Lab on apps development for tablets, smartphones and smartwatches

Week 4: Application Lifecycle & User interface

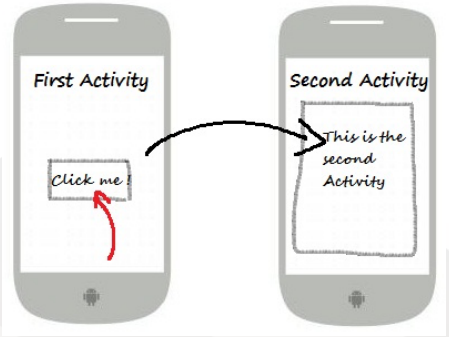
Dr. Marina Zapater, Prof. David Atienza

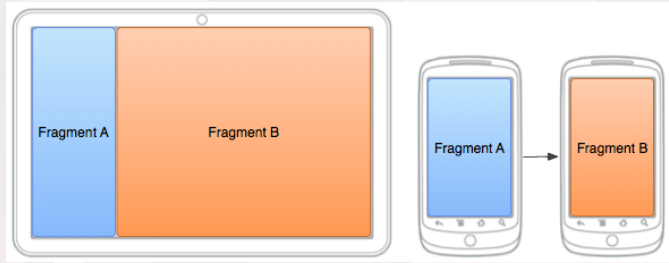
Mr. Grégoire Surrel, Ms. Elisabetta de Giovanni, Mr. Dionisijie Sopic,
Ms. Halima Najibi, Ms. Farnaz Forooghifar

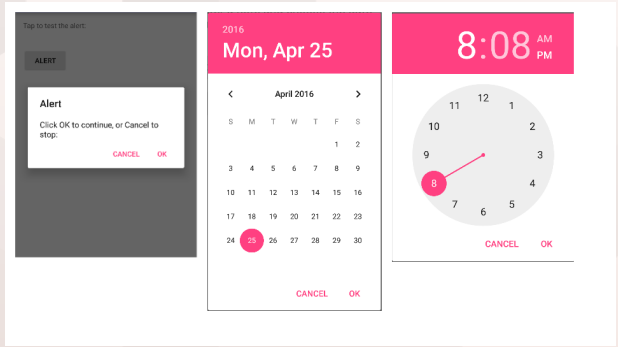
Embedded Systems Laboratory (ESL) – Faculty of Engineering (STI)

Outline of the class

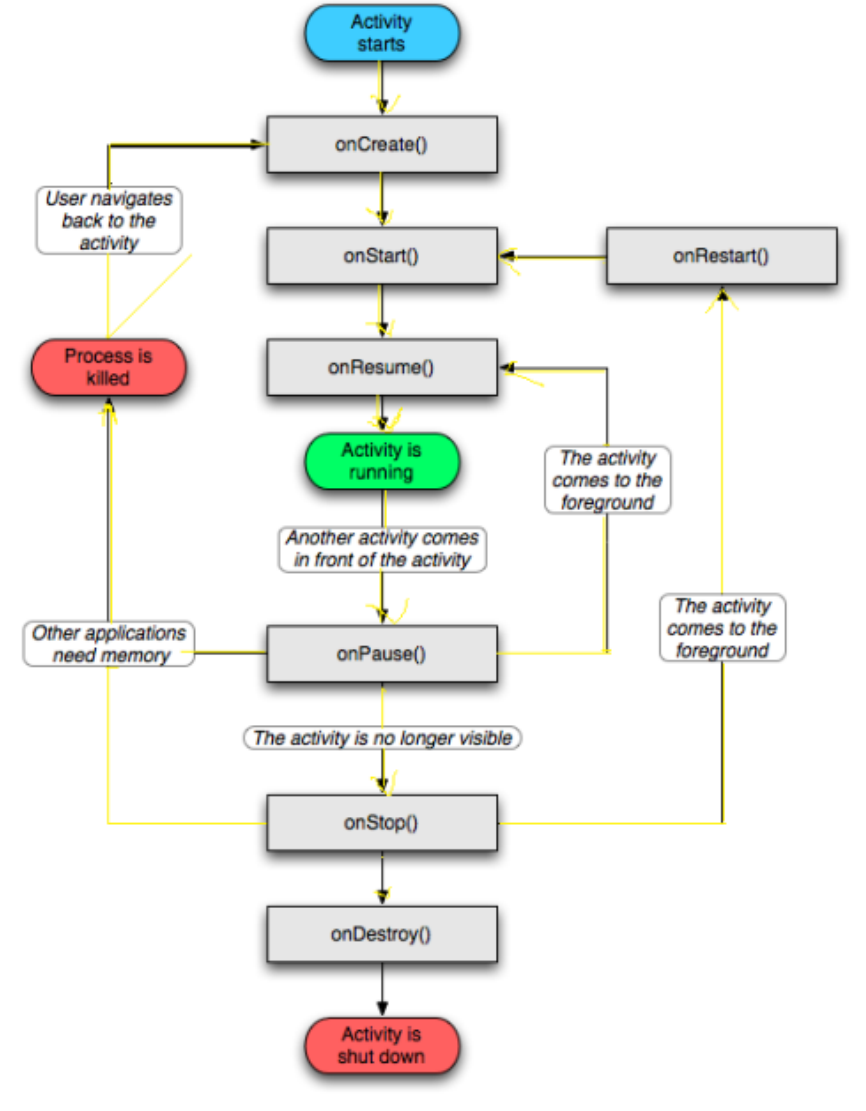
- **Application lifecycle**
  - Application lifecycle
  - Saving and restoring activity state
- Fragments
- User interaction:
  - Lists
  - Dialogs
  - Toasts
  - Menus





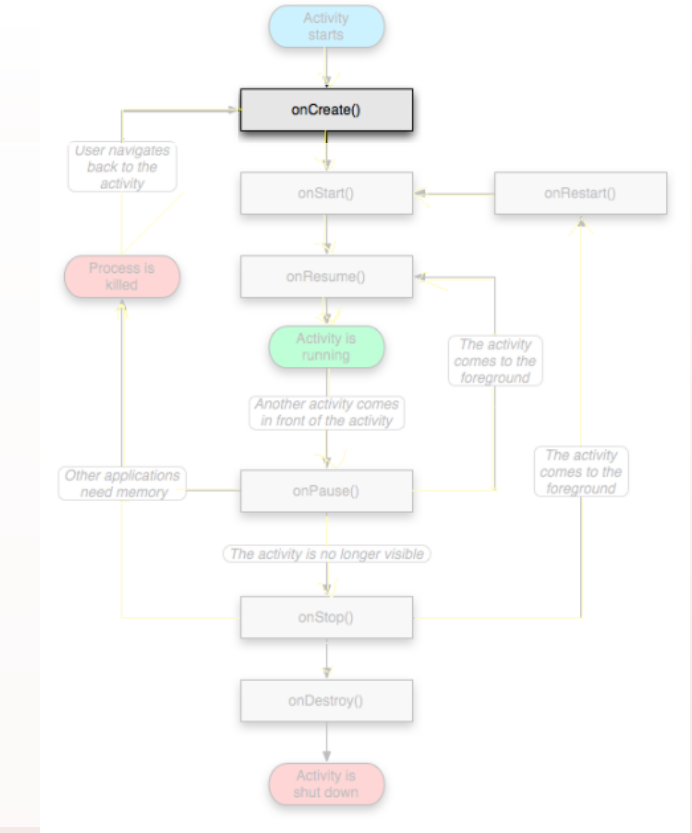


Activity lifecycle



- What is the activity lifecycle?
  - A set of states an activity can be in during its lifetime, from its creation to its destruction.
- More formally:
  - A directed graph of all the states an activity can be in, and the callbacks associated with transitioning from each state to the next one

Activity lifecycle

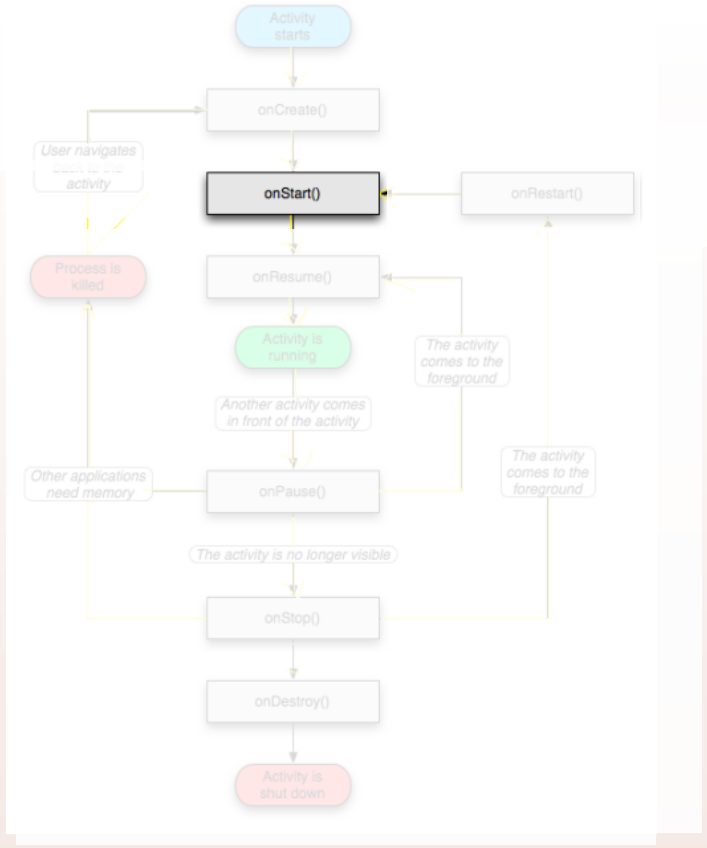


- onCreate()
  - Called when the activity is created
    - for example, we tap launcher icon
  - Does all static setup:
    - create views, bind data to lists, …
  - Only called once during lifetime
  - Should contain the initialization operations
  - Takes a Bundle with all the activities previous state, if it exists
  - If succesfull, the activity is created but not visible, and we call onStart()

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
   //The activity is being created

}
```

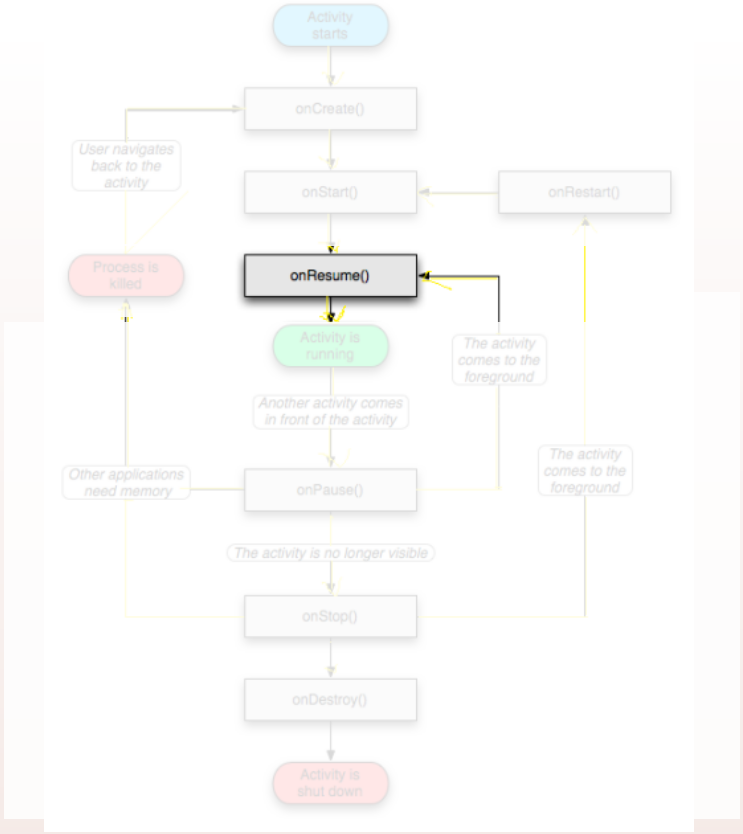
Activity lifecycle



- onStart()
  - Called when onCreate() terminates
  - Called right before activity is visible to user
  - Followed by onResume(), if the activity comes to the foreground, or onStop(), if it becomes hidden

```
@Override
protected void onStart() {
    super.onStart();
    //The activity is about to become visible
}
```

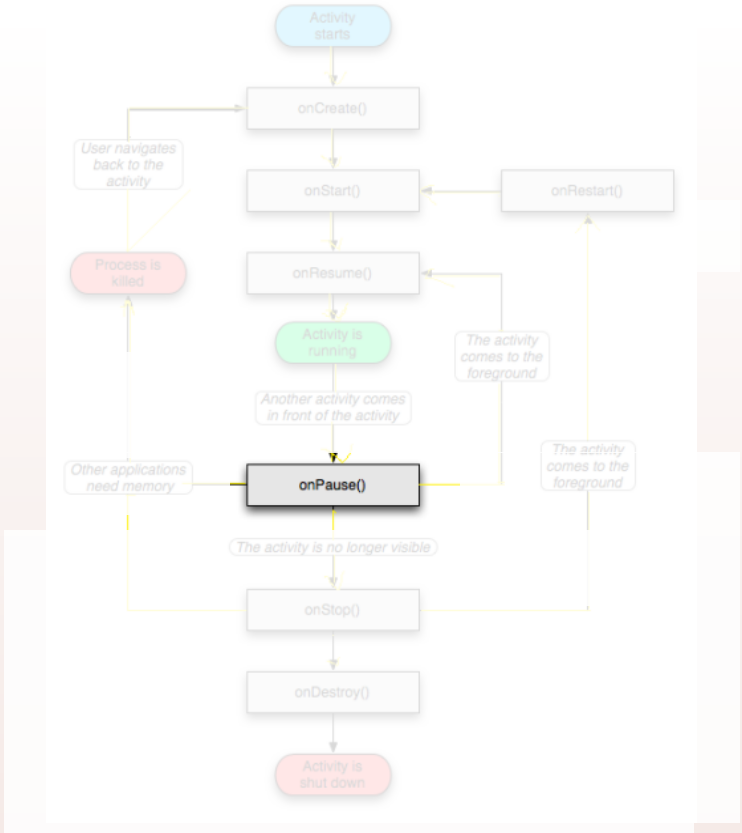
Activity lifecycle



- onResume()
  - Called when the activity is ready to get input from users
  - Called when the activity is resumed too
  - Activity has moved to the top of the activity stack
  - If it successfully terminates, then the Activity is RUNNING
  - Always followed by onPause()

```
@Override
protected void onResume() {
    super.onResume();
    //The activity is visible and resumed
}
```

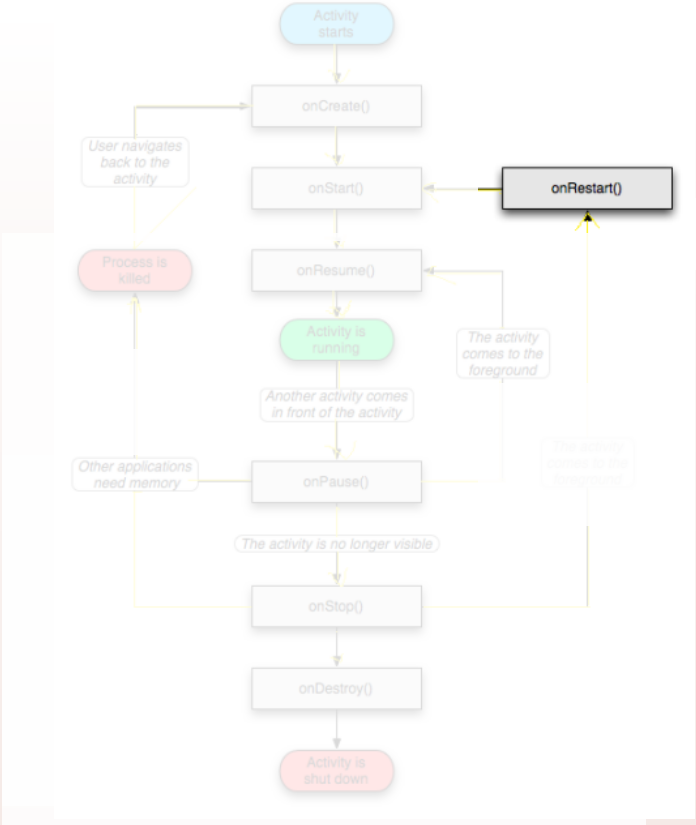
Activity lifecycle



- onPause()
  - Called when another activity comes to the foreground, or when someone presses the back button
  - Used to commit unsaved changes to persistent data
  - Stop CPU-consuming processes
  - Make it fast, as the next activity cannot resume until this method runs
  - Followed by onResume() or onStop()

```
@Override
protected void onPause() {
    super.onPause();
    //Another activity is taking focus
    //this one will be paused
}
```

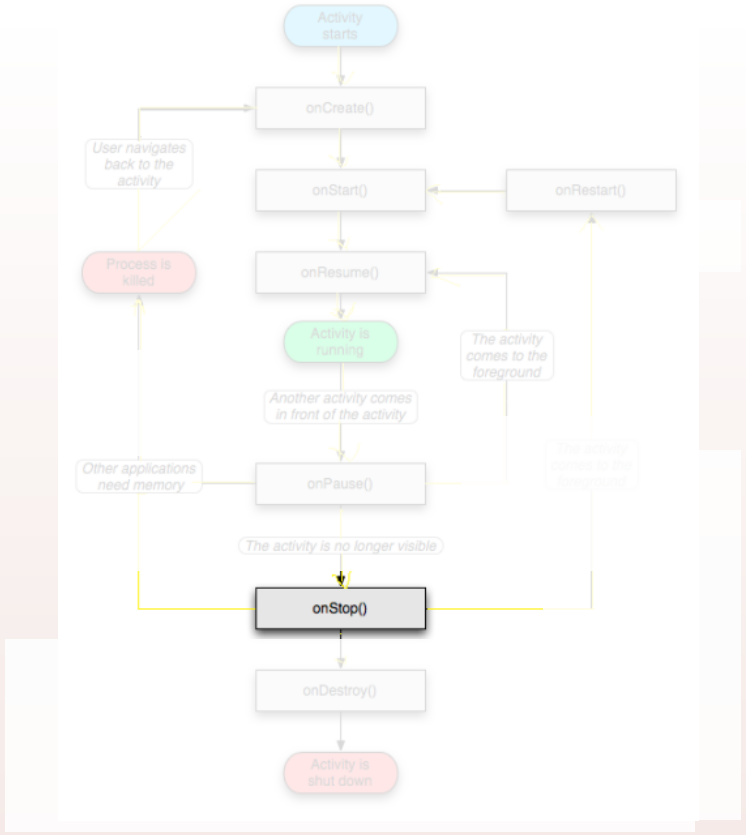
Activity lifecycle



- onRestart()
  - Similar to onCreate()
  - Called after activity has been stopped, inmediately before it is started again
  - Transient state
  - Always followed by onStart()

```
@Override
protected void onRestart() {
    super.onRestart();
    //Activity between stop and start
}
```

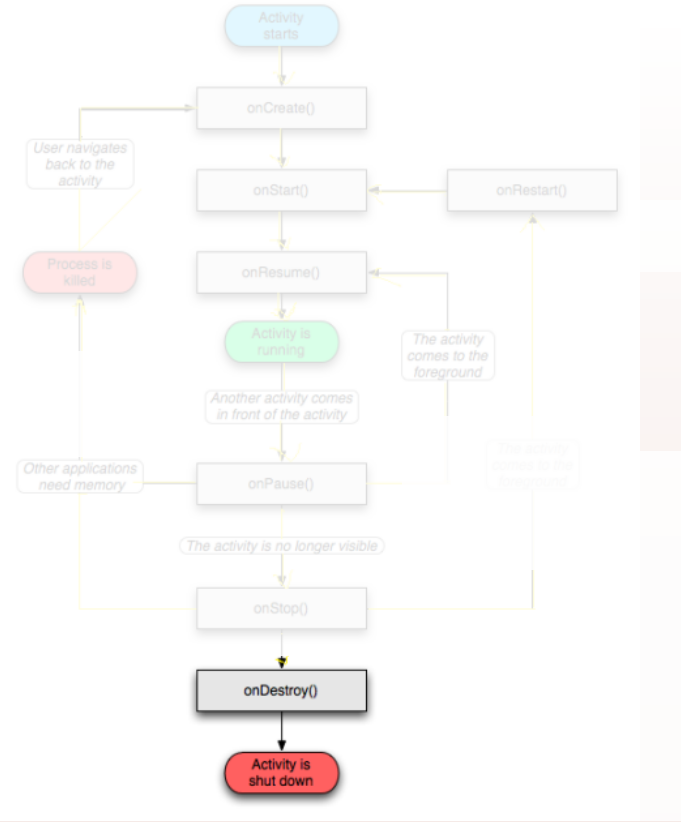
Activity lifecycle



- onStop()
  - Activity is no longer visible to the user
  - Could be called because:
    - the activity is about to be destroyed
    - another activity comes to the foreground
    - operations that were too heavy for onPause()
  - Followe by either onRestart() if we are going to interact with user, or onDestroy() if it is going away

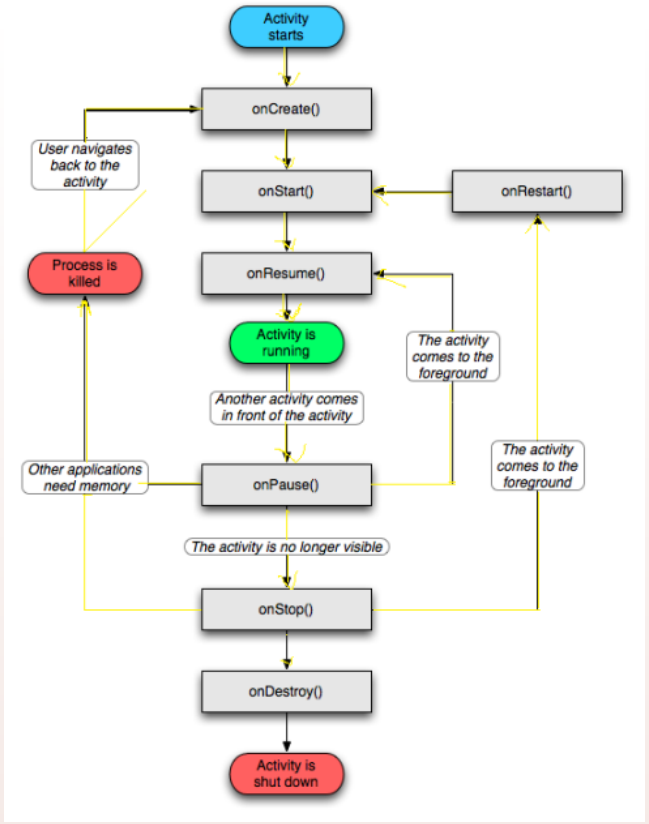
```
@Override
protected void onStop() {
    super.onStop();
    //Activity is now stopped
}
```

Activity lifecycle



- onDestroy()
  - The activity is about to be destroyed (final call before destruction)
  - Could happen because:
    - User navigates to previous activity, or configuration changes
    - Someone called finish() method on this activity
    - The Android system need some stack space
  - We can check with isFinishing()
  - The system may destroy activity without calling this function
    - Save data on onPause() or onStop()

Activity loops

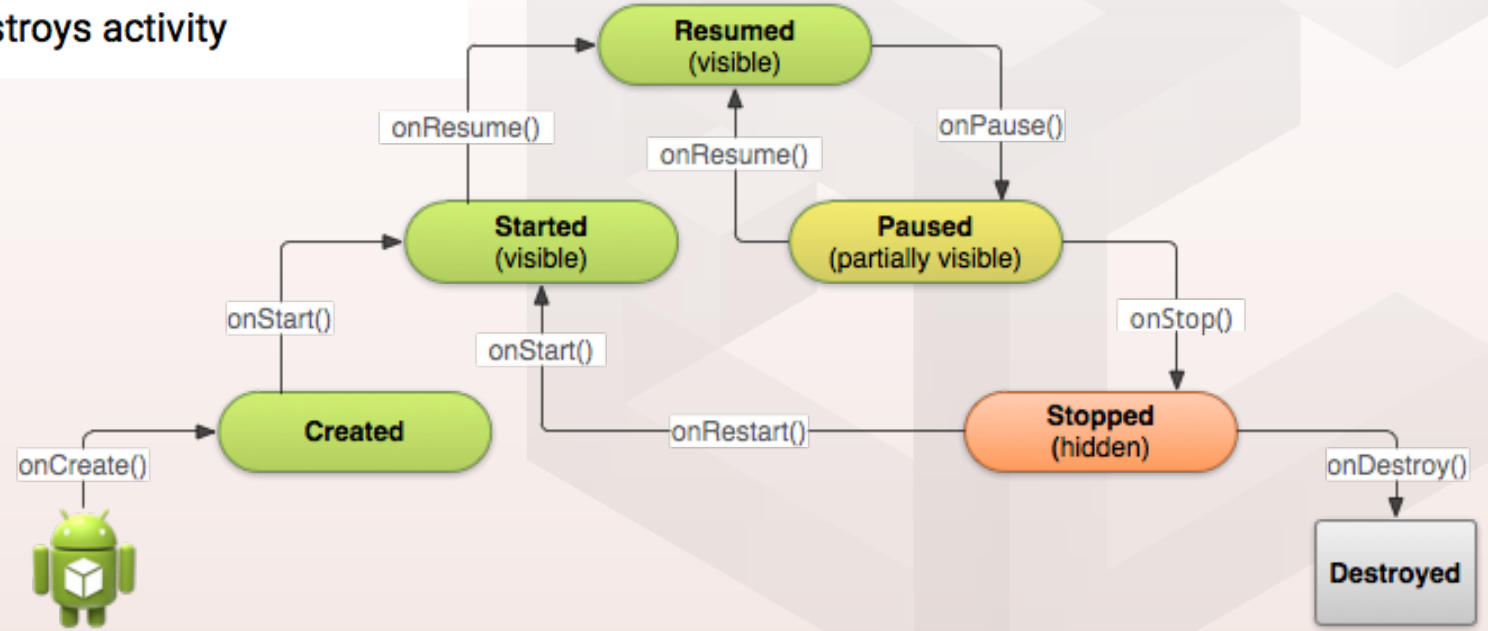


Mainly 3 different loops

- **Entire lifetime**
  - Between onCreate() and onDestroy().
  - Setup of global state in onCreate()
  - Release remaining resources in onDestroy()
- **Visible lifetime**
  - Between onStart() and onStop().
  - Maintain resources that have to be shown to the user.
- **Foreground lifetime**
  - Between onResume() and onPause().
  - Code should be light.

States and callbacks graph

- **onCreate(Bundle savedInstanceState)**—static initialization
  - **onStart()**—when activity (screen) is becoming visible
  - **onRestart()**—called if activity was stopped (calls onStart())
    - **onResume()**—start to interact with user
    - **onPause()**—about to resume PREVIOUS activity
  - **onStop()**—no longer visible, but still exists and all state info preserved
- **onDestroy()**—final call before Android system destroys activity



Activity instance state

- When does configuration change?
  - Configuration changes invalidate the current layout or other resources when the user:
    - Rotates the device
    - Chooses different system language
- What happens on a config change?
  - On a configuration change, Android:
    1. Shuts down activity calling: onPause() → onStop() → onDestroy()
    2. Then starts it over calling: onCreate() → onStart() → onResume()
- State information is created while the activity runs:
  - Counter, user text, animation progression
- State is lost when there is a configuration change!
  - Device rotate, back-button pressed…

Saving (and restoring) activity state

- You are responsible for saving activity and user progress data!
  - System only saves: state of views with unique ID (android:id) such as text entered into EditText
- To save data, implement onSaveInstanceState() in your activity
  - Called by Android runtime where the activity can be destroyed
  - Saves data only for this instance of the activity during the current session

```
@Override
protected void onSaveInstanceState (Bundle outState){
    super.onSaveInstanceState(outState);
    //Add information for saving something (i.e., a counter)
    // to the outState bundle
    outState.putString("count", String.valueOf(mShowCount.getText()));
}
```

Restoring Activity State

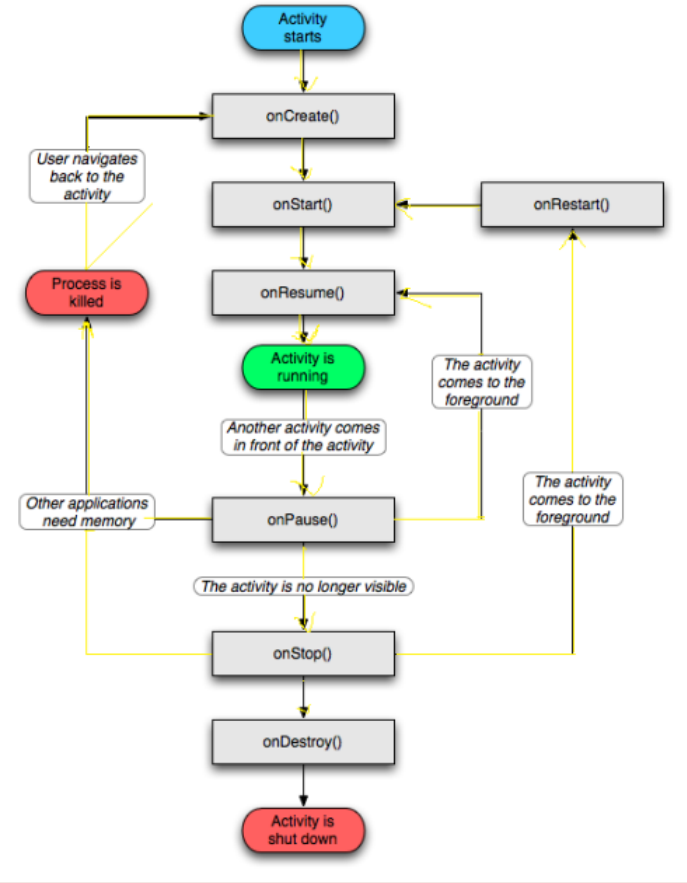
- Two ways to retrieve the saved bundle:
  - in onCreate(Bundle mySavedState)
    - Preferred method, to ensure that your user interface is back up and running as quickly as possible
  - Implement callback (called after onStart()):
    - onRestoreInstanceState (Bundle mySavedState)

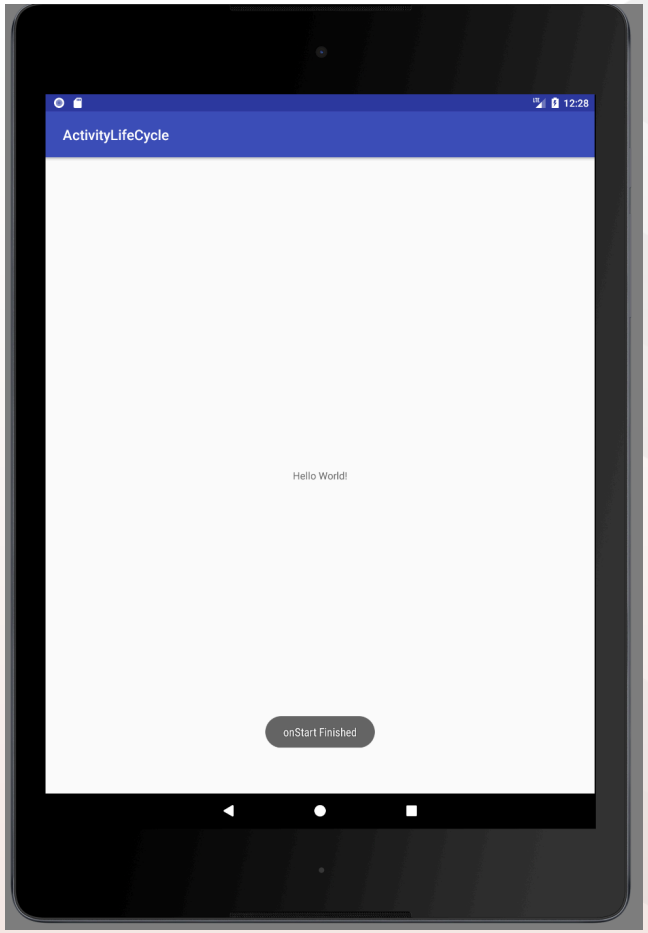
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mShowCount = (TextView) findViewById(R.id.show_count);
}

@Override
protected void onSaveInstanceState (Bundle outState){
    super.onSaveInstanceState(outState);
    //Add information for saving something (i.e., a counter)
    // to the outState bundle
    outState.putString("count", String.valueOf(mShowCount.getText()));
}
```

An example, please!

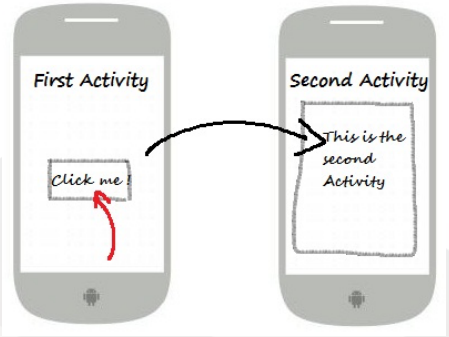
- In Moodle you'll find a code named "Application lifecycle example"
  - A simple "Hello World": shows messages in each method called

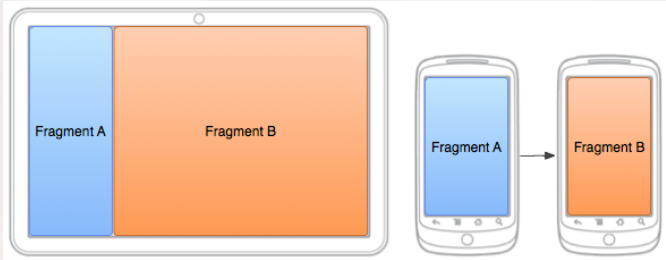


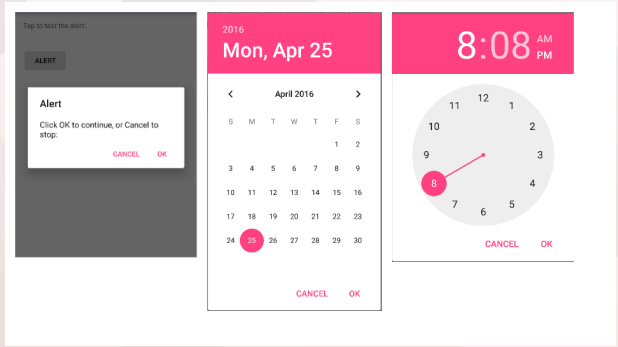


Outline of the class

- Application lifecycle
  - Application lifecycle
  - Saving and restoring activity state
- **Fragments**
- User interaction:
  - Lists
  - Dialogs
  - Toasts
  - Menus



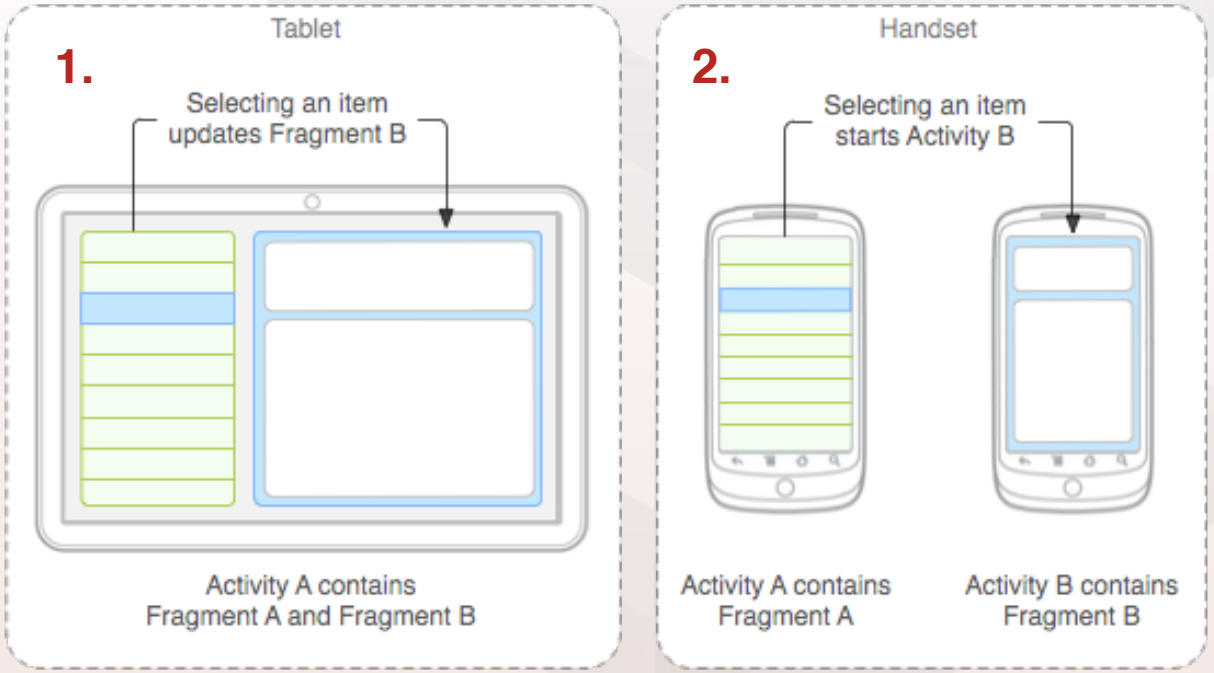




What is a Fragment?

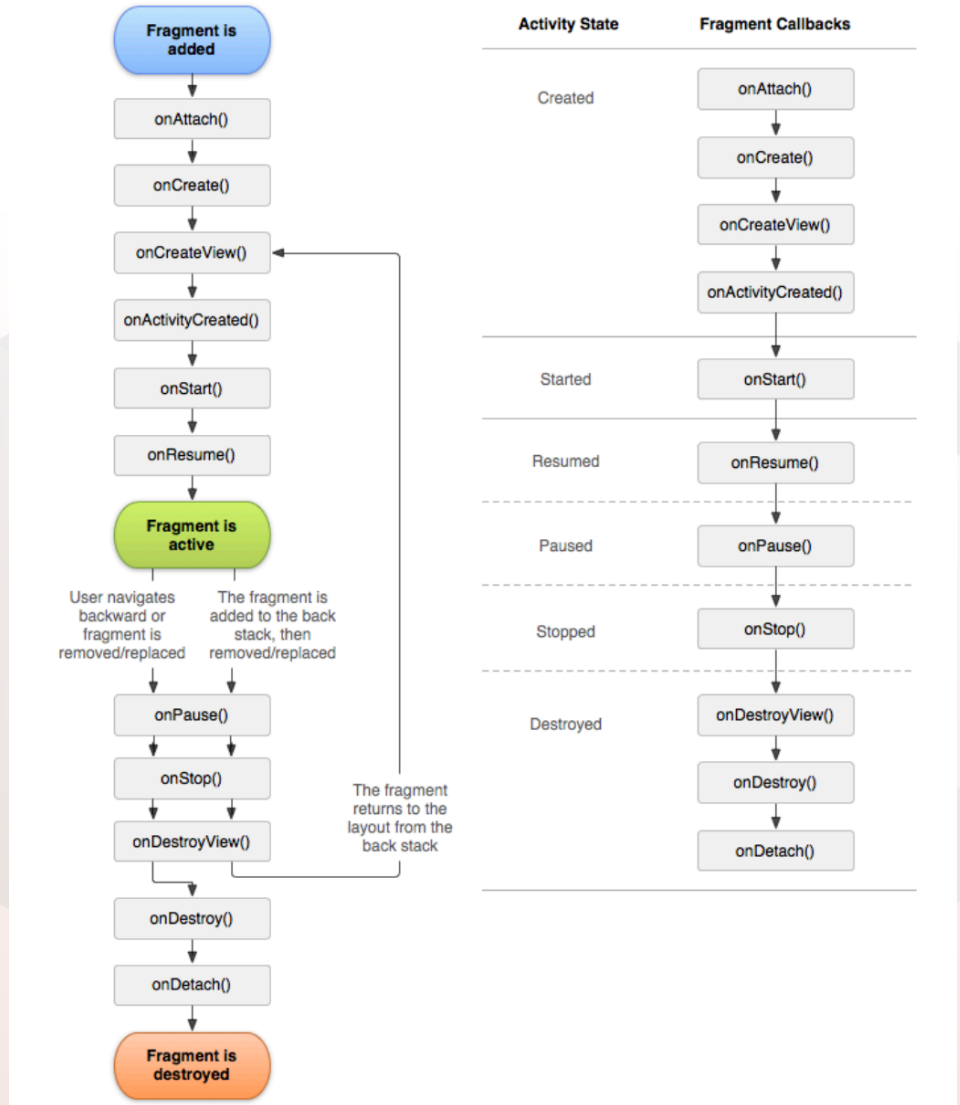
- A **Fragment** is a behavior or portion of a user activity
  1. Modular section of an activity: multiple fragments can be combined into a single activity in a multi-pane UI
  2. Fragments can be reused in more than one activity

- Introduced in Android 3 (API 11) to support more dynamic and flexible UIs



Lifecycle of a Fragment

- A fragment has its own lifecycle
  - But its lifecycle is affected by the activity
    - When the activity is paused, fragments are paused
    - When the activity is destroyed, fragments are destroyed
- Usually, we should implement:
  - onCreate() → initialize essential components
  - onCreateView() → called when it draws the UI for the first time (returns a View)
  - onPause() → to commit changes



Subclasses of Fragments

- Instead of extending from the Fragment class, we can extend from a different subclass, to have a specific functionality:
  - Subdialog Fragment: displays a floating dialog.
  - ListFragment: Displays a list of items managed by an Adapter
  - PreferenceFragment: hierarchy of preference objects (useful when crearing a “Settings” activity for your app).

Adding a fragment to an activity

- Two ways of adding a fragment to an activity:
  1. Via the activity XML file
     - The <android:name> class specifies the subclass to instantiate
     - Each fragment requires a unique identifier that the system can use to restore the fragment if the activity is restarted.
       - android:id or android:tag
       - or the id of the container view

1.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name="com.example.news.ArticleListFragment"
            android:id="@+id/list"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
    <fragment android:name="com.example.news.ArticleReaderFragment"
            android:id="@+id/viewer"
            android:layout_weight="2"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
</LinearLayout>
```

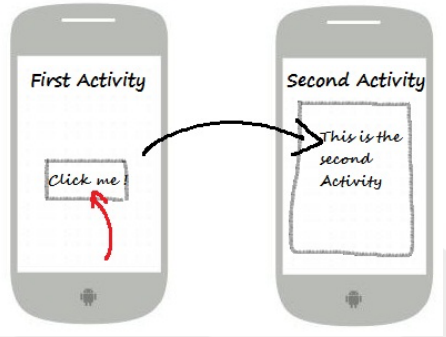
  2. Programmatically, adding the fragment to a ViewGroup
     - Using the FragmentTransation to add/remove/replace a Fragment

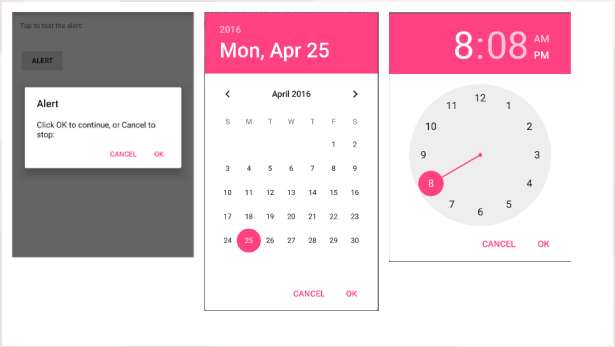
```
ExampleFragment fragment = new ExampleFragment();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.commit();

FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
```

Outline of the class

- Application lifecycle
  - Application lifecycle
  - Saving and restoring activity state
- Fragments
- **User interaction:**
  - Lists
  - Dialogs
  - Toasts
  - Menus



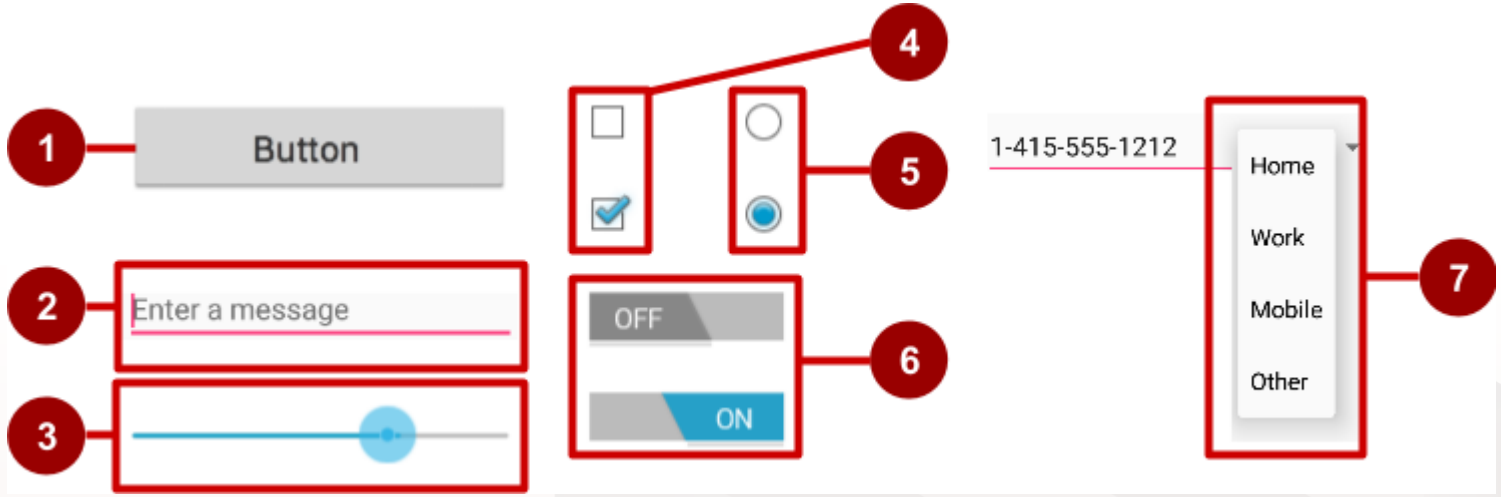


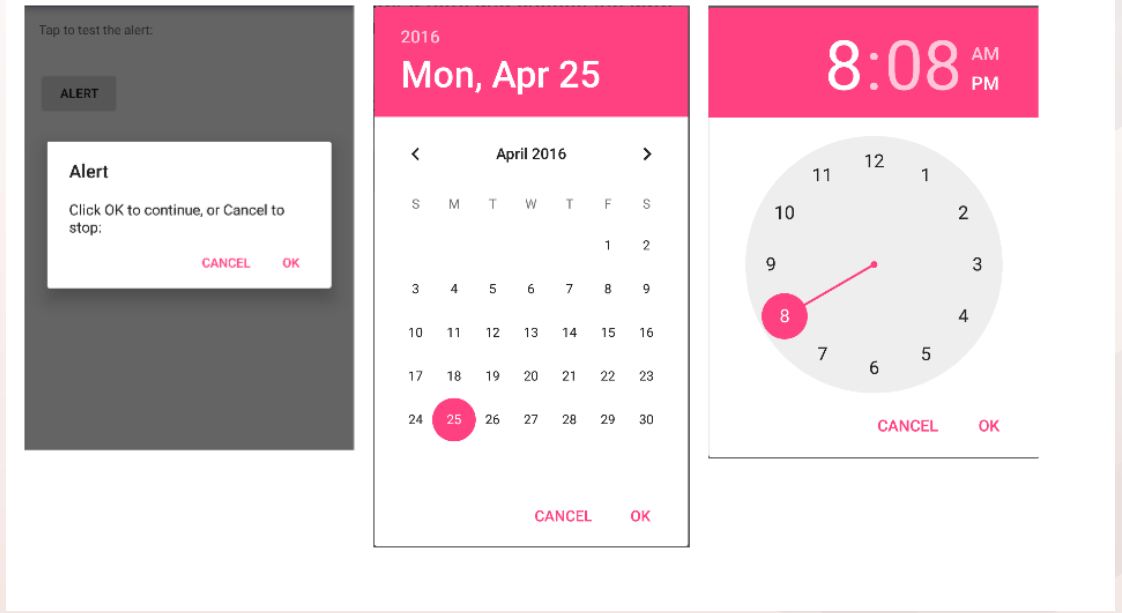
User interaction

- User input controls:

1. Button
2. Text field
3. Seek bar
4. Checkboxes
5. Radio buttons
6. Toggle
7. **Spinner**

- Dialogs:
  - Alert dialog
  - Date picker
  - Time picker

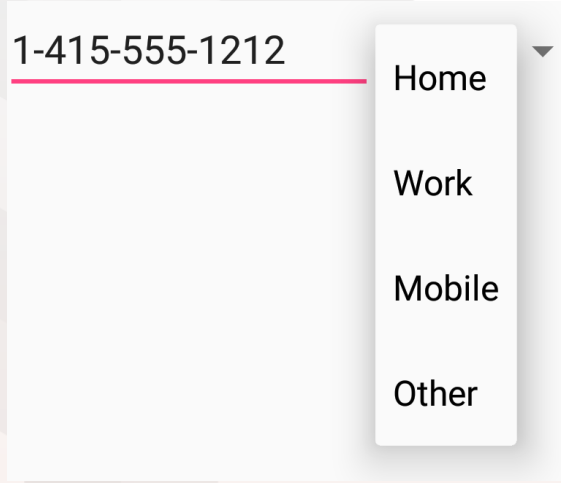




Spinners

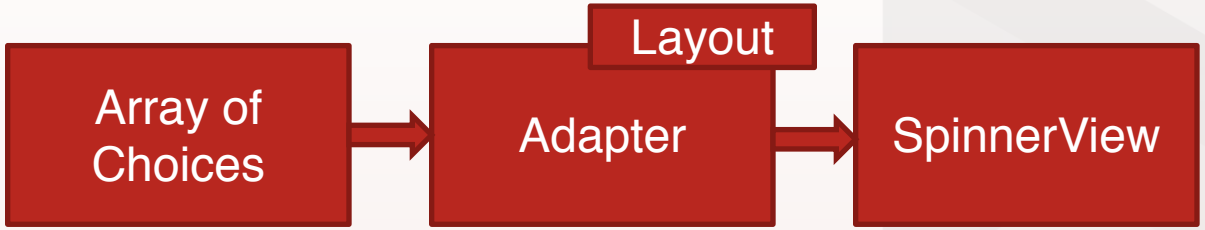
- Spinner: Quick way to select value from a set
  - Drop-down list of all values, users can select only one

- Implementing Spinners:
  1. Create Spinner UI element in the XML layout
  2. Define spinner choices in an array
  3. Create Spinner and set onItemSelectedListener
  4. Create an **adapter** with default spinner layouts
  5. Attach the adapter to the spinner
  6. Implement onItemSelectedListener method



What is an adapter?

- An **adapter** is like a bridge between two incompatible interfaces
- When the content for your layout is dynamic or not pre-determined, the items are automatically inserted to the layout using an adapter
  - It pulls content from a source such as an array or database query and converts each item result into a view that's placed into the layout



- An adapter is a ViewGroup subclass. Some of them are:
  - Spinner
  - ListView and GridView
  - Gallery

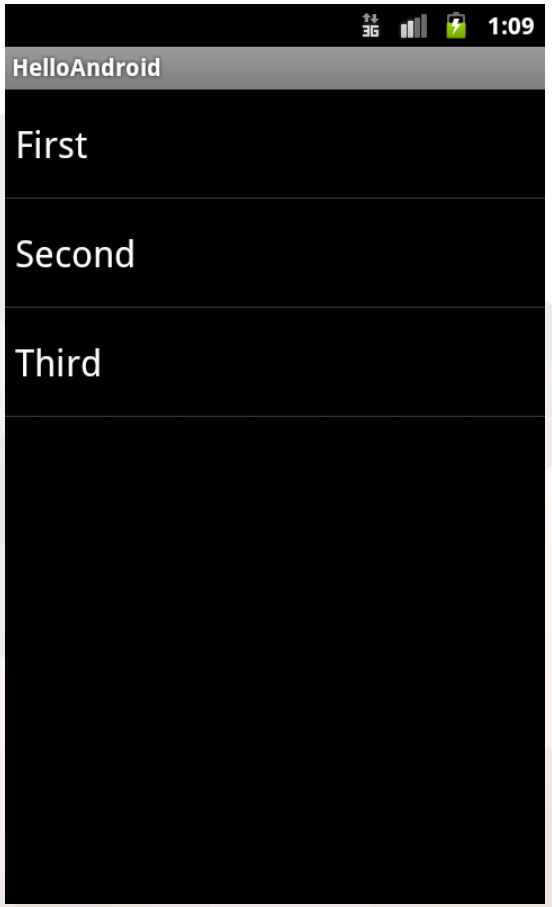
ListView example (can useFragments)

- We will exercise this in today's Lab4

```
<ListView android:id="@id/android:list"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"/>
```

```
public class HelloAndroidActivity extends Activity {
…
   @Override
   public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.list);

     String[] data = {"First", "Second", "Third"};
     ListView lv = (ListView)findViewById(R.id.list);
       lv.setAdapter(new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, data));
   }
}
```



Dialogs

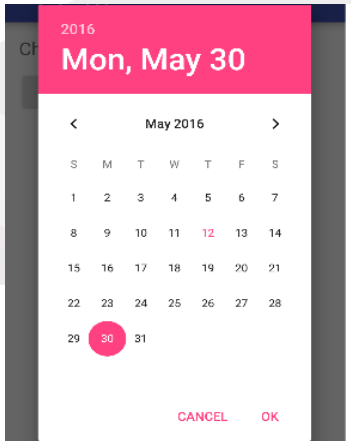
- Dialogs appear on top, interrupting the flow of the activity, and require an action to be dismissed

- Different types:
  - Alert dialog, date picker, time picker

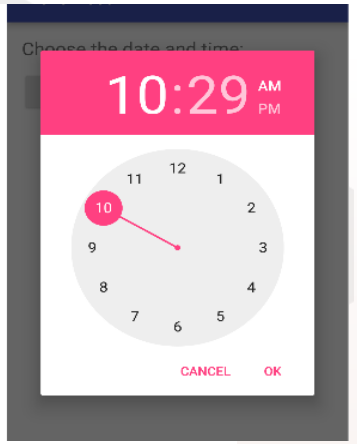
- AlertDialog can show:
  1. Title (optional)
  2. Content area
  3. Action buttons

```
public void onClickShowAlert(View view) {
    AlertDialog.Builder alertDialog = new
                        AlertDialog.Builder(MainActivity.this);
    alertDialog.setTitle("Connect to Provider");
    alertDialog.setMessage(R.string.alert_message);
    ...
```

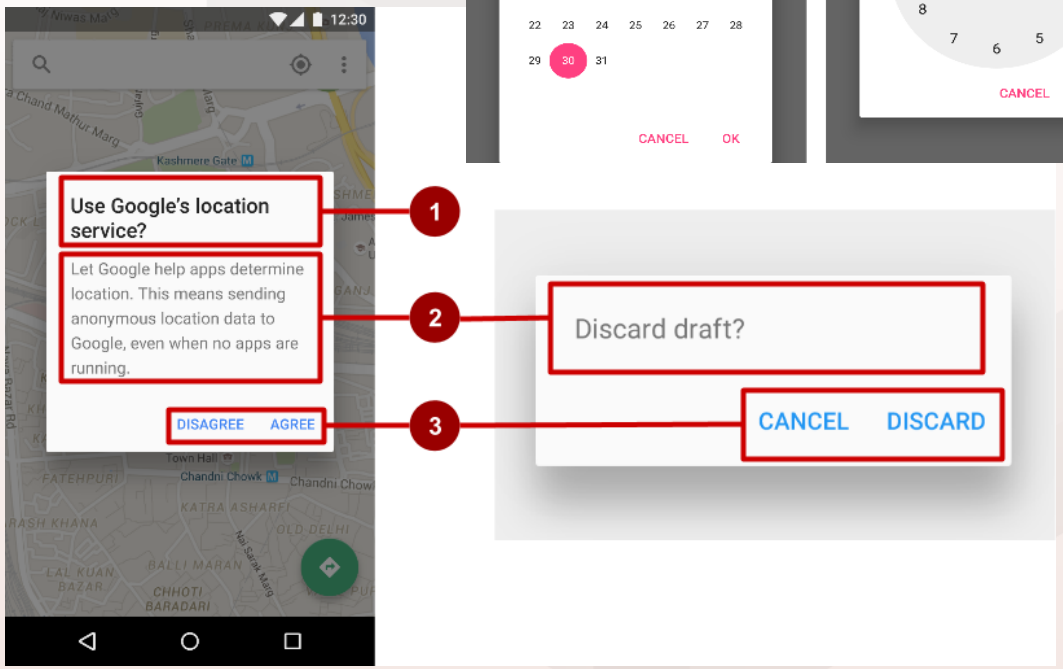
DatePicker



TimePicker



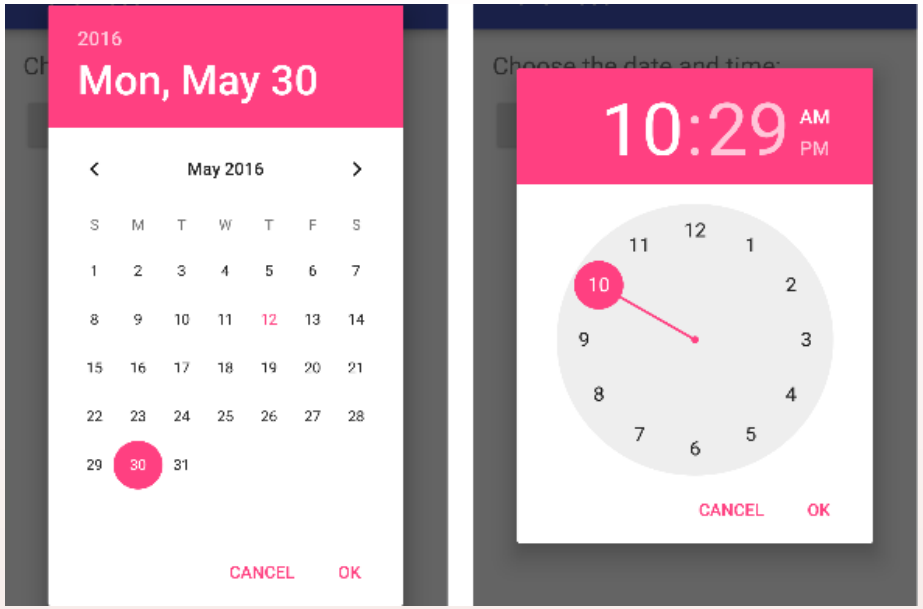
AlertDialog



Creating a DatePicker

- We will see a specific example during the lab session:

DatePicker

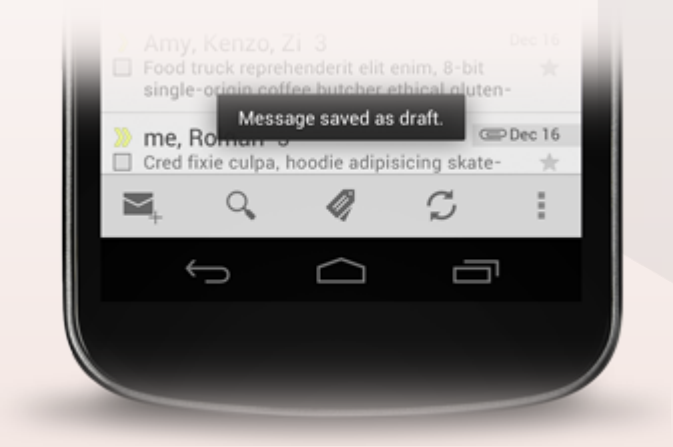



```java
final Button btn = fragmentView.findViewById(R.id.btnBirthday);
final DatePickerDialog.OnDateSetListener mDateSetListener = new
DatePickerDialog.OnDateSetListener() {
    @Override
    public void onDateSet(DatePicker datePicker, int year, int month, int day) {
        // Format the result as a Date object
        Calendar calendar = Calendar.getInstance();
        calendar.set(year, month, day);
        Date date = calendar.getTime();
        // Format the date as a string according to the user's locale settings
        java.text.DateFormat dateFormat = DateFormat.getDateFormat(getActivity());
        // Display the time
        btn.setText(dateFormat.format(date));
    }
};
btn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        // Use the java.util.Calendar class and NOT android.icu.util.calendar!!!
        Calendar cal = Calendar.getInstance();
        // The calendar is created with today's date, so we create the date
        // picker with this date, along with a reference to the defined listener
        // and the theme we want for the picker:
        DatePickerDialog dialog = new DatePickerDialog(getActivity(),
                R.style.Theme_AppCompat_Light_Dialog,
                mDateSetListener,
                cal.get(Calendar.YEAR),
                cal.get(Calendar.MONTH),
                cal.get(Calendar.DAY_OF_MONTH));
        dialog.show();
    }
});
```

Toast: Making a Toast

- Tiny messages over the Activity
- Used to signal to the user some confirmation, error, etc.
- Can control the duration of the Toast
- As simple as:

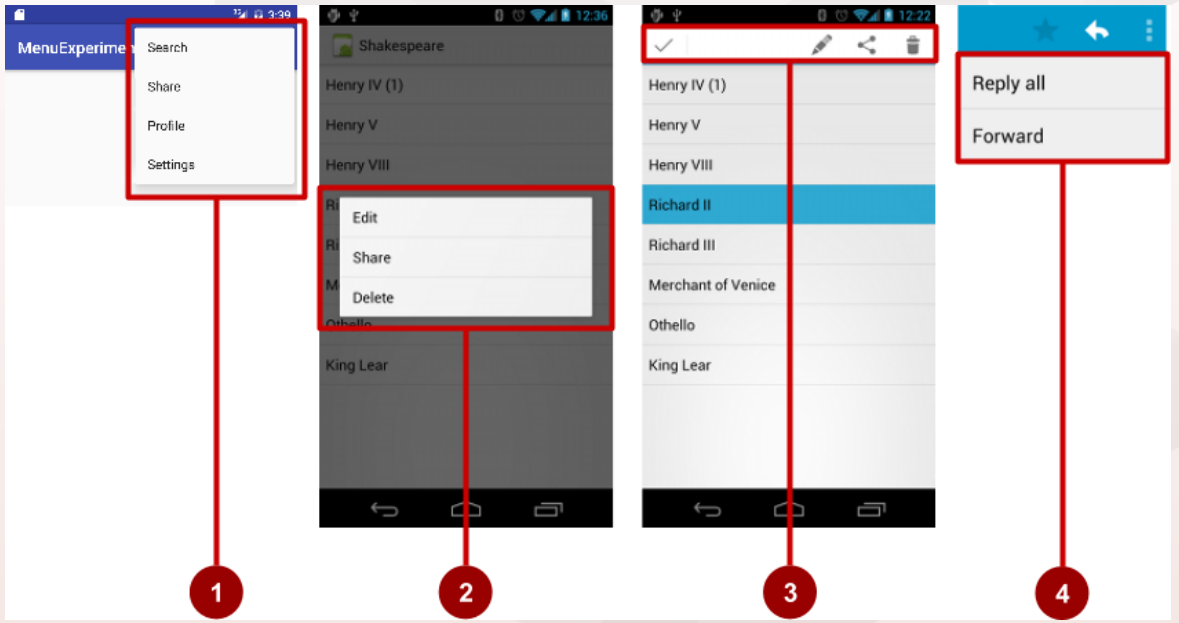
```
Toast msg = Toast.makeText(this, “Toast!”, Toast.LENGTH_SHORT).show();
```



Menus

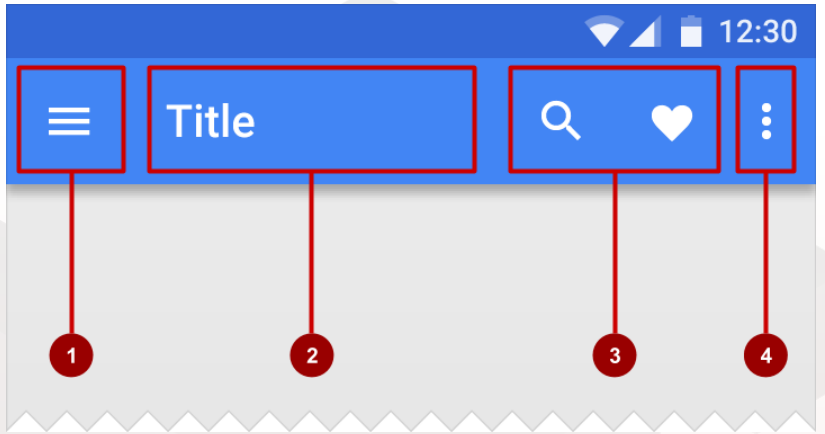
- They appear whenever the user presses the menu button
- Useful for giving different options without leaving the current Activity
  - Your projects should have menus!! → at least one, please!

- Types of menus
  - Application bar with options menus
  - Contextual menu
  - Contextual action bar
  - Popup menu

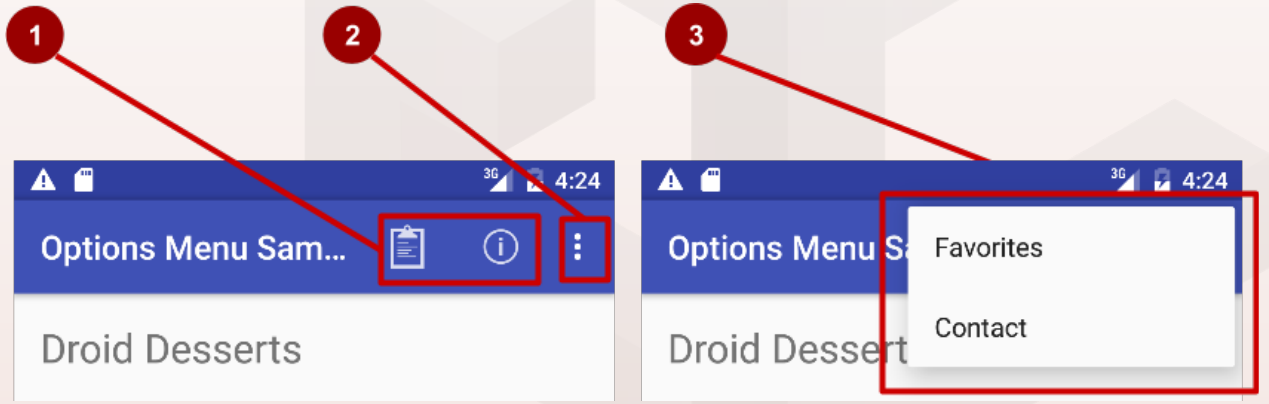


Application bar with options menu

- Bar at the top of each screen, usually the same for all screens
  1. Navigation icon to open navigation drawer
  2. Title of the current activity
  3. Icons for **options menu** items
  4. Action overflow button for rest of options

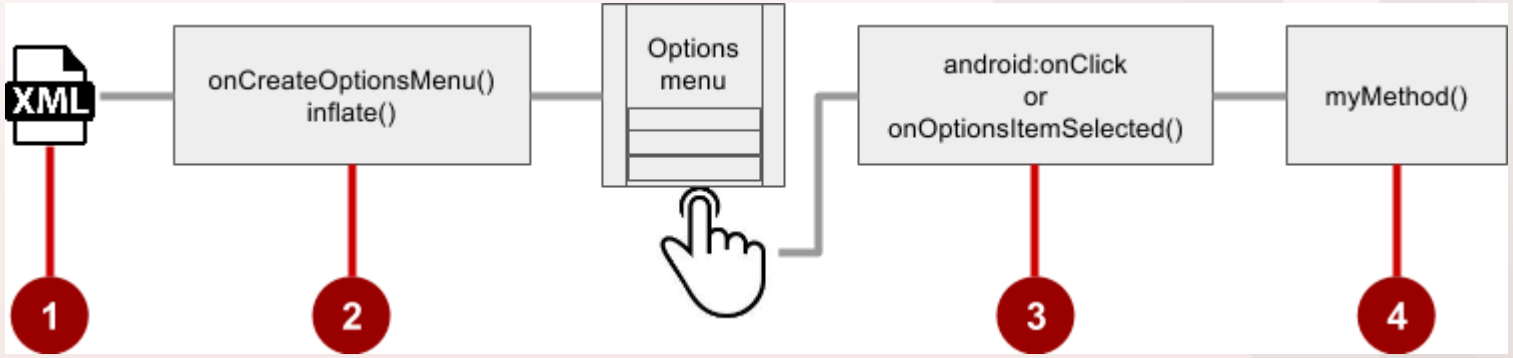


- What is the options menu?
  - Actions for important items (1)
  - By tapping the overflow part (2) you get more options



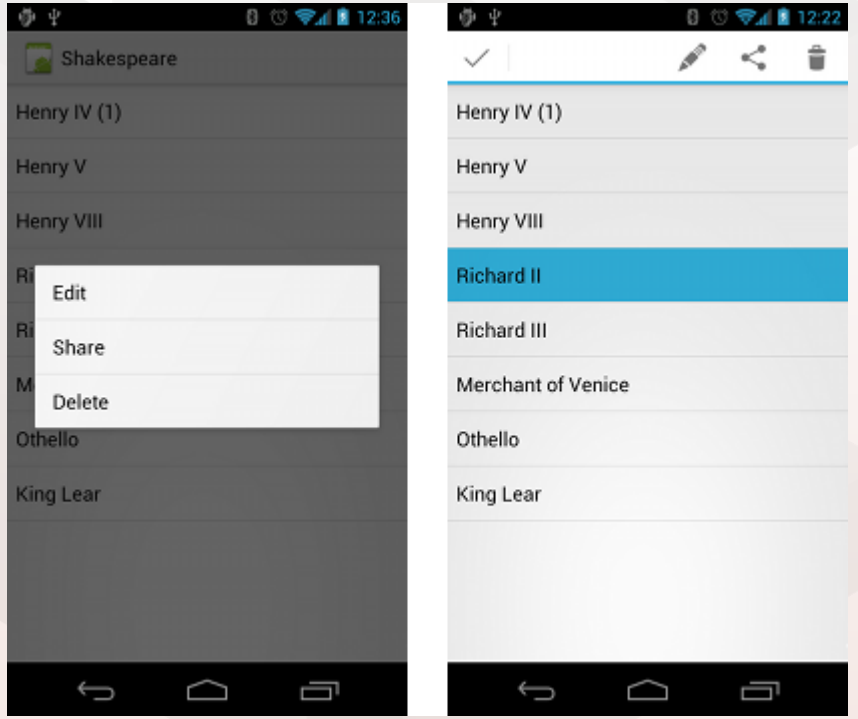
Steps to implement options menu

- As always, we develop the menu in XML and Java:
  1. XML menu resource (menu_main.xml)
     - Placing new file inside “res/menu”
  2. onCreateOptionsMenu() to inflate the menu inside the activity
  3. onClick attribute or onOptionsItemSelected()
  4. Method to handle item click



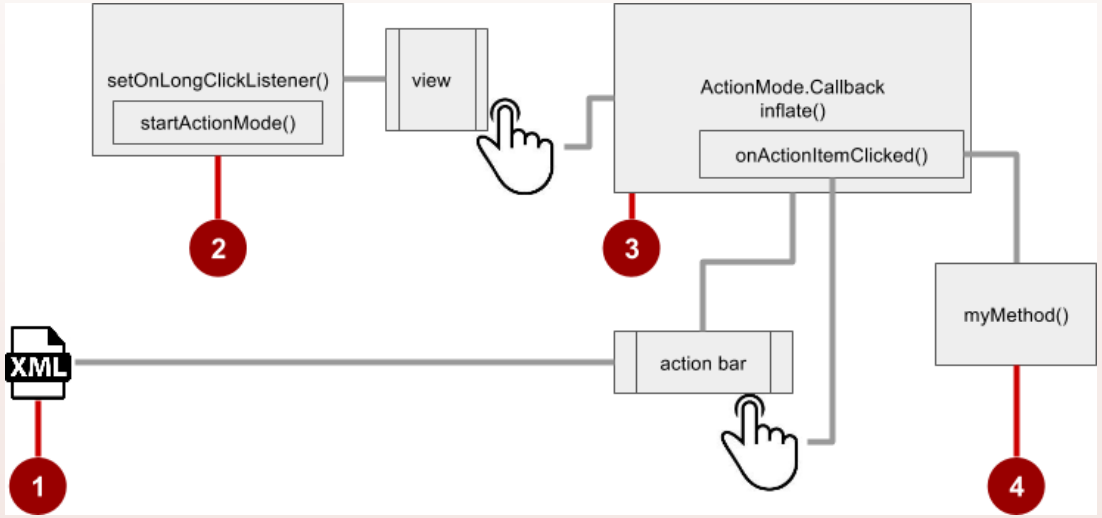
Contextual menus

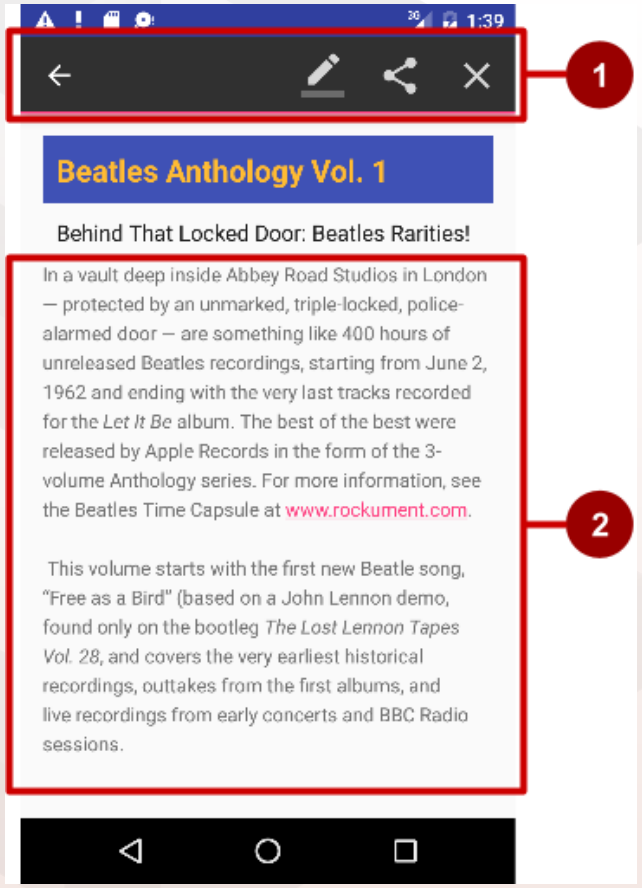
- Allow users to perform an action on a selected view or content
- Can be deployed on any View object
- Two types:
  - Floating context menus
    - Floating list of menu items
    - Users can modify the View element or use it
    - Users perform a contextual action
  - Contextual action mode
    - Temporary action bar in place of or underneath the app bar
    - Users can perform action on multiple elements



Contextual action bar

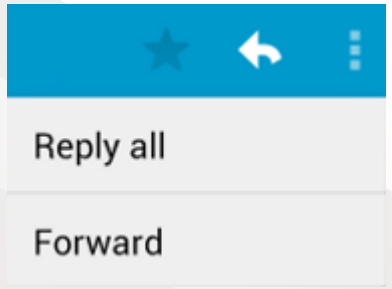
- Long-tap on the view shows contextual action bar
  1. Contextual action bar with actions:
     - Edit, Share, Delete…
     - Done (left arrow icon) on the left side
  2. View on which long press triggers the contextual action bar
     - Action bar is available until user tap is done

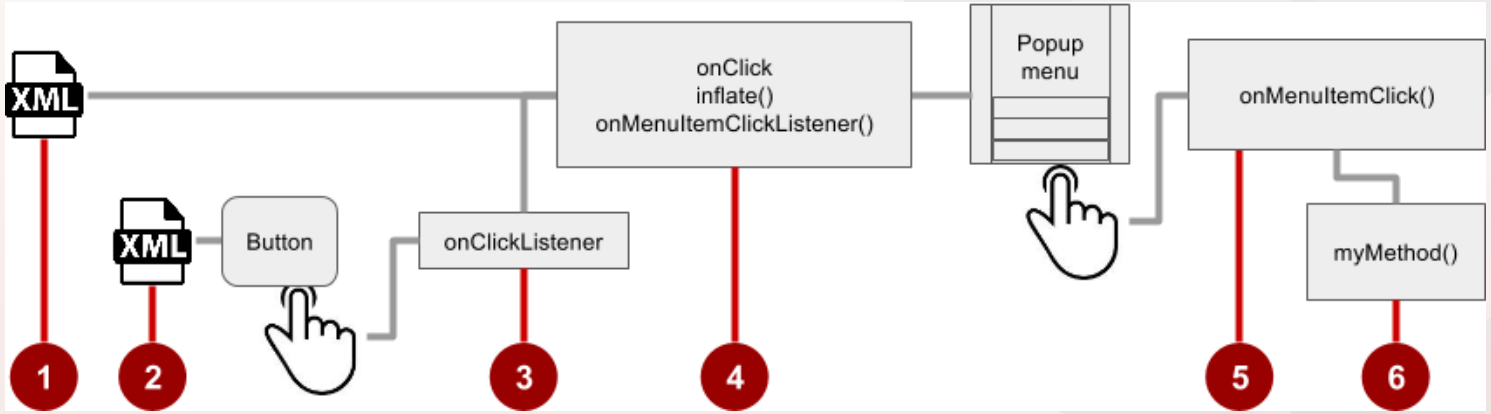




Popup menu

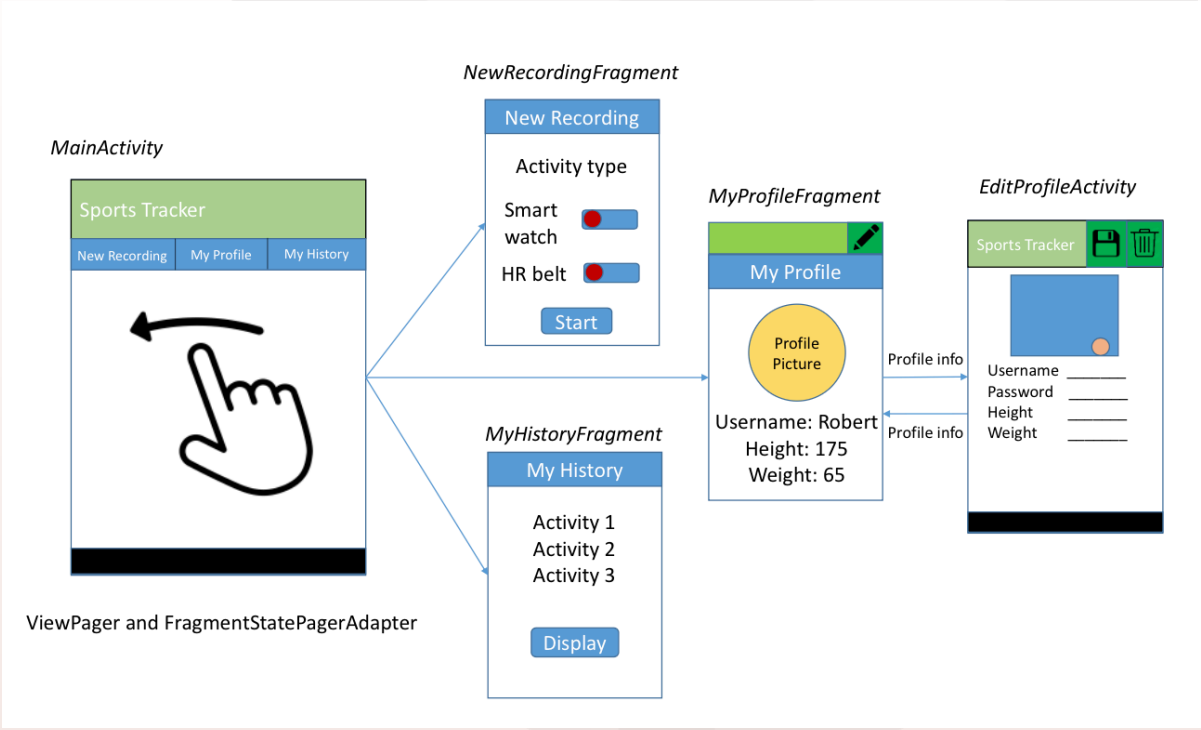
- Typically a list of items anchored to a view (visible icon)
- Actions should not direcly affect the content view:
  - The options menu overflow that opens Settings
  - For example, in an email app, Reply All and Forward are related to the email message, but don't affect or act on the message





Today's lab

- Adding **fragments** to our sports tracker app
  - The ViewPager layout
  - Moving contents from MainActivity to ViewPager
- UI: Toasts, **menus**, dialogs…
  - Adding an action bar menu



Questions?