# CMPUT 301
# Lab 3 Presentation

## Overview

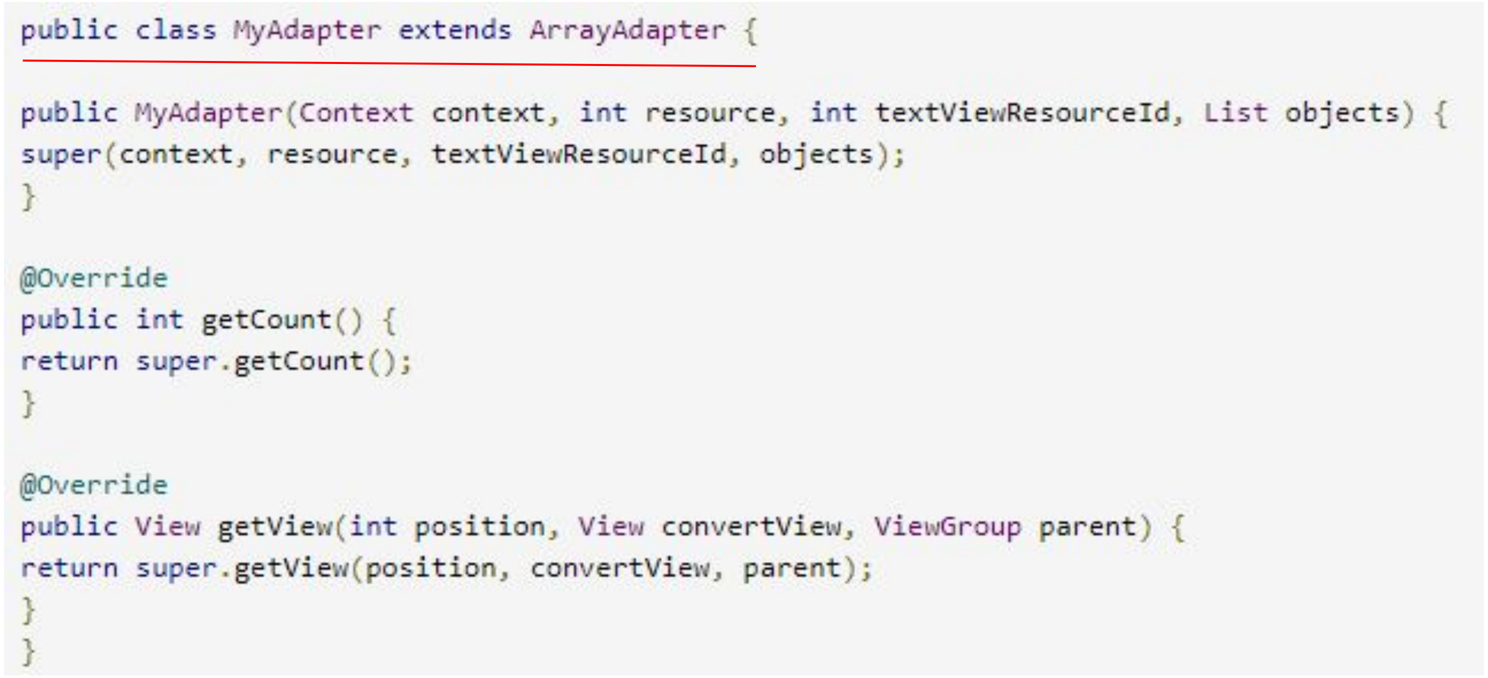
Custom Array Adapter

Fragments

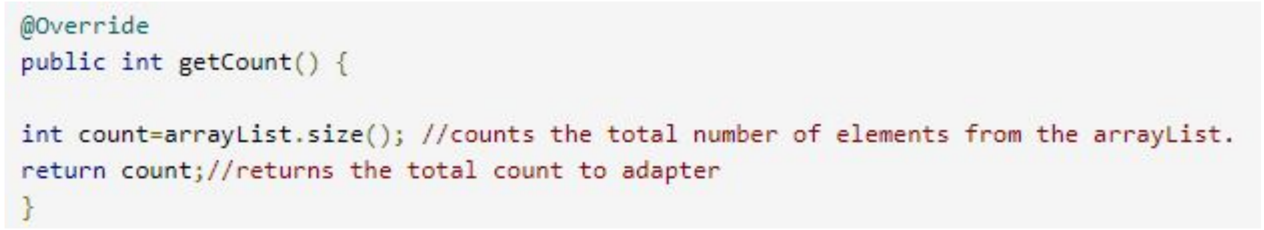
Java conventions

Custom Arrayadapter Lab demo

## More than one item to display? Customize ArrayAdapter!

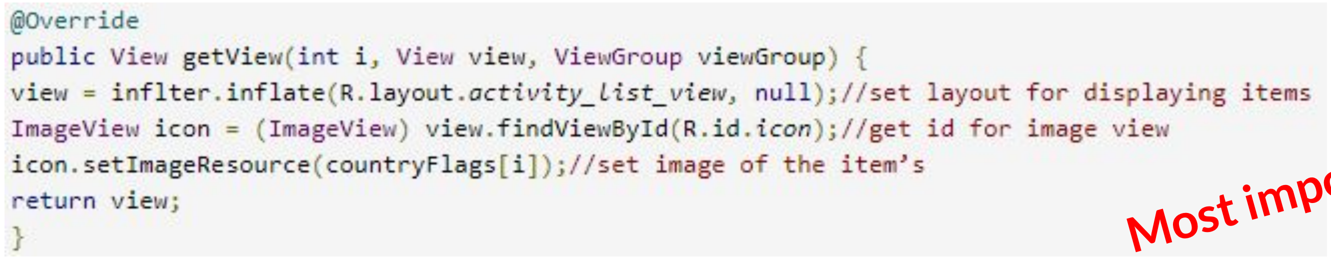


```
public class MyAdapter extends ArrayAdapter {

public MyAdapter(Context context, int resource, int textViewResourceId, List objects) {
super(context, resource, textViewResourceId, objects);
}

@Override
public int getCount() {
return super.getCount();
}

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



```java
@Override
public int getCount() {

int count=arrayList.size(); //counts the total number of elements from the arrayList.
return count;//returns the total count to adapter
}
```

The getCount() function returns the total number of items to be displayed in a list. It counts the value from arraylist size or an array's length. For example if we have an list of elements in a array list and we have to display the items in a list view then we can count the total number of elements using the size function and then that integer value is returned by the function getCount() as shown.

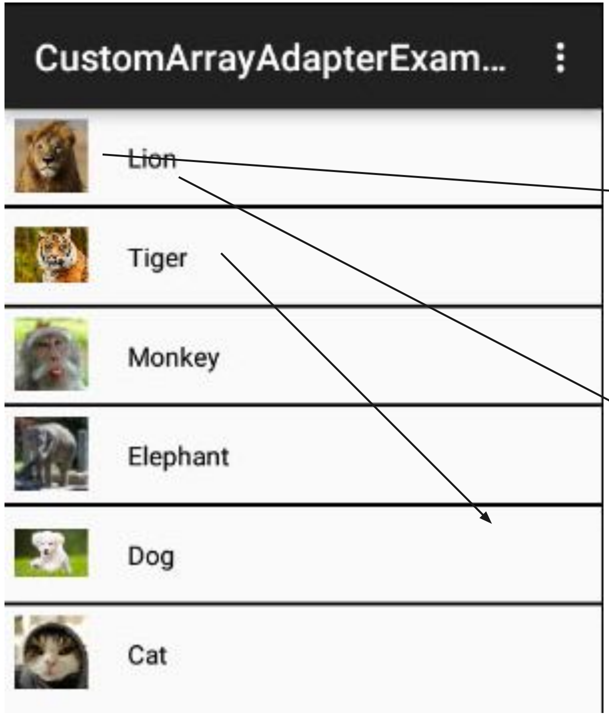


```
@Override
public View getView(int i, View view, ViewGroup viewGroup) {
view = inflter.inflate(R.layout.activity_list_view, null);//set layout for displaying items
ImageView icon = (ImageView) view.findViewById(R.id.icon);//get id for image view
icon.setImageResource(countryFlags[i]);//set image of the item’s
return view;
}
```

Most important function!

This function is automatically called when the list item view is ready to be displayed or about to be displayed. In this function we set the layout for list items using LayoutInflater class and then add the data to the views like ImageView, TextView etc.

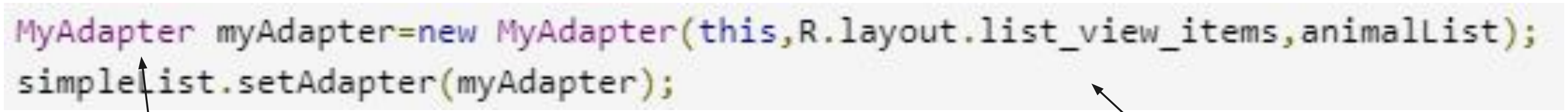
Above is the getView function’s example code with explanation included in which we set the layout using LayoutInflater and then get the view’s id and implement them.



```xml
<?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:orientation="horizontal">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:padding="5dp"
        android:src="@drawable/ic_launcher" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:padding="@dimen/activity_horizontal_margin"
        android:text="Demo"
        android:textColor="#000" />
</LinearLayout>
```

You don’t need image for your assignment. However, it would be really cool to have images for your project!

## The way to use customized adapter is really similar to arrayadapter.

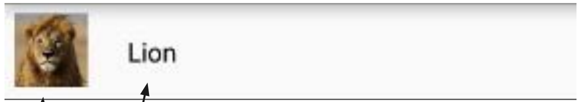


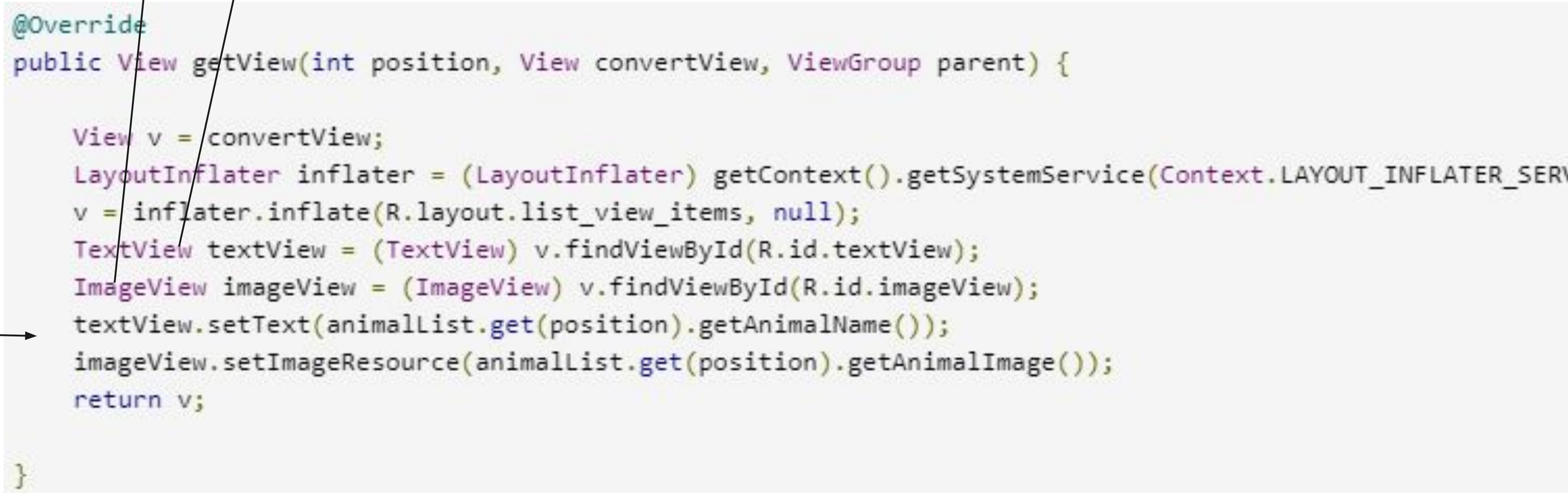
```
MyAdapter myAdapter=new MyAdapter(this,R.layout.list_view_items,animalList);
simpleList.setAdapter(myAdapter);
```

User defined adapter

This is the .xml file you have created. It was just a textview for lab2. We require more information for lab3

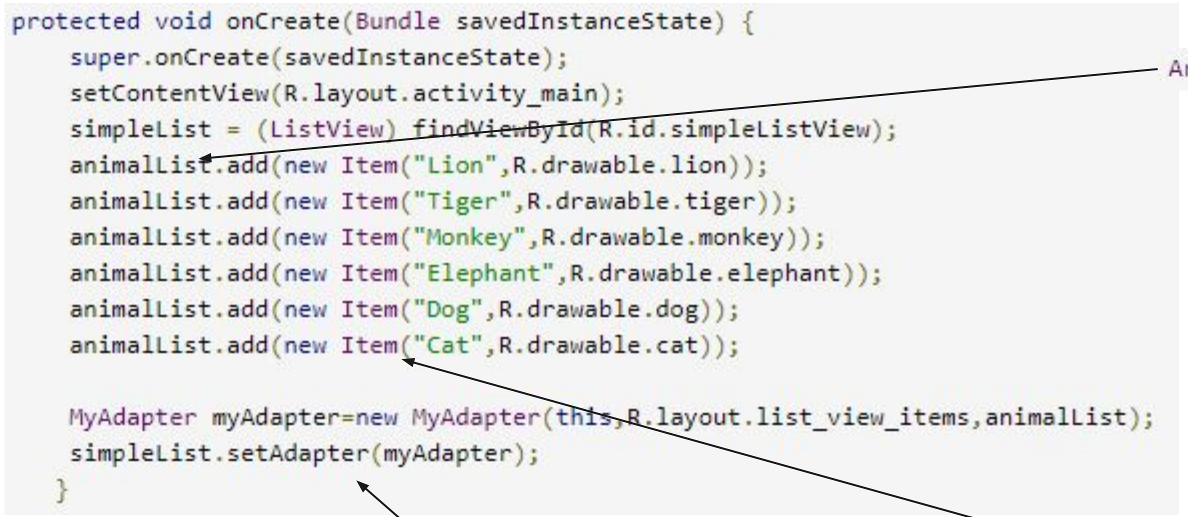
# Java code for adapter getView function



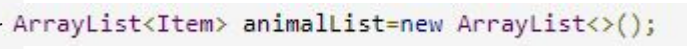


```java
@Override
public View getView(int position, View convertView, ViewGroup parent) {

    View v = convertView;
    LayoutInflater inflater = (LayoutInflater) getContext().getSystemService(Context.LAYOUT_INFLATER_SER
    v = inflater.inflate(R.layout.list_view_items, null);
    TextView textView = (TextView) v.findViewById(R.id.textView);
    ImageView imageView = (ImageView) v.findViewById(R.id.imageView);
    textView.setText(animalList.get(position).getAnimalName());
    imageView.setImageResource(animalList.get(position).getAnimalImage());
    return v;

}
```

Set view $\longrightarrow$

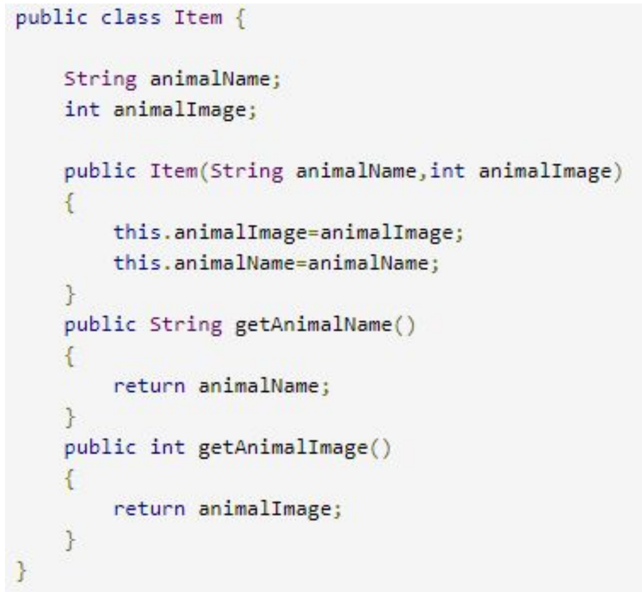


```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    simpleList = (ListView) findViewById(R.id.simpleListView);
    animalList.add(new Item("Lion",R.drawable.lion));
    animalList.add(new Item("Tiger",R.drawable.tiger));
    animalList.add(new Item("Monkey",R.drawable.monkey));
    animalList.add(new Item("Elephant",R.drawable.elephant));
    animalList.add(new Item("Dog",R.drawable.dog));
    animalList.add(new Item("Cat",R.drawable.cat));

    MyAdapter myAdapter=new MyAdapter(this,R.layout.list_view_items,animalList);
    simpleList.setAdapter(myAdapter);
  }
```



```
ArrayList<Item> animalList=new ArrayList<>();
```

Define a new class



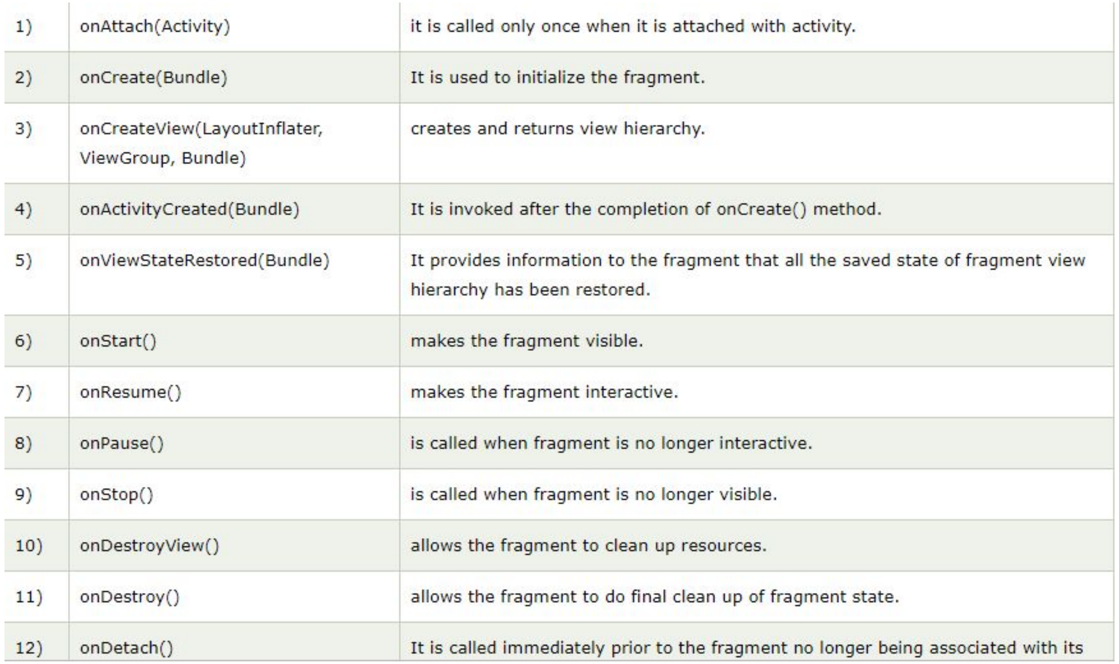
```
public class Item {

    String animalName;
    int animalImage;

    public Item(String animalName,int animalImage)
    {
        this.animalImage=animalImage;
        this.animalName=animalName;
    }
    public String getAnimalName()
    {
        return animalName;
    }
    public int getAnimalImage()
    {
        return animalImage;
    }
}
```

Display on main activity!

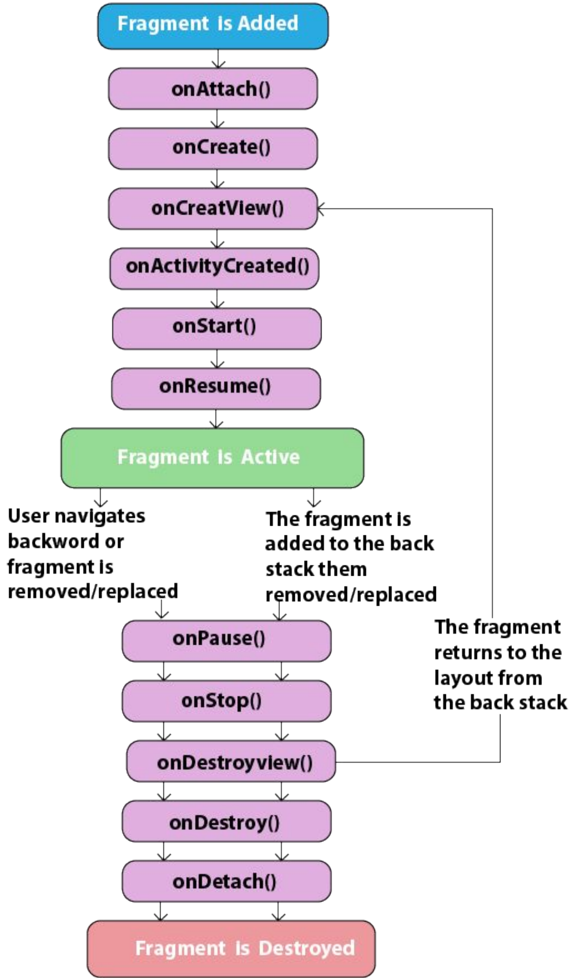
## Fragments

A Fragment represents a behavior or a portion of user interface in a FragmentActivity. You can **combine multiple fragments** in a single activity to build a multi-pane UI and **reuse** a fragment in multiple activities. You can think of a fragment as a modular section of an activity, which has its own **lifecycle**, receives its own input events, and which you can add or remove while the activity is running (sort of like a "sub activity" that you can reuse in different activities).

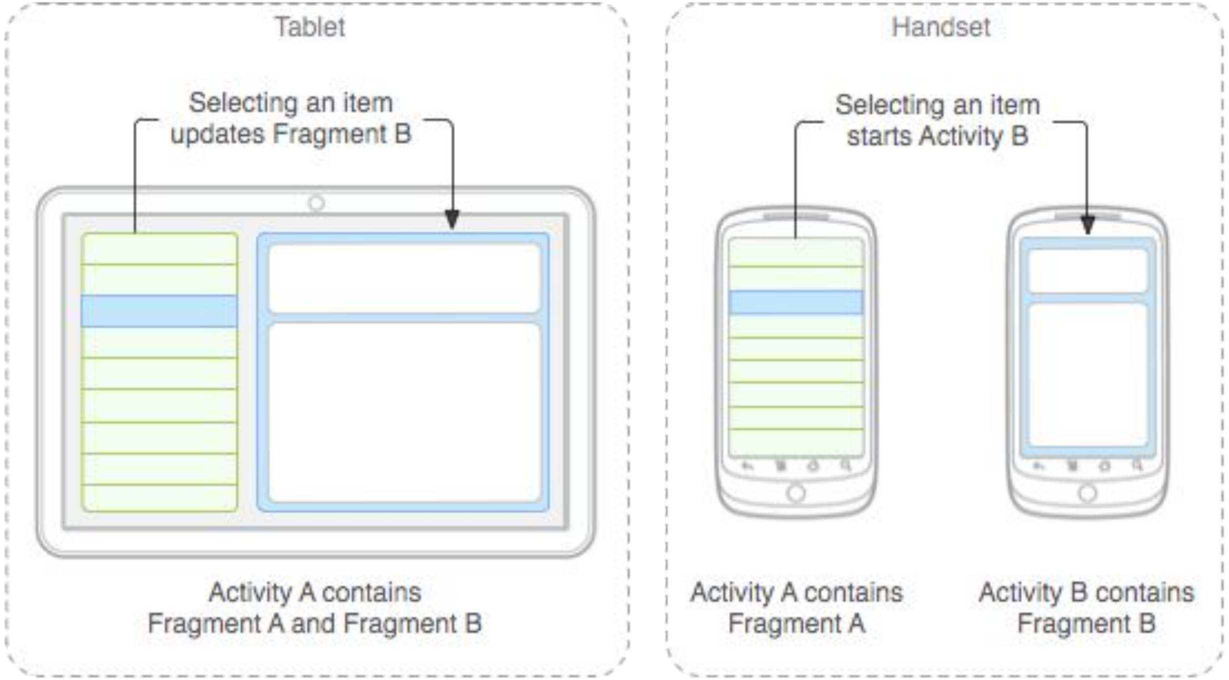
# Fragments lifecycle



| | | |
|---|---|---|
| 1) | onAttach(Activity) | it is called only once when it is attached with activity. |
| 2) | onCreate(Bundle) | It is used to initialize the fragment. |
| 3) | onCreateView(LayoutInflater, ViewGroup, Bundle) | creates and returns view hierarchy. |
| 4) | onActivityCreated(Bundle) | It is invoked after the completion of onCreate() method. |
| 5) | onViewStateRestored(Bundle) | It provides information to the fragment that all the saved state of fragment view hierarchy has been restored. |
| 6) | onStart() | makes the fragment visible. |
| 7) | onResume() | makes the fragment interactive. |
| 8) | onPause() | is called when fragment is no longer interactive. |
| 9) | onStop() | is called when fragment is no longer visible. |
| 10) | onDestroyView() | allows the fragment to clean up resources. |
| 11) | onDestroy() | allows the fragment to do final clean up of fragment state. |
| 12) | onDetach() | It is called immediately prior to the fragment no longer being associated with its |



Fragments is not required, but nice to have. It could speed up your application and save resources


Tablet
Selecting an item updates Fragment B
Activity A contains Fragment A and Fragment B
Handset
Selecting an item starts Activity B
Activity A contains Fragment A
Activity B contains Fragment B

## Java conventions!

## Credit

https://abhiandroid.com/ui/custom-arrayadapter-tutorial-example.html

https://www.javatpoint.com/android-fragments