjeg razvoj potrebno je kreirati entitetne klase, odnosno klase koje će opisivati zadatke i novosti, a čiji će objekti kasnije biti spremni u bazu podataka.

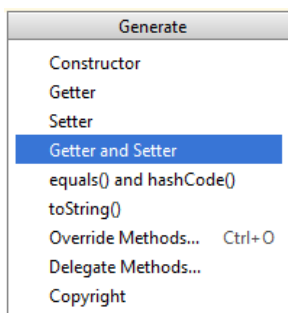
## 5.4 Izrada entitetnih klasa

Za entitetne klase treba napraviti novi paket i nazvati ga `entities`, a unutar istog kreirati tri klase, koje reprezentiraju tri glavna entiteta koja Memento koristi; *zadatak*, *kategorija zadatka* i *vijest*. Kako će zadatak i kategorija zadatka biti zapisani u lokalnoj bazi podataka njihove će klase imati prefiks `Db`, a klasa za vijesti koje se učitavaju putem *web servisa* će imati prefiks `Ws`. Dakle, kreirati `DbCategory.java`, `DbTask.java` i `WsNewItem.java`.

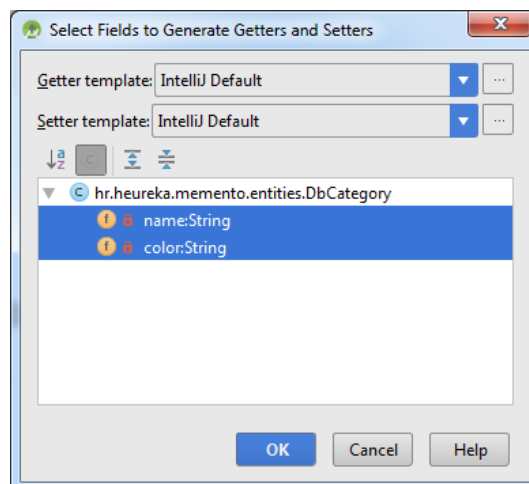
`DbCategory.java` sadrži dva atributa, za naziv kategorije i kôd boje:

```
public class DbCategory {
    private String name;
    private String color;
}
```

Korištenjem AndroidStudio alata, kreirati mutatorske metode i konstruktor. Na bijeli prostor u području za pisanje kôda desnim klikom odabrati opciju „Generate“ (ili kraće `Alt+Insert`), nakon čega se pojavljuje kontekstualni izbornik (Slika 88-1), unutar kojeg treba označiti sve attribute (Slika 88-2).



(1)



(2)

Slika 88. Generiranje mutatorskih metoda

Na gotovo identičan način se dodaje konstruktor, no na kontekstualnom izborniku prikazanom na Slika 88-1 ovaj puta odabrati opciju „Constructor“, a kao parametre označiti svojstva `name` i `color`, što će rezultirati sa:

```
public class DbCategory {
    private String name;
    private String color;

    public DbCategory(String name, String color) {
        this.name = name;
        this.color = color;
    }
}
```

```

}
public String getName() { return name; }

public void setName(String name) { this.name = name; }

public String getColor() { return color; }

public void setColor(String color) { this.color = color; }
}

```

Isto je potrebno napraviti za klasu DbTask čiji konstruktor treba primiti parametre name, dueDate i category:

```

public class DbTask {

    private String name;
    private Date dueDate;
    private DbCategory category;
    private int completed;

    public DbTask(String name, Date dueDate, DbCategory category) {
        this.name = name;
        this.dueDate = dueDate;
        this.category = category;
        this.completed = 0;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Date getDueDate() {
        return dueDate;
    }

    public void setDueDate(Date dueDate) {
        this.dueDate = dueDate;
    }

    public DbCategory getCategory() {
        return category;
    }

    public void setCategory(DbCategory category) {
        this.category = category;
    }

    public int getCompleted() {
        return completed;
    }

    public void setCompleted(int completed) {
        this.completed = completed;
    }
}

```

I na kraju klasa WsNewsItem.java:

```

public class WsNewsItem {

    private String name;
    private String text;
    private Date date;
    private String image_path;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    public String getImage_path() {
        return image_path;
    }

    public void setImage_path(String image_path) {
        this.image_path = image_path;
    }
}

```

Kako trenutno lokalna baza podataka još nije napravljena, potrebno je napraviti demo, odnosno testne podatke koji će se prikazivati u listama. Za tu svrhu, jednostavno kreirati novu klasu nazvanu MockDataLoader unutar novog paketa kojeg treba nazvati helpers.

MockDataLoader.java:

```

public class MockDataLoader {

    public static List<DbTask> getDemoData(){

        List<DbTask> mItems = new ArrayList<>();
        mItems.add(new DbTask("Tennis on Sunday", new Date(), new
DbCategory("Sport", "#000080")));
        mItems.add(new DbTask("Math", new Date(), new DbCategory("Homework",
"#FF0000")));
        mItems.add(new DbTask("Drone flight", new Date(), new DbCategory("Hobby",
"#CCCCCC")));

        return mItems;
    }
}

```

Da bi se ti podaci prikazali, potrebno je dovršiti `TaskViewHolder` klasu proširenjem koje povezuje kôd sa elementima grafičkog korisničkog sučelja, a putem konstruktora dobiva referencu na odgovarajući `RecyclerView` te popis elemenata, odnosno zadataka za prikaz. `TaskViewHolder` postaje:

```
public class TaskViewHolder extends RecyclerView.ViewHolder{

    public ActiveTasksRecyclerViewAdapter adapter;
    // povezivanje sa grafičkim elementima
    @Bind(R.id.tv_rv_name)
    public TextView taskTitle;

    @Bind(R.id.tv_rv_deadline)
    public TextView taskDate;

    @Bind(R.id.lv_category)
    public LinearLayout linearLayout;

    private Context context;
    private List<DbTask> mItems;
    // konstruktor prima parametar trenutnog pogleda, adapter i elemente liste koje
    će prikazati
    public TaskViewHolder(final View itemView, ActiveTasksRecyclerViewAdapter
    adapter, List<DbTask> mItems) {
        super(itemView);

        ButterKnife.bind(this, itemView);
        this.context = itemView.getContext();

        this.adapter = adapter;
        this.mItems = mItems;
    }
}
```

Također, treba završiti `ActiveTasksRecyclerViewAdapter` koji će koristiti instancu klase `TaskViewHolder` i pomoću nje postaviti vrijednosti atributa za prikaz:

```
public class ActiveTasksRecyclerViewAdapter extends
RecyclerView.Adapter<TaskViewHolder>{

    List<DbTask> taskItems;
    Context context;

    public ActiveTasksRecyclerViewAdapter(List<DbTask> taskItems, Context context){
        super();
        this.context = context;
        this.taskItems = taskItems;
    }

    @Override
    public TaskViewHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
        View v =
        LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.task_item, viewGroup,
        false);
        return new TaskViewHolder(v, this, taskItems);
    }

    @Override
    public void onBindViewHolder(TaskViewHolder viewHolder, int i) {
        viewHolder.taskTitle.setText(taskItems.get(i).getName());
        SimpleDateFormat sdf = new SimpleDateFormat("dd.MMM.yyyy, HH:mm",
        context.getResources().getConfiguration().locale);
```

```

viewHolder.taskDate.setText(sdf.format(taskItems.get(i).getDueDate().getTime()));

viewHolder.linearLayout.setBackgroundColor(Color.parseColor(taskItems.get(i).getCategory().getColor()));
}

@Override
public int getItemCount() {
    return taskItems.size();
}
}

```

Posljednji korak prije testiranja prikaza jest u `ActiveTasksFragment` fragmentu dohvatiti `RecyclerView` i prosljediti mu demo podatke za prikaz:

```

public class ActiveTasksFragment extends Fragment {

    @Bind(R.id.rv_active_tasks)
    RecyclerView recycleView;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_active_tasks, container, false);
        ButterKnife.bind(this, rootView);
        return rootView;
    }

    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);

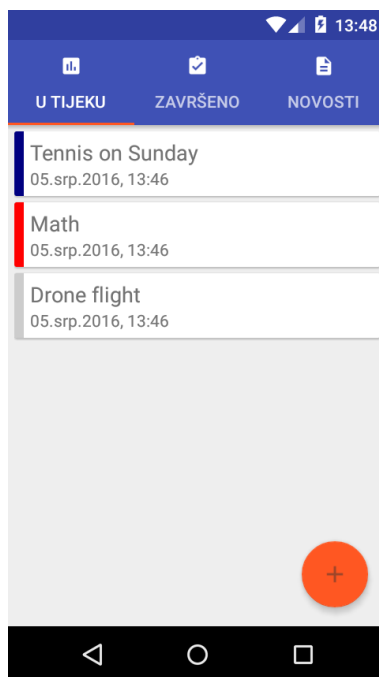
        // prikazadi RecyclerView uvijek iste veličine, bez obzira na broj
        // elemenata koje prikazuje
        recycleView.setHasFixedSize(true);

        // trenutnom RecyclerView-u pridružiti objekt za pozicioniranje pogleda
        // (LayoutManager)
        LinearLayoutManager llm = new LinearLayoutManager(getActivity());
        recycleView.setLayoutManager(llm);

        // postavljanje adaptera
        recycleView.setAdapter(new
ActiveTasksRecyclerViewAdapter(MockDataLoader.getDemoData(), getContext()));
    }
}

```

Memento je spreman za testiranje prikaza demo podataka. Pokretanje mobilne aplikacije rezultira ekranom koji prikazuje tri unesena demo podatka prikazana na slici (Slika 89).



*Slika 89. Prikaz demo podataka*

## 5.5 Unos i prikaz podataka – rad s dijalozima i fragmentima pogleda

Kako bi se korisnicima mobilne aplikacije Memento omogućilo unošenje podataka, potrebno je kreirati dijalog koji će sadržavati polja za unos naziva zadatka, vremena i datuma te odabira kategorije. Dijalog se aktivira pritiskom na FAB.

Prvi korak za postizanje tog cilja jest kreirati novu `task_dialog.xml` datoteku u `/res/layouts` direktoriju koja opisuje dizajn dijaloga:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="15dp">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:text="@string/dialog_title"
        android:id="@+id/dialogTaskName" />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/dialogEditTaskName" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:text="@string/dialog_date"
        android:id="@+id/dialogTaskDate" />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="date"
        android:ems="10"
        android:id="@+id/dialogEditDate" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:text="@string/dialog_time"
        android:id="@+id/dialogTaskTime" />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="time"
        android:ems="10"
        android:id="@+id/dialogEditTime"
        android:layout_gravity="right" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:text="@string/dialog_category"
        android:id="@+id/dialogTaskCategory" />
```

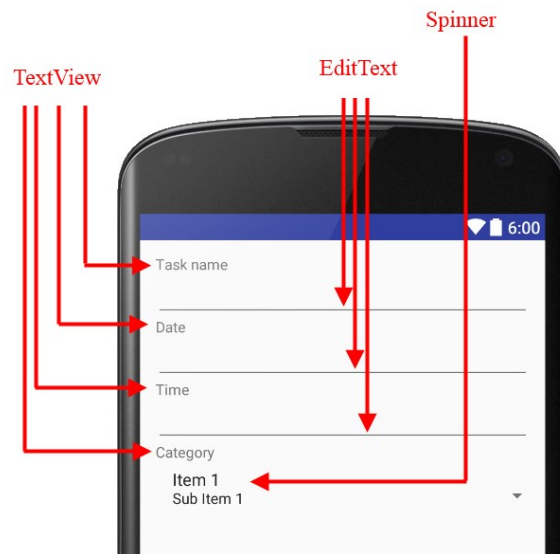


```

<Spinner
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/dialogCategorySpinner" />
</LinearLayout>

```

Grafička reprezentacija navedenog kôda prikazana je na slici (Slika 90). Koriste se tri `TextView` pogleda koji samo navode sadržaj koji korisnik unosi, tri `EditText` pogleda u koje se zapisuje korisnikov odabir i `Spinner` koji omogućuje odabir kategorije iz padajućeg izbornika.



*Slika 90. Vizualizacija `task_dialog.xml` kôda s naglašenim grafičkim elementima*

Kako bi se prikazao kreirani dijalog pritiskom na FAB, u klasi `ActiveTaskFragment` treba dodati metodu koju `ButterKnife` aktivira prilikom `onClick` događaja:

```

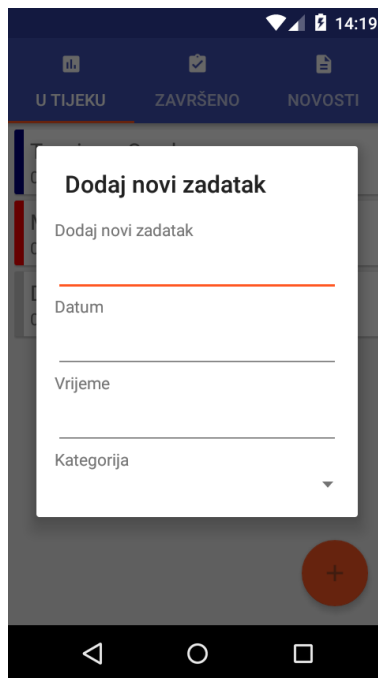
@OnClick(R.id.fab)
public void onFabClicked(View view) {
    AlertDialog.Builder dialog = new AlertDialog.Builder(getContext());
    View dialogView =
    getActivity().getLayoutInflater().inflate(R.layout.task_dialog, null);

    dialog.setView(dialogView);

    dialog.setTitle(getString(R.string.dialog_title));
    dialog.show();
}

```

Pokretanjem Mementa i pritiskom na FAB, otvara se dijalog, no kako trenutno `Spinner` nema asociiranih podataka, nije moguće odabrati kategoriju. Rezultat pokretanja vidljiv je na slici (Slika 91).



*Slika 91. Dijalog za dodavanje novog zadatka (bez dodanih kategorija)*

Nije poželjno da korisnik samostalno unosi vrijeme i datum zbog mogućnosti unosa različitih formatiranja podataka, zbog čega će se uz EditText poglede asocirane uz datum i vrijeme aktivirati posebni dijalozi Android sustava koji služe odabiru vremenskih podataka. Za to je potrebno dodati novu klasu u paket helpers naziva DialogHelper.java i ona će biti odgovorna za prikaz dijaloga za unos datuma i vremena, za asociranje Spinner-a sa podacima te za spremanje unosa korisnika, najprije samo u privremenu memoriju, a kasnije i u bazu podataka. Sadržaj klase je:

```
public class DialogHelper {

    private Context context;

    @Bind(R.id.dialogEditTaskName)
    EditText editTaskName;

    @Bind(R.id.dialogEditDate)
    EditText editDate;

    @Bind(R.id.dialogEditTime)
    EditText editTime;

    @Bind(R.id.dialogCategorySpinner)
    Spinner categorySpinner;

    Calendar selectedDate = Calendar.getInstance();

    public DialogHelper(Context context, View parentView) {
        this.context = context;
        ButterKnife.bind(this, parentView);

        populateSpinner();
    }

    private void populateSpinner() {
        List<String> spinnerStringItems = new ArrayList<>();

        // dohvatiti trenutne "demo" kategorije
        for (DbTask taskCategory : MockDataLoader.getDemoData()) {
            spinnerStringItems.add(taskCategory.getCategory().getName());
        }
    }
}
```

```

    }

    // napraviti adapter koji string-ove dodjeljuje elementima spinnera
    ArrayAdapter<String> spinnerAdapter = new ArrayAdapter<String>(
        context, android.R.layout.simple_spinner_item, spinnerStringItems);

spinnerAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_ite
m);

    // postavi adapter
    categorySpinner.setAdapter(spinnerAdapter);
}
}

```

Za prikaz zadanih dijaloga Android sustava koji omogućuju unos datuma i vremena u točno definiranom formatu (npr. mm/dd/yyyy - HH:mm) potrebno je pomoću ButterKnife biblioteke reagirati na onFocusChange događaj za navedene EditText poglede. Isto tako nakon prikaza pojedinog dijaloga za odabir datuma ili vremena, potrebno je dohvatiti unos korisnika. U tu svrhu potrebno je dodati dvije metode koje reagiraju na onFocusChange događaj te dvije metode koje reagiraju na događaj završetka unosa podataka korisnika. Na kraj klase DialogHelper.java dodati:

```

// prikaz dijaloga za odabir vremena
@OnFocusChange(R.id.dialogEditTime)
public void onEditTimeClick(View view, boolean hasFocus){
    if(!hasFocus) return;
    TimePickerDialog timePickerDialog = new TimePickerDialog(context,
onTimeSetListener,
        selectedDate.get(Calendar.HOUR_OF_DAY),
selectedDate.get(Calendar.MINUTE), true);
    timePickerDialog.show();
}

// prikaz dijaloga za odabir datuma
@OnFocusChange(R.id.dialogEditDate)
public void onEditDateClick(View view, boolean hasFocus){
    if(!hasFocus) return;
    DatePickerDialog datePickerDialog = new DatePickerDialog(context,
onDateSetListener,
        selectedDate.get(Calendar.YEAR), selectedDate.get(Calendar.MONTH),
selectedDate.get(Calendar.DAY_OF_MONTH));
    datePickerDialog.show();
}

// dohvaćanje odabranog vremena
TimePickerDialog.OnTimeSetListener onTimeSetListener = new
TimePickerDialog.OnTimeSetListener() {
    @Override

    public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
        selectedDate.set(Calendar.HOUR_OF_DAY, hourOfDay);
        selectedDate.set(Calendar.MINUTE, minute);

        SimpleDateFormat df = new SimpleDateFormat("HH:mm");
        editTime.setText(df.format(selectedDate.getTime()));
    }
};

// dohvaćanje odabranog datuma
DatePickerDialog.OnDateSetListener onDateSetListener = new
DatePickerDialog.OnDateSetListener() {

```

```

@Override
public void onDateSet(DatePicker view, int year, int monthOfYear, int
dayOfMonth) {
    selectedDate.set(Calendar.YEAR, year);
    selectedDate.set(Calendar.MONTH, monthOfYear);
    selectedDate.set(Calendar.DAY_OF_MONTH, dayOfMonth);

    SimpleDateFormat df = new SimpleDateFormat("dd.MM.yyyy");
    editDate.setText(df.format(selectedDate.getTime()));
}
};

```

I na kraju je potrebno dodati instancu klase `DialogHelper` u `ActiveTasksFragment` klasu koja će se kasnije koristiti za iniciranje postupka pohrane podataka. U klasu `ActiveTasksFragment`, odnosno metodu `onFabClicked` dodati podebljano liniju kôda:

```

@OnClick(R.id.fab)
public void onFabClicked(View view) {
    AlertDialog.Builder dialog = new AlertDialog.Builder(getContext());
    View dialogView =
    getActivity().getLayoutInflater().inflate(R.layout.task_dialog, null);

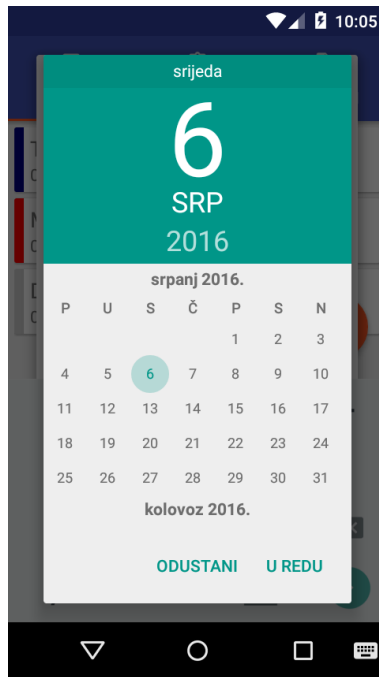
    dialog.setView(dialogView);

    final DialogHelper dialogHelper = new DialogHelper(getContext(), dialogView);

    dialog.setTitle(getString(R.string.dialog_title));
    dialog.show();
}

```

Memento je sada moguće testirati na način da se ponovno pritisne FAB i odabere `EditText` za unos datuma, što će aktivirati zadani prozor Android sustava kao što je prikazano na slici (Slika 92). Uz to ovaj put je i `Spinner` popunjen demo podacima.



*Slika 92. Zadani dijalog Androida za odabir datuma*

## 5.6 Mobilna baza podataka

Za olakšano korištenje baze podataka dobra je praksa primijeniti neku biblioteku treće strane jer bez toga je korištenje SQLite baze podataka koju nudi Android sustav dosta nepraktično i složeno. U ovom projektu koristi se **ActiveAndroid**, za što je potrebno u `build.gradle` (`Project:Memento`) dodati Maven repozitorij:

```
allprojects {
    repositories {
        jcenter()
        // za ActiveAndroid
        mavenCentral()
        maven { url "https://oss.sonatype.org/content/repositories/snapshots/" }
    }
}
```

Nakon čega je u `build.gradle` (`Module:app`) dodati zavisnost prema **ActiveAndroid** biblioteci:

```
// active android
compile 'com.michaelpardo:activeandroid:3.1.0-SNAPSHOT'
```

Za dovršetak procesa dodavanja baze podataka u `AndroidManifest.xml` potrebno je dodati meta podatke o bazi podataka, odnosno njeno ime i verziju. Nakon svake promjene strukture baze podataka, odnosno entitetnih klasa verziju treba ručno povećati za jedan. Neposredno nakon taga koji se odnosi na glavu aktivnost, no prije završetka taga aplikacije dodati:

```
<meta-data
    android:name="AA_DB_NAME"
    android:value="memento.db" />
<meta-data
    android:name="AA_DB_VERSION"
    android:value="1" />
```

Nakon ovog koraka, baza podataka je spremna za korištenje.

Kako bi ranije dodane klase, postale entitetne klase koje se mogu spremati i čitati iz baze podataka **Mementa** potrebno je proširiti klasom `Model` iz **ActiveAndroid** biblioteke, klasi dodati anotaciju koja definira naziv pripadajuće tablice, a atributima dodati antoacije koje definiraju nazive pripadajućih atributa. Za klasu `DbCategory` proširenje izgleda ovako:

```

@Table(name = "category")
public class DbCategory extends Model{

    @Column(name = "name")
    private String name;
    @Column(name = "color")
    private String color;

    public DbCategory() {
    }

    // Ostatak klase se ne mijenja
}

```

Klasa DbTask postaje:

```

@Table(name="task")
public class DbTask extends Model{

    @Column(name = "name")
    private String name;
    @Column(name = "date")
    private Date dueDate;
    @Column(name = "category")
    private DbCategory category;

    @Column(name = "completed")
    private int completed;

    public DbTask() {}

    // Ostatak klase se ne mijenja
}

```

I konačno, dodati inicijalizaciju ActiveAndroid biblioteke u onCreate metodu glavne aktivnosti, MainActivity:

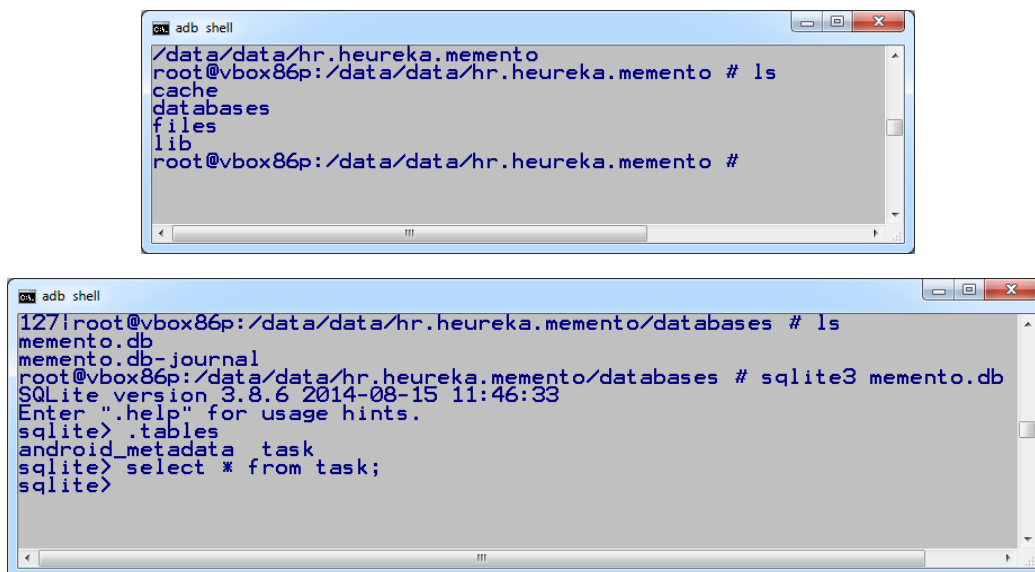
```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // inicijalizacija ActiveAndroid biblioteke - baze podataka
    ActiveAndroid.initialize(this);
    // aktivacija ButterKnife biblioteke
    ButterKnife.bind(this);

    // Ostatak se ne mijenja
}

```

Nakon pokretanja Mementa, korištenjem adb-a moguće je navigirati do baze podataka, te koristiti Sqlite3 za izvršavanje kao što prikazuje slika (Slika 93).



*Slika 93. Postavljanje SQL upita direktno na bazi podataka mobilnog uređaja*

Kako je vidljivo na slici (Slika 93), trenutno nema podataka, stoga potrebno je izmijeniti klasu `MockDataLoader` tako da koristi bazu podataka i popuni je demo podacima. No prije toga klasama `DbTask` i `DbCategory` treba dodati metode kojima će se dohvaćati njihovi podaci iz baze podataka. Za `DbCategory` to će biti metoda `getAll` koja dohvaća sve kategorije i `getCategoryByName` koja dohvaća kategoriju prema zadanom nazivu:

```

public static DbCategory getCategoryByName(String name){
    return new Select().from(DbCategory.class).where("name = ? ",
name).executeSingle();
}

public static List<DbCategory> getAll(){
    return new Select().from(DbCategory.class).execute();
}
    
```

Za klasu `DbTask` to će biti metoda `getAll` koja će dohvatiti sve završene ili nezavršene zadatke, ovisno o zadanom parametru:

```

public static List<DbTask> getAll(int completed){
    return new Select().from(DbTask.class).where("completed = ? ",
completed).orderBy("date DESC").execute();
}
    
```



Da bi klasa `MockDataLoader` koristila bazu podataka potrebno je napraviti listu objekata klasa sa `Db` prefiksom, nakon čega se nad svim objektima poziva metoda `save` iz superklase `Model` `ActiveAndroid-a`:

```
public class MockDataLoader {

    public static void loadMockData() {

        // demo kategorije
        List<DbCategory> mCategories = new ArrayList<>();
        mCategories.add(new DbCategory("Sport", "#000080"));
        mCategories.add(new DbCategory("Books", "#00AA00"));
        mCategories.add(new DbCategory("Homework", "#FF0000"));
        mCategories.add(new DbCategory("Hobby", "#CCCCCC"));

        // pohrana kategorija u bazu podataka
        for (DbCategory category : mCategories){
            category.save();
        }

        // ne završeni demo zadaci
        List<DbTask> mItems = new ArrayList<>();
        mItems.add(new DbTask("Football on Sunday", new Date(),
        DbCategory.getCategoryByName("Sport")));
        mItems.add(new DbTask("Concert", new Date(),
        DbCategory.getCategoryByName("Hobby")));
        mItems.add(new DbTask("Math homework", new Date(),
        DbCategory.getCategoryByName("Homework")));

        // pohrana ne završenih zadataka
        for (DbTask task : mItems){
            task.save();
        }
        mItems.clear();

        // završeni demo zadaci
        mItems.add(new DbTask("Cycling cup", new Date(),
        DbCategory.getCategoryByName("Sport")));
        mItems.add(new DbTask("FPV AUV Flight", new Date(),
        DbCategory.getCategoryByName("Hobby")));

        // pohrana završeni zadataka
        for (DbTask task : mItems){
            task.setCompleted(1);
            task.save();
        }
    }
}
```

U klasi `ActiveTasksFragment`, treba ažurirati metodu `onViewCreated` kako bi ona čitala podatke iz baze podataka:

```
// postavljanje adaptera
recyclerView.setAdapter(new ActiveTasksRecyclerViewAdapter(DbTask.getAll(0),
getContext()));
```

U klasi `DialogHelper`, izmijeniti metodu `populateSpinner` da bi prikazivala podatke iz baze podataka:

```
// dohvatiti trenutne "demo" kategorije
for (DbCategory category : DbCategory.getAll()){
```

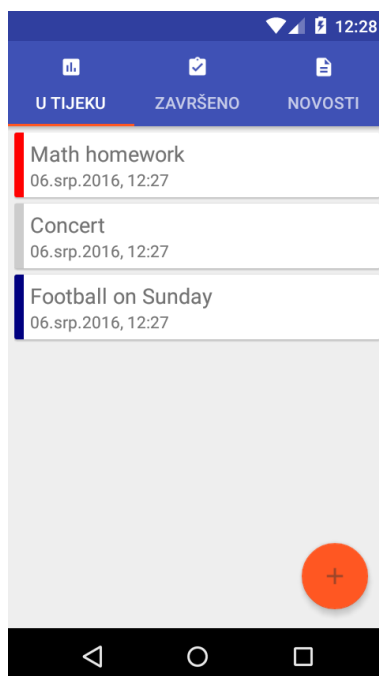
```
spinnerStringItems.add(category.getName());
}
```

U klasi MainActivity, za svrhe testiranja, u onCreate metodi pozvati spremanje demo podataka ukoliko je baza podataka prazna sa sljedećim kôdom:

```
if(DbTask.getAll().isEmpty()){
    MockDataLoader.loadMockData();
}
```

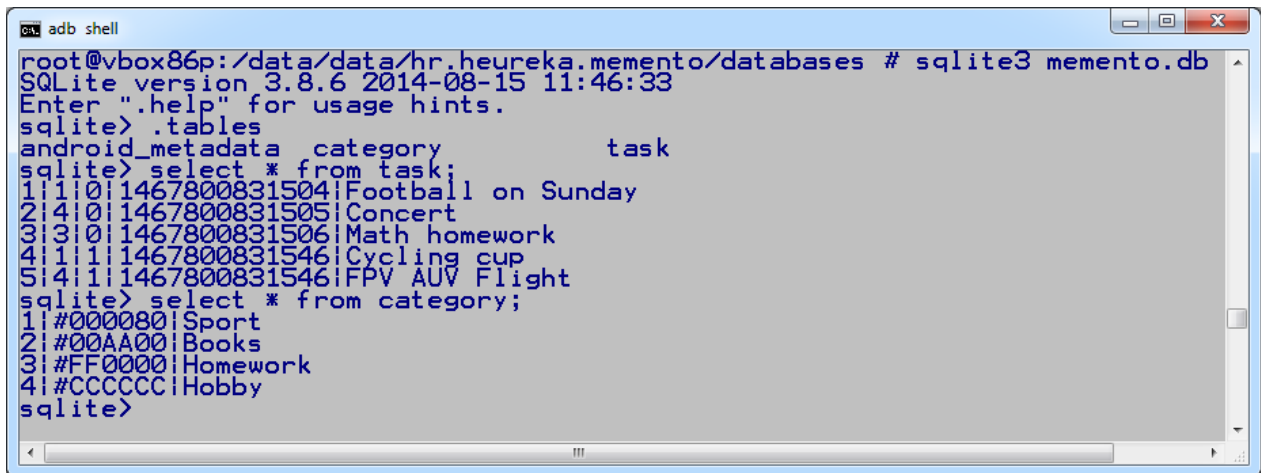
Kako će u konačnoj verziji korisnik popunjavati bazu podataka osobnim zadacima, ovaj kôd se u kasnijoj fazi, nakon testiranja rada sa bazom podataka može obrisati.

Memento se sada može pokrenuti, čime se dobivaju podaci iz baze podataka:



*Slika 94. Zadaci učitani iz mobilne baze podataka*

Slično kao ranije, putem adb-a je moguće upitima nad bazom podataka provjeriti unesene podatke te njima manipulirati direktno iz ljuske Android operacijskog sustava, što je prikazano na slici (Slika 95).



```

adb shell
root@vbox86p:/data/data/hr.heureka.memento/databases # sqlite3 memento.db
SQLite version 3.8.6 2014-08-15 11:46:33
Enter ".help" for usage hints.
sqlite> .tables
android_metadata  category          task
sqlite> select * from task;
1|1|0|1467800831504|Football on Sunday
2|4|0|1467800831505|Concert
3|3|0|1467800831506|Math homework
4|1|1|1467800831546|Cycling cup
5|4|1|1467800831546|FPV AUV Flight
sqlite> select * from category;
1|#000080|Sport
2|#00AA00|Books
3|#FF0000|Homework
4|#CCCCCC|Hobby
sqlite>

```

*Slika 95. Pregled unesenih podataka putem ljuske Android operacijskog sustava*

Kako trenutno završeni zadaci nisu vidljivi na tabu „Završeno“, potrebno je na vrlo sličan način kao i ranije za aktivne zadatke, ažurirati fragment za prikaz završenih zadataka. U ovom slučaju, pogled fragment\_complete\_tasks.xml za razliku od fragment\_active\_tasks.xml ne sadrži FAB, već samo RecyclerView drugog naziva:

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="5dp">

    <android.support.v7.widget.RecyclerView
        android:id="@+id/rv_completed_tasks"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>

```

Dok je kôd klase CompleteTasksFragment, sličan kôdu klasi ActiveTasksFragment, uz drugi parametar za metodu DbTask.getAll(). Ovoga puta parametar je 1 i označava dohvaćanje završenih zadataka. Klasi CompleteTasksFragment potrebno je dodati zatamnjene linije kôda:

```

public class CompleteTasksFragment extends Fragment {
    @Bind(R.id.rv_completed_tasks)
    RecyclerView recycleView;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_complete_tasks,
container, false);
        ButterKnife.bind(this, rootView);
        return rootView;
    }

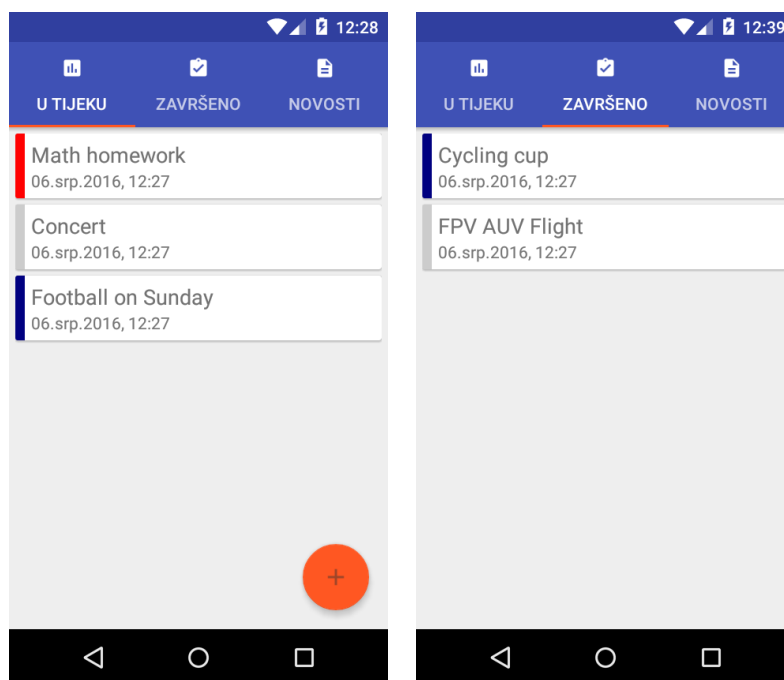
    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);

        recycleView.setHasFixedSize(true);

        LinearLayoutManager llm = new LinearLayoutManager(getActivity());
        recycleView.setLayoutManager(llm);

        recycleView.setAdapter(new ActiveTasksRecyclerViewAdapter(DbTask.getAll(1),
getContext()));
    }
}
    
```

Memento je spreman za testiranje, a nakon pokretanja, navigiranjem na sljedeći tab „Završeno“, prikazuju se i završeni zadaci dohvaćeni iz baze podataka. Rezultat je vidljiv na slici (Slika 96).



Slika 96. Prikaz završenih aktivnosti na tabu „Završeno“

### 5.6.1 Unos, pohrana i prikaz podataka

Umjesto da se prikazuju demo podaci koji su u svrhu testiranja umjetno dodani u bazu podataka, potrebno je proširiti trenutnu implementaciju mobilne aplikacije kako bi se omogućio zapis podataka korisnika.

Prvi korak k tome je dodavanje `save` metode u `DialogHelper` klasu koja će u novu instancu klase `DbTask` zapisati podatke koje korisnik unese putem dijaloga, a nakon toga nad tim objektom izvršiti metodu `save` ActiveAndroida. Na kraj klase `DbHelper` dodati metodu `save` sadržaja:

```
public void save() {
    DbTask task = new DbTask();
    task.setName(editTaskName.getText().toString());
    task.setDueDate(selectedDate.getTime()); //
    String category =
categorySpinner.getItemAtPosition(categorySpinner.getSelectedItemPosition()).toString();
    task.setCategory(DbCategory.getCategoryByName(category));
    task.save();
}
```

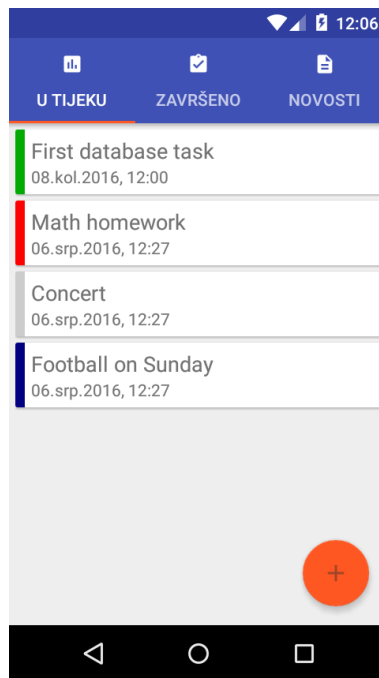
Isto tako, potrebno je proširiti `onFabClicked` metodu iz klase `ActiveTasksFragment` kako bi uhvatila događaj zatvaranja dijaloga i putem instance objekta `DialogHelper` inicirala upravo dodanu metodu `save`. Sadržaj metode `onFabClicked`:

```
dialog.setTitle(getString(R.string.dialog_title));
dialog.show();
```

zamijeniti sa:

```
dialog.setTitle(getString(R.string.dialog_title));
dialog.setPositiveButton(getString(R.string.save), new
DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int which) {
        dialogHelper.save();
        recycleView.setAdapter(new ActiveTasksRecyclerViewAdapter(DbTask.getAll(0),
getContext()));
    }
});
dialog.setNegativeButton(getString(R.string.cancel), null);
dialog.show();
```

Memento je spreman za pokretanje. Aktivacijom dijaloga za unos podataka pritiskom na FAB, u dijalogu treba unijeti novi zadatak „First database task“ 8.8.2016 u podne pod kategorijom Books. Slika 97 prikazuje rezultat uspješnog dodavanja.



Slika 97. Prikaz novoga zadatka unesenog putem dijaloga „First database task“

Trenutno dodane aktivnosti nije moguće brisati ili označiti kao završenima, stoga je potrebno dodati novi dijalog koji će korisniku to omogućiti. Novi dijalog aktivirati će se događajem „produženog pritiska“ (engl. long click). Kako se radi o pritisku na element liste RecyclerView pogleda, taj događaj je potrebno obraditi u pripadajućoj ViewHolder klasi. U ovom slučaju radi se o TaskViewHolder-u.

Prvi korak prema obradi spomenutog događaja je proširenje klase TaskViewHolder sučeljem View.OnLongClickListener:

```
public class TaskViewHolder extends RecyclerView.ViewHolder implements
View.OnLongClickListener
```

Nakon čega je potrebno generirati metode čiju implementaciju nalaže programsko sučelje (kraći način implementacije je već spomenut; Alt+Enter), što rezultira klasom metodom onLongClick:

```
@Override
public boolean onLongClick(View v) {
    return false;
}
```

Unutar navedene metode implementira se reakcija na događaj nakon što korisnik pritisne i nastavi neko vrijeme držati prst na ekranu, odnosno nad nekim zadatkom. U ovom slučaju, korisniku će se prikazati dijalog sa tri opcije: a) opcija kojom se zadatak označi kao završeni, b) opcija kojom se zadatak obriše i c) opcija kojom se dijalog otkaže.

Kreiranje dijaloga i obrada opcije brisanja (dodati unutar onLongClick metode, prije return naredbe) (BUTTON\_NEUTRAL, lijevi gumb).

```
AlertDialog dialog = new AlertDialog.Builder(context).create();
dialog.setTitle(context.getString(R.string.dialog_title_long));
dialog.setButton(AlertDialog.BUTTON_NEUTRAL, context.getString(R.string.delete),
new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {

        // dohvati element i obriši ga iz baze podataka
        mItems.get(getAdapterPosition()).delete();
        // dohvati element liste i obriši ga iz liste
        mItems.remove(getAdapterPosition());
        // dojaviti listi da je došlo do promjena kako bi se ažurirala
        adapter.notifyDataSetChanged();

        dialog.dismiss();
    }
});
```

Odmah nakon prethodnog kôda, potrebno je definirati ponašanje prilikom označavanja zadatka kao završenog (BUTTON\_POSITIVE, srednji gumb):

```
dialog.setButton(AlertDialog.BUTTON_POSITIVE,
context.getString(R.string.completed), new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        DbTask selectedTask = mItems.get(getAdapterPosition());

        // ako trenutni zadatak nije završen, postavi mu svojstvo završenost
        if(selectedTask.getCompleted() == 0) {
            selectedTask.setCompleted(1);
            selectedTask.save();
            // nakon pohrane u bazi podataka, pomakni ga na drugi tab (završenih)
            mItems.remove(getAdapterPosition());
            adapter.notifyDataSetChanged();
        }

        dialog.dismiss();
    }
});
```

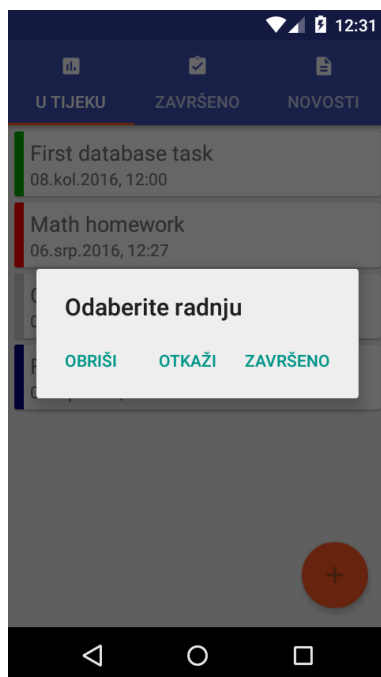
I zadnja opcija (BUTTON\_NEGATIVE, srednji gumb) je otkazivanje dijaloga:

```
dialog.setButton(AlertDialog.BUTTON_NEGATIVE, context.getString(R.string.cancel),
new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        dialog.dismiss();
    }
});
dialog.show();
```

Čime je metoda `onLongClick` završena (i ona vraća `false`). Jedino što preostaje jest prijaviti trenutnu klasu `TaskViewHolder` kao slušača događaja `onLongClick` unutar konstruktora:

```
// slušaj longClick događaj
itemView.setOnLongClickListener(this);
```

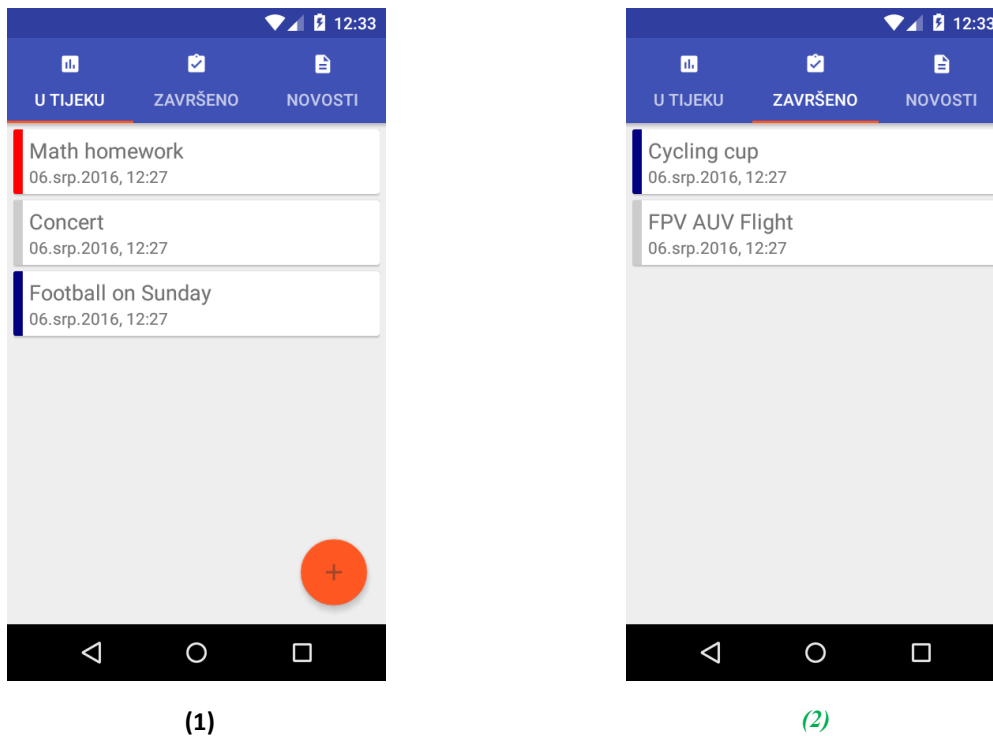
Nakon pokretanja Mementa te dužim pritiskom na neki od zadataka, aktivira se dijalog prikazan na slici (Slika 98).



*Slika 98. Dijalog koji omogućuje brisanje zadatka i označavanje zadatka kao završenog*

Nastavno na prethodnu radnju, ako se ranije dodani zadatak „First database task“ označi kao završen odabirom opcije „Završeno“, on se uklanja s liste. No, navigiranjem do taba „Završeno“ može se uočiti da nije prikazan, što je ilustrirano na slici (Slika 99).





*Slika 99. Nakon označavanja zadatka kao završenog, tab Završeno ne osvježava listu*

Rješenje navedenog problema je da prilikom pomicanja tabova, `ViewPager` kontrola inicira osvježavanje fragmenta za prikaz završenih zadataka. Za to je u klasi `MainActivity`, odnosno metodi `setupViewPager` potrebno dodati slušač događaja koji reagira na primjenu stranice. Na kraj metode `setupViewPager` dodati `addOnPageChangeListener`:

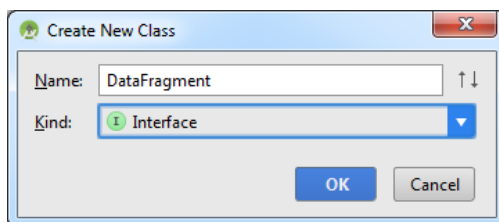
```
// osvježi nakon odabira
viewPager.addOnPageChangeListener(new ViewPager.OnPageChangeListener() {
    @Override
    public void onPageScrolled(int position, float positionOffset, int
positionOffsetPixels) {}

    @Override
    public void onPageSelected(int position) {
        // implementacija osvježavanja lista!
    }

    @Override
    public void onPageScrollStateChanged(int state) {}
});
```

Metoda `onPageSelected` treba inicirati osvježavanje podataka iz baze podataka prikazanih na fragmentu. Iako trenutno postoje samo dva takva fragmenta, kako bi u budućnosti uz moguće nadogradnje, a i sada bio jednostavniji pristup osvježavanju podataka, potrebno je napraviti programsko sučelje. Takvo programsko sučelje će sadržavati metodu koja inicira osvježavanje nad svim klasama koje ga implementiraju. Također, na taj način moguće je proći listom svih fragmenata i ukoliko je promatrani fragment instanca tog sučelja, pozvati metodu za osvježavanje.

U ovom slučaju, klase `ActiveTasksFragment` i `CompleteTasksFragment` će implementirati sučelje `DataFragment` koje je potrebno kreirati unutar `helpers` paketa. Desnim klikom na paket, odabrati opciju „New“, „Java Class“, nakon čega u dijalogu opciju „Kind“ odabrati „Interface“, kao što je prikazano na slici. `DataFragment` će implementirati svi fragmenti koji koriste bazu podataka.



*Slika 100. Kreiranje programskog sučelja*

Sučelje `DataFragment` sadrži prototip samo jedne metode, čija implementacija se mora pobrinuti za dohvaćanje odgovarajućih podataka:

```
public interface DataFragment {
    public void reloadData();
}
```

Klasa `CompleteTasksFragment`, će dohvatiti i prikazati zadatke koji su završeni, a programski kôd klase `ActiveTaskFragments` će dohvatiti i prikazati završene zadatke. Najprije, klasu `CompleteTasksFragments` treba proširiti klasom `Fragment` te njome implementirati `DataFragment` programsko sučelje:

```
public class CompleteTasksFragment extends Fragment implements DataFragment {
```

Nakon čega je potrebno dodati nedostajuću metodu `reloadData` (Alt+Enter) čiji kôd je:

```
@Override
public void reloadData() {
    recyclerView.setAdapter(new ActiveTasksRecyclerViewAdapter(DbTask.getAll(1),
getContext()));
}
```

Na isti način, sučelje treba implementirati u klasi `ActiveTasksFragment`, no ovoga puta u implementaciji, odnosno kod poziva `getAll` metode prosljeđuje se drugi parametar:

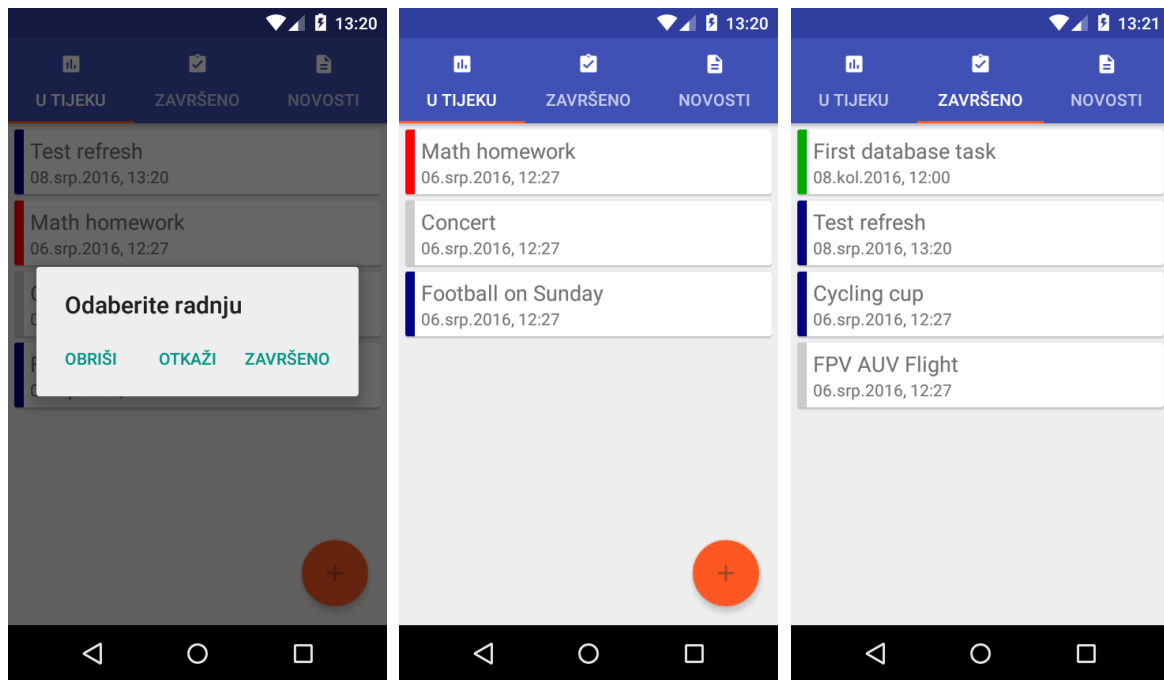
```
@Override
public void reloadData() {
    recyclerView.setAdapter(new ActiveTasksRecyclerViewAdapter(DbTask.getAll(0),
getContext()));
}
```

Za završetak implementacija osvježavanja sadržaja fragmenata koji prikazuju podatke iz mobilne baze podataka u klasi `MainActivity`, metodi `setupViewPager` na mjestu komentara „*implementacija osvježavanja lista*“, dodati kôd koji prolazi kroz sve tabove, odnosno fragmente, a nad onima koji implementiraju programsko sučelje za osvježavanje podataka, izvršiti metodu `reloadData`:

```
if(adapter.getItem(position) instanceof DataFragment){
    DataFragment selectedFragment = (DataFragment) adapter.getItem(position);
    selectedFragment.reloadData();
}
```

Mobilna aplikacija spremna je za testiranje. Pokrenuti aplikaciju i dodati novi zadatak naslovljen „Test refresh“ u bilo koje vrijeme i u bilo koju kategoriju, te ga potom označiti kao završenog. Nakon otvaranja drugog taba „Završeno“ upravo označeni zadatak „Test refresh“ će se pojaviti na listi, kao što je prikazano na slici (Slika 101).

Uz označavanje završenosti pojedinih zadataka, može se testirati i brisanje zadataka.



*Slika 101. Osvježavanjem podatkovnih fragmenata završeni zadaci se ispravno prikazuju*

## 5.7 Korištenje web servisa

Web aplikacija koja omogućuje unos vijesti koje se kasnije putem REST web servisa dohvaćaju u Memento mobilnoj aplikaciji prikazana je na slici (Slika 102). Ona predstavlja samo demo sustav putem kojeg se unosi naslov, datum, poruka vijesti te povezana slika te se zapisuje u bazu podataka na poslužitelju. Isto tako, ona sadrži REST web servis putem kojeg je te vijesti moguće dohvatiti u JSON obliku u mobilnoj aplikaciji.

**Nova vijest**

Naslov:

Datum:

Poruka:

Odaberi sliku...

Submit

[Prikaži kao JSON](#)

**Sve novosti**

Datum	Naslov	Poruka	Slika
2016-07-11 07:55:51	Obavijest o terminima upisa u 1. razrede	Obavještavamo učenike osmih razreda da će se upisi u prve razrede održavati u: ponedjeljak 11. srpnja 2016. od 12:00 do 15:00 sati utorak 12. srpnja 2016. od 8:00 do 14:00 sati srijeda 13. srpnja 2016. od 8:00 do 14:00 sati četvrtak 14. srpnja 2016. od 8:00 do 12:00 sati. Na upise je potrebno donijeti: za gimnaziju: upisnicu, svjedodžbe 7. i 8. razreda, ostale dokumente na temelju kojih su ostvareni...	 <a href="#">Obriši</a>

Slika 102. Korisničko sučelje jednostavne web aplikacije za unos novosti, s implementacijom REST web servisa

Struktura odgovora je prikazana u sljedećem paragrafu. Kao što je vidljivo, sastoji se od dva osnovna atributa, `count` koji sadrži vrijednost broja dohvaćenih rezultata (odnosno novosti), te atribut `results` koji nosi sadržaj pojedinih vijesti. Jedna vijest sastoji se od ranije spomenutih, `name` (naziva novosti), `text` (sadržaja vijesti), `date` (vremena objave), te `image_path` (što je tekst koji predstavlja putanju do slike na serveru, umjesto da je slika upisana direktno u bazu podataka).

```
{
  "count":3,
  "results":[
    {
      "name":"Obavijest o terminima upisa u 1. razrede",
      "text":"Obavještavamo učenike osmih razreda da će se upisi u prve razrede održavati u: ponedjeljak 11. srpnja 2016. od 12:00 do 15:00 sati utorak 12. srpnja 2016. od 8:00 do 14:00 sati srijeda 13. srpnja 2016. od 8:00 do 14:00 sati četvrtak 14. srpnja 2016. od 8:00 do 12:00 sati. Na upise je potrebno donijeti: za gimnaziju: upisnicu, svjedodžbe 7. i 8. razreda, ostale dokumente na temelju kojih su ostvareni...",

```

```

        "date": "2016-07-11 07:55:51",
        "image_path": "022e3217ea06751a1f86f6176c5591bd.jpg"
    },
    {
        "name": "Program za ispraćaj maturanata",
        "text": "U prilogu donosimo program ispraćaja maturanata koji će se održati
u utorak 28. lipnja s početkom u 18 sati.\r\n",
        "date": "2016-07-11 07:57:33",
        "image_path": "addae267482e9ab82bd5c25a5dc374cd.jpg"
    },
    {
        "name": "Učenici odličaši u školskoj godini 2015./2016. ",
        "text": "U prilogu donosimo popis učenika koji su u šk. godini 2015./2016.
postigli odličan uspjeh na kraju školske godine. Čestitamo učenicima!\r\n",
        "date": "2016-07-11 07:58:39",
        "image_path": "1ff07fe88bed0325cc32a93638e88e7c.jpg"
    }
]
}

```

Iste atribute treba sadržavati i lokalna klasa u koju će se zapisivati rezultati odgovora web servisa, odnosno prema kojoj će se obraditi JSON odgovor. Klasa `WsNewsItem` već sadrži sve potrebne elemente.

### 5.7.1 Priprema za ispis podataka

Prije samog početka rada sa web servisom, potrebno je pripremiti `NewsFragment` na način da se kreira novi `RecyclerView`, odgovarajući adapter te opiše element liste. Obzirom na to prvi korak je dodati novi prikaz u `res/layout` direktorij koji će opisivati izgled jedne dohvaćene vijesti i nazvati ga `news_item.xml`:

```

<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.CardView
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:id="@+id/cv_news"
android:layout_marginBottom="5dp"
android:clickable="true"
android:focusable="true"
android:foreground="?android:attr/selectableItemBackground">

<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:padding="5dp"
    android:weightSum="1">

    <ImageView
        android:layout_width="64dp"
        android:layout_height="64dp"
        android:id="@+id/rv_ni_image"
        android:paddingLeft="5dp" />

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:paddingLeft="5dp">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"

```

```

        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Medium Text"
        android:id="@+id/rv_ni_name" />

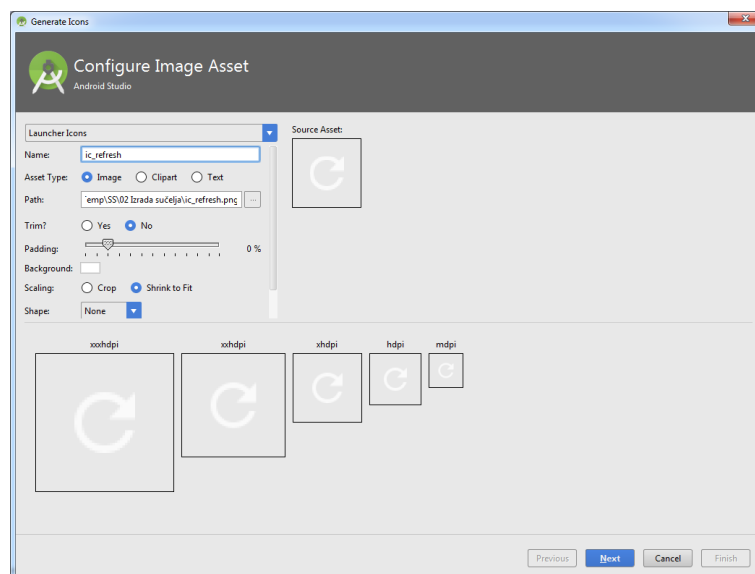
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:text="Small Text"
            android:id="@+id/rv_ni_date" />
    </LinearLayout>

</LinearLayout>

</android.support.v7.widget.CardView>

```

Kako će se vijesti dohvatiti samo kada korisnik to zatraži, `fragment_news.xml` će sadržavati i FAB koji će inicirati poziv web servisa i prikaz odgovora. Stoga, u projekt je potrebno dodati ikonu koju će FAB prikazivati. Kao i ranije, desnim klikom na `/res` odabrati opciju „New“, „Image asset“. U prikazanom dijalogu odabrati „Launcher icons“, Asset Type: „Image“, a nakon toga pronaći ikonu za osvježavanje. Naziv ikone treba postaviti na `ic_refresh`. Pod Shape kao i ranije odabrati „None“, nakon čega odabrati opciju „Next“ i na kraju „Save“. Vezani dijalog prikazan je na slici (Slika 103).



*Slika 103. Dodavanje ikone za osvježavanje*

Nakon prethodnog koraka, moguće je proširiti `fragment_news.xml` dodavanjem `RecyclerView`-a koji će prikazivati novosti te FAB-om za iniciranje web servisa sljedećim kôdom:

```

<?xml version="1.0" encoding="utf-8" ?>

<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"

```

```

        android:layout_height="match_parent"
        android:padding="5dp"
        android:id="@+id/newsLinearLayout">

        <android.support.v7.widget.RecyclerView
            android:id="@+id/rv_news_items"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />

    </LinearLayout>
    <android.support.design.widget.FloatingActionButton
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="16dp"
        android:clickable="true"
        android:src="@mipmap/ic_refresh"
        app:layout_anchor="@id/newsLinearLayout"
        app:layout_anchorGravity="bottom|right|end"/>
</android.support.design.widget.CoordinatorLayout>

```

Kao i ranije, za RecyclerView potrebno je kreirati novi adapter te ViewHolder. U paketu adapters stoga treba kreirati klasu NewsRecyclerViewAdapter, a u paket viewholders dodati klasu NewsViewHolder.

Klasu NewsViewHolder treba proširiti klasom RecyclerView.ViewHolder te povezati elemente grafičkog korisničkog sučelja s kôdom putem ButterKnife anotacija:

```

public class NewsViewHolder extends RecyclerView.ViewHolder {
    @Bind(R.id.rv_ni_name)
    public TextView tvName;

    @Bind(R.id.rv_ni_date)
    public TextView tvDate;

    @Bind(R.id.rv_ni_image)
    public ImageView tvImage;

    public NewsViewHolder(View itemView) {
        super(itemView);
        ButterKnife.bind(this, itemView);
    }
}

```

NewsRecyclerViewAdapter klasa se proširuje klasom RecyclerView.Adapter<NewsViewHolder>, nakon čega se nudi opcija automatskog dodavanja neimplementiranih metoda koji treba prihvatiti (klikom na ikonu žarulje ili kombinacijom na tipkovnici Alt+Enter). To će dati sljedeći rezultat:

```

public class NewsRecyclerViewAdapter extends RecyclerView.Adapter<NewsViewHolder>{
    @Override
    public NewsViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        return null;
    }

    @Override
    public void onBindViewHolder(NewsViewHolder holder, int position) {

    }

    @Override

```



```

public int getItemCount() {
    return 0;
}

```

Prije nego što je moguće potpuno implementirati generirane metode, potrebno je dodati referencu na novu biblioteku treće strane, Picasso koja će olakšati rad sa slikama dohvaćenim putem web servisa. U `build.gradle` (`Module:app`) dodati:

```

// rad sa slikama
compile 'com.squareup.picasso:picasso:2.5.2'

```

A za `NewsRecyclerViewAdapter` dodati sličan kôd kao i za zadatke. Potrebna je lista trenutnih vijesti, metoda za povezivanje sa odgovarajućim `ViewHolder`om `onCreateViewHolder`, te najvažnije, grafičkim elementima pridružiti naslov, datum i sliku u metodi `onBindViewHolder`. Sljedeći kôd prikazuje implementaciju `NewsRecyclerViewAdapter`-a:

```

public class NewsRecyclerViewAdapter extends RecyclerView.Adapter<NewsViewHolder> {

    ArrayList<WsNewsItem> newsItems;
    Context context;

    public NewsRecyclerViewAdapter(ArrayList<WsNewsItem> newsItems, Context
context){
        this.newsItems = newsItems;
        this.context = context;
    }

    @Override
    public NewsViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View v =
        LayoutInflater.from(parent.getContext()).inflate(R.layout.news_item, parent,
false);
        return new NewsViewHolder(v);
    }

    @Override
    public void onBindViewHolder(NewsViewHolder holder, int position) {

        SimpleDateFormat sdf = new SimpleDateFormat("dd.MMM.yyyy, HH:mm",
context.getResources().getConfiguration().locale);

        holder.tvName.setText(newsItems.get(position).getName());

        holder.tvDate.setText(sdf.format(newsItems.get(position).getDate().getTime()));

        // učitavanje slike sa weba

        Picasso.with(context).load(context.getResources().getString(R.string.image_ws) +
newsItems.get(position).getImage_path())
            .resize(64, 64)
            .centerCrop()
            .into(holder.tvImage);
    }

    @Override
    public int getItemCount() {
        return newsItems.size();
    }
}

```

### 5.7.2 Implementacija infrastrukture za rad sa web servisima

Za rad sa web servisima koriste se dvije temeljne biblioteke treće strane, Retrofit i GSON [37] [38]. Retrofit je HTTP klijent koji omogućuje definiranje ruta do web servisa te sigurno asinkrono pozivanje istih. U kombinaciji sa bibliotekom GSON za serijalizaciju JSON formatiranih podataka omogućuje vrlo elegantno korištenje REST web servisa, na način da olakšava definiranje putanja do web servisa, čitanje rezultata web servisa i upravljanje HTTP komunikacijom bez potrebe za direktnom implementacijom višenitnog rada.

Za korištenje navedenih biblioteka u `build.gradle` (Module:app) je potrebno dodati njih te njihove zavisnosti:

```
// retrofit and gson
compile 'com.google.code.gson:gson:2.4'
compile 'com.squareup.retrofit2:retrofit:2.0.2'
compile 'com.squareup.retrofit2:converter-gson:2.0.2'
compile "com.squareup.retrofit2:adapter-rxjava:2.0.2"
compile 'com.squareup.okhttp3:logging-interceptor:3.2.0'
```

Memento će čitavo vrijeme pristupati slikama dohvaćenima sa web servisa, pa kako ne bi čitavo vrijeme bilo potrebe za upisivanjem duge putanje do lokacije slika, u `strings.xml` dodati putanju, koja će se kasnije koristiti u programskom kôdu:

```
<string name="image_ws">http://www.poslužitelj.domena/aplikacija/img/</string>
```

Sljedeće što je potrebno jest dodati infrastrukturu za rad sa web servisima. Za tu svrhu, kreirati novi paket i nazvati ga `webservice`, a unutar istog kreirati dvije klase i jedno sučelje; klasa `WsDataLoader` će pozivati web servis, klasa `WsResponse` opisuje izgled odgovora koji će se prilagoditi lokalnoj klasi `WsNewsItem` i konačno sučelje `WsCaller` sadrži putanju i naziv metode web servisa koji se poziva.

WsResponse:

```
public class WsResponse {
    public int count;
    public ArrayList<WsNewsItem> results;

    public int getCount() {
        return count;
    }

    public void setCount(int count) {
        this.count = count;
    }

    public ArrayList<WsNewsItem> getResults() {
        return results;
    }

    public void setResults(ArrayList<WsNewsItem> results) {
        this.results = results;
    }
}
```

Važno je uočiti da je sadržaj ove klase jednak odgovoru s web servisa, a koristan sadržaj koji se prikazuje na fragmentu vijesti zapisan je u obliku liste `WsNewsItem` elemenata. Prilagodbu odgovora, odnosno serijalizaciju radi GSON biblioteka. Jako je važno da atributi klase `WsResponse` odgovaraju atributima sirovog JSON odgovora, a atributi klase `WsNewsItem` odgovaraju atributima `results` JSON odgovora jer u suprotnom serijalizaciju treba napraviti samostalno.

`WsCaller` sadrži programsko sučelje koje koristi Retrofit biblioteka. Tim sučeljem pomoću anotacije `@POST` koja odgovara POST poruci HTTP protokola definirana je ruta do web servisa koji ispisuje sve poruke. Od strane servera, ta metoda definirana je datotekom `json.php` (iako je češće definirana samom lokacijom, bez ekstenzije datoteke), a od strane mobilne aplikacije Memento, metoda je definirana nazivom `getAllNews`, što će Retrofit zapisati u klasu `WsResponse`.

WsCaller:

```
public interface WsCaller {
    @POST("/heureka/src/json.php")
    Call<WsResponse> getAllNews();
}
```

Konačno, najsloženija klasa je `WsDataLoader`. Koja sadrži samo dvije lokalne varijable, `BASE_URL` i `result` te dvije metode `loadDataFromWebService` i `getFetchedNewsArray`.

Varijabla `BASE_URL` sadrži putanju do korijena web servisa, a u varijablu `result` se zapisuje odgovor s web servisa, čim stigne. Naime, ovdje se koristi asinkroni pristup, što znači da nakon poziva web servisa, odnosno dugotrajne usluge, aplikacija nije blokirana.

Metoda `loadDataFromWebService` priprema GSON instancu, definira format datuma te ju pridružuje instanci Retrofita. Isto tako, poziva web servis te postavlja slušač događaja koji se okida nakon što stigne odgovor, odnosno metoda `onResponse` koja će u tijelu odgovora sadržavati i podatke, naravno ako je uspješna.

Metoda `getFetchedNewsArray` vraća trenutno zapisane podatke u lokalnoj varijabli `result`, koji se zapisuju u `onResponse` metodi.

`WsDataLoader`:

```
public class WsDataLoader {

    private String BASE_URL = "http://www.posluzitelj.domena/";
    private WsResponse result;

    public void loadDataFromWebService(final NewsFragment callerFragment){
        result = new WsResponse();

        Gson gson = new GsonBuilder()
            .setDateFormat("yyyy-MM-dd HH:mm") // format u JSON odgovoru
            .create();

        // kreiranje Retrofit objekta i povezivanje sa GSONom
        Retrofit retrofit = new Retrofit.Builder()
            .baseUrl(BASE_URL)
            .addConverterFactory(GsonConverterFactory.create(gson))
            .build();

        // kreiranje i poziv web servisa
        WsCaller serviceCaller = retrofit.create(WsCaller.class);
        Call<WsResponse> call = serviceCaller.getAllNews();
        // definiranje događanja nakon dolaska podataka
        call.enqueue(new Callback<WsResponse>() {
            @Override
            public void onResponse(Call<WsResponse> call, Response<WsResponse>
response) {
                if (response.isSuccessful()) {
                    result = response.body();
                    // JAVITI POZIVATELJU DA SU PODACI STIGLI...
                }
            }

            @Override
            public void onFailure(Call<WsResponse> call, Throwable t) {
                System.out.println("Something went wrong! ");
            }
        });
    }

    // metoda koja vraća trenutno dohvaćene podatke
    public ArrayList<WsNewsItem> getFetchedNewsArray(){
        if(result != null) {
            return result.getResults();
        }
    }
}
```

```

        else{
            return null;
        }
    }
}

```

Nakon što korisnik na fragmentu za prikaz vijesti pritisne FAB potrebno je pozvati web servis, dakle klasu NewsFragment treba proširiti tako da se najprije poveže sa odgovarajućim RecyclerView-om:

```

public class NewsFragment extends Fragment {

    @Bind(R.id.rv_news_items)
    RecyclerView recyclerView;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_news, container, false);
        ButterKnife.bind(this, rootView);
        return rootView;
    }

    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);

        recyclerView.setHasFixedSize(true);
        LinearLayoutManager llm = new LinearLayoutManager(getActivity());

        ArrayList<WsNewsItem> items = new ArrayList<>();
        recyclerView.setLayoutManager(llm);
    }
}

```

I na kraju, potrebno je kreirati instancu `WsDataLoader` klase, kojom će se pozvati web servis, te implementirati poziv pritiskom na FAB putem `ButterKnife` biblioteke. U `NewsFragment` dodati zatamnjene linije kôda:

```
public class NewsFragment extends Fragment {

    @Bind(R.id.rv_news_items)
    RecyclerView recyclerView;
    WsDataLoader dl;
    ProgressDialog progress;

    //WsDataLoader klasi prosljedi
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_news, container, false);
        ButterKnife.bind(this, rootView);
        dl = new WsDataLoader();
        return rootView;
    }

    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);

        recyclerView.setHasFixedSize(true);
        LinearLayoutManager llm = new LinearLayoutManager(getActivity());

        ArrayList<WsNewsItem> items = new ArrayList<>();
        recyclerView.setLayoutManager(llm);

        if (dl.getFetchedNewsArray() != null) {
            recyclerView.setAdapter(new
NewsRecyclerViewAdapter(dl.getFetchedNewsArray(), getContext()));
        }

        // nakon pritiska FAB-a podigni dialog za učitavanje s porukom
        // instanci WsDataLoader-a prosljedi referencu na sebe (za pristup metodi
showLoadedData)
        @OnClick(R.id.button)
        public void clicked(View v) {
            dl.loadDataFromWebService(this);
            progress = ProgressDialog.show(getActivity(), getString(R.string.loading),
getString(R.string.loading_message), true);
        }

        // postavi podatke (poziva se iz WsDataLoaderKlase)
        public void showLoadedData(ArrayList<WsNewsItem> newsItems) {
            recyclerView.setAdapter(new NewsRecyclerViewAdapter(newsItems,
getContext()));
            progress.dismiss();
        }
    }
}
```

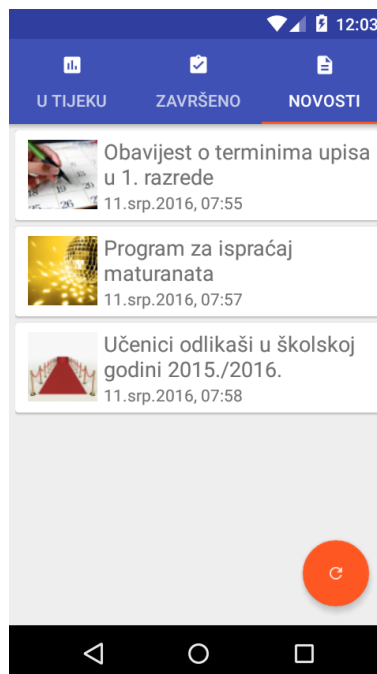
A kako bi se `NewsFragmentu` dojavilo da su podaci stigli, `WsDataLoader` već ima instancu fragmenta, samo što još preostaje je pozvati metodu kojom se prikazuju dohvaćene novosti. Kod `WsDataLoader.java` na mjesto "JAVITI POZIVATELJU DA SU PODACI STIGLI...": postaviti poziv metode za učitavanje podataka:

```
callerFragment.showLoadedData(response.body().getResults());
```

I za kraj, u `AndroidManifest.xml` potrebno je dodati, odnosno zatražiti korisnika za dozvolu korištenja Interneta prilikom instalacije. Prije taga `application` dodati:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Memento je spreman za pokretanja i testiranje. Navigacijom do taba „Novosti“ te pritiskom na FAB, novosti su uspješno dohvaćeni, a rezultat je prikazan na slici (Slika 104).



*Slika 104. Pregled novosti dohvaćenih putem web servisa*

Čime je demo verzija aplikacije završena te je otvorena za daljnje nadogradnje i promjene.

## 5.8 Pitanja za provjeru znanja

1. Za koje sve platforme je moguće razvijati Android aplikacije putem Android Studija?
2. Što je API level?
3. Koliko datoteka obično vežemo uz Android aktivnosti?
4. U kojoj se datoteci definiraju ovisnosti o vanjskim bibliotekama?
5. Unutar kojeg direktorija u strukturu Android aplikacije se nalaze XML datoteke sa postavkama, dizajnom, grafičkim korisničkim sučeljima, mjerama itd.?
6. Zašto Android Studio prilikom uvoza slike uvijek kreira više instanci?
7. Zašto se koriste Java paketi?
8. Što se omogućuje kvalifikatorima XML datoteka?
9. Čemu služi RecyclerView?
10. Putem koje temeljne klase se dohvaćaju resursi Android aplikacije?
11. Što je FAB?
12. Koja je razlika između TextView i TextEdit?
13. Na koji se način prevodi korisničko sučelje Android aplikacije na druge jezike?
14. Pojasnite pojam događaja u radu s Android korisničkim sučeljem i pojasnite čemu služi biblioteka Butterknife.
15. Što je Spinner?
16. Pojasnite što su dijalози.
17. Pojasnite i skicirajte sve elemente potrebne za rad sa RecyclerView-om.
18. Čemu služi programsko sučelje, u kontekstu događaja?
19. Što je XML i pojasnite njegovu strukturu?
20. Što je JSON i pojasnite njegovu strukturu?
21. Pojasnite čemu služe „rute“ kod Retrofit biblioteke.
22. Pojasnite što je Intent?
23. Objasnite razliku između aktivnosti i fragmenta.
24. Zašto je važan dizajn grafičkog korisničkog sučelja kod razvoja mobilnih aplikacija?
25. Komentirajte primjenu biblioteka treće strane naspram razvoja vlastite biblioteke.

## 5.9 Resursi za samostalan rad

- ✓ *Google Developers*, službena web stranica  
<https://developer.android.com/training/index.html>
- ✓ *Online tečajevi:*  
*Udacity Android Basics Nanodegree by Google, Android Developer Nanodegree by Google*  
<https://www.udacity.com/courses/android>



- ✓ *Kilobolt online, Uvod u razvoj igara s Androidom*  
[www.kilobolt.com/game-development-tutorial.html](http://www.kilobolt.com/game-development-tutorial.html)
- ✓ *YouTube kanal „AndroidDevelopers“*,  
<https://www.youtube.com/user/androiddevelopers>
- ✓ *YouTube video „Android Programming Tutorial 2015 | The Complete Tutorial to Learn Android“* <https://www.youtube.com/watch?v=FUbcoQ-8kRQ>



## RJEČNIK STRUČNIH POJMOVA

**Arhitekturni dizajn** (eng. architectural design): Niz aktivnosti koje se provode u svrhu definiranja arhitekture programskog proizvoda. Te aktivnosti su popraćene korištenjem odgovarajućih alata za izradu odgovarajućih dijagrama. Rezultat arhitekturnog dizajna su osnovne komponente sustava, njihove uloge i međusobne veze.

**Dizajn programskog proizvoda** (eng. software design): Niz aktivnosti koje se provode u svrhu definiranja unutarnje strukture, funkcionalnosti i ponašanja dijela programskog proizvoda (modula) ili programskog proizvoda u cjelini. Te aktivnosti su popraćene korištenjem odgovarajućih alata za izradu dijagrama i ostale dokumentacije. Rezultat faze dizajna programskog proizvoda su detaljno razrađene specifikacije funkcionalnosti, ponašanja i strukture svih gradbenih dijelova računalnog programa.

**Generalizacija:** Predstavlja proces grupiranja klasa objekata po promatranim zajedničkim i/ili istovjetnim odlikama. Generalizacijom se klase objekata grupiraju, a potom opisuju pomoću nadklasa.

**Integrirano razvojno okruženje** (eng. Integrated Development Environment): Predstavlja skup alata za provedbu svih aktivnosti u procesu razvoja, testiranja i objave programskih proizvoda. Napredna razvojna okruženja višestruko ubrzavaju proces razvoja.

**Internet stvari** (eng. Internet of Things): Predstavlja mrežu sačinjenu od fizičkih objekata (stvari, uređaja) stalno spojenih na Internet s kojeg mogu primiti i slati podatke. Najčešće se koristi kako bi se opisalo skup fizičkih uređaja koji imaju mogućnost prikupljati podatke (pomoću senzora) i slati ih u centralnu bazu ili pak prikazati rezultat njihove obrade.

**Internet svega** (eng. Internet of Everything): Predstavlja mrežu sačinjenu od fizičkih i nefizičkih objekata (stvari, uređaja, usluga, ljudi, računalnih agenata, ugrađenih uređaja...) stalno spojenih na Internet s kojeg mogu primiti i slati podatke. Ostale karakteristike su slične konceptu *Internet stvari*.

**Klasa objekata:** Predstavlja skup jedinki (entiteta) iz realnog svijeta koje imaju istovrsna svojstva i ponašanja. Na primjer: svi ljudi mogu biti opisani klasom „Čovjek“, dok sve životinje mogu biti opisane klasom „Životinja“. Ipak, i ljudi i neke životinje mogu imati istovjetna svojstva i ponašanja, što također možemo opisati klasom „Sisavac“.

**Ključna funkcionalnost** (eng. key feature): Predstavlja osnovnu funkcionalnost mobilne aplikacije oko koje se definiraju sve dodatne funkcionalnosti. Primjeri osnovnih funkcionalnosti mogu biti *evidencija osobne potrošnje* ili *podsjetnik za neobavljene zadatke*.

**Metodika razvoja programskog proizvoda** (eng. Software Development Methodology): predstavlja sistematiziran pristup u provedbi procesa razvoja programskog proizvoda. Metodika definira ključne faze i aktivnosti razvoja, ključne uloge i njihove odgovornosti te ključne artefakte (ulaze i izlaze) iz definiranih aktivnosti.

**Mobilna aplikacija** (eng. mobile application): Programski proizvod izrađen za bilo koju vrstu mobilnog uređaja (pametnog telefona, tableta, nosivog uređaja i slično, uređaja proširene stvarnosti...) koji ima korisničko sučelje. Mobilne aplikacije ne uključuju programske proizvode izrađene za ugrađene uređaje.

**Nasljeđivanje** (eng. Inheritance): Jedan od osnovnih objektno orijentiranih koncepata koji omogućuje specijaliziranim klasama naslijediti svojstva i metode generalizirane (nadređene) klase. U ispravnom objektno orijentiranom dizajnu jedna specijalizirana klasa može direktno naslijediti samo jednu nadklasu.

**Objekt**: Predstavlja pojavu (entitet) iz stvarnog svijeta. Objekt je instanca (jedinka) određene klase objekata. Na primjer: iz klase objekata „Čovjek“ možemo instancirati objekte kao što su „Marta“, „Igor“, „Emanuel“ ili „Tomislav“.

**Objektno orijentirani dizajn** (eng. Object oriented design - OOD): Predstavlja proces dizajna programskog proizvoda temeljenog na *objektu* kao osnovnom konceptu. Neki od alata korištenih u OOD-u su dijagram klase koji prikazuje klase objekata i njihove međusobne veze, dijagram objekata koji prikazuje veze između objekata, to jest instanci klase, dijagram paketa koji prikazuje klase grupirane u pakete i veze između paketa, dijagram slijeda, komunikacije i druge koji prikazuju interakciju između objekata i druge.

**Objektno orijentirano programiranje** (eng. Object oriented programming - OOP): Predstavlja proces implementacije programskog proizvoda primjenom objektno orijentiranih koncepata. OOP je slijedi nakon OOD-a, a osnovni koncepti koji se pri tome primjenjuju su *objekt*, *nasljeđivanje*, *učahurivanje* i *polimorfizam*.

**Pametni telefon** (eng. smart phone, smarhpone): Mobilni telefon koji osim osnovnih funkcionalnosti uspostave telefonskog poziva, može izvršavati računalni softver, te može autonomno prikupljati i obrađivati podatke, a rezultate obrade pohraniti, dijeliti ili dati korisniku na uvid.

**Pametni uređaj** (eng. smart device): Bilo koji uređaj koji autonomno prikuplja i obrađuje podatke, te ih pohranjuje, dijeli ili daje korisniku na uvid. Primjeri pametnih uređaja mogu biti mobilni telefoni, satovi, narukvice, naočale ili pak oprema za sport, zabavu, učenje i slično.

**Polimorfizam** / višeobličje (eng. Polymorphism): Kao jedan od osnovnih objektno orijentiranih koncepata omogućuje korištenje metoda naslijeđene nadklase ali i promijenjenih metoda u specifičnoj klasi. Isti objekt može imati više oblika, od kojih se svaki može koristiti uz naglasak na promjeni tipa objekta.

**Povezani uređaj** (eng. connected device): Uređaj koji se povremeno ili trajno povezuje na Internet ili drugu mrežu. Ovi uređaji mogu komunicirati sa poslužiteljima (serverima) putem interneta, ali, ovisno o modelu, mogu biti i direktno povezani i/ili izmjenjivati podatke.

**Proces razvoja programskog proizvoda** (eng. Software development process): Predstavlja proces ili niz aktivnosti koji pretvara korisničke zahtjeve u programski proizvod

**Proširena stvarnost** (eng. augmented reality): Zajednički naziv za skup tehnologija koje se koriste u svrhu kombiniranja elemente realne stvarnosti (slika, tekst, zvuk, video...) kako bi ih se nadopunilo i nadopunjene prezentiralo korisniku. Prezentacija se obično vrši kroz specifičan medij kao što su posebne naočale.

**Razvoj mobilnog softvera** / razvoj mobilnih aplikacija (eng. mobile software development): Predstavlja skup aktivnosti koje provode članovi projektnog tima u svrhu osmišljavanja, izrade, puštanja u rad te održavanja programskog proizvoda. Razvoj možemo promatrati kroz sljedeće osnovne faze: razrada projektne ideje i definiranje funkcionalnosti, dizajn izgleda proizvoda, dizajn arhitekture i strukture proizvoda, izrada (programiranje), testiranje, puštanje u rad i održavanje. Sve

navedene faze uključuju izradu dokumentacije te provedbu metodičkog (strukturiranog) pristupa u njihovoj provedbi.

**Refaktoriranje kôda:** Predstavlja aktivnost promjene programskog kôda bez promjene funkcionalnosti. Najčešće se provodi kako bi se programski kôd očistio, optimizirao, učinio jednostavnijim za čitanje i nadogradnju.

**Set za razvoj aplikacija** (eng. Software Development Kit - SDK): Predstavlja set klasa i osnovnih alata ciljano kreiranih za razvoj određenih aplikacija. Na primjer *Android SDK* donosi sve osnovne klase i alate koji u suradnji sa integriranim razvojnim okruženjem čine minimum potreban za razvoj Android aplikacija.

**Specijalizacija:** Predstavlja proces izdvajanja specifičnih (različitih) klasa objekata po promatranim odlikama iz generalizirane nadklase koja sadrži zajednička svojstva. Specijalizirana klasa nasljeđuje sva svojstva i metode nadklase, a potom ih može ili ne mora izmijeniti ili nadopuniti.

**Strukturalni dizajn / dizajn strukture** (eng. structural design): vidi *Dizajn programskog proizvoda*.

**SWEBOK** (eng. Software Engineering Body of Knowledge): međunarodni ISO/IEC standard koji predstavlja globalno prihvaćeni korpus znanja (eng. body of knowledge) u području softverskog inženjerstva. Kreiran je suradnjom više stručnih udruženja i predstavnika industrije te je objavljen pod okriljem IEEE organizacije. Trenutno se priprema nova verzija ovog dokumenta.

**Učahurivanje** (eng. Encapsulation): Kao jedan od osnovnih objektno orijentiranih koncepata predstavlja mogućnost objekta da objedini podatke (o objektu) i metode izmjene tih podataka (ponašanje objekta). Učahurivanje omogućuje skrivanje podataka i implementaciju metoda od drugih objekata, ali i pristup isključivo pomoću unaprijed definiranog sučelja (javnih svojstava ili metoda).

**Ugrađeni uređaj** (eng. embedded device): Uređaj bez korisničkog sučelja koji najčešće ima vlastiti mikroprocesor, te ima ulogu prikupljati, obrađivati i distribuirati podatke unutar većeg sustava u koji je ugrađen.

**Virtualna stvarnost** (eng. Virtual reality): Zajednički naziv za skup tehnologija koje se koriste u svrhu zamjene elemenata realne stvarnosti (sliku, tekst, zvuk, video...) koji potom kroz specifičan medij prezentiraju korisniku. Na primjer, noseći posebne naočale korisnik može „oko sebe“ vidjeti grad u kojem se fizički ne nalazi.

**Vodeća funkcionalnost** (eng. Kill feature): Predstavlja najvažniju funkcionalnost mobilnog programskog proizvoda koju nemaju drugi proizvodi na tržištu. Oko ove funkcionalnosti, koja u direktnom prijevodu s engleskog jezika znači „funkcionalnost koja ubija“, se obično gradi marketinški plan i plan promocije proizvoda kako bi se naglasila njegova najvažnija prednost u odnosu na ostale proizvode.



## POPIS KRATICA I AKRONIMA

ADB – Most za ispravljanje pogrešaka u aplikacijama za Android (eng. Android Debug Bridge)

API – Aplikacijskom programibilno sučelje (eng. Application Programming Interface)

ART – Izvršne datoteke za Android (eng. Android Runtime)

AVD – Virtualni uređaj za Android (eng. Android Virtual Device)

IDE – Integrirano razvojno okruženje (eng. Integrated Development Environment)

IKT – Informacijko-komunikacijske tehnologije

IoT – Internet stvari (eng. Internet of Things)

IoE – Internet svega (eng. Internet of Everything)

IT – Informacijske tehnologije

NFC – Kratkodometna bežična komunikacija (eng. Near Field Communication)

OCR – Optičko prepoznavanje znakova (eng. Optical Character Recognition)

OO – Objektno orijentirano

OOD – Objektno orijentirani dizajn

OOP – Objektno orijentirano programiranje

SDK – Set za razvoj aplikacija (eng. Software Development Kit)

SDLC – Životni ciklus razvoja sustava (eng. Systems Development Life Cycle)

SWEBOK – Korpus znanja iz softverskog inženjerstva (eng. Software Engineering Body of Knowledge)

XML – Proširivi opisni jezik (eng. Extensible Markup Language)





## KORIŠTENA LITERATURA

- [1] "IEEE Standard Glossary of Software Engineering Terminology (610.12-1990)", IEEE Std 610.121990, 1990.
- [2] "Guide to the software engineering body of knowledge (SWEBOK V3) - Software engineering models and methods (Chapter 10 - Unpublished - In Review)", Technical report, 2012.
- [3] Centers for Medicare and Medicaid Services (CMS), Office of information Services, "Selecting a development approach". 2008.
- [4] G. Elliott, *Global business information technology: an integrated systems approach*. Harlow, England; New York: Pearson Addison Wesley, 2004.
- [5] "Guide to the software engineering body of knowledge 2004 version: SWEBOK", Los Alamitos, CA, Technical report ISO/IEC TR 19759, 2004.
- [6] K. Schwaber and J. Sutherland, "The Scrum Guide - The definitive guide to Scrum: The rules of the game". Scrum.org, 2011.
- [7] D. Wells, "Extreme Programming: A Gentle Introduction", 2009. [Online]. Dostupno na: <http://www.extremeprogramming.org/>. [Accessed: 22-Apr-2016].
- [8] J. Sutherland, "Jeff Sutherland's Scrum Handbook", Scrum Training Institute Press, 2010.
- [9] C. Murphy, "Adaptive Project Management Using Scrum", *Methods Tools*, vol. 12, no. 4, pp. 10–22, 2004.
- [10] Y. D. Liang, *Introduction to JAVA Programming - Comprehensive Version*, 9. izdanje. Boston: Pearson, 2013.
- [11] IEEE Computer Society, *IEEE Standard for Floating-Point Arithmetic*. New York, NY, USA: IEEE, 2008.
- [12] C. Lin, "Floating Point". University of Maryland, 2003.
- [13] H. Schmidt, *IEEE 754 Converter*. Bonn, Germany.
- [14] B. Kurniawan, "Introduction to OOP in VB.NET", *Windowsdevcenter.com*, 2002. .
- [15] M. Mullin, *Object Oriented Program Design*. Massachusetts: Addison-Wesley, 1989.
- [16] I. Pokec, "Usporedba razvojnih okruženja za razvoj android mobilnih aplikacija", Diplomski rad, Sveučilište u Zagrebu, Fakultet organizacije i informatike Varaždin, Varaždin, Hrvatska, 2016.
- [17] Google Inc. "Android Studio - The Official IDE for Android", *Android Developers*, 2016. [Online]. Dostupno na: <https://developer.android.com/studio/index.html>.
- [18] Google Inc. "Platform Architecture", *Android Developers*, 2016. [Online]. Dostupno na: <https://developer.android.com/guide/platform/index.html>.
- [19] Google Inc. "Activities", *Android Developers API Guides*, 2016. [Online]. Dostupno na: <https://developer.android.com/guide/components/activities.html>.
- [20] Google Inc. "Managing the Activity Lifecycle", *Android Developers Training*, 2016. [Online]. Dostupno na: <https://developer.android.com/training/basics/activity-lifecycle/index.html>.
- [21] Google Inc. "App Manifest", *Android Developers API Guides*, 2016. [Online]. Dostupno na: <https://developer.android.com/guide/topics/manifest/manifest-intro.html>.
- [22] Gradle Inc. "Gradle Build Tool - Polyglot build automation system", *Gradle.org*, 2016. [Online]. Dostupno na: <https://gradle.org/>.
- [23] T. Berglund and M. McCullough, *Building and testing with Gradle*. Sebastopol, CA: O'Reilly Media, Inc, 2011.
- [24] M. McCullough and T. Berglund, *Gradle beyond the basics: [customizing next-generation builds]*. Beijing: O'Reilly, 2013.
- [25] K. Kousen, *Gradle recipes for Android: master the new build system for Android*. 2016.
- [26] Google Inc. "Storage Options", *Android Developers API Guides*, 2016. [Online]. Dostupno na: <https://developer.android.com/guide/topics/data/data-storage.html>.
- [27] Z. R. Mednieks, Ed., *Programming Android*, 1st ed. Sebastopol, Calif: O'Reilly, 2011.

- [28] Google Inc. "Settings", *Android Developers API Guides*, 2016. [Online]. Dostupno na: <https://developer.android.com/guide/topics/ui/settings.html>.
- [29] D. R. Hipp, "About SQLite", *SQLite.org*, 2016. [Online]. Dostupno na: <https://www.sqlite.org/about.html>.
- [30] R. Manger, *Baze podataka*, 1. izdanje. Element d.o.o, 2012.
- [31] C. J. Date, *An introduction to database systems*, 8. izdanje. Boston: Pearson/Addison Wesley, 2004.
- [32] Google Inc. "Saving Data in SQL Databases", *Android Developers Training*, 2016. [Online]. Dostupno na: <https://developer.android.com/training/basics/data-storage/databases.html>.
- [33] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, Eds., *Extensible Markup Language (XML) 1.0*, 5. izdanje. W3C Consortium, 2008.
- [34] ECMA International, *Standard ECMA-404 The Json Data Interchange Format*, 1. izdanje. Geneva: ECMA International, 2013.
- [35] T. W3C XML Protocol Working Group, *SOAP Specifications*. W3 Consortium, 2007.
- [36] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard, Eds., *Web Services Architecture*. W3 Consortium, 2004.
- [37] I. Square, "Retrofit - A type-safe HTTP client for Android and Java", 2013. [Online]. Dostupno na: <http://square.github.io/retrofit/>.
- [38] ..., "Gson Git - A Java serialization/deserialization library that can convert Java Objects into JSON and back", *Github.com*, 2016. [Online]. Dostupno na: <https://github.com/google/gson>.
- [39] Google Inc. "Klasa CoordinatorLayout", *Android Developers*, 2016. [Online]. Dostupno na: <https://developer.android.com/reference/android/support/design/widget/CoordinatorLayout.html>.
- [40] Google Inc. "Material Design - Introduction", *Material Design*, 2016. [Online]. Dostupno na: <https://material.google.com/>.
- [41] Google Inc. "Klasa AppBarLayout", *Android Developers*, 2016. [Online]. Dostupno na: <https://developer.android.com/reference/android/support/design/widget/AppBarLayout.html>.
- [42] Google Inc. "Klasa TabLayout", *Android Developers*, 2016. [Online]. Dostupno na: <https://developer.android.com/reference/android/support/design/widget/TabLayout.html>.
- [43] Google Inc. "Using ViewPager for Screen Slides", *Android Developers Training*, 2016. [Online]. Dostupno na: <https://developer.android.com/training/animation/screen-slide.html>.
- [44] Google Inc. "Klasa RecyclerView", *Android Developers*, 2016. [Online]. Dostupno na: <https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html>.
- [45] Google Inc. "Creating Lists and Cards - Create Cards", *Android Developers Training*, 2016. [Online]. Dostupno na: <https://developer.android.com/training/material/lists-cards.html#CardView>.
- [46] Google Inc. "Adapter Interface", *Android Developers*, 2016. [Online]. Dostupno na: <https://developer.android.com/reference/android/widget/Adapter.html>.
- [47] Google Inc. "Apstraktna klasa FragmentStatePagerAdapter", *Android Developers*, 2016. [Online]. Dostupno na: <https://developer.android.com/reference/android/support/v13/app/FragmentStatePagerAdapter.html>.
- [48] Google Inc. "Providing Resources", *Android Developers API Guides*, 2016. [Online]. Dostupno na: <https://developer.android.com/guide/topics/resources/providing-resources.html>.