MATURSKI RAD

Kreiranje Android aplikacije

Autor:
Lazar S. Stojković

Mentor:
Marko S. Milošević

Gimnazija "Svetozar Marković"
Niš, 08.04.2013.

1 Uvod

Android aplikacije se pišu u programskom jeziku Java. Android SDK[1] alatke kompajliraju kôd (zajedno sa svim resursima i podacima koje aplikacija koristi) i formiraju *Android paket*, fajl sa ekstenzijom `.apk`. Svi podaci u jednom `.apk` fajlu predstavljaju jednu aplikaciju koju uređaji sa Android OS-om mogu instalirati.
Nakon što je instalirana, svaka aplikacija postoji nezavisno od ostalih.

- Android je višekorisnički operativni sistem u kojem je svaka aplikacija zaseban korisnik.
- Sistem (podrazumevano) svakoj aplikaciji dodeljuje jedinstveni korisnički Linux ID (njega upotrebljava samo sistem i aplikaciji je nepoznat). Samo korisnik sa odgovarajućim ID-om ima pristup svim fajlovima jedne aplikacije.
- Svaki proces ima sopstvenu virtualnu mašinu (VM) kako bi se kôd svake aplikacije mogao izvršavati izolovano od kodova ostalih aplikacija.
- Svaka aplikacija funkcioniše u sopstvenom Linux procesu. Android pokreće proces kada bilo koja komponenta aplikacije treba da bude izvršena i zaustavlja ga kada više ne bude bio potreban ili kada treba osloboditi memoriju za neku drugu aplikaciju.

Android sistem radi po *principu neophodne privilegije*, tj. svaka aplikacija ima pristup **samo** onim komponentama sistema koje su neophodne za njeno trenutno funkcionisanje. Ovo stvara veoma sigurno okruženje u kojem aplikacije imaju ograničen pristup sistemu.

- Moguće je podesiti da dve aplikacije imaju isti korisnički Linux ID i pritom će jedna aplikacija imati pristup i fajlovima druge aplikacije. Radi očuvanja sistemskih resursa, aplikacije sa istim ID-om se izvršavaju u istim Linux procesima i dele istu VM.
- Aplikacija takođe može imati pristup ostalim podacima i komponentama uređaja, kao što su kontakti, SMS poruke, SD kartica, kamera, Bluetooth i sl. Odobrenja ovakvih pristupa se obavezno traže od korisnika uređaja prilikom instalacije.

[1] Software Development Kit

2 Komponente aplikacije

Komponente aplikacije su neophodni blokovi od kojih je izgrađena svaka aplikacija. Svaka komponenta predstavlja različito mesto komunikacije sistema sa aplikacijom i ima svoju ulogu u definisanju ponašanja aplikacije.

Postoje četiri različite vrste komponenti. Svaka od njih ima različitu ulogu i različiti životni ciklus koji određuje kako se komponente stvaraju, žive i uništavaju.

2.1 Aktivnosti (Activities)

Aktivnost je komponenta aplikacije koja korisniku pruža interaktivni interfejs (npr. za kreiranje i slanje email-ova, za pregled i izmenu slika, za navigaciju itd). Svakoj aktivnosti je dat jedan prozor u kojem ona crta korisnički interfejs. Prozor se najčešće nalazi preko celog ekrana a njegove dimenzije se mogu i definisati.

Aplikacija se najčešće sastoji od više međusobno povezanih aktivnosti. Često se jedna od njih definiše kao glavna (*main*), i to je ona aktivnost koja se prva prikazuje nakon pokretanja aplikacije.

Svaka aktivnost može, po potrebi, pokrenuti (pozvati) drugu. Svaki put kada se neka aktivnost pokrene na ovaj način, aktivnost koja ju je pozvala ne biva uništena već se stopira i čuva u *back stack*-u. Nova aktivnost se zatim stavlja na isti stek i nalazi na njegovom vrhu, time preuzimajući fokus. Back stack radi po LIFO mehanizmu, što znači da kada korisnik pritisne *back* dugme, aktivnost sa vrha stack-a se uništava i fokus preuzima ona koja se nalazi ispod.

2.1.1 Kreiranje aktivnosti

Kako bi se kreirala aktivnost neophodno je kreirati Java klasu sa ekstenzijom `Activity`, deklarisati je u *Manifest fajlu* i eventualno pokrenuti (o deklaraciji i pokretanju aktivnosti će biti reči u skecijama 4 i 3).

```
public class MainMenuActivity extends Activity {
    /* . . . */
}
```

Slika 1: Ekstenzija Java klase

Klasa mora sadržati određene *Callback metode*, tj. funkcije koje će sistem pozivati po potrebi. Te funkcije direktno određuju ponašanje aktivnosti u različitim stadijumima njenog životnog ciklusa (kao sto su njena kreacija, stopiranje, uništenje itd.)

Životni ciklus aktivnosti

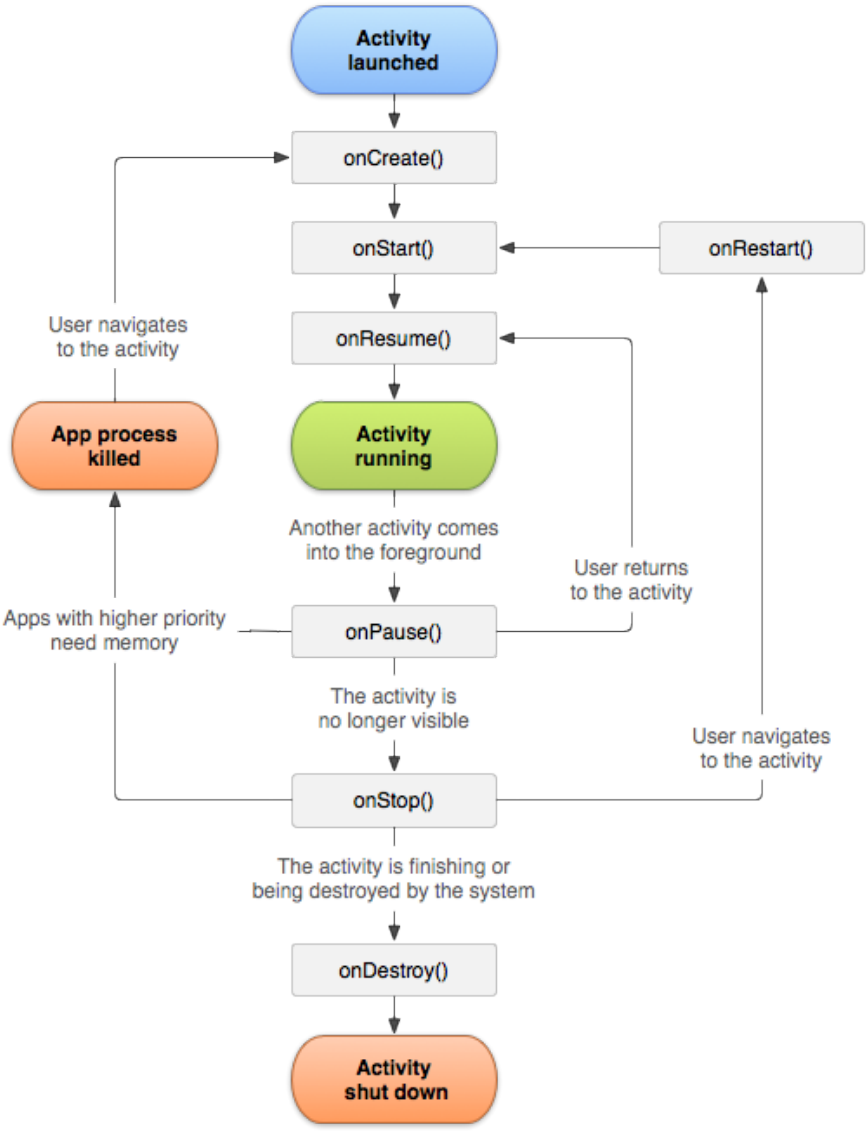
Životni ciklus aktivnosti je određen implementiranim Callback metodama i njegova optimizacija je krucijalna za kvalitetan i fleksibilan rad aplikacije.

Aktivnost može postojati u tri različita stanja:

nastavljeno (*resumed, running*): aktivnost je prikazana na ekranu i ima sav fokus;

pauzirano (*paused*): aktivnost je nevidljiva, transparentna ili zauzima samo deo ekrana, dok je fokus na drugoj aktivnosti. Pauzirana aktivnost je u potpunosti 'živa' i radi u pozadini. U svakom trenutku se fokus može prebaciti na nju ili ona uništiti (preko *window manager*-a ili direktno);

zaustavljano (*stopped*): aktivnost je u pozadini i potpuno nevidljiva korisniku (ne vidi se ni u *window manager*-u). Ona je i dalje živa ali je sistem može automatski uništiti usled nedostatka memorije.



Slika 2: Životni ciklus aktivnosti

Implementacija Callback metoda

Metoda	Opis
`onCreate()`	Poziva se uvek kada se aktivnost kreira. U njoj treba definisati izgled i generalno ponašanje aktivnosti (pozadinu, *view*-ove, uključiti muziku i sl.) Može joj se proslediti *Bundle* objekat koji sadrži informacije o prethodnom stanju aktivnosti (za slučaj da treba biti nastavljena). Nakon ove metode se uvek poziva `onStart()`.
`onRestart()`	Poziva se nakon što je aktivnost stopirana i pokreće je ponovo. Nakon nje se poziva `onStart()`.
`onStart()`	Poziva se u trenutku pre no što aktivnost postane vidljiva korisniku. Za njom se može pozivati `onResume()` ukoliko se aktivnost stavlja u prvi plan, ili `onStop()` ukoliko se ona krije, zaustavlja.
`onResume()`	Poziva se u trenutku pre no što aktivnost postane interaktivna, spremna da registruje korisnikov unos, procesira ga i reaguje na njega. Aktivnost se nalazi na vrhu *back stack*-a i ima sav fokus. Nakon nje se poziva `onPause()`.
`onPause()`	Poziva se kada sistem treba da pokrene neku drugu aktivnost. Kôd ove metode najčešće služi za pamćenje podataka o pauziranoj aktivnosti, zaustavljanje animacija, zvukova i ostalih stvari koje mogu smetati ili zauzimati resurse. Mora biti brza jer sistem čeka njeno okončanje kako bi pozvao metode za pokretanje/nastavljanje druge aktivnosti. Ovoj metodi može da sledi `onResume()` kako bi se aktivnost vratila u prvi plan ili `onStop()` kako bi se ona zaustavila i postala nevidljiva.
`onStop()`	Poziva se kada aktivnost treba biti sakrivena, bilo u procesu njenog uništenja ili kako bi neka druga aktivnost dobila sav fokus. Nakon nje može da se pozove `onRestart()` kako bi se aktivnost iznova pokrenula, ili `onDestroy()` kako bi se ona uništila.
`onDestroy()`	Poziva se pre no što je aktivnost uništena. Poziv ove metode je poslednji poziv u životu jedne aktivnosti. Može joj se pristupiti tako što će se negde u kodu pozovati `finish()` ili je sistem može automatski pozvati usled nedostatka memorije (resursa). Način na koji joj se pristupa se može odrediti pomoću `isFinishing()` metode.

Tabela 1: Callback metode aktivnosti i njihov opis

Sve ove metode zajedno definišu ponašanje aktivnosti tokom njenog životnog ciklusa. Svakoj mora prethoditi `@Override` anotacija koja je zaslužna za sankcionisanje mogućih grešaka prilikom pozivanja metoda. Ove metode su veoma bitne za pravilno funkcionisanje aktivnosti (pa i aplikacije) i ne mogu se priuštiti moguće greške kod njihovog pozivanja ili samog koda. Implementacijom ovih metoda mogu se uočiti tri ugnježđena stadijuma životnog ciklusa:

- **celokupan život** aktivnosti se odvija između poziva metoda `onCreate()` i `onDestroy()`. Sve aktivnosti bi trebalo programirati po 'opštoj' logici - u `onCreate()` definisati *layout* (izgled i raspored view-ova), započeti željene procese (muziku, animacije, snimanje kamerom, download-ovanje i sl.) a u `onDestroy()` iste prekinuti i osloboditi sve resurse.

- deo života u kojem je aktivnost **vidljiva** odvija se između poziva metoda `onStart()` i `onStop()`. Korisnik tada može videti aktivnost i interagovati sa njom. Na primer, u `onStart()` se može registrovati `BroadcastReciever` koji će beležiti sve promene na interfejsu, a u `onStop()` zaustaviti jer aktivnost onda postaje nevidljiva. Metode `onStart()` i `onStop()` mogu se pozivati više puta za vreme celokupnog života aktivnosti.

- deo života u kojem je aktivnost u **prvom planu** se odvija između poziva `onResume()` i `onPause()` metoda. Tada je aktivnost primarna na ekranu i ima sav fokus. Aktivnost može više puta prelaziti iz pauziranog u nastavljeno stanje i obratno. Metoda `onPause()` se automatski poziva kada se prikaže dijalog, uređaj uspava, dobije poziv i sl, ili manualno, kroz kôd, što je slučaj kod pauziranja igrice. Iz razloga što se ove dve metode pozivaju vrlo često, njihov kôd bi trebao biti lak i brz, kako bi se izbeglo preopterećenje sistema.

Ispod se nalaze primeri Callback metoda iz kôda aplikacije (igrice) Lost Illusions, tačnije, njene Java klase `MainMenuActivity.java` u kojoj je definisana prva aktivnost koja se poziva sa pokretanjem aplikacije (početni meni igrice).

```
@Override
protected void onPause() {
    super.onPause();
    mPaused = true;
    mp.pause();
}

@Override
protected void onDestroy() {
    super.onDestroy();
    mp.stop();
}
```

```java
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.mainmenu);
    mPaused = true;

    mBackground = findViewById(R.id.mainMenuBackground);

    mTitle = findViewById(R.id.mainMenuTitle);
    mTitle.setOnClickListener(sStartButtonListener);

    mButtonFlickerAnimation = AnimationUtils.loadAnimation(this,
            R.anim.button_flicker);
    mFadeOutAnimation = AnimationUtils.loadAnimation(this, R.anim.fade_out);
    mAlternateFadeOutAnimation = AnimationUtils.loadAnimation(this,
            R.anim.fade_out);
    mFadeInAnimation = AnimationUtils.loadAnimation(this, R.anim.fade_in);
    mBackgroundAnimSlide = AnimationUtils.loadAnimation(this,
            R.anim.title_screen_slide);

    mTitle.startAnimation(mBackgroundAnimSlide);

    Toast.makeText(this, "Click to Play!", Toast.LENGTH_LONG).show();
    /* . . . */
}
```

```java
@Override
protected void onResume() {
    super.onResume();
    mPaused = false;
    mp.start();
    mButtonFlickerAnimation.setAnimationListener(null);
    if (mTitle != null) {

        // Change "start" to "continue" if there's a saved game.
        SharedPreferences prefs = getSharedPreferences(
                PreferenceConstants.PREFERENCE_NAME, MODE_PRIVATE);
        final int row = prefs.getInt(
                PreferenceConstants.PREFERENCE_LEVEL_ROW, 0);
        final int index = prefs.getInt(
                PreferenceConstants.PREFERENCE_LEVEL_INDEX, 0);
        if (row != 0 || index != 0) {
            mTitle.setOnClickListener(sContinueButtonListener);
        } else {
            mTitle.setOnClickListener(sStartButtonListener);
        }
        /* . . . */
    }
```

Implementacija korsničkog interfejsa

Korisnički interfejs aktivnosti je sačinjen od hijerarhije *view*-ova, interaktivnih objekata definisanih u `View` klasi. Svaki view zauzima i kontroliše određeni pravougaoni deo prozora koji ispunjava njegova aktivnost. Na primer, view može biti dugme čijim se klikom pokreće muzika.

Android pruža na desetine već definisanih view-ova koji se mogu koristiti za dizajniranje i optimizaciju korisničkog interfejsa. View-ovi koji čine vizualne i interaktivne elemente na ekranu takođe se nazivaju *widgets*. To su dugmići, radio dugmići, tekst polja, slike itd. View-ovi koji određuju formu i generalni raspored widget-a na korisničkom interfejsu nazivaju se *layouts*[2]. Nekoliko njih je već definisano u `ViewGroup` klasi i to su *linear*, *absolute*, *frame*, *grid* i *relative* layout. Moguće je definisati sopstvene view-ove u `View` i `ViewGroup` klasama.

Korisnički interfejs se najlakše dizajnira kroz XML layout fajlove koji se dodaju kao resursi i kompajliraju uz aplikaciju. Na ovaj način se odvaja kôd koji definiše

[2] Layout je takođe i naziv za celokupni izgled jedne aktivnosti.

izgled aktivnosti od koda koji definiše njeno ponašanje. Layout se lako povezuje sa aktivnošću pomoću `setContentView()`, koja kao argument ima resursni ID layout-a. U već datom primeru `onCreate()` metode može se videti povezivanje aktivnosti sa layout-om.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" >

    <AbsoluteLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" >

        <ImageView
            android:id="@+id/mainMenuTitle"
            android:layout_width="510dp"
            android:layout_height="340dp"
            android:layout_x="0dp"
            android:layout_y="-61dp"
            android:adjustViewBounds="true"
            android:src="@drawable/title_background" />

        <ImageView
            android:id="@+id/Title"
            android:layout_width="538dp"
            android:layout_height="339dp"
          </ImageView>
    ........
```

Slika 3: Deo koda XML layout fajla koji je povezan sa `MainMenuActivity.java`

2.2 Servisi (Services)

Servisi su komponente aplikacije koje izvršavaju dugotrajne operacije u pozadini i najčešće ne pružaju korisnički interfejs. Druga komponenta aplikacije može zatražiti stvaranje servisa koji će nastaviti sa radom čak iako korisnik pređe na drugu aplikaciju. Moguće je povezati neku komponentu aplikacije sa servisom i tako omogućiti njihovu komunikaciju (*interprocess comunication, IPC*). Na primer, servis u pozadini može download/upload-ovati fajlove, puštati muziku, komunicirati sa dobavljačem sadržaja i sl.

Servisi se dele na:

- pokrenute (*started*): pozvane od strane neke komponente aplikacije pomoću `startService()`. Jednom kada je pokrenut, servis može raditi neograničeno dugo, čak i nakon što je komponenta uništena. Pokrenuti servisi najčešće ne vraćaju neki rezultat komponenti koja ga poziva, već samo izvrše zahtevani zadatak i zaustave se. Na primer, servisi za download i upload su ovog tipa.

- vezane (*bound*): pozvane od strane neke komponente aplikacije pomoću metode `bindService()` i vezane za nju. Vezani servisi kroz IPC omogućavaju klijent-server interfejs, slanje zahteva, vraćanje rezultata, i sl. Ovakvi servisi rade sve dok su vezani za neku komponentu i moguće je vezati ih za više komponenti istovremeno. Kada se u potpunosti odvežu oni bivaju uništeni.

Kreirani servis radi u procesu komponente koja ga je pozvala, dakle on ne započinje novi proces ili novi *thread* ukoliko drugačije nije navedeno. U slučaju da servis radi veliki posao ili više njih istovremeno, onda se preporučuje stvaranje novih thread-ova unutar njega. Time se smanjuje rizik od pucanja aplikacije.

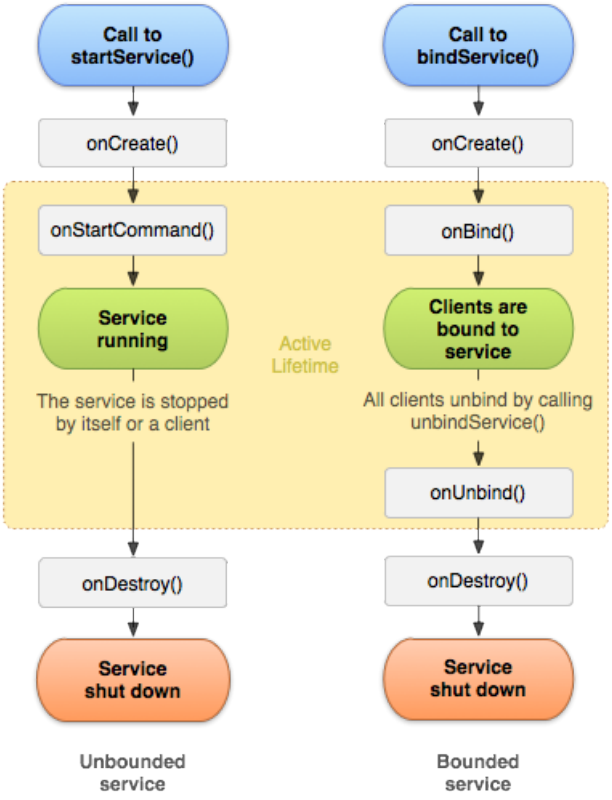
2.2.1 Kreiranje servisa

Za kreiranje servisa, neophodno je stvoriti klasu sa `Service` proširenjem i deklarisati je u Manifest fajlu. U klasi treba implementirati Callback metode i tako definisati serivs.

Metoda	**Opis**
`onStartCommand()`	Sistem poziva ovu metodu kada neka od komponenti aplikacije zatraži kreaciju servisa pomoću metode `startService()`. Nakon pozivanja ove metode kreira se *pokrenuti* servis i radi u pozadini nedefinisano dugo. Takođe bi trebalo narediti servisu njegovo uništenje kada svoj posao odradi pomoću metode `stopSelf()` ili `stopService()` koja se poziva iz komponente. Implementacija ove metode je nepotrebna prilikom kreacije vezanih servisa.
`onBind()`	Sistem poziva ovu metodu kada neka komponenta zahteva kreaciju vezanog servisa pomoću metode `bindService()`. Implementacija ove metode treba pružiti klijentima (komponentama koje je pozivaju) interfejs preko koga će oni komunicirati sa servisom i to radi tako što metoda vraća vrednost `IBinder`. Ova metoda mora uvek biti implementirana a ukoliko nije potrebno stvaranje vezanog servisa ona treba vratiti vrednost `null`.
`onCreate()`	Ova metoda se poziva odmah nakon što komponenta zatraži kreaciju servisa (pre no što se `onStartCommand()` ili `onBind()` pozovu). U kodu ove metode se implementira sama svrha servisa. Ukoliko je servis već pokrenut i radi, ova metoda se ne poziva.
`onDestroy()`	Poziva se nakon što servis izvrši svoj zadatak (kroz `stopService()`), ili se odveže od svih komponenata (`unbindService()`), i uništava ga.

Tabela 2: Bitnije Callback metode servisa i njihov opis

Životni ciklus servisa



Slika 4: Životni ciklus servisa

Životni ciklus servisa je jednostavniji od životnog ciklusa aktivnosti. Kod servisa treba više obratiti pažnju na samu njihovu kreaciju i uništenje jer korisnik najčešće nije svestan njihovog funkcionisanja i nema kontrolu nad njima.

Implementacijom Callback metoda mogu se uočiti dva ugnježdena stadijuma životnog ciklusa servisa:

- **celokupan život** servisa nalazi se između poziva `onCreate()` i `onDestroy()` metoda. Podrazumevano, u `onCreate()` treba definisati servis i njengovo ponašanje a u `onDestroy()` osloboditi sve resurse koje je on zauzimao. Bez obzira na to da li je servis pokrenut ili vezan on mora biti kreiran i uništen.

- **aktivan život** servisa počinje pozivom `onStartCommand()` ili `onBind()` metode. Ovim metodama se, prilikom pozivanja servisa, prosleđuje Intent objekat (objašnjen u sekciji 3) kroz `startService()` ili `bindService()` respektivno. Aktivan život pokrenutog servisa se završava u istom trenutku kada i celokupan dok se kod vezanih servisa on završava pozivom `onUnbind()` metode.

Pokrenuti i vezani tip servisa nisu striktno razdvojeni, tj. servis pokrenut sa `startService()` može biti naknadno vezan. Na primer, pokrenut može biti servis koji pušta muziku u pozadini ali se on naknadno može vezati za aktivnost Music player-a ukoliko korisnik zaželi kontrolu nad njim.

Servisi mogu obaveštavati korisnika o svom radu na dva načina, preko *toast* poruka ili preko obaveštenja u statusnoj liniji i statusnom prozoru.

Toast poruke su kratke poruke u crnim pravougaonicima koji se pojavljuju na ekranu preko primarne aktivnosti i nestaju posle par sekundi.

Statusna linija se kod standardnog Android interfejsa (i aplikacija koje nisu full-screen) nalazi na samom vrhu ekrana i moguće je njeno proširenje na statusni prozor. U statusnoj liniji se često mogu videti obaveštenja o progresu servisa (npr. otpočeto preuzimanje sadržaja sa interneta, preuzimanje je u toku, preuzimanje je završeno itd.) U statusnom prozoru se može videti više informacija o progresu servisa i eventualno interagovati sa njihovim rezultatima (npr. pokretanje ili pregled preuzetog sadržaja, otvaranje primljenog email-a u većem prozoru itd.)

2.3 Dobavljači sadržaja (Content providers)

Dobavljači sadržaja omogućuju aplikaciji pristup deljivim podacima sa raznih lokacija. Ti podaci se mogu nalaziti u sistemu, na internetu (što je slučaj sa bazama podataka) ili bilo kojoj drugoj lokaciji kojoj aplikacija može pristupiti. Kroz dobavljače sadržaja aplikacija može podacima slati upite ili ih modifikovati, ukoliko joj dobavljač za to da dozvolu.

Na primer, Android sistem pruža dobavljača sadržaja koji uređuje korisnikovu listu kontakta. Tako se svaka aplikacija, uz određeno odobrenje, može poslužiti dobavljačem, ili nekim njegovim segmentom (kao što je `ContactsContract.Data`), kako bi učitala ili upisala informacije kod određene osobe u kontaktima.

Dobavljači sadržaja su takodje korisni kod rada sa podacima koji su privatni za jednu aplikaciju. Tako Note Pad koristi ovaj mehanizam kako bi sačuvao svoje beleške.

Dobavljači sadržaja se implementiraju u Java klasama sa `ContentProvider` proširenjem, u kojima treba implementirati standardni skup API[3]-ja koji će omogućiti aplikacijama da vrše transakcije.

2.4 Prijemnici emisija (Broadcast recievers)

Prijemnici emisija su komponente koje reaguju na emitovana obaveštenja i najave u sistemu. Mnoge emisije stižu od samog sistema, kao što je obaveštenje da je baterija prazna, da je ekran isključen, da je *print screen* komanda uspešno sačuvala sliku itd. Aplikacije takođe mogu emitovati obaveštenja, na primer kada uspešno skinu sadržaj sa interneta i žele da obaveste druge aplikacije da ga mogu koristiti.

Iako prijemnici emisija ne mogu pružiti korisnički interfejs, oni mogu stvoriti obaveštenje u statusnoj liniji i tako obavestiti korisnika. Prijemnici emisija najčešće samo signaliziraju drugim komponentama da počnu sa izvršavanjem određenih zadataka i sami po sebi ne rade velike poslove.

Prijemnici se implementiraju u Java klasama sa proširenjem `BroadcastReciever` i aktivira ih prosleđeni Intent objekat, slično kao i aktivnosti.

[3]Application Programming Interface

3 Aktivacija komponenti

Tri od četiri tipa komponenti - aktivnosti, servisi i prijemnici emisja - pokreću se pomoću asihrone poruke zvane Intent. Intent objekat spaja individualne komponente za vreme *runtime*-a[4] bez obzira na to da li komponente pripadaju istoj aplikaciji. Intent objekat se može zamisliti kao glasnik koji od komponente zahteva izvesnu akciju ili delovanje. (Slika 6)

Intent se stvara kao objekat u kom se definiše poruka koja se šalje komponenti. Eksplicitni Intent objekat može aktivirati određenu komponentu dok implicitni može aktivirati određeni *tip* komponenti.

Intent objekat prosleđen aktivnostima i servisima definiše akciju koju treba da izvršei (npr. da prikažu ili pošalju nešto) i takođe može sadrzati URI podataka nad kojima ona treba da bude izvršena (kao i ostale stvari koje bi novopokrenuta komponenta trebala da zna). Na primer, njime se od aktivnosti može zatražiti da prikaže određenu sliku ili otvori određenu web stranicu. U nekim slučajevima je moguće pokrenuti aktivnost kako bi ona vratila rezultat, takođe kroz Intent objekat (na primer, on može pitati korsinika da odabere neku osobu iz kontakta i vraćeni Intent objekat će sadržati URI tog kontakta).

Intent objekat prosleđen prijemniku emisija samo definiše poruku koja će biti prikazana kao obaveštenje.

Poslednji tip komponenti, dobavljači sadržaja, se ne aktivira Intent objektima, već zahtevima koje mu `ContentResolver` može uputiti. Content Resolver kontroliše sve direktne transakcije između komponente i dobavljača kako komponenta ne bi morala to da radi. Ona samo poziva metode iz `ContentResolver` objekta i tako se stvara apstraktni odnos komponenta-dobavljač radi sigurnosti.

Svaki tip komponenti se aktivira posebnim metodama:

- Aktivnost se pokreće (ili joj se zadaje novi zadatak) prosleđivanjem Intent objekta metodi `startActivity()`. Kada se od aktivnosti očekuje da vrati rezultat, on se prosleđuje metodi `startActivityForResult()`.
- Servis se pokreće (ili mu se daju nove instrukcije ukoliko je već pokrenut) prosleđivanjem Intent objekta metodi `startService()`. Servis se može vezati prosleđivanjem Intent objekta metodi `bindService()`.
- Emitovanje se može inicirati prosleđivanjem Intent objekta metodi `sendBroadcast()`, `sendOrderedBroadcast()` ili `sendStickyBroadcast()`.
- Dobavljaču sadržaja se može postaviti upit pozivanjem metode `query()` kod `ContentResolver`-a.

4 Manifest fajl

Pre nego što Android sistem pokrene komponentu on se mora uveriti da komponenta postoji, čitajući `AndroidManifest.xml` fajl. Aplikacija mora deklarisati sve svoje komonente u ovom fajlu.

Pored deklaracije komponenti u Manifest fajlu se deklarišu i:

[4]Vreme kada se kod zapravo izvršava i ne može biti menjan.

- dozvole koje će aplikacija tražiti od korisnika prilikom instalacije. Na primer, dozvola da aplikacija pristupi listi korisnikovih kontakta;
- minimalni API nivo, koji zavisi od toga koje API-e aplikacija koristi;
- hardver i softver koji aplikacija koristi ili zahteva, kao što su kamera, multitouch ekran ili bluetooth usluge;
- API biblioteke sa kojima aplikacija treba da se poveže (API-i koji nisu u standardnom Android okviru), kao što je Google Maps library;
- druge informacije.

4.1 Deklaracija komponenti

Primarini zadatak Manifest fajla je da informiše sistem o komponentama aplikacije.

U `<application>` elementu fajla se opisuje aplikacija. Tako `android:icon` predstavlja resurs koji je ikona aplikacije, `android:label` ime aplikacije koje korisnik vidi, itd. (Slika 5)

Element `<application>` sadrži `<activity>` elemente koji opisuju deklarisane aktivnosti (i elemente koji opisuju ostale komponente). Na primer, atributi `<activity>` elementa su `android:screenOrientation`, koji određuje orijentaciju ekrana (može imati vrednost *landscape* ili *portrait*), `android:name`, koji predstavlja ime aktivnosti koje je vidljivo samo sistemu, itd.

Sve komponente se deklarišu na ovaj način, samo u različitim elementima:

- `<activity>` za aktivnosti;
- `<service>` za servise;
- `<reciever>` za prijemnike emisija;
- `<provider>` za dobavljače sadržaja.

Aktivnosti, dobavljači sadržaja i servisi koji nisu deklarisani u Manifest fajlu postoje u aplikaciji ali ih sistem nikako ne može videti i nikada neće biti aktivirani. Prijemnici emisija mogu biti deklarisani u Manifest fajlu ili dinamički stvoreni kroz kôd (kao `BroadcastReciever` objekti) i registrovani kod sistema pozivom metode `registerReciever()`.

4.2 Deklaracija mogućnosti komponenti

Kao što je već rečeno, aktivnosti, servisi i prijemnici emisija se mogu pokrenuti pomoću Intent objekata. To se može uraditi tako što se kod Intent objekta navede ime klase komponente koja se treba pokrenuti. Prava moć Intent objekata leži u njihovoj mogućnosti da od sistema zatraže izvršenje željene akcije (*intent action*). Tako je moguće u Intent objektu navesti akciju koju sistem treba izvršiti (eventualno i podatke nad kojima treba biti izvršena) i sistem će naći komponentu u uređaju koja je u mogućnosti da obavi zadatak i pokrenuti je. Ukoliko je više komponenti sposobno da izvrši željenu akciju, korisnik bira koja će biti pokrenuta.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.replica.replicaisland"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />
    <uses-permission android:name="android.permission.VIBRATE" />
    <supports-screens
        android:anyDensity="true"
        android:largeScreens="true"
        android:normalScreens="true"
        android:smallScreens="false" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/icon"
        android:label="@string/app_name"
        android:theme="@android:style/Theme.Black.NoTitleBar.Fullscreen" >
        <activity
            android:name=".MainMenuActivity"
            android:configChanges="keyboardHidden|orientation"
            android:label="@string/app_name"
            android:screenOrientation="landscape" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="AndouKun"
            android:configChanges="keyboardHidden|orientation"
            android:label="@string/app_name"
            android:launchMode="singleTask"
            android:screenOrientation="landscape" />
        <activity
            android:name="Map"
            android:configChanges="keyboardHidden|orientation"
            android:label="@string/app_name"
            android:screenOrientation="landscape" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>

        .........
```

Slika 5: Deo koda Manifest fajla aplikacije Lost Illusions

Sistem identifikuje komponente koje mogu izvršiti akciju opisanu u Intent objektu tako što ga poredi sa *Intent filterima* koji su deklarisani u Manifest fajlu svake aplikacije u uređaju.

Prilikom deklaracije komponente u Manifest fajlu mogu se u njenom elementu deklarisiati i Intent filteri koji će joj omogućiti da odgovori Intent objektima iz drugih aplikacija. Filteri se deklarišu u `<intent-filter>` elementima. (Slika 5)

Na primer, email aplikacija sa aktivnošću za kompoziciju nove poruke može u svom Manifestu fajlu imati deklarisan Intent filter koji će joj omogućiti da odgovori svim "pošalji" Intent objektima. Tada aktivnost neke aplikacije može stvoriti Intent objekat sa akcijom "pošalji" (`ACTION_SEND`), koji će sistem povezati sa aktivnošću u aplikaciji za slanje email-ova i pokrenuti je kada se Intent objekat prosledi metodi `startActivity()`.

```
protected class StartActivityAfterAnimation implements
        Animation.AnimationListener {
    private Intent mIntent;

    StartActivityAfterAnimation(Intent intent) {
        mIntent = intent;
    }

    public void onAnimationEnd(Animation animation) {

        startActivity(mIntent);

        if (UIConstants.mOverridePendingTransition != null) {
            try {
                UIConstants.mOverridePendingTransition.invoke(
                        MainMenuActivity.this, R.anim.activity_fade_in,
                        R.anim.activity_fade_out);
            } catch (InvocationTargetException ite) {
                DebugLog.d("Activity Transition",
                        "Invocation Target Exception");
            } catch (IllegalAccessException ie) {
                DebugLog.d("Activity Transition",
                        "Illegal Access Exception");
            }
        }
    }
```

Slika 6: Primer upotrebe Intent objekata

4.3 Deklaracija potreba i zahteva komponenti

Postoji mnogo različitih Android uređaja i nemaju svi iste karakteristike. Kako bi se sprečila instalacija aplikacije na uređaj koji ne može ispuniti sve njene zahteve i pružiti joj određene usluge, u njenom Manifest fajlu treba deklarisati njene softverske potrebe. Te deklaracije su najčešće samo informativne prirode i sistem ih ne čita, za razliku od Google Play-a koji zahvaljujući njima korisnicima prikazuje samo aplikacije koje su u skladu sa njihovim uređajem.

Na primer, ukoliko aplikacija zahteva upoterbu kamere i stvorena je u skladu sa API-ima Android 2.1 verzije (API nivo 7), treba deklarisati ove neophodnosti u njenom Manifest fajlu. Onda uređaji koji nemaju kameru ili imaju verziju Androida stariju od 2.1, ne mogu preuzeti i instalirati aplikacju sa Google Play-a.

Moguće je i deklarisati da aplikacija može koristiti kameru, ali da ona *nije neophodna*. Onda će uređaj instalirati aplikaciju u svakom slučaju ali će ona proveriti da li uređaj poseduje kameru i ukoliko ne, isključiti sve svoje funkcije koje su je zahtevale.

Ovo su neke od karakteristika uređaja koje treba uzeti u obzir prilikom kreiranja aplikacija i deklaracije njihovih potreba:

Veličina i gustina ekrana: Usled raznovrsnosti ekrana Android uređaja, posmatraju se njihove dve karakteristike: veličina ekrana (fizičke dimenzije, širina i dužina) i gustina ekrana (gustina piksela na ekranu ili dpi[5]). Na ovaj način se svi ekrani svrstavaju u sledeće ketagorije:

- po veličini ekrana: mali, normalni, veliki i ekstra veliki;
- po gustini piksela: male, srednje, visoke i ekstra visoke gustine.

Aplikacija je podrazumevano kompaktibilna sa svim tipovima ekrana jer Android sistem sam modifikuje UI aplikacije u odnosu na ekran. Sve jedno, trebalo bi dizajnirati posebne layout-e za svaki tip ekrana i te tipove deklarisati

[5]dots per inch

u Manifest fajlu (u elementu `<supports-screens>`) radi boljih performansi aplikacije.

Mehanizmi za ulaz (*input*): Postoji više ulaznih uređaja koji se mogu povezati sa Android uređajem (npr. tastatura, postolje za crtanje, trackball itd). Ukoliko aplikacija zahteva prisustvo posebnog ulaznog uređaja, taj zahtev treba deklarisati u Manifest fajlu, u elementu `<uses-configuration>`.

Karakteristike uređaja: Postoji mnogo hardverskih i softverskih karakteristika koje Android uređaji mogu posedovati ili ne. Takve su kamera, senzor za svetlost, bluetooth, određena verzija OpenGL-a ili određena preciznost touchscreen-a. Nikada ne treba pretpostavljati da svi uređaji poseduju određenu karakteristiku (ukoliko ona nije podrazumevana Android standardima) i njenu neophodnost treba deklarisati u elementu `<uses-feature>` Manifest fajla.

Verzija platforme: Različiti uređaji imaju različite verzije Android OS-a (ili platforme), kao što su Android 1.2, Android 3.3, Android 4.0 itd. Svaka sukcesivna verzija koristi skup novih API-a koji su nepoznati njenim prethodnicima. Kako bi se znalo koje API-e neka verzija platforme poseduje, ona precizira svoj API nivo (npr. Android 1.0 je API nivo 1 a Android 2.3 je API nivo 9). Dakle, u elementu `<uses-sdk>` Manifest fajla treba deklarisati minimalni API nivo čije API-e aplikacija zahteva.

Veoma je važno deklarisati sve potrebe i zahteve aplikacije kako bi je Google Play mogao distribuirati samo korisnicima sa čijim je uređajima ona kompaktibilna.

5 Resursi aplikacije

Android aplikacija nije sačinjena samo od koda - nju čine i mnogi resursi koji su odvojeni od njega, kao što su slike, audio fajlovi i mnogi drugi resursi koji su zaslužni za vizuelni deo aplikacije. Na primer, animacije, meniji, boje, style-ovi i layout-i aktivnosti korisničkog interfejsa se definišu kao XML fajlovi. Korišćenje resursa omogućuje laku modifikaciju izgleda aplikacije bez promene koda kao i njenu optimizaciju alternativnim skupom resursa (promenu jezika, rezolucije slika itd).

Svakom resursu koji se doda u Android projekat sistem dodeljuje unikatni celobrojni ID koji kôd aplikacije koristi kako bi resursu pristupio. Na primer, ukoliko se projektu doda fajl `logo.png` (sačuvan u `res/drawable` direktorijumu), SDK alatke generišu jedinstveni ID nazvan `R.drawable.logo` koji se u kodu može koristiti za dalju manipulaciju slike.

Jedna od najbitnijih posledica odvajanja resursa od koda jeste mogućnost pružanja alternativnih resursa za različite konfiguracije uređaja. Na primer, kod definisanja stringova UI-a moguće je stvoriti dva XML fajla - jedan sa stringovima na engleskom, a drugi sa istim stringovima francuskom jeziku. Tada će se u zavisnosti od *kvalifikatora* kod naziva direktorijuma u kojem se XML fajl sa stringovima nalazi (npr. stringove na francuskom treba smestiti u direktorijum `res/values-fr`) i generalnih podešavanja uređaja, aplikacija prikazati sa stringovima na određenom jeziku (npr. francuskom).

Android pruža mnogo različitih kvalifikatora za alternativne resurse. Kvalifikatori su kratki stringovi koji se dodaju kod naziva direktorijuma u kojima se smeštaju alternativni resursi, kako bi sistem na osnovu podešavanja telefona prepoznao odgovarajući skup resursa. Pored alternativnih stringova, trebalo bi dizajnirati i alternativne layout-e - za različite veličine, orijentacije i gustine ekrana. Na primer, možda nije poželjno da dugmići budu isto raspoređeni u landscape i portrait orijentaciji ekrana. Onda treba dizajnirati layout-e za obe orijentacije i smestiti ih u foldere sa odgovarajućim kvalifikatorima, kako bi sistem znao kada koji da upotrebi.

6 Literatura i resursi

developer.android.com
wikipedia.org
code.google.com
Lost Illusions (Luka Lovre, Lazar Stojković, Replica Island developers)

QR kod aplikacije Lost Illusions