Mobilne Aplikacije

Nositelj: doc. dr. sc. Nikola Tanković
Izvođač: dr. sc. Robert Šajina
Asistent: mag. inf. Alesandro Žužić

Ustanova: Sveučilište Jurja Dobrile u Puli, Fakultet informatike u Puli

Fakultet informatike u Puli

[6] - Adapters i ViewHolder pattern

Posljednje ažurirano: 11. prosinca 2025.

Sadržaj

Adapteri

Adapter u Androidu predstavlja ključnu komponentu koja povezuje izvor podataka i njihov vizualni prikaz u korisničkom sučelju. Koristi se svaki put kada trebamo prikazati kolekciju elemenata, poput lista, nizova ili objekata, u obliku koji se može skrolati ili dinamički ažurirati.

Adapter ima nekoliko osnovnih zadataka:

- obavještava **RecyclerView** koliko elemenata treba prikazati
- određuje izgled **pojedinog elementa** liste, napuhavanjem layouta (*inflating layout*)
- **puni** svaki element konkretnim podacima (*tekst, slika, ikona, ...*)

Drugim riječima, Android bez adaptera ne zna:

- koliko stavki lista sadrži
- kako izgleda pojedina stavka
- kako se svaki redak puni sadržajem

Adapter tako preuzima ulogu prevoditelja između "suhih" podataka i stvarnog prikaza na ekranu. Omogućuje odvajanje logike prikaza od logike podataka, što olakšava održavanje, proširivanje i ponovno korištenje koda.

Glavne komponente

- **RecyclerView** – prikazuje listu elemenata
- **LayoutManager** – definira raspored (*vertikalni, horizontalni, mrežni*)
- **Adapter** – povezuje podatke s prikazom, kreira **ViewHolder** i puni elemente
- **ViewHolder** – čuva reference na UI elemente unutar itema radi bržeg bindanja
- **ItemDecoration** – dodaje razmake, obrube ili linije između itema
- **ItemAnimator** – animira dodavanje, brisanje i promjene stavki

> Prije se koristio *ListView*, međutim moderni Android danas preferira *RecyclerView*, jer *ListView* ima ograničenja

ViewHolder pattern

`ViewHolder` pattern koristi se kako bi *RecyclerView* radio učinkovitije, posebno kada lista sadrži veliki broj elemenata. Njegova glavna svrha je **izbjegavanje ponovljenih i skupih poziva** `findViewById`, koji bi inače usporavali listu prilikom pomicanja.

Kada se jedan prikaz reciklira, *RecyclerView* ponovno koristi već postojeći View umjesto da kreira novi. Bez *ViewHoldera*, svaki put bi se moralo ispočetka tražiti reference na UI elemente unutar itema (*npr. TextView, ImageView*), što je neefikasno.

ViewHolder rješava taj problem tako da **jednom pronađe sve potrebne view elemente i spremi ih u objekte**, pa adapter može direktno pristupiti tim referencama.

- U `onCreateViewHolder()` se napuhuje (*inflate*) layout za jedan item i inicijalno kreira *ViewHolder* koji čuva sve UI reference
- U `onBindViewHolder()` adapter samo postavlja podatke u te već spremljene view elemente bez dodatnog pretraživanja, što omogućuje brzo i glatko pomicanje

RecyclerView

`RecyclerView` je napredna komponenta za prikaz lista i kolekcija podataka. Dizajniran je tako da radi brzo, učinkovito i fleksibilno, posebno kada radimo s velikim brojem elemenata. Ključni princip je **recikliranje prikaza** – umjesto da za svaki element stvara novi View, *RecyclerView* ponovno koristi već postojeće prikaze koji više nisu na ekranu.

> *RecyclerView* je dio AndroidX biblioteka, stoga ga je potrebno dodati u projekt putem `build.gradle` datoteke prije korištenja ili ga preuzeti unutar *Palette* GUI-a kod XML pregleda.

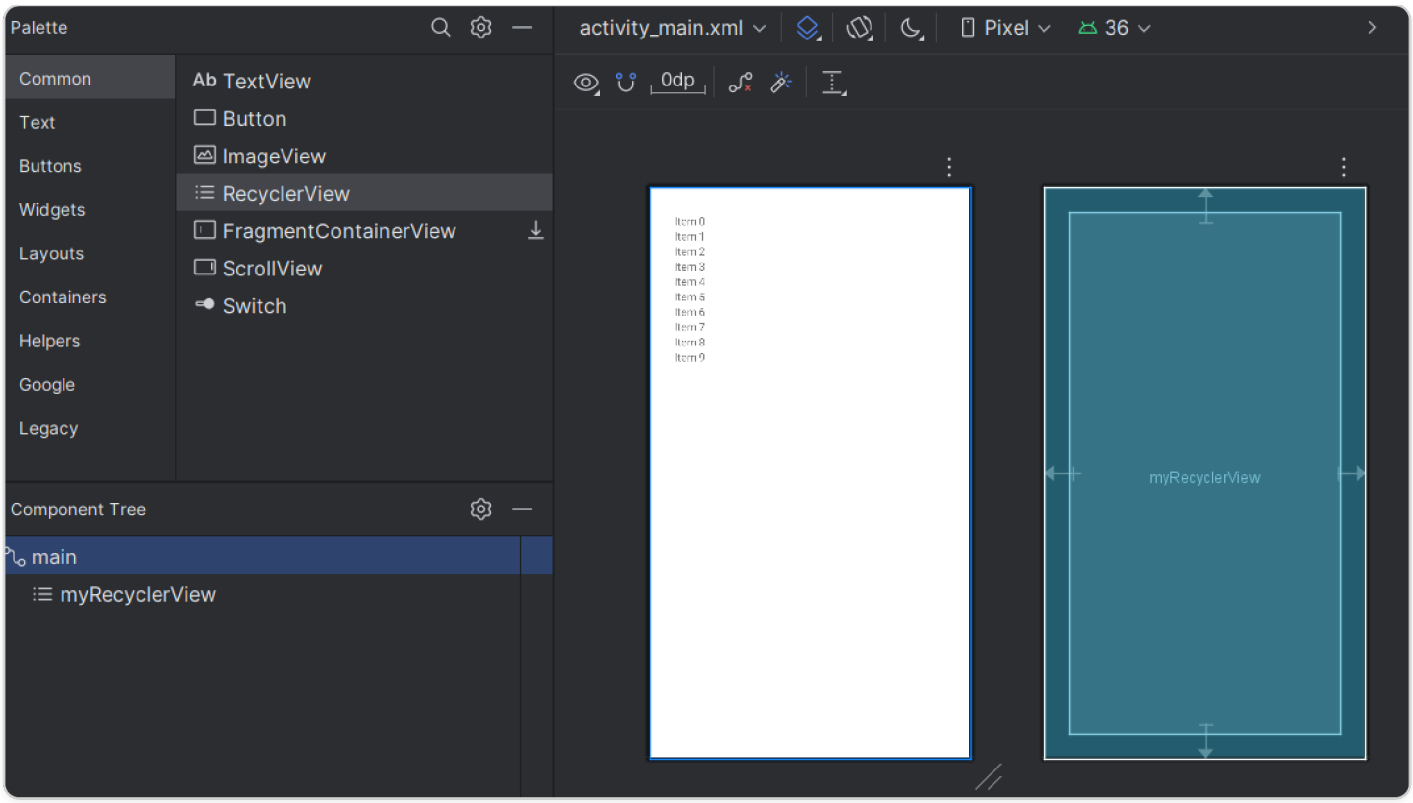
Za razliku od starijeg `ListView`, `RecyclerView` omogućuje:

- potpunu kontrolu nad rasporedom elemenata putem različitih `LayoutManagera`
- dodavanje ukrasnih elemenata i razmaka preko `ItemDecoration`
- animacije pri dodavanju, uklanjanju i promjenama elemenata
- lakše korištenje *ViewHolder* patterna koji je obvezan i poboljšava performanse

Primjer korištenja RecyclerView

Dodat ćemo **RecyclerView** u `activity_main.xml` iz *Palette* GUI-a:

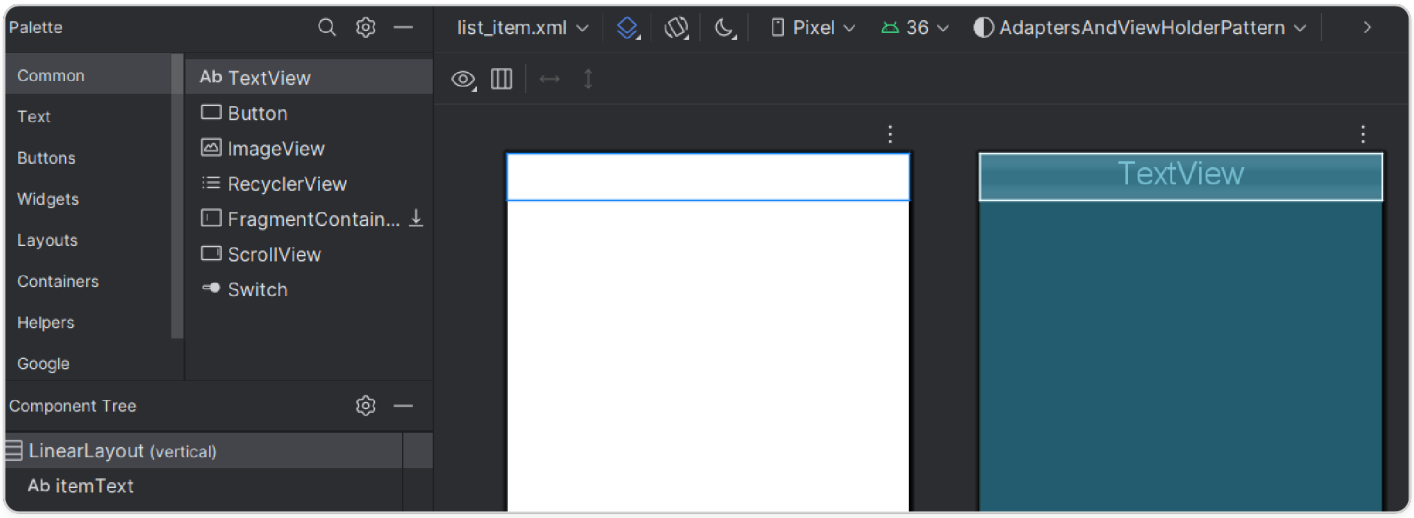
```
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/myRecyclerView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"
        android:layout_marginStart="32dp"
        android:layout_marginTop="32dp"
        android:layout_marginEnd="32dp"
        android:layout_marginBottom="32dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```



Dodavanje RecyclerView

Zatim ćemo napraviti jedan item (`list_item.xml`) u *layout* direktoriju:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <TextView
        android:id="@+id/itemText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="12dp"
        android:textSize="18sp" />
</LinearLayout>
```



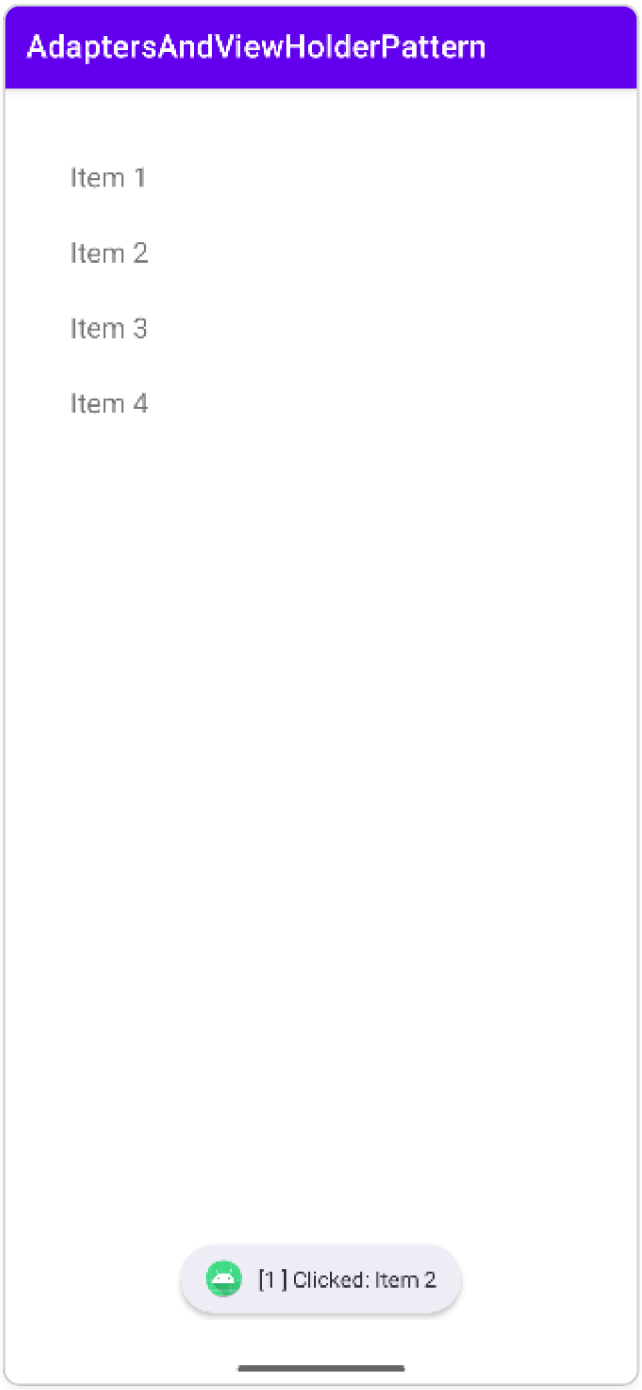
list_item.xml

Sada možemo napraviti **Adapter** klasu `MyAdapter.java`:

```
public class MyAdapter extends RecyclerView.Adapter<MyAdapter.MyViewHolder> {

    private final List<String> items;
    private final OnItemClickListener listener;

    public MyAdapter(List<String> items, OnItemClickListener listener) {
        this.items = items;
        this.listener = listener;
    }
    public interface OnItemClickListener {
        void onItemClick(String item, int position);
    }

    public static class MyViewHolder extends RecyclerView.ViewHolder {
        TextView text;

        public MyViewHolder(@NonNull View itemView) {
            super(itemView);
            text = itemView.findViewById(R.id.itemText);
        }
        void bind(String item, OnItemClickListener listener, int position) {
            text.setText(item);
            itemView.setOnClickListener(v -> {
                if (listener != null) listener.onItemClick(item, position);
            });
        }
    }

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

    @Override
    public void onBindViewHolder(@NonNull MyViewHolder holder, int position) {
        String item =items.get(position);
        holder.bind(item, listener, position);
    }

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

Korištenje u aktivnosti `MainActivity.java`:

```
RecyclerView recycler = findViewById(R.id.myRecyclerView);
MyAdapter adapter = new MyAdapter(
    Arrays.asList("Item 1", "Item 2", "Item 3", "Item 4"),
    (item, pos) -> {
        Toast.makeText(this, "[" + pos + " ] Clicked: " + item,
Toast.LENGTH_SHORT).show();
    }
);
recycler.setAdapter(adapter);
```



RecyclerView primjer

`MyAdapter` klasa je velika tako da ćemo je razdijeliti u sekcije i pobjasniti. Prvo kreiramo klasu adaptera koja nasljeđuje `RecyclerView.Adapter` i koristi vlastiti `ViewHolder`. U adapteru držimo listu podataka i listener za klikove na elemente.

```
public class MyAdapter extends RecyclerView.Adapter<MyAdapter.MyViewHolder> {

    private final List<String> items;
    private final OnItemClickListener listener;

    public MyAdapter(List<String> items, OnItemClickListener listener) {
        this.items = items;
        this.listener = listener;
    }

    public interface OnItemClickListener {
        void onItemClick(String item, int position);
    }

    ...
```

- `items` sadrži sve stavke koje želimo prikazati
- `listener` omogućuje vanjskom kodu da reagira kada korisnik klikne na redak
- Interface `OnItemClickListener` definira metodu koja prima item i njegovu poziciju

Nakon toga definiramo `ViewHolder` koji drži reference na UI elemente reda i može ih popuniti podacima.

```
    ...
    public static class MyViewHolder extends RecyclerView.ViewHolder {
        TextView text;

        public MyViewHolder(@NonNull View itemView) {
            super(itemView);
            text = itemView.findViewById(R.id.itemText);
        }

        void bind(String item, OnItemClickListener listener, int position) {
            text.setText(item);
            itemView.setOnClickListener(v -> {
                if (listener != null) listener.onItemClick(item, position);
            });
        }
    }
    ...
```

- Konstruktor *ViewHoldera* pronalazi `TextView` u layoutu
- Metoda `bind` postavlja tekst i dodaje click listener koji poziva callback iz interfacea

Adapter **mora kreirati** *ViewHolder* kada *RecyclerView* zatraži novi element.

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

- Layout za jedan red napuhujemo (inflate) iz `list_item.xml`
- Vraćamo novi *ViewHolder* koji čuva reference na UI elemente

Kada RecyclerView treba prikazati podatke na određenoj poziciji, adapter bind-a podatke u ViewHolder.

```
    @Override
    public void onBindViewHolder(@NonNull MyViewHolder holder, int position) {
        String item = items.get(position);
        holder.bind(item, listener, position);
    }
```

- Dohvaća se item iz liste prema poziciji
- Poziva se `bind` da se postavi tekst i aktivira click listener

Na kraju postavljamo *getter* ta broj elemenata:

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

- Vraća veličinu liste, što *RecyclerView* koristi za stvaranje i recikliranje redaka

Ažuriranje RecyclerView

Kada stvorimo *adapter*, želimo ga i ažurirati s novom listom. Tako da ćemo dodati dodatnu metodu `setItems()`.

```
    ...
    public void setItems(List<String> newItems) {
        this.items = newItems;
        notifyDataSetChanged();
    }
    ...
```

`notifyDataSetChanged()` koristimo kada se podaci u adapteru promijene i želimo da RecyclerView ponovno nacrta sve elemente.

- Bez ovog poziva RecyclerView neće automatski osvježiti prikaz, pa će stari podaci ostati prikazani
- Ova metoda obavještava adapter da su svi itemi potencijalno promijenjeni i da je potrebno ponovno izračunavanje svih ViewHolder-a
- Koristi se npr. nakon dodavanja, brisanja ili izmjene elemenata u listi podataka

> U praksi se `notifyDataSetChanged()` rijetko koristi jer ponovno bindanje svih itema nije učinkovito. Bolje je koristiti `DiffUtil` ili specifične metode poput `notifyItemInserted()`, `notifyItemRemoved()` ili `notifyItemChanged()` koje osvježavaju samo promijenjene elemente, što poboljšava performanse RecyclerView-a.

Zato ćemo ažurirati `setItems()` da koristi **DiffUtil** tako da ćemo napraviti `MyDiffCallback` klasu:

```
public class MyDiffCallback extends DiffUtil.Callback {
    private final List<String> oldList;
    private final List<String> newList;
    public MyDiffCallback(List<String> oldList, List<String> newList) {
        this.oldList = oldList;
        this.newList = newList;
    }
    @Override
    public int getOldListSize() { return oldList.size(); }
    @Override
    public int getNewListSize() { return newList.size(); }
    @Override
    public boolean areItemsTheSame(int oldItemPosition, int newItemPosition) {
        return areContentsTheSame(oldItemPosition, newItemPosition);
    }
    @Override
    public boolean areContentsTheSame(int oldItemPosition, int newItemPosition) {
        return oldList.get(oldItemPosition).equals(newList.get(newItemPosition));
    }
}
```

Prima novu i staru listu te ih uspoređuje (*pojedine stavke `areItemsTheSame()` i njihova svojstva, tjst. sadržaj `areContentsTheSame()`*) i vraća rezultat usporedbe listi. Sada možemo koristiti ovu klasu za ažuriranje *adaptera*.

```
    ...
    public void setItems(List<String> newItems) {
        DiffUtil.DiffResult diffResult =
                DiffUtil.calculateDiff(new MyDiffCallback(this.items, newItems));
        this.items = newItems;
        diffResult.dispatchUpdatesTo(this);
    }
    ...
```