Mobile Application Development

Introduction to Fragments

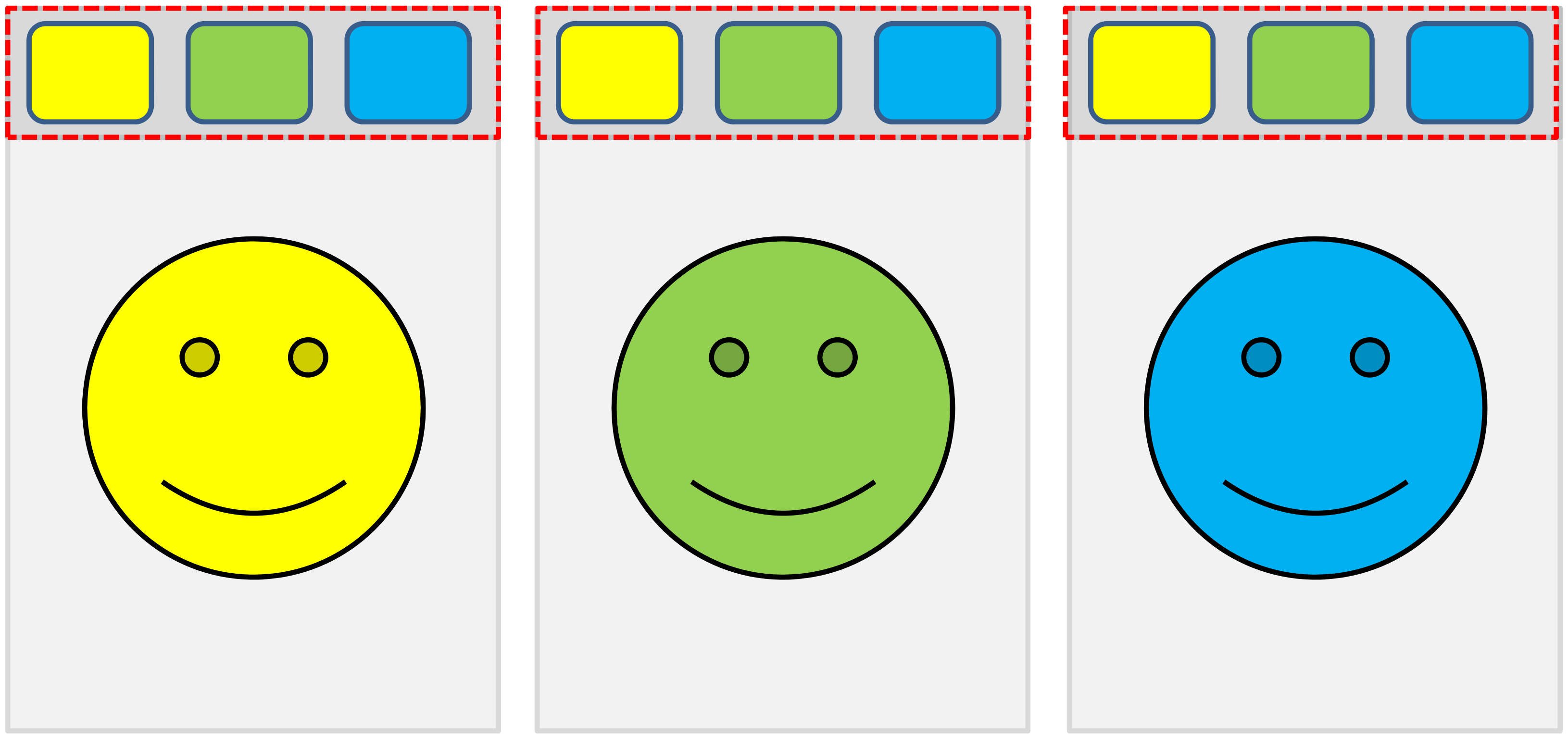
Fragments

- A Fragment represents a **behavior** or a **portion of user interface** in an Activity.
- 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

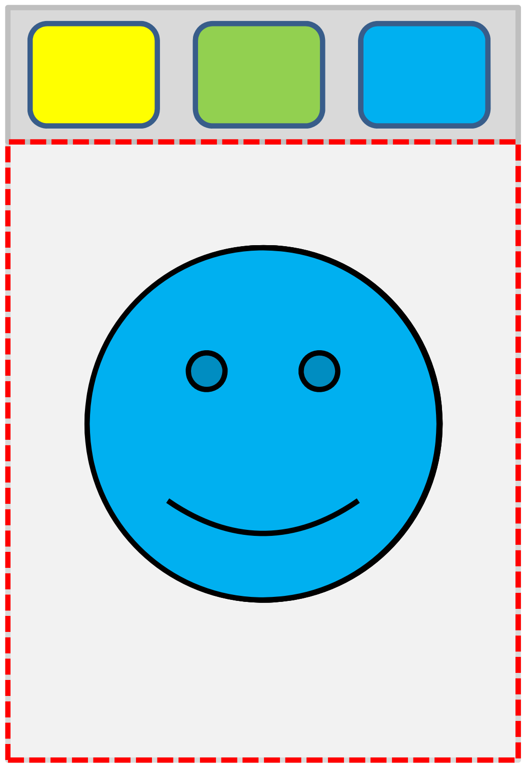
Why Fragments

- Reuse a Fragment in multiple Activities
- Multiple Fragments in an Activity
- Flexible Layouts on Larger Screens

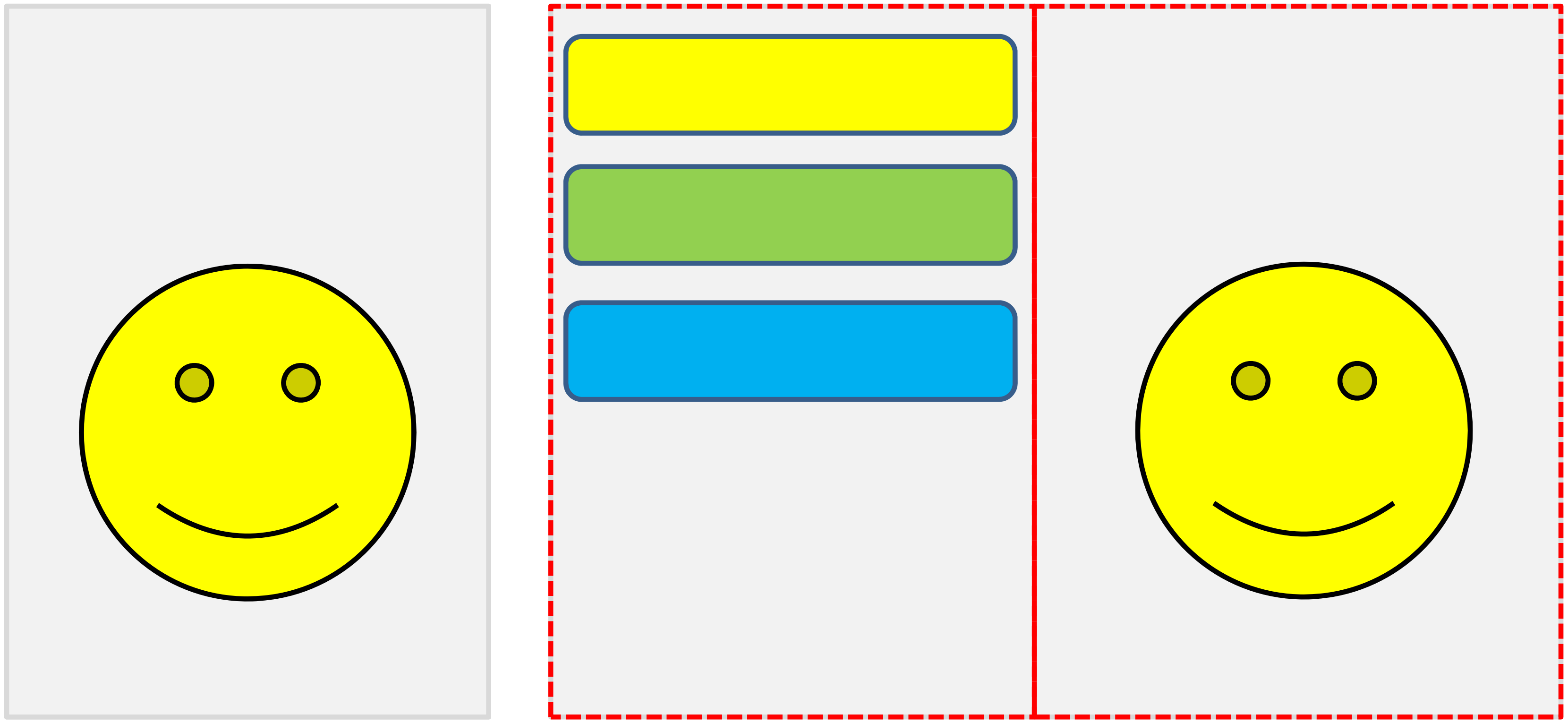
Reuse Fragments in Multiple Activities



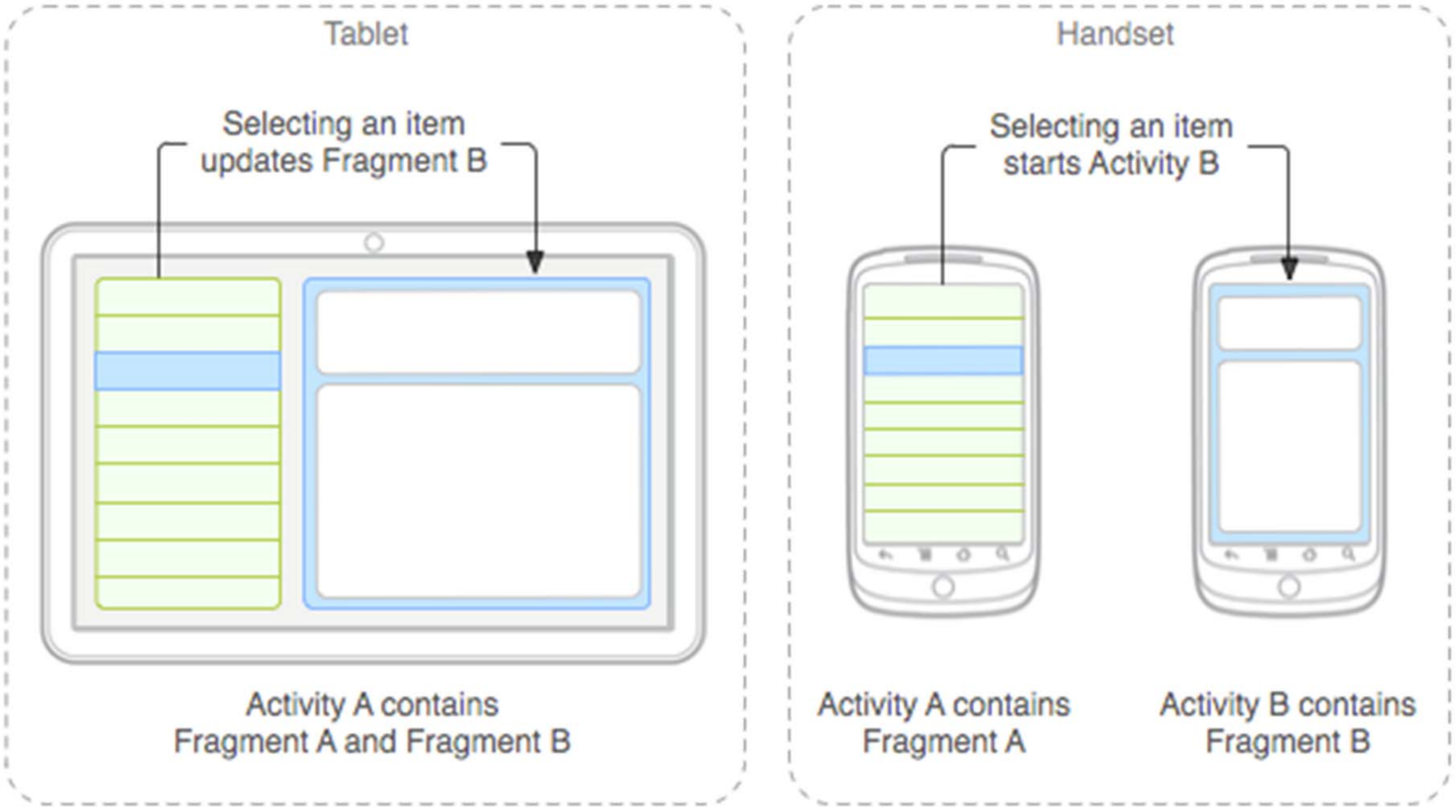
Multiple Fragments in an Activity



Flexible Layouts on Larger Screens



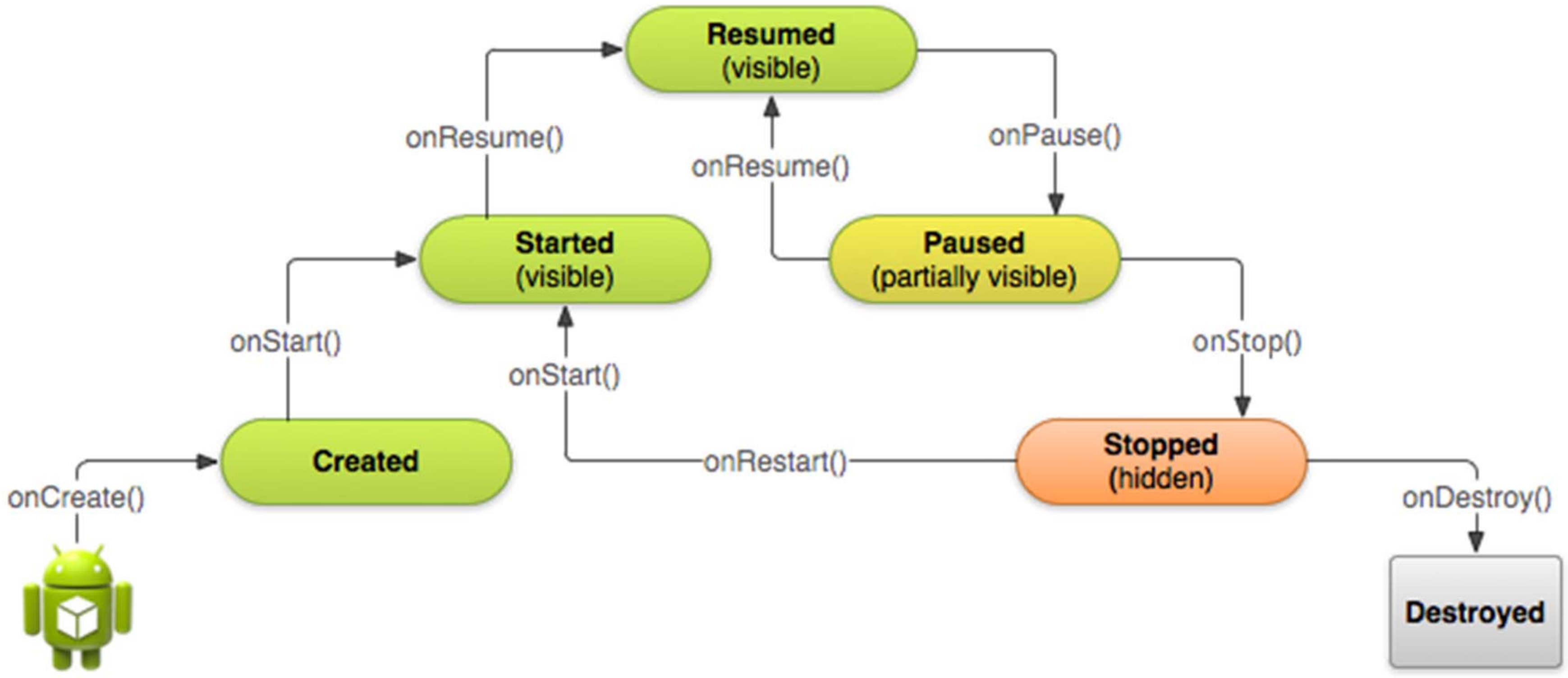
Flexible Layouts on Larger Screens


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

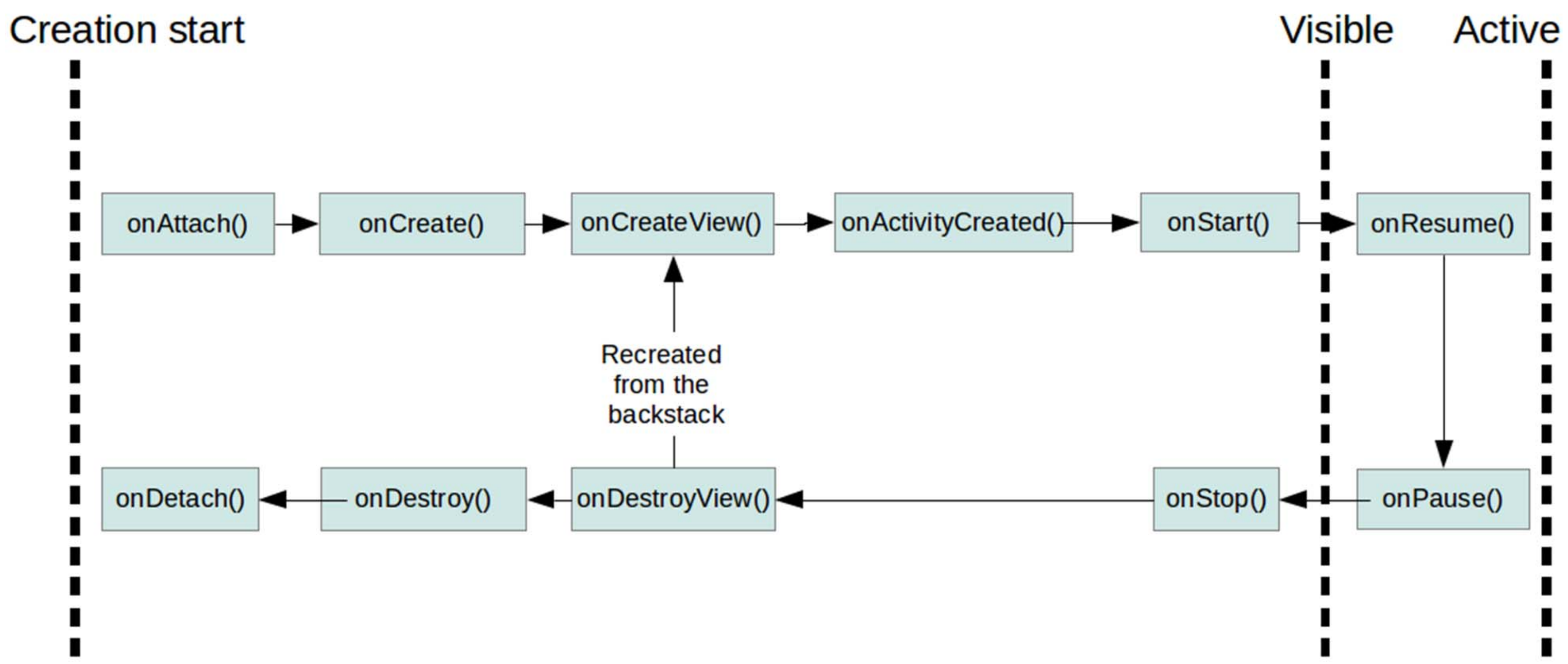
Fragment’s Lifecycle

- A **fragment must always be embedded in an activity** and the fragment's lifecycle is directly affected by the host activity's lifecycle.
- For example, when the activity is **paused**, so are all fragments in it, and when the activity is **destroyed**, so are all fragments.
- However, while an **activity is running** (it is in the resumed lifecycle state), you can manipulate each fragment independently, such as **add or remove them**.

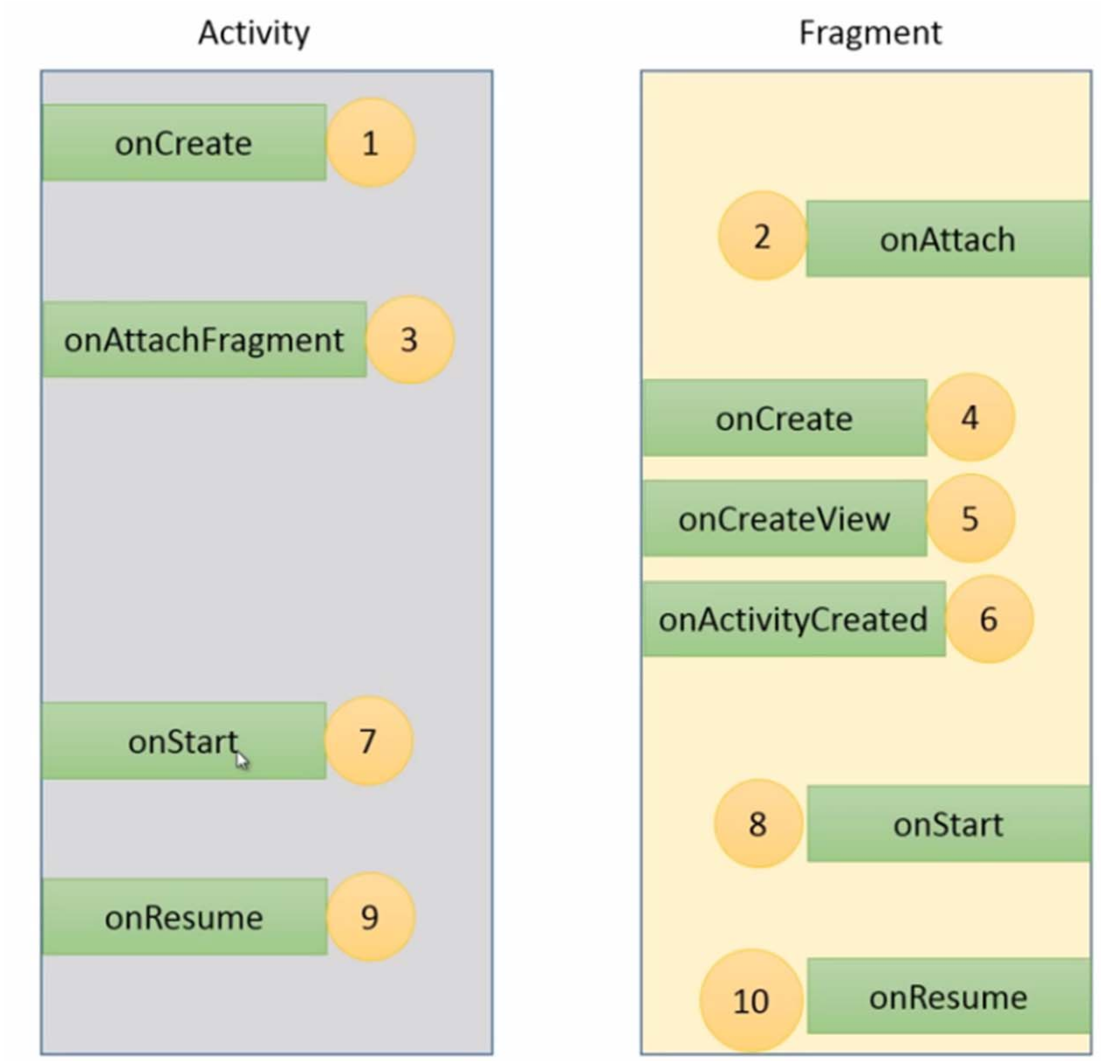
Activity Lifecycle States & Callbacks



Fragment Lifecycle Callbacks



Activity / Fragment Callbacks



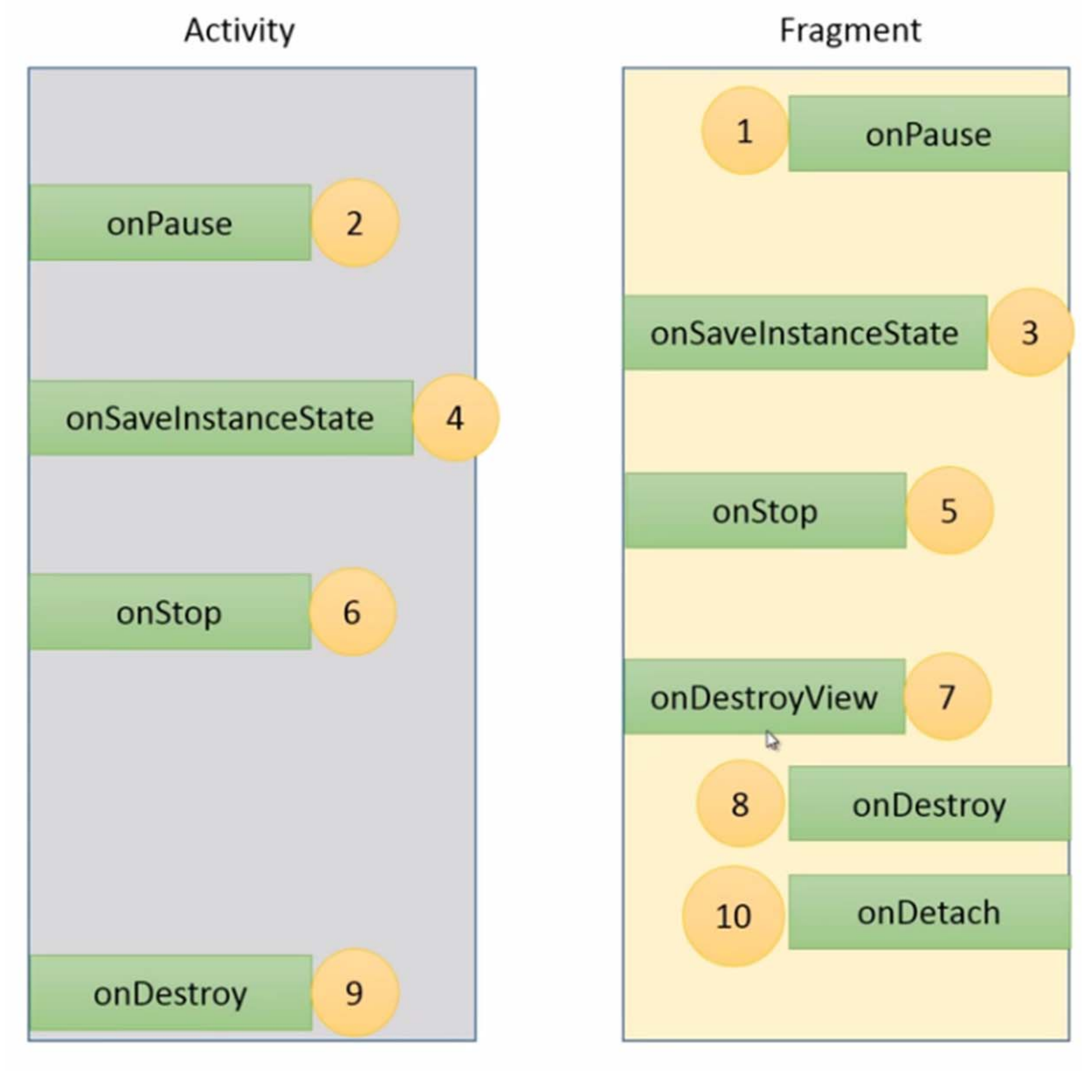
onAttach is called after Fragment is associated with its Activity
Gets a reference to the Activity object which can be used as Context

onCreate Don't use onCreate to access View hierarchy because Activity's onCreate may/may not be finished. Create background threads here for long running operations

onCreateView You are expected to return a View Hierarchy for your fragment

onActivityCreated Called after Activity onCreate has completed execution
Use this method to access/modify UI elements

Activity / Fragment Callbacks



onSaveInstanceState Use this to save information inside a Bundle object

onDestroyView Called after the Fragment View Hierarchy is no longer accessible

onDestroy Called after fragment is not used. It still exists as a Java object attached to the Activity

onDetach Fragment is not tied to the Activity and does not have a View hierarchy

Fragment

- When you add a fragment as a part of your activity layout, it **lives in a View inside the activity's view hierarchy** and the **fragment defines its own view layout**.

- You can insert a fragment into your activity layout

  - **Statically:** By declaring the fragment in the **activity's layout file,** as a **<fragment>** element

  - **Dynamically:** From your **application code** by adding it to an existing View.

ADDING FRAGMENT

Adding Fragment

- Create a Fragment by extending Fragment class
- Create Fragment's XML Layout
- Create an Activity class
- In Activity's XML Layout file and insert **<fragment>** element.
  - In <fragment> **android:name** attribute provide fully qualified name of Fragment class

Add Fragment in Activity’s XML

- Create an Activity
- Open Activity’s XML Layout file and insert **<fragment>** element.

```
<RelativeLayout ...>

    <fragment
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:name="pk.edu.riu.myapp.TestFragment"
        android:id="@+id/my_fragment"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

</RelativeLayout>
```

FRAGMENT MANAGER

Fragment Manager

- To manage the fragments in your activity, you need to use FragmentManager.
  - It maintains reference to all fragments inside the activity
  - Use **findFragmentById()** or **findFragmentByTag()** to get reference to a particular fragment.
- To get it, call **getFragmentManager()** from your activity.

Fragment Manager

```
public class MyActivity extends Activity {
    FragmentManager manager;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_my);

        manager=getFragmentManager();
    }

    public void SomeMethod() {
        SomeFragment f=(SomeFragment)

    manager.findFragmentById(R.id.container_v);
        . . .
    }
}
```

FRAGMENT TRANSACTIONS

Fragment Transactions

- A great feature about using fragments in your activity is the ability to **add**, **remove, replace**, and perform other actions with them, in response to user interaction.

- Each set of changes that you commit to the activity is called a **transaction** and you can perform one using APIs in **FragmentTransaction**.

- You can also save each transaction to a **back stack** managed by the activity, allowing the user to navigate backward through the fragment changes (similar to navigating backward through activities).

Fragment Transactions

- You can acquire an instance of FragmentTransaction from the FragmentManager like this:

```
. . .
FragmentManager manager;
. . .
manager=getFragmentManager();
. . .
FragmentTransaction transaction = manager.beginTransaction();
. . .
```

Fragment Transaction (add)

```
. . .
FragmentManager manager;
. . .
manager=getFragmentManager();
. . .
TestFragment f=new TestFragment();
. . .
FragmentTransaction transaction =
manager.beginTransaction();
transaction.add(R.id.container_view,f,"TF");
transaction.addToBackStack("fAdded");
transaction.commit();
```

Fragment Transaction (remove)

```
. . .
FragmentManager manager;
. . .
manager=getFragmentManager();
. . .
TestFragment f=new TestFragment();
. . .
FragmentTransaction transaction =
manager.beginTransaction();

transaction.remove(f);
transaction.addToBackStack("fRemoved");
transaction.commit();
```

Fragment Transaction (detach)

```
. . .
FragmentManager manager;
. . .
manager=getFragmentManager();
. . .
TestFragment f=new TestFragment();
. . .
FragmentTransaction transaction =
manager.beginTransaction();

transaction.detach(f);
transaction.addToBackStack("fDetached");
transaction.commit();
```

Fragment Transaction (attach)

```
. . .
FragmentManager manager;
. . .
manager=getFragmentManager();
. . .
TestFragment f=new TestFragment();
. . .
FragmentTransaction transaction =
manager.beginTransaction();

transaction.attach(f);
transaction.addToBackStack("fAttached");
transaction.commit();
```

Fragment Transaction (replace)

```
. . .
FragmentManager manager;
. . .
manager=getFragmentManager();
. . .
TestFragment f=new TestFragment();
AnotherFragment a=new AnotherFragment();
. . .
FragmentTransaction transaction =
manager.beginTransaction();

transaction.replace(R.id.container_view, a);
transaction.addToBackStack("fReplaced");
transaction.commit();
```

Override onBackPressed()

```
. . .
@Override
public void onBackPressed() {
  if (getFragmentManager().getBackStackEntryCount() > 0 )
  {
       getFragmentManager().popBackStack();
  } else {
       super.onBackPressed();
  }
}
```

Fragment Transactions

- When you statically add <fragment> element in your Activity, you can not remove() or replace() it.
- Instead you can use fragmentTransaction,'s **hide()** and **show()** methods for statically added fragment to manipulate it existence on screen.

OTHER NOTES

Get Reference to Activity in Fragment

- To get reference of host activity a fragment can use **getActivity()** method.
- You will need to get Activity’s reference:
  - To get reference to views on fragment itseld.
  - To get context.

```
Button btn=(Button) getActivity().findViewById(R.id.my_btn);
btn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent i=new Intent(getActivity(), NewActivity.class);
        startActivity(i);
    }
});
```

Design Philosophy

- You should **design each fragment as a modular and reusable** activity component.

- That is, because each fragment **defines its own layout and its own behavior with its own lifecycle callbacks**, you can include one fragment in multiple activities, so you should design for reuse and avoid directly manipulating one fragment from another fragment.

References

- http://developer.android.com/guide/components/fragments.html
- http://developer.android.com/reference/android/app/Fragment.html
- http://developer.android.com/training/basics/fragments/index.html
- http://developer.android.com/guide/practices/tablets-and-handsets.html
- http://vinsol.com/blog/2014/09/15/advocating-fragment-oriented-applications-in-android/
- https://corner.squareup.com/2014/10/advocating-against-android-fragments.html

Q & A