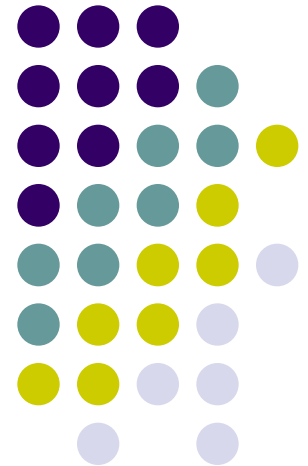


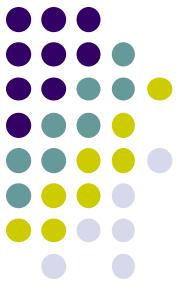
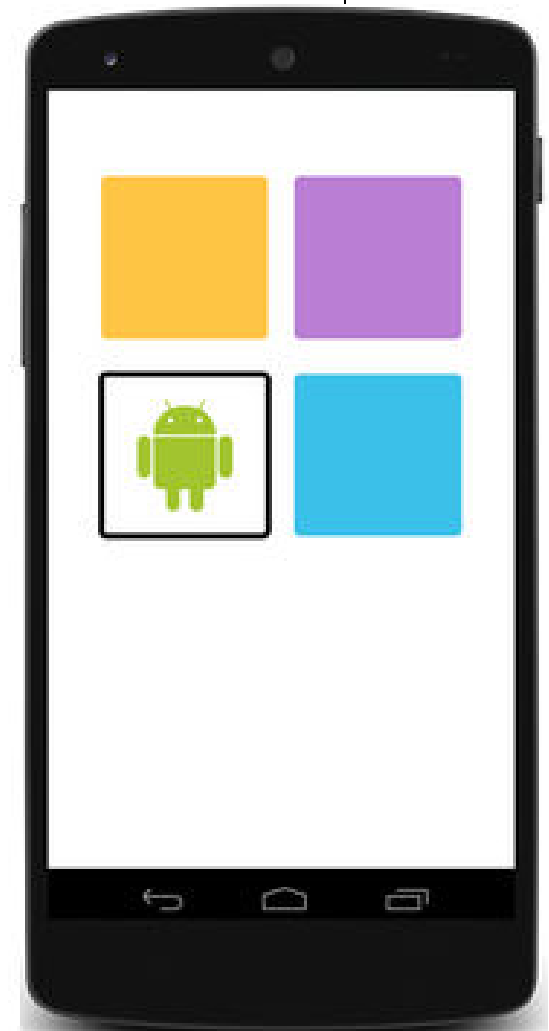
Mobile Application Development

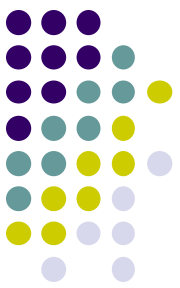
Android Activities



Activity Definition

- An **activity** represents a single screen with a user interface (Java file + xml layout file).
- An **app** usually consists of multiple activities that are loosely bound to each other.
- One **Activity** is flagged as “**main**” and it is started at the application launch time.
- An **Activity** can launch other activities to create an app **UI workflow**.
- The **Activity** has an Intent attribute that determines how android treats it.





Activity

```
public class Activity
extends ContextThemeWrapper implements LayoutInflater.Factory2, Window.Callback, KeyEvent.Callback,
View.OnCreateContextMenuListener, ComponentCallbacks2
```

java.lang.Object

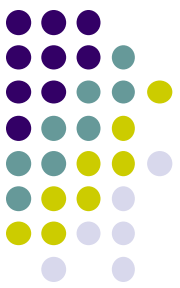
- ↳ android.content.Context
 - ↳ android.content.ContextWrapper
 - ↳ android.view.ContextThemeWrapper
 - ↳ android.app.Activity

▼ Known direct subclasses

AccountAuthenticatorActivity, ActivityGroup, AliasActivity, ExpandableListActivity, ListActivity, NativeActivity

▼ Known indirect subclasses

LauncherActivity, PreferenceActivity, TabActivity



AppCompatActivity

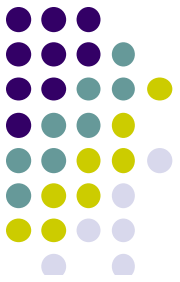
added in version 25.1.0

```
public class AppCompatActivity  
extends FragmentActivity implements AppCompatActivity, TaskStackBuilder.SupportParentable,  
ActionBarDrawerToggle.DelegateProvider
```

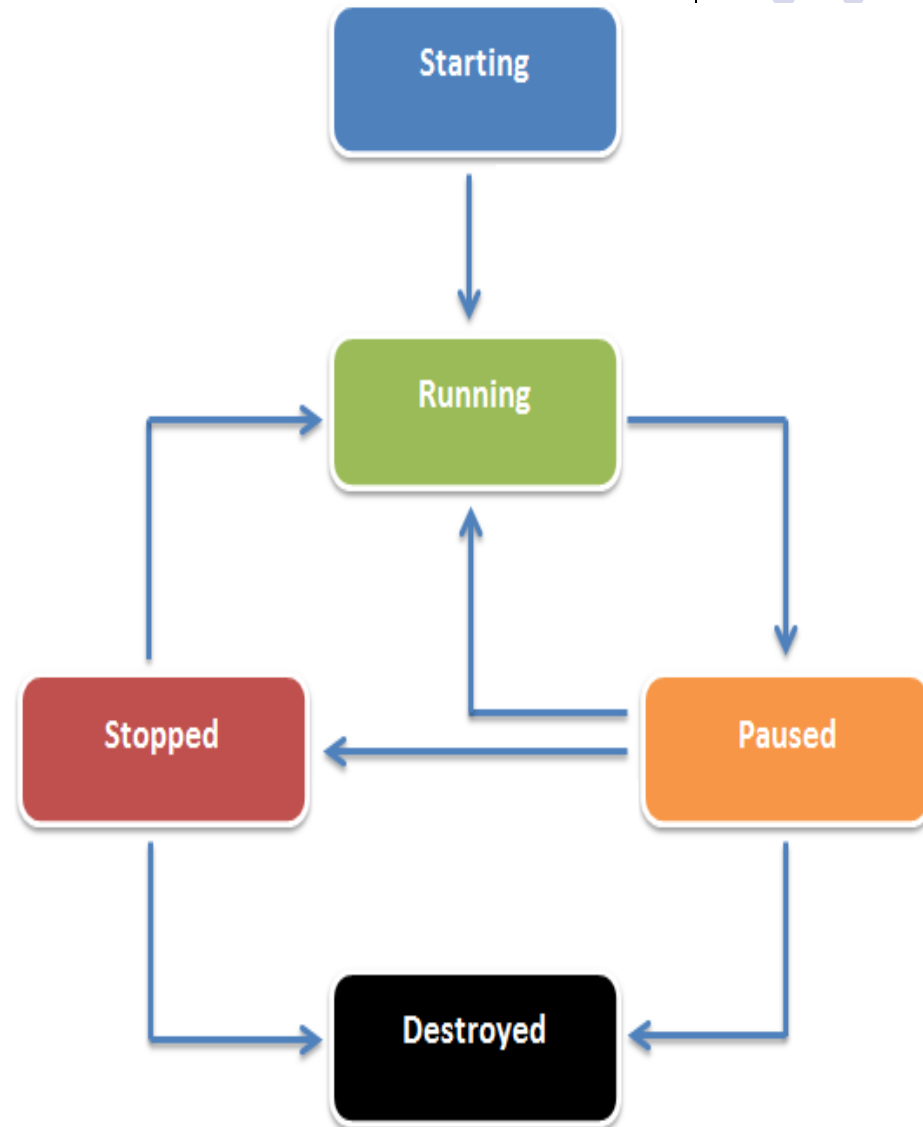
[java.lang.Object](#)

- ↳ [android.content.Context](#)
 - ↳ [android.content.ContextWrapper](#)
 - ↳ [android.view.ContextThemeWrapper](#)
 - ↳ [android.app.Activity](#)
 - ↳ [android.support.v4.app.FragmentActivity](#)
 - ↳ [android.support.v7.app.AppCompatActivity](#)

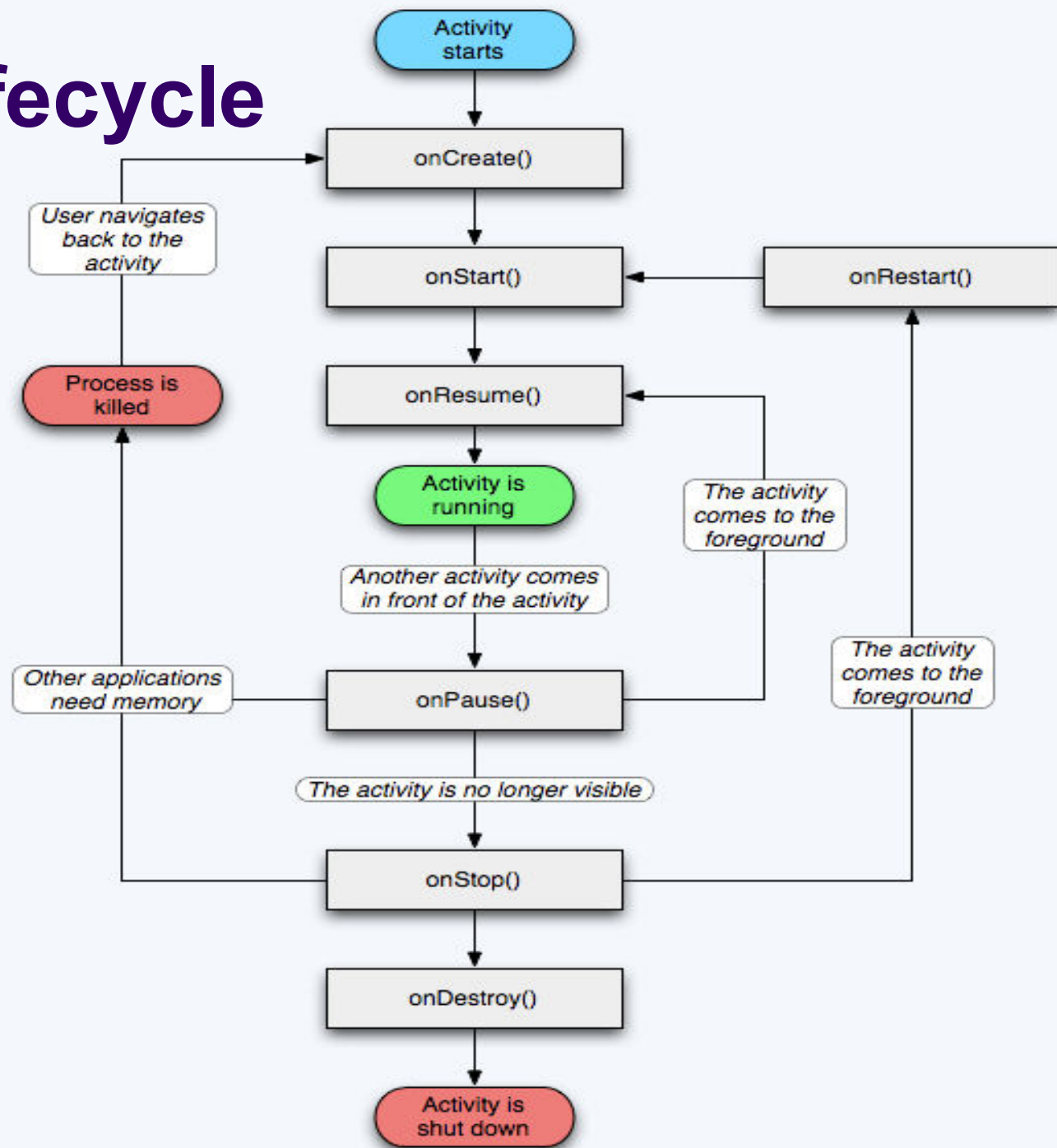
States of an activity



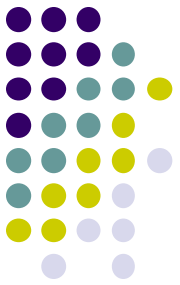
- Starting State
 - initial setup.
- Resumed/Running State
 - visible, user interacting
 - are considered active or running if they are in the foreground.
- Paused State
 - visible, user not interacting,
 - can be terminated*
 - if the device goes to sleep or if it is covered with another Activity partially or completely.
- Stopped State
 - are stopped or in the background with the lowest priority.
 - not visible,
 - can be terminated



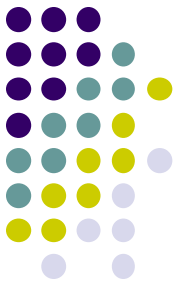
Activity lifecycle



Some Activity Methods

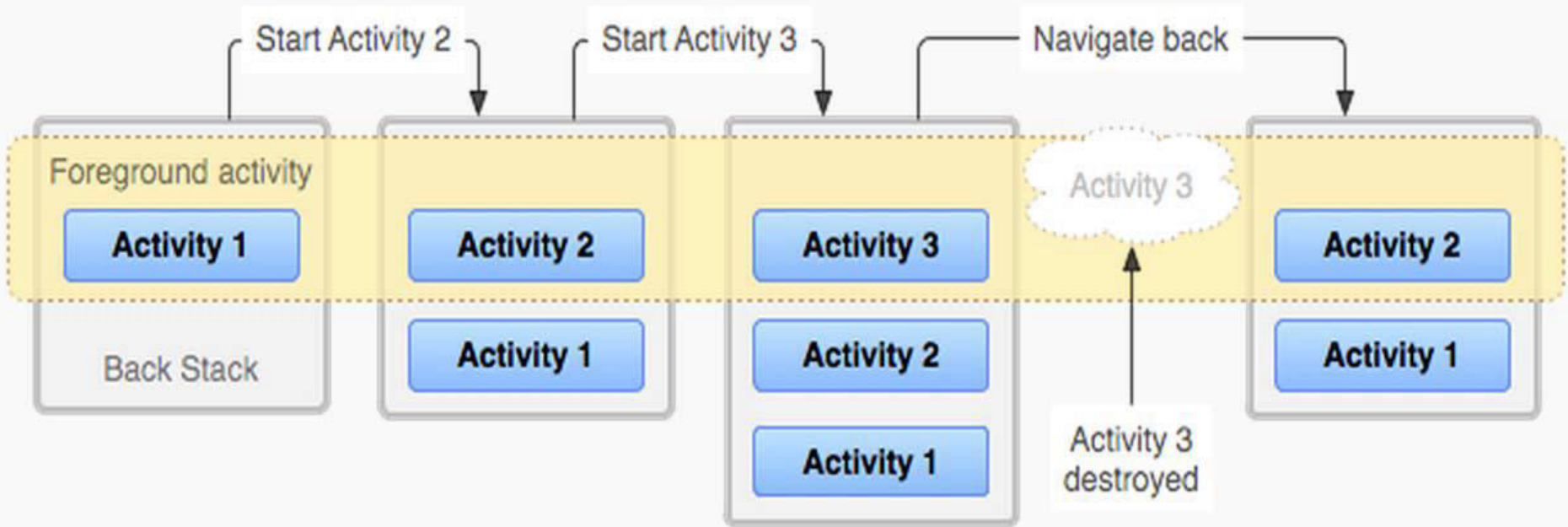


- **onCreate():**
 - is called when an Activity is getting created for the first time.
- **onStart():**
 - used onStart to reset Activity data, reinitialize variables etc.
- **onResume():**
 - gets called when an Activity comes into the foreground, and it becomes visible to the user. At this point, the user can start interacting with the Activity.
- **onPause():**
 - is called when another android activity comes on top of an Activity.
- **onStop():**
 - is called when an Activity is no longer visible to the user, it is similar to onPause but here you will not see your android activity entirely.
- **onRestart():**
 - It is similar to onCreate, but onRestart gets called only after onStop.
- **OnDestroy:**
 - This is the method which will be called when your Activity is getting killed. This is the final call the Activity will receive in its Lifecycle.

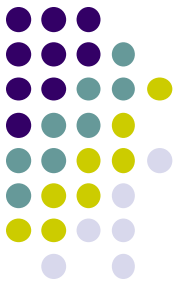


Activity stack

- When a new activity is started, it is placed on the top of the stack and becomes the running activity.
- The previous activity always remains below it in the stack.
- The previous activity will not come to the foreground again until the new activity exits. (“Back stack”)
- Navigation forward/back triggered by user actions



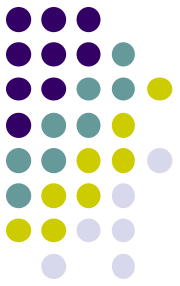
Three different life cycles of an activity



- The **entire lifetime**
 - Starts by first call to `onCreate(Bundle)` and ends by a single final call to `onDestroy()`.
- The **visible lifetime**
 - Starts by a call to `onStart()` and ends by a corresponding call to `onStop()`. During this time the user can see the activity on-screen, though it may not be in the foreground and interacting with the user.
- The **foreground lifetime**
 - Starts by a call to `onResume()` and ends a corresponding call to `onPause()`. During this time the activity is in front of all other activities and interacting with the user. An activity can frequently go between the resumed and paused states.

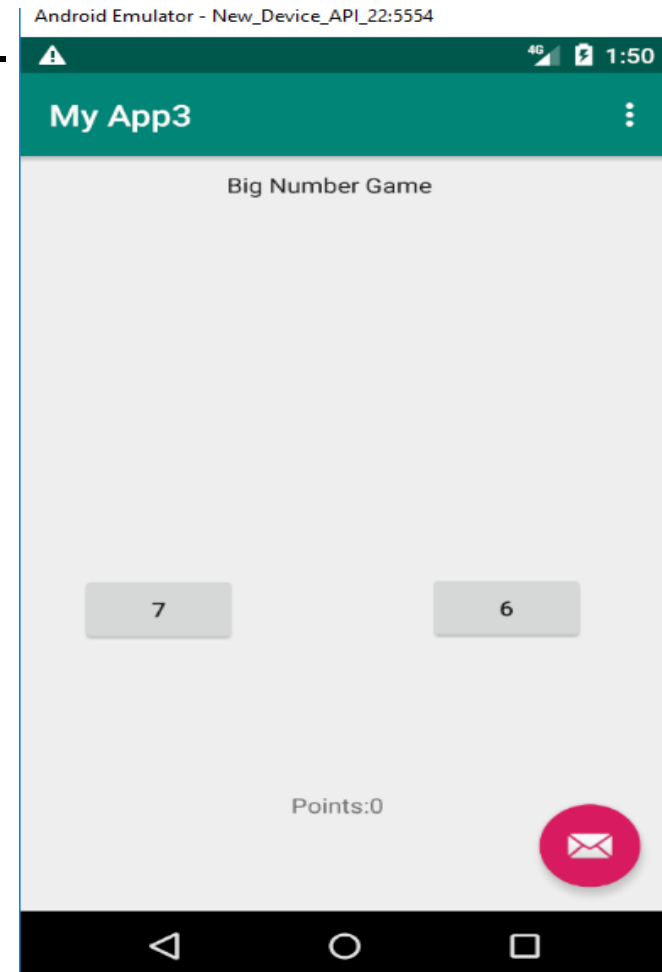


Top-down design

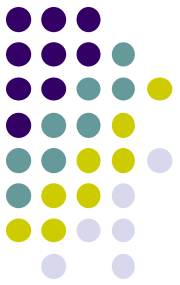


Let's start from a design of an app that we want to create and then learn the necessary skills to build that app.


- "Bigger Number" game
 - user is shown two numbers
 - must choose which one is bigger by clicking on the appropriate button
 - game pops up brief "correct" / "incorrect" message after each guess
 - get points for each correct answer (lose points for incorrect answers)



Creating a new project



Create New Project

 **New Project**
Android Studio

Configure your new project

Application name:

Company Domain:

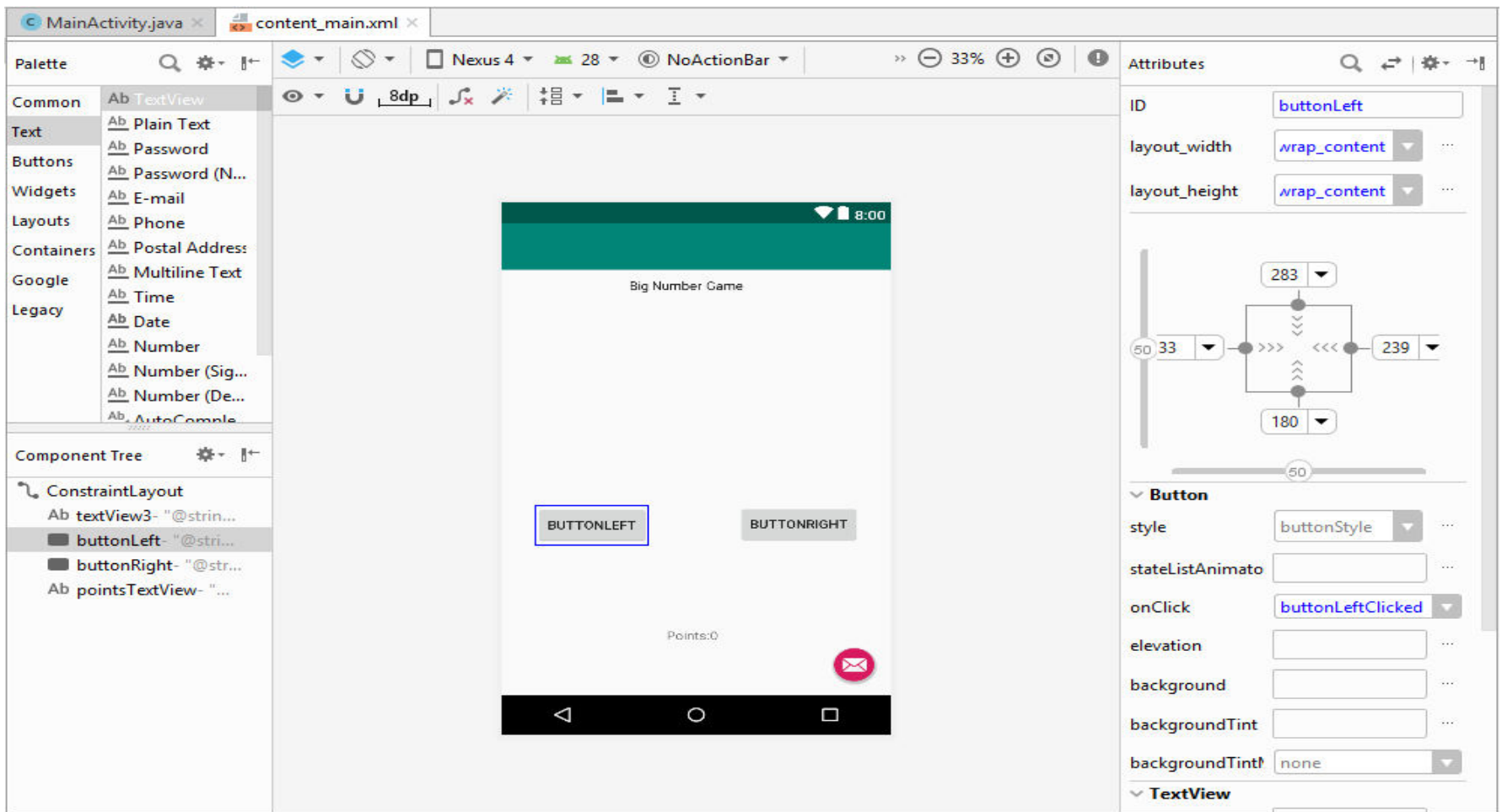
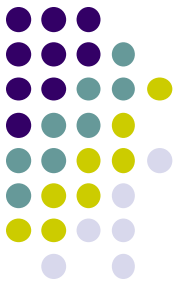
Package name: [Edit](#)

Project location:

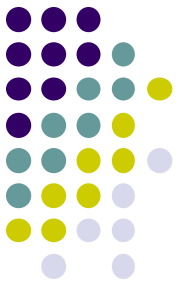
Designing a user interface

open XML file for your layout (e.g. activity_main.xml)

- drag widgets from left **Palette** to the preview image
- set their properties in lower-right **Properties** panel



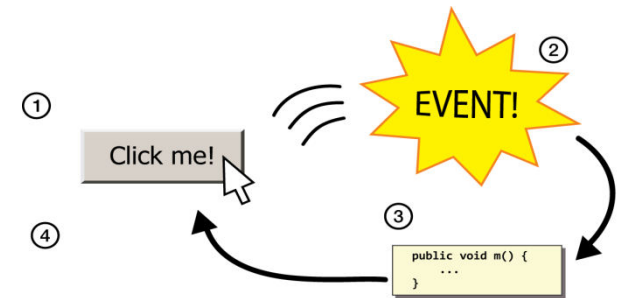
Events



- **event: An external stimulus your program can respond to.**

- Common kinds of events include:

- Mouse motion / tapping, Keys pressed,
- Timers expiring, Network data available



- **event-driven programming: Overall**

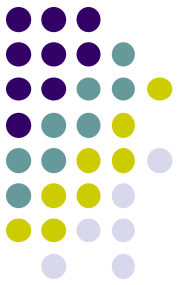
execution of your program is largely dictated by user events.

- Commonly used in graphical programs.

- To respond to events in a program, you must:

- Write methods to handle each kind of event ("listener" methods).
- Attach those methods to particular GUI widgets.

Setting an event listener



- select the widget in the Design view
- scroll down its Properties until you find onClick
- type the name of a method you'll write to handle the click
- switch to the Text view and find the XML for that button
- click the "Light Bulb" and choose to "Create" the method

The screenshot displays the Android Studio IDE with three main components visible:

- Design View:** Shows a visual representation of the UI. Two buttons are present, labeled "BUTTONLEFT" and "BUTTONRIGHT". The "BUTTONLEFT" button is selected, indicated by a blue border.
- Properties Panel:** Located on the right, it shows the properties for the selected "Button" widget. The "onClick" property is set to "buttonLeftClicked".
- XML Editor:** The bottom pane shows the XML code for the selected button. The XML is as follows:

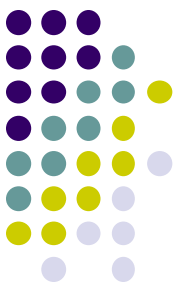
```
<Button
    android:id="@+id/buttonLeft"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="33dp"
    android:layout_marginTop="283dp"
    android:layout_marginEnd="239dp"
    android:layout_marginBottom="180dp"
    android:onClick="buttonLeftClicked"
    android:text="@string/buttonleft"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"/>
```

The line containing `android:onClick="buttonLeftClicked"` is highlighted in yellow, and a lightbulb icon is visible in the left margin, indicating an IDE suggestion or action.

Additionally, a snippet of Java code is visible in the background, showing the implementation of the `buttonLeftClicked` method:

```
/*
 * Called when the player clicks the left number button.
 */
public void buttonLeftClicked(View view) {
    check(num1, num2);
}
```

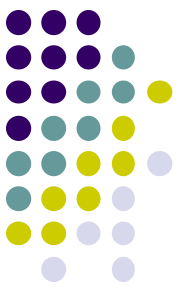
Event listener Java code



MainActivity.java x

content_main.xml x

```
1 package com.example.i7ec.myapp3;
2
3 import ...
16
17 public class MainActivity extends AppCompatActivity {
18
19     private int num1;    // the numbers on the left and right buttons
20     private int num2;
21     private int points=0; // player's point total; initially 0
22
23     /*
24      * Called when the player clicks the left number button.
25      */
26     public void buttonLeftClicked(View view) {
27         check(num1, num2);
28     }
```

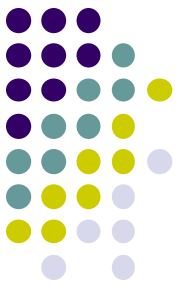


Generated Java code

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
    roll();    // <-- we added this line to set initial button random numbers

    FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
    fab.setOnClickListener((view) → {
        Snackbar.make(view, text: "Replace with your own action", Snackbar.LENGTH_LONG)
            .setAction(text: "Action", listener: null).show();
    });
}
```


This Is the Toast message



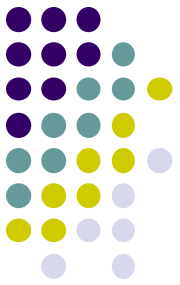
Displaying Toasts

`Toast.makeText(this, "message", duration).show();`

– where *duration* is `Toast.LENGTH_SHORT` or `LENGTH_LONG`

- A "Toast" is a pop-up message that appears for a short time.
- Useful for displaying short updates in response to events.
- Should not be relied upon extensively for important info.

```
C MainActivity.java × content_main.xml ×
37      /*
38      * Updates the player's score based on whether they guessed correctly.
39      * Also shows a 'toast' which is a brief popup message.
40      */
41      private void check(int a, int b) {
42          if (a > b) {
43              points++;
44              Toast.makeText(context: this, text: "Correct!",
45                          Toast.LENGTH_SHORT).show();
46          } else {
47              points--;
48              Toast.makeText(context: this, text: "You are Wrong.", Toast.LENGTH_SHORT).show();
49          }
50
51          TextView pointsView = (TextView) findViewById(R.id.pointsTextView);
52          pointsView.setText("Points: " + points);
53          roll();
54      }
```



References

- Activity class
 - <https://developer.android.com/reference/android/app/Activity.html#Activity>
- **Getting Started**
 - <https://developer.android.com/training/index.html>

