UNIVERSITÀ DEGLI STUDI DE L'AQUILA

disim

Mobile Programming Laboratory

ANDROID
Containers

Teachers

Ing. Tarquini Francesco, Ph.D

Ph.D in Computer Science Engineering

francesco.tarquini@univaq.it

Ing. D’Errico Leonardo

Ph.D Student in Computer Science Engineering

leonardo.derrico@graduate.univaq.it

Teaching Materials

Available on MOODLE platform

http://www.didattica.univaq.it

Google Drive Repository

https://drive.google.com/drive/folders/1ISqZfn0i9Ub3eWNXbvW00rd0hD9ya8OL?usp=sharing

Topics

- The Adapter
- ListView
  - GridView
  - ExpandableListView
- RecyclerView

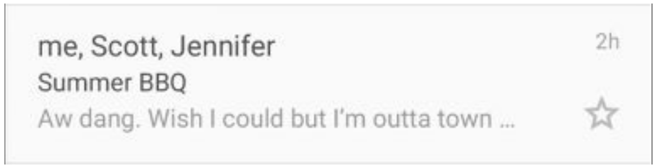
The Adapter

If the developer designs a list of items in own application, he have to develop a container:

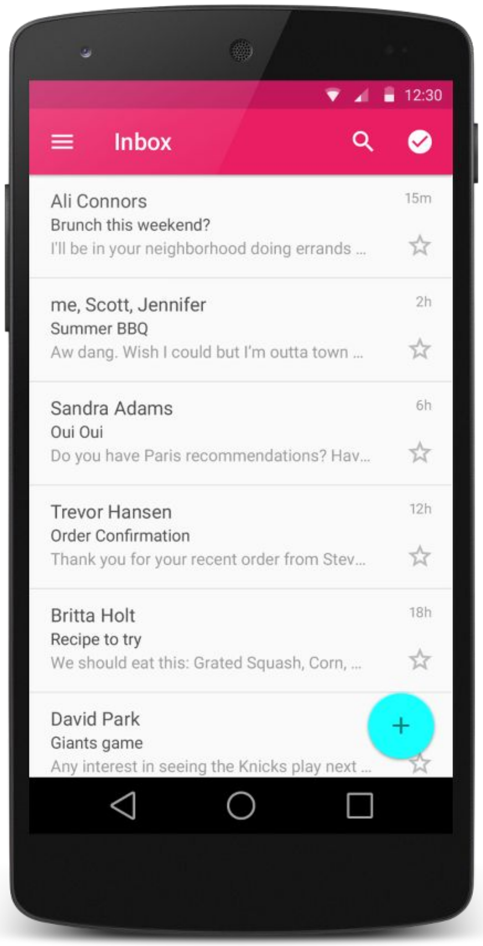
- ListView
- GridView
- RecyclerView

Every container item is an **adapter view** that does not know the details, such as type and contents, of the views it contains.

The Adapter is the manager of the data and views displayed inside the list.



me, Scott, Jennifer 2h
Summer BBQ
Aw dang. Wish I could but I'm outta town ...



12:30

Inbox

Ali Connors 15m
Brunch this weekend?
I'll be in your neighborhood doing errands ...

me, Scott, Jennifer 2h
Summer BBQ
Aw dang. Wish I could but I'm outta town ...

Sandra Adams 6h
Oui Oui
Do you have Paris recommendations? Hav...

Trevor Hansen 12h
Order Confirmation
Thank you for your recent order from Stev...

Britta Holt 18h
Recipe to try
We should eat this: Grated Squash, Corn, ...

David Park
Giants game
Any interest in seeing the Knicks play next ...

The Adapter

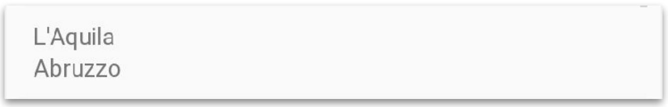
The developer, to implement an Adapter, has to create the XML layout of the single item of the list.

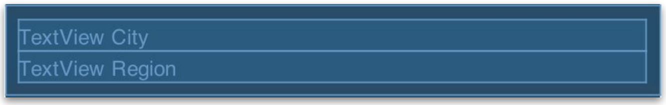
If you want to create the layout on top right, you need of a LinearLayout to align the two TextView vertically.

In the bottom right there is the XML code to design the layout on top.

The Java implementation of the Adapter depends on the list item used.

Be careful! In both the TextView the identifier was defined.





```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:padding="8dp"
  android:background="#EEE"
  android:orientation="vertical">

  <TextView
     android:id="@+id/city"
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:text="TextView City"/>

  <TextView
     android:id="@+id/region"
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:text="TextView Region"/>

</LinearLayout>
```

ListView

ListView is included in Android since v1.0.

This container is very old and requires of a specific pattern to work well, named ViewHolder.

The developer, to implements a ListView, has to add the corresponding item to the layout XML of the Activity (like the code to right).

In the Java Activity class, the developer has to get back the ListView from XML, to link it to a specific Java instance.

Activity give us a method to get back the item from XML to Java: **findViewById()**

```
ListView listView = findViewById(R.id.list);
```

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.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:layout_width="match_parent"
  android:layout_height="match_parent">

  <ListView
     android:id="@+id/list"
     android:layout_width="0dp"
     android:layout_height="0dp"
     app:layout_constraintBottom_toBottomOf="parent"
     app:layout_constraintEnd_toEndOf="parent"
     app:layout_constraintStart_toStartOf="parent"
     app:layout_constraintTop_toTopOf="parent"
     tools:listitem="@layout/adapter" />

</android.support.constraint.ConstraintLayout>
```

ListView

Now the developer can implement the Java Adapter.

In this example we extends the BaseAdapter class to create the our adapter.

BaseAdapter is an abstract class, thus we must implement the abstract methods.

- **getCount()**: the length of the data
- **getItem()**: return the item in specific position
- **getItemId()**: used when the data came from database
- **getView()**: manages the layout of the single item

```
public class Adapter extends BaseAdapter {

        @Override
        public int getCount() {
                return 0;
        }

        @Override
        public City getItem(int position) {
                return null;
        }

        @Override
        public long getItemId(int position) {
                return 0;
        }

        @Override
        public View getView(int position, View convertView,
ViewGroup parent) {
                return null;
        }
}
```

ListView

We add, to start, a constructor of the class, where pass in input the data to display on screen.

With these data in input we can complete some abstract methods.

```
public class Adapter extends BaseAdapter {

        private City[] data;

        public Adapter(City[] cities){
                this.data = cities;
        }

        @Override
        public int getCount() {
                return this.data.length;
        }

        @Override
        public City getItem(int position) {
                return this.data[position];
        }

        @Override
        public long getItemId(int position) {
                return 0;
        }

        @Override
         public View getView(int position, View convertView,
ViewGroup parent) {  ...  }

}
```

ListView

We has to implement the **ViewHolder** pattern to complete the code inside the getView methods.

A ViewHolder object stores each of the component views inside the tag field of the Layout, so we can immediately access them without the need to look them up repeatedly.

We link the XML layout of the adapter to a generic View by the method **inflate** of the **LayoutInflater** class.

```
public class Adapter extends BaseAdapter {
        ...

  @Override
  public View getView(int position, View convertView, ViewGroup
parent) {
      ViewHolder holder;
      if(convertView == null) {
          convertView = LayoutInflater.from(convertView.getContext())
                .inflate(R.layout.adapter, parent, false);

          holder = new ViewHolder();
         holder.city = convertView.findViewById(R.id.city);
         holder.region = convertView.findViewById(R.id.region);
         convertView.setTag(holder);
      } else {
         holder = (ViewHolder) convertView.getTag();
      }

      City city = getItem(position);
      holder.city.setText(city.getCity());
      holder.region.setText(city.getRegion());
      return convertView;
  }

  private class ViewHolder {
      TextView city;
      TextView region;
  }
}
```

ListView

In the end we can set the adapter to our ListView define in Java Activity class.

The example shows how to implement the click on every items of the list.

```
ListView listView = findViewById(R.id.list);

Adapter adapter = new Adapter(cities);
listView.setAdapter(adapter);

listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
  @Override
  public void onItemClick(AdapterView<?> parent, View view, int position, long id) {

      City city = (City) parent.getItemAtPosition(position);
      System.out.println("Clicked " + city.getCity() + " city");
  }
});
```

ListView

Other containers exist in Android SDK.

GridView displays the data as a grid but the adapter can be the same of the ListView.

ExpandableListView display the data collected in groups. In this case the adapter is a bit more complex of the BaseAdapter because it manages the group view too.

Spinner is a view that displays one child at a time and lets the user pick among them.

RecyclerView

Since Android v5.0 Lollipop, Google introduced a new container to replace ListView and GridView.

In order that the developer can design an application also to Android version older than 5.0, Google introduced a collection on support libraries. These libraries allow the last and new Android components to all last versions of Android (at least 4.0).

RecyclerView obligates the developer to use the ViewHolder pattern.

If the developer want use the RecyclerView must add the relative dependency in build.gradle (module app) file:

```
dependencies {
   implementation fileTree(dir: 'libs', include: ['*.jar'])
   implementation 'com.android.support:appcompat-v7:27.1.1'
   implementation 'com.android.support.constraint:constraint-layout:1.1.2'
   ...
   implementation 'com.android.support:recyclerview-v7:27.1.1'
}
```

RecyclerView

In this case the Adapter class extends RecyclerView.Adapter class.

The parent class is parametric and its value is a class that extends the RecyclerView.ViewHolder class.

The RecyclerView.Adapter is abstract, so we have to implement the abstract methods.

getItemCount(): return the length of the data

onCreateViewHolder(): return the ViewHolder linked to specific layout by LayoutInflater

onBindViewHolder(): using to set the ViewHolder fields

```
public class Adapter extends
RecyclerView.Adapter<Adapter.ViewHolder> {

  @Override
  public int getItemCount() {
     return 0;
  }

  @NonNull
  @Override
  public ViewHolder onCreateViewHolder(@NonNull ViewGroup
parent, int viewType) {
     return null;
  }

  @Override
  public void onBindViewHolder(@NonNull ViewHolder holder,
int position) {
  }

  …

}
```

RecyclerView

The developer has to create the constructor of the class.

Then he has to create a ViewHolder class extends RecyclerView.ViewHolder.

```
public class Adapter extends
RecyclerView.Adapter<Adapter.ViewHolder> {

  private City[] data;

  public Adapter(City[] cities){
     this.data = cities;
  }

  ...

  class ViewHolder extends RecyclerView.ViewHolder {

     TextView city;
     TextView region;

     public ViewHolder(View itemView) {
        super(itemView);

        city = itemView.findViewById(R.id.city);
        region = itemView.findViewById(R.id.region);
     }
  }
}
```

RecyclerView

Now he can implements the abstract methods.

```
@Override
public int getItemCount() {
   return this.data.length;
}

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

@Override
public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
   City city = this.data[position];
   holder.city.setText(city.getCity());
   holder.region.setText(city.getRegion());
}
```

RecyclerView

Both in the layout XML and in the java Code of the Activity, the developer must use the RecyclerView object.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.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:layout_width="match_parent"
  android:layout_height="match_parent">

  <android.support.v7.widget.RecyclerView
     android:id="@+id/list"
     android:layout_width="0dp"
     android:layout_height="0dp"
     app:layout_constraintBottom_toBottomOf="parent"
     app:layout_constraintEnd_toEndOf="parent"
     app:layout_constraintStart_toStartOf="parent"
     app:layout_constraintTop_toTopOf="parent"
     tools:listitem="@layout/adapter" />

</android.support.constraint.ConstraintLayout>
```

```
RecyclerView recyclerView = findViewById(R.id.list);
recyclerView.setLayoutManager(new
LinearLayoutManager(getApplicationContext()));


Adapter adapter = new Adapter(cities);
recyclerView.setAdapter(adapter);
```

RecyclerView

Be Careful! Using the RecyclerView not exists the **setOnItemClickListener** method

Thus if the developer wants implement the click performance on the item, he has to add the **OnClickListener** on View instance of the ViewHolder.

```
class ViewHolder extends RecyclerView.ViewHolder {

    TextView city;
    TextView region;

    public ViewHolder(View itemView) {
        super(itemView);

        city = itemView.findViewById(R.id.city);
        region = itemView.findViewById(R.id.region);

        itemView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                System.out.println("Clicked " + data[getAdapterPosition()] + " city");
            }
        });
    }
}
```