# 2

# Osnove aplikacija: aktivnosti i namere

Svaka Android aplikacija sastoji se od jednog Android projekta. U svakom receptu, naveden je pregled strukture projekta, uključujući i kratak opis osnovnih gradivnih blokova koji čine aplikaciju. Zatim se poglavlje usredsređuje na aktivnosti i namere koje ih pokreću.

## Uvod u projektovanje Android aplikacija

Svaka Android aplikacija sastoji se od raznih funkcionalnosti. Kao primere funkcionalnosti možemo navesti unošenje i ispravljanje beleške, reprodukovanje muzičke datoteke, aktiviranje alarma ili unošenje novog telefonskog kontakta. Te funkcionalnosti mogu se raspodeliti na četiri Androidove komponente, prikazane u tabeli 2.1, a svakoj odgovara različita osnovna klasa u jeziku Java.

**Tabela 2.1** Četiri moguće komponente Android aplikacije

| Funkcionalnost | Osnovna Java klasa | Primeri |
|---|---|---|
| Operacija koju korisnik može obavljati | `Activity` | Unošenje beleške, igranje računarske igre |
| Pozadinski proces | `Service` | Reprodukovanje muzike, ažuriranje izgleda ikonice za vremensku prognozu |
| Primanje poruka | `BroadcastReceiver` | Aktiviranje alarma pri određenom događaju |
| Unošenje i učitavanje podataka | `ContentProvider` | Otvaranje novog telefonskog kontakta |

Svaku aplikaciju čini jedna li više navedenih komponenata, koje instancira operativni sistem (OS) Android kada zatrebaju. Mogu ih koristiti i druge aplikacije, u granicama zadatih ovlašćenja.

Kad god se od operativnog sistema zahteva određena funkcionalnost (koju ne mora obavezno da pruža tekuća aplikacija, kao što je obrada dolaznog telefonskog poziva), svaka komponenta prolazi kroz ciklus formiranja, dobijanja fokusa, gubljenja fokusa i uništavanja. To podrazumevano ponašanje može se menjati prema potrebama aplikacije; recimo, kada zatreba da sačuvate vrednosti promenljivih ili obnovite elemente korisničkog interfejsa (UI).

Izuzev komponente `ContentProvider`, sve druge komponente aktivira asinhrona poruka koja se zove *namera* (engl. *intent*). Namera može sadržati *blok pratećih podataka* (engl. *bundle*) koji opisuju komponentu. To je način na koji komponente međusobno razmenjuju informacije.

Preostali deo ovog poglavlja ilustruje navedene koncepte na primerima upotrebe najuobičajenije komponente, a to je komponenta `Activity` koja predstavlja određenu aktivnost korisnika. Pošto aktivnosti gotovo uvek predstavljaju neku interakciju s korisnikom, za svaku aktivnost automatski se otvara nov prozor. Iz tog razloga dodat je i kratak uvod u korisnički interfejs. Što se drugih komponenata tiče, komponente `Service` i `BroadcastReceiver` opisane su u poglavlju 3, „Niti, servisi, prijemnici i alarmi“ a komponenta `ContentProvider` opisana je u poglavlju 9, „Metode za skladištenje podataka“.

## Recept: Izrada projekta i pokretanje aktivnosti

Najjednostavniji način da napravite Android projekat jeste u integrisanom razvojnom okruženju Eclipse. Tako automatski dobijate odgovarajuću strukturu pratećih datoteka. Postupak otvaranja novog Android projekta sastoji se od sledećih koraka:

1. U okruženju Eclipse izaberite **File → New → Android Project**; otvara se prozor New Android Project.
2. U polje Project name upišite ime novog projekta, na primer, **SimpleActivity= Example**.
3. Na listi Build Target izaberite jednu od opcija. Sadržaj te liste zavisi od verzije SDK-a koju ste instalirali na svom razvojnom računaru.
4. U polje Application name upišite ime aplikacije, kao što je **Primer jednostavne aktivnosti**.
5. U polje Package name upišite ime paketa, kao što je **com.cookbook.simple_ activity**.
6. Da biste u istom koraku zadali i glavnu aktivnost, obavezno potvrdite opciju **Create Activity** i u polje Activity name upišite ime te aktivnosti, kao što je **SimpleActivity**.

Sve aktivnosti proširuju apstraktnu klasu `Activity` ili neku od njenih potklasa. Ulazna tačka svake aktivnosti je metoda `onCreate()`, koja se gotovo uvek redefiniše tako da inicijalizuje aktivnost. Na primer, u kodu te metode možete formirati elemente UI, definisati osluškivače za obradu događaja dugmadi, inicijalizovati određene parametre i pokretati nove niti.

Ako glavnu aktivnost niste zadali pri otvaranju novog projekta ili je potrebno da projektu dodate novu aktivnost, postupak dodavanja nove aktivnosti sastoji se od sledećih koraka:

1. Napravite klasu koja proširuje klasu `Activity`. (U okruženju Eclipse, to možete uraditi ako desnim tasterom miša pritisnete projekat, izaberete **New → Class**, a zatim kao superklasu zadate `android.App.Activity`.)
2. Redefinišite funkciju `onCreate()`. (U okruženju Eclipse, to možete uraditi ako desnim tasterom miša pritisnete datoteku klase, izaberete opciju **Source → Override/Implement Methods...**, a zatim potvrdite polje pored metode `onCreate()`.)
3. Kao i većina redefinisanih funkcija, metoda `onCreate()` mora pozvati i istoimenu metodu superklase jer se inače može dogoditi da se pri izvršavanju javi izuzetak. U ovom primeru, prvo treba da pozovete `super.onCreate()` da biste pravilno inicijalizovali aktivnost (listing 2.1).

**Listing 2.1** src/com/cookbook/simple_activity/SimpleActivity.java

```
package com.cookbook.simple_activity;
import android.App.Activity;
import android.os.Bundle;

public class SimpleActivity extends Activity {

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

4. Ako klasa ima korisnički interfejs, definišite njegove elemente u XML datoteci koju ćete smestiti u direktorijum **res/layout/**. U ovom primeru, ta datoteka se zove **main.xml** (listing 2.2.)
5. Povežite definiciju korisničkog interfejsa i aktivnost pomoću metode `setContentView()`, kojoj ćete kao parametar proslediti identifikator resursa koji predstavlja XML datoteku defincije UI. U ovom primeru, vrednost tog identifikatora je `R.layout.main`, kao što se vidi iz listinga 2.1.

**Listing 2.2** res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas. Android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<TextView
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="@string/hello"
   />
</LinearLayout>
```

6. Deklarišite svojstva aktivnosti u datoteci AndroidManifest.xml. To je detaljnije opisano u listingu 2.5.

Imajte u vidu da su resursi znakovnog tipa definisani u datoteci **strings.xml**, koja se nalazi u direktorijumu **res/values/** (listing 2.3). To je centralno mesto gde se čuvaju svi znakovni nizovi u slučaju da vam zatreba da izmenite ili ponovo upotrebite određeni znakovni niz.

**Listing 2.3** res/values/strings.xml

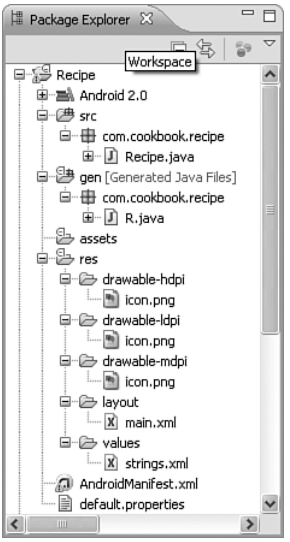
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="hello">Hello World, SimpleActivity!</string>
   <string name="app_name">SimpleActivity</string>
</resources>
```

Sada prelazimo na detaljnije razmatranje strukture direktorijuma ovog projekta i na sadržaj koji se automatski generiše za svaki nov projekat.

## Struktura direktorijuma projekta i sadržaj koji se automatski generiše

Slika 2.1 prikazuje primer strukture projekta, onako kako se prikazuje u prozoru Package Explorer razvojnog okruženja Eclipse.

Izuzev biblioteke Android 2.0, struktura projekta sastoji se od kombinacije datoteka koje generiše korisnik i onih koje se generišu automatski.




**Slika 2.1** Struktura direktorijuma Android projekta koja se vidi u prozoru Package Explorer razvojnog okruženja Eclipse.

Korisnik generiše sledeće datoteke:

- Direktorijum **src/** sadrži Java pakete koje programer piše ili uvozi u aplikaciju. Svaki paket može sadržati više .java datoteka koje sadrže pojedine klase.
- Direktorijum **res/layout/** sadrži XML datoteke koje određuju izgled i sadržaj svakog ekrana aplikacije.
- Direktorijum **res/values/** sadrži XML datoteke koje referenciraju druge datoteke.
- Direktorijumi **res/drawable-hdpi/**, **res/drawable-mdpi/** i **res/drawable-ldpi/** su direktorijumi sa slikama koje se koriste u aplikaciji. Te slike su grupisane u visoku, srednju i nisku rezoluciju (broj tačaka po inču).
- Direktorijum **assets/** sadrži dodatne, nemedijske datoteke koje se koriste u aplikaciji.
- Datoteka **AndroidManifest.xml** opisuje projekat operativnom sistemu Android.

Sledeće datoteke generišu se automatski:

- Direktorijum **gen/** sadrži automatski generisan kôd, uključujući i generisanu klasu **R.java**.
- Datoteka **default.properties** sadrži parametre projekta. Iako se ona automatski generiše, trebalo bi da je stavite pod kontrolu sistema za upravljanje verzijama koda.

Resursima aplikacije pripadaju XML datoteke definicija korisničkog interfejsa, XML datoteke koje sadrže vrednosti kao što su znakovni nizovi i natpisi na elementima korisničkog interfejsa. Resursi su i dodatne prateće datoteke, kao što su slike i zvučni zapisi. U trenutku prevođenja koda projekta, reference na sve resurse koje aplikacija koristi grupišu se u automatski generisanu omotačku klasu (engl. *wrapper class*) koja se zove **R.java**. Tu datoteku automatski generiše alatka Android Asset Packaging Tool (aapt). Listing 2.4 prikazuje kako izgleda sadržaj te datoteke za projekat u receptu „Izrada projekta i aktivnosti".

**Listing 2.4** gen/com/cookbook/simple_activity/R.java

```
/* AUTO-GENERATED FILE. DO NOT MODIFY.
*
* This class was automatically generated by the
* aapt tool from the resource data it found. It
* should not be modified by hand.
*/

package com.cookbook.simple_activity;

public final class R {
   public static final class attr {
   }
   public static final class drawable {
      public static final int icon=0x7f020000;
   }
   public static final class layout {
      public static final int main=0x7f030000;
   }
   public static final class string {
      public static final int app_name=0x7f040001;
      public static final int hello=0x7f040000;
   }
}
```

U toj datoteci, svakom resursu je dodeljena jedinstvena celobrojna vrednost. Na taj način klasa **R.java** omogućava referenciranje spoljnih resursa u Java kodu. Na primer, XML datoteka definicije korisničkog interfejsa, **main.xml**, u Java kodu se referencira pomoću celobrojne vrednosti `R.layout.main`. Ta ista datoteka se u XML datotekama referencira pomoću znakovnog niza „`@layout/main`".

Način referenciranja resursa u Java ili XML datotekama prikazan je u tabeli 2.2. Zapamtite sledeće: kada definišete identifikator (ID) za novo dugme čije je ime `home_button`, znakovnom nizu koji predstavlja to dugme morate dodati znak plus: `@+id/home_button`. Više informacija o resursima naći ćete u poglavlju 4, „Korisnički interfejs aplikacije" ali je ovo zasad dovoljno za recepte iz ovog poglavlja.

**Tabela 2.2** Kako se različite vrste resursa referenciraju u Java i u XML datotekama

| Resurs | Način referenciranja u Java kodu | Način referenciranja u XML kodu |
|---|---|---|
| **res/layout/main.xml** | `R.layout.main` | `@layout/main` |
| **res/drawable-hdpi/icon.png** | `R.drawable.icon` | `@drawable/icon` |
| `@+id/home_button` | `R.id.home_button` | `@id/home_button` |
| `<string name="hello">` | `R.string.hello` | `@string/hello` |

## Android paket i datoteka manifesta

Android projekat, koji se ponekad naziva i Android paket, sastoji se od grupe Java paketa. Pojedini Android paketi mogu sadržati Java pakete sa istim imenima, ali ime Android paketa mora biti jedinstveno među svim aplikacijama instaliranim na Android uređaju.

Da bi operativni sistem mogao da joj pristupi, svaka aplikacija mora da deklariše sve svoje komponente u jednoj XML datoteci čije je ime AndroidManifest.xml. Osim toga, ta datoteka sadrži opise ovlašćenja i ponašanja koji su potrebni za rad aplikacije. Listing 2.5 prikazuje kako ta datoteka izgleda za recept „Izrada projekta i aktivnosti".

**Listing 2.5** AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas. Android.com/apk/res/android"
          package="com.cookbook.simple_activity"
          android:versionCode="1"
          android:versionName="1.0">
       <application android:icon="@drawable/icon"
              android:label="@string/app_name">
          <activity android:name=".SimpleActivity"
                 android:label="@string/app_name">
             <intent-filter>
                <action android:name="android.intent. Action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
             </intent-filter>
          </activity>
       </application>
       <uses-sdk android:minSdkVersion="3" />
</manifest>
```

Prvi red je obavezan i standardan u svim XML datotekama na Androidu, a svrha mu je da definiše način kodiranja znakovnih nizova. Element `manifest` definiše ime i verziju Android paketa. Atribut `versionCode` je celobrojna vrednost koju programi mogu učitavati i tumačiti kao redni broj verzije aplikacije. Atribut `versionName` je format čitljiv za ljude u kojem se mogu deklarisati broj glavne verzije i broj podverzije.

Element `application` definiše ikonicu i natpis (engl. *label*) koje će korisnik videti u meniju Android uređaja. Natpis je znakovni niz koji bi trebalo da bude dovoljno kratak da se može prikazati ispod ikonice na korisnikovom uređaju. Da bi se prikazao ceo, bez odsecanja, taj natpis najčešće se sastoji od dve reči, svaka s najviše deset znakova.

Element `activity` definiše glavnu aktivnost koja se pokreće pri podizanju aplikacije i čije se ime vidi na naslovnoj traci dok je aktivnost aktivna. U tom elementu treba zadati ime Java paketa, što je u ovom primeru `com.cookbook.simple_activity.SimpleActivity`. Pošto je ime Java paketa najčešće isto kao i ime Android paketa, često se koristi skraćeni oblik: `.SimpleActivity`. Međutim, najbolje da ne gubite iz vida da se imena Android paketa i Java paketa odnose na različite stvari.

Element `intent-filter` opisuje operativnom sistemu Android odlike komponente. U njemu možete zadati više raznih akcija, kategorija ili elemenata za pristupanje podacima. Primere toga videćete u pojedinim receptima.

Element `uses-sdk` definiše nivo API (*application programming interface*) paketa koji je potreban za rad aplikacije. Nivo API paketa najčešće se zadaje u sledećem obliku:

```
<uses-sdk android:minSdkVersion="integer"
          android:targetSdkVersion="integer"
          android:maxSdkVersion="integer" />
```

Pošto je OS Android projektovan tako da bude kompatibilan unapred, upotreba atributa `maxSdkVersion` se ne preporučuje, a na uređajima na kojima radi Android 2.0.1 ili noviji, taj atribut nije čak ni podržan. Atribut `targetSdkVersion` nije obavezan, ali na uređajima koji imaju zadatu verziju SDK, omogućava da isključe parametre za kompatibilnost, što može da ubrza rad uređaja. Trebalo bi da uvek zadate atribut `minSdkVersion` kako biste izbegli rušenje aplikacije kada ona radi na platformi koja ne podržava sve mogućnosti potrebne za rad aplikacije. Kada zadajete taj atribut, uvek zadajte najniži mogući nivo API paketa.

Datoteka AndroidManifest.xml može sadržati i parametre koji opisuju bezbednosna ovlašćenja potrebna za rad aplikacije. Više informacija o tim opcijama naći ćete poglavljima koja slede, ali zasad je ovo dovoljno za recepte iz ovog poglavlja.

## Preimenovanje delova aplikacije

Ponekad je potrebno preimenovati deo Android projekta. Možda ste određenu datoteku ručno iskopirali u projekat, na primer, iz ove knjige. Možda ste tokom razvoja izmenili ime aplikacije i morate ga preslikati na celo stablo sistema datoteka. Za tu namenu postoje automatske alatke koje obezbeđuju automatsko ažuriranje referenci. Na primer, u okruženju Eclipse, delove aplikacije možete preimenovati na jedan od sledećih načina:

- Android projekat možete preimenovati ovako:
  1. Desnim tasterom miša pritisnite projekat, pa izaberite **Refactor → Move** i zadajte nov direktorijum u sistemu datoteka.
  2. Desnim tasterom miša pritisnite projekat, pa izaberite **Refactor → Rename** i zadajte novo ime projekta.

- Android paket možete preimenovati ovako:
  1. Desnim tasterom miša pritisnite paket, pa izaberite **Refactor → Rename** i zadajte novo ime paketa.
  2. Ažurirajte datoteku **AndroidManifest.xml** tako da sadrži novo ime paketa.
- Android klasu (kao što je jedna od glavnih komponenata tipa `Activity`, `Service`, `BroadcastReceiver` ili `ContentProvider`), možete preimenovati ovako:
  1. Desnim tasterom miša pritisnite **.java** datoteku, a zatim izaberite **Refactor → Rename** i zadajte novo ime klase.
  2. Ažurirajte datoteku **AndroidManifest.xml** tako da element `android:name` sadrži novo ime komponente.

Imajte u vidu da preimenovanje drugih datoteka, kao što su XML datoteke, najčešće zahteva ručno menjanje odgovarajućih referenci u Java kodu.

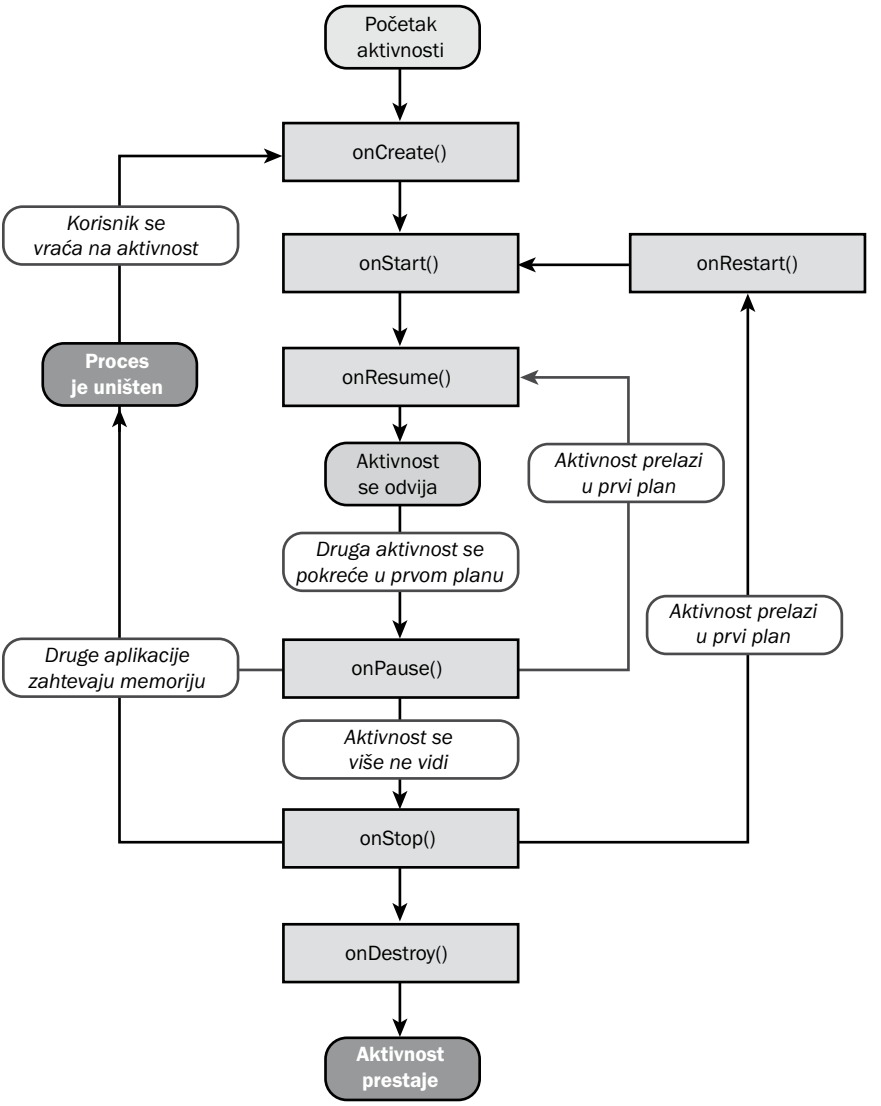
## Životni ciklus aktivnosti

Svaka aktivnost u aplikaciji prolazi kroz svoj životni ciklus. Funkcija `onCreate()` izvršava se samo jedanput, i to kada aktivnost počinje. Ako se aktivnost završi, izvršava se funkcija `onDestroy()`. Između te dve funkcije, odvijaju se razni događaji koji čine da aktivnost prelazi u razna stanja, kao što je ilustrovano slikom 2.2. Sledeći recept sadrži po jedan primer svake od tih funkcija.

## Recept: Upotreba drugih funkcija životnog ciklusa aktivnosti

Sledeći recept pruža jednostavan prikaz životnog ciklusa u akciji. Ilustracije radi, svaka redefinisana funkcija je eksplicitno navedena a na ekranu se pomoću klase `Toast` prikazuje poruka kad god se izvršava jedna od tih funkcija (više informacija o klasi `Toast` naći ćete u poglavlju 3). Listing 2.6 prikazuje kôd aktivnosti. Pokrenite je na Android uređaju i ispitajte razne slučajeve. Obratite pažnju na sledeće:

- Kada se promeni orijentacija ekrana, aktivnost se uništava i ponovo formira od početka.
- Kada pritisnete dugme Home, aktivnost se zaustavlja, ali ne uništava.
- Kada pritisnete ikonicu aplikacije, može se pokrenuti nova instanca (primerak) aktivnosti, čak i ako postojeća nije uništena.
- Kada se ekran isključi, aktivnost prelazi u stanje pauze, a ponovno uključivanje ekrana čini da se aktivnost nastavi. (To je slično obradi dolaznog poziva.)




**Slika 2.2** Životni ciklus aktivnosti (izvor: http://developer. Android.com/)

**Listing 2.6** src/com/cookbook/activity_lifecycle/ActivityLifecycle.java

```
package com.cookbook.Activity_lifecycle;

import android.App.Activity;
import android.os.Bundle;
import android.widget.Toast;

public class ActivityLifecycle extends Activity {
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Toast.makeText(this, "onCreate", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onStart() {
        super.onStart();
        Toast.makeText(this, "onStart", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onResume() {
        super.onResume();
        Toast.makeText(this, "onResume", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onRestart() {
        super.onRestart();
        Toast.makeText(this, "onRestart", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onPause() {
        Toast.makeText(this, "onPause", Toast.LENGTH_SHORT).show();
        super.onPause();
    }

    @Override
    protected void onStop()  {
        Toast.makeText(this, "onStop", Toast.LENGTH_SHORT).show();
        super.onStop();
    }

    @Override
    protected void onDestroy() {
        Toast.makeText(this, "onDestroy", Toast.LENGTH_SHORT).show();
        super.onDestroy();
    }
}
```

Kao što vidite, razne uobičajene akcije korisnika mogu učiniti da aktivnost pauzira, bude uništena ili čak pokrenuti više primeraka iste aplikacije. Pre nego što nastavimo, korisno je da razmotrimo još dva jednostavna recepta za upravljanje ponašanjem aktivnosti.

## Recept: Pokretanje jedinstvenog primerka aktivnosti

Kada korisnik s tekuće pređe na drugu aktivnost ili se vrati na prethodnu, može doći do pokretanja više instanci iste aktivnosti na uređaju. Suvišna instanca aktivnosti pre ili kasnije se uništava radi oslobađanja memorije, ali u međuvremenu može prouzrokovati čudne situacije. Da bi se to izbeglo, programer može odrediti ponašanje svake aktivnosti u datoteci AndroidManifest.xml.

Da bi se na uređaju pokretala uvek samo jedna instanca aktivnosti, u elementu `activity` kojem su pridruženi filtri `MAIN` i `LAUNCHER` zadajte sledeće:

```
android:launchMode="singleInstance"
```

Time podešavate da u svakom datom trenutku može raditi samo jedna instanca svake aktivnosti unutar datog posla. Osim toga, svaka aktivnost koja je potomak postojeće aktivnosti pokreće se unutar vlastitog posla (engl. *task*). Da biste zadali još strožije ograničenje da se sve aktivnosti aplikacije odvijaju unutar jednog posla, zadajte sledeće:

```
android:launchMode="singleTask"
```

Time omogućavate da aktivnosti unutar istog posla lako međusobno dele podatke.

Osim toga, može biti poželjno da se čuva tekuće stanje posla, bez obzira na to kako korisnik dođe do aktivnosti. Na primer, ako korisnik izađe iz aplikacije i posle je ponovo pokrene, podrazumevano ponašanje je često da se posao vraća u početno stanje. Kako biste obezbedili da se korisnik uvek vraća poslu u poslednjem stanju, zadajte sledeće u elementu `activity` korenske aktivnosti u poslu:

```
android:alwaysRetainTaskState="true"
```

## Recept: Kako fiksirati orijentaciju ekrana

Svaki Android uređaj opremljen meračem ubrzanja može utvrditi gde se nalazi „dole“. Kada se uređaj prebaci iz uspravnog u ležeći položaj, podrazumevana akcija je zaokretanje ekrana aplikacije u skladu s time. Međutim, kao što se vidi iz recepta „Upotreba drugih funkcija životnog ciklusa“, aktivnost se uništava i ponovo započinje pri svakoj promeni orijentacije ekrana. Kada se to dogodi, može se izgubiti tekuće stanje aktivnosti, što ometa korisnika.

Jedna mogućnost da obradite promene orijentacije ekrana jeste da sačuvate podatke o tekućem stanju pre promene i da stanje obnovite posle promene. Drugi, jednostavniji način koji bi bio upotrebljiv jeste da podesite fiksnu orijentaciju ekrana. Za svaku aktivnost, u datoteci AndroidManifest.xml možete zadati orijentaciju ekrana pomoću atributa `screenOrientation`. Na primer, da biste zadali da ekran aktivnosti uvek ostaje u uspravnom položaju, elementu `activity` dodajte sledeće:

```
android:screenOrientation="portrait"
```

Slično tome, horizontalnu orijentaciju zadaćete na sledeći način:

```
android:screenOrientation="landscape"
```

Međutim, ti atributi i dalje čine da se aktivnost uništava i ponovo započinje kada izvučete tastaturu. Iz tog razloga, postoji i treći način: obavestite OS Android da sama aplikacija obrađuje događaje promene orijentacije ekrana i izvlačenja tastature. To se radi dodavanjem sledećeg atributa u element `activity`:

```
android:configChanges="orientation|keyboardHidden"
```

Možete zadati samo taj atribut ili ga kombinovati sa atributom `screenOrientation` da biste zadali željeno ponašanje aplikacije.

## Recept: Snimanje i restauriranje podataka o aktivnosti

Neposredno pre nego što se aktivnot uništi, uvek se poziva funkcija `onSaveInstanceState()`. Redefinišite je da biste sačuvali odgovarajuće podatke. Kada se zatim ista aktivnost obnovi, poziva se funkcija `onRestoreInstanceState()`. Redefinišite tu funkciju da biste učitali sačuvane podatke. Time korisniku pružate utisak glatkog rada aplikacije kada ona prolazi kroz promene u svom životnom ciklusu. Imajte u vidu da većinom promena stanja nema potrebe da sami upravljate jer se o tome standardno stara sam sistem.

Funkcija `onSaveInstanceState()` radi drugačije od funkcije `onPause()`. Na primer, ako se ispred aktivnosti pokrene neka druga komponenta, poziva se funkcija `onPause()`. Kasnije, ukoliko je aktivnost i dalje u stanju pauze kada operativnom sistemu zatrebaju slobodni resursi, on poziva funkciju `onSaveInstanceState()` pre nego što uništi aktivnost.

Listing 2.7. prikazuje primer snimanja i obnavljanja stanja instance iz znakovne vrednosti i niza vrednosti tipa float.

**Listing 2.7** Primeri upotrebe funkcija onSaveInstanceState() i onRestoreInstanceState()

```
float[] localFloatArray = {3.14f, 2.718f, 0.577f};
String localUserName = "Euler";

@Override
protected void onSaveInstanceState(Bundle outState) {
   super.onSaveInstanceState(outState);
   //snimamo odgovarajuće podatke
   outState.putString("name", localUserName);
   outState.putFloatArray("array", localFloatArray);
}

@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
   super.onRestoreInstanceState(savedInstanceState);
   //obnavljamo odgovarajuće podatke
   localUserName = savedInstanceState.getString("name");
   localFloatArray = savedInstanceState.getFloatArray("array");
}
```

Imajte u vidu da funkcija `onCreate()` ima argument tipa `Bundle` čije je ime `savedInstanceState`. U slučaju da se aktivnost ponovo inicijalizuje posle uništavanja, takav objekat tipa `Bundle` snimljen u funkciji `onSaveInstanceState()` prosleđuje se i funkciji `onCreate()`. Pošto se u svim slučajevima takav objekat prosleđuje funkciji `onRestoreInstanceState()`, prirodno je da se on koristi za obnavljanje stanja.

## Istovremeno odvijanje više aktivnosti

Čak i najjednostavnije aplikacije pružaju više od jedne funkcionalnosti. Iz tog razloga, često je potrebno raditi s više aktivnosti istovremeno. Na primer, računarska igra može imati dve aktivnosti: prikazivanje ekrana s postignutim rezultatima i prikazivanje ekrana same igre. Rokovnik može imati tri aktivnosti: prikaz liste beležaka, prikaz izabrane beleške i ažuriranje izabrane beleške ili unošenje nove.

Glavna aktivnost, definisana u datoteci AndroidManifest.xml, započinje pokretanjem aplikacije. Ta aktivnost može da pokrene drugu aktivnost, najčešće kao odziv na određeni okidački događaj (engl. *trigger event*). To čini da glavna aktivnost prelazi u stanje pauze dok je aktivna sekundarna aktivnost. Kada se završi sekundarna aktivnost, glavna aktivnost se vraća u prvi plan i nastavlja.

Da bi se aktivirala data komponenta aplikacije, koristi se namera u kojoj se eksplicitno zadaje ime te komponente. Ako se zahtevi aplikacije mogu formulisati pomoću filtra za namere, može se upotrebiti implicitna (podrazumevana) namera (engl. *intent*). U tom slučaju sam sistem određuje koja je komponenta (ili komponente) najprikladnija za upotrebu, čak i ukoliko je ta komponenta u drugoj aplikaciji ili sastavni deo operativnog sistema. Imajte u vidu da za razliku od drugih aktivnosti, implicitne namere koje su definisane u drugim aplikacijama ne treba deklarisati u datoteci AndroidManifest.xml tekuće aplikacije.

Android koristi implicitne namere kad god je to moguće, čime obezbeđuje moćan okvir za modularnu funkcionalnost. Kada razvijete novu komponentu koja zadovoljava dati filtar za implicitne namere, ta komponenta se može koristiti umesto Androidove interne namere. Na primer, recimo da je na Android uređaj instalirana nova aplikacija za prikazivanje telefonskih kontakata. Kada korisnik izabere neki kontakt, operativni sistem Android pronalazi sve raspoložive aktivnosti pomoću odgovarajućeg filtra za pregledanje kontakata i pita korisnika koju od tih aktivnosti želi da upotrebi.

### Recept: Upotreba dugmadi i natpisa

Da bismo u punoj meri ilustrovali istovremeno odvijanje više aktivnosti, upotrebićemo okidački događaj. U tu svrhu ovde uvodimo događaj „pritiskanje dugmeta“. Postupak dodavanja dugmeta na ekran i pridruživanje određene akcije pritiskanju dugmeta sastoji se od sledećih koraka:

1. Definišite dugme u odgovarajućoj XML datoteci sadržaja ekrana:
```
<Button android:id="@+id/trigger"
      android:layout_width="100dip" android:layout_height="100dip"
      android:text="Press this button" />
```
2. U kodu aktivnosti deklarišite dugme na koje se odnosi ID dugmeta u datoteci sadržaja ekrana:
```
Button startButton = (Button) findViewById(R.id.Trigger);
```
3. Zadajte osluškivač za događaj kada korisnik pritisne dugme:
```
//definiše osluškivač za dugme
startButton.setOnClickListener(new View.OnClickListener() {
      //ovde dolazi kôd za obradu događaja onClick
});
```
4. Redefinišite funkciju `onClick` tako da ona obavlja potrebnu akciju:
```
public void onClick(View view) {
      //ovde dolazi kôd
}
```

Da biste prikazali rezultat određene akcije, korisno je da izmenite tekst natpisa na ekranu. Postupak definisanja natpisa i menjanja njegovog sadržaja iz koda programa izgleda ovako:

1. Definišite natpis kojem je pridružen ID u odgovarajućoj ciljnoj XML datoteci ekrana. Možete ga inicijalizovati nekom vrednošću (u ovom primeru, to može biti znakovni niz čije je ime „hello“ a nalazi se u datoteci strings.xml):
```
<TextView android:id="@+id/hello_text"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/hello" />
```
2. Deklarišite objekat tipa `TextView` koji upućuje na ID sadržan u elementu `TextView` u datoteci ekrana:
```
private TextView tv = (TextView) findViewById(R.id.hello_text);
```
3. Ako je potrebno izmeniti tekst, upotrebite funkciju `setText`:
```
tv.setText("new text string");
```

Navedene dve tehnike definisanja elemenata korisničkog interfejsa koriste se i u narednim receptima u ovoj knjizi. Opširniju ilustraciju tehnika za izradu korisničkog interfejsa naći ćete u poglavlju 4.

## Recept: Pokretanje druge aktivnosti iz koda za obradu događaja

U ovom receptu, glavna aktivnost je `MenuScreen`, kao što se vidi iz listinga 2.8. Ona pokreće novu aktivnost, `PlayGame`. U ovom slučaju, okidački događaj je pritiskanje dugmeta.

Kada korisnik pritisne dugme, izvršava se funkcija `startGame()` koja pokreće aktivnost `PlayGame`. Kada korisnik pritisne dugme u aktivnosti `PlayGame`, poziva se funkcija `finish()` koja vraća kontrolu pozivajućoj aktivnosti. Postupak pokretanja nove aktivnosti sastoji se od sledećih koraka:

1. Deklarišite nameru (`Intent`) koja upućuje na dodatnu aktivnost koju treba pokrenuti.
2. Za tu nameru pozovite funkciju `startActivity`.
3. Deklarišite dodatnu aktivnost u datoteci AndroidManifest.xml.

**Listing 2.8** src/com/cookbook/launch_activity/MenuScreen.java

```
package com.cookbook.launch_activity;

import android.App.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MenuScreen extends Activity {

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      //definišemo funkciju za obradu događaja
      Button startButton = (Button) findViewById(R.id.play_game);
      startButton.setOnClickListener(new View.OnClickListener() {
         public void onClick(View view) {
            startGame();
         }
      });
   }

   private void startGame() {
      Intent launchGame = new Intent(this, PlayGame.class);
      startActivity(launchGame);
   }
}
```

### Prosleđivanje tekućeg konteksta anonimnoj internoj klasi

U listingu 2.8 obratite pažnju na dodatne mere koje je potrebno preduzeti kada se aktivnost pokreće kao odziv na pritiskanje dugmeta. Nameri je potreban kontekst. Međutim, prečica `this` u kodu funkcije `onClick` nije pravilno definisana. Tekući kontekst možete proslediti anonimnoj internoj klasi na jedan od sledećih načina:

- Zadajte `Context.this` umesto `this`.
- Zadajte `getApplicationContext()` umesto `this`.
- Eksplicitno navedite ime klase `MenuScreen.this`.

Pozovite funkciju koja je definisana na odgovarajućem nivou konteksta. Tako je urađeno u listingu 2.8: startGame().

Uglavnom je svejedno koju ćete od navedenih metoda primeniti. Upotrebite onu koja je najjasnija u datoj situaciji.

Aktivnost `PlayGame`, prikazana u listingu 2.9, samo je dugme za koje funkcija `onClick` poziva funkciju `finish()` koja vraća kontrolu glavnoj aktivnosti. Pozvanoj aktivnosti možete dodati proizvoljnu funkcionalnost, a funkcija `finish()` može se pozvati i iz više grana koda.

**Listing 2.9** src/com/cookbook/launch_activity/PlayGame.java

```
package com.cookbook.launch_activity;

import android.App.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class PlayGame extends Activity {
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.game);

      //definišemo funkciju za obradu događaja
      Button startButton = (Button) findViewById(R.id.end_game);
      startButton.setOnClickListener(new View.OnClickListener() {
         public void onClick(View view) {
            finish();
         }
      });
   }
}
```

Glavnom ekranu treba dodati dugme, na način prikazan u listingu 2.10, čiji ID `play_game` odgovara onome deklarisanom u listingu 2.8. U ovom slučaju je deklarisana i veličina dugmeta, u pikselima nezavisnim od uređaja (dip), što je detaljnije opisano u poglavlju 4.

**Listing 2.10** res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas. Android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
   <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/hello"
      />
```

```
   <Button android:id="@+id/play_game"
      android:layout_width="100dip"   android:layout_height="100dip"
      android:text="@string/play_game"
      />
</LinearLayout>
```

Aktivnost `PlayGame` referencira vlastiti ID dugmeta `end_game` u resursu `R.layout.game`, koji odgovara XML datoteci ekrana **game.xml**, kao što se vidi u listingu 2.11.

**Listing 2.11** res/layout/game.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas. Android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<Button android:id="@+id/end_game"
   android:layout_width="100dip" android:layout_height="100dip"
   android:text="@string/end_game" android:layout_centerInParent="true"
   />
</LinearLayout>
```

Mada se tekst na dugmetu uvek može eksplicitno zadati u kodu, preporučuje se da za svaki znakovni niz definišete promenljivu. U ovom receptu, treba deklarisati dve znakovne vrednosti, `play_game` i `end_game`, u XML datoteci za znakovne resurse, kao što se vidi u listingu 2.12.

**Listing 2.12** res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="hello">This is the Main Menu</string>
   <string name="app_name">LaunchActivity</string>
   <string name="play_game">Play game?</string>
   <string name="end_game">Done?</string>
</resources>
```

I najzad, u datoteci AndroidManifest.xml morate registrovati podrazumevanu akciju za novu klasu `PlayGame` (listing 2.13).

**Listing 2.13** AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas. Android.com/apk/res/android"
      android:versionCode="1"
      android:versionName="1.0"   package="com.cookbook.launch_activity">
   <application android:icon="@drawable/icon"
               android:label="@string/app_name">
      <activity android:name=".MenuScreen"
               android:label="@string/app_name">
         <intent-filter>
            <action android:name="android.intent.Action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
      <activity android:name=".PlayGame"
               android:label="@string/app_name">
         <intent-filter>
            <action android:name="android.intent.Action.VIEW" />
            <category android:name="android.intent.category.DEFAULT" />
         </intent-filter>
      </activity>
   </application>
   <uses-sdk android:minSdkVersion="3" />
</manifest>
```

## Recept: Pretvaranje govora u tekst kao primer aktivnosti koja vraća rezultat

Ovaj recept ilustruje pokretanje aktivnosti radi postizanja određenog rezultata. Recept takođe pokazuje kako se koristi funkcionalnost pretvaranja govora u tekst pomoću Googleove aktivnosti `RecognizerIntent` i ispisivanja njenog rezultata na ekranu. U ovom primeru, okidački događaj je pritiskanje dugmeta, što pokreće aktivnost `RecognizerIntent`, koja prepoznaje govor na osnovu zvuka iz mikrofona i pretvara ga u tekst. Kada se završi generisanje teksta, on se prosleđuje pozivajućoj aktivnosti.

Pri povratku u pozivajuću aktivnost, prvo se poziva funkcija `onActivityResult()` s vraćenim podacima, a zatim se poziva i funkcija `onResume()` da bi se nastavilo normalno odvijanje pozivajuće aktivnosti. Može se dogoditi da se rad pozvane aktivnosti završi neuspehom i pojavi se problem. Iz tog razloga, pre nego što nastavite obradu povratnih podataka, trebalo bi da uvek prethodno ispitate vrednost `resultCode` da biste se uverili da je to `RESULT_OK`.

Imajte u vidu da praktično svaka aktivnost koju pokrenete iz druge aktivnosti i koja vraća neke podatke, čini da se poziva ista funkcija – `onActivityResult()`. Iz tog razloga je uobičajeno da se zada šifra zahteva (`requestCode`) na osnovu koje se utvrđuje koja je aktivnost vratila podatke. Kada se završi pozvana aktivnost, ona vraća kontrolu pozivajućoj aktivnosti i poziva funkciju `onActivityResult()` s tom istom šifrom zahteva.

Postupak pokretanja druge aktivnosti iz tekuće sastoji se od sledećih koraka:

1. Pozovite funkciju `startActivityForResult()` i prosledite joj odgovarajuću nameru, koja definiše aktivnost za pokretanje i vrednost `requestCode`.
2. Redefinišite funkciju `onActivityResult()` tako da ispituje status povratnog rezultata, proverava da li je vraćena odgovarajuća vrednost `requestCode` i obrađuje povratne podatke.

Aktivnost `RecognizerIntent` koristi se na sledeći način:

1. Deklarišite nameru čija je akcija `ACTION_RECOGNIZE_SPEECH`.
2. Nameri dodajte sve potrebne dopunske podatke; morate zadati barem parametar `EXTRA_LANGUAGE_MODEL`. Njegova vrednost može biti `LANGUAGE_MODEL_FREE_FORM` ili `LANGUAGE_MODEL_WEB_SEARCH`.
3. Povratni blok tipa `Bundle` sadrži listu znakovnih nizova koji se poklapaju sa izvornim tekstom. Upotrebite funkciju `data.getStringArrayListExtra` da biste učitali te podatke i pretvorite ih u tip `ArrayList` kako biste mogli dalje da ih obrađujete.

Tako dobijeni podaci prikazuju se na ekranu u obliku natpisa (`TextView`). Glavna aktivnost predstavljena je u listingu 2.14.

Potrebne su i dve dodatne prateće datoteke, **main.xml** i **strings.xml**, u kojima ćete definisati dugme i polje za tekst u kojem se prikazuje rezultat. To se postiže pomoću koda iz listinga 2.10 i 2.12 iz recepta „Pokretanje druge aktivnosti iz događaja“. U datoteci AndroidManifest.xml treba deklarisati samo glavnu aktivnost, koja je ista kao ona u osnovnom receptu „Pokretanje aktivnosti“. Aktivnost `RecognizerIntent` ugrađena je u operativni sistem Android i nema potrebe da je eksplicitno deklarišete da biste je mogli koristiti.

**Listing 2.14** src/com/cookbook/launch_for_result/RecognizerIntent Example.java

```
package com.cookbook.launch_for_result;

import java.util.ArrayList;

import android.App.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.speech.RecognizerIntent;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
```

```
public class RecognizerIntentExample extends Activity {
   private static final int RECOGNIZER_EXAMPLE = 1001;
   private TextView tv;

   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      tv = (TextView) findViewById(R.id. Text_result);

      //obrada događaja dugmeta
      Button startButton = (Button) findViewById(R.id. Trigger);
      startButton.setOnClickListener(new View.OnClickListener() {
         public void onClick(View view) {
            // RecognizerIntent poziva korisnika da govori, a zatim vraća tekst
            Intent intent =
               new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);

            intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
               RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
            intent.putExtra(RecognizerIntent.EXTRA_PROMPT,
            "Say a word or phrase\nand it will show as text");
            startActivityForResult(intent, RECOGNIZER_EXAMPLE);
         }
      });
   }

   @Override
   protected void onActivityResult(int requestCode,
                                   int resultCode, Intent data) {
      //upotrebite naredbu switch ako treba da ispitujete
      //više od jednog koda zahteva
      if (requestCode==RECOGNIZER_EXAMPLE && resultCode==RESULT_OK) {
         // povratni podaci sastoje se od liste poklapanja s govorniim podacima
         ArrayList<String> result =
         data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
         //prikaz na ekranu
         tv.setText(result. ToString());
      }

      super.onActivityResult(requestCode, resultCode, data);
   }
}
```

## Recept: Izrada liste opcija

U aplikacijama često korisniku treba ponuditi listu opcija s koje on bira. To se lako postiže pomoću aktivnosti `ListActivity` (potklasa od `Activity`) i okidanjem događaja koji zavisi od korisnikovog izbora.

Postupak izrade liste opcija sastoji se od sledećih koraka:

1. Napravite klasu koja nasleđuje klasu `ListActivity` umesto klase `Activity`:
```
public class ActivityExample extends ListActivity {
      //ovde dolazi kôd
}
```
2. Napravite niz znakovnog tipa koji će sadržati tekst svih opcija:
```
static final String[] ACTIVITY_CHOICES = new String[] {
         "Action 1",
         "Action 2",
         "Action 3"
};
```
3. Pozovite funkciju `setListAdapter()` s objektom tipa `ArrayAdapter` kojem ćete kao parametre zadati listu opcija i oblik njihovog prikazivanja na ekranu:
```
setListAdapter(new ArrayAdapter<String>(this,
       android.R.layout.simple_list_item_1, ACTIVITY_CHOICES));
getListView().setChoiceMode(ListView.CHOICE_MODE_SINGLE);
getListView().setTextFilterEnabled(true);
```
4. Definišite osluškivač tipa `OnItemClickListener` da biste utvrdili koju je opciju korisnik izabrao i obradite je na odgovarajući način:
```
getListView().setOnItemClickListener(new OnItemClickListener() {
      @Override
      public void onItemClick(AdapterView<?> arg0, View arg1,
                int arg2, long arg3) {
            switch(arg2) {// naredbu switch dopunite potrebnim opcijama
            case 0:
               //kôd za akciju 1
               break;
            case 1:
               //kôd za akciju 2
               break;
            case 2:
               //kôd za akciju 3
               break;
            default: break;
            }
         }
});
```

Opisana tehnika se koristi u sledećem receptu.

## Recept: Upotreba implicitnih namera za pokretanje aktivnosti

Implicitne namere ne definišu eksplicitno koju komponentu treba koristiti. Umesto toga, one pomoću filtra zadaju samo potrebnu funkcionalnost, a operativni sistem Android mora odrediti koja je komponenta najprikladnija za tu namenu. Filtar za namere može određivati akciju, podatke ili kategoriju.

Najčešće se koristi filtar za namere koji određuje akciju, a najuobičajenija akcija je `ACTION_VIEW`. Taj oblik filtra zahteva da zadate i URI (*uniform resource identifier – jedinstven identifikator resursa*) ciljnog resursa s kojeg se korisniku prikazuju podaci. Za dati URI izvršava se akcija koja se smatra najprikladnijom. Na primer, implicitne namere u slučajevima broj 0, 1 i 2 u sledećem primeru sve imaju istu sintaksu ali daju različite rezultate.

Postupak pokretanja aktivnosti pomoću implicitne namere sastoji se od sledećih koraka:

1. Deklarišite nameru za koju zadate odgovarajući filtar (`ACTION_VIEW`, `ACTION_WEB_SEARCH` itd.).
2. Ako je potrebno, nameri pridružite dodatne informacije koje su neophodne za izvršavanje aktivnosti.
3. Prosledite nameru funkciji `startActivity()`.

Postupak za više namera ilustrovan je u listingu 2.15.

**Listing 2.15** src/com/cookbook/implicit_intents/ListActivityExample.java

```
Package com.cookbook.implicit_intents;

import android.App.ListActivity;
import android.App.SearchManager;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.AdapterView.OnItemClickListener;

public class ListActivityExample extends ListActivity {
      static final String[] ACTIVITY_CHOICES = new String[] {
         "Open Website Example",
         "Open Contacts",
         "Open Phone Dialer Example",
         "Search Google Example",
         "Start Voice Command"
      };
      final String searchTerms = "superman";

      protected void onCreate(Bundle savedInstanceState) {
         super.onCreate(savedInstanceState);

         setListAdapter(new ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_1, ACTIVITY_CHOICES));
         getListView().setChoiceMode(ListView.CHOICE_MODE_SINGLE);
         getListView().setTextFilterEnabled(true);
```

```
        getListView().setOnItemClickListener(new OnItemClickListener()
        {
           @Override
           public void onItemClick(AdapterView<?> arg0, View arg1,
                 int arg2, long arg3) {
              switch(arg2) {
              case 0: //pokreće čitač veba i prelazi na zadatu veb lokaciju
                 startActivity(new Intent(Intent. ACTION_VIEW,
                       Uri.parse("http://www. Android.com/")));
                 break;
              case 1: //pokreće aplikaciju za kontakte radi pretraživanja
                      //kontakata
                 startActivity(new Intent(Intent. ACTION_VIEW,
                       Uri.parse("content://contacts/people/")));
                 break;
              case 2: //pokreće birač telefonskih brojeva kojem prosleđuje
                      //zadati broj
                 startActivity(new Intent(Intent. ACTION_VIEW,
                    Uri.parse("tel:12125551212")));
                 break;
              case 3: //traži na Googleu zadati znakovni niz
                 Intent intent= new Intent(Intent. ACTION_WEB_SEARCH );
                 intent.putExtra(SearchManager.QUERY, searchTerms);
                 startActivity(intent);
                 break;
              case 4: //pokreće aplikaciju za izdavanje glasovnih komandi
                 startActivity(new Intent(Intent.ACTION_VOICE_COMMAND));
                 break;
              default: break;
              }
           }
        });
     }
}
```

## Recept: Kako aktivnostima proslediti osnovne tipove podataka

Ponekad je potrebno da pokrenutoj aktivnosti prosledite određene podatke. Katkada pokrenuta aktivnost generiše podatke koje treba da prosledi pozivajućoj aktivnosti. Na primer, završni rezultat postignut u igri treba proslediti na ekran koji prikazuje najbolje rezultate. Podaci se mogu razmenjivati između aktivnosti na sledeće načine:

- Deklarišite odgovarajuću promenljivu u pozivajućoj aktivnosti (na primer, `public int zavrsniRezultat`) i dodelite joj vrednost u pozvanoj aktivnosti (na primer, `PozivajucaAktivnost.zavrsniRezultat=rezultat`).
- Upotrebite objekat tipa `Bundle` (ilustrovano u narednom primeru).

- Upišite podatke u preference iz kojih ćete ih kasnije učitati (opisano u poglavlju 5, „Događaji korisničkog interfejsa“).
- Iskoristite bazu podataka SQLite da biste u nju upisali podatke koje ćete kasnije učitati (opisano u poglavlju 9).

Objekat tipa `Bundle` preslikava podatke tipa `String` u razne tipove definisane u interfejsu Parcelable. Taj objekat ćete napraviti tako što nameri pridružite dodatne podatke. Naredni primer prikazuje prosleđivanje podataka iz glavne aktivnosti u pokrenutu aktivnost, gde se oni menjaju i zatim vraćaju u glavnu aktivnost.

U aktivnosti `StartScreen` deklarisane su dve promenljive (u ovom slučaju tipa `integer` i `String`). Pošto napravimo nameru koja poziva klasu `PlayGame`, te promenljive pridružujemo nameri pomoću metode `putExtra`. Kada pozvana aktivnost vrati rezultat, vrednosti tih promenljivih mogu se učitati pomoću metode `getExtras`. Pozivanje tih metoda prikazano je u listingu 2.16.

**Listing 2.16** src/com/cookbook/passing_data_activities/StartScreen.java

```
package com.cookbook.passing_data_activities;

import android.App.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class StartScreen extends Activity {
   private static final int PLAY_GAME = 1010;
   private TextView tv;
   private int meaningOfLife = 42;
   private String userName = "Douglas Adams";

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      tv = (TextView) findViewById(R.id.startscreen_text);

      //prikazuje početne vrednosti
      tv.setText(userName + ":" + meaningOfLife);

      //obrada događaja dugmeta
      Button startButton = (Button) findViewById(R.id.play_game);
      startButton.setOnClickListener(new View.OnClickListener() {
         public void onClick(View view) {
            startGame();
         }
      });
   }
```

```
   @Override
   protected void onActivityResult(int requestCode,
         int resultCode, Intent data) {
      if (requestCode == PLAY_GAME && resultCode == RESULT_OK) {
         meaningOfLife = data.getExtras().getInt("returnInt");
         userName = data.getExtras().getString("userName");
         //prikazujemo izmene
         tv.setText(userName + ":" + meaningOfLife);
      }
      super.onActivityResult(requestCode, resultCode, data);
   }

   private void startGame() {
      Intent launchGame = new Intent(this, PlayGame.class);

      //prosleđuje podatke pozvanoj aktivnosti
      launchGame.putExtra("meaningOfLife", meaningOfLife);
      launchGame.putExtra("userName", userName);

      startActivityForResult(launchGame, PLAY_GAME);
   }
}
```

Vrednosti promenljivih koje smo prosledili aktivnosti `PlayGame` možemo učitati pomoću metoda `getIntExtra` i `getStringExtra`. Kada se pozvana aktivnost završi i pripremi nameru za povratak u pozivajuću aktivnost, pomoću metode `putExtra` možete proslediti vrednosti nazad u pozivajuću aktivnost. Listing 2.17 prikazuje kako se to radi.

**Listing 2.17** src/com/cookbook/passing_data_activities/PlayGame.java

```
package com.cookbook.passing_data_activities;

import android.App.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class PlayGame extends Activity {
   private TextView tv2;
   int answer;
   String author;

   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.game);
```

```
        tv2 = (TextView) findViewById(R.id.game_text);

        //čitanje podataka prosleđenih ovoj aktivnosti
        //namera koja je pokrenula ovu aktivnost
        Intent i = getIntent();
        //vraća -1 ako vrednost nije inicijalizovana u pozivajućoj aktivnosti
        answer = i.getIntExtra("meaningOfLife", -1);
        //vraća [] ako vrednost nije inicijalizovana u pozivajućoj aktivnosti
        author = i.getStringExtra("userName");

        tv2.setText(author + ":" + answer);

        //menjamo ulazne vrednosti kao primer povratnih vrednosti
        answer = answer - 41;
        author = author + " Jr.";

        //obrada događaja dugmeta
        Button startButton = (Button) findViewById(R.id.end_game);
        startButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {

                //prosleđujemo podatke pozivajućoj aktivnosti
                Intent i = getIntent();
                i.putExtra("returnInt", answer);
                i.putExtra("returnStr", author);
                setResult(RESULT_OK, i);
                finish();
            }
        });
    }
}
```