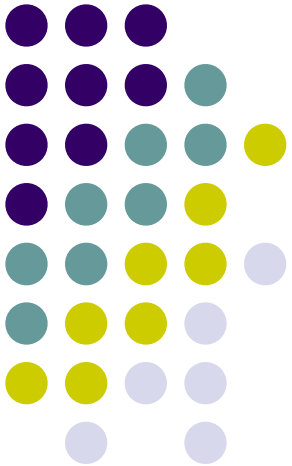
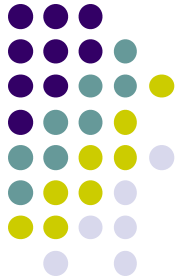
Android Programming

03 – User Interfaces

Part - 2

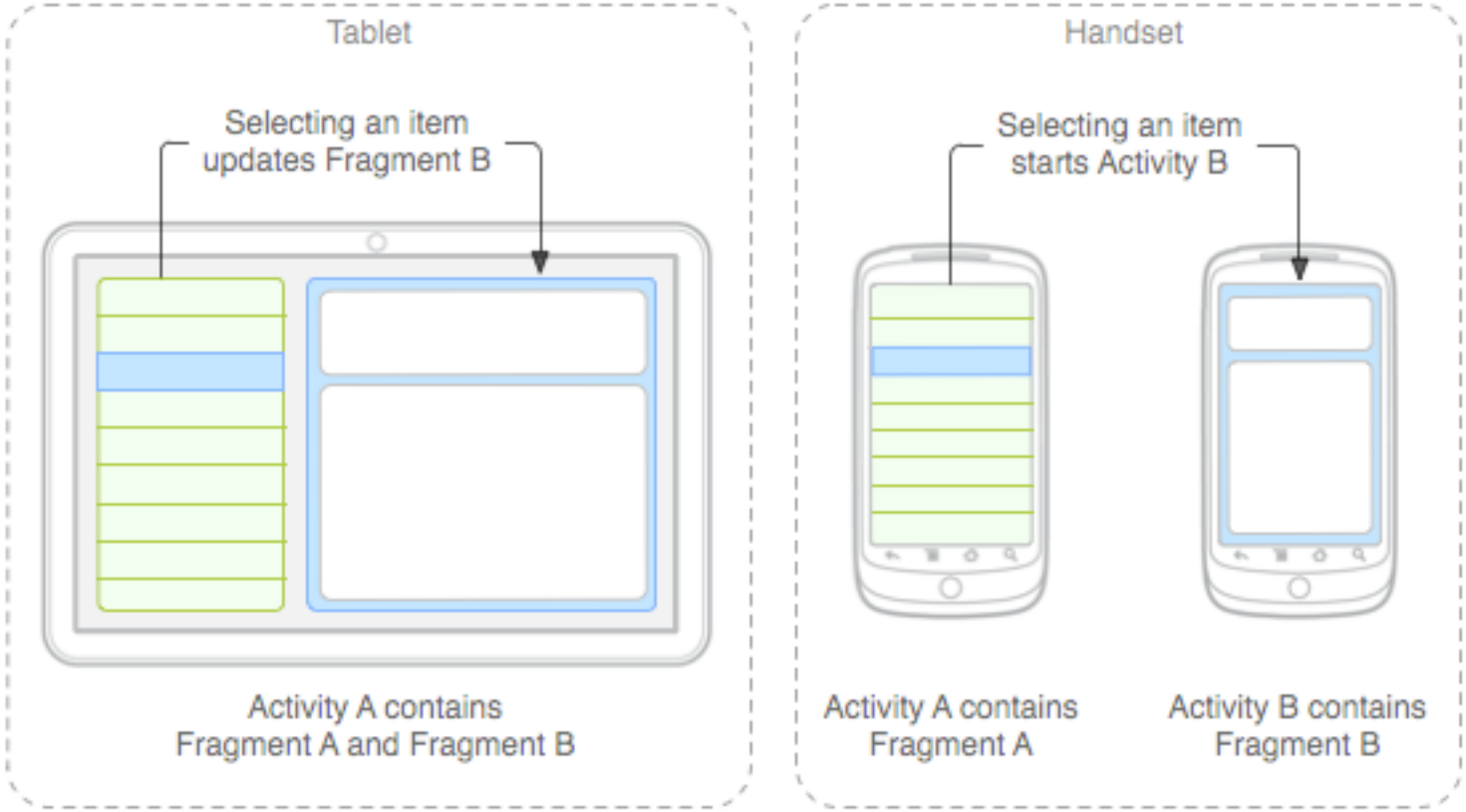


Fragments



- A Fragment represents a behavior or a portion of user interface in an Activity.
- You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities.
- Requires min API level 11
- Mostly used in tablets with larger displays

Using Fragments



Coding Fragments

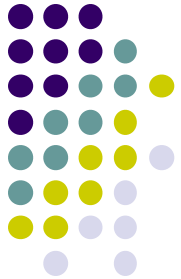
- Extend the Fragment class, implement required methods

```
public static class ExampleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.example_fragment, container,
false);
    }
}
```

```
<?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>
```

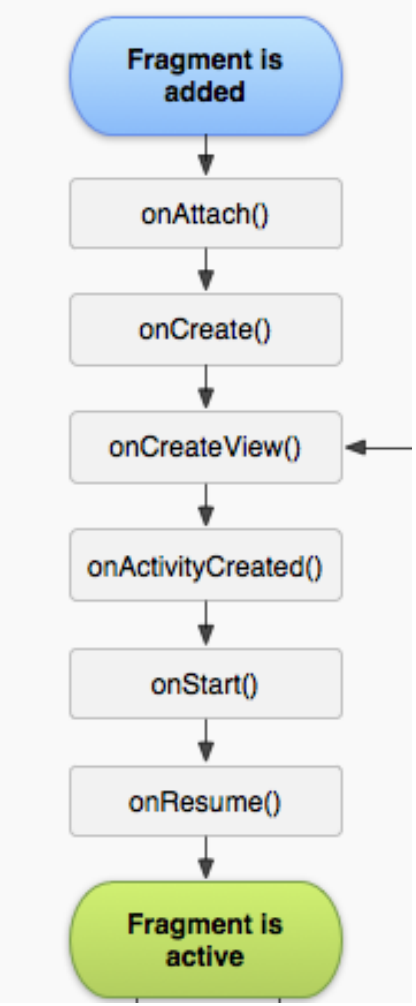
If your fragment is a subclass of ListFragment, the default implementation returns a ListView from onCreateView(), so you don't need to implement it. In this case, implement onActivityCreated()

Managing Fragments

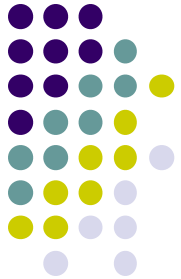


- To manage the fragments in your activity, you need to use FragmentManager. To get it, call getFragmentManager() from your activity.
- Some methods:
  - findFragmentById(), findFragmentByTag(), popBackStack(), addToBackStack()

Fragment Lifecycle

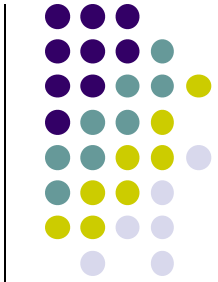


Fragment Lifecycle Methods



- onAttach()
  - This signifies that the Activity has attached fragment on it and it doesn't mean that all the views of Activity are created not it means that Activity is fully functional. This is just a point where **you can reference the activity**
- onCreate()
  - Just a point which shows that Fragment is in the process of creation and in this method just try to **access those values** that you have saved on **onSavedInstanceState(Bundle outState)**

Fragment Lifecycle Methods



- onCreateView()
  - Here **we inflate the layout** or simply create the view and further if you have to do anything that takes reference to Activity don't do it like creating accessing views of Activity etc because,this place doesn't ensure that hosting Activity is fully functional
- onActivityCreated()
  - This place signifies that **our hosting Activity views are created and hosting Activity is functional** and this is the right place to do all your Activity related task. (Activity onCreate() has completed)

Fragment Transitions

- To make fragment transactions in your activity (such as add, remove, or replace a fragment), you must use APIs from FragmentTransaction.

Create fragment transition

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

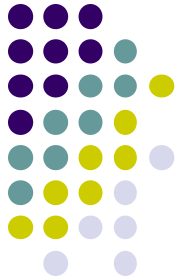
Add fragment to a viewgroup

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

Replace existing fragment

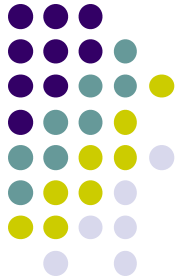
```
// Replace whatever is in the fragment_container view with this fragment,
// and add the transaction to the back stack
transaction.replace(R.id.fragment_container, newFragment);
transaction.addToBackStack(null);

// Commit the transaction
transaction.commit();
```

Transition Methods



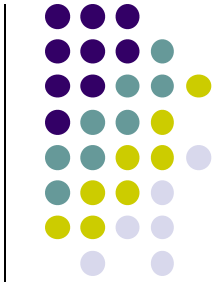
- add()
  - Add a fragment to the activity state. This fragment may optionally also have its view (if Fragment.onCreateView returns non-null) into a container view of the activity.
- attach()
  - add() must be called before
  - Re-attach a fragment after it had previously been deatched from the UI with detach(Fragment). This causes its view hierarchy to be re-created, attached to the UI, and displayed.

Transition Methods



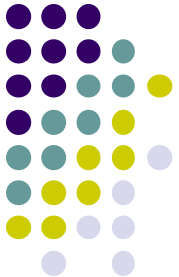
- detach()
  - Detach the given fragment from the UI. This is the same state as when it is put on the back stack: the fragment is removed from the UI, however its state is still being actively managed by the fragment manager. When going into this state its view hierarchy is destroyed.
- remove()
  - Remove an existing fragment. If it was added to a container, its view is also removed from that container. Fragment state lost

Transition Methods



- replace()
  - Replace an existing fragment that was added to a container. This is essentially the same as calling remove(Fragment) for all currently added fragments that were added with the same containerViewId and then add(int, Fragment, String) with the same arguments given here

Communicating with the Activity



Access a component of Activity from Fragment or vice versa:

```
View listView = getActivity().findViewById(R.id.list);
```

```
ExampleFragment fragment = (ExampleFragment)
getFragmentManager().findFragmentById(R.id.example_fragment);
```

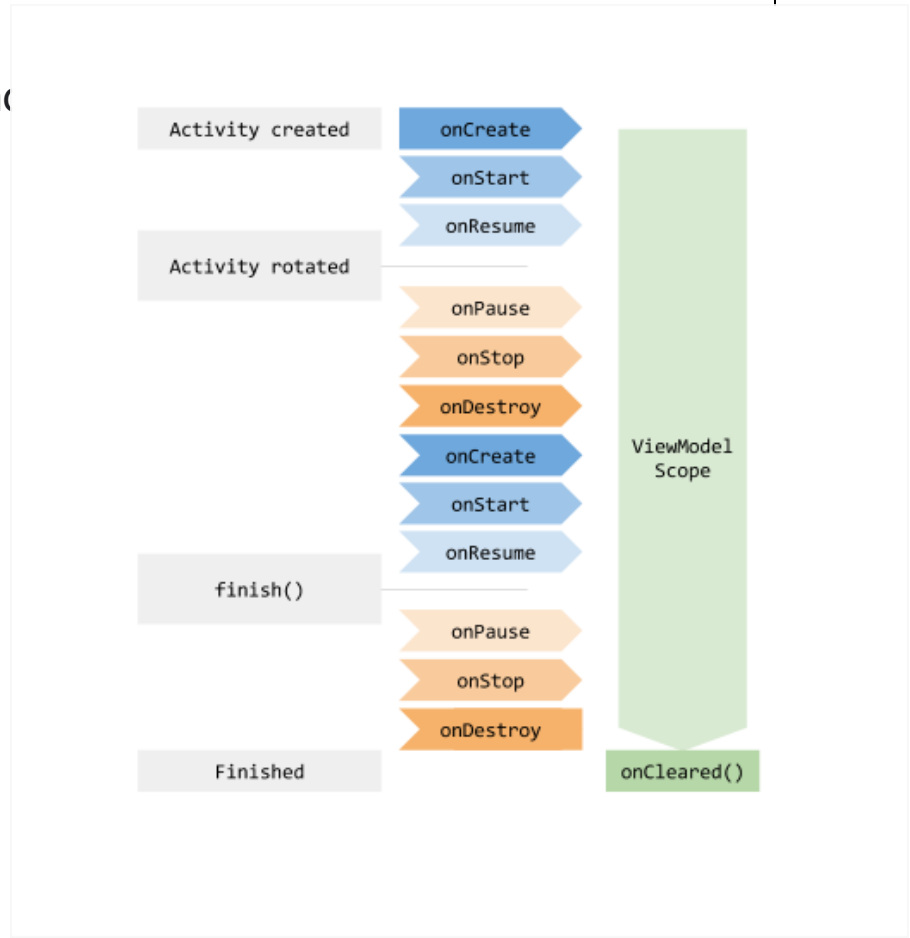
Creating event call backs:

```
public static class FragmentA extends ListFragment {
    // Container Activity must implement this interface
    public interface OnArticleSelectedListener {
        public void onArticleSelected(Uri articleUri);
    }
}
```

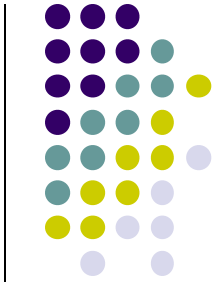
```
public static class FragmentA extends ListFragment {
    OnArticleSelectedListener mListener;
    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        try {
            mListener = (OnArticleSelectedListener) activity;
        } catch (ClassCastException e) {
            throw new ClassCastException(activity.toString() + " must implement
OnArticleSelectedListener");
        }
    }
}
```

ViewModel for Fragment Communication

- The ViewModel class is designed to store and manage UI-related data in a lifecycle conscious way.
- The ViewModel class allows data to survive configuration changes such as screen rotations.
- ViewModel objects are scoped to the Lifecycle passed to the ViewModelProvider when getting the ViewModel.
- The ViewModel remains in memory until the Lifecycle it's scoped to goes away permanently: in the case of an activity, when it finishes, while in the case of a fragment, when it's detached.



Coding and Sharing ViewModel



- The activity does not need to do anything, or know anything about this communication.
- Fragments don't need to know about each other besides the `SharedViewModel` contract.
- Each fragment has its own lifecycle, and is not affected by the lifecycle of the other one.

Common ViewModel:

```
public class SharedViewModel extends ViewModel {
    private final MutableLiveData<Item> selected = new MutableLiveData<Item>();

    public void select(Item item) {
        selected.setValue(item);
    }

    public LiveData<Item> getSelected() {
        return selected;
    }
}
```

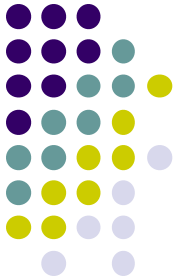
Fragment sending data:

```
public class ListFragment extends Fragment {
    private SharedViewModel model;

    public void onViewCreated(@NonNull View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        model = new ViewModelProvider(requireActivity()).get(SharedViewModel.class);
        itemSelector.setOnClickListener(item -> {
            model.select(item);
        });
    }
}
```

Fragment receiving data:

```
public class DetailFragment extends Fragment {

    public void onViewCreated(@NonNull View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        SharedViewModel model = new ViewModelProvider(requireActivity()).get(SharedViewModel.class)
        model.getSelected().observe(getViewLifecycleOwner(), item -> {
            // Update the UI.
        });
    }
}
```

Styles & Themes



- A style is a collection of properties that specify the look and format for a View or window.
- A theme is group of ‘style’s applied to an entire Activity or application, rather than an individual View

Defining Styles

- To create a set of styles, save an XML file in the res/ values/ directory of your project as follows, filename is arbitrary

inheritance of built-in style

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CodeFont" parent="@android:style/TextAppearance.Medium">
        <item name="android:layout_width">fill_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

inheritance of user defined style

```
<style name="CodeFont.Red">
        <item name="android:textColor">#FF0000</item>
    </style>
```

Applying Styles & Themes

- Two ways to apply a style
  - To an individual View, by adding the style attribute to a View element in the XML for your layout
  - Or, to an entire Activity or application, by adding the android:theme attribute to the <activity> or <application> element in the Android manifest

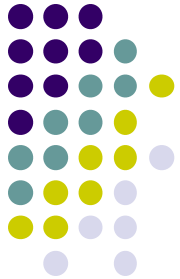
values/styles.xml

```
<color name="custom_theme_color">#b0b0ff</color>
<style name="CustomTheme" parent="android:Theme.Light">
    <item name="android:windowBackground">@color/custom_theme_color</item>
    <item name="android:colorBackground">@color/custom_theme_color</item>
</style>
```

AndroidManifest.xml

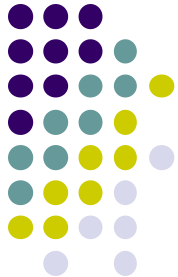
```
 <application
…
        android:theme="@style/CustomTheme" >
…
```

Customizing Background



- You can shape the background style of views, like adding rounded corners.
  - Create an XML file under res/drawable
  - Root tag must be <shape>
  - refer to the xml file from view's XML:
    - android:background="@drawable/myshape"
- See code of the XML on the next slide

Shape XML

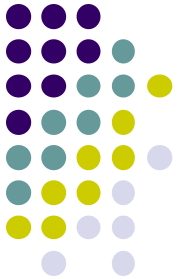


Custom Shape File under res/drawable

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
android:shape="rectangle">
    <gradient android:startColor="#FFFFFF" android:endColor="#CCCCCC"
android:angle="270"/>
     <corners android:radius="4dp"/>
     <padding android:left="4dp" android:top="4dp" android:right="4dp"
android:bottom="4dp"/>
     <stroke android:width="4dp" android:color="#000000"/>
</shape>
```

Referring to the custom shape with android:background

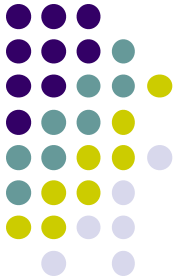
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" android:background="@drawable/roundedbackground"
android:layout_margin="4dp”>
…
</LinearLayout>
```



Menus

- Options Menu
  - The options menu is the primary collection of menu items for an activity. It's where you should place actions that have a global impact on the app, such as "Search," "Compose email," and "Settings."
- Context Menu and contextual action mode
  - A context menu is a floating menu that appears when the user performs a long-click on an element.
  - When developing for Android 3.0 and higher, you should instead use the contextual action mode to enable actions on selected content.
- Popup Menu
  - A popup menu displays a list of items in a vertical list that's anchored to the view that invoked the menu.

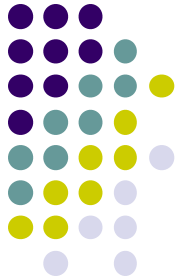
Defining a Menu in XML



- For all menu types, Android provides a standard XML format to define menu items.
- Instead of building a menu in your activity's code, you should define a menu and all its items in an XML menu resource.
- You can then inflate the menu resource (load it as a Menu object) in your activity or fragment.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game"
          android:icon="@drawable/ic_new_game"
          android:title="@string/new_game"
          android:showAsAction="ifRoom"/>
    <item android:id="@+id/help"
          android:icon="@drawable/ic_help"
          android:title="@string/help" />
</menu>
```

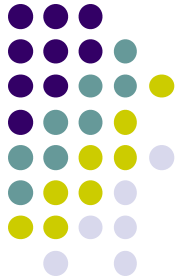
Creating an Options Menu



- Registered with onCreateOptionsMenu() in activity
- XML menu resources for layout, filled with MenuInflator (from activity getMenuInflator())
- Items can be placed on action bar (after android 3.0) by using showAsAction attribute
- Devices after Android 3.0 might not support the options button

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

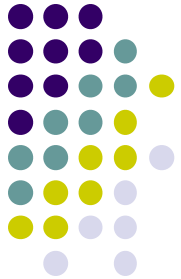
Interacting with Options Menu



- Upon click the menu calls Activity's onOptionsItemSelected() method

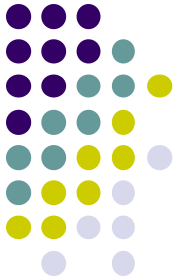
```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case R.id.new_game:
            newGame();
            return true;
        case R.id.help:
            showHelp();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

How to create an App Menu



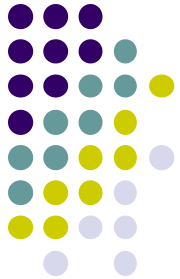
- If your application contains multiple activities and some of them provide the same Options Menu, consider creating an activity that implements nothing except the onCreateOptionsMenu() and onOptionsItemSelected() methods. Then extend this class for each activity that should share the same Options Menu.

Creating Floating Context Menus



- Register the View to which the context menu should be associated by calling registerForContextMenu() and pass it the View.
- Implement the onCreateContextMenu() method in your Activity or Fragment
- Implement onContextItemSelected().
  - If different views provide different context menus, you should check which view is selected in this method

Creating Floating Context Menus (cont'd)



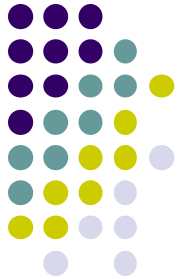
Creating the menu

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.context_menu, menu);
}
```

Menu Interaction

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();
    switch (item.getItemId()) {
        case R.id.edit:
            editNote(info.id);
            return true;
        case R.id.delete:
            deleteNote(info.id);
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}
```

Using the Contextual Action Mode



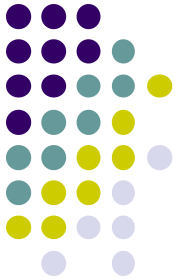
- Valid After Android 3.0
- Enables context menu items in Action Bar
- Usage:
  1. Implement the ActionMode.Callback interface. In its callback methods, you can specify the actions for the contextual action bar, respond to click events on action items, and handle other lifecycle events for the action mode
  2. Call startActionMode() when you want to show the bar (such as when the user long-clicks the view)

Using the Contextual Action Mode (cont'd)

```
private ActionMode.Callback mActionModeCallback = new ActionMode.Callback() {

    // Called when the action mode is created; startActionMode() was called
    @Override
    public boolean onCreateActionMode(ActionMode mode, Menu menu) {
        // Inflate a menu resource providing context menu items
     return true;
    }

    // Called each time the action mode is shown. Always called after
onCreateActionMode, but
    // may be called multiple times if the mode is invalidated.
    @Override
    public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
        return false; // Return false if nothing is done
    }

    // Called when the user selects a contextual menu item
    @Override
    public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
}

    // Called when the user exits the action mode
    @Override
    public void onDestroyActionMode(ActionMode mode) {
        mActionMode = null;
    }
};
```

Implementing the callback

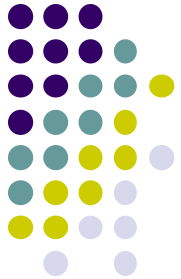
Using the Contextual Action Mode (cont'd)



```
someView.setOnLongClickListener(new View.OnLongClickListener() {
    // Called when the user long-clicks on someView
    public boolean onLongClick(View view) {
        if (mActionMode != null) {
            return false;
        }

        // Start the CAB using the ActionMode.Callback defined above
        mActionMode = getActivity().startActionMode(mActionModeCallback);
        view.setSelected(true);
        return true;
    }
});
```

Displaying the menu on an item long click

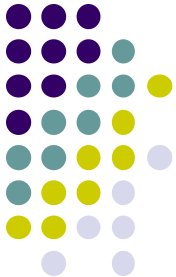
Creating Popup Menus



- A PopupMenu is a modal menu anchored to a View. It appears below the anchor view if there is room, or above the view otherwise.
- Usage:

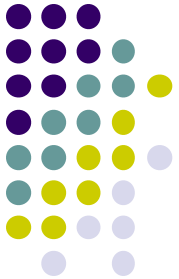
1. Instantiate a PopupMenu with its constructor, which takes the current application Context and the View to which the menu should be anchored.
2. Use MenuInflater to inflate your menu resource into the Menu object returned by PopupMenu.getMenu(). On API level 14 and above, you can use PopupMenu.inflate() instead.
3. Call PopupMenu.show().

Coding Popup Menus



```
btn5.setOnClickListener(new View.OnClickListener() {

@Override
public void onClick(View v) {
PopupMenu popMenu = new PopupMenu(MenuExampleActivity.this, v);
popMenu.getMenuInflater().inflate(R.menu.popupmenu, popMenu.getMenu());
popMenu.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {

@Override
public boolean onMenuItemClick(MenuItem item) {
Toast.makeText(MenuExampleActivity.this, item.getTitle(), Toast.LENGTH_SHORT).show();
return true;
}
});
popMenu.show();


}
});
```

More About The Action Bar



You can display both text and image of the options menu items

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
          xmlns:app="http://schemas.android.com/apk/res-auto">
    <item android:id="@+id/menu_save"
          android:icon="@drawable/ic_menu_save"
          android:title="@string/menu_save"
          app:showAsAction="ifRoom|withText" />
</menu>
```

You can insert different views

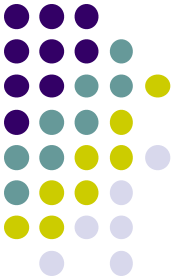
```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android=http://schemas.android.com/apk/res/android
                    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item android:id="@+id/menu_search"
          android:title="@string/menu_search"
          android:icon="@drawable/ic_menu_search"
          app:showAsAction="always"
          app:actionViewClass="android.support.v7.widget.SearchView" />
</menu>
```

android:actionLayout can also be used

In onCreateOptionsMenu() you can access the view:

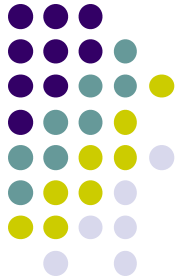
```
SearchView searchView = (SearchView) menu.findItem(R.id.menu_search).getActionView();
```

Example



- Create an app with a main menu of three items
- When each item clicked display an activity
- From the main activity pass an object to another activity and use it

More About the Action Bar



- API level 11 or higher
- For previous levels, V7 support lib is required
- Menu items can be added using menu.xml "showAsAction" attribute
- Display and hide:

```
public void showBar(View v){
     ActionBar bar = getSupportActionBar();
     if(bar.isShowing()){
          bar.hide();
     }else{
          bar.show();
     }
}
```

Navigating up with the App Icon

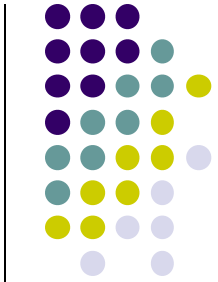
- Enable up button

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_details);

    ActionBar actionBar = getSupportActionBar();
    actionBar.setDisplayHomeAsUpEnabled(true);
    ...
}
```

- Edit manifest, set parent

```
<activity
        android:name="com.example.myfirstapp.MainActivity" .../>
 <!-- A child of the main activity -->
    <activity
        android:name="com.example.myfirstapp.DisplayMessageActivity"
        android:label="@string/title_activity_display_message"
        android:parentActivityName="com.example.myfirstapp.MainActivity" >
        <!-- Parent activity meta-data to support API level 7+ -->
        <meta-data
            android:name="android.support.PARENT_ACTIVITY"
            android:value="com.example.myfirstapp.MainActivity" />
    </activity>
```

View Binding



- No need to call findViewById() to access views.

1- Enable View Binding in module.gradle:

```
buildFeatures {
        viewBinding = true
    }
```

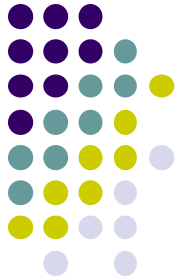
1. Call the static inflate() method included in the generated binding class. This creates an instance of the binding class for the activity to use.
2. Get a reference to the root view by either calling the getRoot() method or using Kotlin property syntax.
3. Pass the root view to setContentView() to make it the active view on the screen.

```
private ResultProfileBinding binding;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    binding = ResultProfileBinding.inflate(getLayoutInflater());
    View view = binding.getRoot();
    setContentView(view);
}
```

You can now use the instance of the binding class to reference any of the views:

```
binding.getName().setText(viewModel.getName());
binding.getButton().setOnClickListener(new
View.OnClickListener() {
    viewModel.userClicked()
});
```

Data Binding



Data binding is the process of integrating views in an XML layout with data objects.

Enable View Binding in module.gradle:

```
buildFeatures {
        dataBinding = true
    }
```

1. Declare a `<layout>` tag, which will wrap your existing layout file at the root level
2. Declare variables under the `<data>` tag, which will go under the `<layout>` tag
3. Declare necessary expressions to bind data inside the view elements

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

    <data>
        <variable name="txt1" type="java.lang.String" />
        <variable name="person1"  type="edu.sabanciuniv.viewbindingex1.Person" />
    </data>

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity">
<TextView
            android:id="@+id/textView2"
            android:text="@{txt1}"
         />

        <TextView
            android:id="@+id/textView3"
            app:layout_constraintTop_toBottomOf="@+id/txtName"
            tools:text="@{String.valueOf(person1.id) + ' ' +person1.name}" />

    </androidx.constraintlayout.widget.ConstraintLayout>
</layout>
```