IT332: Mobile Application Development

Lecture # 17 : Using The Toolbar

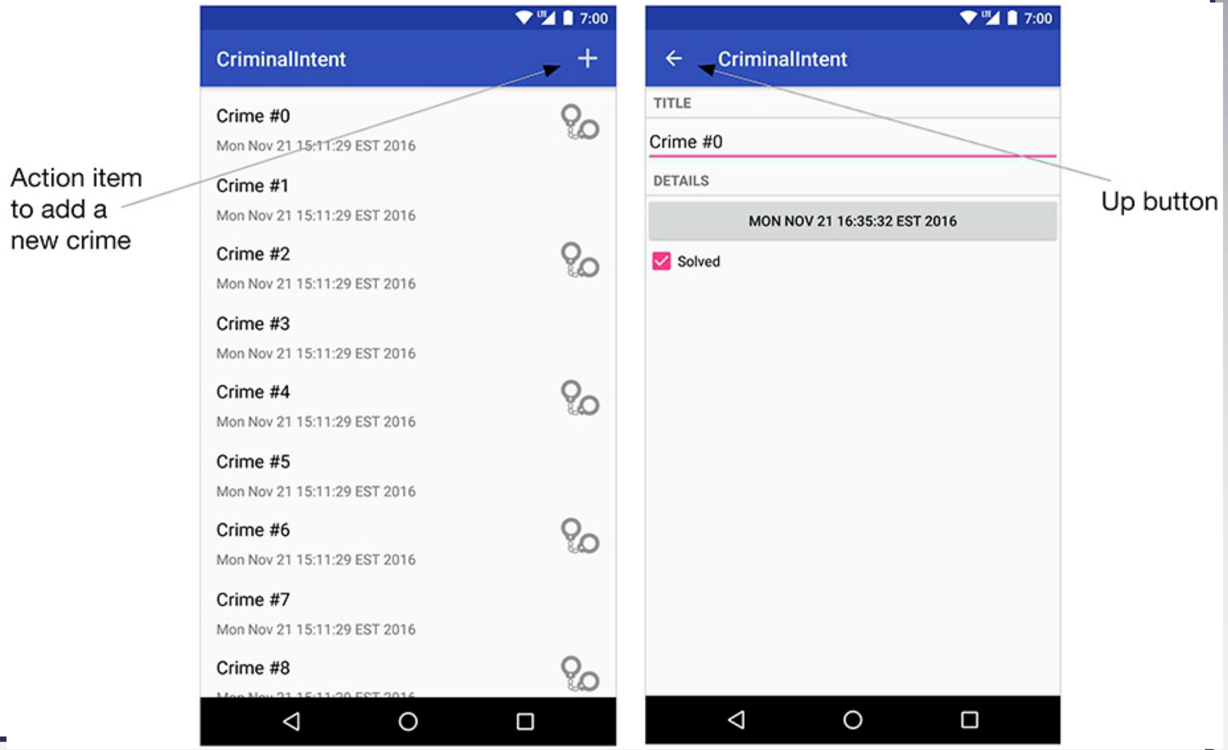
Muhammad Imran

Outline

- Final Objective Today
- The Toolbar
- Menus
- Defining a menu in XML
- Using Android Asset Studio
- Creating the menu
- Responding to menu selections
- Enabling Hierarchical Navigation
- An Alternative Action Item

Final Objective Today

- We will create a menu for CrimeReporting App that will be displayed in the toolbar.
- This menu will have an action item that lets users add a new crime.
- We will also enable the Up button in the toolbar



Action item to add a new crime
CriminalIntent
+
Crime #0
Mon Nov 21 15:11:29 EST 2016
Crime #1
Mon Nov 21 15:11:29 EST 2016
Crime #2
Mon Nov 21 15:11:29 EST 2016
Crime #3
Mon Nov 21 15:11:29 EST 2016
Crime #4
Mon Nov 21 15:11:29 EST 2016
Crime #5
Mon Nov 21 15:11:29 EST 2016
Crime #6
Mon Nov 21 15:11:29 EST 2016
Crime #7
Mon Nov 21 15:11:29 EST 2016
Crime #8
CriminalIntent
TITLE
Crime #0
DETAILS
MON NOV 21 16:35:32 EST 2016
Solved
Up button

The Toolbar

- A key component of any well-designed Android app is the toolbar.
- The toolbar includes actions that the user can take, provides an additional mechanism for navigation, and also provides design consistency and branding.

Menus

- The top-right area of the toolbar is reserved for the toolbar’s menu.
- The menu consists of action items (sometimes also referred to as menu items), which can perform an action on the current screen or on the app as a whole.
- We will add an action item to allow the user to create a new crime.

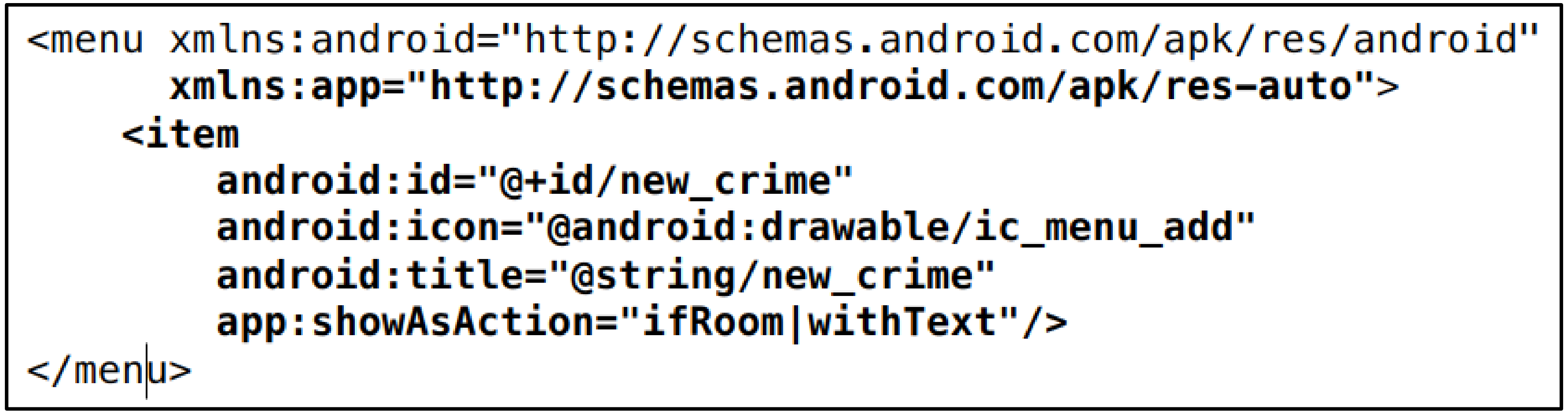
Adding strings for menus (res/values/strings.xml)

- Our menu will require a few string resources.
- We can add them to strings.xml

```
<resources>
    ...
    <string name="date_picker_title">Date of crime:</string>
    <string name="new_crime">New Crime</string>
    <string name="show_subtitle">Show Subtitle</string>
    <string name="hide_subtitle">Hide Subtitle</string>
    <string name="subtitle_format">%1$d crimes</string>

</resources>
```

Defining a menu in XML

- Menus are a type of resource **similar** to layouts.
- We create an **XML description** of a menu and place the file in the res/menu directory of the project.
- Android generates a **resource ID** for the menu file that we then use to **inflate** the menu in **code**.



```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/new_crime"
        android:icon="@android:drawable/ic_menu_add"
        android:title="@string/new_crime"
        app:showAsAction="ifRoom|withText"/>
</menu>
```

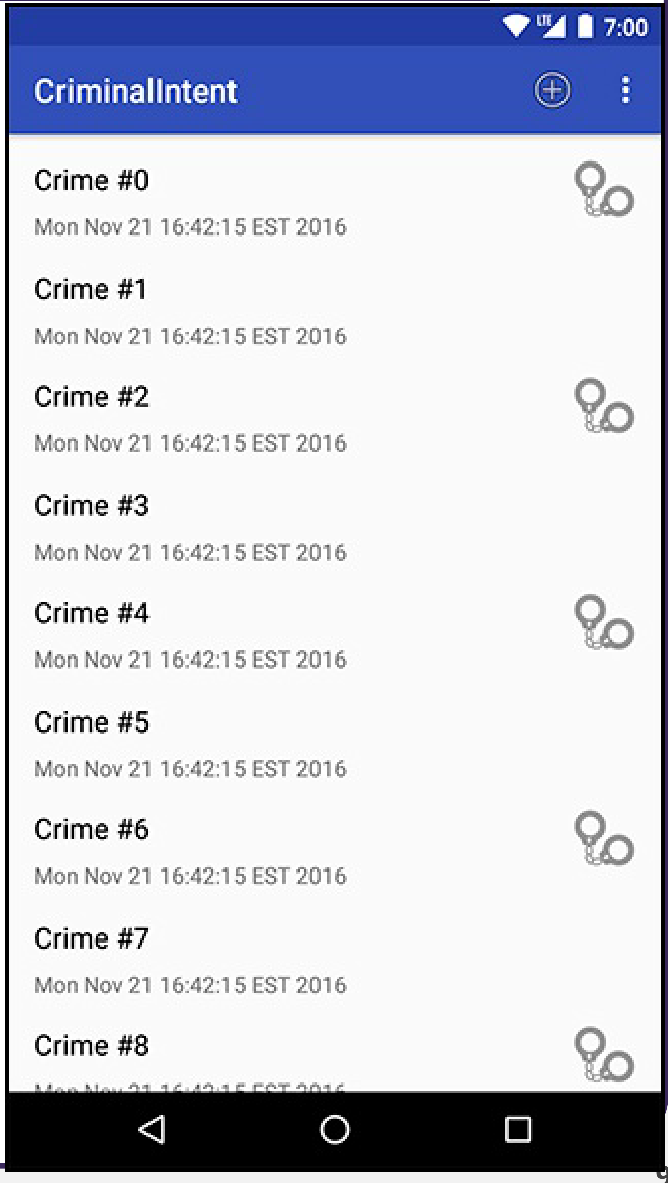
Creating a menu resource for CrimeListFragment (res/menu/fragment_crime_list.xml)

Defining a menu in XML

- The **showAsAction** attribute refers to whether the item will appear in the toolbar itself or in the overflow menu.
- We have piped together two values, **ifRoom** and **withText**, so the item's icon and text will appear in the toolbar **if there is room**.
- **If there is room** for the icon but not the text, then only the icon will be visible.
- **If there is no room** for either, then the item will be relegated to the overflow menu
- Other options for showAsAction include **always** and **never**.
- Using always is **not recommended**; using never is a good choice for **less-common actions**.

Defining a menu in XML

- If we have items in the overflow menu, those items will be represented by the three dots on the far right side of the toolbar
- We should only put action items that users will use frequently in the toolbar to avoid cluttering the screen.


7:00
CriminalIntent
Crime #0
Mon Nov 21 16:42:15 EST 2016
Crime #1
Mon Nov 21 16:42:15 EST 2016
Crime #2
Mon Nov 21 16:42:15 EST 2016
Crime #3
Mon Nov 21 16:42:15 EST 2016
Crime #4
Mon Nov 21 16:42:15 EST 2016
Crime #5
Mon Nov 21 16:42:15 EST 2016
Crime #6
Mon Nov 21 16:42:15 EST 2016
Crime #7
Mon Nov 21 16:42:15 EST 2016
Crime #8

Using Android Asset Studio

- In the **android:icon** attribute, the value **@android:drawable/ic_menu_add** references a system icon.
- A system icon is one that is found on the device rather than in your project's resources.
- **One** alternative is to create our own icons from **scratch**.
- A **second** alternative is to find **system icons** that meet your app's needs and copy them directly into your project's drawable resources.
- The **third** and easiest alternative is to use the **Android Asset Studio**.
- The Asset Studio allows you to create and customize an image to use in the toolbar.

Using Android Asset Studio

- In the **android:icon** attribute, the value **@android:drawable/ic_menu_add** references a system icon.
- A system icon is one that is found on the device rather than in your project's resources.
- **One** alternative is to create our own icons from **scratch**.
- A **second** alternative is to find **system icons** that meet your app's needs and copy them directly into your project's drawable resources.
- The **third** and easiest alternative is to use the **Android Asset Studio**.
- The Asset Studio allows you to create and customize an image to use in the toolbar.

Creating the menu

- Menus are managed by **callbacks** from the Activity class.
- When the menu is needed, Android calls the Activity method **onCreateOptionsMenu**(**Menu**).
- Fragment comes with its **own set** of menu callbacks, which we will implement in CrimeListFragment.
- The methods for **creating** the menu and **responding** to the selection of an action item are:

```
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater)
public boolean onOptionsItemSelected(MenuItem item)
```

Creating the menu

- We can override **onCreateOptionsMenu(Menu, MenuInflater)** to inflate the menu defined in an xml layout file.
- Within this method, we call **MenuInflater.inflate(int, Menu)** and pass in the resource ID of your menu file.
- This populates the **Menu** instance with the items defined in our file.

Inflating a menu resource (CrimeListFragment.java)

```
@Override
public void onResume() {
    super.onResume();
    updateUI();
}

@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_crime_list, menu);
}
...
```

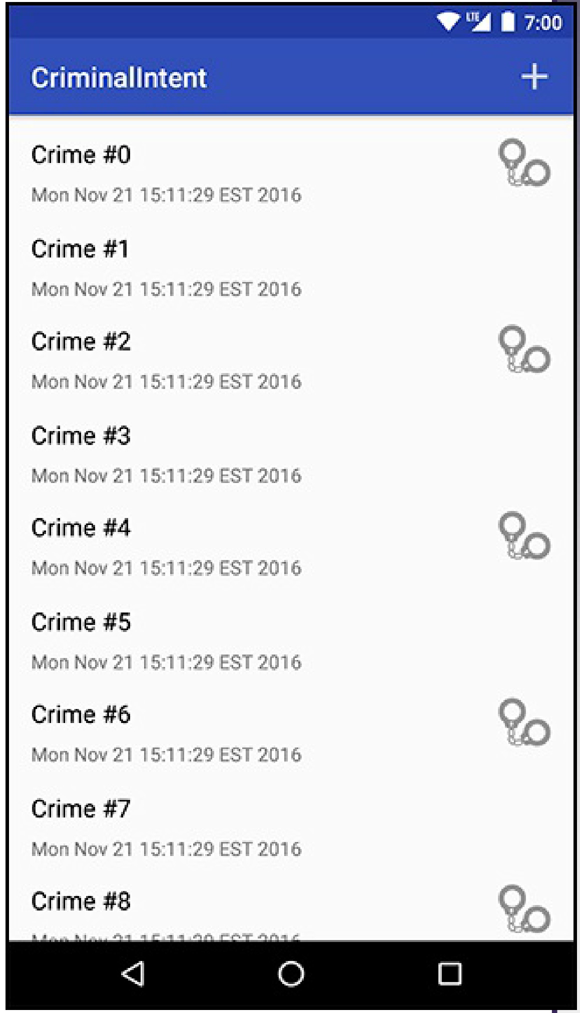
Creating the menu

- The **FragmentManager** is responsible for calling **Fragment.onCreateOptionsMenu(Menu, MenuInflater)** when the activity receives its **onCreateOptionsMenu(…)** callback from the OS.
- We must explicitly tell the FragmentManager that our fragment should receive a call to **onCreateOptionsMenu(…).**
- We do this by calling the following method:

```
public void setHasOptionsMenu(boolean hasMenu)
```

Receiving menu callbacks (CrimeListFragment.java)

```
public class CrimeListFragment extends Fragment {

    private RecyclerView mCrimeRecyclerView;
    private CrimeAdapter mAdapter;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
```



Crime #0
Mon Nov 21 15:11:29 EST 2016
Crime #1
Mon Nov 21 15:11:29 EST 2016
Crime #2
Mon Nov 21 15:11:29 EST 2016
Crime #3
Mon Nov 21 15:11:29 EST 2016
Crime #4
Mon Nov 21 15:11:29 EST 2016
Crime #5
Mon Nov 21 15:11:29 EST 2016
Crime #6
Mon Nov 21 15:11:29 EST 2016
Crime #7
Mon Nov 21 15:11:29 EST 2016
Crime #8

Responding to menu selections

- To respond to the user pressing the New Crime action item, we need a way to add a new Crime to our list of crimes.
- Adding a new crime (CrimeLab.java)

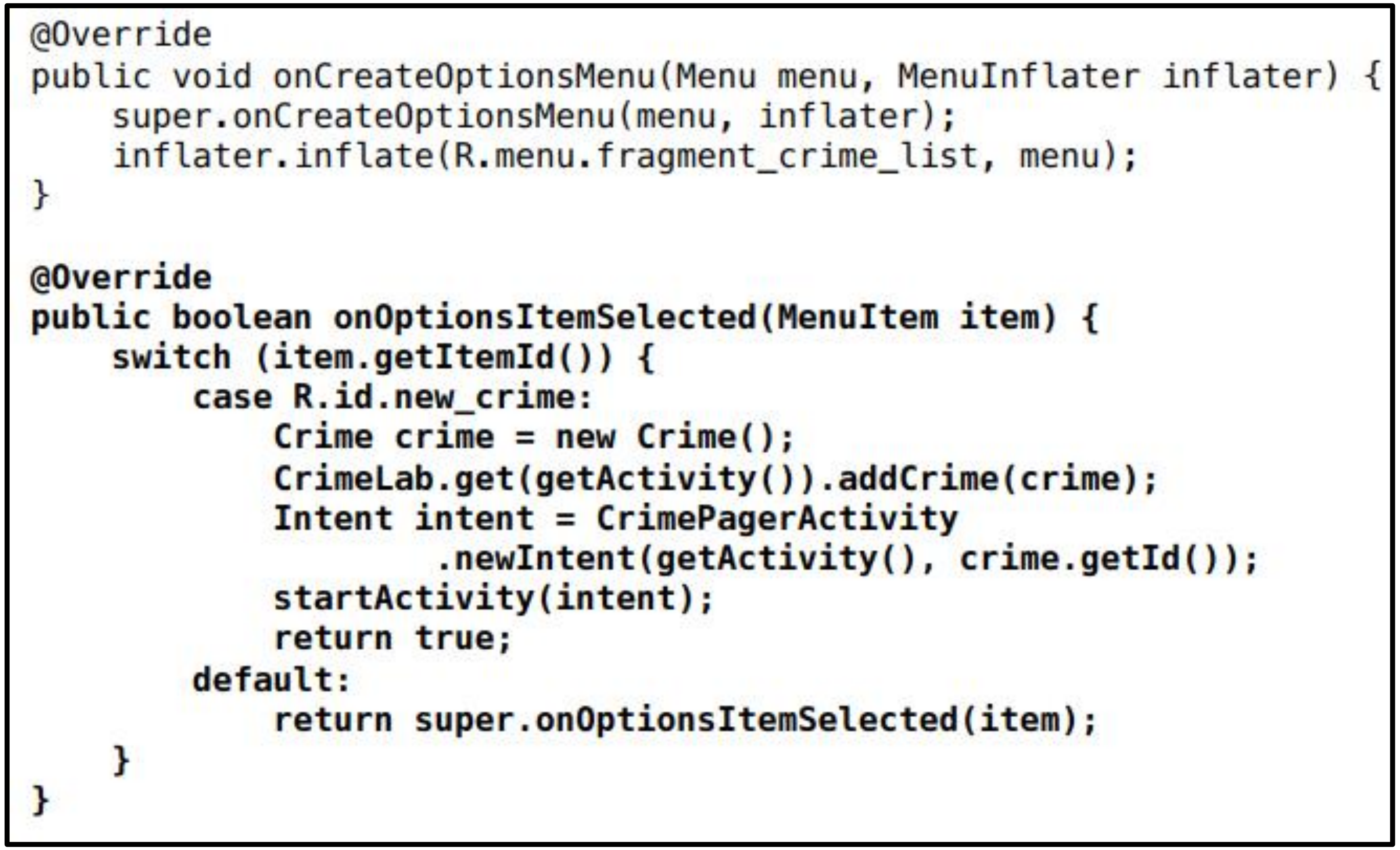
```
public void addCrime(Crime c) {
    mCrimes.add(c);
}
```

- When the user presses an action item, our fragment receives a callback to the method **onOptionsItemSelected(MenuItem).**
- This method receives an instance of **MenuItem** that describes the user’s selection.

Responding to menu selections

- Although our menu only contains one action item, menus often have **more** than one.
- We can determine **which action** item has been selected by **checking the ID** of the **MenuItem** and then respond appropriately.
- This ID corresponds to the ID we assigned to the MenuItem in your **menu file**.

Responding to menu selection (CrimeListFragment.java)



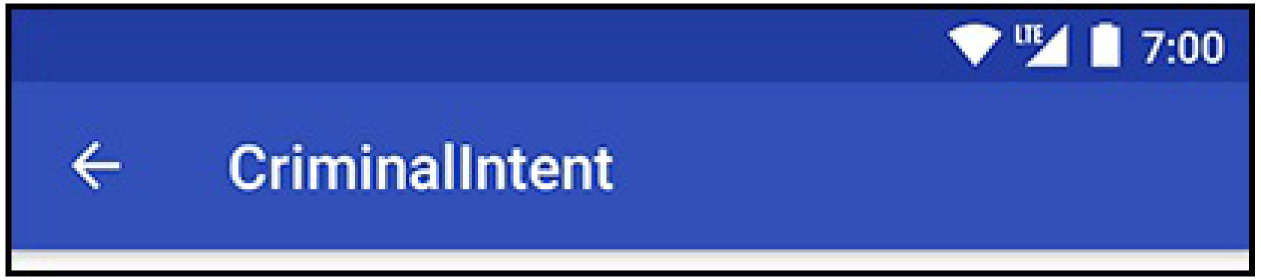
```java
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_crime_list, menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.new_crime:
            Crime crime = new Crime();
            CrimeLab.get(getActivity()).addCrime(crime);
            Intent intent = CrimePagerActivity
                    .newIntent(getActivity(), crime.getId());
            startActivity(intent);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Creating a new Crime, adding it to CrimeLab, and then starting an instance of CrimePagerActivity to edit the new Crime.

Enabling Hierarchical Navigation

- Using the Back Button to navigate around the app is **temporal navigation**.
- It takes us to where we were last.
- Hierarchical navigation, on the other hand, takes us up the app hierarchy. (It is sometimes called **ancestral navigation**.)

```
<activity
    android:name=".CrimePagerActivity"
    android:parentActivityName=".CrimeListActivity">
</activity>
```



An Alternative Action Item

Adding SHOW SUBTITLE action item (res/menu/fragment_crime_list.xml)

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/new_crime"
        android:icon="@android:drawable/ic_menu_add"
        android:title="@string/new_crime"
        app:showAsAction="ifRoom|withText"/>

    <item
        android:id="@+id/show_subtitle"
        android:title="@string/show_subtitle"
        app:showAsAction="ifRoom"/>
</menu>
```

Setting the toolbar's subtitle (CrimeListFragment.java)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    ...
}

private void updateSubtitle() {
    CrimeLab crimeLab = CrimeLab.get(getActivity());
    int crimeCount = crimeLab.getCrimes().size();
    String subtitle = getString(R.string.subtitle_format, crimeCount);

    AppCompatActivity activity = (AppCompatActivity) getActivity();
    activity.getSupportActionBar().setSubtitle(subtitle);
}
    ...
```

Responding to SHOW SUBTITLE action item (CrimeListFragment.java)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.new_crime:
            ...
            return true;
        case R.id.show_subtitle:
            updateSubtitle();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Toggling the action item title

- Now the subtitle is visible, but the action item still reads SHOW SUBTITLE.
- It would be better if the action item toggled its title and function to show or hide the subtitle.
- When **onOptionsItemSelected(MenuItem)** is called, we are given the MenuItem that the user pressed as a parameter.
- We could update the text of the SHOW SUBTITLE item in this method, but the subtitle change would be lost as you rotate the device and the toolbar is re-created.

Toggling the action item title

- A better solution is to update the SHOW SUBTITLE **MenuItem** in **onCreateOptionsMenu(…)** and trigger a re-creation of the toolbar when the user presses on the subtitle item.
- This allows us to share the code for updating the action item in the case that the user selects an action item or the toolbar is re-created.

Keeping subtitle visibility state (CrimeListFragment.java)

```
public class CrimeListFragment extends Fragment {

    private RecyclerView mCrimeRecyclerView;
    private CrimeAdapter mAdapter;
    private boolean mSubtitleVisible;
    ...
```

Updating a MenuItem (CrimeListFragment.java)

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_crime_list, menu);

    MenuItem subtitleItem = menu.findItem(R.id.show_subtitle);
    if (mSubtitleVisible) {
        subtitleItem.setTitle(R.string.hide_subtitle);
    } else {
        subtitleItem.setTitle(R.string.show_subtitle);
    }
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.new_crime:
            ...
        case R.id.show_subtitle:
            mSubtitleVisible = !mSubtitleVisible;
            getActivity().invalidateOptionsMenu();
            updateSubtitle();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Showing or hiding the subtitle (CrimeListFragment.java)

```
private void updateSubtitle() {
    CrimeLab crimeLab = CrimeLab.get(getActivity());
    int crimeCount = crimeLab.getCrimes().size();
    String subtitle = getString(R.string.subtitle_format, crimeCount);

    if (!mSubtitleVisible) {
        subtitle = null;
    }

    AppCompatActivity activity = (AppCompatActivity) getActivity();
    activity.getSupportActionBar().setSubtitle(subtitle);
}
```

Two more Issues

- **First**, when creating a new crime and then returning to CrimeListActivity with the Back button, the number of crimes in the subtitle will not update to reflect the new number of crimes.
- **Second**, the visibility of the subtitle is lost across rotation.

- For the first issue: The solution is to update the subtitle text when returning to **CrimeListActivity**. Trigger a call to **updateSubtitle()** in **onResume().**
- Our **updateUI()** method is already called in onResume() and onCreateView(…).
- Add a call to updateSubtitle() to the updateUI() method.

Showing the most recent state (CrimeListFragment.java)

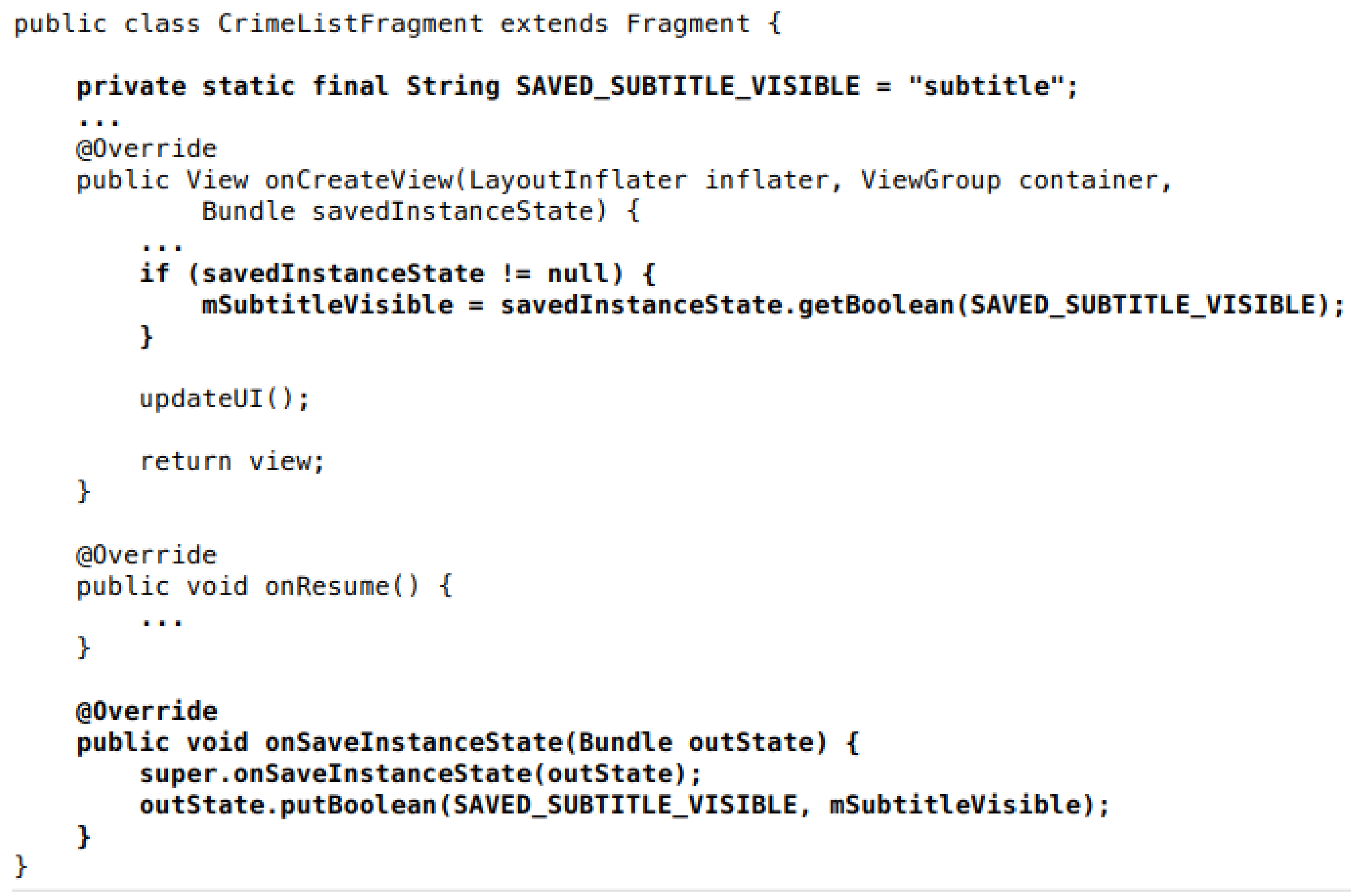
```
private void updateUI() {
    CrimeLab crimeLab = CrimeLab.get(getActivity());
    List<Crime> crimes = crimeLab.getCrimes();

    if (mAdapter == null) {
        mAdapter = new CrimeAdapter(crimes);
        mCrimeRecyclerView.setAdapter(mAdapter);
    } else {
        mAdapter.notifyDataSetChanged();
    }

    updateSubtitle();
}
```

The second Issue

- To solve the rotation issue we can save the **mSubtitleVisible** instance variable across rotation with the saved instance state mechanism.

Saving subtitle visibility (CrimeListFragment.java)

- To solve the rotation issue we can save the **mSubtitleVisible** instance variable across rotation with the saved instance state mechanism.



```
public class CrimeListFragment extends Fragment {

    private static final String SAVED_SUBTITLE_VISIBLE = "subtitle";
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        ...
        if (savedInstanceState != null) {
            mSubtitleVisible = savedInstanceState.getBoolean(SAVED_SUBTITLE_VISIBLE);
        }

        updateUI();

        return view;
    }

    @Override
    public void onResume() {
        ...
    }

    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putBoolean(SAVED_SUBTITLE_VISIBLE, mSubtitleVisible);
    }
}
```

InClass Task 12 (Deleting Crimes)

- Once a crime has been created in CrimeReporting App, there is no way to erase that crime from the official record.
- For this task, add a new action item to the CrimeFragment that allows the user to delete the current crime.
- Once the user presses the new delete action item, be sure to pop the user back to the previous activity with a call to the finish() method on the CrimeFragment’s hosting activity.

InClass Task 13 (Plural String Resources)

- The subtitle is not grammatically correct when there is a single crime.
- 1 crimes just does not show the right amount of attention to detail for your taste.
- For this task, correct this subtitle text.
- You could have two different strings and determine which one to use in code, but this will quickly fall apart when you localize your app for different languages.
- A better option is to use plural string resources (sometimes also called quantity strings).
- First, define a plural string in your strings.xml file.

```
<plurals name="subtitle_plural">
    <item quantity="one">%1$d crime</item>
    <item quantity="other">%1$d crimes</item>
</plurals>
```

- Then, use the getQuantityString method to correctly pluralize the string.

```
int crimeSize = crimeLab.getCrimes().size();
String subtitle = getResources()
    .getQuantityString(R.plurals.subtitle_plural, crimeSize, crimeSize);
```

InClass Task 14 (An Empty View for the RecyclerView)

- Currently, when CriminalIntent launches it displays an empty RecyclerView – a big white void.
- You should give users something to interact with when there are no items in the list.
- For this challenge, display a message like, There are no crimes and add a button to the view that will trigger the creation of a new crime.
- Use the setVisibility method that exists on any View class to show and hide this new placeholder view when appropriate.

Recommended Readings

- Page # 247 to 268, Chapter 13: The Toolbar from Android Programming: The Big Nerd Ranch Guide, 3rd Edition by Bill Phillips, 2017