OSC:ADA

A game of activities

Android Developer Akademija – predavanje 4

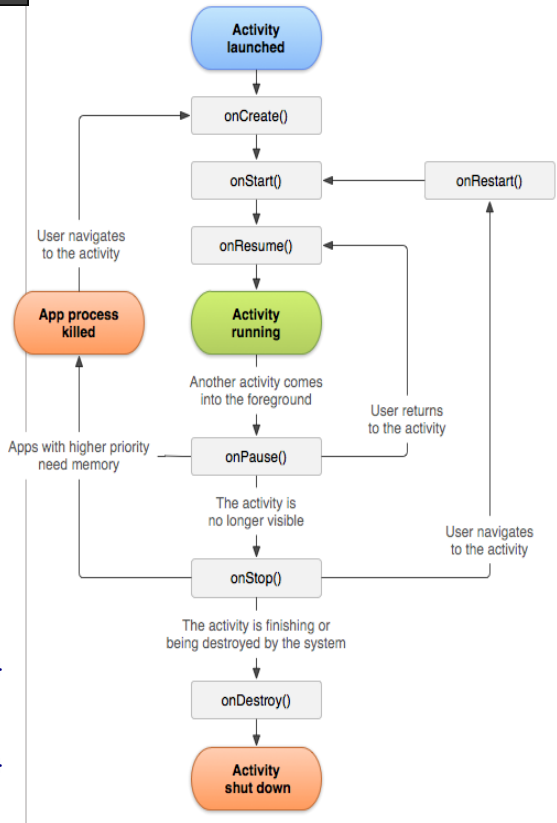
READ SHARE

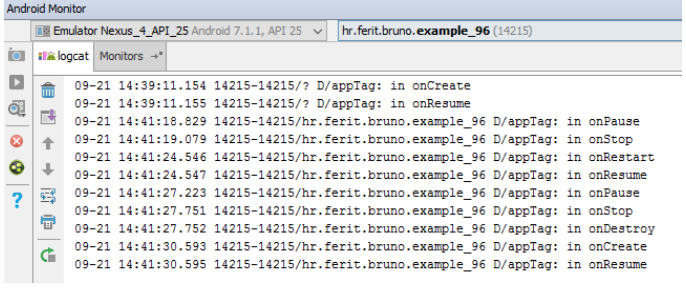
Čekaj, može biti VIŠE activitya?

Životni ciklus activitya

Primjer 96. – MainActivity.java

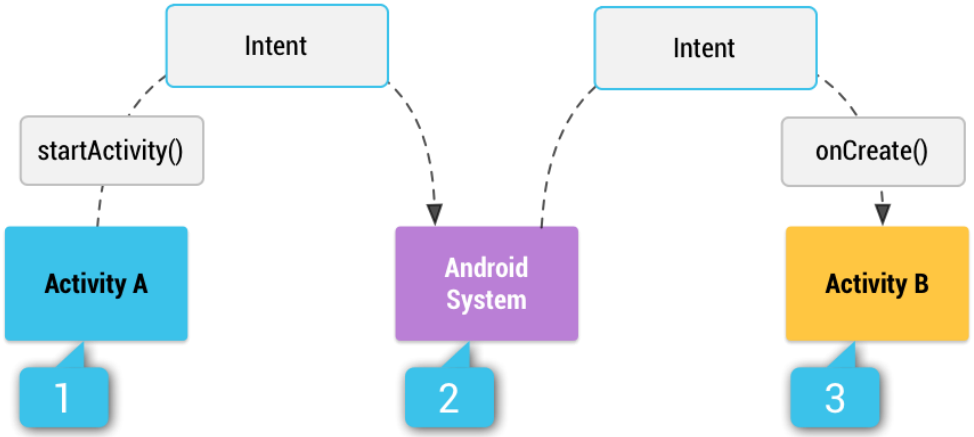
```
public class MainActivity extends Activity {

    private static final String TAG = "appTag";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState); Log.d(TAG, "in onCreate");
        setContentView(R.layout.activity_main);
    }

    @Override
    protected void onResume() {super.onResume(); Log.d(TAG, "in onResume"); }

    @Override
    protected void onPause() { super.onPause(); Log.d(TAG, "in onPause"); }

    @Override
    protected void onStop() { super.onStop(); Log.d(TAG, "in onStop"); }

    @Override
    protected void onDestroy() { super.onDestroy(); Log.d(TAG, "in onDestroy"); }

    @Override
    protected void onRestart() { super.onRestart(); Log.d(TAG, "in onRestart"); }
}
```



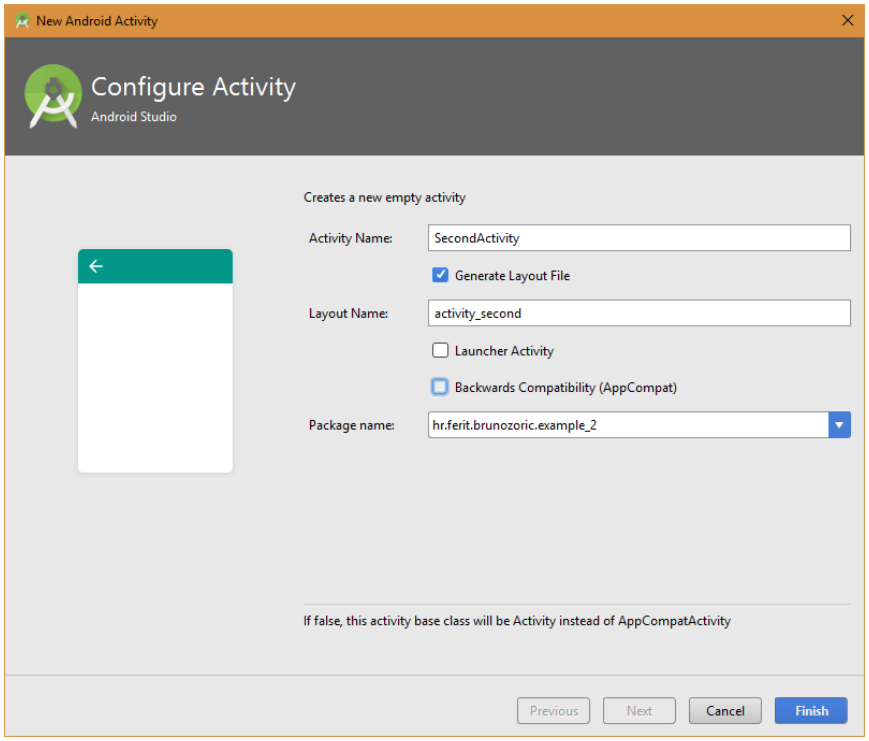


Intent

- Radi se o objektima za slanje poruka kojima komuniciramo s drugim komponentama aplikacije, primjerice drugi Activity pokrećemo korištenjem StartActivity() metode
- Intent omogućuje i prenošenje podataka
- Skup informacija čiji su osnovni atributi akcija (opća radnja koju je potrebno izvršiti, engl. *action*) i data (podaci nad kojima se radi, engl. *data*)
- https://developer.android.com/reference/android/content/Intent.html



- Eksplicitni Intent definira komponentu koju sustav treba pozvati korištenjem klase kao identifikatora
- Ukoliko komponenta nije dio istog paketa, moguće ju je pokrenuti korištenjem punog identifikatora
- Ako se navodi komponenta koju se pokreće, to se radi u parametru „component name" Intent objekta – ovo čini Intent eksplicitnim jer se pokreće isključivo ciljana komponenta
- Potrebno je u projekt dodati drugi Activity kroz izbornik File->New->Other...-> Android Activity, te je nužno odrediti ime i layout novom Activityu. Ovaj Activity automatski će biti dodan u manifest datoteku, a u slučaju da se ručno ubacuje u projekt potrebno ga je i ručno registrirati u manifest datoteci.



- Klikom na gumb u prvom Activityu pokreće se drugi Activity
- Eksplicitno je navedena komponenta koja se pokreće
- Moguće navesti i pri pozivu konstruktora

Primjer – onClick metoda u MainActivity.java

```
@Override
public void onClick(View v) {
            Intent intent = new Intent();
            intent.setClass(this, SecondActivity.class);
            startActivity(intent);
}
```

Prosljeđivanje podataka

- Intent objekt može sadržavati podatke koji se nazivaju *extra* podacima i imaju oblik parova ključ-vrijednost
- Metodom putExtra() moguće je proslijeđivati podatke iz jedne komponente u drugu
- Također je moguće koristiti Bundle objekte u koje se najprije postave svi željeni podaci, a zatim se taj objekt umeće u Intent
- Ako se šalju vlastiti objekti moguće je koristiti sučelje Serializable ili Parcelable u kombinaciji s putExtra() metodom

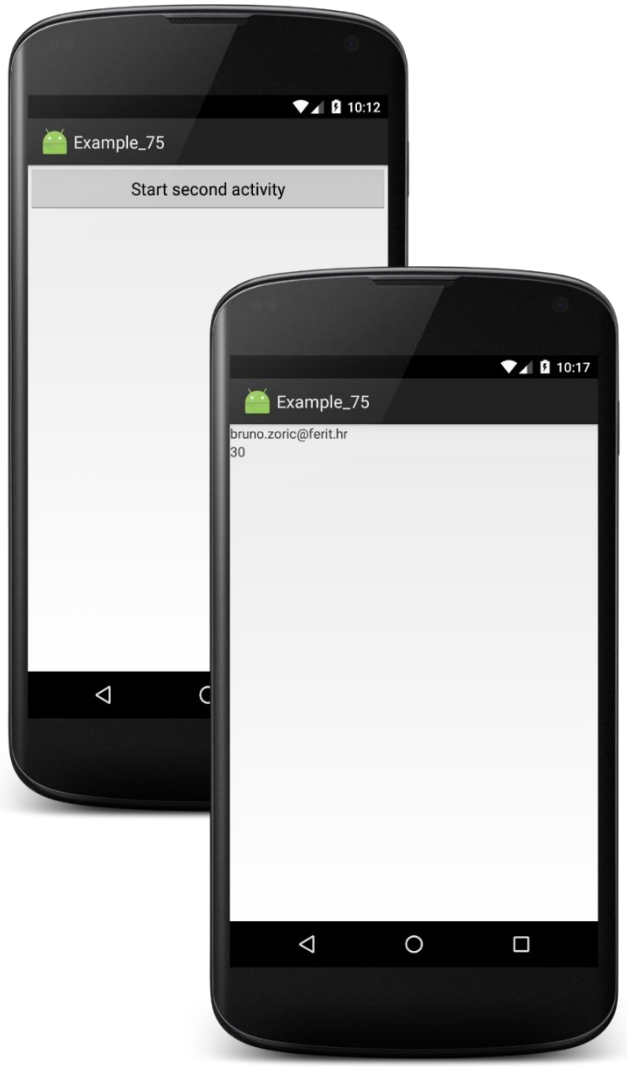
- S jedne strane preopterećenom metodom putExtra() umeće se sadržaj, a s druge strane odgovarajućom get metodom dohvaća
- Uvijek se navodi oznaka kod slanja, a defaultna vrijednost kod dohvaćanja

Primjer – onClick metoda u MainActivity.java

```
@Override
public void onClick(View v) {
            Intent intent = new Intent(this, SecondActivity.class);
            // This can be read from EditText
            String email = "bruno.zoric@ferit.hr";
            int age = 30;
            intent.putExtra(SecondActivity.EMAIL_EXTRA, email);
            intent.putExtra(SecondActivity.AGE_EXTRA, age);

            this.startActivity(intent);
}
```

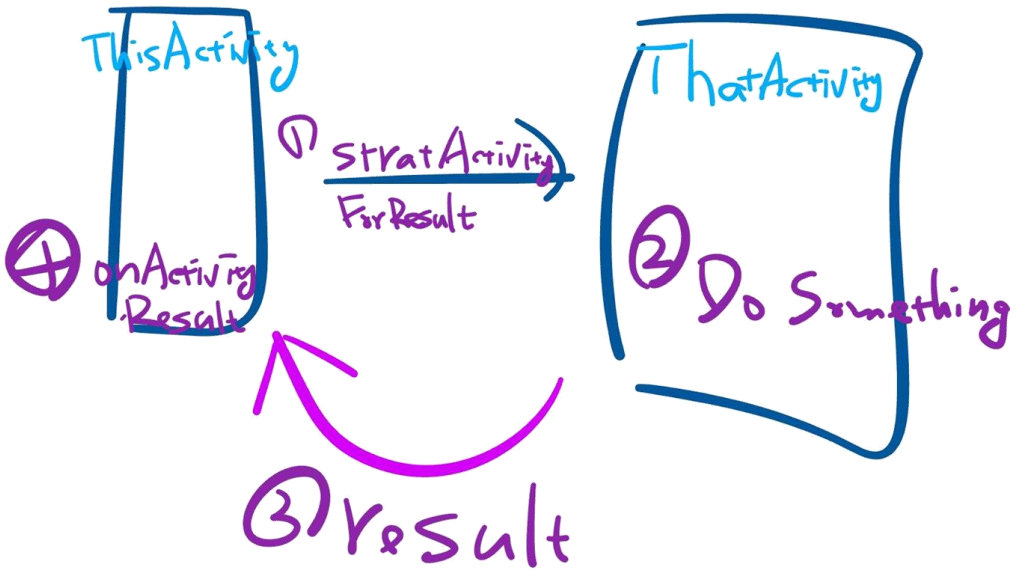
Primjer – dohvaćanje sadržaja iz Intenta

```
private void handleExtraData(Intent startingIntent) {
  if(startingIntent.hasExtra(AGE_EXTRA)){
    int age = startingIntent.getIntExtra(AGE_EXTRA, 0);
    tvAgeDisplay.setText(String.valueOf(age));
  }
  ...
```



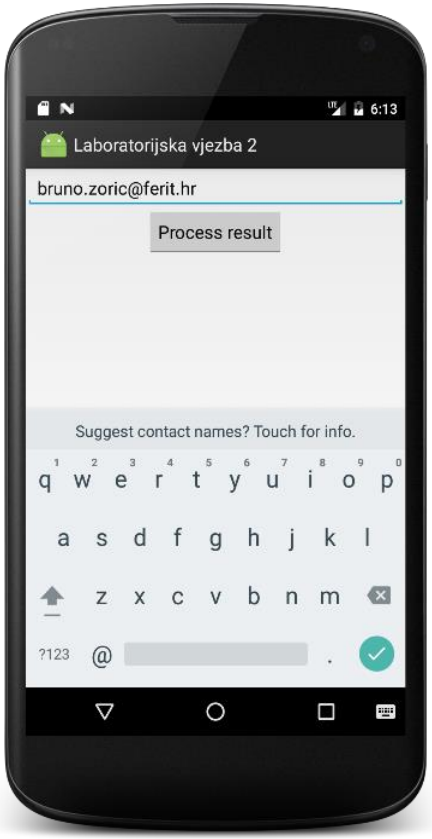
Pokretanje activitya radi rezultata

- Korištenjem StartActivity metode pokreće se drugi Activity unutar aplikacije, no bez mogućnosti interakcije s Activityem koji ga je pozvao
- Ukoliko se želi kreirati novi Activity koji bi služio za prikupljanje podataka, određene izračune i sl., a koji ima potrebu te podatke vratiti natrag pozivajućem Activityu, tada se koristi StartActivityForResult metoda.
- Kako bi se znalo da postoji rezultat kao i na koji točno poziv se odgovara, koriste kodovi zahtjeva i odgovora – Request i Result kodovi
- Moguće je pokrenuti i druge aplikacije ovim načinom, ukoliko one to dopuštaju



Primjer – Slanje zahtjeva i dohvaćanje rezultata

```
@Override
public void onClick(View view) {
    Intent explicitIntent = new Intent(getApplicationContext(), ResultingActivity.class);
    this.startActivityForResult(explicitIntent,KEY_REQUEST_EMAIL);
}
```



Primjer – Slanje zahtjeva i dohvaćanje rezultata

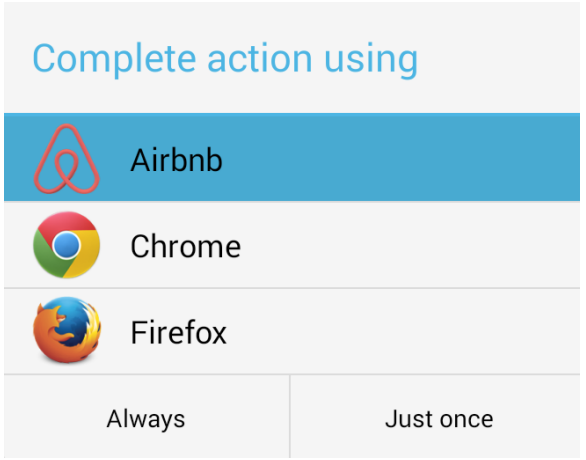
```
@Override
public void onClick(View view) {
    String email = this.etMailInput.getText().toString();
    Intent resultIntent = new Intent();
    resultIntent.putExtra(StartingActivity.KEY_EMAIL, email);
    this.setResult(RESULT_OK, resultIntent);
    this.finish();
}
```

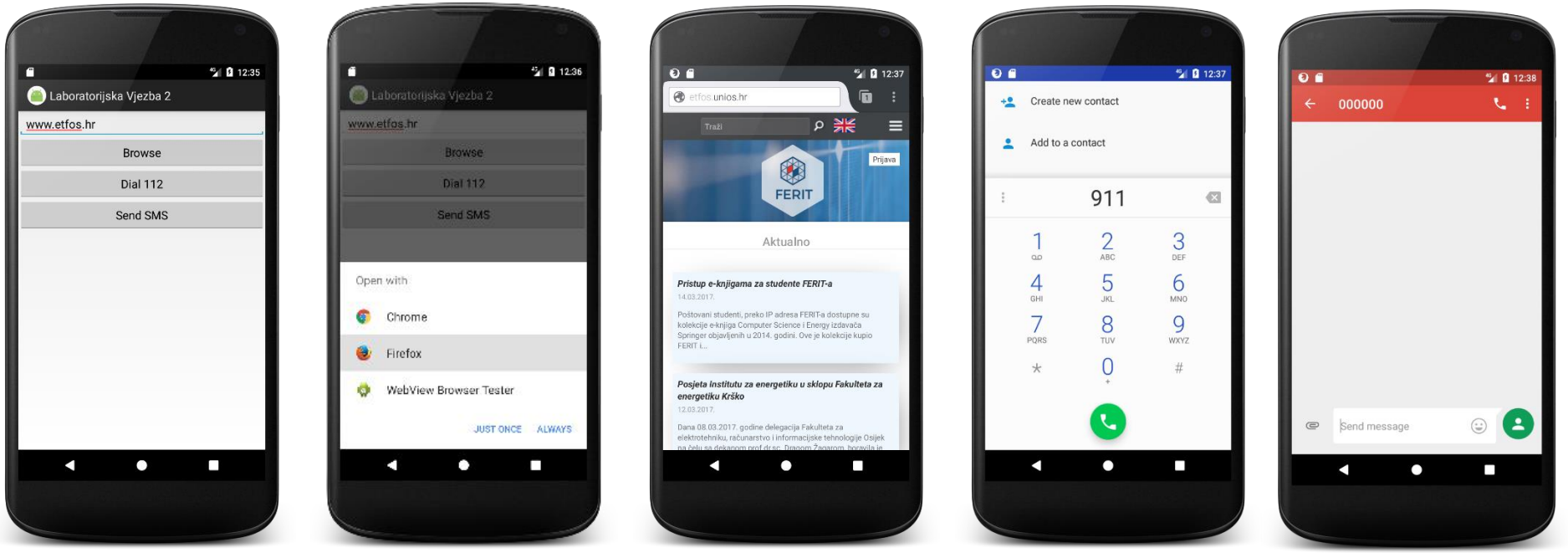
Primjer – Slanje zahtjeva i dohvaćanje rezultata

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    switch (requestCode){
        case KEY_REQUEST_EMAIL:
            if(resultCode == RESULT_OK){
                processEmail(data.getExtras());
            }
            break;
    }
}
```

Implicitni intent

- Implicitni Intent ne definira komponentu koju sustav treba pozvati, već opisuje komponentu kakva je potrebna za obradu nekog zadatka nad određenom vrstom podataka i prepušta odluku Android sustavu, odnosno korisniku u krajnjoj liniji
- Korištenjem implicitnog intenta uobičajeno se pokreću Aktivnosti van aplikacije koja kreira takav Intent
- Kod kreiranja implicitnog Intenta uobičajeno se navode akcija koju je potrebno izvesti te po potrebi podaci na kojima ju je potrebno izvršiti.
- Također je moguće koristiti *putExtra()* metodu za slanje dodatnih podataka što se primjerice i radi kod slanja maila gdje se adrese daju kao extra podaci.
- Android tijekom izvođenja određuje Activity koji je najprikladniji za dani zadatak.
- Kada postoji više Aktivnosti korisnik odabire željenu





Primjer 76. – Stvaranje implicitnog Intenta

```
 case (R.id.bDial):
                implicitIntent = new Intent();
                Uri phoneNumber = Uri.parse("tel:" + EMERGENCY_NUMBER);
                implicitIntent.setAction(Intent.ACTION_DIAL);
                implicitIntent.setData(phoneNumber);
                break;
...
```

- Ne postoji garancija kako u sustavu postoji komponenta koja će izvršiti određenu akciju, a to je moguće provjeriti prije poziva *startActivity()* metodi
- Provjeru je moguće izvršiti pozivom *resolveActivity()* metodi korištenjem kreiranog Intent objekta
- Kao što je prikazano u primjeru, korišten je PackageManager objekt koji čuva informacije o svim aplikacijskim paketima instaliranima u sustavu. Instanca ove klase dobiva se getPackageManager() metodom. Više je moguće vidjeti na:
  - http://developer.android.com/reference/android/content/pm/PackageManager.html

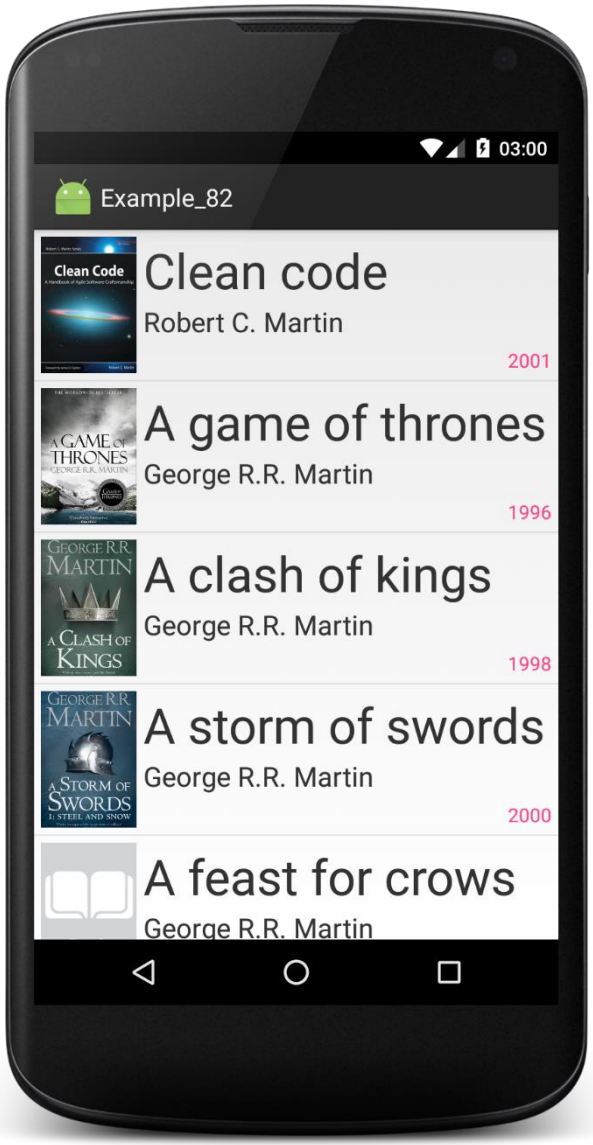
Primjer – Pomoćna metoda za provjeru

```
private boolean canBeCalled(Intent implicitIntent) {
    PackageManager packageManager = this.getPackageManager();
    if(implicitIntent.resolveActivity(packageManager) != null){
        return true;
    }
    return false;
}
```

Liste i pogledi

Listview

- ListView je kontrola koja omogućuje prikaz duge liste elemenata
- Elmenti u listi mogu biti objekti klasa koje koristimo u aplikaciji (npr. Vijest, ToDoItem, Kontakt, Mail ...)
- Najjednostavnija inačica prikazuje String reprezentacije objekata u listi gdje se svaki string prikazuje u vlastitom TextViewu
- Moguće je definirati složenije liste koje koriste složene layoute definirane u XML-u kako bi prikazale objekte u listi na kvalitetniji način.
- Sadržaj koji se prikazuje može biti statičan – definiran kao resurs ili dinamičan odnosno definiran kao neka od podatkovnih struktura
- Za umetanje sadržaja u listu koriste se Adapteri
- Za baratanje klikovima na elemente liste koristi se sučelje onItemClickListener



- Kreira se vlastita klasa kao model podataka te se za nju kreira layout u XML-u koji će se rabiti kao jedan element liste, odnosno kreira se izgled objekta u listi.

Primjer – Book.java

```
public class Book {
    private String mAuthor, mTitle, mCoverUrl;
    private int mYear;

    public Book( String author, String title, int year, String coverUrl)
{
        mAuthor = author;
        mTitle = title;
        mYear = year;
        mCoverUrl = coverUrl;
    }

    public String getAuthor() { return mAuthor; }
    public String getTitle() { return mTitle; }
    public int getYear() { return mYear; }
    public String getCoverUrl() { return mCoverUrl; }
}
```

Primjer – list_item_book.xml

- Važno je napomenuti kako se rabe dva oblikovna obrasca kod uporabe ListView-a, a to su *recycler* i *holder*

- Kako bi se izbjeglo stalno kreiranje View objekata za listu, recikliraju se oni objekti koji izlaze iz vidnog polja – tome služi convertView objekt

- Holder se koristi kako bi se izbjeglo stalno pozivanje findViewById metode, a potrebno je implementirati vlastitu klasu koja drži reference na View objekte elementa liste te njen objekt postaviti kao Tag.

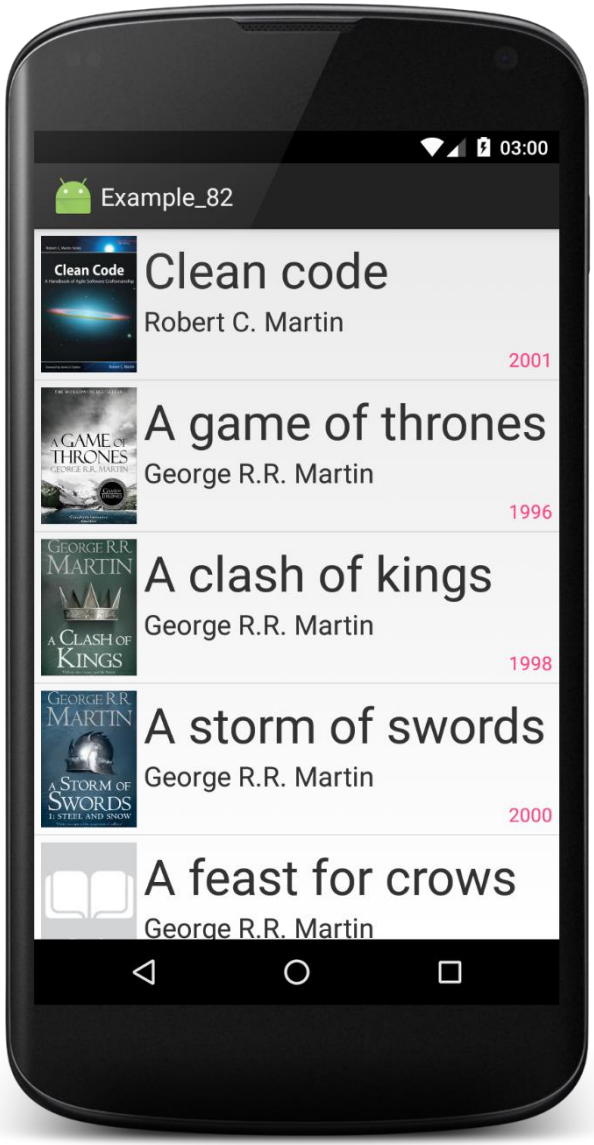
Primjer – BookViewHolder.java

```
static class BookViewHolder {

    @BindView(R.id.ivBookCover) ImageView ivBookCover;
    @BindView(R.id.tvBookTitle) TextView tvBookTitle;
    @BindView(R.id.tvBookAuthor) TextView tvBookAuthor;
    @BindView(R.id.tvBookYear) TextView tvBookYear;

    public BookViewHolder(View view) {
        ButterKnife.bind(this,view);
    }
}
```

Primjer – MainActivity.java

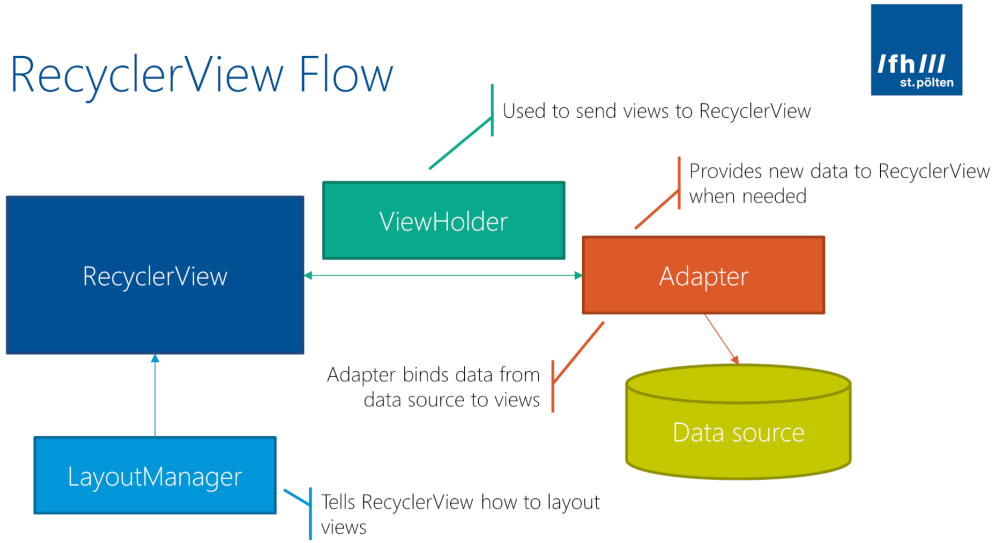
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    ButterKnife.bind(this);
    ArrayList<Book> booksList = loadBooks();
    BookAdapter adapter = new BookAdapter(booksList);
    lvBooks.setAdapter(adapter);
}
```

- Potrebno je generirati listu objekata, kreirati adapter korištenjem dane liste te ga postaviti na ListView



RecyclerView

- Kontrola nastala kao rješenje za ograničenja ListView kontrole
- Zahtjeva nešto više posla oko inicijalnog postavljanja, no nudi značajno veće mogućnosti prilagodbe
- RecyclerView delegira posao rasporeda elemenata (engl. *itema*) LayoutManageru, pa je tako moguće postići primjerice pomicanje sadržaja i horizontalno i vertikalno
- On **zahtijeva** korištenje ViewHolder oblikovnog obrasca
- Omogućuje bolju interakciju s elementima liste, animaciju pojedinih elemenata, prilagodbu razmaknica među elementima



- Potrebno je nešto više podešavanja prilikom korištenja RV-a
  - Zadaje se layout manager, moguće izvesti i vlastiti
  - Zadaje se razdjelnik među elementima
  - Zadaje se animator
  - Kreira se i postavlja adapter

Primjer – MainActivity.java

```
private void setUpRecyclerView() {
    LinearLayoutManager linearLayout =
            new LinearLayoutManager(this, LinearLayoutManager.VERTICAL, false);

    DividerItemDecoration divider =
            new DividerItemDecoration(this, linearLayout.getOrientation());

    BookAdapter adapter = new BookAdapter(
            FakeDatabase.getInstance().getBooks(), mOnBookClickListener);

    rvBooks.setLayoutManager(linearLayout);
    rvBooks.addItemDecoration(divider);
    rvBooks.setAdapter(adapter);
}
```

- Unutar adaptera recikliranje više nije opcija već obaveza
- Isto vrijedi i za korištenje holder obrasca
- Klasa koja predstavlja ViewHolder izvodi se iz RecyclerView.ViewHolder klase
- Klasa koja predstavlja adapter izvodi se iz RecyclerView.Adapter klase koju je potrebno parametrizirati korištenjem konkretne ViewHolder klase (ako se to ne učini, koristit će se osnovna klasa u metodama adaptera)

Primjer – BookAdapter.java

```
class BookAdapter extends RecyclerView.Adapter<BookAdapter.BookViewHolder> {

    private List<Book> mBooks;

    public BookAdapter(List<Book> books) {
        mBooks = new ArrayList<>();
        this.refreshData(books);
    }
// ...
```

- Tri su ključne metode u ovoj priči
- Jedna se poziva kada se view objekt za pojedini element podatkovne strukture po prvi puta kreira
- Druga se poziva kada se view objekt reciklira i potrebno mu je promijeniti sadržaj
- Treća omogućuje dohvaćanje broja elemenata

Primjer – BookAdapter.java

```
@NonNull
@Override
public BookViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    View view = LayoutInflater.from(parent.getContext())
            .inflate(R.layout.item_book, parent, false);
    return new BookViewHolder(view);
}

@Override
public void onBindViewHolder(@NonNull BookViewHolder holder, int position) {
    Book current = mBooks.get(position);
    holder.tv_itemBook_title.setText(current.getTitle());
    holder.getTv_itemBook_author.setText(current.getAuthor());
    Picasso.get()
            .load(current.getImageUrl())
            .centerCrop()
            .fit()
            .error(R.mipmap.ic_launcher)
            .placeholder(R.mipmap.ic_launcher)
            .into(holder.iv_itemBook_cover);
}

@Override
public int getItemCount() { return mBooks.size(); }
```

- View holder je jednostavna klasa kojoj je zadaća držati reference na elemente korisničkog sučelja pojedinog view objekta kojeg RecyclerView prikazuje
- Izbjegava se tako stalno pozivanje findViewById() metode

Primjer – BookAdapter.java

```
public class BookViewHolder extends RecyclerView.ViewHolder {

    @BindView(R.id.iv_itemBook_cover) ImageView iv_itemBook_cover;
    @BindView(R.id.tv_itemBook_title) TextView tv_itemBook_title;
    @BindView(R.id.tv_itemBook_author) TextView tv_itemBook_author;

    public BookViewHolder(View itemView) {
                super(itemView);
                ButterKnife.bind(this, itemView);
    }
}
```

- Ako se dodaje ponašanje na interakciju s elementima RecyclerView kontrole, tada je potrebno nešto više posla nego što bi to bio slučaj s ListView kontrolom.
- U ovom konkretnom slučaju kreirat ćemo sučelje koje omogućuje osluškivanje na pritisak i dugi pritisak na pojedini element unutar RecyclerView-a.

Primjer – BookClickCallback.java

```
public interface BookClickCallback {
    void onClick(Book book);
    boolean onLongClick(Book book);
}
```

- Kreira se objekt anonimne klase koja implementira opisano sučelje
- Izmjeni se adapter klasa tako da u konstruktoru prihvaća opisano sučelje

Primjer – Konkretan listener.java

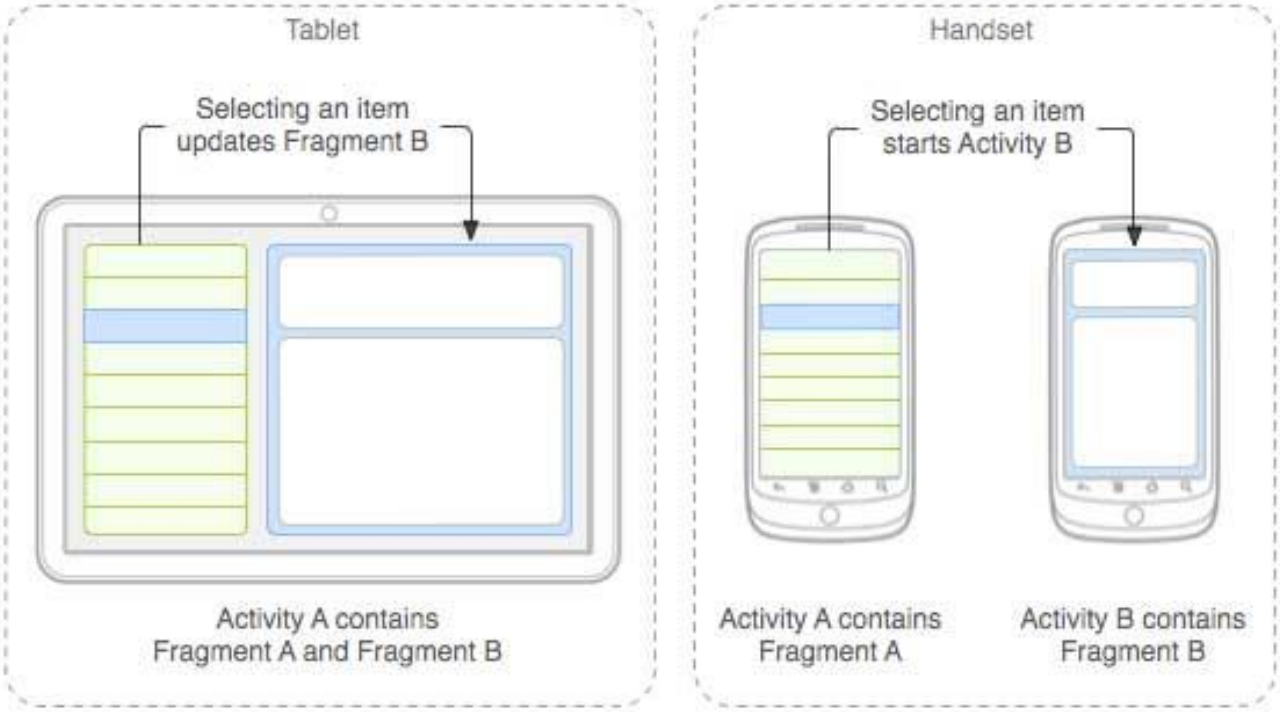
```
private BookClickCallback mOnBookClickListener = new BookClickCallback() {
    @Override
    public void onClick(Book book) {
        String message = book.getTitle();
        Toast.makeText(getApplicationContext(), message, Toast.LENGTH_SHORT).show();
    }

    @Override
    public boolean onLongClick(Book book) {
        FakeDatabase.getInstance().delete(book);
        ((BookAdapter)(rvBooks.getAdapter())).refreshData(FakeDatabase.getInstance().getBooks());
        return true;
    }
};
```

- Moguće je postaviti *listenere* unutar ViewHolder klase na odgovarajuće View objekte
- U ovom slučaju bit će riječ o cijelom View objektu koji predstavlja jedan prikazani element, no relativno je jednostavno prilagoditi ovu priču i postaviti, primjerice, klik samo na sliku

Primjer – Postavljanje listenera.java

```
public class BookViewHolder extends RecyclerView.ViewHolder {

    @BindView(R.id.iv_itemBook_cover) ImageView iv_itemBook_cover;
    @BindView(R.id.tv_itemBook_title) TextView tv_itemBook_title;
    @BindView(R.id.tv_itemBook_author) TextView getTv_itemBook_author;

    BookClickCallback mCallback;

    public BookViewHolder(View itemView, final BookClickCallback callback) {
        super(itemView);
        mCallback = callback;
        ButterKnife.bind(this, itemView);
    }

    @OnClick
    public void onBookClick(){
        mCallback.onClick(mBooks.get(getAdapterPosition()));
    }

    @OnLongClick
    public boolean onBookLongClick(){
        return mCallback.onLongClick(mBooks.get(getAdapterPosition()));
    }
}
```

Fragmenti



- Fragmenti su neovisni moduli koji omogućuju podjelu Activitya na komponente, gdje svaka komponenta ima svoje sučelje i svoj životni ciklus
- Životni ciklusi Fragmenata usko su vezani uz životni ciklus Activitya, ali imaju vlastite Callback metode
- Ne moraju se registrirati u manifest datoteci jer mogu postojati samo unutar Activitya
- Omogućuju izgradnju fleksibilnih i dinamičkih korisničkih sučelja
- Uvedeni u verziji Androida 3.0
- Za prastare verzije podržani su unutar support librarya (nije ista klasa)

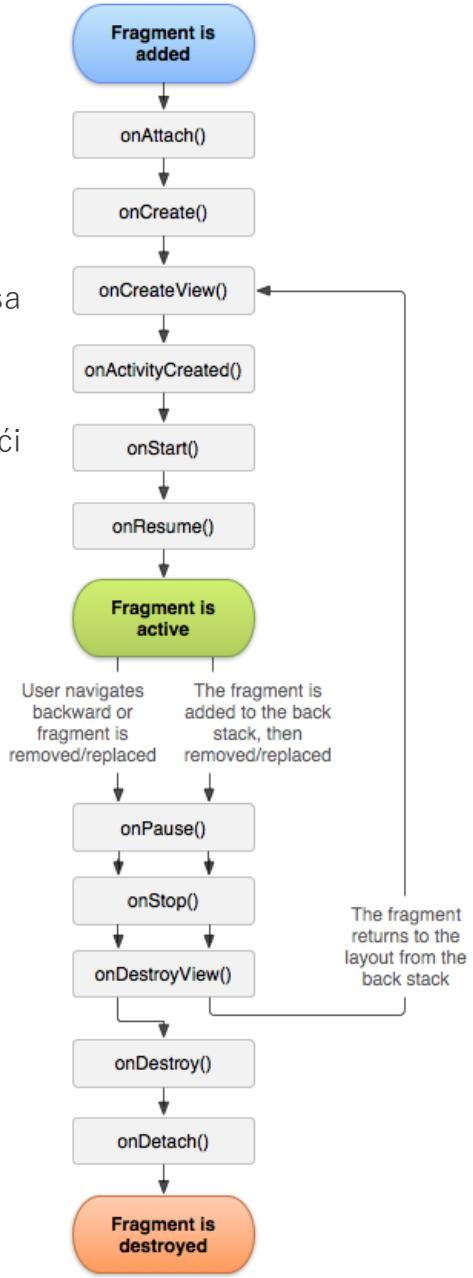
Ključnih pet pitanja (M. Murphy):

What?
Where??
Who?!?
When?!!?
Why?!?!?
OMGOMGOMG, HOW?!?!??

Google I/O 2016: What The Fragment

https://www.youtube.com/watch?v=k3IT-IJ0J98

- **onAttach**()
  - Kada se fragment povezuje s Activityem
- **onCreate**()
  - Kada se instancira objekt klase fragment
- **onCreateView**()
  - Kada je potrebno kreirati hijerarhiju View klasa koje sadržava
- **onActivityCreated**()
  - Kada se kreira objekt klase Activity i pripadajući View objekti
- **onStart**()
  - Kada Fragment postaje vidljiv
- **onResume**()
  - Kada je Fragment u prvom planu
- **onPause**()
  - Kada se fragment miče iz prvog plana
- **onStop**()
  - Kada će fragment prestati s radom
- **onDestroyView**()
  - Prije nego će Fragment biti uništen
- **onDestroy**()
  - Na kraju životnog vijeka
- **onDetach**()
  - Kada se Fragment odvaja od Aktivnosti



- Dodavanje fragmenta moguće je napraviti tako da se on u Activity doda statički ili dinamički.
- Kod statičkog dodavanja samo se u layout dodaje fragment tag s odgovarajućim informacijama
- Kod dinamičkog, na layout se postavlja kontejnerski View u koji su umeće fragment

Primjer – Main activity - layout

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

- Fragmenti se dodaju u kontejner korištenjem FragmentManagera
- Kreira se transakcija u koju se dodaju svi koraci
- Kada je sve spremno poziva se commit naredba koja izvršava opisanu transakciju
- Kod dodavanja fragmenta moguće mu je dodati i tag, odnosno String oznaku po kojoj ga je moguće kasnije pronaći

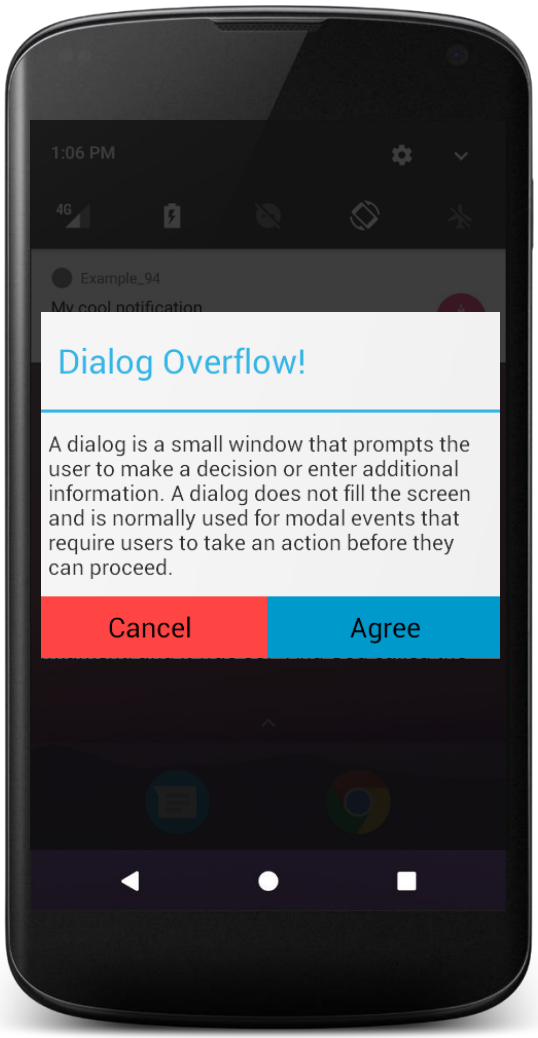
Primjer – Main activity – dodavanje fragmenta

```
private void setUpFragment() {
    FragmentTransaction transaction = mFragmentManager.beginTransaction();
    LoginFragment loginFragment = new LoginFragment();
    loginFragment.setLoginEventListener(new LoginFragment.LoginEventListener() {
        @Override
        public void onNavigateToRegisterClicked() {
            swapLoginForRegister();
        }
    });
    transaction.add(R.id.framelayout_main_container, loginFragment, LOGIN_FRAGMENT_TAG);
    transaction.commit();
}
```

- Zamjena fragmenata obavlja se metodom replace koja se poziva na objektu FragmentTransaction klase
- Dodaje se i novi tag fragmentu koji se ubacuje
- Poziv addToBackStack metodi s null argumentom stavlja fragment iz kojeg se odlazi na „back stack” i omogućuje povratak na njega tipkom za povratak

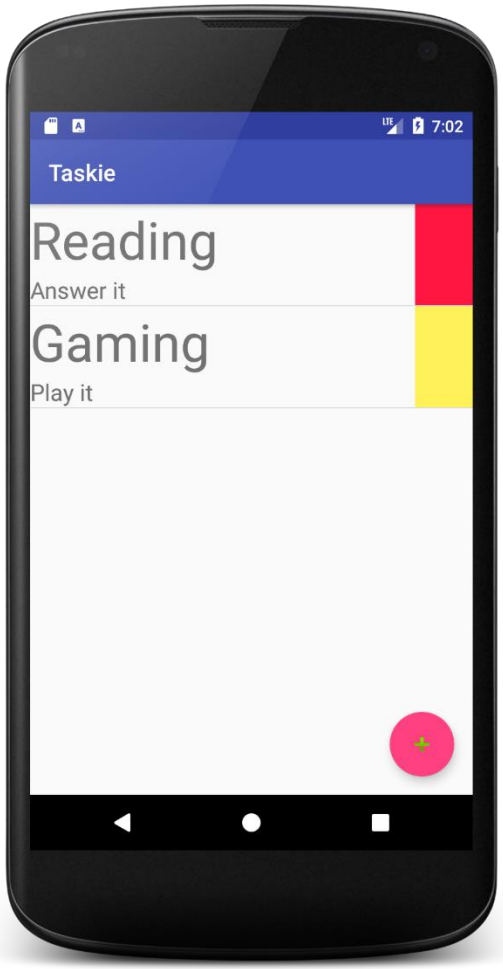
Primjer – Main activity – zamjena fragmenata

```
private void swapLoginForRegister() {
   FragmentTransaction transaction = mFragmentManager.beginTransaction();
   RegisterFragment registerFragment = new RegisterFragment();
   transaction.addToBackStack(null);
   transaction.replace(R.id.framelayout_main_container, registerFragment, REGISTER_FRAGMENT_TAG);
   transaction.commit();
}
```



- Dijalog je mali prozor koji dopušta interakciju s korisnikom i od njega uobičajeno zahtijeva nekakav unos, potvrdu ili odabir
- Dijalozi: http://developer.android.com/guide/topics/ui/dialogs.html
- Osnovna klasa za sve dijaloge je Dialog, ali uobičajeno se instancira jedna od izvedenih klasa.
- Također je uobičajeno da se koristi DialogFragment kao kontejner za dijalog
- Oni su uvijek dio Activitya i imaju fokus sve dok ih korisnik ne zatvori odabirom jedne od mogućnosti
- Moguće je kreirati i vlastite dijaloge i izgled im definrati u XML-u
- Dijalozi bi trebali predstavljati događaje na razini sustava, poput odabira računa, prikaza pogreške i sl., dobra je praksa ograničiti njihovo korištenje kao i ograničiti razinu prilagodbe njihova izgleda
- Dizajn: http://developer.android.com/design/building-blocks/dialogs.html

Homework



Zadatak 1. (Broji li se pod samo jedan?)

- Dodati provjeru unosa prilikom stvaranja novog zadatka.
- Dodati podatak o krajnjem datumu zadatka u klasu Task. Proširiti primjer s predavanja tako da podržava i ovu funkcionalnost. Koristiti odgovarajuću kontrolu za izbor datuma. Datum ne smije biti raniji od trenutnog.
- Omogućiti izmjenu zadatka i njegovu pohranu (update) u „bazi”. Koristiti isti Activity kao i za stvaranje novog zadatka, ali prilikom njegova pokretanja popuniti sučelje starim vrijednostima.
- Dodati toggle button koji predstavlja stanje zadatka. Omogućiti interakciju s gumbom i „završavanje/nezavršavanje” zadatka. (Postaviti osluškivanje na konkretan element na itemView objektu)
- Omogućiti izmjenu prioriteta zadatka klikom na boju/sličicu prioriteta. (Logiku promjene riješiti u klasi Task (Low→Med→High→Low), postaviti osluškivanje na klik na boju/sličicu).
- Kreirajte menu kroz koji omogućite sortiranje taskova u recyclerview kontroli. Sortirati po prioritetu.
- Omogućiti filtriranje prikaza unutar recyclerviewa tako da se mogu prikazati svi zadaci ili samo nezavršeni.

When you play the game of fragments…

… you either win, or you throw an Illegal state exception.

READ SHARE