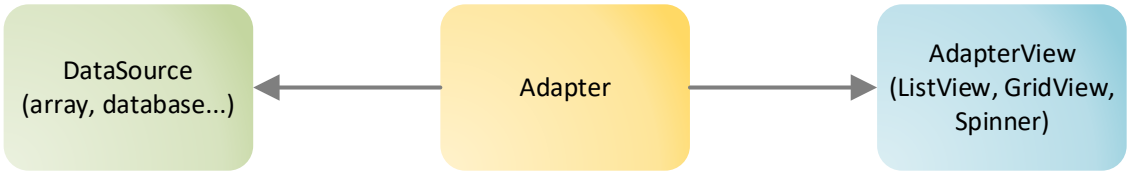
Custom Adapter

Base Adapter

Adapter klasa predstavlja spregu između podataka i *AdapterView* komponente. Njegov zadatak je da pruži model podataka *AdapterView* komponenti i da konvertuje podatke u polja *AdapterView* komponente.



Slika 1 - Komunikacija izvora podataka

Android omogućava i kreiranje Custom Adapter-a, kreiranjem klase koja nasleđuje *BaseAdapter* klasu. To znači da izgled *ListView* komponente u potpunosti možemo prilagoditi potrebama problema.

BaseAdapter je apstraktna klasa i kao takva ne može biti instancirana. Služi kao osnovna klasa pri kreiranju Adaptera, pa je i roditeljska klasa *ArrayAdapter*-a.

Pre kreiranja konkretne implementacije *BaseAdapter*-a, najpre se mora kreirati *layout* datoteka u kojoj se definiše izgled i organizacija jednog elementa (reda) *ListView* komponente, kao i model klasa za elemente unutar *ListView*-a. *Layout* datoteka se smešta u direktorijum **res/layout.**

Model klasa treba da sadrži polja koja odgovaraju elementima koji se nalaze u prethodno kreiranoj layout datoteci koja predstavlja jedan red liste, konstruktor, kao i metode za dobavljanje i postavljanje vrednosti polja.

```
<!--row_layout.xml-->

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

    android:layout_width="match_parent"

    android:orientation="horizontal"

    android:layout_height="match_parent">

    <ImageView

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:id="@+id/image"/>

    <TextView

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:id="@+id/name"/>

</LinearLayout>
```

```
import android.graphics.drawable.Drawable;

public class Character {

    private String mName;

    private Drawable mImage;

    public Character(String name, Drawable drawable) {

        this.mName = name;

        this.mImage = drawable;}

    public String getmName() {

        return mName;}

    public void setmName(String mName) {

        this.mName = mName;}

    public Drawable getmImage() {

        return mImage;}

    public void setmImage(Drawable mImage) {

        this.mImage = mImage;}

}
```

Potom se kreira *Custom Adapter* kreiranjem nove Java klase koja nasleđuje *BaseAdapter*. Nasleđivanjem ove klase, moraju se implementirati sledeće metode:

- `int getCount()` – povratna vrednost je broj elemenata u listi.
- `Object getItem(int position)` – povratna vrednost je objekat/ element liste pronađen na zadatoj poziciji.
- `long getItemId(int position)` – povratna vrednost je jedinstveni identifikator elementa zadatog po poziciji u listi.

Kao polja ove klase potrebno je dodati:

- Context aplikacije
- Izvor podataka za Adapter

Takođe, potrebno je kreirati i konstruktor, kako bi se *Custom Adapter* mogao instancirati.

```
public class CharacterAdapter extends BaseAdapter {

    private Context mContext;

    private ArrayList<Character> mCharacters;

    public CharacterAdapter(Context context) {

        mContext = context;

        mCharacters = new ArrayList<Character>();}

    @Override

    public int getCount() {

        return mCharacters.size();

}

    @Override

    public Object getItem(int position) {

        Object rv = null;

        try {

            rv = mCharacters.get(position);

        } catch (IndexOutOfBoundsException e) {

            e.printStackTrace();

        }

        return rv;

    }

    @Override

    public long getItemId(int position) {

        return position;

    }
```

- `View getView(int position, View convertView, ViewGroup parent)`

Ova metoda dobavlja *View* objekat zadužen za prikaz podataka. View objekat se može kreirati ručno ili posredstvom metode inflate, iz XML datoteke. Pozivom metode *inflate* generiše se nova *View* hijerarhija od prosleđenog *layout* resursa.

ListView komponenta implementira *Recycle* mehanizam. Ovaj mehanizam ima za zadatak da ne kreira uvek novi *View* objekat za svaki podatak već da ponovo upotrebi već kreirane. Na taj način, u slučaju velikog broja podataka, postojaće samo onaj broj *View* objekata koji je moguće prikazati na ekranu. Pri spuštanje liste na dole, ili podizanju liste na gore, *View* objekti se neće uništavati, već će se menjati samo podaci koji oni prikazuju. Na ovaj način se štedi memorija za prikaz podataka. Parametar *position* metode *getView* predstavlja poziciju elementa unutar kolekcije podataka, dok parametar *convertView* sadrži *View* objekat koji bi trebalo kreirati ili menjati. Pre korišćenja ovog parametra, treba proveriti da je adekvatnog tipa i da li je različit od *null*. Kako bi se korišćenje *ListView* komponente optimizovalo, metoda *inflate* bi se trebala pozivati samo kada je ***convertView == null,*** zbog toga što se metoda *getView* poziva svaki put kada lista prikazuje novi red na ekranu. Iz ovog razloga se ova metoda poziva samo kada je je ***convertView == null,*** a svaki sledeći poziv *getView* metode samo treba da ažurira sadržaj ***convertView***-a.

```
public class CharacterAdapter extends BaseAdapter {

   @Override

    public View getView(int position, View convertView, ViewGroup parent){

     View view  = convertView;

     if(view == null) {

           //inflate the layout for each list row

            LayoutInflater inflater = (LayoutInflater) mContext.getSystemService(

                    Context.LAYOUT_INFLATER_SERVICE);

            view = inflater.inflate(R.layout.element_layout, null);

        }

        //get current item to be displayed

        Character character = (Character) getItem(position);

        //get the TextView and ImageView

        TextView name = view.findViewById(R.id.name);

        ImageView image = view.findViewById(R.id.image);

        //set the name and image for Character

        name.setText(character.getmName());

        image.setImageDrawable(character.getmImage());

        return view;

    }
```

Potrebno je instancirati adapter u okviru *Activity*-ja, te ga povezati sa instancom *ListView* komponente pozivom metode *setAdapter()*. Kako bismo adapter napunili podacima, potrebno je kreirati adekvatnu metodu za dodavanje elemenata u adapter, te je pozvati nad instancom adaptera.

```
public void addCharacter(Character character) {

     mCharacters.add(character);

     notifyDataSetChanged();}


//MainActivity.java

CharacterAdapter adapter = new CharacterAdapter(this);

adapter.addCharacter(new Character(getString(R.string.Character1),

                    getDrawable(R.drawable.character1)));

ListView list = findViewById(R.id.lista);

list.setAdapter(adapter);
```

ViewHolder

ViewHolder je dizajn šablon koji omogućava pristupanje *View* objektu svakog elementa, izuzimajući pretraživanje objekta pozivom *findViewById()* metode. Obavljanje ove operacije nije nimalo trivijalno, posebno kada se lista pomera često na gore / na dole pri čemu se poziva *getView* metoda. Na ovaj način postižemo bolje performanse *ListView* komponente. ViewHolder se implementira kao unutrašnja klasa koja sadrži reference na sve *View* objekte jednog reda liste. Čuva se kao komponenta ***tag.*** Korišćenjem ovog pristupa, poziv *findViewById()* metode će se obaviti samo kada se layout prvi put kreira.

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
   View view  = convertView;
   if(view == null) {
           LayoutInflater inflater = (LayoutInflater) mContext.getSystemService(
                    Context.LAYOUT_INFLATER_SERVICE);
           view = inflater.inflate(R.layout.element_layout, null);
            ViewHolder holder = new ViewHolder();
            holder.image = view.findViewById(R.id.image);
            holder.name = view.findViewById(R.id.name);
            view.setTag(holder);
        }
        Character character = (Character) getItem(position);
        ViewHolder holder = (ViewHolder) view.getTag();
        holder.name.setText(character.getmName());
        holder.image.setImageDrawable(character.getmImage());

        return view;
    }
    private class ViewHolder {
        public ImageView image = null;
        public TextView name = null;
    }
```

Zadatak za vežbanje

1. Kreirati nov Android Studio projekat „Imenik“.
2. Napraviti novu .xml datoteku sa opisom layout-a jednog reda elementa liste. Ovaj element liste treba da sadrži ime i prezime kontakta, kao i sliku i broj telefona.
3. Kreirati odgovarajuću model klasu, Kontakt.java.
4. Napraviti KontaktAdapter.java Adapter koji nasleđuje BaseAdapter i implementirati odgovarajuće metode.
5. Demonstrirati rad KontaktAdapter-a instanciranjem u okviru MainActivity komponente i uvezivanjem sa listom kreiranom u activity_main.xml. Dodati barem 10 kontakata u listu.
6. Brisanje kontakta vršiti dugim pritiskom na element liste.