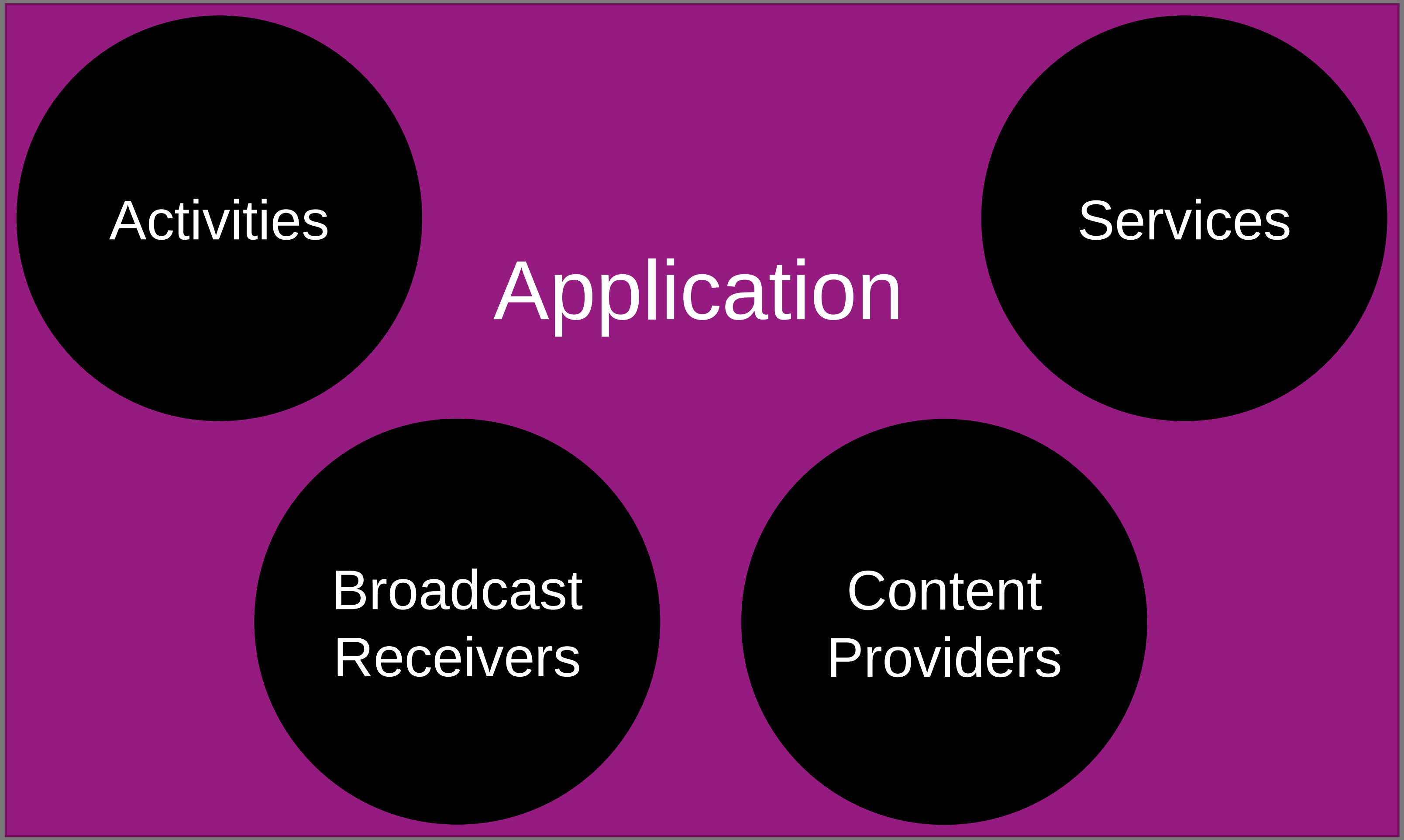
ANDROID ACTIVITIES

Peter Larsson-Green

Jönköping University

Spring 2020

FUNDAMENTAL APP COMPONENTS



AndroidManifest.xml

```xml
<manifest
  package="se.ju.larpet.myapplication"
  xmlns:android="http://schemas.android.com/apk/res/android"
>
   <application android:label="My Cool App">
      <!-- Here we list all our fundamental app components. -->
   </application>
</manifest>
```

ACTIVITIES

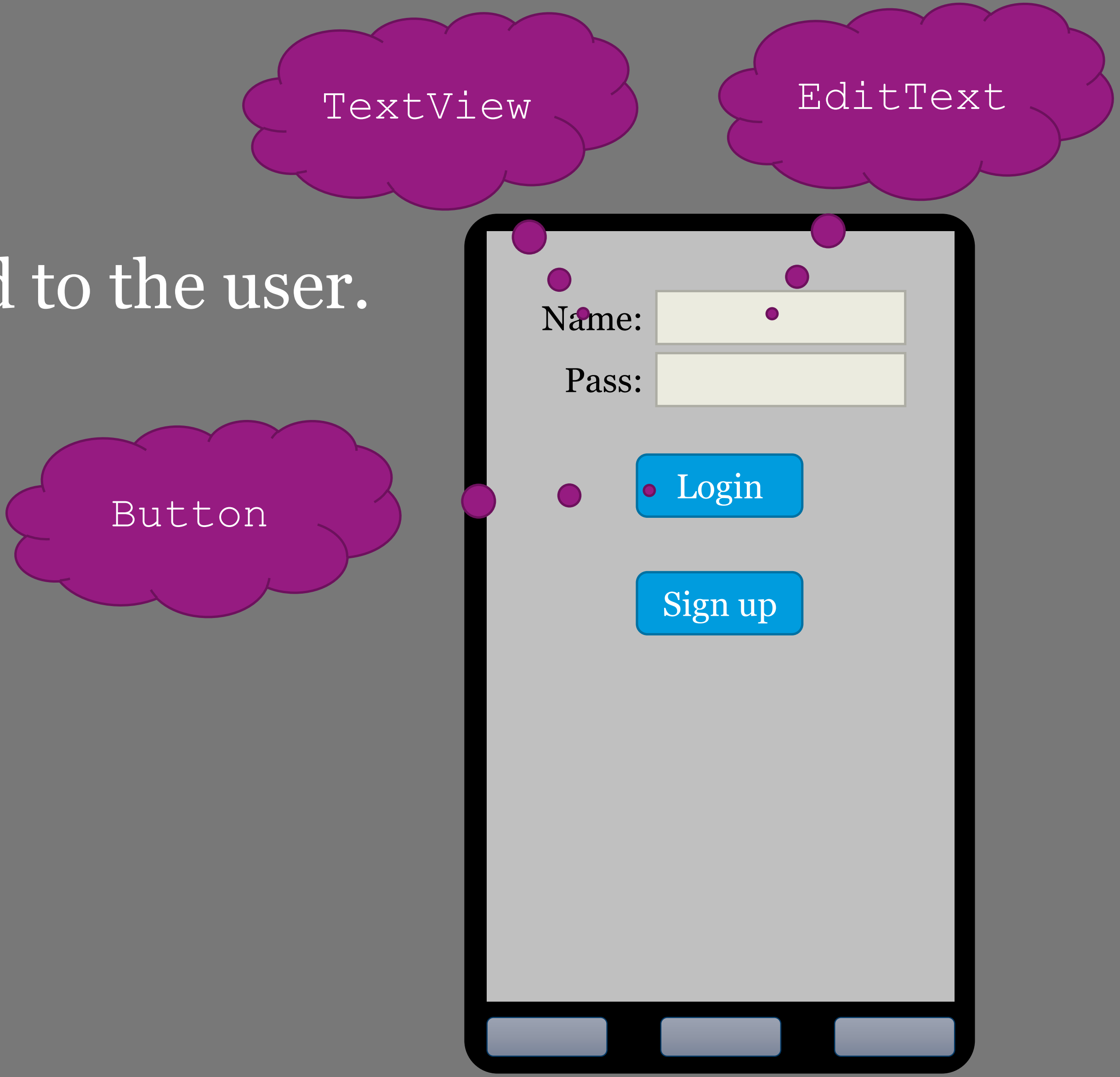
```
public class MyActivity extends Activity{
  @Override
  protected void onCreate(Bundle savedInstanceState){
    // Setup the GUI.
  }
}
```

```
<intent-filter>
  <category android:name="android.intent.category.LAUNCHER"/>
  <action android:name="android.intent.action.MAIN"/>
</intent-filter>
```

```
<manifest ...>
  <application ...>
    <activity android:name=".MyActivity" android:label="Main">
    </activity>
  </application>
</manifest>
```

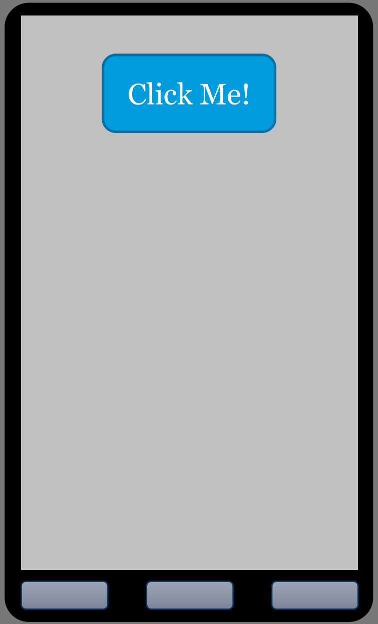
ACTIVITIES

- Activity = one screen presented to the user.
- Small screen → do one thing.
- Consists of `Views`.
  - Exists over 100 different.
  - Widget = `View` you can see.
  - `ViewGroup` contains Widgets.



ACTIVITIES

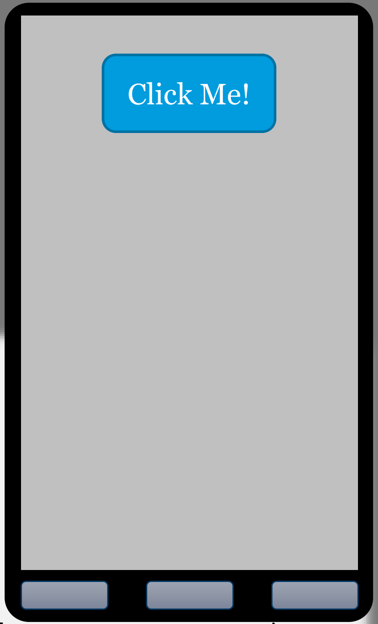
- An `Activity` contains one `ViewGroup` by default.
  - Has the id `android.R.id.content`.
- `theActivity.findViewById(theId)`



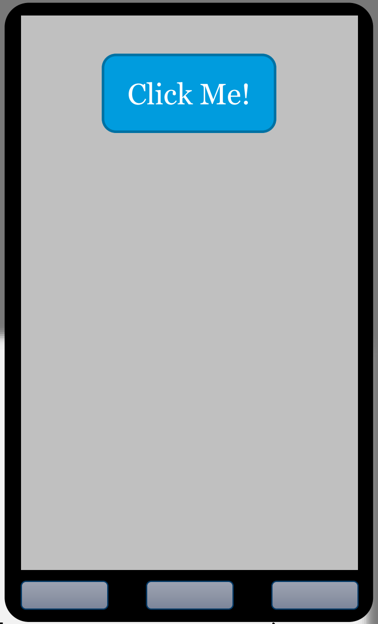
```
public class MyActivity extends Activity{
  @Override
  protected void onCreate(Bundle savedInstanceState){
    ViewGroup rootView = (ViewGroup) findViewById(android.R.id.content);
  }
}
```

ACTIVITIES

```
public class MyActivity extends Activity{
  @Override
  protected void onCreate(Bundle savedInstanceState){
    ViewGroup rootView = (ViewGroup) findViewById(android.R.id.content);
    Button button = new Button(this);
    button.setText("Click Me!");
    rootView.addView(button);
  }
}
```

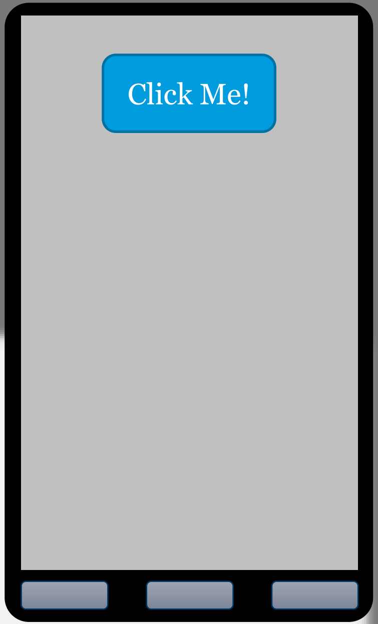


ACTIVITIES



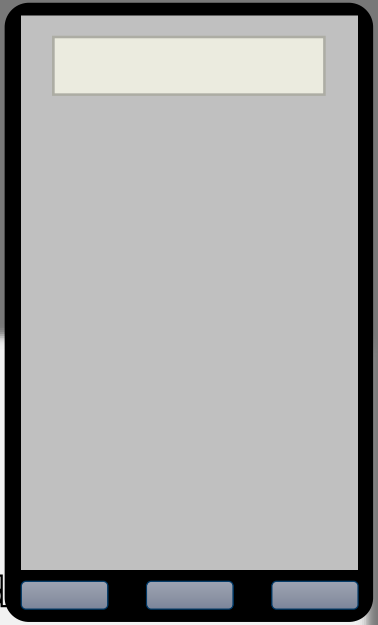
```
public class MyActivity extends Activity{
  @Override
  protected void onCreate(Bundle savedInstanceState){
    ViewGroup rootView = (ViewGroup) findViewById(android.R.id.content);
    Button button = new Button(this);
    button.setText("Click Me!");
    button.setOnClickListener(new View.OnClickListener(){
      @Override
      public void onClick(View v){ /* Button clicked. */ }
    });
    rootView.addView(button);
  }
}
```

ACTIVITIES



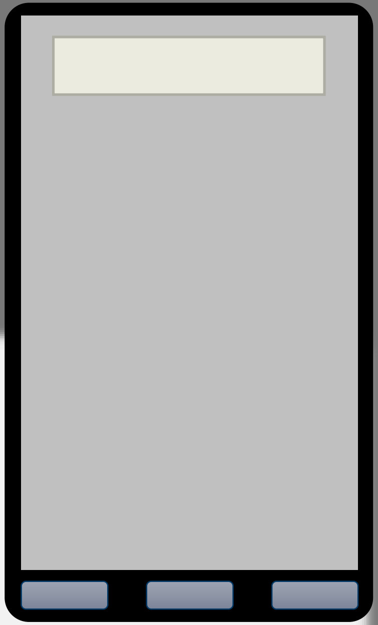
```
public class MyActivity extends Activity
  implements View.OnClickListener{
  @Override
  protected void onCreate(Bundle savedInstanceState){
    ViewGroup rootView = (ViewGroup) findViewById(android.R.id.content);
    Button button = new Button(this);
    button.setText("Click Me!");
    button.setOnClickListener(this);
    rootView.addView(button);
  }
  @Override
  public void onClick(View v){ /* Button clicked. */ }
}
```

ACTIVITIES



```
public class MyActivity extends Activity{
  @Override
  protected void onCreate(Bundle savedInstanceState){
    ViewGroup rootView = (ViewGroup) findViewById(android.R.id
    EditText editText = new EditText(this);
    editText.addTextChangedListener(new TextWatcher(){
      @Override
      public void afterTextChanged(Editable s){ /* Text changed. */ }
    });
    rootView.addView(editText);
  }
}
```

ACTIVITIES

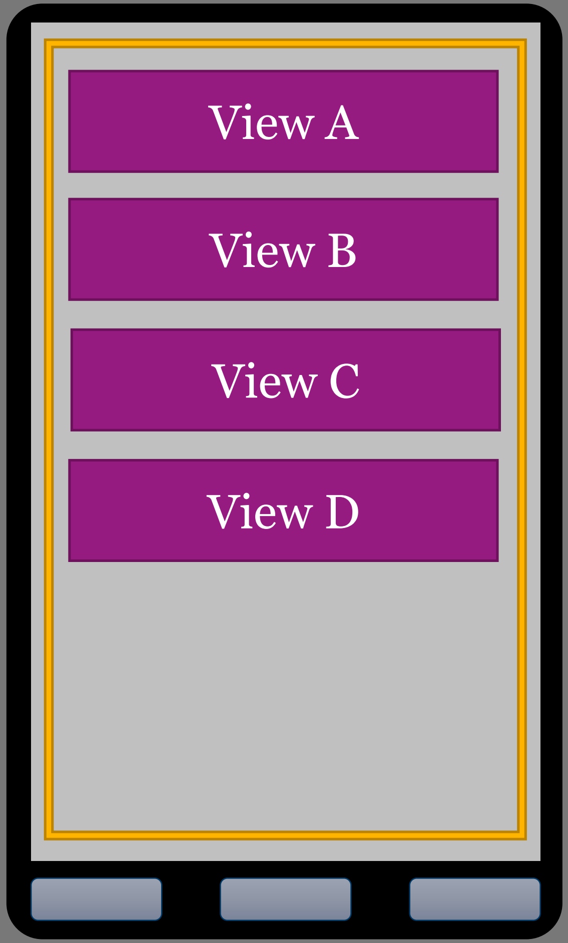


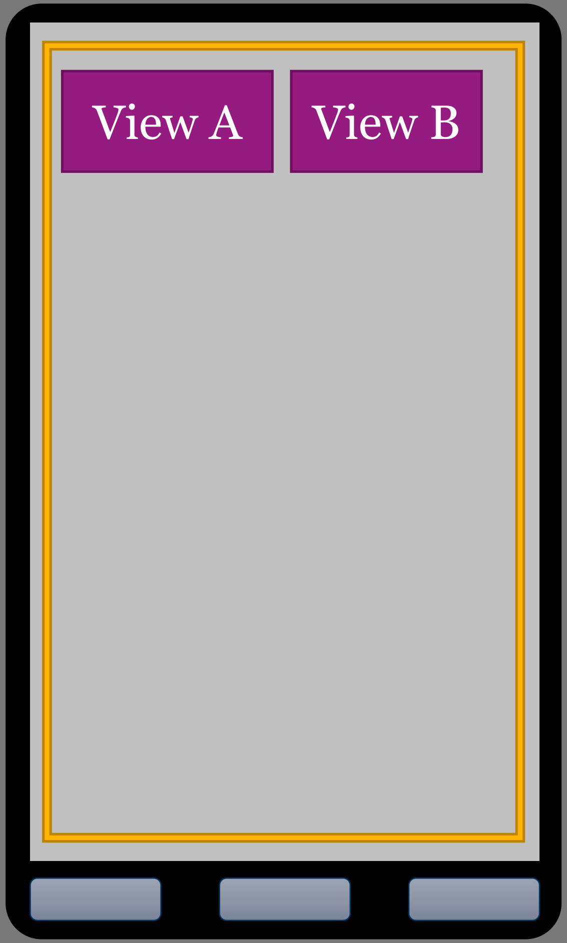
```
public class MyActivity extends Activity
  implements TextWatcher{
  @Override
  protected void onCreate(Bundle savedInstanceState){
    ViewGroup rootView = (ViewGroup) findViewById(android.R.id.content);
    EditText editText = new EditText(this);
    editText.addTextChangedListener(this);
    rootView.addView(editText);
  }
  @Override
  public void afterTextChanged(Editable s){ /* Text changed. */ }
}
```

LAYOUTS

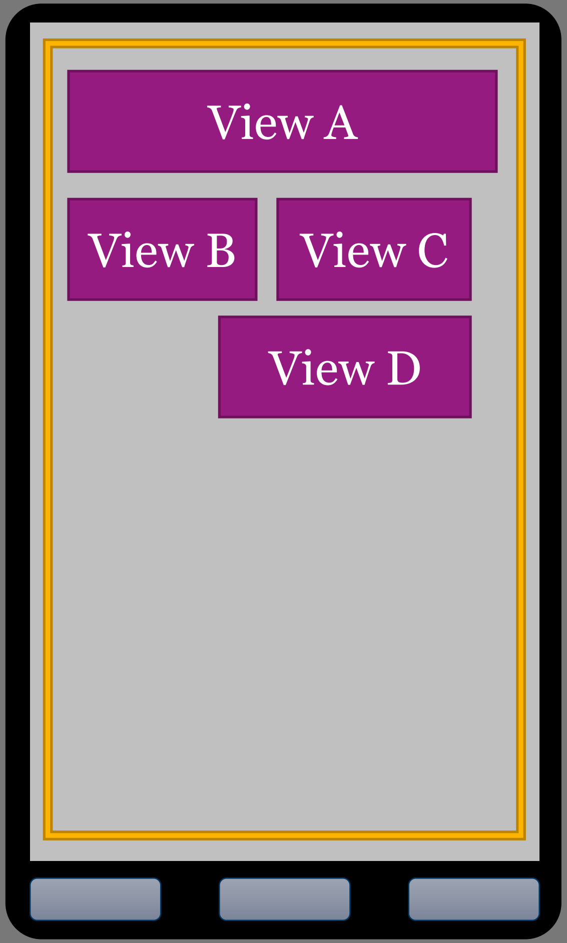
- Different `*Layout` classes extends `ViewGroup`.
- Different layouts positions the element differently on the screen.

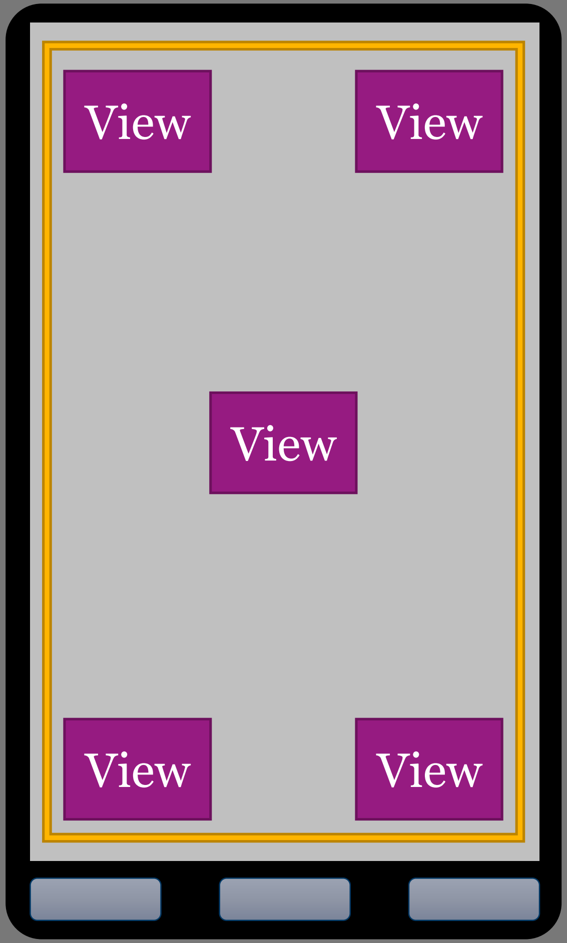
LINEAR LAYOUT



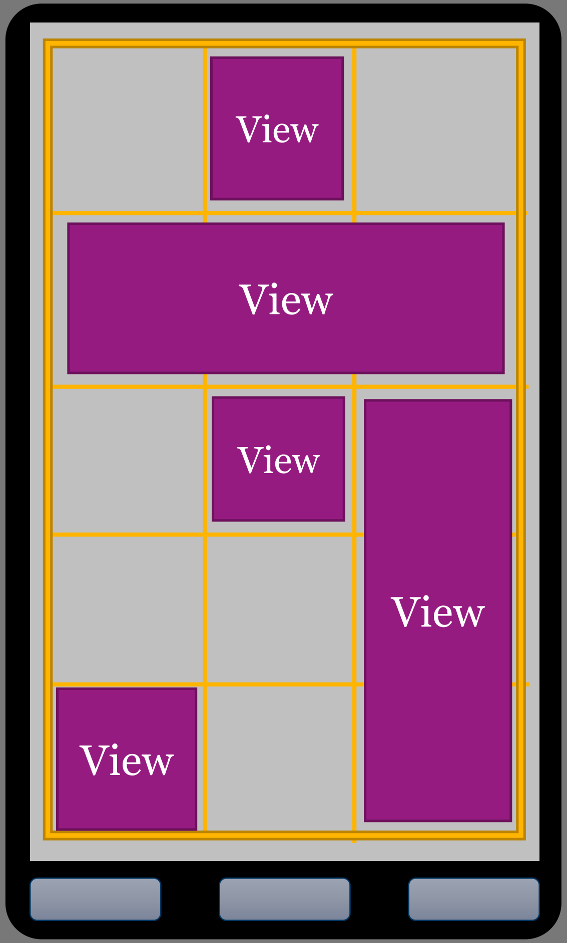


RELATIVE LAYOUT





GRID LAYOUT



LAYOUT FILES

Creating the entire GUI in Java is hard.

- Android allows you to specify the GUI in XML files.

```
<LinearLayout
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical">
   <Button
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:text="Click Me!" />
   <!-- And more buttons, or other views... -->
</LinearLayout>
```

res/layout/my_layout.xml

```
public class MyActivity
  extends Activity{
    @Override
     protected void onCreate(
       Bundle savedInstanceState
     ){
        setContentView(R.layout.my_layout);
     }
}
```

LAYOUT FILES

Listen for clicks on `Views` defined in layout files?

- Give the `View` an id, then use `theActivity.findViewById(R.id.theId)`.

```
<LinearLayout
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical">
   <Button
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:text="Click Me!"
     android:id="@+id/theId"/>
</LinearLayout>
```

STRING RESOURCES

Android has built in support for i18n.

- Do not hard code text in your code; use string resources:
  - Write all your text in `res/values/strings.xml`:

```
<resources>
  <string name="view">View</string>
  <string name="select_one">Select One</string>
</resources>
```

```
<resources>
  <string name="view">Visa</string>
  <string name="select_one">Välj En</string>
</resources>
```

  - Basically create one file for each language you support.
  - Android can then fetch the strings from the file corresponding to the user's selected language.

STRING RESOURCES

Android has built in support for i18n.

- Do not hard code text in your code; use string resources.
- To obtain one in XML (e.g. layouts):
  - `@string/select_one`
- To obtain one in Java:
  - `String theString = aContext.getString(R.string.view);`
  - (`Activity` inherits from `Context`).

```
<resources>
  <string name="view">View</string>
  <string name="select_one">Select One</string>
</resources>
```

INTENTS

Intent = request to start an app component.

- *Explicit Intent*: You decide which app component (usually your own).
  - The activity does not need an `<intent-filter>`.

```
Intent intent = new Intent(aContext, OtherActivity.class);
intent.putExtra("id", 26);
aContext.startActivity(intent);
```

- *Implicit Intent*: OS/user decides which app component.
  - The activity needs to use an `<intent-filter>`.

```
Uri uri = Uri.parse("tel:5551234");
Intent intent = new Intent(Intent.ACTION_DIAL, uri);
aContext.startActivity(intent);
```

CLOSING AN ACTIVITY

An activity can close itself by calling the `finish()` method.

- The default behavior when the user presses the back button is to close the activity.

```
public class MyActivity extends Activity{
  @Override
  protected void onBackPressed(){
    finish();
  }
}
```

START ACTIVITY FOR RESULT

Somewhere in `MyActivity.java`:

```
Intent intent = new Intent(aContext, PickContactActivity.class);
int requestCode = 1234;
anActivity.startActivityForResult(intent, requestCode);
```

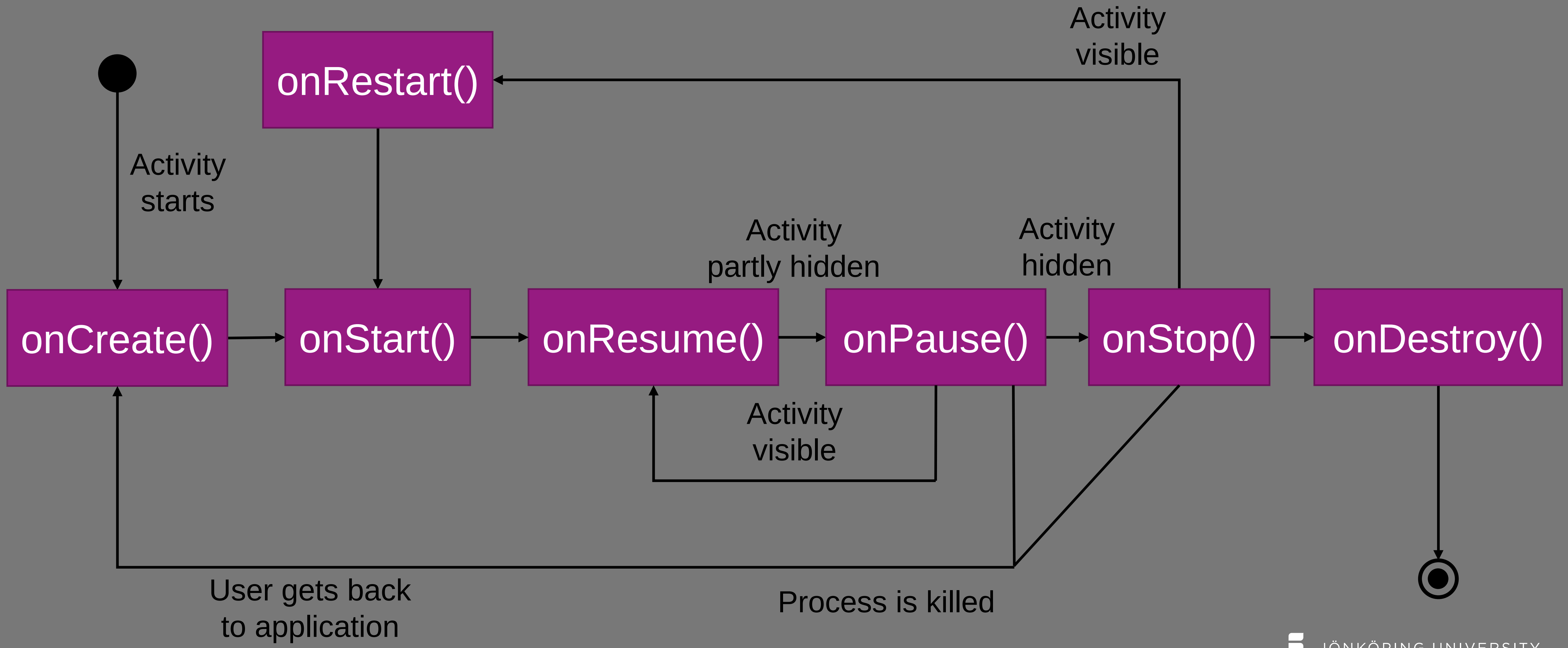
```
public class MyActivity extends Activity{
  @Override
  protected void onActivityResult(int requestCode, int resultCode,
                                                     Intent data){
  }
}
```

RETURNING A RESULT

Somewhere in `PickContactActivity.java`:

```
// When the user has selected a contact:
Intent data = new Intent();
data.putExtra("id", 6);
int resultCode = Activity.RESULT_OK;
this.setResult(resultCode, data);
this.finish();
```

AN ACTIVITY'S LIFE CYCLE


Activity visible
onRestart()
Activity starts
Activity partly hidden
Activity hidden
onCreate()
onStart()
onResume()
onPause()
onStop()
onDestroy()
Activity visible
User gets back to application
Process is killed