SVEUČILIŠTE U ZAGREBU

FILOZOFSKI FAKULTET

ODSJEK ZA INFORMACIJSKE I KOMUNIKACIJSKE ZNANOSTI

Vladimir Bralić

Diplomski rad

Lokacijski servisi na Android uređajima

Mentor:

prof. dr. sc. Vladimir Mateljan

Zagreb, lipanj 2016.

Sažetak:

Ovaj rad opisuje funkcionalnosti i načine implementacije lokacijskih servisa (Google maps API, Location manager i Geocoder) na mobilnim Android uređajima. Rad pokušava obraniti tezu da su javno dostupni lokacijski servisi dovoljno precizni i jednostavni za implementaciju te da omogućavaju brz i jednostavan razvoj aplikacija koje se na njih oslanjanju. U svrhu obrane ove teze rad prikazuje aplikaciju koja se oslanja na lokacijske servise za prepoznavanje gradske ulice u kojoj se uređaj nalazi i omogućava automatsko slanje SMS poruke za plaćanje parkinga. Rad detaljno prikazauje implementaciju lokacijskih servisa, servisa za slanje SMS poruka i ostale programerske tehnike korištene pri izradi aplikacije i njenog korisničkog sučelja na Android mobilnom uređaju.

Ključne riječi: Android, mobilne aplikacije, razvoj aplikacija, lokacijski servisi, GPS, SMS poruke.

Abstract:

This paper describes functionalities and implementation methods of location services (Google maps API, Location manager, Geocoder) on mobile Android devices. It attempts to defend the thesis that publicly available location services are sufficiently accurate and implementation firendly to allow rapid and simple development of applications which rely on these services. To achive this goal the paper presents a mobile application which allows automatic sending of parking payment text messages. It describes, in detail, implementation of location services, SMS services and other programming techniques used in the development of the application and its user interface on an Android mobile device.

Keywords: Android, mobile applications, application development, location services, GPS, SMS messages.

Sadržaj

1. Uvod

Cilj ovog rada je obraniti tezu da su javno dostupni, besplatni lokacijski servisni na suvremenim mobilnim uređajima, konkretno Android mobilnim telefonima i tabletima, dovoljno precizni i jednostavni za uporabu te da omogućavaju rapidni razvoj aplikacija koje se oslanjaju na ove servise. Rad će detaljno opisati načine rada lokacijskih servisa i tehnike implementacije lokacijskih servisa u Android aplikacijama. U svrhu opisa implementacije autor rada je napisao Android aplikaciju koja se oslanja na ove servise. Veći dio izvornog koda ove aplikacije je prikazan i objašnjen u radu.

Cilj aplikacije je omogućiti jednostavnu naplatu parkinga u područjima koja omogućavaju plaćanje parkinga SMSom (engl. text message) porukama. Ovaj postupak poznat je svim suvremenim vozačima i sastoji se od utvrđivanja trenutne zone naplate (potragom za najbližom fizičkom oznakom zone –tablom), utvrđivanja telefonskog broj na kojem se vrši naplata (čita se sa iste table), te na kraju slanjem SMS poruke koja sadrži registarsku tablicu automobila na telefonski broj za naplatu parkinga u konkretnoj zoni. Aplikacija koja ima za cilj olakšati ovaj postupak mora prepoznati trenutni geografski položaj korisnika, povezati položaj korisnika sa ulicom u kojoj se naplaćuje parking te na kraju sama poslati SMS poruku na telefonski broj koji služi za naplatu u konkretnoj zoni.

Prvi dio rada upoznati će čitatelja sa nekim specifičnostima mobilnih aplikacija u odnosu na klasične desktop i web aplikacije te sa Android operacijskim sustavom i metodama razvoja istih, to jest sa osnovama objektnog programiranja u Javi. Nakon predstavljanja Android mobilnih uređaja predstavljene su metode utvrđivanja geografskom položaja, GPS i utvrđivanje položaja prepoznavanjem okolnih Wi-Fi mreža i komunikacijskih tornjeva. U posljednjem i najvećem dijelu rada opisuje se konkretna implementacija ranije raspravljenih tema kroz aplikaciju za plaćanje parkinga – SmsParking.

2. Mobilne aplikacije

Prije nego što rad prijeđe na tehničku problematiku razvoja sustava za geografsku lokaciju treba ukratko opisati značajke mobilnih aplikacija. To je tema ovog poglavlja.

Jedna od najpopularnijih zanimljivosti koju ljudi vole iznositi je ta, da današnji mobilni uređaji imaju više procesorske snage nego računala koja su ljude dovela na mjesec, a da ih koristimo za rasprave s ljudima na Internetu i beskrajne autoportrete. Dok je, naravno, istina da su današnji mobilni uređaji znatno „jači“ nego što su to bila čak i računala prije, recimo desetak godina, ono što se često zaboravlja je da ti mobilni uređaji služe svrhama koje prije 10 godina nismo mogli ni zamisliti, i to upravo zahvaljujući mobilnim aplikacijama.

No kako najbolje objasniti mobilne aplikacije? To su mali specijalizirani programi koje pokrećemo na mobilnim uređajima. Za razliku od desktop i, posebno, web aplikacija,mobilne aplikacije često pružaju lokaliziranu i izoliranu funkcionalnost.

Jednostavnost i specijaliziranost je upravo ono što je pridonijelo popularnosti mobilnih aplikacija i uređaja koji ih pogone. Većina „pametnih“ mobilnih uređaja već dolazi s nekolicinom predinstaliranih

aplikacija koje omogućuju korisniku lako i brzo snalaženje na uređaju, i obično uz to nude nekolicinu dodatnih sadržaja kao što su kalkulator, vrijeme, GPS navigaciju i slično. Uz te svoje aplikacije, većina mobilnih proizvođača također nudi i svoj vlastiti virtualni dućan, gdje korisnici mogu kupovati i dodatne aplikacije. iTunes, Appleov iPhone dućan, ima vrlo rigorozna pravila oko toga kakve su točno aplikacije dostupne, i prije nego ih naprave dostupnima prolaze vrlo rigorozna testiranja i provjere kako bi se osiguralo da se aplikacije koje je razvila neka treća strana pridržavaju svih Appleovih pravila i smjernica. To isto vrijedi i za Windows phone store, dok Androidov Google Play store također ima svoja pravila i smjernice, no generalno se smatra puno opuštenijim i otvorenijim prema novim idejama svojih partnera. Uz ove naravno postoji još nekolicina drugih, manjih dućana aplikacijama koji su vlasništvo respektivnih developera tih mobilnih operativnih sustava, od kojih svaki ima neka svoja ograničenja, pravila i politike.

Količina mobilnih aplikacija u virtualnim dućanima ponekad otežava snalaženje i pronalazak točno određene aplikacije. Na iTunesima se u prosjeku pojavljuje oko 4000 novih aplikacija dnevno, prema podacima iz 2015. godine. U prosjeku se 2015. preuzelo 25 milijardi mobilnih aplikacija na iOSu i 25 milijardi na Androidu, od kojih se najviše preuzimalo aplikacije koje olakšavaju sudjelovanje u društvenim mrežama. Uz razne društvene mreže, među najpopularnijim aplikacijama na svim platformama nalaze se mobilne igre, a iza njih su poslovne, obrazovne i takozvane „lifestyle“ aplikacije[1].

Sadržaj i količina nisu jedino što se mijenjalo s vremenom. Kako su pametni telefoni postajali sve bolji i brži, tako su i developeri mogli razvijat i sve veće i kompleksnije aplikacije. Od inicijalno dopuštene maksimalne veličine od 23 MB (60 za igre) u 2011, danas je ta brojka otprilike tri puta vreća za igre, i dva puta veća za ostale aplikacije.

No s velikim brojem malih aplikacija koje se preuzimaju na mobilne uređaje pojavilo se i pitanje mobilne sigurnosti. To je posebno točno za tvrtke i korporacije koje su usvojile „bring your own device“ politiku gdje zaposlenici koriste svoje vlastite, privatne uređaje u poslovne svrhe. Preuzimanjem neprovjerenih sadržaja i aplikacija, korisnici su izloženi industrijskoj špijunaži ili krađi podataka. Posljedica toga nije samo gubitak povjerljivih informacija, već izlaže tvrtke napadima i ucjenama. Kao rezultat te prijetnje pojavile su se dvije mogućnosti zaštite korisnika. „App wrapping“ koristi dinamičku biblioteku koja kontrolira određene dijelove aplikacije. Nažalost, ta metoda osiguranja spada pod takozvanu sivu legalnu zonu (budući da može u ime sigurnosti, između ostalog, prisiljavati korisnike da koriste VPN treće strane), i kao alternativu „Enterprise mobility management“, set sistema koji sprječavaju neovlašten pristup informacijama uporabom raznih enkripcija, lozinki, ili u krajnjem slučaju, mogućnošću udaljenog brisanja podataka.

3. Mobilni uređaji bazirani na Android operativnom sustavu

Android je operativni sustav, prvenstveno pisan za mobilne uređaje. Malo poznati, službeni naziv Androida je Open Handset Distribution (OHD). Ovaj naziv naslijeđen je organizacije koja ga danas razvija i održava - Open Handset Alliance. Open Handset Alliance je organizacija softverskih kompanija (Google, eBay itd.), proizvođača poluvodiča (ARM, Intel itd.), mobilnih operatera, proizvođača mobilnih uređaja (npr. HTC, Acer, Garmin, LG, Samsung itd.) i komercijalizacijskih kompanija čiji je cilj

[1] Usp. The statistics portal http://www.statista.com/statistics/270291/popular-categories-in-the-app-store/

izgradnja boljih mobilnih uređaja i razvoj otvorenih sustava na kojima će se ovi uređaji bazirati[2]. Bez sumnje najveći uspjeh ove organizacije je razvoj Android sustava. Izvorno Android je izrađen od kompanije Android Inc., koju je Google kupio 2005. Sustav je prezentiran javnosti 2007. kada je i oformljen Open Handset Alliance. Sam Android Inc osnovali su 2003. od strane Andy Rubin, Rich Miner, Nick Sears i Chris White.

Minimalni hardverski uvjeti koje mora zadovoljiti mobilni uređaji da bi pokrenuo Android navedeni su u tablici 1. Kao što je vidljivo iz tablice ovi uvjeti su zaista niski i daleko od performansi suvremenih mobilnih uređaja koji svojim karakteristikama ponekad mogu konkurirati stolnim računalima.

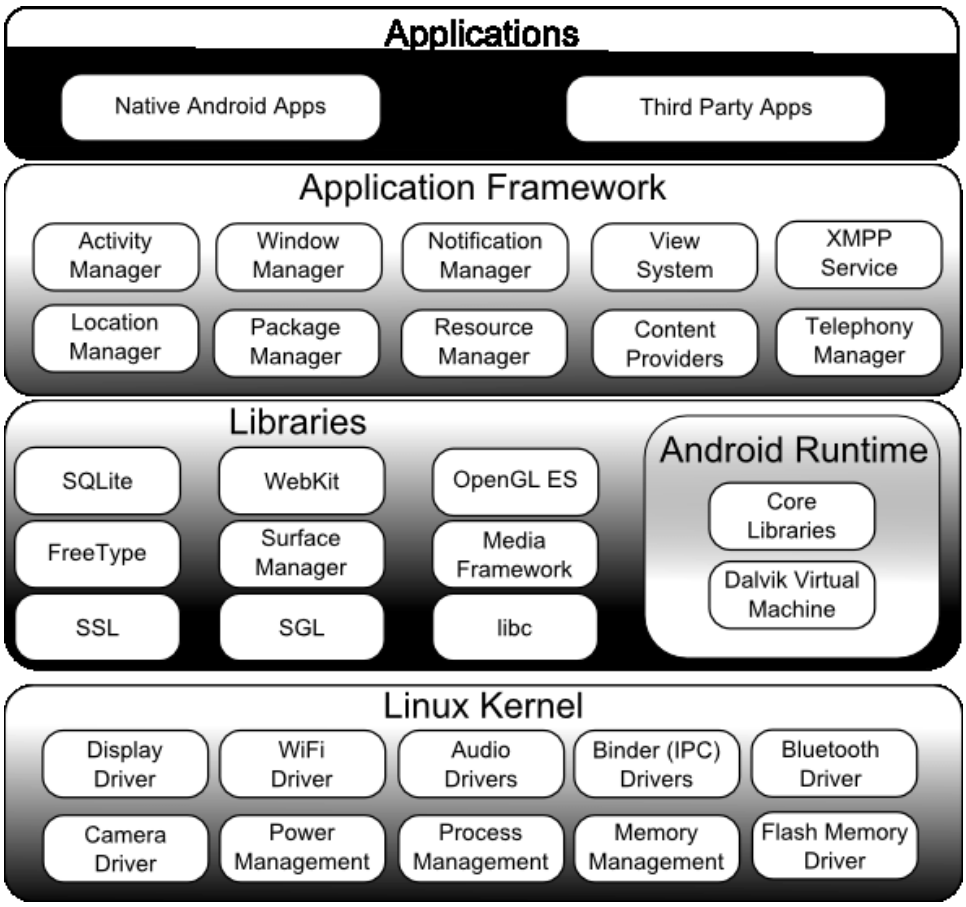
Osobina	**Minimalni zahtjev**
Chipset	ARM
Memorija	128 MB RAM; 256 MB vanjske flash memorije
Sustav pohrane	Mini ili Micro SD
Primarni zaslon za prikaz	QVGA TFT LCD; 16-bitna dubina boje
Upravljanje	5 navigacijskih tipki sa 5 aplikacijskih tipki; posebna tipka za paljenje uređaja, kamere i kontrolu glasnoće (dvije tipke)
Kamera	2 megapiksela CMOS
USB	mini-B USB
Bluetooth	1.2

Tablica 1 Minimalni hardverski zahtjevi za Android uređaje. Izvor: Android Platform Development Kit[3]

3.1 Android operativni sustav

Operativni sustav Android strukturian je slojevito. Na najnižem sloju nalazimo jezgru sustava. Idući sloj obuhvaća runtime komponente. Treći sloj sadržava okvir (engl. framework) koji olakšava izvršenje aplikacija. U ovom sloju nalazimo mnoge važne komponente poput Location Managera, Android klase kojom se veći dio ovog rada i bavi. Zadnji, četvrti sloj odnosi se na same aplikacije. Njega možemo podijeliti na nativne aplikacije koje se distribuiraju sa operativnim sustavom i aplikacije koje su razvili drugi pojedinci, kompanije ili ustanove (engl. third party apps). Kako je Android otvoreni sustav pojavile su i izvedene inačice (engl. Fork) proizvođača koji se iz raznih razloga ne slažu uvjetima korištenja Androida. Vjerojatno najpoznatija i najraširenija inačica Androida je Amazonov Fire OS (operativni sutav korišten na Kindle Fire tabletima).

[2] Usp. Open handset alliance: http://www.openhandsetalliance.com/oha_overview.html
[3] Usp. Android Platform Development Kit: http://www.netmite.com/android/mydroid/development/pdk/docs/system_requirements.html



Slika 1 Arhitektura Android operacijskog sustava. Izvor: Android Studio 2 Development Essentials Book

Jezgra Android operativnog sustava je Linux Kernel. U jezgri nalazimo i upravljački softver (engl. driver) za hardverske komponente Android uređaja, sustave za upravljanje memorijom i procesima.

Dalvik virtualni stroj (engl. Dalvik virtual machine) izvodi kod na Android uređajima. U odnosu na Javin virtualni stroj Dalvik je optimiziran za izvođenje sa minimalnim memorijskim otiskom (engl. memory footprint)[4].

3.2 Razvoj aplikacija za Android – Java

Android aplikacije se razvijaju u posebnoj verziji programskog jeziku Java. Java je objektno orijentirani programski jezik treće generacije. Razlika između "Android" Jave, to jest između Android API-a i Java API-a je u načinu izvođenja koda. Uobičajeni Java kod se izvodi na Java virtualnom stroju (engl. Java virtual Machine) dok se Android API kod izvodi na Dalvik virtualnom stroju (engl. Dalvik virtual machine) (povučen iz uporabe) ili ART virtualnom stroju (engl. ART virtual machine). Pri samom pisanju koda nema značajnih razlika te možemo reći da su programerske vještine stečene u Javi u potpunosti primjenjive i na razvoj aplikacija za Android. U idućim poglavljima ukratko ćemo objasniti glavne karakteristike to jest koncepte objektno orijentiranog programiranja (objekti, klase, polimorfizam,

[4] Usp. David Ehringer the dalvik virtual machine architecture

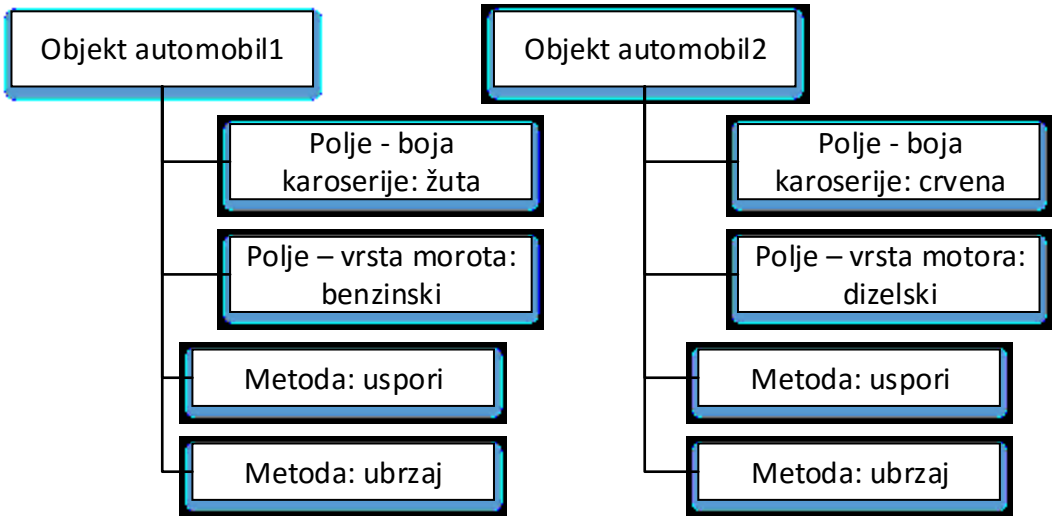
nasljeđivanje, overriding, apstrakcija, sučelje – engl. interface i paketi) te nakon njih neke specifičnosti vezane uz razvoj aplikacija za Android (aktivnosti, pružatelji sadržaja, intenti i servisi).

3.2.1 Objekti

Objekt je osnovni koncept objektno orijentiranog programiranja. U stvarnosti objekti su bilo koji jedinstveni predmeti ili bića. Na primjer automobil može objekt, kuća može biti objekt i neki konkretni čovjek, također, može biti objekt. Objekti su opisani stanjima to jest atributima i ponašanjima (engl. behaviours) to jest metodama.

Atribute (stanja) često zovemo i polja ili varijable i oni označavaju osobine objekta na primjer objekt automobil može imati atribut "boja karoserije" koji je u stanju "crveno" ili "plavo". Drugi atribut ovog objekta može biti "tip motora", "dizelski" ili "benzinski".

Metode (ponašanja) su aktivni postupci koje objekt može izvršiti. Iz ranijeg primjera, objekt automobil mogao bi imati metode "pokreni motor", "ubrzaj", "uspori" i "ugasi motor".[5] Primjer objekta prikazan je na slici niže.

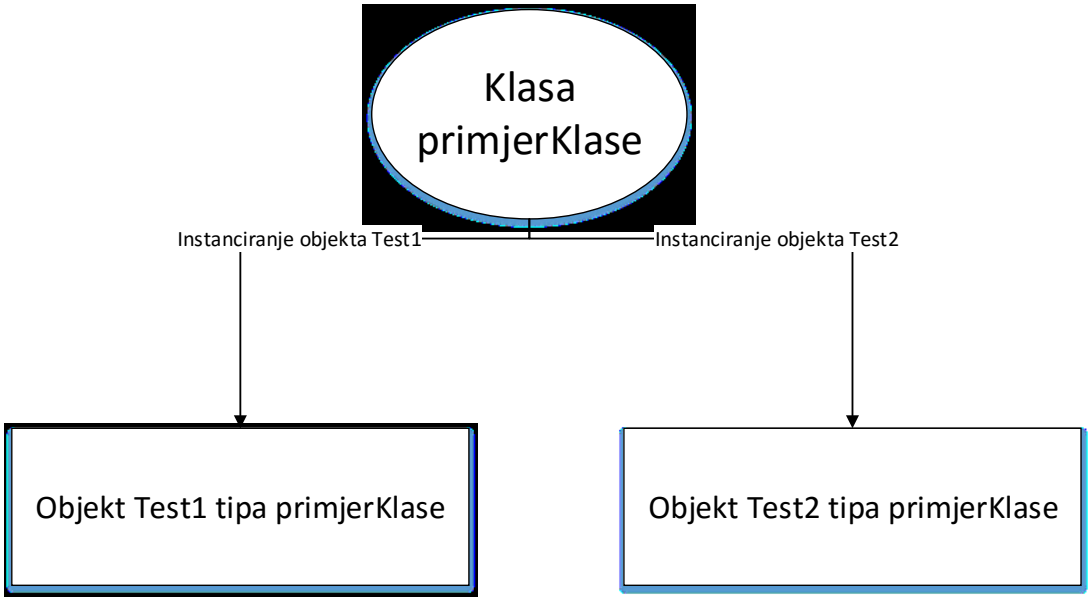


Slika 2 Objekti sa poljima i metodama. Izvor: vlastiti rad autora

3.2.2 Klase

Klasu obično opisujemo kao nacrt objekta. Sukladno ranijem primjeru mogli bi govoriti o Opel Corsa objektu klase Automobil i Fiat Tipo objektu klase Automobil koji se razlikuju po sadržaju nekih atributa.[6] Objekti iste klase se mogu razlikovati samo po sadržaju atributa (polja, varijabli) a ne i po njihovom tipu ili po metodama klase. Ipak stanovite promjene su moguće kroz neke koncepte koji će biti opisani u kasnijim poglavljima (nasljeđivanje, polimorfizam, overriding).

[5] Usp. Oracle Java Documentation Object Oriented Programming Concepts: https://docs.oracle.com/javase/tutorial/java/concepts/
[6] Isto.



Slika 3 Instanciranje objekata iz klase. Izvor: vlastiti rad autora

U Javi bi klasu Automobil mogli implementirati sljedećim kodom:

```
class Automobil {

	string boja;
	string vrstaMotora;
	int brzina = 0;

	void ubrzaj(int promjenaBrzine) {
	brzina = brzina + promjenaBrzine;
	}

	void uspori(int promjenaBrzine) {
	brzina = brzina - promjenaBrzine;
	}

	void pregledBrzine() {
		System.out.println("Brzina:" + brzina + " m/s.";
	}
}
```

Iz ovakvog nacrta, to jest klase možemo stvoriti nove objekte. "Stvaranje" objekta iz klase obično zovemo instanciranje. Primjer instanciranja dva objekta klase automobil je prikazan niže.

```
	Automobil OpelCorsa = new Automobil ();
	Automobil FiatPunto = new Automobil ();
```

Nakon instanciranja ovih objekta možemo pozvati njihove metode i vidjeti ponašanje objekata.

```
OpelCorsa.ubrzaj(10);
FiatPunto.ubrzaj(30);
FiatPunto.uspori(10);

FiatPunto.pregledBrzine();
```

Izlaz ovog koda bi bio:

```
Brzina: 20 m/s.
```

3.2.3 Nasljeđivanje

Nasljeđivanje (engl. inheritence) je ključni koncept objektno orijentiranog programiranja. Nasljeđivanje omogućava da na osnovu jedne klase izvedemo jednu ili više pod klasa. Konkretno izvornu klasu tada zovemo superklasa (engl. superclass) te izvedene klase od nje nasljeđuju sve atribute i metode. Izvedene klase mogu imati i svoje karakteristične, dodatne atribute i metode ali će u svakom slučaju naslijediti svojstva superklase.[7] Java nasljeđivanje implementira upotrebom ključne riječi extends kao što je vidljivo u idućem primjeru:

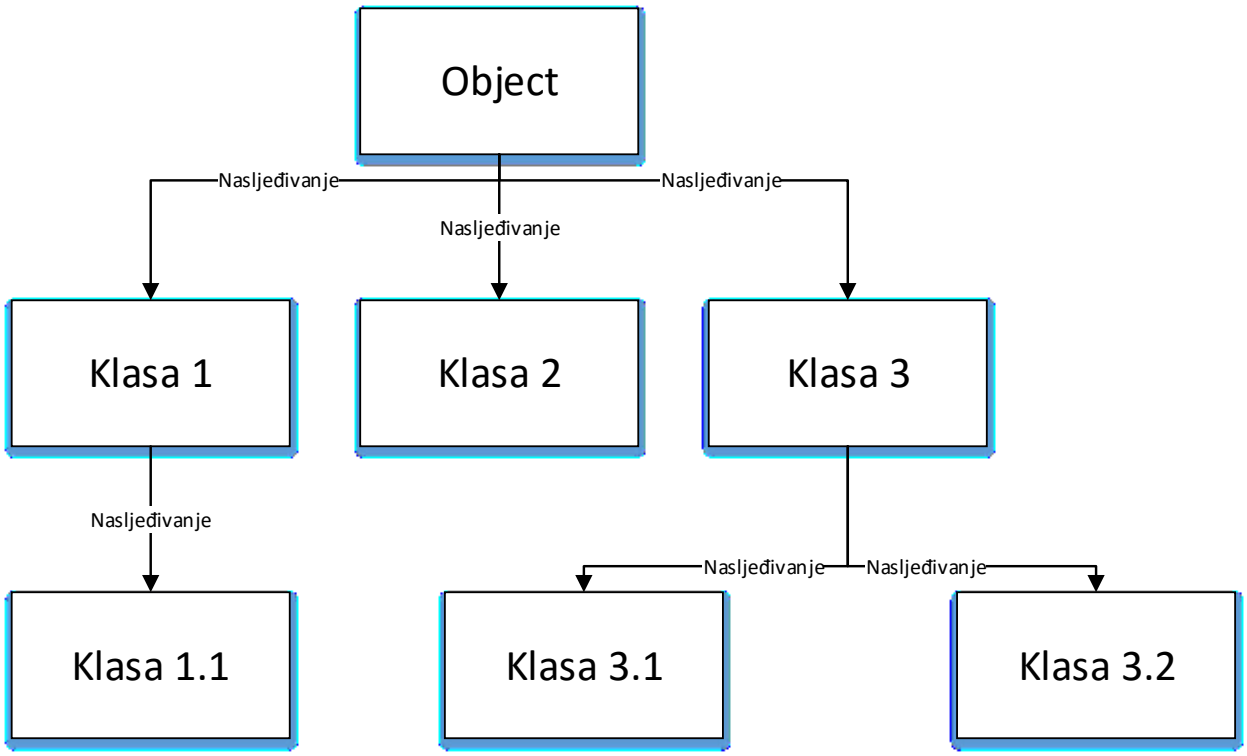
```
class TerenskoVozilo extends Automobil {

int brojRezervnihKotaca;

}
```

Klasa TerenskoVozilo naslijedila bi varijable "boja" i "vrstaMotora" od super klase Automobil i uvela svoju, novu, varijablu "brojRezervnihKotaca".

Nasljeđivanje omogućava stvaranje hijerarhijske strukture objekata i izbjegavanje ponavljanja koda pri deklaraciji sličnih klasa. U Javi sve klase nasljeđuju klasu "Object", sve klase imaju Object kao superklasu te svi objekti, uključujući nizove (engl. arrays) implementiraju metode ove klase[8].

[7] Usp. Oracle Java Documentation Object Oriented Programming Concepts: https://docs.oracle.com/javase/tutorial/java/concepts/

[8] Usp. Oracle Java documentation: https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html



Slika 4 Nasljeđivanje u programskom jeziku Java. Izvor: vlastiti rad autora

3.2.4 Sučelja (interfaces)

Sučelja (engl. Interface) su popis, ili "ugovor" o metodama i atributima koje neki objekt izlaže prema ostalim objektima (ostatku koda, sučelju itd.). Sučelja se implementiraju prilikom instanciranja objekta upotrebom ključne riječi implements. Na primjer metode za upravljanje vozilom iz ranijeg primjera mogle su biti ugrađene u sučelje:

```
public interface upravljacAuta {

	void ubrzaj(int promjenaBrzine);

	void uspori(int promjenaBrzine);

	void pregledBrzine();
}
```

Važno je primijetiti da metode definirane u sučelju ne sadrže blok koda opasan vitičastim zagrada već su samo deklarirane, tj. sastoje se samo od svojeg potpisa (engl. signature).

Da bi deklarirano sučelje iskoristili dodajemo ga klasi upotrebom ključne riječi implements i unutar klase moramo implementirati sve metode koje su opisane u sučelju, inače se kod neće kompilirati (na ovaj način sučelja garantiraju da su sve njihove metode zaista i realizirane te zbog toga sučelja ponekad opisujemo kao "ugovore"). Primjer implementacije sučelja:

```
public class Automobil implements upravljacAuta {

void ubrzaj(int promjenaBrzine) {
      brzina = brzina + promjenaBrzine;
      }

      void uspori(int promjenaBrzine) {
      brzina = brzina - promjenaBrzine;
      }

      void pregledBrzine() {
            System.out.println("Brzina:" + brzina + " m/s.";

}
```

Sučelja se često koriste za skrivanje izvornog koda same metode. Kada neka kompanija želi podijeliti svoj softver sa drugima one često definiraju sučelja, API-e, koji su javo dostupni i iz kojih je moguće iščitati koje su metode dostupne i kako se koriste to jest vidljiva su imena metoda te njihovi ulazi i izlazi (takozvani potpis metoda) ali same metode deklarirane u klasi koja implementira javno dostupno sučelje ali sama nije dostupna u izvornom kodu. Na ovaj način je moguće softver učiniti otvorenim za korištenje i ugradnju u druge sustave bez da se otkriva izvorni kod. Ovo je čest slučaj kod pogonitelja hardverskih uređaja (engl. device drivers) i raznih web API-a.

3.2.5 Nadnaređivanje (overriding)

Pod engleskim pojmom overriding podrazumijevamo sposobnost podklasa da promjene metode svojih superklasa. U primjeru niže vidimo dvije klase, podKlasa nasljeđuje superKlasa-u i mijenja njezinu jedinu metodu:

```
public class superKlasa {

 public void mojaPrvaMetoda() {

   System.out.print("Ovo je rekla superKlasa");
    }
}


public class podKlasa extends superKlasa {

public void mojaPrvaMetoda() {

    System.out.print("Ovo je rekla podKlasa");
    }
}

podKlasa nekiTest = new podKlasa ();
nekiTest.mojaPrvaMetoda();
```

Izlaz ovog koda bio bi:

```
Ovo je rekla podKlasa
```

Da bi metoda iz podklase overrideala metodu iz superklase potrebno je da se metode zovu isto i da imaju isti broj ulaznih i izlaznih varijabli. U slučaju da metode imaju isti naziv ali ne i iste ulazne varijable govorimo o preopterećivanju metoda (engl. overloading).

3.2.6 Enkapsulacija

Enkapsulacija je jedan od četiri osnova koncepta objektno orijentiranog programiranja (uz polimorfizam, apstrakciju i nasljeđivanje). Pod pojmom enkapsulacije podrazumijevamo osobinu objekta da sakriju neke atribute (varijable) i metode te omogućimo pristup istima upotrebom "getter" i "setter" metoda[9]. Varijable se enkapsuliraju (skrivaju) upotrebom ključne riječi private. Sljedeći primjer koda prikazuje enkapsulaciju varijable i deklariranje "getter" i "setter" metoda:

```
public class TestEnkapsulacije{

      private String mojaEnkapsuliranaVarijabla;

      public String getMojaVarijabla(){
            return mojaEnkapsuliranaVarijabla;
      }
      public void setMojaVarijabla (String MojaVarijabla){
            mojaEnkapsuliranaVarijabla = MojaVarijabla;
      }
}
```

U gornjem primjeru varijabla mojaEnkapsuliranaVarijabla nije vidljiva izvan instanciranih objekata klase TestEnkapsulacije ali ju je moguće pročitati metodom getMojaVarijabla ("getter" metoda) i zapisati metodom setMojaVarijabla ("setter" metoda).

```
public class TestGetterSetter{

      public static void main(String args[]){

            TestEnkapsulacije enkap = new TestEnkapsulacije ();
            enkap. setMojaVarijabla ("Velika tajna");


            System.out.print("Moja enkapsulirana, settana i gettana varijabla
            je: " + enkap.getName);
      }
}
```

Izlaz ovog koda bi, očekivano, bio:

```
Moja enkapsulirana, settana i gettana varijabla je: Velika tajna
```

[9] Raymod Lewallen: 4 major principles of Object-Oriented Programming.
http://codebetter.com/raymondlewallen/2005/07/19/4-major-principles-of-object-oriented-programming/

Upotrebom enkapsulacije možemo osigurati da je neke varijable unutar klase moguće samo čitati (engl. read only) ili samo zapisivati (engl. write only), da klasa ima potpunu kontrolu nad sadržajem svojih varijabli (polja) i da je moguće promijeniti ime ili čak tip podataka u klasi bez da je potrebno mijenjati vanjski kod koji koristi klasu.

3.2.7 Polimorfizam

Općenito govoreći možemo reći da je polimorfizam sposobnost objekta da poprimi više oblika. U objektno orijentiranom programiranju pod polimorfizmom podrazumijevamo sposobnost objekata da naslijedi i promjeni svojstva svoje superklase[10]. Ovo se najčešće očituje kroz overriding metode i sučelja (engl. interfaces) klasa.

Vratimo se ranijem primjeru superklase Automobil i klase koja ju nasljeđuje TerenskoVozilo. Metode superklase mogu u podklasi biti overrideane ili mogu biti deklarirane putem sučelja koja implementiraju razne klase ali pri opisu metoda uvode manje ili veće razlike u njihov rad.

3.2.8 Apstrakcija

Apstrakcija je postupak skrivanja detalja implementacije to jest izvođenja nekog proces od korisnika (vanjskog koda) i prikazivanje, to jest dijeljenje, izlaganje samo funkcionalnosti. Java postiže apstrakciju upotrebom apstraktnih klasa i sučelja (engl. interfaces, opisani u kasnijem poglavlju). Apstraktne klase nije moguće implementirati (stvoriti objekt na osnovu njih) ali ih je moguće naslijediti postupkom opisanim u ranijem poglavlju[11]. Ako smo ranije klase opisali kao nacrte objekta nije pogrešno razmišljati o apstraktnim klasa kao o nacrtima za klase.

Klasa koja sadrži ključnu riječ abstract u svojoj deklaraciji je apstraktna klasa. Po svemu ostalome apstraktne klase se ne razlikuju od običnih klasa.

Osim klase i metoda može biti apstraktna. Apstraktne metode mogu biti deklarirane samo u apstraktnoj klasi. Apstraktne metode se samo deklariraju (ne sadrže tijelo metode) na primjer:

```
public abstract class MojaApstraktnaKlasa
{
   private String nekiString;
   private String nekiDrugiString;

   public abstract int mojaApstrakntaMetoda();
}
```

Deklariranje metode apstraktnom ima nekoliko posljedica. Klasa u kojoj je deklarirana apstraktna metoda mora i sama biti apstraktna, klase koje nasljeđuju apstraktnu klasu koja deklarira apstraktnu metodu moraju i same biti apstraktne ili implementirati apstraktnu metodu.

---

[10] Usp. Oracle Java Documentation Polymorphism:
https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html
[11] Usp Oracle Java Documentation Object Oriented Programming Concepts:
https://docs.oracle.com/javase/tutorial/java/concepts/

3.3 Specifičnosti programiranja za Android

Iako je Java jezik Androida i sve gore opisane tehnike programiranja Jave su primjenjive (i nužno ih je primijeniti) i na razvoj aplikacija za Android, postoje mnoge specifičnosti od kojih ćemo u narednim poglavljima proći najvažnije. Ove specifičnosti često su rezultat hardverskih ograničenja (relativno malo memorije i procesorske moći) mobilnih uređaja i uvjeta u kojima se koriste (u vanjskom prostoru, u pokretu).

3.3.1 Aktivnosti

Aktivnosti (engl. activity) su osnova korisničkog sučelja na Android uređajima. Komunikacija sa korisnicima moguća je drugim načinima (npr. "toast" porukama) ali Activity je ono što korisnik vidi u većini slučajeva. Mogli bi reći da je u svijetu desktop aplikacija ekvivalent aktivnosti Windows forma ili u svijetu web aplikacija HTML stranica. Aktivnost se sastoji od XML datoteke koja opisuje korisničko sučelje i .java datoteka koje sadrže kod povezan sa konkretnom aktivnošću. Ove .java datoteke sadrže klase koje nasljeđuju Activity superklasu.

Kao što je rečeno, korisničko sučelje aktivnosti je opisano XML datotekama koje će biti detaljno prezentirane u kasnijem poglavlju (kao i Java kod vezan uz njih). Za sada valja napomenuti da je razvoj aktivnosti tj. korisničkog sučelja izuzetno važan korak u razvoju mobilnih aplikacija zbog specifičnosti platforme i načina na koji se koristi. U odnosu na desktop i web aplikacije, mobilne aplikacije se moraju izvoditi na malim ekranima, čija orijentacija se često mijenja i koje se često letimično promatraju sa udaljenosti (a ne iz sjedećeg položaja koji je predviđen upravo za promatranje zaslona). Mobilni uređaji (pa onda i aplikacije) se često koriste u hodu ili dok se korisnik bavi nekom drugom aktivnošću.

Nadalje, mobilne aplikacije izazivaju značajno veću količinu ljutnje kod korisnika kada uopće ne funkcioniraju ili ne funkcioniraju na način na koji korisnik očekuje. Zadaci koji se odrađuju na mobilnim uređajima kao što su telefoniranje, čitanje i slanje poruka, snalaženje na karti, čak i zadatak aplikacije opisane u ovo radu (automatska naplata parkinga) su relativno jednostavni zadaci i korisnici očekuju da ih je moguće odraditi brzo, pouzdano i na očigledan način te će bilo kakve komplikacije u sučelju brzo izazvati bijes korisnika.

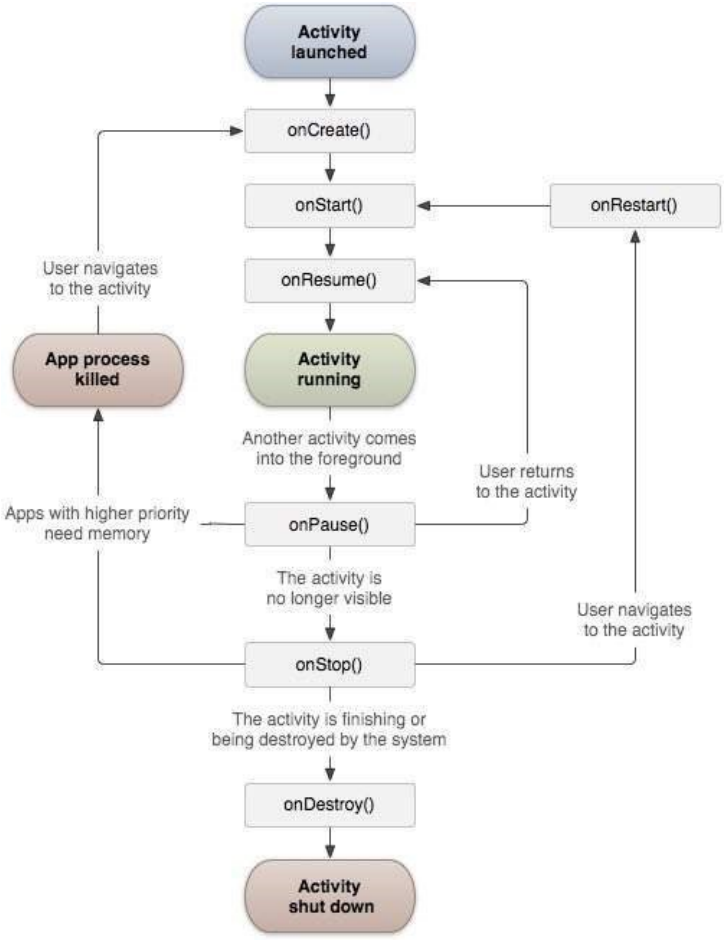
Iz gore navedenih razlog razvoj Android aktivnosti, to jest korisničkog sučelja je značajno kompliciraniji zadatak nego razvoj ekvivalenata u web i desktop aplikacijama. Suvremeni trendovi u dizajniranju korisničkog sučelja su razvili novi pravac koji je specijaliziran za mobilne uređaje ali se polako probija i u desktop (na primjer Microsoft Windows 8 Metro korisničko sučelje) i web aplikacije (na primjer Flat UIree User Interface Kit[12]). Flat dizajn je karakterističan po minimalnosti i jednostavnosti (kako vizualnog dizajna tako i načina interakcije).

Uz ranije opisani flat dizajn svakako je potrebno objasniti i drugi koncept vezan uz korisničko sučelje koji je proizašao iz mobilnih aplikacija – responsivni dizajn. Pod responsivnim dizajnom podrazumijevamo sposobnost korisničkog sučelja da se prilagođava različitim uvjetima prikaza na zaslonu. Iako najčešće govorimo o responsivnom web dizajnu (engl. Responsive web design) njegov razvoj uvelike je povezan sa razvojem mobilnih aplikacija. Uz značajne izmjene u karakteristikama zaslona za prikaz aplikacije

[12] Flat User Interface Kit http://designmodo.github.io/Flat-UI/

(poput dimenzija zaslona, gustoće točaka zaslona, dubine boja itd.) suvremeni responsivni dizajn je u stanju reagirati na promjene u zaslonu čak i realnom vremenu (pri promjeni orijentacije ekrana, iz vodoravne u okomitu i obrnuto).

Uz ove razloge daljnja komplikacija pri izradi aktivnosti je rezultat činjenice da se izvođenje aplikacije (to jest aktivnosti) može privremeno ili trajno zaustaviti u bilo kojem trenutku. Ovo se događa jer je druga aplikacija, koja ima veći prioritet (npr. dolazni telefonski poziv), odjednom zatražila resurse (koji mogu biti ograničeni i na rubu pune iskorištenosti) što rezultira tjeranjem vaše aplikacije u pozadinu (ili čak prisilnim gašenjem aplikacije). Dijagram niže opisuje životni ciklus aktivnosti na Android mobilnom uređaju i navodi (naslijeđene) metode koje su mogu koristiti pri različitim koracima životnog ciklusa da bi se osiguralo normalno izvođenje aplikacije i spriječio gubitak podataka.



Slika 5 Životni ciklus Android aktivnosti. Izvor: Tutorials point - http://www.tutorialspoint.com/android/android_acitivities.htm

3.3.2 Pružatelji sadržaja

Pružatelji sadržaja (engl. content providers) su koncept jedinstven Androidu (barem u ovakvoj implementaciji) koji omogućava dijeljenje podataka između više aplikacija.[13] Sam Android pruža više često korištenih pružatelja sadržaja (kao što je kalendar i popis kontakata pa njihovim podacima). Aplikacije pristupaju pružateljima sadržaja upotrebom ContentReslover klase a stvaranje novih pružatelja sadržaja ostvaruje se upotrebom ContentProvider klase. Autori Android aplikacija se ohrabruju da što više izrađuju vlastite pružatelje sadržaja (naravno, ako njihov sadržaj ima smisla dijeliti) jer se na taj način izbjegava dupliciranje izvođenja nekih radnji, poput obrade podataka ili zahtjeva za podacima koji mogu biti vrlo zahtjevni po ograničene resurse mobilnih uređaja. PassiveNetworkLocationProvider (koji je opisan u kasnijem poglavlju) odličan je primjer dijeljenja podataka sa ciljem očuvanja baterije mobilnog uređaja.

3.3.3 Intenti

Intenti ili namjere (engl. intent) se najčešće opisuju kao "apstraktni opis operacije koja će se izvršiti". U slučaju nekih općenitih, čestih radnji ovi opisi omogućavaju uređaju da ponudi korisniku izbor sustava kojim će planirana akcija biti izvršena[14]. Na primjer: u slučaju najave operacije slanja e-mail poruke Android može korisniku omogućiti izbor e-mail klijenta kojim će slanje biti izvršeno. Donji kod je primjer upotrebe intenta za slanje email poruke[15].

```
Intent email = new Intent(Intent.ACTION_SEND, Uri.parse("mailto:"));
email.putExtra(Intent.EXTRA_EMAIL, recipients);
email.putExtra(Intent.EXTRA_SUBJECT, subject.getText().toString());
email.putExtra(Intent.EXTRA_TEXT, body.getText().toString());
startActivity(Intent.createChooser(email,    "Odaberite    email    klijentski
softver"));
```

3.3.4 Servisi

Servisi su komponente aplikacije ili operativnog sustava koji se kontinuirano izvode u pozadini i obavljaju neki dugotrajni zadatak, česta zadaća servis je čekanje na neki događaj na primjer dolazni telefonski poziv na mobilni uređaj. Zbog ograničenih resursa na mobilnim uređajima servisi su u ovom okruženju puno rjeđi nego u desktop aplikacijama(gdje se redovito, posebno na Windows platformi, izvode stotine ili tisuće servisa). Android servisi se dijele na „pokrenute“ (engl. Started) i „povezane“ (engl. Bound). Razlike između ove dvije vrste jest u tome da se „pokrenuti“ servisi izvode tek kada to neka aplikacija (to jest komponenta aplikacije, aktivnost) zatraži a „“povezani“ servisi pružaju uobičajeno klijent-poslužitelj sučelje za više komponenti ili aplikacija. Kao i aktivnosti Android servisi imaju striktno definiran životni ciklus te koriste onCreate, onStart i slične metode da bi osigurale svoje normalno funkcioniranje.

---

[13] Usp. Android developer documentation Content Providers: https://developer.android.com/guide/topics/providers/content-providers.html
[14] Usp. Android developer documentation Intent: https://developer.android.com/reference/android/content/Intent.html
[15] Usp. Android Intents and filters tutorial: http://www.tutorialspoint.com/android/android_intents_filters.htm

3.4 Integrirana razvojna okruženja za Android

Integrirana razvojna okruženja, skraćeno IDE (engl. integrated development enviroment), su softverske aplikacije koje u sebi sadrže sve osnovne alate koje developeri trebaju za pisanje i testiranje softvera. IDE je dizajniran da olakša posao programerima i developerima, i za to koristi kodiranje bojama, dijagnostiku i prijavu grešaka u kodu, formatiranje izvornog koda i čak inteligentno kompletiranje koda.

Uz svoje osnovne alate većina IDE-a nudi svoj dućan na kojem korisnici mogu pronaći dodatne alate i pluginove za svoje projekte. Još jedna od značajki IDEa je da svaki od njih ima i svoju zajednicu korisnika, koji sudjeluju u razvoju novih pluginova ili pomažu u rješavanju problema drugih programera.

U posljednje se vrijeme IDE sve više pojavljuju u obliku SaS (Software as Service) web servisa. Jedna od prednosti korištenja IDEa u oblaku su između ostalog pristup razvojnim alatima od bilo gdje u svijetu i s bilo kojeg kompatibilnog uređaja, minimalna ili potpuno nepotrebna instalacija, i olakšavanje suradnje developerima koji se nalaze na različitim geografskim lokacijama.

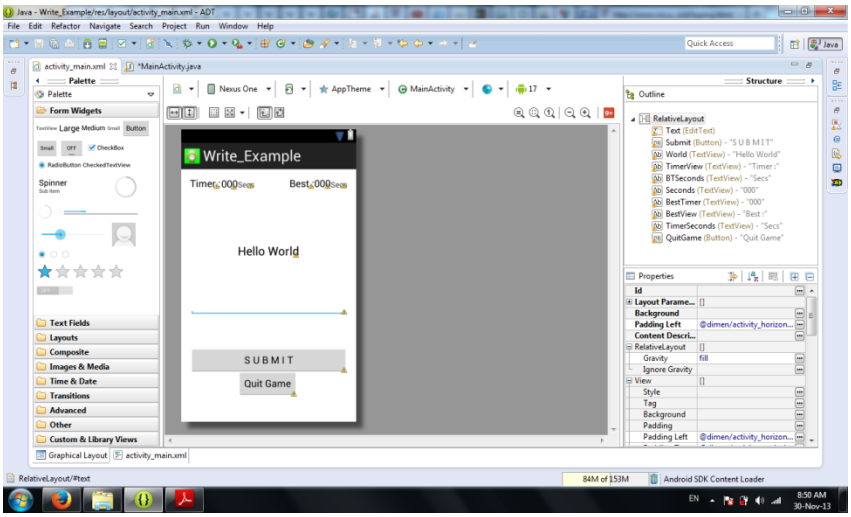
3.4.1 Eclipse ADT

Iako je najpoznatiji kao IDE za razvoj Java aplikacija, Eclipse također nudi i dodatak (engl. plugin) za razvoj Android aplikacija - Eclipse ADT (engl. Android development toolkit).

ADT je plugin za sam Eclipse, i kao što mu ime govori, svrha mu je ponuditi okruženje za razvoj Android aplikacija. Ovdje se radi, kao i u slučaju drugih IDEova, o paketu alata koji omogućuju korisniku da brzo i lako pokrene svoj novi Android projekt, napravi korisničko sučelje i dodaje razne pakete iz Android Framework APIja, kao i uobičajeno otklanjanje grešaka s Androidovim SDK alatima. Naravno, Eclipse ADT omogućuje i eksportiranje .apk datoteka za daljnju distribuciju[16].

Prvi ADT se za Eclipse pojavio u 2012. godini, i trenutno se navodi kao stabilan i u produkciji, no ono što je važno napomenuti je da Android Studio više službeno ne podržava ADT plugin iako je on još uvijek u upotrebi (posebno u obrazovnim institucijama koje još nisu prešle na suvremeni Android IDE - Android Studio).

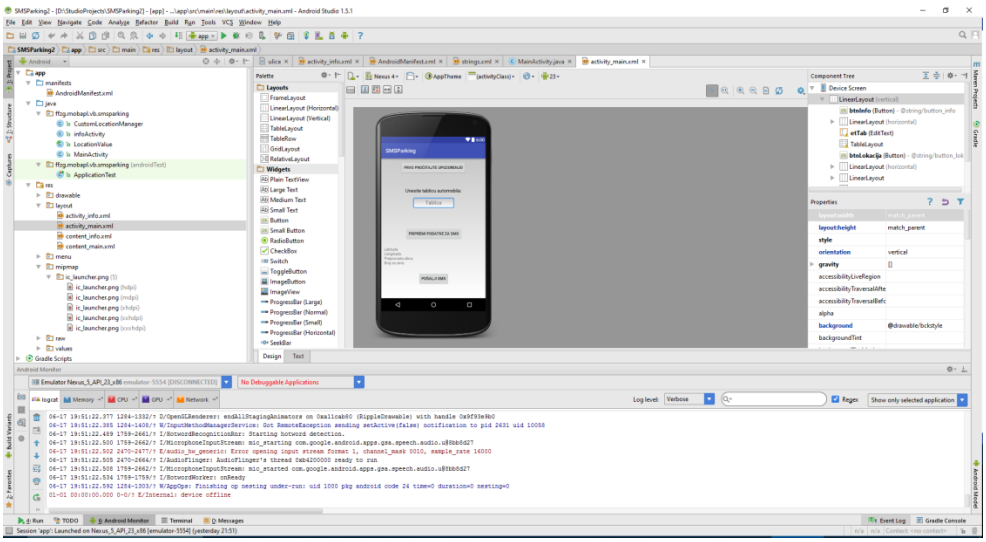
[16] Usp. Eclipse ADT: https://marketplace.eclipse.org/content/android-development-tools-eclipse



Slika 6 Eclipse ADT korisničko sučelje. Izvor: StackOverflow - Android SDK (Eclipse) : Typing Game - http://stackoverflow.com/questions/20297031/android-sdk-eclipse-typing-game-how-to-create-an-array-of-string-resources

3.4.2 Android studio

Sam Android Studio također je IDE, no ovaj put se radi o Googleovom razvojnom okruženju pisanom u Javi i namijenjenom programerima na svim platformama. Zasnovan je na IntelliJ IDEa, no za razliku od njega nudi fleksibilan sustav zasnovan na Gradleu, unificirano okruženje za sve Android uređaje, predloške koda i GitHub integraciju koja pomaže kod izgradnje najčešćih aplikacijskih odlika i uvoz primjera koda, kao i ugrađenu podršku za Google Cloud platformu, čime olakšava integraciju Google Cloud Messengera i aplikacijskog enginea.



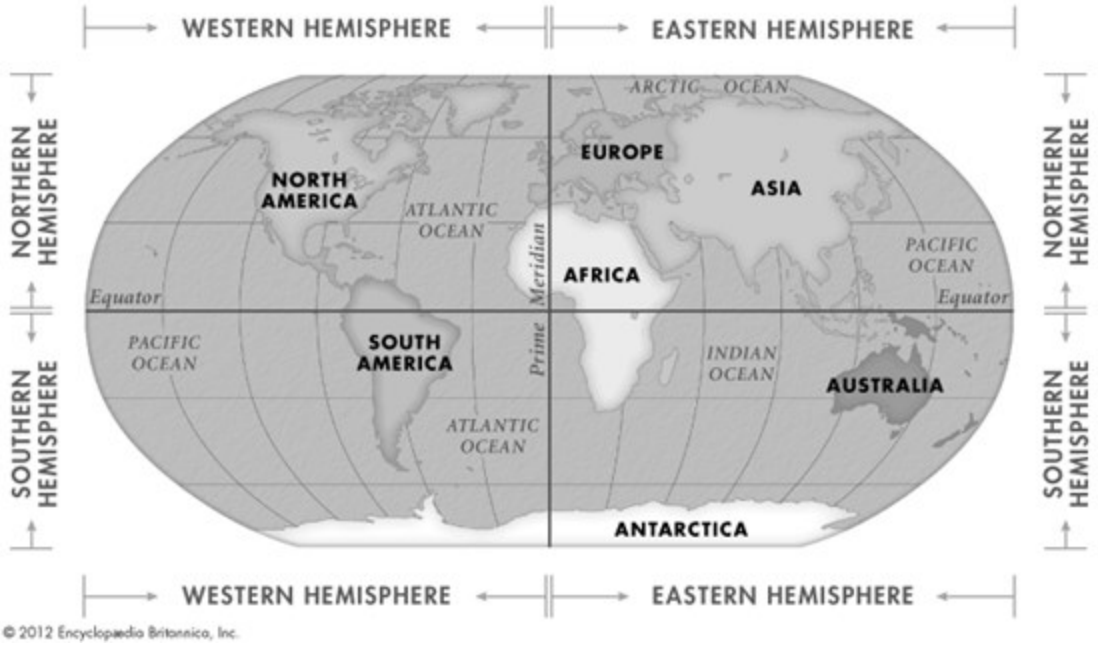
Slika 7 sučelje Android Studia. Izvor:vlastiti rad autora

Vjerojatno najvažnija značajka i Eclipse i Android Studio integriranog razvojnog okruženja je činjenica da pružaju mogućnost testiranja koda na virtualnom uređaju, to jest na više njih. Zbog velikog broja

različitih proizvođača uređaja ovo je izuzetno korisna opcija. Pisanje android koda moguće je bez nekog IDEa ali testiranje koda je kamen spoticanja koji je teško zaobići bez ovih aplikacija. Čak i pri razvoju aplikacije poput SmsParkinga, koja se oslanja na geolokacijske servise, koje je nemoguće potpuno testirati bez pravog uređaja (i kretanja to jest promjena geografskog položaja) Eclipse a kasnije Android Studio je bio od iznimne važnosti za testiranje korisničkog sučelja, programske logike i ostalih komponenti. Iz osobnog iskustva autora može se zaključiti da bi razvoj mobilne aplikacije trajao višestruko duže bez ovih alata.

4. Lokacijski servisi na Android uređajima

Cilj lokacijskih servisa je utvrditi trenutni položaj uređaja, tj. geografske koordinate. Geografske koordinate (zemljopisna duljina i širina) izražene u stupnjevima od ekvatora, sa oznakom N (engl. North) za sjevernu hemisferu i oznakom S (engl. South) za južnu hemisferu (ili negativnim predznakom) i stupnjevima od prvog meridijana (koji prolazi kroz Greenwich) , sa oznakom W (engl. West) za zapadnu hemisferu (ili negativnim predznakom) i oznakom E (engl. East) za istočnu hemisferu.



Slika 8 Hemisfere zemlje. Izvor: Encyclopedia Brittanica.

Postoji više uobičajenih načina zapisa geografskih koordinata. Tradicionalni zapis, „Stupnjevi-minute-sekunde“ prikazuje koordinate u stupnjevima koje dijeli na 60 minuta koje su dalje podijeljene na 60 sekundi. Na primjer geografske koordinate zgrade Filozofskog fakulteta Sveučilišta u zagrebu su:

45°47'47.8"N 15°58'18.1"E

Ili, prirodnim jezikom: 45 stupnjeva, 47 minuta i 47.8 sekundi sjeverno i 15 stupnjeva i 58 minuta i 18.1 sekundi istočno. Kao što je vidljivo ova notacija nije praktična za zapis niti jednostavna za upotrebu u računalnim sustavima[17]. Računalni sustavi (uključujući Android) zapisuju geografske koordinate u decimalnim stupnjevima sa predznakom koji označava hemisferu. Takvom notacijom koordinate su svedene na par decimalnih brojeva gdje longituda može biti u rasponu od -180 do 180 a latituda u rasponu od -90 do 90. Konkretno, gore navedene koordinate, se zapisuju kao:

45.79660, 15.97170

[17] Tomasz Imielinski, Julio C. Navas: GPS-Based Geographic Addressing, Routing, and Resource Discovery

Dok bi koordinate neke lokacije u zapadnoj-i južnoj hemisferi imale negativne predznake. Na primjer, koordinate grada Rio de Janeiro su:

22°54'24.5"S, 43°10'22.4"W

I u decimalnom obliku:

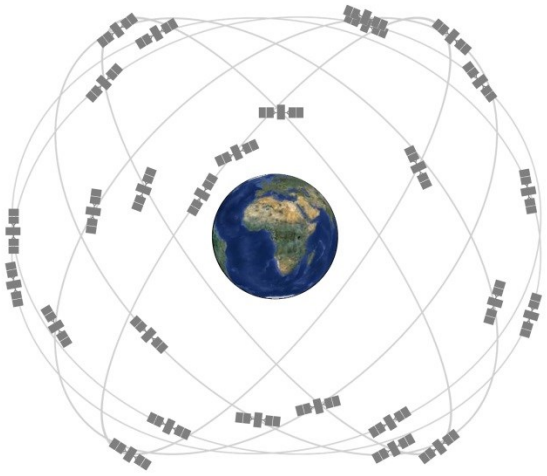
-22.90680, -43.17290

Uz ovdje prikazane sustave notacije uobičajen je i sustav koji prikazuje stupnjeve cijelim brojevima i minute decimalnim brojevima (bez sekundi).

Android omogućava dobivanje informacija o trenutnom položaju (geografske koordinate) uređaja kroz dvije metode (to jest tri ali zadnja, pasivna, ovisi o prve dvije): GPS i Network location provider. Obje metode načina utvrđivanja lokacije uređaja su dostupna kroz Location manager klasu i Google Maps API te će kasnija poglavlja prikazati konkretnu implementaciju ovih tehnika a ovo poglavlje će prikazati na koje vanjske sustave se ove metode oslanjaju te kratko raspraviti prednosti i nedostatke obje.[18]

4.1 Lokacijski servis - Global positioning system

GPS - Globalni sustav pozicioniranja (engl. Global positioning system) je sustav od 28 satelita (24 u aktivnoj uporabi i četiri koja se nalaze u pričuvi te zamjenjuju satelite koji su u kvaru).Projekt je započet 1973. od strane Sjedinjenih američkih država i postao je funkcionalan 1995., kada je lansirano prvih 24 satelita. Sateliti pružaju informacije o položaju i vremenu bilo kojem prijamniku na zemlji pod uvjetom da su barem 3 satelita vidljiva iz položaja prijamnika (engl. reciever) u trenutku zahtjeva. Iako je započeo kao vojni sustav GPS je danas slobodno dostupan bilo kojem prijamniku uz stanovita ograničenja preciznosti.[19]

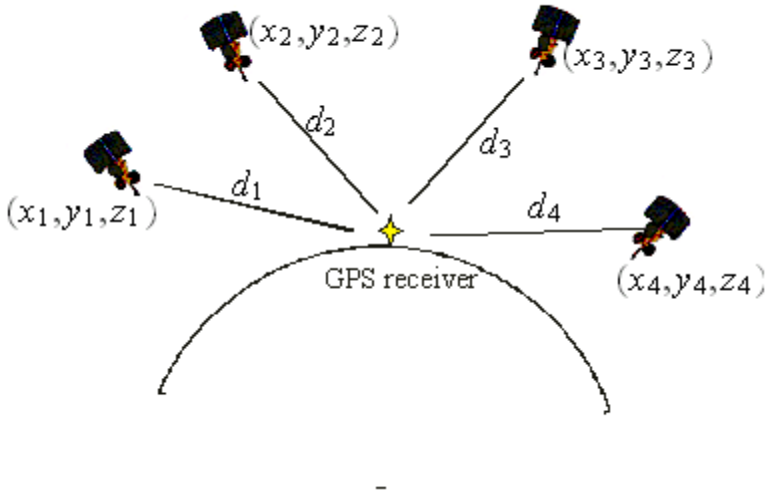


Slika 9 GPS konstelacija satelita. Izvor: Official U.S. Government information about the Global Positioning System (GPS) and related topics.

[18] Anroid location strategies. URL: https://developer.android.com/guide/topics/location/strategies.html
[19] James Bao-Yen Tsui CoFundamentals of Global Positioning System Receivers: A Software Approach

Većina Android uređaja su i GPS prijemnici (to jest sadrže GPS prijemnik), štoviše jedan od Googleovih zahtjeva u specifikaciji Android uređaja je prisutnost GPS prijemnika. U rijetkim slučajevima kada prijamnik nije tvornički ugrađen u uređaj moguće je povezati vanjski GPS prijemnik putem Bluetooh ili USB sučelja (npr. Motorola T805 Bluetooth GPS Receiver[20]).

GPS prijamnici funkcioniraju tako da „osluškuju“ GPS satelite, to jest, primaju pakete sa podacima od njih. Ove pakete sateliti emitiraju mikrovalovima svakih 30 sekundi. Svaki paket sadrži podatke o orbiti satelita i vrijeme kada je odaslan. Svakih 12 minuta šalje se veći paket koji sadrži i atmosferske (vremenske podatke) koji su potrebni da bi se moglo korigirati podatke u skladu sa trenutnim vremenskim prilikama (koje utječu na brzinu mikrovala). Da bi prijamnik primio paket od nekog satelita on ga mora „vidjeti“, to jest, mora biti moguće povući neprekinutu ravnu liniju između prijamnika i satelita. Ako je prijamnik primio pakete od barem četiri satelita usporedbom vremena koje je zapisano u primljenim paketima sa vlastitim satom moguće je izračunati udaljenost svakog satelita od prijamnika (brzina svjetlosti je konstanta i korigirana prema trenutnim atmosferskim podacima)[21]. GPS prijamnik je u paketima uz vrijeme odašiljanja paketa primio i podatke o položaju svakog satelita te je na osnovi izračunatih udaljenosti od satelita u stanju procijeniti vlastiti položaj.



Slika 10 Izračun položaja upotrebom GPS sustava. Izvor: G. Donald Allen, Three Physics Examples - "getting the physics right" plus a little more

GPS se smatra najpreciznijim načinom utvrđivanja geografskih koordinati uređaja ali je najzahtjevniji za bateriju, ne funkcionira ako nisu vidljivi sateliti (unutar neke zgrade na primjer) i ima relativno dugo vrijeme odaziva. Preciznost GPS sustava se navodi kao „manja od 100m“ premda i brzi pogled na bilo koju od mnogih aplikacija koje koriste Google Maps API ostavlja dojam da je ova preciznost puno veća (20 ili manje metara).

[20] Motorola T805 Bluetooth GPS Receiver. URL: http://technogog.com/review/review-of-motorola-t805-bluetooth-gps-receiver/
[21] Greg Pendleton: The Fundamentals of GPS

Vrijedno je spomenuti da iako je GPS sustav trenutno otvoren za upotrebu bilo kome, on je sustav SAD-a te nije nezamislivo da u nekom trenutku zbog sigurnosnih ili drugih razloga SAD odluči ukinuti slobodni pristup. Iz ovog razloga kratko ćemo opisati alternativne navigacijske sustave.

GLObal Navigation Satellite System (GLONASS) je alternativna konstelacija navigacijskih satelita koju je lansirala i održava Ruska federacija. Ovaj sustav aktivan je od 2010. i značajno je manje u civilnoj upotrebi od GPS sustava. GLONASS prijemnici funkcioniraju na istom principu ali se ipak razlikuju od GPS prijemnika i dostupni su na rijetkim Android uređajima, na primjer na nekim Sony Xperia uređajima[22]. Uređaji koji podržavaju i GPS i GLONASS sustav su u značajnoj prednosti nad uređajima koji koriste samo GPS sustav jer imaju gotovo duplo veću količinu satelita koje mogu osluškivati te su puno pouzdaniji u područjima u kojima je teško vidjeti dovoljnu količinu satelita, na primjer u gustim gradskim središtima gdje je prijamnik okružen visokim zgradama.

Trenutno je u izgradnji konstelacija navigacijskih satelita Europske unije, Global navigation satellite system (GNSS) koje će kada bude dovršena pružiti još jednu alternativu upotrebi GPS sustava. Očekuje se da će ovaj sustav biti u upotrebi 2020.

4.2 Lokacijski servis - Network Location provider

Network location provider lokacijski servis se oslanja na dvije metode: prepoznavanje komunikacijskog tornja mobilnog operatera na koje je mobilni uređaj spojen i prepoznavanja Wi-Fi mreže na koju je mobilni uređaj spojen. Ovi podaci uspoređuju se bazama podatka o geografskom položaju konkretnih tornjeva i Wi-Fi mreža te se na osnovu njih može prepoznati manje ili više precizno geografski položaj uređaja. Obje metode ne troše bateriju ništa više od „uobičajene“ upotrebe uređaja te se smatraju jeftinim metodama iz perspektive potrošnje baterije(ovdje pod uobičajeno smatramo da je uređaj spojen na mobilnu GSM, GPRS, EDGE, UTMS ili 4G mrežu i da aktivno traži otvorene Wi-Fi mreže). U usporedbi sa GPS lokacijskim servisom ove metode su dosta neprecizne. Za kombiniranu upotrebu ovih sustava obično se navodi preciznost od 100-500 metra[23], ipak treba napomenuti da preciznost ovog servisa uvelike ovisi o broju tornjeva i Wi-Fi mreža koje se nalaze u blizini te je moguće da sustav postigne i bolje rezultate od GPS-a u razvijenim urbanim sredinama.

4.2.1 Lokacija upotrebom tornjeva mobilnih operatera

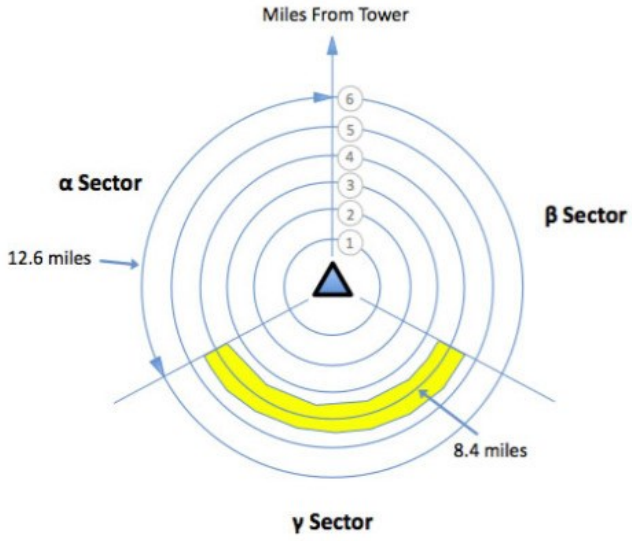
Ova tehnika oslanja na se na činjenicu da su mobilni uređaji (prije svega mobilni telefoni) cijelo vrijeme povezani sa mrežom komunikacijskih tornjeva mobilnih operatera čiji je položaj poznat.

Svaki komunikacijski toranj pri komunikaciji sa mobilnim uređajima šalje svoj identifikacijski broj (engl. Cell ID) i geografsku zonu (engl. Location area code – LAC). Uz ove podatke i toranj i mobilni uređaj u stanju su izračunati udaljenost mobilnog uređaja od tornja na osnovi jačine signala i vremena koje je potrebno za „puni krug“ komunikacije (toranj šalje, uređaj prima, uređaj šalje, toranj prima). Mobilni uređaj može ove podatke proslijediti lokacijskom servisu koji, ako može geografsku zonu i identifikacijski broj tornja povezati sa geografskim koordinatama, može pretpostaviti položaj mobilnog uređaja.

---

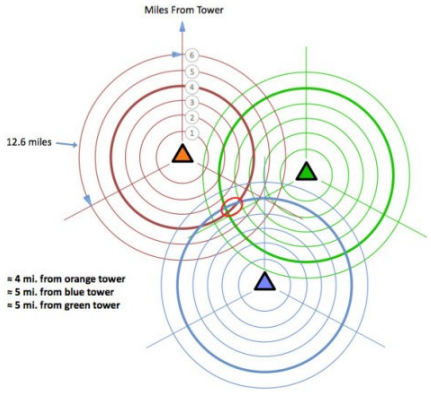
[22] GLONASS Xperia. URL: http://developer.sonymobile.com/knowledge-base/technologies/glonass/
[23] Marc Fasel: A Good Look at Android Location Data

Suvremeni komunikacijski toranj mobilnog operatera sastoji se od tri antene (takozvane „rašlje“). Svaka antena pokriva sektor od 120 stupnjeva u svom dometu od ishodišta to jest do samog tornja. U slučaju da pokušavamo otkriti lokaciju prema udaljenosti od samo jednog tornja dobiti ćemo dosta veliko područje kako je prikazano na slici niže. [24]



Slika 11 Triangulacija geografskog položaja upotrebom jednog komunikacijskog tornja mobilnih operatera. Izvor: Phil Locke Cell Tower Triangulation – How it Works

U slučaju da je dostupno više od jednog tornja moguće je izračunati precizniji položaj kako je prikazano na narednoj slici.



Slika 12 Triangulacija geografskog položaja upotrebom tri komunikacijska tornja mobilnih operatera. Izvor: Phil Locke Cell Tower Triangulation – How it Works

4.2.2 Lokacija upotrebom prepoznavanja vidljivih Wi-Fi mreža

Prepoznavanje položaja upotrebom komunikacijskih tornjeva operatera često se kombinira sa podacima ovog sustava iako on može i nezavisno funkcionirati. Svi Android mobilni uređaji Google-u periodički

[24] Phil Locke: Cell Tower Triangulation – How it Works

šalju anonimne podatke. Ovi podaci, između ostaloga, sadrže i zadnji poznati geografski položaj (otkriven upotrebom GPS i Network lokacijskih servisa) i ime Wi-Fi mreže na koju je uređaj trenutno spojen. Ovo slanje anonimnih podataka moguće je ugasiti. Ovi podaci se koriste da bi nadogradili i ažurirali bazu podataka o geografskim lokacijama Wi-Fi mreža koju je Google izradio kada su fotografirali ulice svjetskih gradova za Google street view servis[25]. Lokacijski servis Network provider koristi podatke o „vidljivim" Wi-Fi mrežama (ako su dostupni) da bi povećao preciznost Network location provider servisa.[26]

4.3 Lokacijski servis - Passive network provider

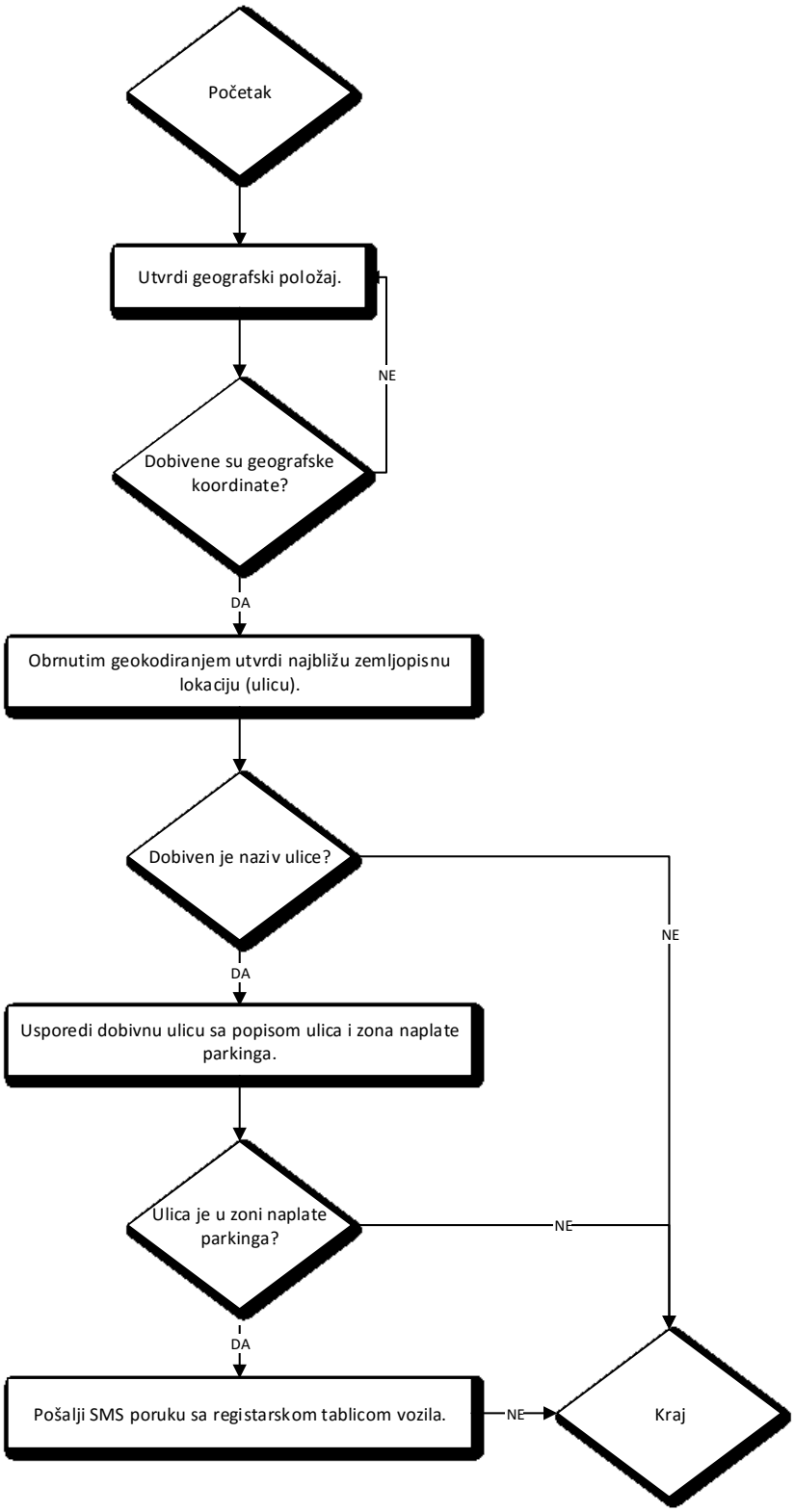
Ovo metoda prikupljanja geografskih podataka podrazumijeva čekanje da neka druga aplikacija zatraži podatke ili od GPS sustava ili od Network location providera i onda koristi podatke koji su namijenjeni drugoj aplikaciji. Ova metoda nije prikladna za bilo koju aplikaciju koja mora na korisnički zahtjev znati lokaciju uređaja već samo za aplikacije koje teko povremeno moraju znati geografski položaj ili im primarna funkcija nije ovisna o geografskom položaju. Velika prednost ove metode je da ona ne izaziva nikakvu dodatnu potrošnju baterije (za razliku od ranije opisanih metoda).

5. Izrada aplikacije i korištene tehnologije

Aplikacija je u potpunosti ostvarena upotrebom besplatnih servisa koje pruža Google kao dio Android API-a. Razvoj aplikacije započeo je u Eclipse-u te je kasnije prenesen u Android studio. Kod je kompiliran za Android API razinu 16 (kodno ime Jelly Bean). Prema podacima iz Google Play Store app-a[27] na dan 9.6.2016. Jelly Bean (API razine 16,17 i 18) ima među Android uređajima udio od 18,9%. S obzirom da su API razine unazad kompatibilne ali ne i unaprijed, izrada i kompiliranje aplikacije za API razine 16 morala bi aplikaciju učiniti dostupnom na 96% Android uređaja. Preostalih 4% odnosi se na uređaje starije od 5 godina.

Funkcioniranje aplikacije možemo podijeliti u nekoliko općenitih koraka koji su prikazani na dijagramu toka niže.

[25] Usp. Google street view servis: https://www.google.com/maps/streetview/
[26] Usp. Wade Burch: Understanding Android: Three Ways to Find Your Location
[27] Web stranica Google Play Store Dashboard: https://developer.android.com/about/dashboards/index.html

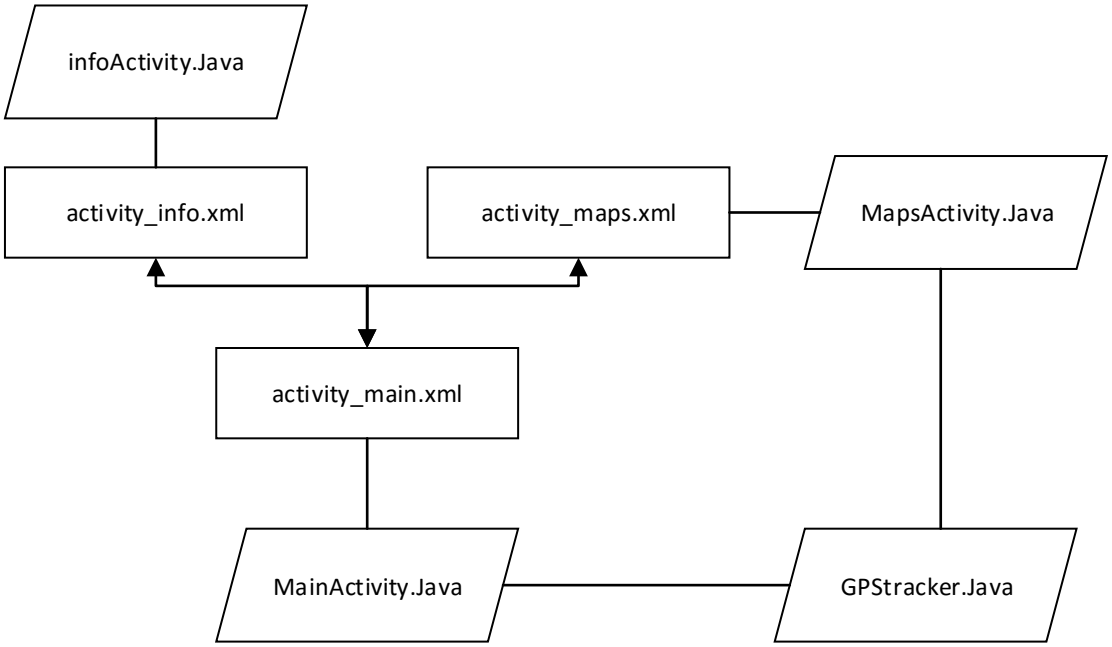


Slika 13 Dijagram toka izvođenja aplikacije SmsParking. Izvor: vlastiti rad autora

Prvi korak je utvrditi geografski položaj korisnika. Ovo postižemo upotrebom Android API klase LocationManager koja se oslanja na tehnike opisane u prethodnom poglavlju. Koordinate koje je

LocationManager pribavio treba povezati sa imenima zemljopisnih položaja, konkretno ulica. Ovaj zadatak obavlja kod ostvaren upotrebom klase GeoCoder. Idući korak u izvođenju aplikacije je pribavljanje podataka o registarskoj tablici automobila. Ovo se ostvaruje jednostavnim unosom podataka od strane korisnika putem korisničkog sučelja. Trenutna verzija aplikacije tablicu ne pohranjuje trajno ali implementacija mogućnosti pohrane podataka o tablicama je jednostavno ostvariva tehnikama opisanim u kasnijem poglavlju o trajnoj pohrani podataka. Nakon što imamo ime ulice u kojoj se korisnik nalazi i registarsku tablicu automobila još treba samo usporediti naziv prepoznate ulice sa podacima o zonama naplate parkinga prema ulica i iz tih podataka dobiti telefonski broj na koji treba poslati SMS poruku za naplatu parkinga.

Arhitektura aplikacije prikazana je na dijagramu niže. Pravokutnici označavaju xml datoteke u kojima je sadržano korisničko sučelje dok su klase koje sadržavaju funkcionalnost aplikacije označene rombovima.



Slika 14 Arhitektura aplikacije SmsParking. Izvor: vlastiti rad autora

5.1 LocationManager

Location manager klasa omogućava implementaciju ranije spomenutih servisa za utvrđivanje geografskog položaja. Niže navedeni kod koji će biti detaljnije pojašnjen realizira prvi korak izvođenja aplikacije SmsParking – utvrđivanje položaja. Uz LocationManager za realizaciju ovog zadatak važno je i LocationListener sučelje[28].Klasa GPSTracker kojom smo realizirali utvrđivanje geografskog položaja

[28] Usp. Android documentation LocationListener:
https://developer.android.com/reference/android/location/LocationListener.html

nasljeđuje klasu Service, u svojim metodama koristi klasu LocationMAnager i implementira sučelje LocationListener. Service je apstraktna klasa koja omogućava realizaciju objekta kao servisa. Servisi su objašnjeni u ranijem poglavlju. Za prezentiranu klasu ovo važno zbog LocationManagera (koji je servis).

Da bi aplikacija mogla koristiti lokacijske servise to jest LocationManager u manifest datoteci je potrebno zatražiti dozvole za korištenje istih. Ovo postižemo kodom:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.INTERNET" />
```

Dozvola ACCESS_FINE_LOCATION obuhvaća korištenje GPS i Network lokacijskih sustava (ACCESS_COARSE_LOACTION dozvoljava korištenje samo Network lokacijskog servisa) ali za potpunu funkcionalnost Network provider servisa on mora kontaktirati Googlove poslužitelje i prikupiti i podatke o geografskoj lokaciji WiFi mreža pa je nužna i INTERNET dozvola (koja aplikaciji omogućava pristup Internetu).

Dalje je naveden sadržaj datoteke GPSTracker.java. Ovakva implementacija Location Manager-a i LocationListenera ostvarena je prema uputama na AnroidHiveu[29]. Uz očekivane funkcionalnosti kod je naslijedio i mogućnost bilježenja stavki o korištenju lokacijskih servisa u dnevnik. Nažalost SmsParking nije do kraja implementirao ovu funkcionalnost.

```
package ffzg.mobapl.vb.smsparking;

import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.os.IBinder;
import android.util.Log;


public class GPSTracker extends Service implements LocationListener {

    // // Pribavi ime klase, koristiti će za stavke u dnevniku, nevažno ali
ostavljeno kao referenca
    private static String TAG = GPSTracker.class.getName();

    private final Context mContext;

    // Pripremi bool za provjeru stanja GPS provider servisa
    boolean isGPSEnabled = false;

    // Pripremi bool za provjeru stanja network provider servisa
    boolean isNetworkEnabled = false;
```

[29] Usp. AndroidHive Android GPS, Location Manager Tutorial http://www.androidhive.info/2012/07/android-gps-location-manager-tutorial/

```
    // Pripremi bool za provjeru stanja GPS tracker servisa
    boolean isGPSTrackingEnabled = false;

    // pripremi lokalne varijable za pohranu geografskog položaja
    Location location;
    double latitude;
    double longitude;


    // Minimalna udaljenost nakon koje treba ažurirati položaj - korisno pri
praćenju ovdje ostavljeno kao referenca
    private static final long MIN_DISTANCE_CHANGE_FOR_UPDATES = 10; // 10
metara

    // Minimalno vrijeme izmedju ažuriranja geografskog položaja
    private static final long MIN_TIME_BW_UPDATES = 1000 * 60 * 1; // 1
minuta

    // Deklaracija LocationManagera
    protected LocationManager locationManager;

    // SVarijabla za pohranu podataka o pružatelju geografskih informacija
pri korištenju NetworkProvidera (GSM pružatelji usluga)
    private String provider_info;


    //Deklaracija GPS trackera - nevažno ostavljeno kao referenca
    public GPSTracker(Context context) {
        this.mContext = context;
        getLocation();
    }

    /**
     * Pokušaj dobivanja geografskog položaja upotrebom GPS ili Network
providera
     */
    public void getLocation() {

        try {
            locationManager = (LocationManager)
mContext.getSystemService(LOCATION_SERVICE);

            //Provjeri stanje GPS providera
            isGPSEnabled =
locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);

            //Provjeri stanje Network Providera
            isNetworkEnabled =
locationManager.isProviderEnabled(LocationManager.NETWORK_PROVIDER);

            // Pokušaj dobiti lokaciju preko GPS providera
            if (isGPSEnabled) {
                this.isGPSTrackingEnabled = true;
                //dodaj stavku u dnevnik - nevažno
                Log.d(TAG, "Koristim GPS provider");
```

```
                // aplikacija će koristit gps provider. ovo bilježimo u
varijablu


                provider_info = LocationManager.GPS_PROVIDER;

            } else if (isNetworkEnabled) { // samo ako GPS provider ne radi
pokušaj koristiti network provider
                this.isGPSTrackingEnabled = true;

                Log.d(TAG, "Koristim network provider");

                // aplikacija će koristit network provider. ovo bilježimo u
varijablu

                provider_info = LocationManager.NETWORK_PROVIDER;

            }

            // ako smo ranije zabilježili providera možemo od njega pokušati
dobiti geografski položaj

            //prvo aktiviramo praćenje položaja u LocationManageru
            if (!provider_info.isEmpty()) {
                locationManager.requestLocationUpdates(
                        provider_info,
                        MIN_TIME_BW_UPDATES,
                        MIN_DISTANCE_CHANGE_FOR_UPDATES,
                        this
                );

                // ako Location Manager funkcionira lokaciju možemo zapisati
u varijablu tipa location
                if (locationManager != null) {
                    location =
locationManager.getLastKnownLocation(provider_info);
                    updateGPSCoordinates();
                }
            }
        }
        catch (Exception e)
        {
            //e.printStackTrace();
            Log.e(TAG, "LocationManager nije dostupan", e);
        }
    }

     // metoda za ažuriranje koordinata GPS trackera

    public void updateGPSCoordinates() {
        if (location != null) {
            latitude = location.getLatitude();
            longitude = location.getLongitude();
        }
    }

    // getter/setter metoda latitude
```

```
    public double getLatitude() {
        if (location != null) {
            latitude = location.getLatitude();
        }

        return latitude;
    }

    // getter/setter metoda longitude

    public double getLongitude() {
        if (location != null) {
            longitude = location.getLongitude();
        }

        return longitude;
    }

    // getter metoda za provjeru trackera

    public boolean getIsGPSTrackingEnabled() {

        return this.isGPSTrackingEnabled;
    }

    // metoda za zaustavljanje trackera
    public void stopUsingGPS() {
        if (locationManager != null) {
            locationManager.removeUpdates(GPSTracker.this);
        }
    }

    //implementacija LocationListener metoda

    @Override
    public void onLocationChanged(Location location) {
    }

    @Override
    public void onStatusChanged(String provider, int status, Bundle extras) {
    }

    @Override
    public void onProviderEnabled(String provider) {
    }

    @Override
    public void onProviderDisabled(String provider) {
    }

    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
}
```

5.2 GeoCoder

Geokodiranje (engl. Geocoding) je postupak povezivanja imena zemljopisnog položaja, poput imena ulice, grada, planine, rijeke ili drugog sa geografskim koordinatama. Obrnuto geokodiranje (engl. Reverse geocoding) je obrnuti postupak, dakle povezivanje ili prevođenje geografskih koordinata u najbliži imenovani geografski položaj. Za potrebe aplikacije opisane u ovom radu nužno je implementirati postupak obrnutog geokodiranja, geografske koordinate (latituda, longituda) koje pribavlja modul opisan u ranijem poglavlju potrebno je povezati sa imenom ulice koja se kasnije može povezati i sa zonom naplate parkinga.

Klasa Geocoder dio je andriod.location APIja[30] te omogućava geokodiranje i obrnuto geokodiranje, postupak koji koristi opisana aplikacija. Ovo poglavlje će prikazati implementaciju Geocoder klase u SMS-parking aplikaciji. Niže prikazani kod uz Geocoder klasu koristi još jednu specifičnost Android APIja, klasu Address[31]. Adress klasa se koristi za opis geografske adrese u kodu, oblik zapisa je pojednostavljena verzija xAL(engl. extensible address language) oblika zapisa adresa[32]. Aplikacija u objekt klase Adress pohranjuje adresu prepoznatu po geografski koordinatama te iz nje dohvaćamo naziv ulice.

U aplikaciji SMS-parking postupak geokodiranja je implementiran u vratiUlicu metodi MainActivity klase. Metoda prihvaća dvije double vrijednosti i vraća i naziv ulice u obliku niza znakova (engl. string). Sljedeći kod deklarira metodu koja će odraditi geokodiranje za aplikaciju:

```
private String vratiUlicu(double lat, double lon) throws IOException {
      String ulica = "";
```

Cijela metoda smještena je u try-catch blok jer su tijekom testiranja (posebno na virtualnim Android uređajima) uočeni česti problemi sa upravom ovom metodom.

```
      try{
            Geocoder geokoder = new Geocoder(this, Locale.ENGLISH);
            List<Address> adrese = geokoder.getFromLocation(lat, lon, 1);
```

Deklariran je novi Geocoder objekt i odmah je preuzeta lista adresa povezanih sa predanom latitudom i longitudom upotrebom getFromLocation metode. Zadnja vrijednost predana geokoderu (broj 1) je konstanta koja označava najveći broj rezultata koje želimo primiti. Geocoder može vratiti listu lokacija povezanih sa predanim geografskim koordinatama u kojem slučaju su one poredane po blizini i važnosti (npr. uz određene koordinate može biti povezano ime ulice te naziv kazališta i fakulteta koji se nalaze u neposrednoj blizini).

---

[30] Usp. Android Developer Geocoder dokumentacija: https://developer.android.com/reference/android/location/Geocoder.html
[31] Usp. Android Developer Address dokumentacija: https://developer.android.com/reference/android/location/Address.html
[32] OASIS CIQ v2.0 committee specifications/standards: https://www.oasis-open.org/committees/ciq/ciq.html

```
            if ((address != null)&&(address.size()!=0)) {
                  Address dohvacenaAdresa = address.get(0);
                  ulica=dohvacenaAdresa.getAddressLine(0).toString();

            /*StringBuilder nizAdresa = new StringBuilder();
            for(int i=0; i<dohvacenaAdresa.getMaxAddressLineIndex(); i++)
            {
            nizAdresa.append(dohvacenaAdresa.getAddressLine(i)).append("\n");
            }*/
            }

            else {
                  ulica = "Nije prepoznata niti jedna ulica";
            }
      }
```

Gornji kod provjerava jesmo li smo primili rezultat koji ima adresu (tj. ulicu). U zakomentiranim linijama(/* */) je ostavljen kod iz ranije verzije aplikacije koja je primala više rezultata te klasom StringBuilder[33] i njezinom metodom append sastavljala niz od više ulica (adresa). testiranje je pokazalo da je ovakav postupak nepotreban i da je dovoljan prvi rezultat ali je kod ostavljen kao primjer.

```
      catch(IOException e){
            // TODO Auto-generated catch block
            e.printStackTrace();
            Toast.makeText(getApplicationContext(), "Nije moguće dobiti
            adresu!\n(možda ne radi geocoder)", Toast.LENGTH_LONG).show();
      }
      return ulica;
}
```

Metoda vraća ulicu ili Toast porukom obavještava korisnika da je došlo do problema sa postupkom geokodiranja. Klasa Geocoder i njezine metode se oslanjaju na vanjski web servis te u slučaju da on nije dostupan ne funkcioniraju. Ovo se tijekom razvoja aplikacije pokazalo kao značajan problem pri testiranju na virtualnim uređajima. Ovi uređaji su imali implementirani Android API a sa njim i Geocoder klasu ali ona nije komunicirala sa vanjskim servisima te nije bilo moguće dobiti adresu bez obzira na navodnu prisutnost i ispravan rad Geocoder klase. Prisutnost Geocoder klase je moguće provjeriti upotrebom isPresent() metode koja vraća boolean vrijednost koja će biti True ako je Geocoder prisutan. Ipak ova provjera ne garantira da Geocoder zaista komunicira s Google APIjem koji omogućava postupak geokodiranja i normalno funkcioniranje klase. Komunikacija sa vanjski web baziranim APIjem pretpostavlja da je aplikaciji dozvoljena komunikacija putem Interneta te je za normalni rad Geocoder objekta potrebno putem manifest datoteke zatražiti dozvolu za pristup Internetu. Konkretno u AndroidManifest.xml mora biti prisutan sljedeći kod:

```
<uses-permission android:name="android.permission.INTERNET" />
```

[33] Usp. Android Developer String Builder: https://developer.android.com/reference/java/lang/StringBuilder.html

5.3 GoogleMaps API

Google maps API je javno dostupan servis koji je do neke granice upotrebe (ograničen je maksimalnim brojem pristupa sa svakog API ključa) besplatan. Osnovna funkcionalnost aplikacije SmsParking je ostvarena kroz ranije opisane lokacijske servise i u SmsParking aplikaciji dodatak GoogleMaps-a je demonstracijske prirode. Dodatna funkcionalnost koja se implementacijom GoogleMaps-a ostvaruje jest mogućnost da korisnik vidi prepoznatu lokaciju na karti i tako dodatno potvrdi da su lokacijski servisi odabrali ispravni geografski položaj.

Ugradnja GoogleMaps-a u mobilne aplikacije je vrlo jednostavna u slučaju razvoja aplikacije upotrebom Android Studia. Android Studio može sam dodati aktivnost tipa GoogleMaps i sam će odraditi većinu posla. U suprotnom dokumentacija API je vrlo detaljna i lako je i njenom uporabom ugraditi GoogleMaps u aplikaciju[34].

Ovdje je naveden i ukratko, komentarima, objašnjen kod iz MapsActivity.java datoteke koji realizira komunikaciju sa Google API-em i prikazivanje mape sa prepoznatom lokacijom korisnika u mapsActivity.xml.

```
package ffzg.mobapl.vb.smsparking;

import android.content.Intent;
import android.support.v4.app.FragmentActivity;
import android.os.Bundle;
import android.view.View;

import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;

public class MapsActivity extends FragmentActivity implements
OnMapReadyCallback {

    private GoogleMap mMap;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        // preuzmi google maps fragment
        SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager()
                .findFragmentById(R.id.map);
```

[34] Usp. Google Maps APIs https://developers.google.com/maps/documentation/android-api/

```
        mapFragment.getMapAsync(this);
    }



    @Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;

        // kod preuzet iz mainActivity.java komunicira sa GPSTrackerom i
dostavlja geografske koordinate Google mapama
        GPSTracker gpsTracker = new GPSTracker(this);

        double lati=0;
        double longi=0;

        if (gpsTracker.getIsGPSTrackingEnabled())
        {
            String stringLongitude = String.valueOf(gpsTracker.longitude);
            String stringLatitude = String.valueOf(gpsTracker.latitude);

            lati=gpsTracker.latitude;
            longi=gpsTracker.longitude;
        }


        // Dodaj marker na mapi prema položaju korisnika
        LatLng lokacija = new LatLng(lati, longi);
        mMap.addMarker(new MarkerOptions().position(lokacija).title("Vaš
položaj"));

        //pomakni kameru na marker i zoomiraj je na razinu ulice (razina
zooma 15)
        mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(lokacija,15));
    }

    //metoda koju poziva tipka iz korisničkog sučelja koja vraća korisnika na
početni ekran
    public void nazadNaMain(View view){
        Intent intent = new Intent(this, MainActivity.class);
        startActivity(intent);

    }
}
```

Uz gore navedeni kod ugrađivanje GoogleMaps aktivnosti upotrebom Android Studia dodati će i novu xml datoteku /res/values mapu – google_maps_api.xml. Ova datoteka sadrži podatke o API ključu korištenom za pristup APIu.

```
<resources>

      <string name="google_maps_key" templateMergeStrategy="preserve"
      translatable="false">
      OvdjeSePostavljaKljuč
      </string>
</resources>
```

Datoteka, u komentarima, sadrži i upute za dobivanje ključa koji je moguće dobiti i na web stranici Google Developers Console[35].

5.4 SMSManager

Zadnji zadatak koji kod mora ostvariti jest slanje SMS poruke na prepoznati telefonski broj. Ovo postižemo uz pomoć SMSManager klase. Slanje SMSa je u aplikaciji SmsParking u potpunosti realizirano unutra MainActivity klase te je ovdje predstavljen i objašnjen dio koji je odgovoran za slanje SMS poruka.[36]Veći dio koda u ovom dijelu se bavi intentima koji će pripremiti SMS slanje i rezultat slanja kroz toast poruku. Sam SMSManager je predstavljen na kraju bloka.

Kod se pokreće kroz onClickListner metodu, to jest, pokreće se kao reakcija na „dodir“ korisnika na element korisničkog sučelja tipa Button i ID-a btnSMS.

```
this.btnSMS.setOnClickListener(new View.OnClickListener() {

      @Override
      public void onClick(View v) {
            try {
                  String SENT = "sent";
                  String DELIVERED = "delivered";

                  Intent sentIntent = new Intent(SENT);
                  PendingIntent sentPI =PendingIntent.getBroadcast
                  (getApplicationContext(), 0, sentIntent,
                  PendingIntent.FLAG_UPDATE_CURRENT);
```

[35] Usp. Google Maps APIs – Get API key https://developers.google.com/maps/documentation/android-api/signup
[36] Usp. Andriod Developer documantation SmsManager https://developer.android.com/reference/android/telephony/SmsManager.html

```
                    Intent deliveryIntent = new Intent(DELIVERED);
                    PendingIntent deliverPI =
PendingIntent.getBroadcast(getApplicationContext(), 0, deliveryIntent,
                    PendingIntent.FLAG_UPDATE_CURRENT);


                    registerReceiver(new BroadcastReceiver() {
                          @Override
                          public void onReceive(Context context, Intent intent)
                                {
                                String result = "";
                                switch (getResultCode()) {
                                      case Activity.RESULT_OK:
                                            result = "SMS je poslan";
                                            break;
                                      case
      SmsManager.RESULT_ERROR_GENERIC_FAILURE:
                                            result = "SMS nije poslan";
                                            break;
                                      case SmsManager.RESULT_ERROR_RADIO_OFF:
                                            result = "Ugašena je antena";
                                            break;
                                      case SmsManager.RESULT_ERROR_NULL_PDU:
                                            result = "PDU mod nije definiran";
                                            break;
                                      case SmsManager.RESULT_ERROR_NO_SERVICE:
                                            result = "Nema signala";
                                            break;
                                      }
                                Toast.makeText(getApplicationContext(), result,
                                Toast.LENGTH_LONG).show();
                                }
                          }, new IntentFilter(SENT));

                          registerReceiver(new BroadcastReceiver() {
                          @Override
                          public void onReceive(Context context, Intent intent)
                                {
                                Toast.makeText(getApplicationContext(),
                                "Delivered", Toast.LENGTH_LONG).show();
                                }

                    }, new IntentFilter(DELIVERED));
```

Kod niže instancira objekt klase SmsManager i šalje poruku. U argumentima koje prima sendTextMessge metoda smsManager objekta sadržane su varijednosti koje je raniji kod osigurao. Oni su redom: telefonski broj na koji se šalje poruka, null polje (inače „service center address“, sadržaj poruke (u ovom slučaju broj registarske tablice), intent slanja poruke, intent primanja potvrde o primitku.

```
                    SmsManager smsManager = SmsManager.getDefault();
                    smsManager.sendTextMessage(Tel, null, Tablica, sentPI,
                    deliverPI);

                    } catch (Exception ex) {
                          Toast.makeText(getApplicationContext(),
ex.getMessage().toString(), Toast.LENGTH_LONG).show();
                          ex.printStackTrace();
                    }
              }
      });
}
```

5.5 Korisničko sučelje

Kako je ranije navedeno korisničko sučelje aplikacije realizirano je kroz tri ekrana to jest aktivnosti ili xml datoteke. U ovom poglavlju objasniti ćemo postupak izrade korisničkog sučelja upotrebom xml datoteka i način komunikacije .java datoteka (i klasa koje sadrže) sa elementima korisničkog sučelja (razne tipe, textBox-evi i slično). Android realizira korisničko sučelje kroz datoteke koje opisuju raspored elementa sučelja (engl. layout). Iduće dvije slike zorno prikazuju elemente korisničkog sučelja čija će realizacija biti objašnjena u daljnjem tesktu.

Slika 15 mainActivity.xml korisničko sučelje. Izvor: vlastiti rad autora

Slika 16 infoActivity.xml korisničko sučelje. Izvor: vlastiti rad autora

5.5.1 XML i java activity datoteke

U ovom poglavlju objašnjena je izrada layout xml datoteka interpretacijom kojih Android formira korisničko sučelje. xml (engl. extensible markup language) je standard zapisa koji služi sa opis i pohranu podataka [37]. Osnovni elementi xml datoteka su deklaracije, elementi i pod elementi. Pogledajmo primjer xml layout datoteke activity_main.xml.

Datoteka počinje definiranjem vršnog elementa, koji je u ovom slučaju LinerLayout i sadrži deklaraciju (**xmlns:android, xmlns:tools)** te osnovne podatke o sučelju.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="${relativePackage}.${activityClass}"
    android:orientation="vertical"
    android:padding="10dp"
    android:weightSum="1"
    android:background="@drawable/bckstyle">
```

Prvi element sučelja je tipka (engl button). Osim osobina koje grafički opisuju tipku važno je uočiti **android:onClick** tag koji poziva metodu MainActivity klase. MainActivity klase pak svojom onCreate metodom pozvati ovu layout datoteku i iscrtati ekran sa korisničkim sučeljem. MainActivity klasu ćemo razmotriti malo kasnije.

Drugi važni element **android:id**. Ovaj element dodjeljuje identifikacijski string (u ovom slučaju „btnInfo“ preko kojeg će biti moguće referencirati ovaj element sučelja u ostatku koda.

```
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_info"
        android:layout_gravity="center_horizontal"
        android:id="@+id/btnInfo"
        android:layout_marginBottom="50dp"
        android:onClick="otvoriInfo"
        />
```

Uz ranije naglašeni onclik i id tag važno je objasniti i **android:text="@string/button_info"** ova linija koda pokazuje Android u kojem resursu će naći string, to jest tekst, koji treba smjestiti na tipku. U ovom slučaju to stavka button_info string.xml datatoke. Varijable koje ne pohranjuju sadržaj direktno pokazuju na vanjski resurs (što je ispravan način pisanja XML layout datoteka) počinju sa znakom @ koji popraćen imenom datoteke i stavke u njoj koju referenciramo.

---

[37] Usp. Jan Egil Refsnes Introduction to XML http://www.xmlfiles.com/xml/

Idući element je ugniježđeni LinearLayout element koji navodi korisnika da unese tablicu vozila za koje želi platiti parking. Ovo je realizirano TextView elementom koji ima zadatak ispisivanja statičnog teksta na ekranu.

```
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:weightSum="1"
    android:layout_gravity="center_horizontal">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Unesite tablicu automobila:"
        android:textColor="#010101"
        android:textSize="17dp"
        android:textIsSelectable="false"
        android:layout_gravity="center" />
</LinearLayout>
```

Element EditText dozvoljava unos teksta. Ovo je jedan od uobičajenih načina prijenosa podataka od korisnika do aplikacije koji nije moguć upotrebom unaprijed definiranih komandi.

```
<EditText
    android:id="@+id/etTab"
    android:layout_marginLeft="15dp"
    android:layout_marginTop="15dp"
    android:layout_marginBottom="20dp"
    android:layout_marginRight="15dp"
    android:layout_centerVertical="true"
    android:layout_width="151dp"
    android:layout_height="38dp"
    android:ellipsize="start"
    android:gravity="center"
    android:hint="Tablica"
    android:inputType="text"
    android:background="@drawable/edittextstyle"
    android:layout_gravity="center_horizontal" />
```

Slijedi još jedna tipka. Aktiviranje ove tipke će javiti aplikaciji da započne proces geokodiranja. Način aktivacije je drukčiji u odnosu na prvu tipku koja je pozivala metodu. Detekcija aktivacije ove tipke se realizira OnClickListener metodom View klase. Ovo će biti objašnjeno malo kasnije.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_lok"
    android:layout_gravity="center_horizontal"
    android:id="@+id/btnLokacija"
    android:layout_marginTop="50dp"
    android:layout_marginBottom="25dp"
    />
```

Sljedeći element uvodi novi LinearLayout koji se sastoji od dva TextView elementa. Potrebno je primijetiti **android:orientation="horizontal"** liniju koja omogućava da elementi koji su ugniježđeni u ovaj LinearLayout budu poredani vodoravno a ne okomito kako je bio slučaj u vršnom LinearLayout elementu (**android:orientation="vertical"**).

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    >
    <TextView
        android:layout_width="150dp"
        android:layout_height="wrap_content"
        android:text="@string/label_lat"/>
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/tvLat"
        />
</LinearLayout>
```

Iduća tri LinearLayout elementa imaju isti oblik kao i prethodni (dva ugniježđena TextView elementa). Zajedno ova četiri elementa omogućuju prikaz prepoznatih geografskih podataka i parking zone.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <TextView
        android:layout_width="150dp"
        android:layout_height="wrap_content"
        android:text="@string/label_lon"/>
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
        android:id="@+id/tvLon"
        />
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <TextView
        android:layout_width="150dp"
        android:layout_height="wrap_content"
        android:text="@string/label_ispis_lok"/>
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/tvLok"
        />
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <TextView
        android:layout_width="150dp"
        android:layout_height="wrap_content"
        android:text="@string/label_tel"/>
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/tvTelBr"
        />
</LinearLayout>
```

Zadnja dva elementa ovog sučelja su dvije tipke. Prva otvara novi ekran (sa Google Maps aktivnošću) i ovo realizira upotrebom **android:onClick="otvorikartu"** poziva metode Mainactivity klase a druga, koja šalje SMS poruku za naplatu parkinga, je realizirana unutar klase kroz upotrebu OnClickListener metode.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_karta"
    android:layout_gravity="center_horizontal"
    android:id="@+id/btnMaps"
    android:layout_marginBottom="50dp"
    android:onClick="otvorikartu"
    />
```

```
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_sms"
        android:layout_gravity="center_horizontal"
        android:id="@+id/btnSMS"
        android:layout_marginTop="25dp"/>

</LinearLayout>
```

Gornji layout kod, to jest xml datoteka, datoteka mora biti pozvana unutar onCreate() metode neke klase. U ovom slučaju to je MainActivity koja je objašnjena niže.

Klasa započinje deklaracijom paketa kojega je dio i popisom Andriod klasa koje će koristiti. U ovom slučaju to su mahom klase koje se bave korisničkim sučeljem poput EditText i TextText.

```
package ffzg.mobapl.vb.smsparking;


import android.app.Activity;
import android.app.PendingIntent;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.location.Address;
import android.location.Geocoder;
import android.os.Bundle;
import android.telephony.SmsManager;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Locale;
```

Klasa MainActicity nasljeđuje klasu Activity te prima njene metode od kojih su za ovu aplikaciju važne onCreate() i onClickListener().

```
public class MainActivity extends Activity {
```

U idućem koraku deklariramo varijable koje će klasa koristiti. Ovdje je dobro primijetiti da su svi elementi korisničkog sučelja koji su na neki način interaktivni, kao na primjer tipke, TextView polja koja će mijenjati sadržaj i EditText polje koje koristimo za unos podataka već ovdje definirani (ali samo njih tip, još nisu povezani sa elementima sučelja).

```
TextView tvLat;
TextView tvLon;
EditText etTab;
Button btnLokacija;
Button btnSMS;
TextView tvLok;
TextView tvTelBr;
String Tel = "";
String Tablica = "nema tablice";
double lati=0;
double longi=0;
```

OnCreate metoda učitava layout datoteku koju smo ranije opisali – activity_main.xml.

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

U ovom koraku povezujemo varijable sa elementima korisničkog sučelja koristeći findviewById metodu. ID vrijednosti preko kojih referenciramo elemente (npr. ***tvLat)*** odgovaraju vrijednostima **android:id** polja iz activity_main.xml datoteke.

```
    this.tvLat = (TextView) this.findViewById(R.id.tvLat);
    this.tvLon = (TextView) this.findViewById(R.id.tvLon);
    this.btnLokacija = (Button) this.findViewById(R.id.btnLokacija);
    this.btnSMS = (Button) this.findViewById(R.id.btnSMS);
    this.tvLok = (TextView) this.findViewById(R.id.tvLok);
    this.tvTelBr = (TextView) this.findViewById(R.id.tvTelBr);
    this.etTab = (EditText) this.findViewById(R.id.etTab);
```

Instanciramo novi gpsTracker prema ranije opisanoj klasi GPSTracker te, ako gpsTracker javlja da je sve u redu preko svoje getISGPSTrackingEnabled metode, preuzimamo zemljopisnu širinu i dužinu te ih odmah upisujemo u relevantne elemente korisničkog sučelja.

```
    GPSTracker gpsTracker = new GPSTracker(this);
```

```
if (gpsTracker.getIsGPSTrackingEnabled())
{
    String stringLongitude = String.valueOf(gpsTracker.longitude);
    String stringLatitude = String.valueOf(gpsTracker.latitude);
    tvLat.setText(stringLatitude);
    tvLon.setText(stringLongitude);
    lati=gpsTracker.latitude;
    longi=gpsTracker.longitude;
}
```

Idući dio koda implementira onClickListener metodu koja je jedan od dva u ovom radu prikazana načina praćenja stanja tipki u korisničkom sučelju. Kada korisnik pritisne tipku koja ima ID **btnLokacija** donji blok koda će se pokrenuti.

```
this.btnLokacija.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
```

Prvi korak u izvođenju ovog bloka koda je preuzimanje vrijednosti editText elementa i pohrana ove vrijednosti u lokalnu varijablu koja će kasnije biti poslana SMS porukom na telefonski broj za naplatu parkinga.

```
Tablica = etTab.getText().toString();


String ulica = null;
```

Pozivamo metodu vratiUlicu, koja će odraditi postupak Geokodiranja i opisana je u ranijem poglavlju. Dobivenu ulicu prikazujemo u elementu korisničkog sučelja **tvLok.**

```
try {
    ulica = vratiUlicu(lati, longi);
}
catch (IOException e) {
    e.printStackTrace();
}

tvLok.setText(ulica);
```

Nakon metode vratiUlicu pozivamo telBrojZone koja čita podatke iz /res/raw/ulice datoteke i opisana je kasnije. Ova metoda će pribaviti telefonski broj na koji će biti poslana SMS poruka. I ovaj broj je prikazan u odgovarajućem elementu korisničkog sučelja.

```
        try {
          Tel = telBrojZone(ulica);
        } catch (IOException e) {
          // TODO Auto-generated catch block
          e.printStackTrace();
        }

        tvTelBr.setText(Tel);

      }
    });
```

Sadržaj OnClickListenera za btnSMS je opisan u ranijem poglavlju koje se bavi SMSManager klasom i njenom implementacijom.

```
        this.btnSMS.setOnClickListener(new View.OnClickListener() {
        };
    }
```

Iduće dvije metode su, kako je ranije navedeno, opisane u drugim poglavljima te ih ovdje preskačemo.

```
    private String vratiUlicu(double lat, double lon) throws IOException {
    }

    public String telBrojZone(String ul) throws IOException {
    }
```

Zadnje dvije metode u klasi ActivityMain su metode koje koriste tipke koje se ne oslanjaju na onClickActivityListener. Ove metode deklariraju intente te njihovom upotrebom aktiviraju nove aktivnosti (ekrane korisničkog sučelja infoActivity i MapsActivity).

```
    public void otvoriInfo(View view){
        Intent intent = new Intent(this, infoActivity.class);
        startActivity(intent);

    }
```

```
    public void otvorikartu(View view){
        Intent intent = new Intent(this, MapsActivity.class);
        startActivity(intent);

    }

}
```

Idući element korisničkog sučelja jest InfoActivity aktivnost koja je očekivano najjednostavniji od tri ekrana preko kojih korisnik komunicira sa aplikacijom. XML datoteka definira 4 textview polja i jednu tipku.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="ffzg.example.prepzonepark.infoActivity">
```

Prvi Textview je naslov upozorenja, jednostavni tekst „Info“. Ovaj tekst je veći od ostatka (25 u odnosu na 17 točaka ostatka teksta).

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Info"
        android:textColor="#010101"
        android:textSize="25dp"
        android:textIsSelectable="false"
        android:layout_gravity="center"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:id="@+id/textView3" />
```

Iduća tri TextView polja prenose upozorenje o nepogodnosti aplikacije izvan obrazovnog okruženja. Za razliku od do sada preporučene metode pohranjivanja stringova u zasebnu datoteku sa jezičnim resursima ovaj ekran pohranjuje svoje stringove direktno u XML datoteku korisničkog sučelja. Ovo nije preporučena metoda rada ali je svakako moguća te je ovdje i demonstrirana.

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Ovaj app je dio diplomskog rada na odsjeku za Inf.
znanosti FFZG-a, te je kao takav predviđen za demostraciju tehnologije i
studentovog vladanja istom a ne za pravu upotrebu."
        android:textColor="#010101"
        android:textSize="17dp"
        android:textIsSelectable="false"
        android:layout_gravity="center"
        android:id="@+id/textView2"
        android:layout_marginTop="19dp"
        android:layout_below="@+id/textView3"
        android:layout_alignParentLeft="true" />


    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Zbog loše dokumentacije o ulicama i zonama naplate
autor mora ručno prepisati ulice i odgovarajuće zone. Kako ovo nije dio ni
tehnologija ni znanja vezanog uz razvoj mobilnih aplikacije (niti
geolokacijskih servisa) u listi ulica trenutno postoje samo četiri ulice:
Lučićeva, Koranska, Koturaška i Zelinska."
        android:textColor="#010101"
        android:textSize="17dp"
        android:textIsSelectable="false"
        android:layout_gravity="center"
        android:id="@+id/textView"
        android:layout_centerVertical="true"
        android:layout_alignParentLeft="true" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="U budućnosti app bi se trebao oslanjati na on-line bazu
podataka a ne na statičnu listu ulica i tada će možda biti opremljen
potpunijom listom ulica. Ali to nije slučaj u ovoj verziji."
        android:textColor="#010101"
        android:textSize="17dp"
        android:textIsSelectable="false"
        android:layout_gravity="center"
        android:id="@+id/textView4"
        android:layout_below="@+id/textView"
        android:layout_alignParentLeft="true"
        android:layout_marginTop="18dp" />
```

Aktivnost activity_info završava tipkom koja omogućava povratak na početni ekran. Tipka ovo postiže pozivom metode „nazad“.

```
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_bck"
        android:layout_gravity="center_horizontal"
        android:id="@+id/btnBck"
        android:onClick="nazad"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true" />

</RelativeLayout>
```

Java datoteka i klasa povezana sa activity_info aktivnošću sadrži samo onCreate metodu koja učitava activity_info xml datoteku i „nazad“ metodu koja pokreće aktivnost MainActivity i time omogućava povratak na početni ekran.

```
package ffzg.mobapl.vb.smsparking;

import android.content.Intent;
import android.os.Bundle;
import android.app.Activity;
import android.view.View;

public class infoActivity extends Activity {

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

    public void nazad(View view){
        Intent intent = new Intent(this, MainActivity.class);
        startActivity(intent);
    }
}
```

Treći ekran korisničkog sučelja je također vrlo jednostavan. Ovaj element sastoji se od fragmenta koji je dodao Android studio prilikom stvaranja Google Maps aktivnosti i tipke za povratak na glavni ekran koju je dodao autor. Oba elementa u ugniježđena u FrameLayout element. Ovakva konfiguracija sučelja rezultira tipkom koja je smještena preko fragmenta koji prikazuje kartu.

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MapActivity" >

    <Button
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_nazadKarta"
        android:layout_gravity="center_horizontal"
        android:id="@+id/btnnazadnaMain"
        android:layout_marginBottom="50dp"
        android:onClick="nazadNaMain"
        />

    <fragment xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:map="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:id="@+id/map"
        android:name="com.google.android.gms.maps.SupportMapFragment"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:scrollbars="vertical"
        tools:context="ffzg.mobapl.vb.smsparking.MapsActivity" />
</FrameLayout>
```

Klasa koja učitava ovu aktivnost objašnjena je u poglavlju o GoogleMaps APIu.

5.6 Distribucija i pohrana podataka na Android uređajima

Iz više razloga preporučljivo je odvajati podatke od koda aplikacija, ne samo u slučaju mobilnih aplikacija. Eksternalizacijom resursa omogućava se lakše održavanje isti (nadogradnje i izmjene podataka) i jednostavniji proces lokalizacije (jer se sadržaji mogu lokalizirati bez diranja izvornog koda. U slučaju Androida i mobilnih aplikacija izuzetno je važna i mogućnost dostavljanja podataka u više različitih oblika za prikaz na zaslonima različitih dimenzija. Android većinu resursa smješta u /res mapu gdje su stringovi pohranjeni u res/values/strings.xml datoteci a ikone i grafike raznih dimenzija u res/drawable/ mapi itd.

Međutim kako pohraniti veće količine podataka? Pri razvoju web i desktop aplikacija često posežemo za vanjskim web servisima i poslužiteljima bazi podataka. Ovakva rješenja moguća su i u slučaju mobilnih Android aplikacija ali s obzirom da je podatkovni promet još uvijek često ograničen ili skup preporučljivo je podatke (ako ih nije potrebno dijeliti i ne mijenjaju se često) distribuirati sa samom aplikacijom.

Android nudi nekoliko rješenja ovog problema i ona će biti prezentirana u ovom poglavlju. Za potrebe aplikacije SmsParking potrebno je distribuirati popise koji svrstavaju ulice u zone parkinga to jest povezuju nazive ulica sa telefonskim brojem na koji treba poslati SMS poruku kojom se plaća parking u određenoj zoni. Trenutna verzija aplikacije ovo rješava smještanjem datoteke sa relevantnim podacima u res/raw mapu Android aplikacije.

5.6.1 Android Resursi

Uobičajeni način pohrane resursa na Androidu podrazumijeva pohranu datoteka u /res mapu i njene podmape[38]. Stuktura ove mape nije striktno odeređena ali sadrži barem sljedeće mape:

```
src/
        MyActivity.java
    res/
        drawable/
            graphic.png
        layout/
            main.xml
            info.xml
        mipmap/
            icon.png
        values/
            strings.xml
```

Gore prikazana struktura sadrži jednu aktivnost sa pripadajućom .java datotekom, i ..xml datotekama u /layout mapi. Ovi elementi čine osnovu same aplikacije a njeni osnovni resursi su pohranjeni u /drawable, /mipmap i /values mapama. Iduća tablica pojašnjava sadržaje uobičajenih mapa.

animator/	XML datoteke sa definicijama animacija
anim/	XML datoteke sa definicijama „tween“ animacija
color/	XML datoteke koje sadrže definicije boja za korisničko sučelje.
drawable/	Slike (bitmape u obliku .png, .jpg, .gif) ili XML datoteke koje sadrže slike
mipmap/	Slike (bitmape) koje sadrže ikonu za pokretanje aplikacija u raznim razlučivostima
layout/	XML datoteke koje definiraju korisničko sučelje (aktivnosti)
menu/	XML datoteke koje sadrže definicije izbornika iz korisničkog sučelja
raw/	Proizvoljne datoteke koje koristi aplikacija. Detaljnije opisano u idućem poglavlju.
values/	Razne XML datoteke koje sadrže vrijednosti koje koristi aplikacija, često ali ne nužno u korisničkom sučelju poput raznih nizova znakova (engl. string) i cijelih brojeva (engl. integer)
xml/	Ostale XML datoteke koje će biti čitane tijekom izvođenja aplikacije.

Tablica 2 opis /res mape

Kod niže jest sadržaj values.xml datoteke SmsPArking aplikacije. Dakle vrijednosti stringova koje aplikacija koristi u komunikaciji sa korisnikom. Kako je vidljivo struktura datoteke je jednostavni XML. Nažalost SmsParking (kako je vidljivo u ranijim isječcima koda) sadrži značajan broj stringova i u samom kodu takozvani „hardkodirani“ stringovi. Ovi stringovi su namjerno ostavljeni da bi se demonstrirao i ovakav način pohrane podataka ali ovdje je nužno naglasit da to nije preporučeni način rada već bi sve poruke korisniku trebale biti pohranjene u /values mapi (i tako olakšati procese ažuriranja poruka i lokalizacije).

---

[1] [38] Usp. Android developer Providing resources https://developer.android.com/guide/topics/resources/providing-resources.html

```
<resources>
    <string name="app_name">SMSParking</string>
    <string name="label_lat">Latituda</string>
    <string name="label_lon">Longituda</string>
    <string name="button_lok">Pripremi podatke za SMS</string>
    <string name="label_ispis_lok">Prepoznata ulica:</string>
    <string name="label_tel">Broj za sms:</string>
    <string name="label_tablica">Tablica automobila:</string>
    <string name="button_sms">Pošalji SMS</string>
    <string name="button_info">Prvo pročitajte upozorenje!</string>
    <string name="title_activity_info">infoActivity</string>
    <string name="button_bck">Nazad na app</string>
</resources>
```

5.6.2 Raw

Raw mapa nalazi se u /res mapi Android projekta i njezin sadržaj se distribuira s aplikacijom. Dakle kada se ažurira aplikacija može se ažurirati i sadržaj ove mape ako su se podaci koji su u njoj pohranjeni promijenjeni. Android automatski indeksira sadržaj raw mape, to jest svim datotekama koje su u njoj pohranjene dodaje ID koji se zapisuje u R klasu te ih je moguće u kod referencirati preko toga ID-a (kao što će biti pokazano u kodu niže). Aplikacija SmsParking u raw mapi pohranjuje datoteku „ulice“ koja sadrži imena ulica popraćena zarezom i telefonskim brojem na koji treba poslati SMS poruku za naplatu parkinga na primjer: Zelinska, 0800200. Svaki ovakav zapis nalazi se u vlastitom retku (odvojeni su prekidom linije - \n).Čitanjem podataka iz ove datoteke i usporedbom sa imenom ulice koji je u ranijim koracima dostavio Geocoder ostvaruje se predzadnji korak u izvođenju aplikacije. Za početak je definirana nova metoda koja prima prepoznati naziv ulice (varijabla „ul“) i vraća telefonski broj na koji je potrebno poslati SMS poruku.

```
public String telBrojZone(String ul) throws IOException {
      String brz = "Nije prepoznata parking zona";
      String[] ulArr=ul.split(" ");
```

Upotrebom klasa InputStreamReader[39] i BufferedReader[40] definiramo novi “čitač” datoteke. Valja primijetiti kako se datoteka s popisom ulica ne učitava prema nekoj apsolutnoj putanji već prema ID-u koji je pohranjen u R.raw.

```
            BufferedReader br = new BufferedReader(new
InputStreamReader(this.getResources().openRawResource(R.raw.ulice)));
            String line;
```

[39] Usp. Android developer documenatatio InputStreamReader: https://developer.android.com/reference/java/io/InputStreamReader.html
[40] Usp. Anddroid developer BufferedReader documentation: https://developer.android.com/reference/java/io/BufferedReader.html

Ostatak koda iterira kroz linije učitane datoteke te upotrebom contains metode klase string[41] uspoređuje ulice iz datoteke sa nazivom ulice koji je u ranijem koraku dostavio Geocoder.

```
            while ((line = br.readLine()) != null) {
                  String[] lstUlArr=line.toLowerCase().split(",");
                  if (lstUlArr[0].contains(ulArr[0].toLowerCase())) {
                        brz = lstUlArr[1];
                  }
            }
            br.close();
            return brz;
      }
```

Ranije verzije aplikacije pohranjivale su podatke na vanjsku memoriju (SD karticu). Ovo je bila testna verzija aplikacije i ovakva pohrana nije prikladna za distribuciju podataka stvarnog. Ipak, niže je priložen kod ovakve varijante rada radi usporedbe i prikaza rada sličnog sustava. Ova metoda koristi tri datoteke koje sadrže samo popis ulica odvojen znakom za prekid linije (engl. Line Break) \n, a telefonski broj zone naplate je sadržan u imenu datoteke.

```
      File sdcard = Environment.getExternalStorageDirectory();

      List<String> fajloviZone = Arrays.asList("700101.txt", "700102.txt",
"700103.txt");

      for (String fajl : fajloviZone) {
            File file = new File(sdcard, fajl);

            try {
            BufferedReader br = new BufferedReader(new FileReader(file));
            String line;

            while ((line = br.readLine()) != null) {
                  if (line.toLowerCase().contains(ulArr[0].toLowerCase())) {
                        brz = fajl.substring(0, 6);
                  }
            }
            br.close();
            } catch (IOException e) {
            }
      }
```

U obje varijante rezultat koda je prepoznata ulica i prepoznati telefonski broj za naplatu parkinga (ili nedostatak istih u slučaju da se korisnik ne nalazi u zoni naplate parkinga.

[41] Usp Android developer String: https://developer.android.com/reference/java/lang/String.html

Ranije poglavlje predlaže pohranu podataka o tablici automobila na telefon. Ovo je isto ostvarivo upotrebom datoteka u /raw mapi. Niže je navedena moguća implementacija takvog rješenja. Ako u stringu „tablica“ nosimo podatke o registarskom broju vozila.

```
FileWriter fw;

try {
    fw = new FileWriter((R.raw.tablica), false);
    fw.write(Tablica);
    fw.close();
    fw.flush();
}
catch (IOException e) {
    e.printStackTrace();
}
```

5.6.3 Assets

Assets folder sličan je raw folderu sa nekoliko važnih iznimki. Prvo, datoteke u assets mapi ne dobivaju R ID i drugo, datoteke u assets mapi ne mogu biti veće od 1 megabajta. Datoteke iz assets mape se teže referenciraju iz koda i sporije učitavaju. Ipak, podaci se često pohranjuju na ovo mjesto, naročito datoteke koje se ne čitaju izravno, na primjer datoteke sa bazama podataka(SQL lite baze podataka). Aplikacija SmsParking ne koristi lokalnu bazu podataka te ne koristi ovu mapu.[42]

5.6.4 SQLite

SQLite je softverska knjižnica koja implementira samostalni, bezserverski i transakcijski mehanizam baze podataka koji ne zahtjeva nikakvu konfiguraciju. SQLite je jedna od najkorištenijih bazi podataka na svijetu, i izvorni kod je otvoren i javno dostupan. Radi se o integriranoj bazi podataka koja nema odvojen serverski proces, već čita i piše direktno u običnu datoteku na disku. Njen format se slobodno može koristiti među platformama i raznim arhitekturama. To ju čini jednim od najpopularnijih izbora za aplikacijski datotečni format.

Za razliku od drugih bazi podataka, SQLite nije zasnovan na klijent-server arhitekturi, već se nalazi integriran u krajnji korisnički program, to jest sadržan je u jednoj datoteci koja se u slučaju mobilnih aplikacija može distribuirati zajedno s aplikacijom. Jedan je od najpopularnijih mehanizama za korištenje u malim programima i aplikacijama upravo zbog svoje malene veličine, kao i zbog jednostavnosti korištenja, no razlikuje se od Oracle, Microsoft ili MySQL inačica SQL bazi podataka između ostalog i po svojim tipovima podataka.

Za razliku od standardnog SQLa koji ne dopušta unošenje drugačijih tipova podataka od onog prvotno deklariranog u redovima ili stupcima, u slučaju da se u SQLite unese drugi tip podataka (npr. broj u tekst ili obrnuto), SQLite uredno te podatke sprema i kao odgovarajući format. SQLite polja se ravnaju prema

---

[42] Usp. Android: Loading files from the Assets and Raw folders
http://www.41post.com/3985/programming/android-loading-files-from-the-assets-and-raw-folders

„afinitetu“ prema određenim tipovima podataka. Što se tiče operatora, SQLite koristi sve standardne matematičke operatore, i konvertira sve operatore u numeričku klasu prije izvršavanja. U slučaju da je operator NULL, kao rezultat uvijek daje NULL, a ako nije numerički i nije NULL, automatski se konvertira u 0 ili 0.0. Automatsko sortiranje (ORDER BY klauzula SQL izraza) daje prednost ćelijama s NULL vrijednostima, nakon toga INTEGER, pa REAL i na kraju TEXT vrijednostima. Grupni SELECT operatori kao što su UNION, INTERSECT i EXCEPT izvršavaju implicitne usporedbe i izvršavaju se kako jesu. Dodatna zanimljivost je također i BLOB tip podataka, koji se sprema upravo kako je i unesen. Povijesno se BLOB zvao NONE, no zbog jasnoće koda je preimenovan, i vrijednosti unesene kao BLOB se nikad ne konvertiraju.

Komunikacija aplikacije sa SQLite poslužitelj uobičajeno se odvija posredstvom ContentPovider klase[43].

5.6.5 Vanjski poslužitelji

Naravno za pohranu podataka moguće je koristi i vanjske izvore. Kao i kod Web i desktop aplikacija ovo se najčešće ostvaruje upotrebom server-klijent arhitekture i bazi podataka. Za Android aplikacije preporučuje se upotreba mySQL poslužitelja[44]. Povezivanje s njima najlakše je ostvariti upotrebom GET i POST metoda. Upotreba GET metoda na Androidu to može izgledati ovako:

```
URL url = new URL(link);
HttpClient client = new DefaultHttpClient();
HttpGet request = new HttpGet();
request.setURI(new URI(link));
```

Deklarira se novi URL objekt koji sadrži adresu mySQL poslužitelja, HTTP klijent i HTTP zahtjev (engl. request).

```
HttpResponse response = client.execute(request);
```

Execute metoda ranije deklariranog HHTP klijenta šalje zahtjev poslužitelju.

```
BufferedReader in = new BufferedReader(new
InputStreamReader(response.getEntity().getContent()));
```

Nakon izvršenja zahtjeva još treba samo uhvatiti odgovor. Za ovo može poslužiti i ranije spomenuti BufferedReader.

```
HttpResponse response = client.execute(request);
BufferedReader in = new BufferedReader
(new InputStreamReader(response.getEntity().getContent()));
```

Gornji primjer podrazumijeva da na poslužitelju uz mySQl server postoje i PHP skripte koje će odraditi samu komunikaciju sa bazom podatka. Da bi ovakvu komunikaciju realizirali treba nam još samo string

---

[43] Usp. Android ddevelopers ConterProvider
https://developer.android.com/reference/android/content/ContentProvider.html
[44] Usp. Tutorials point Android - PHP/MYSQL Tutorial
http://www.tutorialspoint.com/android/android_php_mysql.htm

koji stavljamo u GET metodu (u gornjem primjeru on se nalazi u varijabli link) i mogao bi glasiti (u slučaju autentikacije na bazu podataka):

```
      String link =
      "http://mojphpmojsqlweb.mojserver.me/login.php?username="+  username+"&
password="+password;
```

Autor rada smatra da je upravo ovakvo rješenje preferirano za ovaj sustav ali njegova realizacija u aplikaciji zahtjeva dodatne resurse (mySql poslužitelj) koji nisu realizirani prije pisanja ovog rada. Implementacija mySQL poslužitelja koji bi opskrbljivao SmsParking aplikaciju podacima o ulicama i zonama naplate te web aplikacije koja bi služila za administraciju ove baze podataka dovela bi aplikaciju na razinu kvalitete, i uz redovito ažuriranje podataka, i pouzdanosti koji bi omogućili komercijalizaciju.

6. Zaključak

Kao što je i postavljeno u uvodu korištenjem podataka koje pruža slobodno dostupni GPS sustava, i njihovom kombinacijom s postojećim bazama podataka o lokacijama tornjeva pružatelja GSM usluga te poznatim lokacijama WiFi mreža te podataka o imenima ulica, moguće je s dovoljno preciznosti i u realnom vremenu odrediti geografski položaj određenog mobilnog uređaja. Dodatkom i implementacijom tablice referentnih telefonskih brojeva, moguće je ostvariti jednostavan i učinkovit način plaćanja parkinga SMSom.

Android Studio se pokazao kao jednostavan i funkcionalan alat za razvoj android aplikacija koji upotrebom javno dostupnih Android klasa poput Location Managera, Geocodera, Google Maps APIja i SMSManagera, omogućuje razvoj brzi razvoj aplikacija koje prepoznaju korisnikovu geografsku lokaciju i daljnjom obradom geografskih i drugih podataka izvršavaju zadatke koji su pred njih postavljeni.

Aplikacije poput SmsParkinga posebno bi mogle koristiti osobama koje često putuju u sklopu posla, ili starijim osobama kojima je nezgodno svaki put ručno unositi broj registarskih tablica i tražiti točan telefonski broj.

Tijekom testiranja aplikacije autor je uočio da je GPS pružatelj usluga značajno pouzdaniji izvor podataka o geografskom položaju od Network pružatelja usluga koji je često radio greške od 100m ili više. Ipak, GPS pružatelju usluga je potrebno više vremena da uspostavi kontakt sa satelitima, značajno povećava potrošnju baterije mobilnog uređaja i često nije dostupan u zatvorenim prostorima. Možemo zaključiti da oba sustava imaju prednosti i mane u odnosu na drugi ali autor ovog rada ipak preferira korištenje GPS providera. Ovo je uostalom i očekivani zaključak, GPS je ipak građen sa ciljem geolokacije njegovih korisnika, a kod Network providera utvrđivanje geografskog položaja se postiže upotrebom tehnologija koje su namijenjene prvenstveno komunikaciji (GSM tornjevi i WiFi mreže).

7. Literatura

[1] Adroid App Respurces. URL: https://developer.android.com/guide/topics/resources/providing-resources.html (10.6.2016)

[2] Android developer Address. URL: https://developer.android.com/reference/android/location/Address.html (10.6.2016)

[3] Android developer android.location paket. URL: https://developer.android.com/reference/android/location/package-summary.html (10.6.2016)

[4] Android developer ConterProvider. URL: https://developer.android.com/reference/android/content/ContentProvider.html (10.6.2016)

[5] Android developer documentation Log. URL: https://developer.android.com/reference/android/util/Log.html (10.6.2016)

[6] Android developer Geocoder. URL: https://developer.android.com/reference/android/location/Geocoder.html (10.6.2016)

[7] Android developer Intent. URL: https://developer.android.com/reference/android/content/Intent.html (10.6.2016)

[8] Android developer Location strategies. URL: https://developer.android.com/guide/topics/location/strategies.html (10.6.2016)

[9] Android developer LocationListener. URL: https://developer.android.com/reference/android/location/LocationListener.html (10.6.2016)

[10] Android developer LocationManager. URL: https://developer.android.com/reference/android/location/LocationManager.html (10.6.2016)

[11] Android developer Providing resources. URL: https://developer.android.com/guide/topics/resources/providing-resources.html (10.6.2016)

[12] Android developer SMS manager. URL: https://developer.android.com/reference/android/telephony/SmsManager.html (10.6.2016)

[13] Android Location API – Tutorial. URL: http://www.vogella.com/tutorials/AndroidLocationAPI/article.html (10.6.2016)

[14] Android Location Providers – gps, network, passive. URL: http://developerlife.com/tutorials/?p=1375 (10.6.2016)

[15] Android Platform Development Kit. URL: http://www.netmite.com/android/mydroid/development/pdk/docs/ (10.6.2016)

[16] Android: Loading files from the Assets and Raw folders. URL: http://www.41post.com/3985/programming/android-loading-files-from-the-assets-and-raw-folders (10.6.2016)

[17] AndroidHive Android GPS, Location Manager Tutorial. URL: http://www.androidhive.info/2012/07/android-gps-location-manager-tutorial/ (10.6.2016)

[18] Bao-Yen Tsui, J. CoFundamentals of Global Positioning System Receivers: A Software Approach. John Wiley & Sons, Inc., 2000. URL: http://kakukoto.free.fr/F/019/Fundamentals%20of%20Global%20Positioning%20System%20Receivers/Fundamentals%20of%20Global%20Positioning%20System%20Receivers.pdf (10.6.2016)

[19] Burch, W. Understanding Android: Three Ways to Find Your Location. 2012. URL: http://www.idyllramblings.com/2012/03/understanding-android-three-ways-to-find-your-location.html (10.6.2016)
[20] Čović, Z.; Babić D. Development and Implementation of Location Based Native Mobile Application. International journal of electrical and computer engineering systems, Vol.5. No.1., 2014. URL: http://hrcak.srce.hr/index.php?show=clanak&id_clanak_jezik=198376 (10.6.2016)
[21] Ehringer, D. The dalvik virtual machine architecture. 2010. URL: http://davidehringer.com/software/android/The_Dalvik_Virtual_Machine.pdf (10.6.2016)
[22] Elgin, B. Google Buys Android for Its Mobile Arsenal. Bloomberg, 2005. URL: http://www.webcitation.org/5wk7sIvVb (10.6.2016)
[23] Fasel, M. A Good Look at Android Location Data. 2011. URL: https://blog.shinetech.com/2011/10/14/a-good-look-at-android-location-data/ (10.6.2016)
[24] Geer, D. Eclipse Becomes the Dominant Java IDE. Industry trends, 2005. URL: https://www.computer.org/csdl/mags/co/2005/07/r7016.pdf (10.6.2016)
[25] GLONASS Xperia. URL: http://developer.sonymobile.com/knowledge-base/technologies/glonass/ (10.6.2016)
[26] Google Maps APIs. URL: https://developers.google.com/maps/documentation/android-api/ (10.6.2016)
[27] Google maps Geolocation. URL: https://developers.google.com/maps/articles/geolocation?hl=nl (10.6.2016)
[28] Grønli,T.M.;Hansen, J.; Ghinea, G. Android vs Windows Mobile vs Java ME A comparative Study of Mobile Development Environments. URL: ftp://ftp.inf.ufrgs.br/pub/geyer/POD/artigos/TL2-2011-1/POD-2011-1-TL2-KA-Android-WinMob-JME.pdf (10.6.2016)
[29] Imielinski Julio, T.; Navas, C. GPS-Based Geographic Addressing, Routing, and Resource Discovery. Communications of the acm, Vol. 42, No. 4, 1999. URL: http://128.8.127.3/class/fall2009/cmsc828r/PAPERS/Gps_basedp86-imielinski[1].pdf (10.6.2016)
[30] Introduction to XML. URL: http://www.xmlfiles.com/xml/ (10.6.2016)
[31] Kumar,S.;Abdul Qadeer, M.;Gupta, A. Location based services using android (LBSOID). Internet Multimedia Services Architecture and Applications (IMSAA), 2009. URL: https://www.researchgate.net/publication/224127685_Location_based_services_using_android_LBSOID (10.6.2016)
[32] Lewallen, R. 4 major principles of Object-Oriented Programming. 2005. URL: http://codebetter.com/raymondlewallen/2005/07/19/4-major-principles-of-object-oriented-programming/ (10.6.2016)
[33] Locke, P. Cell Tower Triangulation – How it Works. 2012. URL: https://wrongfulconvictionsblog.org/2012/06/01/cell-tower-triangulation-how-it-works/ (10.6.2016)
[34] Making Your App Location-Aware. URL: https://developer.android.com/training/location/index.html (10.6.2016)
[35] Murphy, M. L. The Busy Coder's Guide to Android Development. CommonsWare, LLC, 2008. URL: http://libros.metabiblioteca.org/jspui/bitstream/001/414/8/978-0-9816780-0-9.pdf (10.6.2016)

[36] OASIS CIQ v2.0 committee specifications/standards. URL: https://www.oasis-open.org/committees/ciq/ciq.html (10.6.2016)
[37] Official U.S. Government information about the Global Positioning System (GPS) and related topics. URL: http://www.gps.gov/systems/gps/space/ (10.6.2016)
[38] Oracle Java Documentation Object Oriented Programming Concepts. URL: https://docs.oracle.com/javase/tutorial/java/concepts/ (10.6.2016)
[39] Pendleton, G. The Fundamentals of GPS. Directions Magazine, 2002.URL: http://www.directionsmag.com/entry/the-fundamentals-of-gps/124028 (10.6.2016)
[40] Radhika Rani, Ch.; Praveen Kumar, A.; Adarsh, D.; Krishna Mohan, K.; Kiran, K.V. Location based services in android. International Journal of Advances in Engineering & Technology, 2012. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.664.9581&rep=rep1&type=pdf (10.6.2016)
[41] Ratabouil S. Android NDK Beginner's Guide Second Edition. Packt Publishing, 2015. URL: http://file.allitebooks.com/20150821/Android%20NDK-%20Beginner's%20Guide%20-%20Second%20Edition.pdf (10.6.2016)
[42] Receiving Location Updates. URL: https://developer.android.com/training/location/receive-location-updates.html (10.6.2016)
[43] Rex, J. Android Studio vs. Eclipse: What You Need To Know. URL: https://www.airpair.com/android/android-studio-vs-eclipse (10.6.2016)
[44] Serder, A. Android: GPS positioning and location strategies. 2014. URL: https://blog.codecentric.de/en/2014/05/android-gps-positioning-location-strategies/ (10.6.2016)
[45] Shu, X.;Du, Z;Chen, R. Research on Mobile Location Service Design Based on Android. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.470.9223&rep=rep1&type=pdf (10.6.2016)
[46] Singhal, M.; Shukla, A. Implementation of Location based Services in Android using GPS and Web Services. IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 1, No 2, 2012. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.403.2450&rep=rep1&type=pdf (10.6.2016)
[47] Tutorials point Android - PHP/MYSQL Tutorial. URL: http://www.tutorialspoint.com/android/android_php_mysql.htm (10.6.2016)
[48] Vogel/a Android Intents Tutorial. URL: http://www.vogella.com/tutorials/AndroidIntent/article.html (10.6.2016)

8. Popis slika

9. Popis skraćenica

[1] ADT – Android Development Kit
[2] API - Application programming interface
[3] app - application
[4] ARM - Acorn RISC Machine, Advanced RISC Machine
[5] EDGE - Enhanced Data rates for GSM Evolution
[6] GLONASS - Global Navigation Satellite System
[7] GNSS - Global Navigation Satellite System
[8] GPRS - General Packet Radio Service
[9] GPS – Global postioning system
[10] GSM - Global System for Mobile Communications
[11] IDE – Integrated development enviroment
[12] LTE (4G LTE) - Long-Term Evolution
[13] RISC - reduced instruction set computing
[14] SaS – Software as a Service
[15] SMS – Short message system
[16] SQL - Structured Query Language
[17] UMTS - *Universal Mobile Telecommunications System*
[18] USB – Universal system bus
[19] xAL - extensible Address Language
[20] XML – extensible markup language