Module 3
UI components

COS30017 COS80019

Software Development for Mobile Devices

Overview of module

- This module will introduce:
  - UI elements
  - ActionBar/ToolBar and menus
  - Toasts and snackbars
  - Lists and RecyclerView
  - Splitting up activities into fragments
  - Form design and data passing

UI elements

- UI elements
- Toasts and snackbars
- ActionBar/ToolBar and menus
- Form design and data passing
- Lists and RecyclerView
- Splitting up activities into fragments

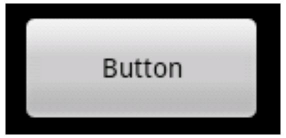
Simple UI elements

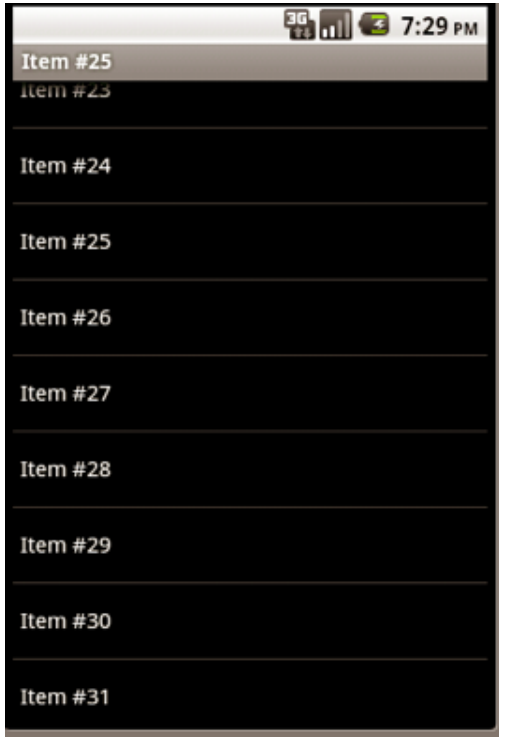
- Text
  - Labels
  - Inputs
- Images
- Buttons
  - Regular buttons
  - Floating action buttons (FABs)
  - Require onClick handler
- Checkboxes:
  - Require onClick handler
- Radio buttons: select one option from all displayed

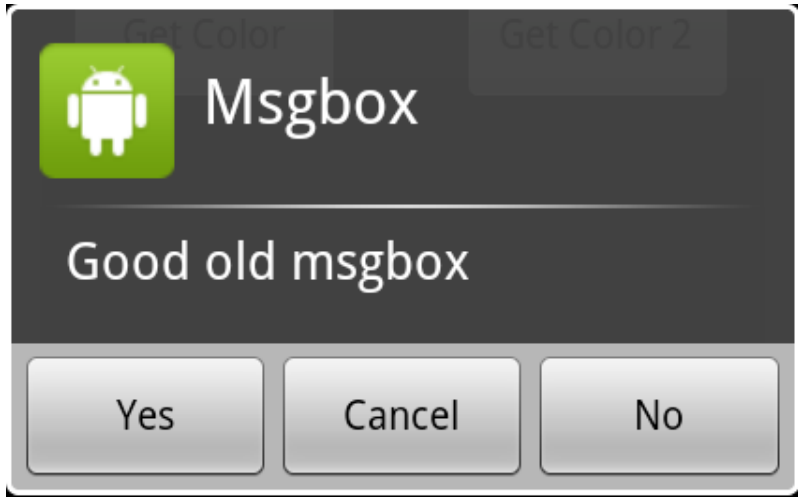
Simple UI elements

- Toggle buttons: one of two states (on/off etc.)
  - Includes toggles and switches
  - Use CompundButton.OnCheckedChangeListener
- Spinners: dropdowns, select one option from a list
  - Needs string-array in a file in /res/values with items
  - Activity needs to implement AdapterView.OnItemSelectedListener
- Pickers: for times and dates
  - Display as dialog box

UI controls







Good old msgbox

These are a small subset of available controls

Toasts and snackbars

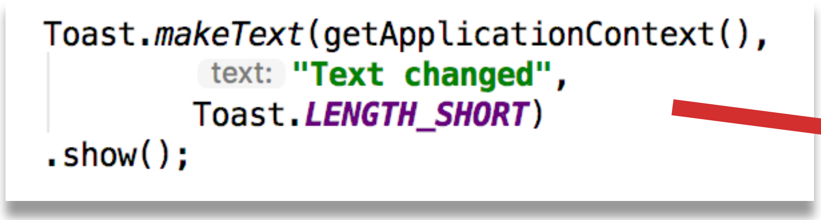
- UI elements
- Toasts and snackbars
- ActionBar/ToolBar and menus
- Form design and data passing
- Lists and RecyclerView
- Splitting up activities into fragments

Informing users

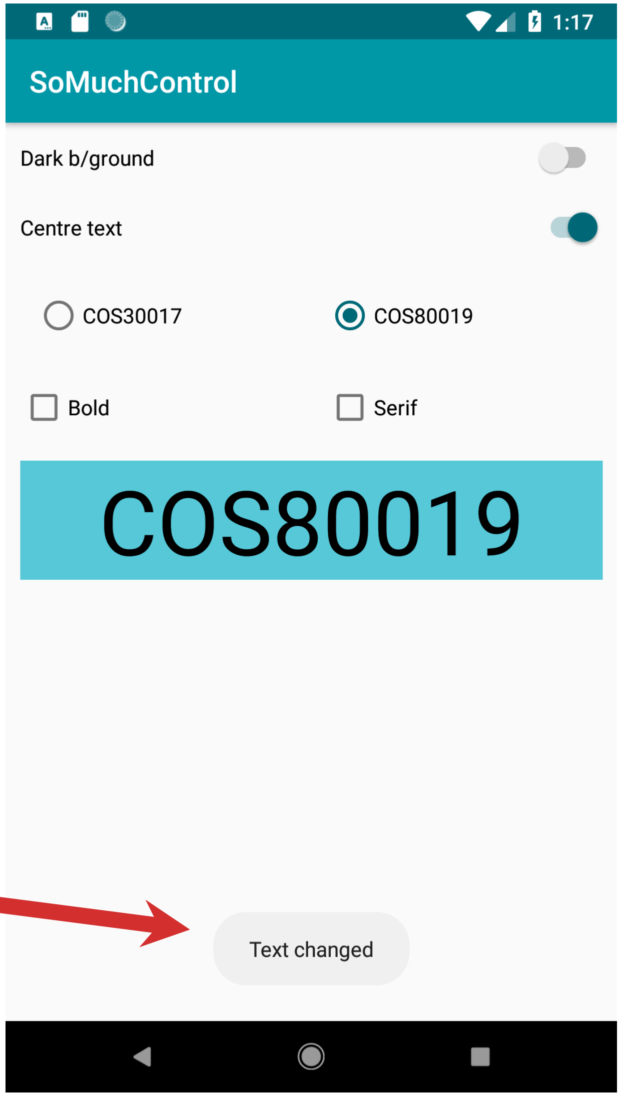
- There are situations where users need to be notified that something has happened, that might or might not need their attention.

Toasts

- Allow the user to be informed that something has happened.
- Does not require the user to still be on the same activity.
- Are not interactive.
- Requires the duration of display to be set.



```
Toast.makeText(getApplicationContext(),
        text: "Text changed",
        Toast.LENGTH_SHORT)
.show();
```



1:17

SoMuchControl

Dark b/ground

Centre text

◯ COS30017 [✓ COS80019]

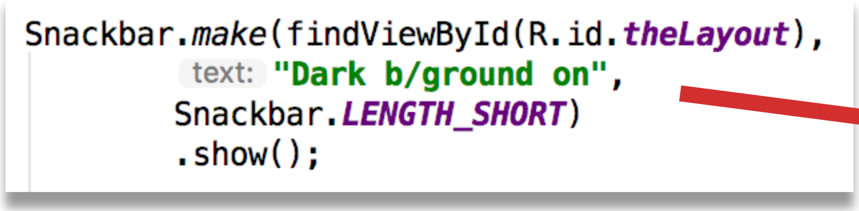
☐ Bold ☐ Serif

COS80019

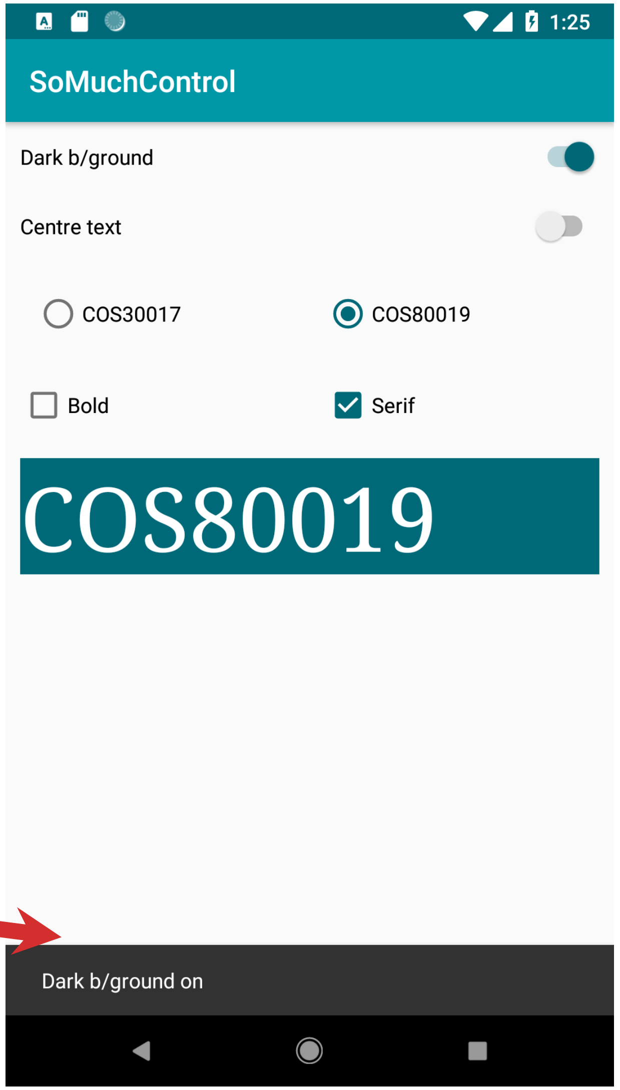
Text changed

Snackbars

- New in Android 22.2, and are now the preferred way of notifying users rather than Toasts.
- Provide the user with feedback about an event.
- Should appear in the same activity window.
- Best used with a CoordinatorLayout; this can be included around the rest of your layout.



```
Snackbar.make(findViewById(R.id.theLayout),
        text: "Dark b/ground on",
        Snackbar.LENGTH_SHORT)
        .show();
```



Dark b/ground [✓ on]

Centre text [off]

COS30017 [✓ COS80019]

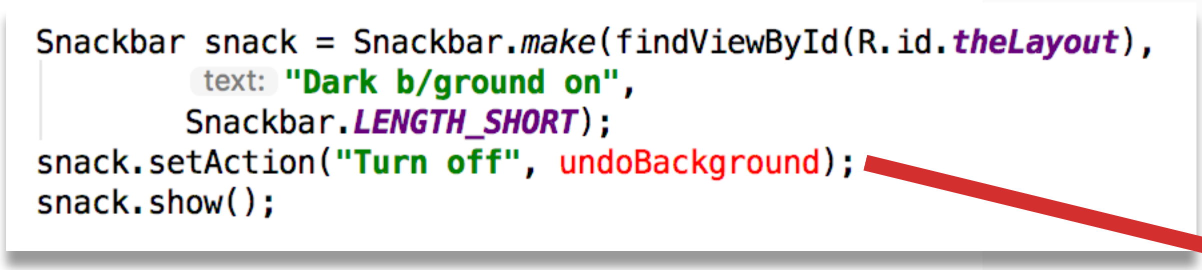
Bold [✓ Serif]

COS80019

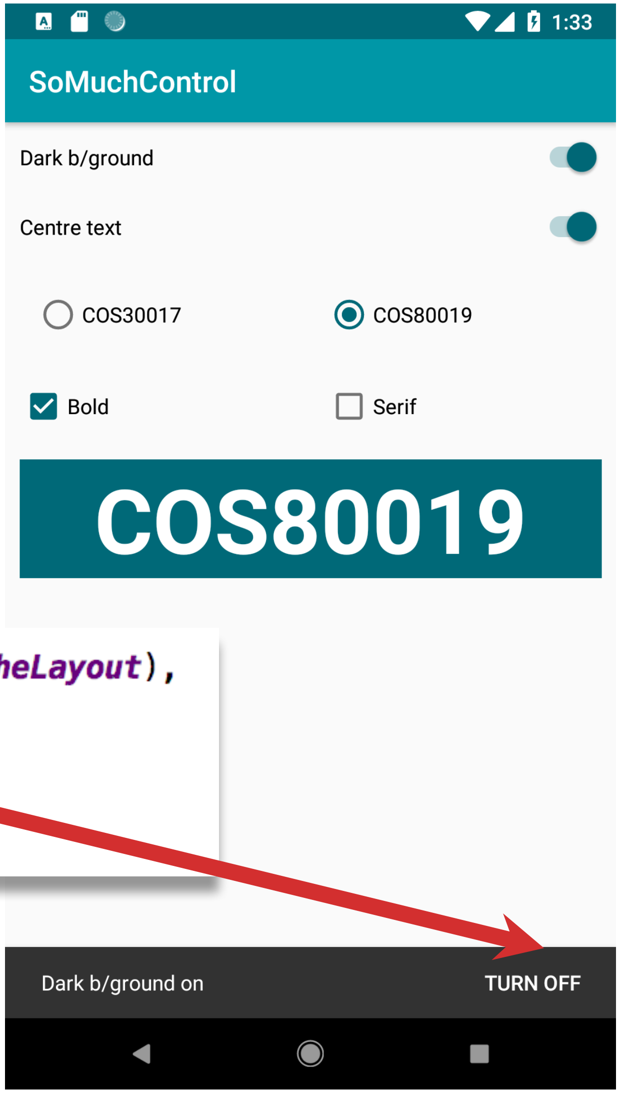
Dark b/ground on

Snackbars

- Snackbars can have actions attached to them, so the user can interact with them.



```
Snackbar snack = Snackbar.make(findViewById(R.id.theLayout),
          text: "Dark b/ground on",
          Snackbar.LENGTH_SHORT);
snack.setAction("Turn off", undoBackground);
snack.show();
```



1:33

SoMuchControl

Dark b/ground

Centre text

COS30017 [✓ COS80019]

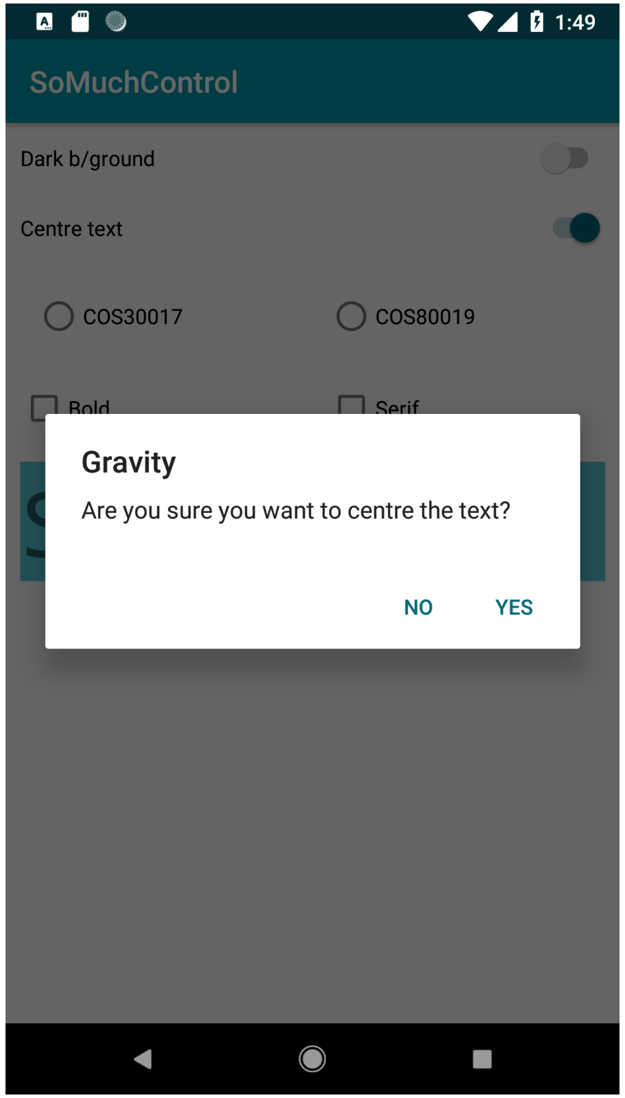
[✓ Bold] Serif

COS80019

Dark b/ground on TURN OFF

Dialogs

- For comparison, a dialog is used when a response is needed and action might need to be taken.
- Dialogs need to be acknowledged. The user cannot do anything else until the dialog is dismissed.
- Dialogs can contain lists (either single choice of multiple choice).



Gravity

Are you sure you want to centre the text?

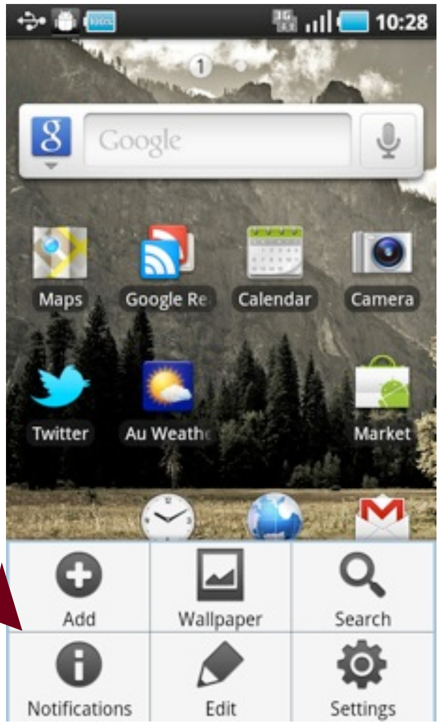
ActionBar/ToolBar and menus

- UI elements
- Toasts and snackbars
- ActionBar/ToolBar and menus
- Form design and data passing
- Lists and RecyclerView
- Splitting up activities into fragments

Menus - Historically

- Android offers a 'Menu' button, iOS does not.
- Menus' can be customised per activity.

Menu



Menus have a label and an icon

Icon is strongly recommended by the Android UI guidelines

Icons at different resolutions should be provided (ideally)

Menus Deprecated

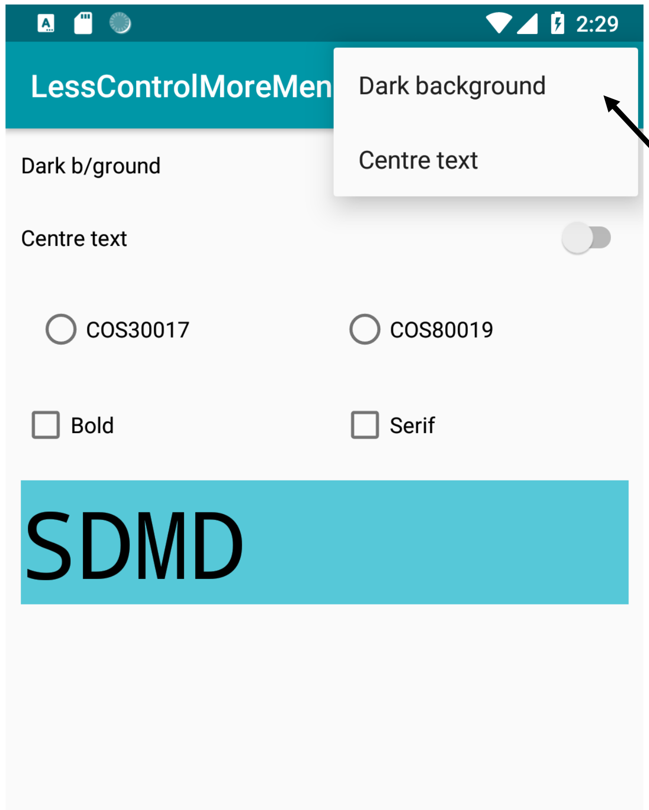
- From developer.android:
  - *Beginning with Android 3.0 (API level 11), Android powered devices were no longer required to provide a dedicated Menu button…*
  - *…instead provide an ActionBar to present common user actions.*
  - *Now known as Toolbar.*

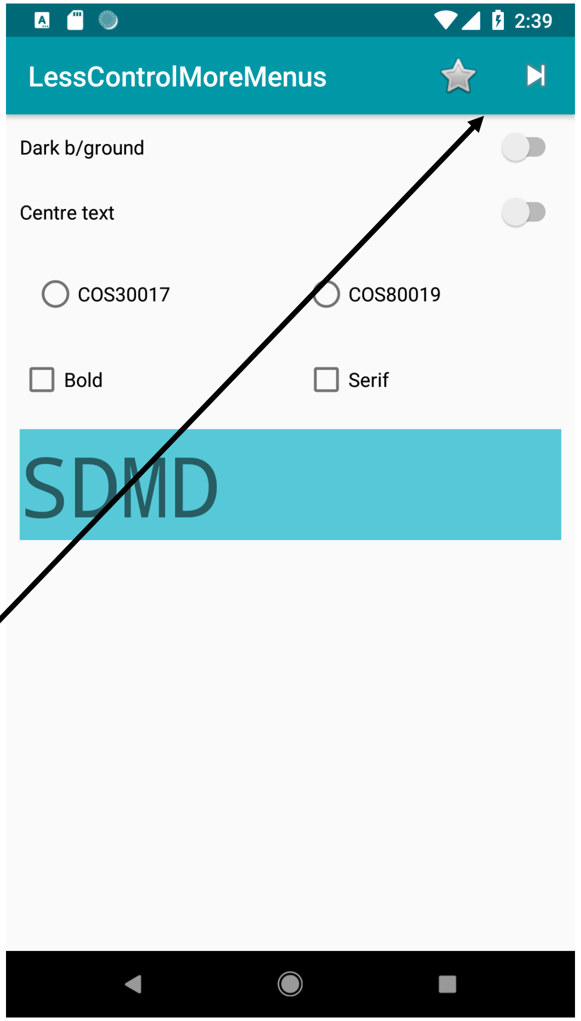
Basic Android Menu Taxonomy

- **Options menu** and app bar:
  - primary collection of menu items for an activity
  - items with “global impact” on your app
- **Context menu**:
  - floating menu after “long click” on item
- **Popup menu**:
  - appears when clicked, anchored to UI View

Options Menu

- Appears in the app bar.
- We can programmatically change or add to this for different activities.

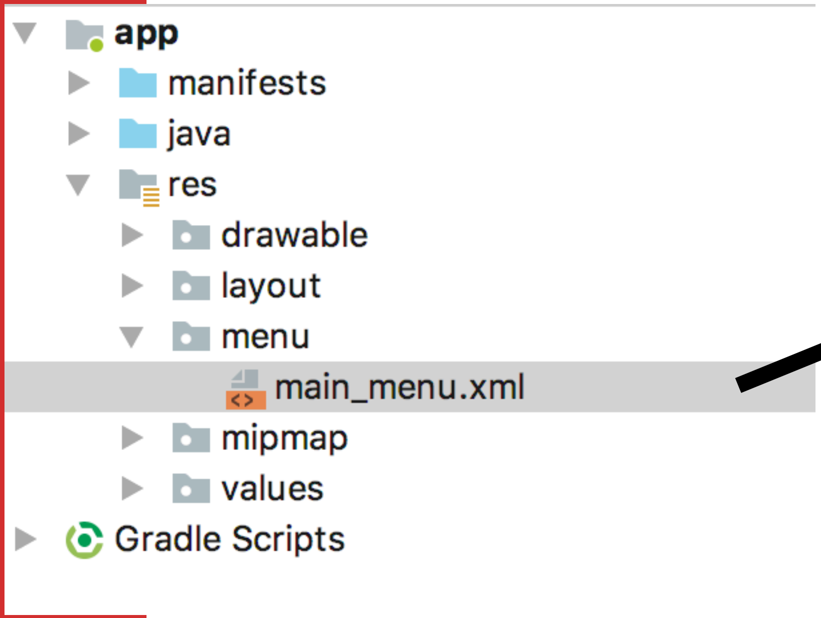


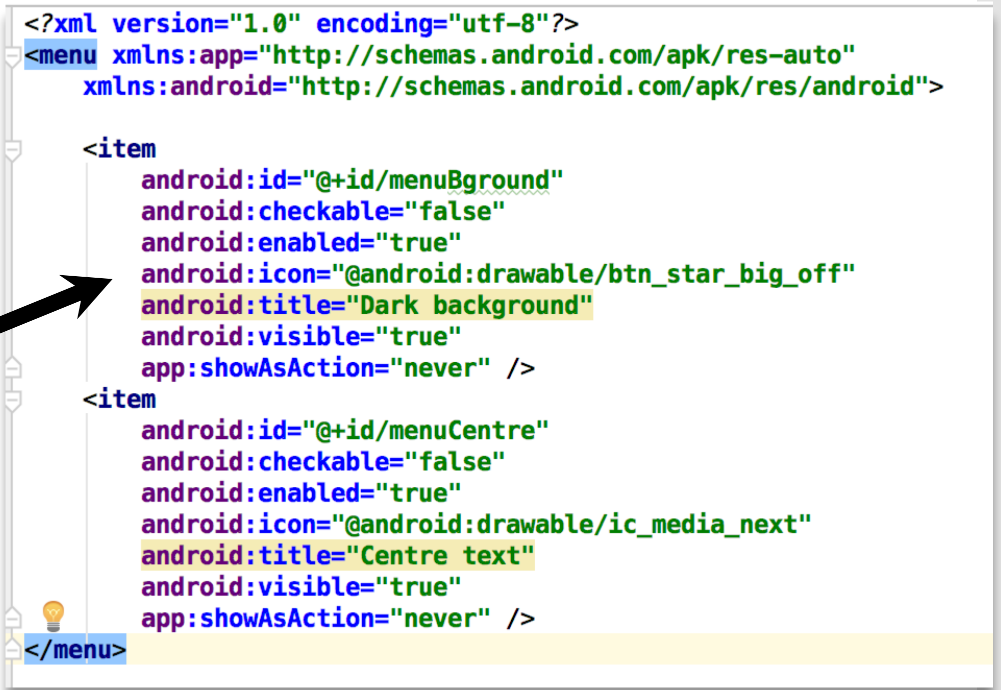


provides options relevant to app and/or activity

Menus are resources

- Convention is to place menu details in ‘menu’ folder.

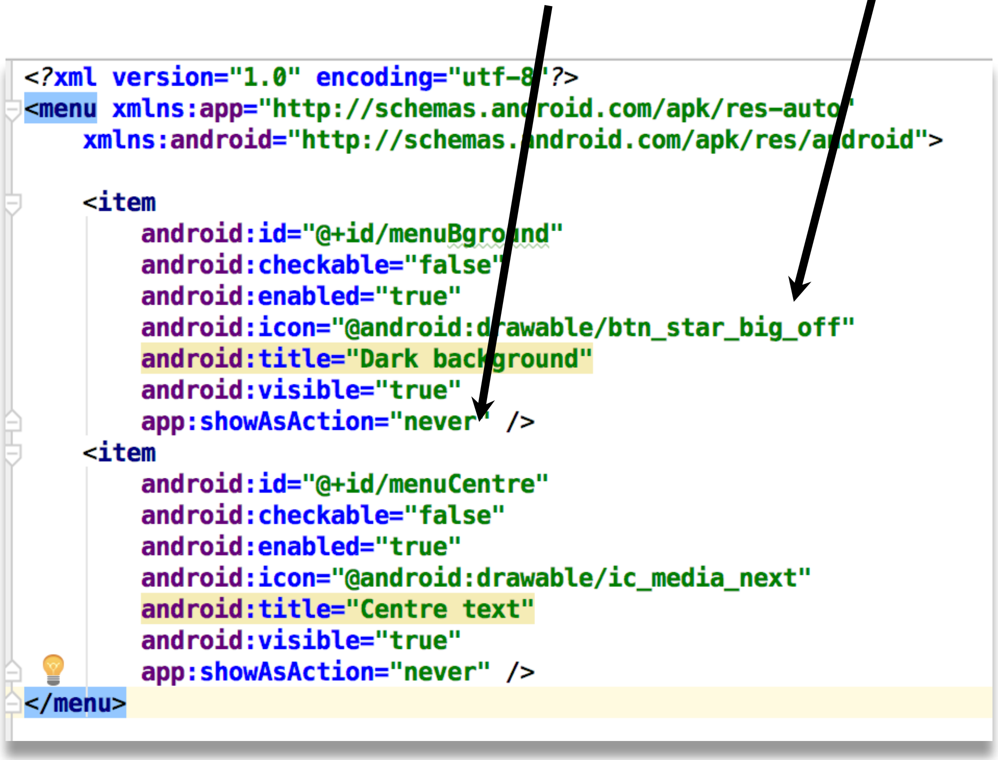




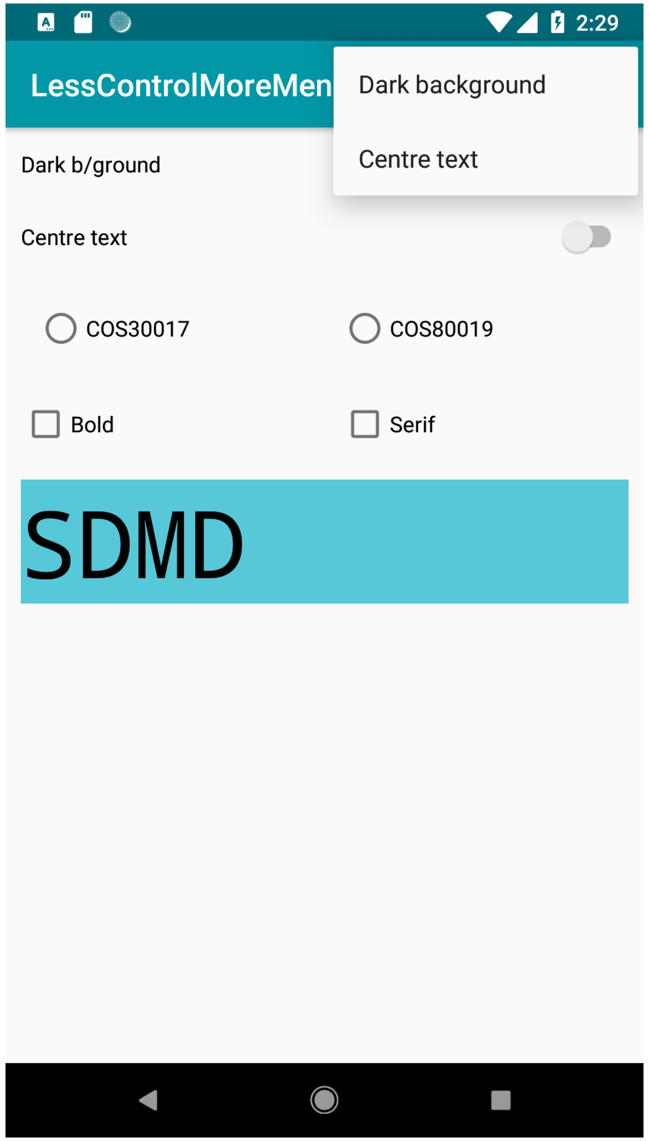
```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/menuBground"
        android:checkable="false"
        android:enabled="true"
        android:icon="@android:drawable/btn_star_big_off"
        android:title="Dark background"
        android:visible="true"
        app:showAsAction="never" />
    <item
        android:id="@+id/menuCentre"
        android:checkable="false"
        android:enabled="true"
        android:icon="@android:drawable/ic_media_next"
        android:title="Centre text"
        android:visible="true"
        app:showAsAction="never" />
</menu>
```

Creating a Menu

- Menus are defined as a resource.
- Icons provided by the SDK can also be used.
- Another option is to have items always in the toolbar or the menu, using the android:showAsAction attribute.

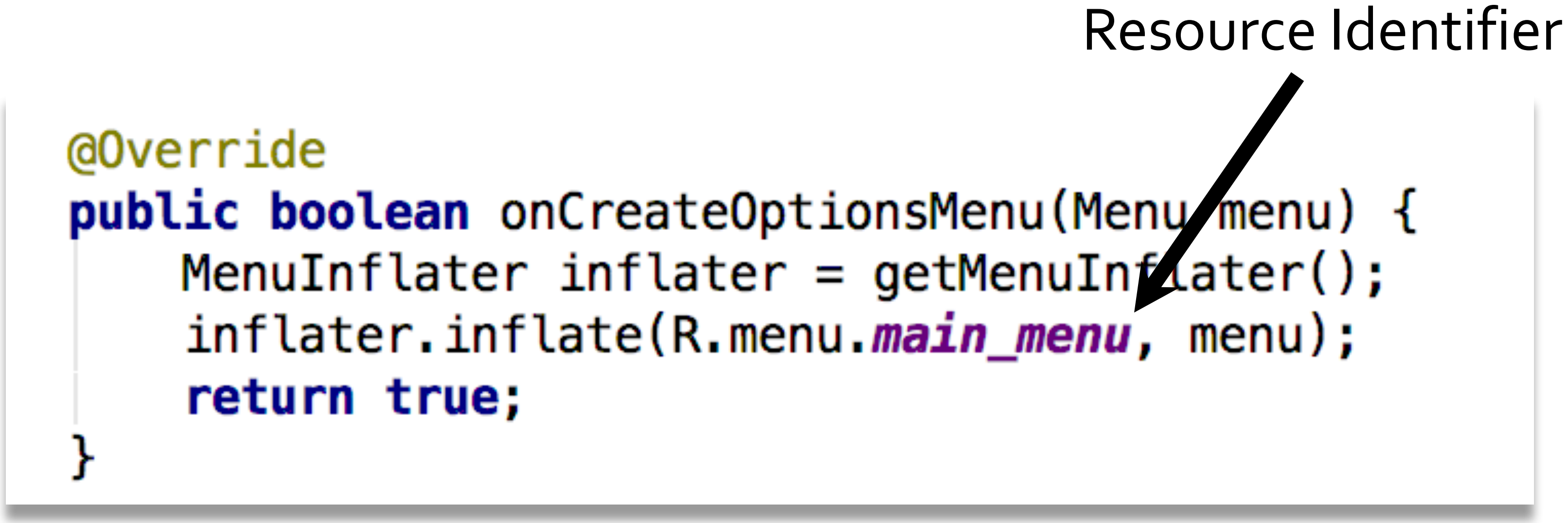


```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/menuBground"
        android:checkable="false"
        android:enabled="true"
        android:icon="@android:drawable/btn_star_big_off"
        android:title="Dark background"
        android:visible="true"
        app:showAsAction="never" />
    <item
        android:id="@+id/menuCentre"
        android:checkable="false"
        android:enabled="true"
        android:icon="@android:drawable/ic_media_next"
        android:title="Centre text"
        android:visible="true"
        app:showAsAction="never" />
</menu>
```



Wiring up a Menu to Show

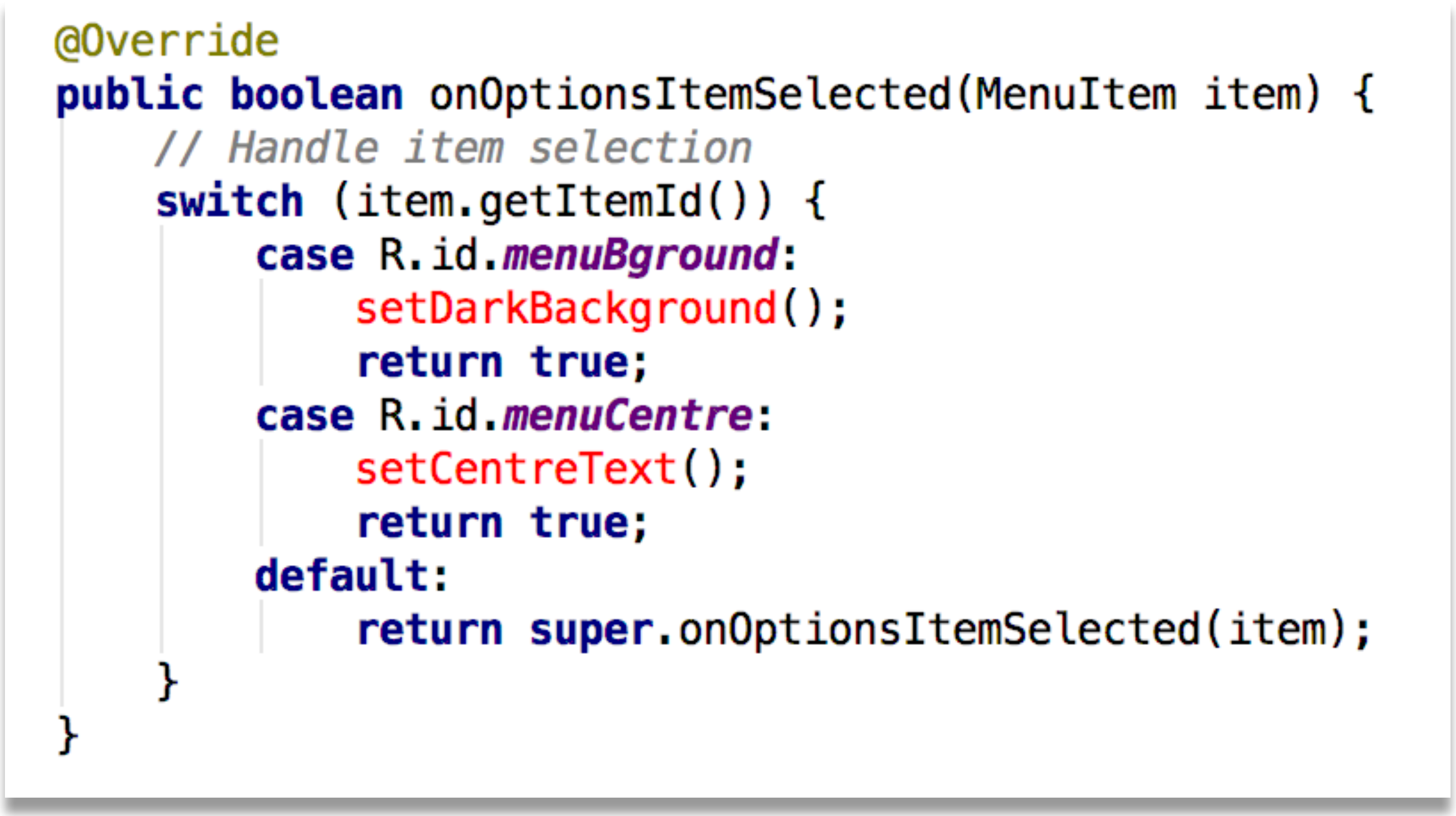
- A **MenuInflater** is an object that is able to create Menu from xml resources.
- The onCreateOptionMenu(Menu) is called when the menu button of the device is pressed, or either Activity.openOptionsMenu() is called.
- The XML Resource file is converted (*inflated*) into a Menu object that will be rendered (*shown*) on screen



```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.main_menu, menu);
    return true;
}
```

Handling Menu Click

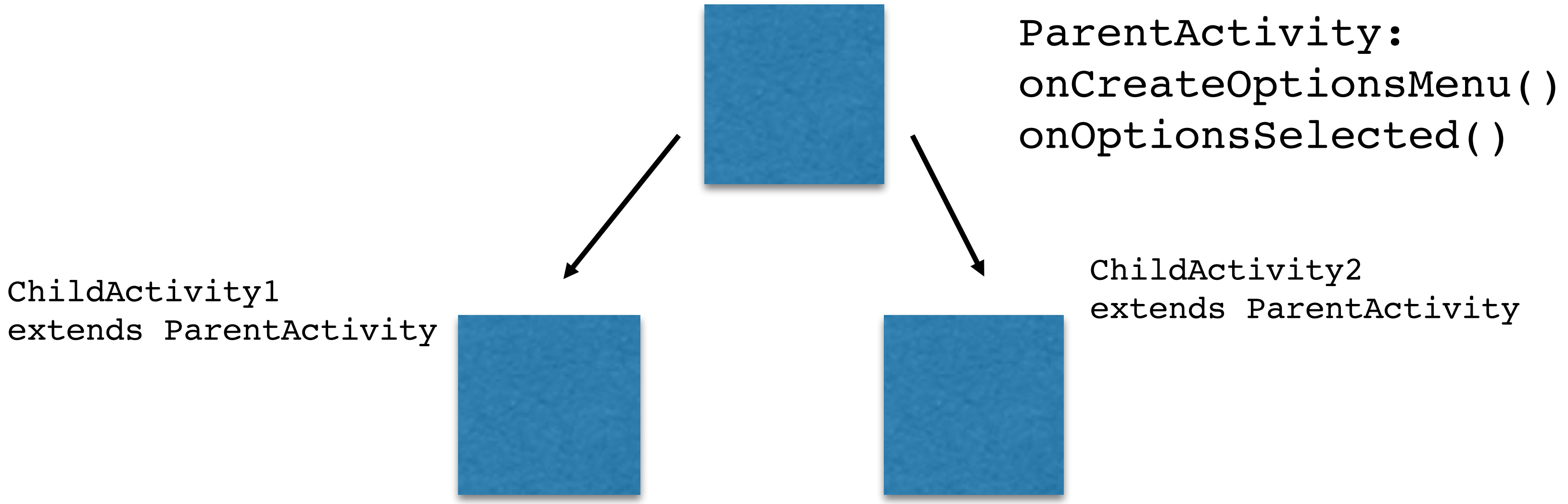
- We also require a callback method for handled selection of items. This can be handled as a switch or if statements.



```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case R.id.menuBground:
            setDarkBackground();
            return true;
        case R.id.menuCentre:
            setCentreText();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Menu Tips

- If developing for Android 3.0 or higher, you can add the android:onClick attribute for each menu item in XML
  - The callback method must be public and accept a single MenuItem
- If your application contains multiple Activities requiring same menu items:
  - implement the onCreateOptionsMenu() and onOptionsSelected() methods in a parent Activity class … and extend
  - Child Activities can also add items using menu.add() and then call super.onCreateOptionsMenu()

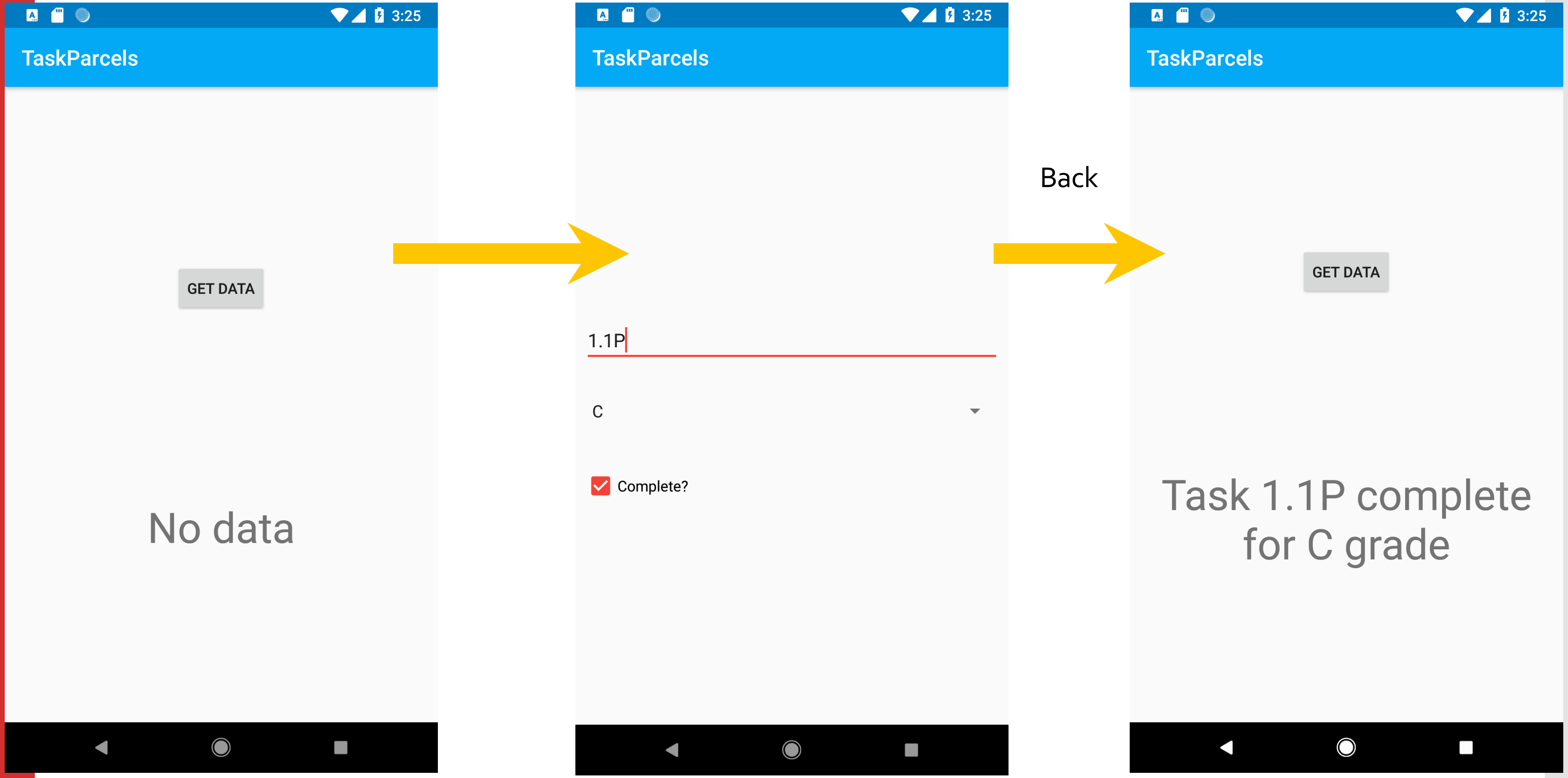


Form design and data passing

- UI elements
- Toasts and snackbars
- ActionBar/ToolBar and menus
- Form design and data passing
- Lists and RecyclerView
- Splitting up activities into fragments

Sending Data Back to Caller

- The simplest option is to use a Bundle (esp. for primitives).
- We can also send *Objects* back (but involves more work)



No data

1.1P

C

Complete?

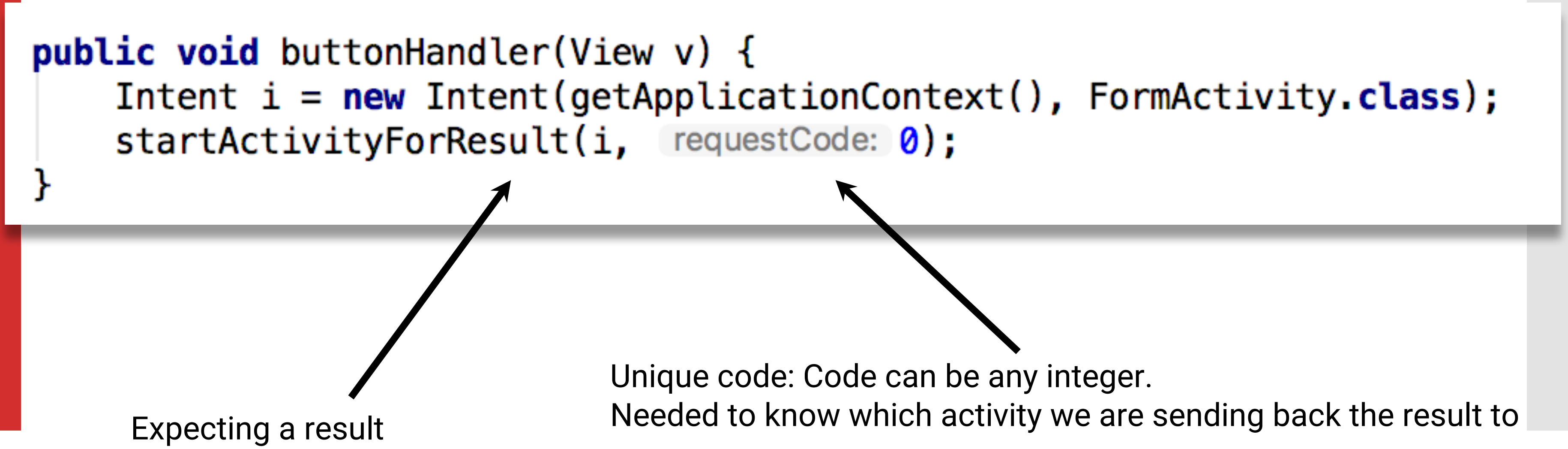
Task 1.1P complete for C grade

Parcelables

- Parcelables allow the sending of objects between activities (and also processes).
- Between processes, however, you would need to ensure that both processes have access to the same class (e.g., as part of a library in common).
- Parcelables are to Android what Serialization is to Java, to some extent: the process of transforming an object into a format for transmission.
- While Serialization involves only adding an implements clause to an object, Parcelable involves writing extra methods. However, the performance of Parcelable is better than Serializable, and a lot of the code can be automatically generated by the IDE.

Expecting Data (What is involved?)

- Activities communicate asynchronously
  - They send e-mail like messages to each other
- When we start an activity, we need to indicate that we are expecting a result (*explicitly*).
- This example of expecting a result is between two activities in the same app; the same process could apply to an intent for another app.



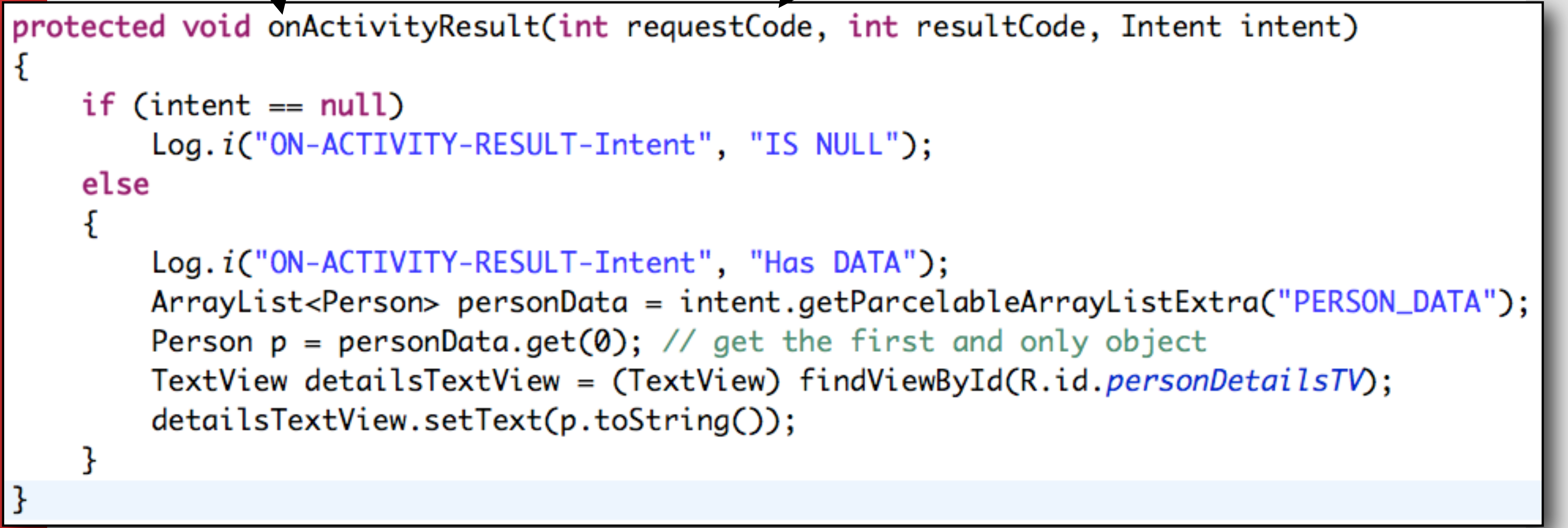
```
public void buttonHandler(View v) {
    Intent i = new Intent(getApplicationContext(), FormActivity.class);
    startActivityForResult(i,  requestCode: 0);
}
```

Expecting a result
Unique code: Code can be any integer.
Needed to know which activity we are sending back the result to

Receiving the Data

Call back method

Request Code that we sent when activity was started

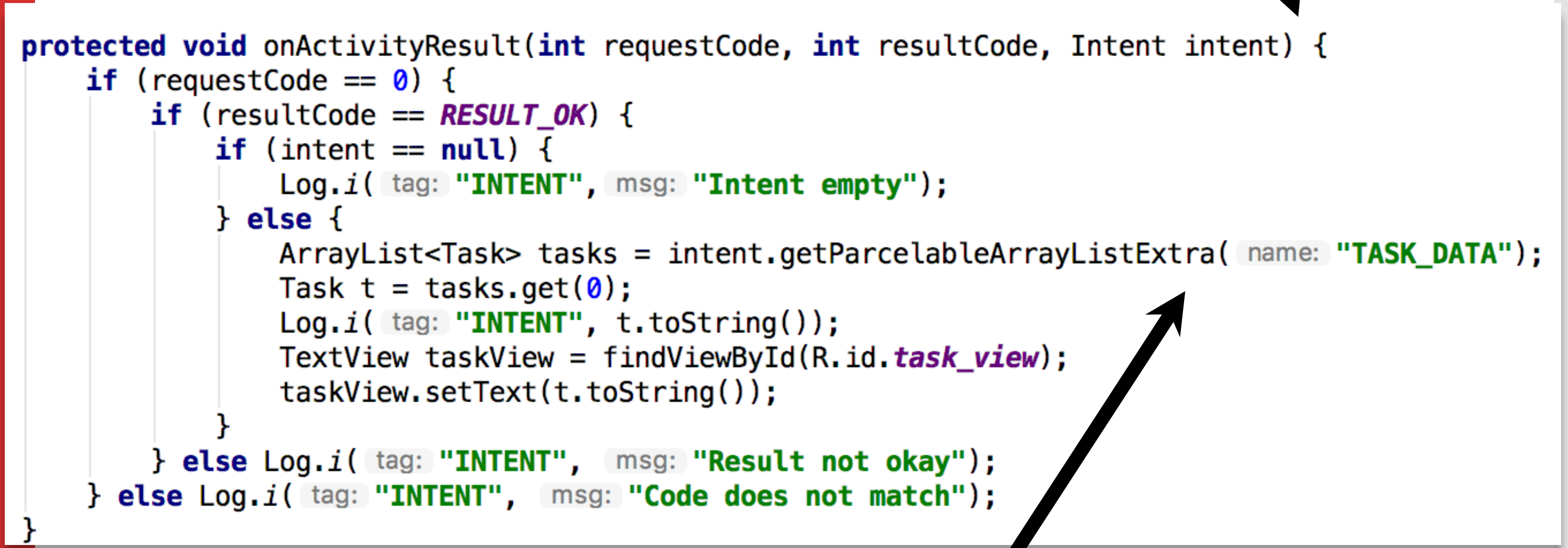


```
protected void onActivityResult(int requestCode, int resultCode, Intent intent)
{
    if (intent == null)
        Log.i("ON-ACTIVITY-RESULT-Intent", "IS NULL");
    else
    {
        Log.i("ON-ACTIVITY-RESULT-Intent", "Has DATA");
        ArrayList<Person> personData = intent.getParcelableArrayListExtra("PERSON_DATA");
        Person p = personData.get(0); // get the first and only object
        TextView detailsTextView = (TextView) findViewById(R.id.personDetailsTV);
        detailsTextView.setText(p.toString());
    }
}
```

We use the request code to determine which activity is calling back

Extracting the Data

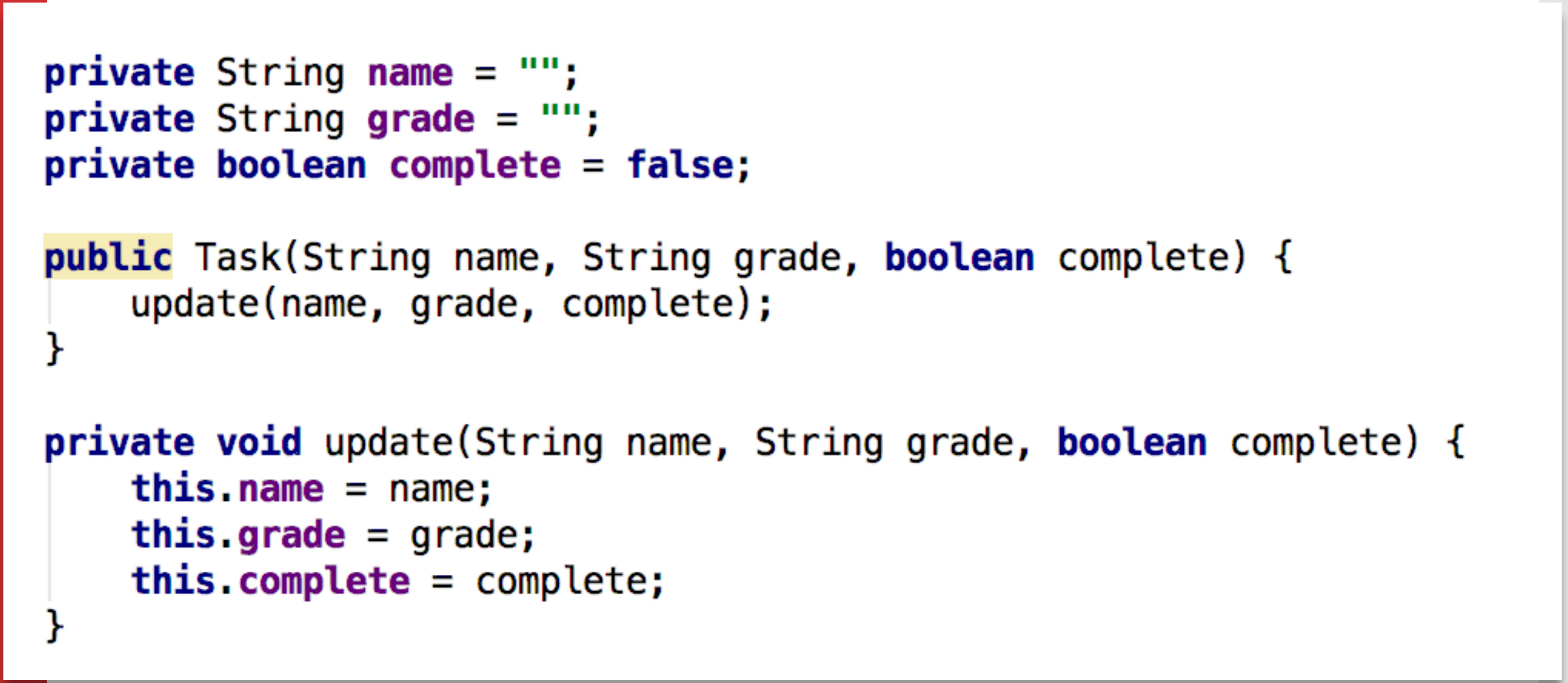
Data is packaged in the Intent



```
protected void onActivityResult(int requestCode, int resultCode, Intent intent) {
    if (requestCode == 0) {
        if (resultCode == RESULT_OK) {
            if (intent == null) {
                Log.i( tag: "INTENT", msg: "Intent empty");
            } else {
                ArrayList<Task> tasks = intent.getParcelableArrayListExtra( name: "TASK_DATA");
                Task t = tasks.get(0);
                Log.i( tag: "INTENT", t.toString());
                TextView taskView = findViewById(R.id.task_view);
                taskView.setText(t.toString());
            }
        } else Log.i( tag: "INTENT",  msg: "Result not okay");
    } else Log.i( tag: "INTENT",  msg: "Code does not match");
}
```

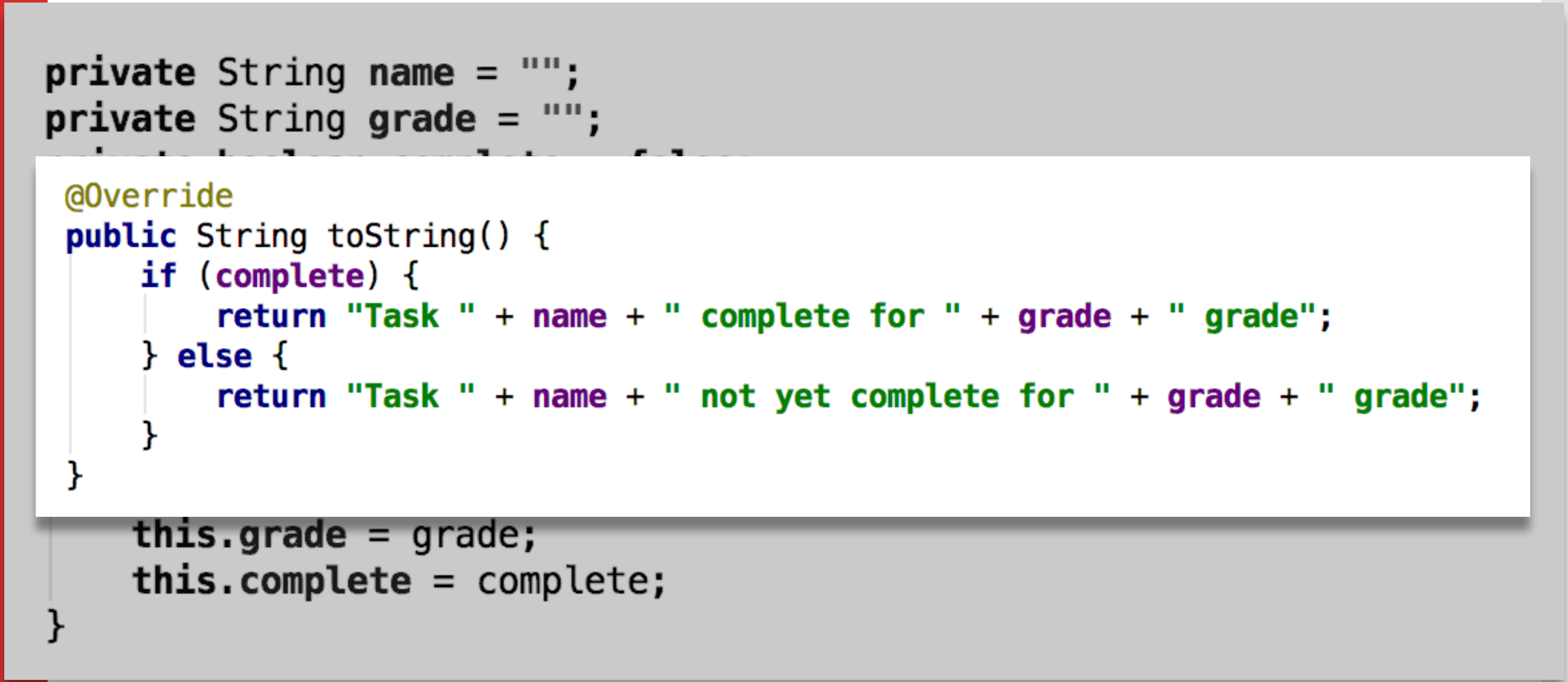
We obtain data from a Parcel
(like a Bundle, but stores objects)

The Object that is passed - Task



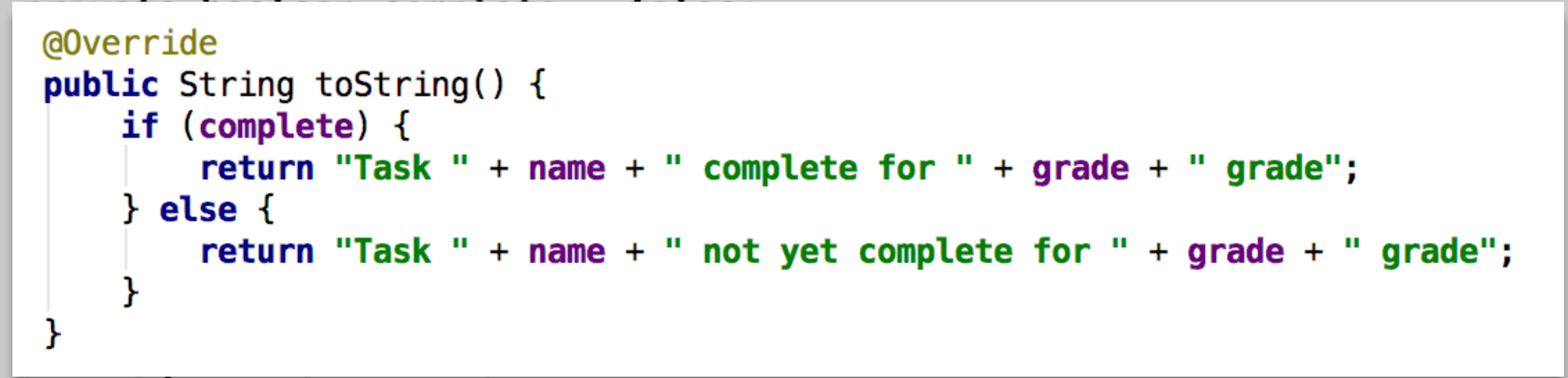
```
private String name = "";
private String grade = "";
private boolean complete = false;

public Task(String name, String grade, boolean complete) {
    update(name, grade, complete);
}

private void update(String name, String grade, boolean complete) {
    this.name = name;
    this.grade = grade;
    this.complete = complete;
}
```

Task Object



```java
private String name = "";
private String grade = "";
```

```java
    this.grade = grade;
    this.complete = complete;
}
```

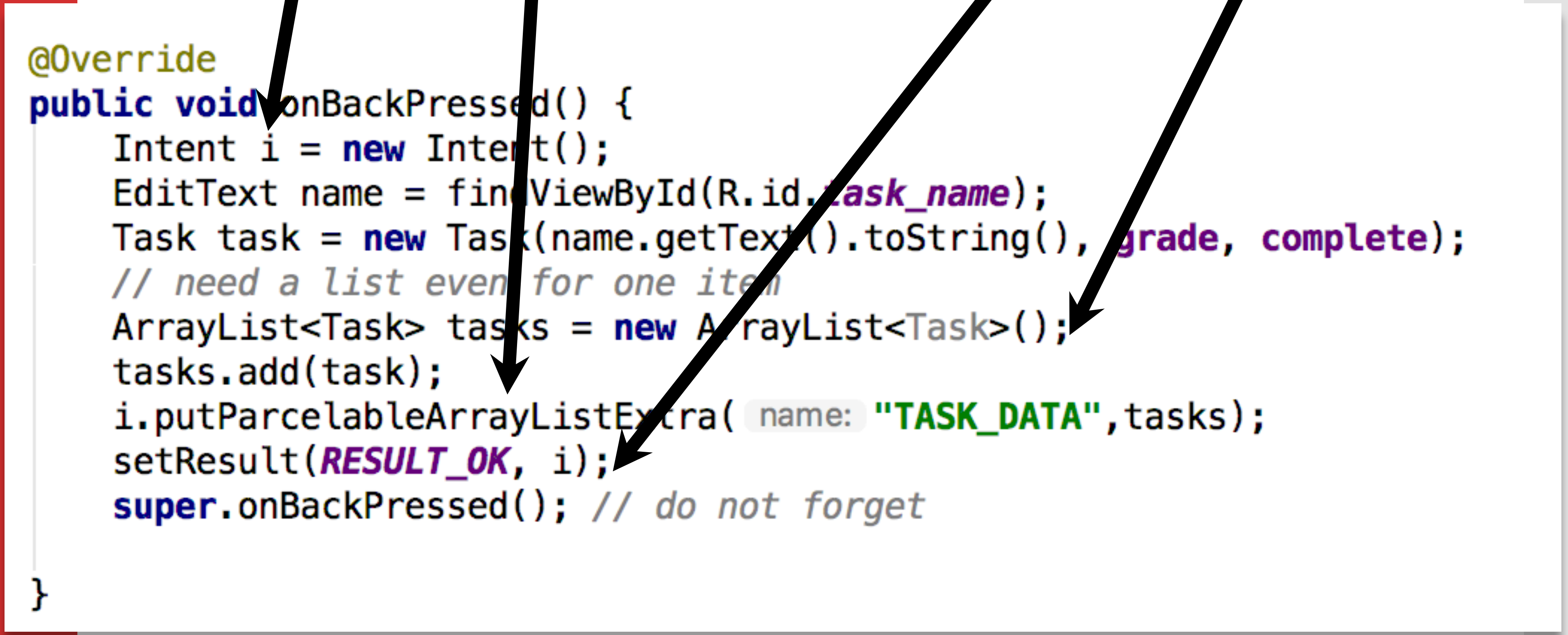


```java
@Override
public String toString() {
    if (complete) {
        return "Task " + name + " complete for " + grade + " grade";
    } else {
        return "Task " + name + " not yet complete for " + grade + " grade";
    }
}
```

Simple data holder also has toString() method

Sending the data back

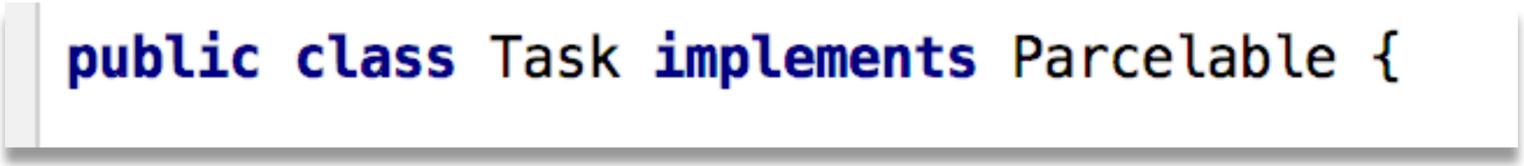
- Do this when “Back” button pressed.
- Create intent, put Task objects into ArrayList for transport, store ArrayList in intent, and send intent back to caller.



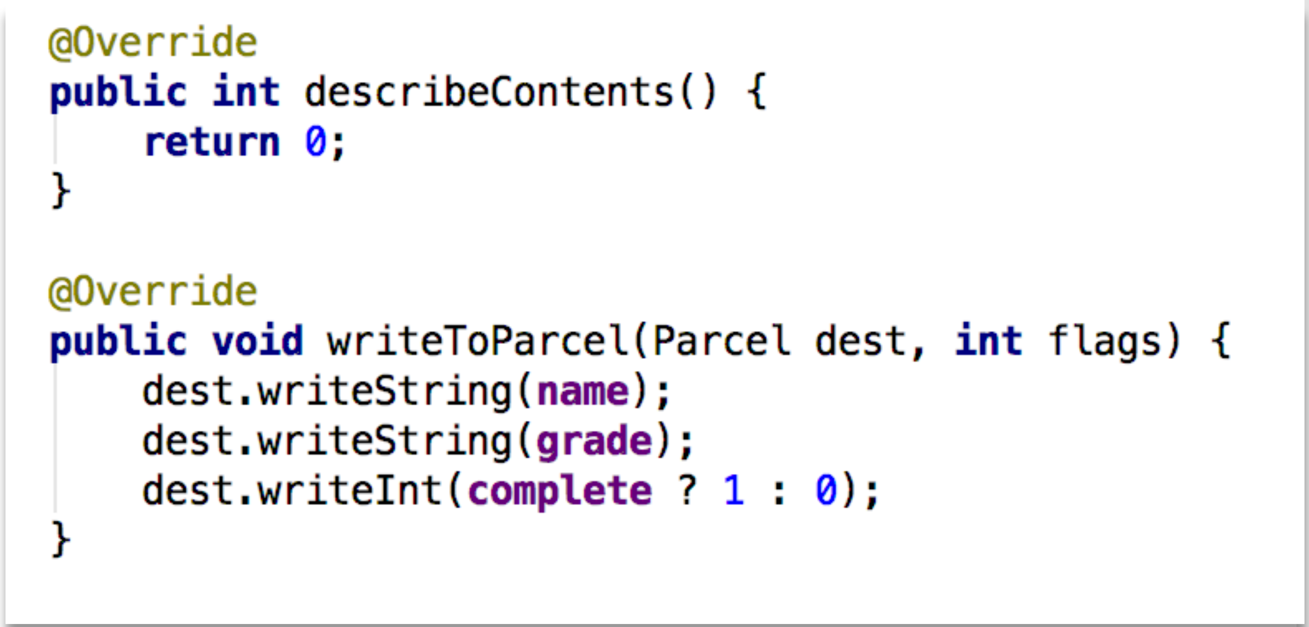
```java
@Override
public void onBackPressed() {
    Intent i = new Intent();
    EditText name = findViewById(R.id.task_name);
    Task task = new Task(name.getText().toString(), grade, complete);
    // need a list even for one item
    ArrayList<Task> tasks = new ArrayList<Task>();
    tasks.add(task);
    i.putParcelableArrayListExtra( name: "TASK_DATA",tasks);
    setResult(RESULT_OK, i);
    super.onBackPressed(); // do not forget

}
```

The magic behind the scenes

- Sadly, there is a bit of messy code that makes all of this work.



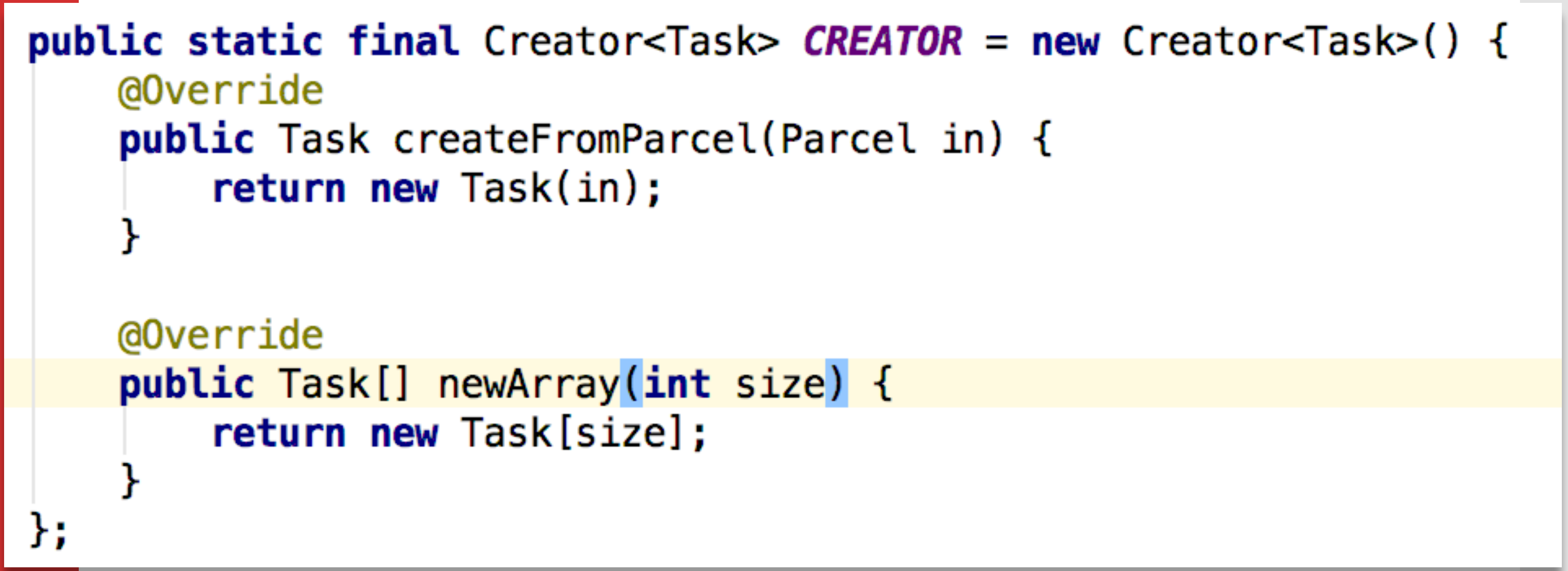
```java
public class Task implements Parcelable {
```



```java
@Override
public int describeContents() {
    return 0;
}

@Override
public void writeToParcel(Parcel dest, int flags) {
    dest.writeString(name);
    dest.writeString(grade);
    dest.writeInt(complete ? 1 : 0);
}
```

Parcel Protocol: CREATOR

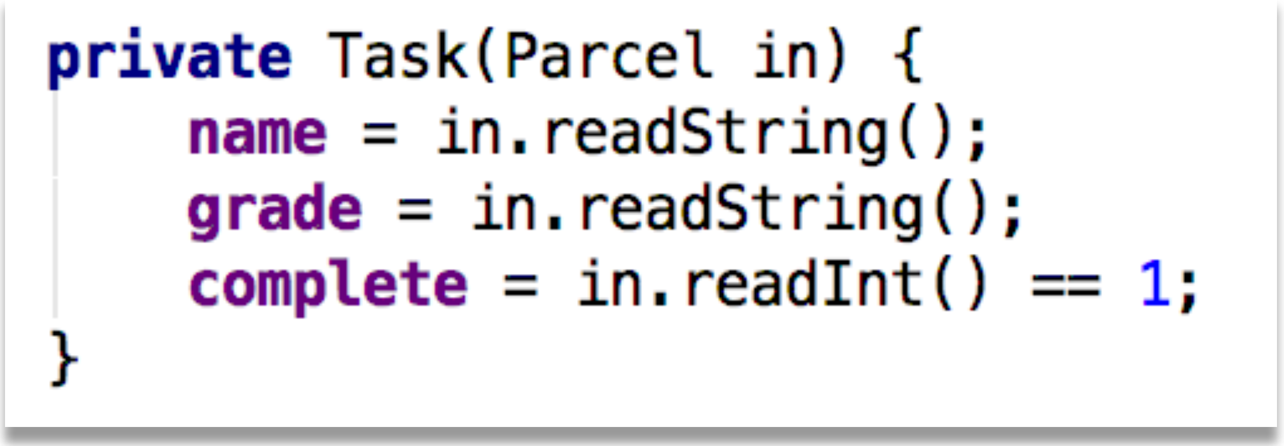
- This constant has to be created and named CREATOR.



```java
public static final Creator<Task> CREATOR = new Creator<Task>() {
    @Override
    public Task createFromParcel(Parcel in) {
        return new Task(in);
    }

    @Override
    public Task[] newArray(int size) {
        return new Task[size];
    }
};
```

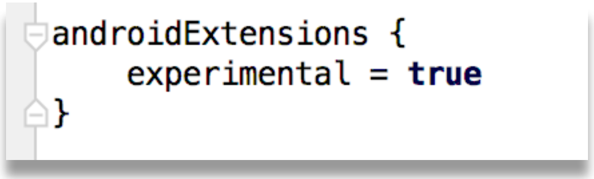
Parcel Protocol: constructor

- Private Constructor that we have to write: reconstructs the object from the Parcel.

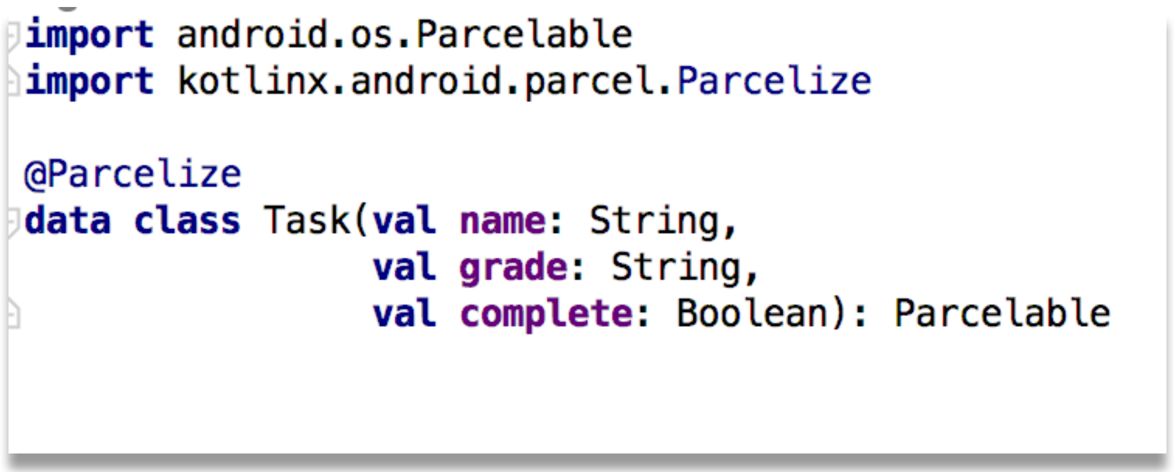


```java
private Task(Parcel in) {
    name = in.readString();
    grade = in.readString();
    complete = in.readInt() == 1;
}
```

In the future



```
androidExtensions {
    experimental = true
}
```



```
import android.os.Parcelable
import kotlinx.android.parcel.Parcelize

@Parcelize
data class Task(val name: String,
                val grade: String,
                val complete: Boolean): Parcelable
```

- (I had to change to a regular class and override toString for minimal changes.)

What if we have complex data?

- What if we need to send a more complex block of data back?
    - We will have to look at other options
- Potential options:
    - Shared Preferences (environment variables)
    - External SQL database
    - Application object

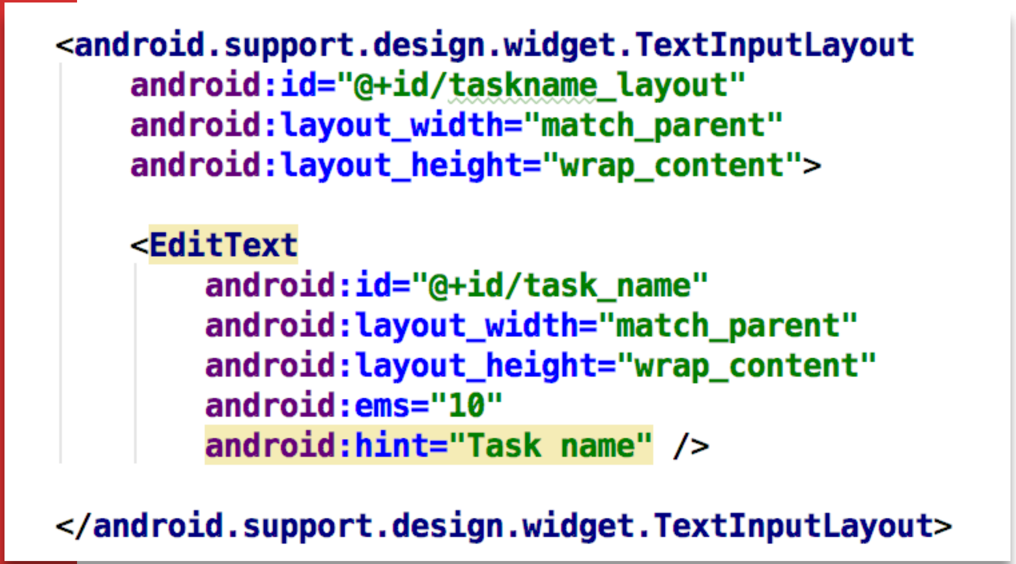
M5

Form design

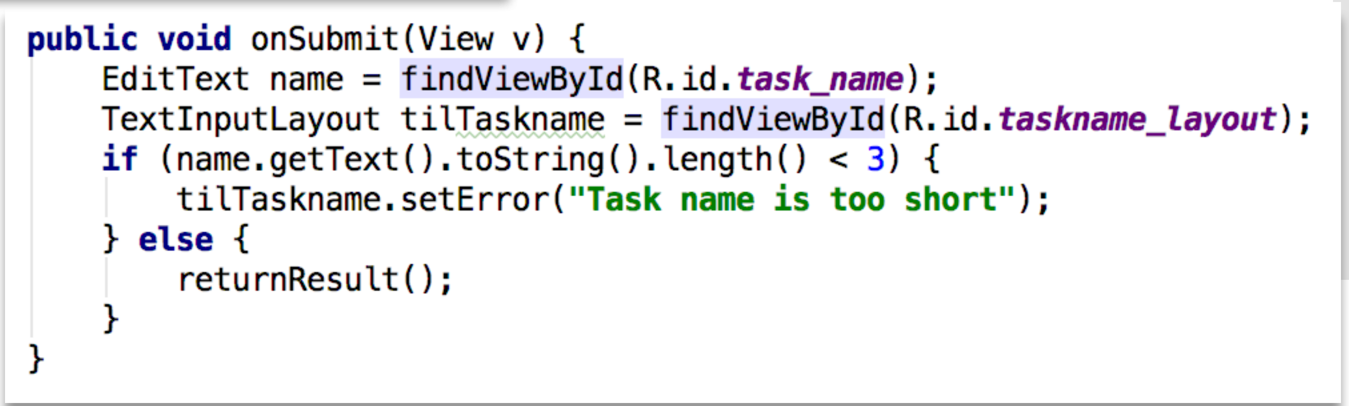
- Event listeners and handlers are needed to handle events from the UI, e.g., if a drop-down item is selected, if a checkbox is selected.
- Focus is also important: which view has the focus and where does it go next?
- Best practice for user input: https://developer.android.com/training/best-user-input.html

Validating input

- One approach: use TextInputLayout from the Design Support Library.
- For example, check for missing or too short answers.



```
<android.support.design.widget.TextInputLayout
    android:id="@+id/taskname_layout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <EditText
        android:id="@+id/task_name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Task name" />

</android.support.design.widget.TextInputLayout>
```



```
public void onSubmit(View v) {
    EditText name = findViewById(R.id.task_name);
    TextInputLayout tilTaskname = findViewById(R.id.taskname_layout);
    if (name.getText().toString().length() < 3) {
        tilTaskname.setError("Task name is too short");
    } else {
        returnResult();
    }
}
```

Module 3
UI components

COS30017 COS80019

Software Development for Mobile Devices

- UI elements
- Toasts and snackbars
- ActionBar/ToolBar and menus
- Form design and data passing
- Lists and RecyclerView
- Splitting up activities into fragments