Android Programming

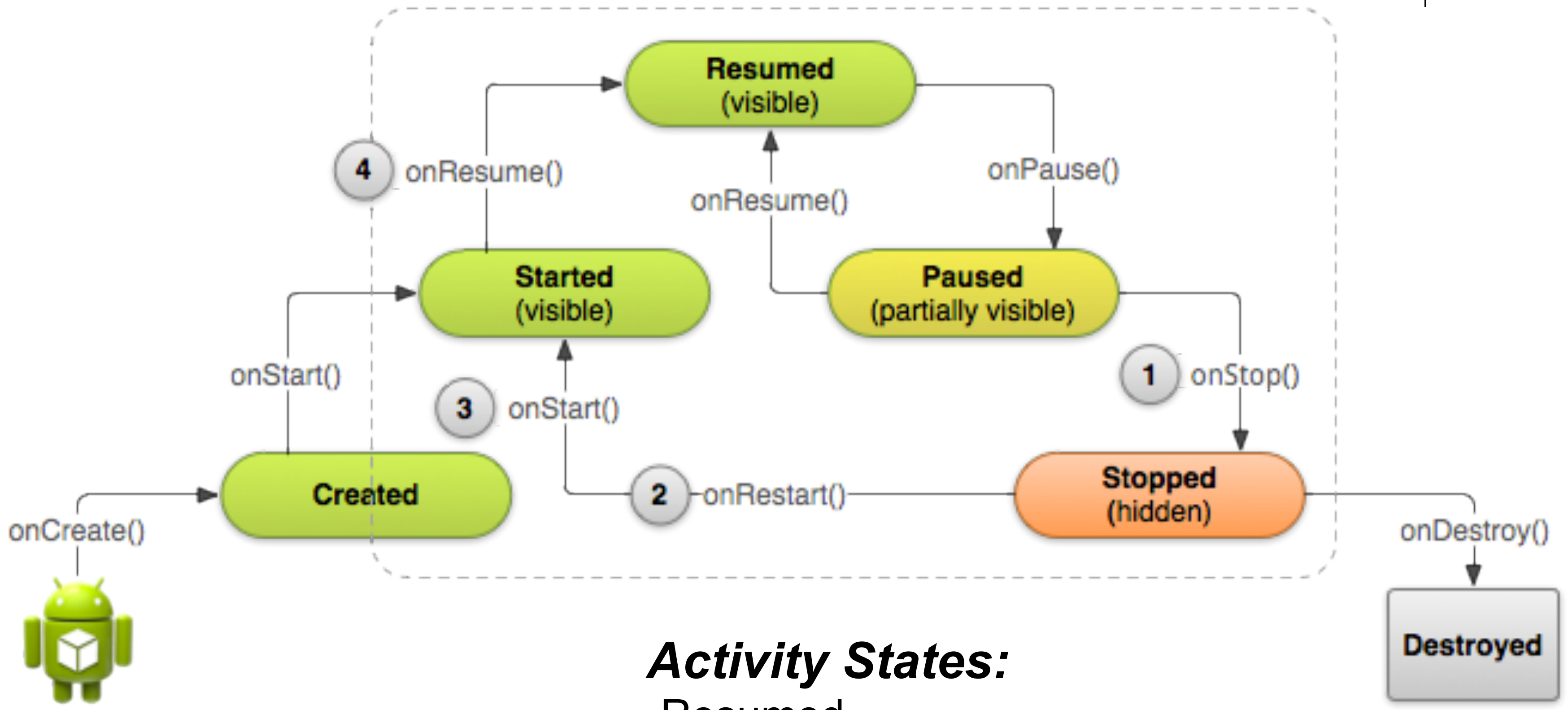
02 – User Interfaces

Part -1

Activities

- Major visual component
- Must be subclass of Activity
- Has a lifecycle
  - onCreate(): The system calls this when creating your activity. Component must be initialized. setContentView() must be called to define layout
  - More on activity methods later...

The Activity Lifecycle



Activity States:

-Resumed
-Paused
-Stopped
-Dead

Activity States

- **Resumed**: Activity is started by the user, is running and is in the foreground
- **Paused**: Activity is started by the user, is running and is visible, but another activity is overlaying part of the screen with transparent background.
- **Stopped**: Activity is started by the user, is running but is hidden by other activities that have been launched or switched to.
- **Dead**: Either the activity was never started or the activity was terminated *(by system or Activity.finish())*

Lifecycle Methods

Method	Purpose
onCreate()	Called when the Activity is created. Setup done here. Also provided is access to any previously stored state as Bundle
onStart()	Called when the Activity is becoming visible on the screen
onResume()	Called when the Activity starts interacting with the user, always called whether starting or restarting
onPause()	Called when Activity is pausing or reclaiming CPU and other resources , App crashes cause only onPause() called
onStop()	Called to stop the Activity and transition it to a nonvisible phase and subsequent lifecycle events
onDestroy()	Called when an Activity is being completely removed from system memory

Note: Your implementation of these lifecycle methods must always call the superclass implementation before doing any work

Activities (cont'd)

- Must be declared int the manifest

```
<manifest ... >
  <application ... >
      <activity android:name=".ExampleActivity" />
      ...
  </application ... >
  ...
</manifest >
```

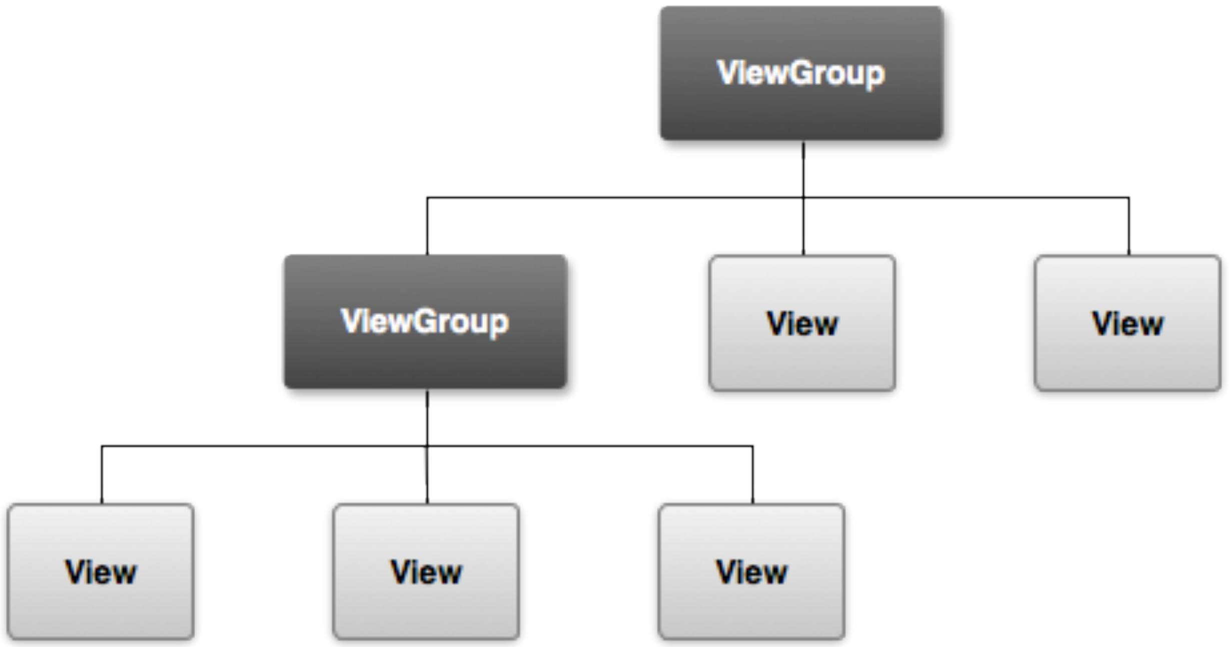
- Main Activity declaration

```
<activity android:name=".ExampleActivity" android:icon="@drawable/
app_icon">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Activities (cont’d)

- Has a hierarch of views. Each view is a subcass of *View* class (Ex. Button, TextView, EditText, RadioButton, …)
- Views are organized in layouts. Layouts are subclasses of *ViewGroup* class (Ex. LinearLayout, TableLayout, …)

UI Organization



Most Common Widgets

- TextView
    - A standard readonly text label, supports multiline display, string formatting, and word wrapping
- EditText
    - An editable text entry box. Accepts multiline entry and word wrapping
- ListView
    - A view group that creates and manages a group of views used to display the items in a list

Most Common Widgets

- Spinner
  - Composite control that displays a textview and associated ListView that lets you select an item
- Button
  - Classical clickable button
- Checkbox
  - Two state button : checked, unchecked
- RadioButton
  - List of options, only one selectable at a time

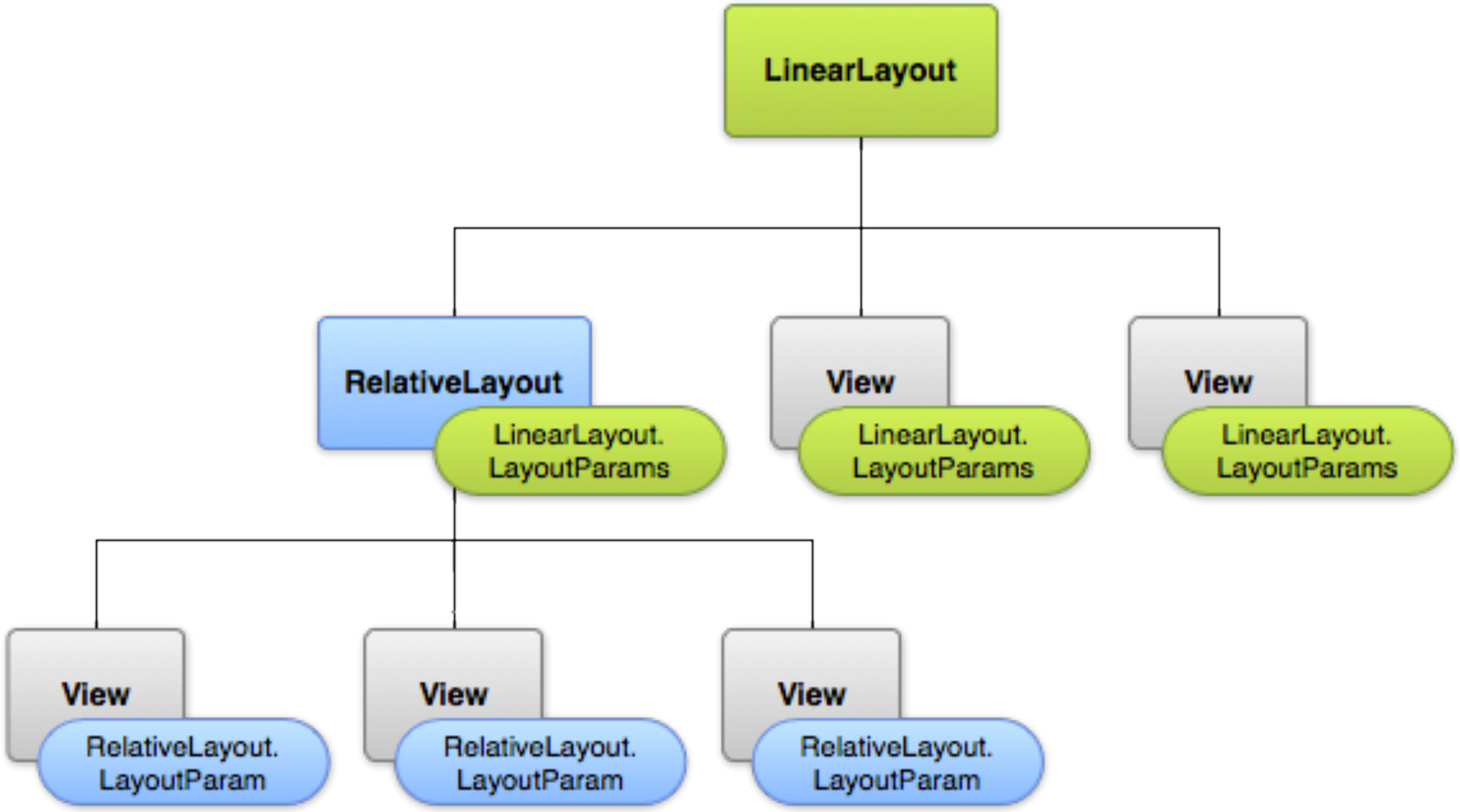
Layouts

- FrameLayout
  - It's basically a blank space on your screen that you can later fill with a single object. Objects put at top left, children will be drawn over
- LinearLayout
  - Aligns all children in a single direction — vertically or horizontally, depending on how you define the orientation attribute. All children are stacked one after the other.

Layouts

- TableLayout
  - positions its children into rows and columns. TableLayout containers do not display border lines for their rows, columns, or cells. The table will have as many columns as the row with the most cells
- ConstraintLayout
  - Lets child views specify their position relative to the parent view or to each other (specified by ID). So you can align two elements by right border, or make one below another

Embedding Layouts



A Layout Example

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />
    <Button
        android:id="@+id/btnCalc"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Make Calculation" />
    <EditText
        android:id="@+id/txtname"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="text" />
    <Button
        android:id="@+id/btnOk"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/btn_ok" />

    <TextView
        android:id="@+id/txtoutput"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

Always should be stated, here fills the current view

The hello text defined in res/values/strings.xml

unique id to be added to R.id

filter type of input text field

Loading The Layout

- Layouts are defined in res/layout folder
- Each activity can have a layout, or you can create the activity by hard coding
- In order to load layout:

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
}
```

$\downarrow$

points to res/layout/main.xml file

Layouts - ConstraintLayout

- Compatible with Android 2.3 (API level 9) and higher (Using support libraries).
- Allows you create large and complex layouts with a flat view hierarchy (no nested view groups).
- It's similar to RelativeLayout in that all views are laid out according to relationships between sibling views and the parent layout, but it's more flexible than RelativeLayout and easier to use with Android Studio's Layout Editor.

Layouts - ConstraintLayout

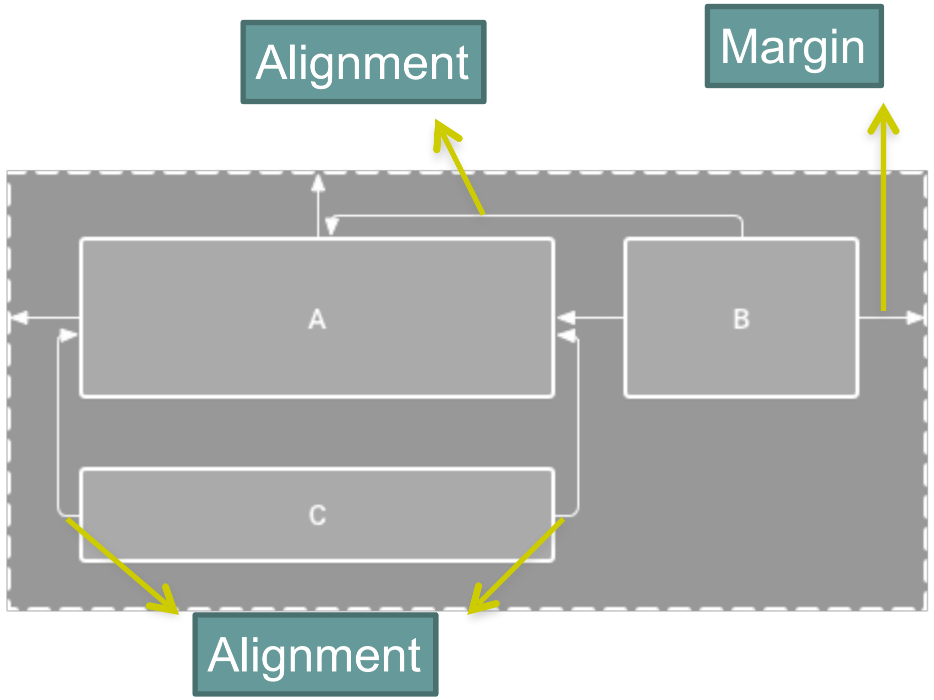


Figure 1. The editor shows view C below A, but it has no vertical constraint

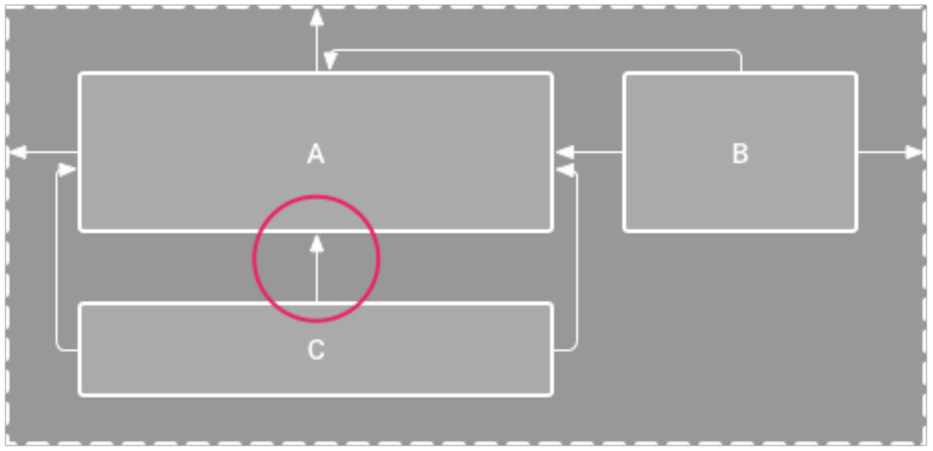


Figure 2. View C is now vertically constrained below view A

More Info: *https://developer.android.com/training/constraint-layout/index.html*

Configure ConstraintLayout

- Add required tools: Tools->Android->ADK Manager, Add ConstraintLayout for Android using the Support Repository
- Add the library dependency: build.gradle (module)

```
dependencies {
    compile 'com.android.support.constraint:constraint-
layout:1.0.1'
}
```

- Synch Gradle using the A/S toolbar

Logging

- To give output to embedded LogCat use:
  - Log.v([TAG],[MESSAGE]) : DEBUG LEVEL
  - Log.d([TAG],[MESSAGE]) : DEBUG LEVEL
  - Log.i([TAG],[MESSAGE]) : INFO LEVEL
  - Log.w([TAG],[MESSAGE]) : WARNING LEVEL
  - Log.e([TAG],[MESSAGE]) : ERROR LEVEL

Input Events

- The approach is to capture the events from the specific View object that the user interacts with
- Common events:

Listener Interface (View class)	Callback Method
OnClickListener	onClick()
OnLongClickListener	onLongClick()
OnFocusChangeListener	onFocusChange()
OnKeyListener	onKey()
OnTouchListener	onTouch()

Examples of OnClick Event

```
// Create an anonymous implementation of OnClickListener
private OnClickListener mCorkyListener = new OnClickListener() {
    public void onClick(View v) {
      // do something when the button is clicked
    }
};
protected void onCreate(Bundle savedValues) {
    ...
    // Capture our button from layout
    Button button = (Button)findViewById(R.id.corky);
    // Register the onClick listener with the implementation above
    button.setOnClickListener(mCorkyListener);
    ...
}
```

Metod 1

```
public class ExampleActivity extends Activity implements OnClickListener {
    protected void onCreate(Bundle savedValues) {
        ...
        Button button = (Button)findViewById(R.id.corky);
        button.setOnClickListener(this);
    }

    // Implement the OnClickListener callback
    public void onClick(View v) {
      // do something when the button is clicked
    }
    ...
}
```

Metod 2

Using android:onClick

- In your layout file you can use the onClick attribute of views to call methods in the context

```
<Button android:layout_width="match_parent" android:layout_height="wrap_content"
android:id="@+id/btn4" android:text="Button 4" android:onClick="testClick" />
```

Method must return void and have a View type parameter

```
public void testClick(View v){

    Log.i("INFO","Button 4 clicked!!!!!");

    }
```

Starting an Activity

- You can start another activity by calling startActivity(), passing it an Intent that describes the activity you want to start

```
Intent intent = new Intent(FirstActivity.this, SignInActivity.class);
startActivity(intent);
```

- Sending/getting data from one Activity to another (the object must be serializable)

```
intent.putExtra("STR_VAR", someString);
startActivity(intent);
```

```
getIntent().getExtras().getString(KEY);
```

StartActivityForResult

- Sometimes, you might want to receive a result from the activity that you start, for ex: selections, inputs, etc

1- In the caller activity create ActivityResultLauncher to catch the result data

```
ActivityResultLauncher<Intent> launcher = registerForActivityResult(new
ActivityResultContracts.StartActivityForResult(),
        new ActivityResultCallback<ActivityResult>() {
            @Override
            public void onActivityResult(ActivityResult result) {
                if(result.getResultCode()==RESULT_OK){

txtSelectedColor.setText(result.getData().getStringExtra("color").toString());
                }}});
```

2- Calller Activity: Create an intent and send it with a variable to differentiate the result

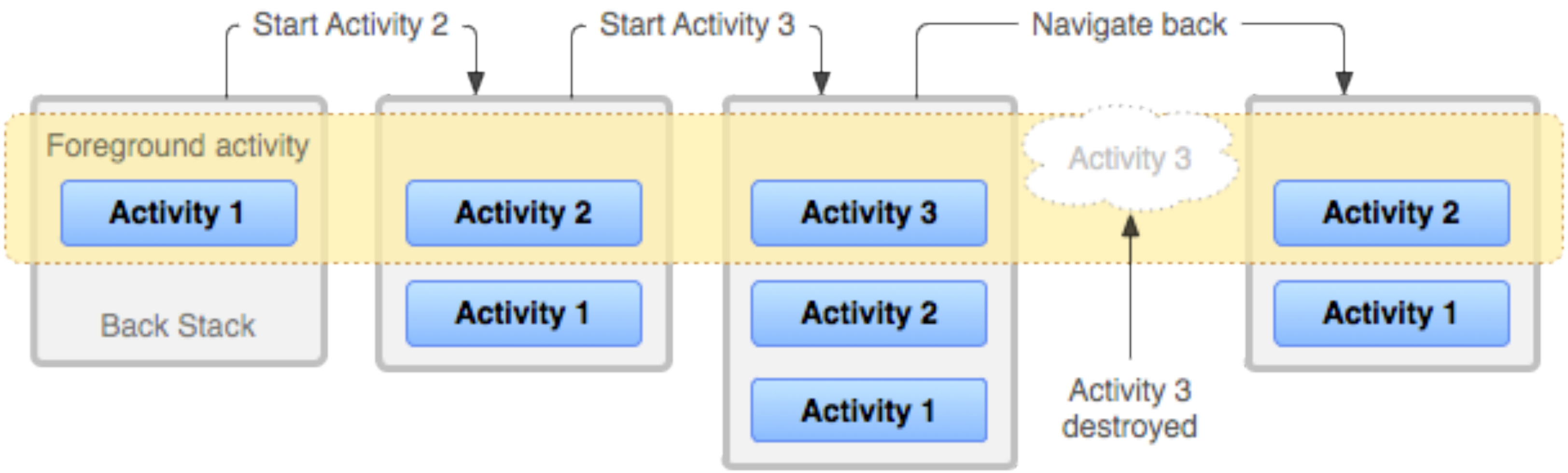
```
Intent i = new Intent(this,SelectColorActivity.class);
launcher.launch(i);
```

3- In the second activity get the result and send it back

```
Intent i = new Intent(this,TestActivity.class);
i.putExtra("color",selectedColor);
setResult(RESULT_OK,i);
finish();
```

Tasks and Backstack

- A task is a collection of activities that users interact with when performing a certain job.
- The activities are arranged in a stack (the "back stack"), in the order in which each activity is opened



Launch Mode in Manifest

- Setting the “launchMode” attribute in manifest file also defines the task behavior of Activities
- Possible values are:
  - **standard**
  - **singleTop** :If activity is at top of back stack, a new instance is not created, onNewIntent() called
  - **singleTask** :Reuse activity if exists, else create new task
  - **singleInstance** :Same as singleTask but no other activities are ever inserted into the created task stack.

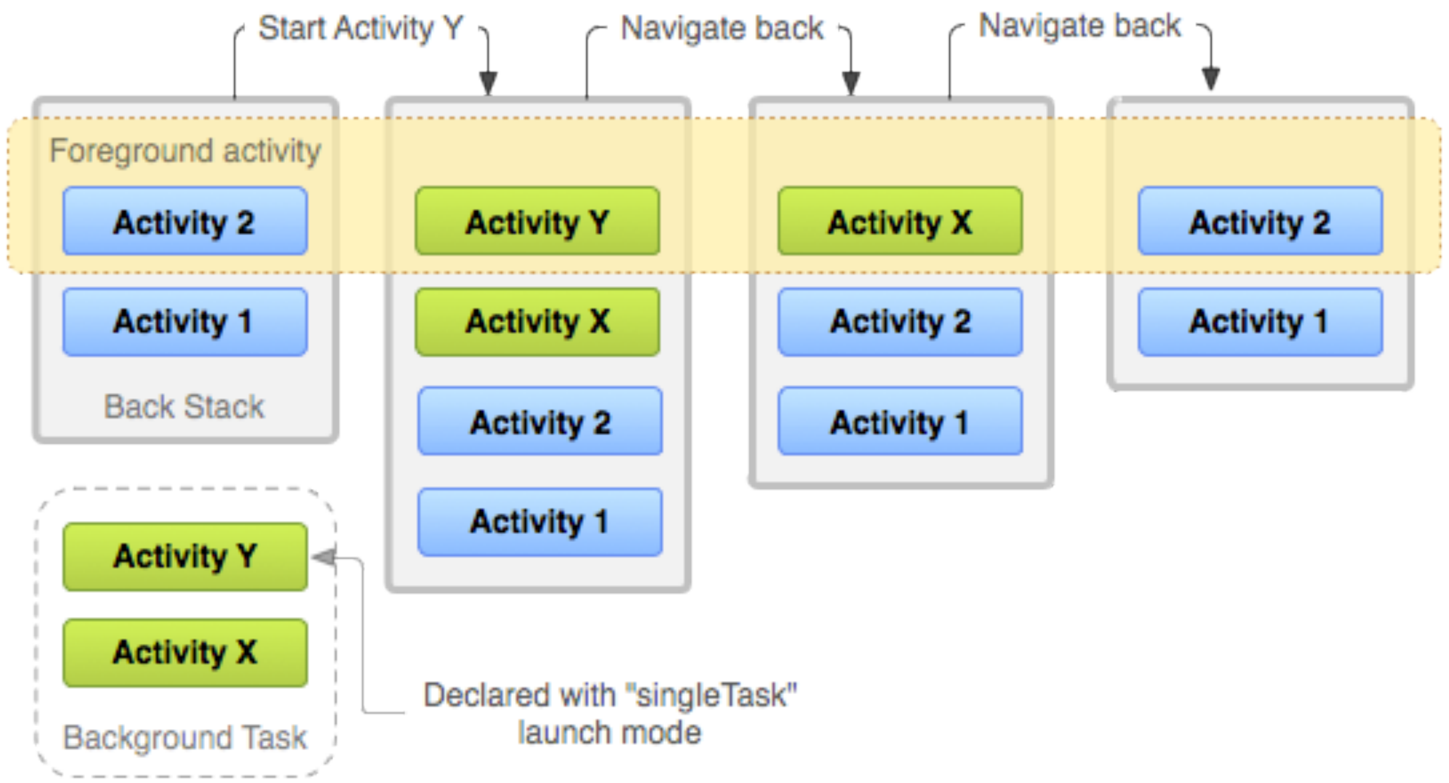
Controlling Task Behavior

- On the caller intent use setFlags(), some of flags:
  - FLAG_ACTIVITY_NEW_TASK
    - Start the activity in a new task. If a task is already running for the activity you are now starting, that task is brought to the foreground with its last state restored and the activity receives the new intent in onNewIntent(). (= singleTask)
  - FLAG_ACTIVITY_CLEAR_TOP
    - activities: A, B, C, D. If D calls startActivity() with an Intent that resolves to the component of activity B, then C and D will be finished and B receive the given Intent, resulting in the stack now being: A, B

Controlling Task Behavior

- FLAG_ACTIVITY_SINGLE_TOP
  - If the activity being started is the current activity (at the top of the back stack), then the existing instance receives a call to onNewIntent(), instead of creating a new instance of the activity

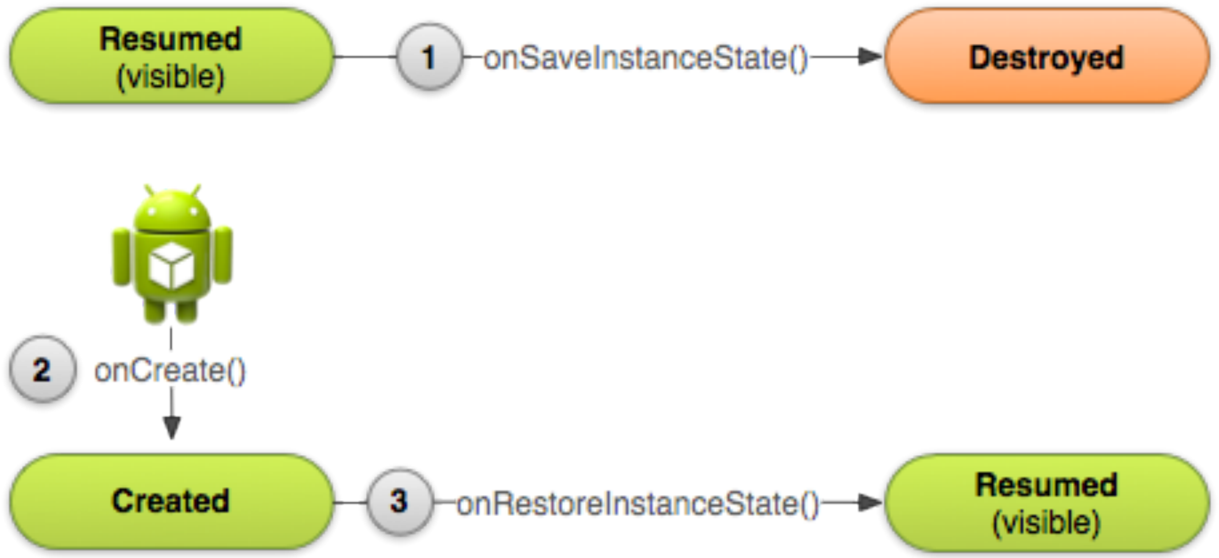
A Backstack Example



Saving Instance State

- Data throughout the application lifetime can be saved in the application bundle
- To store data through activity state changes override onSaveInstanceState and save the data in bundle parameter
- To store data between application instances (ie permanently) use SharedPreferences. This data is written to the database on the device and is available all the time.

Store State Application Wide



- Can restore at onCreate() (Check if bundle parameter is null)
- Or at onRestoreInstanceState() (Called after onStart if bundle is not empty)

Store State Application Wide

Example: onSaveInstanceState

```
@Override
protected void onSaveInstanceState(Bundle outState) {
     super.onSaveInstanceState(outState);
     outState.putString("TEXT", txtStore.getText().toString());
}
```

Example: onRestoreInstanceState

```
@Override
protected void onRestoreInstanceState(Bundle savedInstanceState) {
     super.onRestoreInstanceState(savedInstanceState);
     if(savedInstanceState!=null){
          txtStore.setText(savedInstanceState.getString("TEXT"));
     }
}
```

Store State in Preferences (Permanently)

Storing the State

MODE_WORLD_READABLE
MODE_WORLD_WRITABLE
MODE_PRIVATE

```
  @Override
    protected void onPause() {
        super.onPause();
        SharedPreferences pref = getPreferences(MODE_PRIVATE);
        SharedPreferences.Editor editor = pref.edit();

        editor.putString("num1", txtNum1.getText().toString());
        editor.putString("num2", txtNum2.getText().toString());
        editor.commit();
    }
```

Getting the state back

```
//inside onCreate() method
SharedPreferences pref = getPreferences(MODE_PRIVATE);
txtNum1.setText(pref.getString("num1", null));
txtNum2.setText(pref.getString("num2", null));
```