CSCI3310 Mobile Computing & Application Development

Lab 6 – Fragment : Favourite Meal

A Fragment is a self-contained component with its own user interface (UI) and lifecycle that can be reused in different parts of an app's UI. (A Fragment can also be used without a UI, in order to retain values across configuration changes, but this lesson does not cover that usage.)

A Fragment can be a *static* part of the UI of an Activity, which means that the Fragment remains on the screen during the entire lifecycle of the Activity. However, the UI of an Activity may be more effective if it adds or removes the Fragment *dynamically* while the Activity is running.

One example of a dynamic Fragment is the DatePicker object, which is an instance of DialogFragment, a subclass of Fragment. The date picker displays a dialog window floating on top of its Activity window when a user taps a button or an action occurs. The user can click **OK** or **Cancel** to close the Fragment.

This lab, which is adopted from Codelabs Advanced Android 01.1, introduces the Fragment class and shows you how to include a Fragment as a static part of a UI, as well as how to use Fragment transactions to add, replace, or remove a Fragment dynamically.

Objective

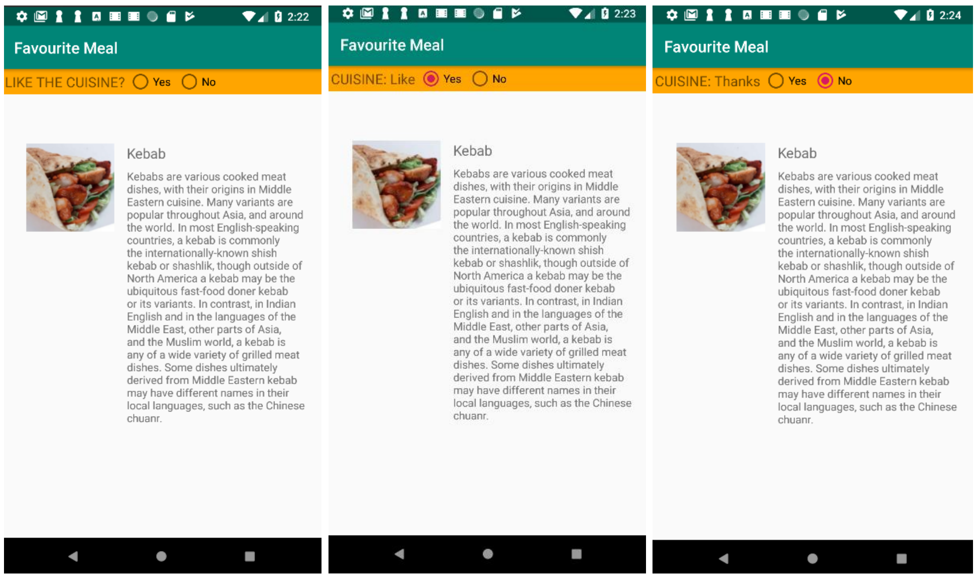
- Create a Fragment with an interactive UI.
- Add a Fragment to the layout of an Activity.
- Add, replace, and remove a Fragment while an Activity is running.

Todo

- Create a Fragment to use as a UI element that gives users a "yes" or "no" choice.
- Add interactive elements to the Fragment that enable the user to choose "yes" or "no".
- Include the Fragment for the duration of an Activity.
- Use Fragment transactions to add, replace, and remove a Fragment while an Activity is running.

The app displays an image and the title and text of a cuisine description. It also shows a Fragment that enables users to provide feedback for the cuisine. In this case, the feedback is very simple: just "Yes" or "No" to the question "Like the cuisine?" Depending on whether the user gives positive or negative feedback, the app displays an appropriate response.

The Fragment is skeletal, but it demonstrates how to create a Fragment to use in multiple places in your app's UI.



Kebab

Kebabs are various cooked meat dishes, with their origins in Middle Eastern cuisine. Many variants are popular throughout Asia, and around the world. In most English-speaking countries, a kebab is commonly the internationally-known shish kebab or shashlik, though outside of North America a kebab may be the ubiquitous fast-food doner kebab or its variants. In contrast, in Indian English and in the languages of the Middle East, other parts of Asia, and the Muslim world, a kebab is any of a wide variety of grilled meat dishes. Some dishes ultimately derived from Middle Eastern kebab may have different names in their local languages, such as the Chinese chuanr.

Kebab

Kebabs are various cooked meat dishes, with their origins in Middle Eastern cuisine. Many variants are popular throughout Asia, and around the world. In most English-speaking countries, a kebab is commonly the internationally-known shish kebab or shashlik, though outside of North America a kebab may be the ubiquitous fast-food doner kebab or its variants. In contrast, in Indian English and in the languages of the Middle East, other parts of Asia, and the Muslim world, a kebab is any of a wide variety of grilled meat dishes. Some dishes ultimately derived from Middle Eastern kebab may have different names in their local languages, such as the Chinese chuanr.

Kebab

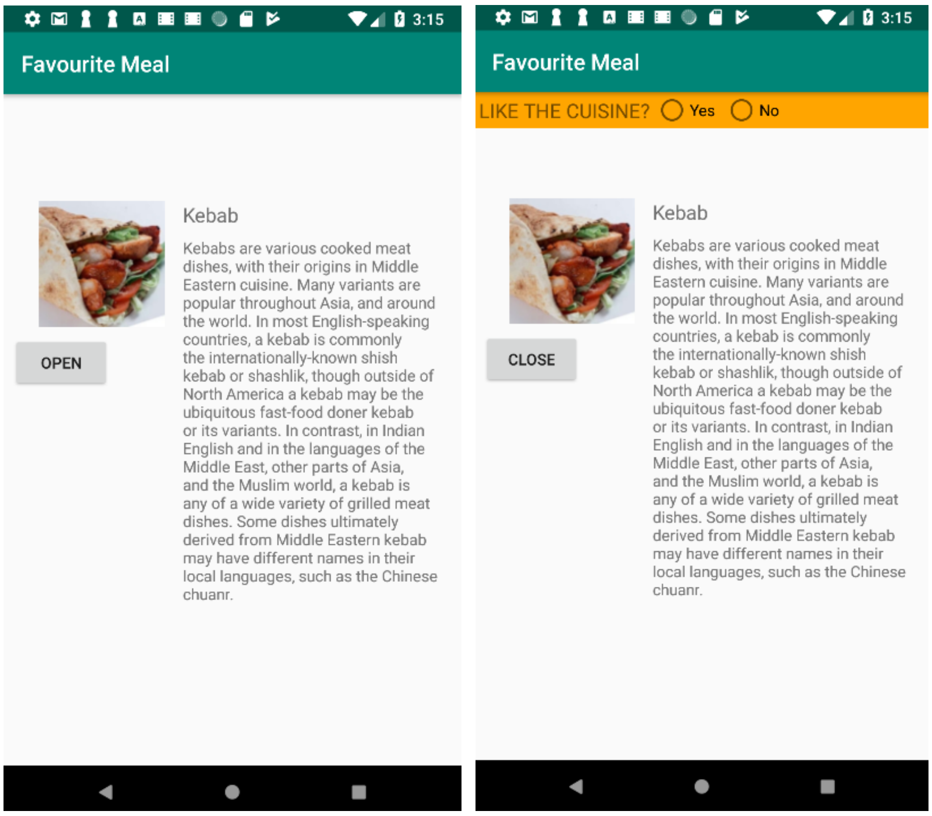
Kebabs are various cooked meat dishes, with their origins in Middle Eastern cuisine. Many variants are popular throughout Asia, and around the world. In most English-speaking countries, a kebab is commonly the internationally-known shish kebab or shashlik, though outside of North America a kebab may be the ubiquitous fast-food doner kebab or its variants. In contrast, in Indian English and in the languages of the Middle East, other parts of Asia, and the Muslim world, a kebab is any of a wide variety of grilled meat dishes. Some dishes ultimately derived from Middle Eastern kebab may have different names in their local languages, such as the Chinese chuanr.

In the first task, you add the Fragment *statically* to the Activity layout so that it is displayed for the entire duration of the Activity lifecycle. The user can interact with the radio buttons in the Fragment to choose either "Yes" or "No," as shown in the figure above.

- If the user chooses "Yes," the text in the Fragment changes to "Cuisine: Like."
- If the user chooses "No," the text in the Fragment changes to "Cuisine: Thanks."

In the second task, you add the Fragment *dynamically*— your code adds, replaces, and removes the Fragment while the Activity is running. You will change the Activity code and layout to do this.

As shown below, the user can tap the **Open** button to show the Fragment at the top of the screen. The user can then interact with the UI elements in the Fragment. The user taps **Close** to close the Fragment.



Kebab

Kebabs are various cooked meat dishes, with their origins in Middle Eastern cuisine. Many variants are popular throughout Asia, and around the world. In most English-speaking countries, a kebab is commonly the internationally-known shish kebab or shashlik, though outside of North America a kebab may be the ubiquitous fast-food doner kebab or its variants. In contrast, in Indian English and in the languages of the Middle East, other parts of Asia, and the Muslim world, a kebab is any of a wide variety of grilled meat dishes. Some dishes ultimately derived from Middle Eastern kebab may have different names in their local languages, such as the Chinese chuanr.

Kebab

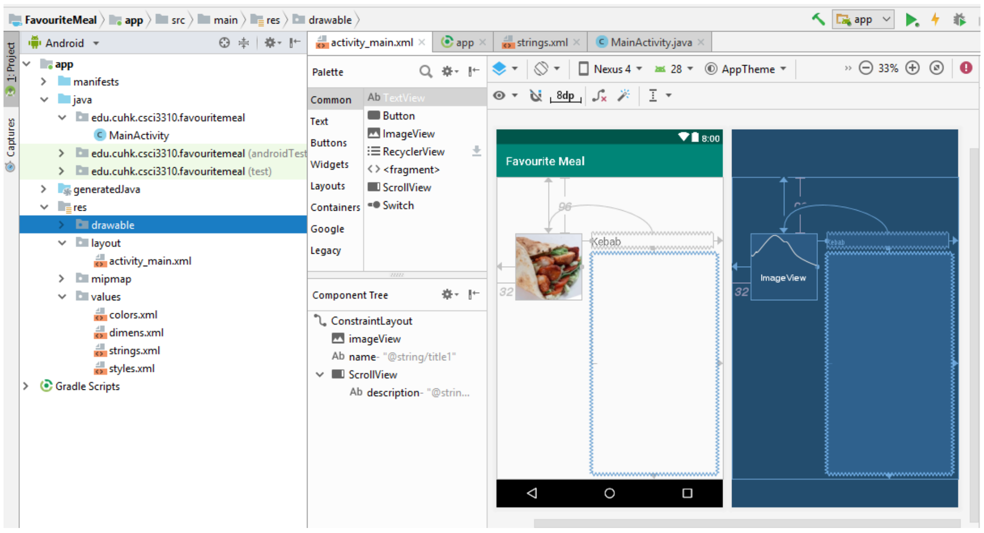
Kebabs are various cooked meat dishes, with their origins in Middle Eastern cuisine. Many variants are popular throughout Asia, and around the world. In most English-speaking countries, a kebab is commonly the internationally-known shish kebab or shashlik, though outside of North America a kebab may be the ubiquitous fast-food doner kebab or its variants. In contrast, in Indian English and in the languages of the Middle East, other parts of Asia, and the Muslim world, a kebab is any of a wide variety of grilled meat dishes. Some dishes ultimately derived from Middle Eastern kebab may have different names in their local languages, such as the Chinese chuanr.

1. Include a Fragment for the entire Activity lifecycle

In this task, you modify a starter app to add a Fragment *statically* as a part of the layout of the Activity, which means that the Fragment is shown during the entire lifecycle of the Activity. This is a useful technique for consolidating a set of UI elements (such as radio buttons and text) and user interaction behavior that you can reuse in layouts for other activities.

1.1 Open the starter app project

Download the FavouriteMeal Android Studio project from Blackboard. You may optionally refactor and rename the project to another name. (For help with copying projects and refactoring and renaming, see How to copy and rename a project.) Explore the app using Android Studio.

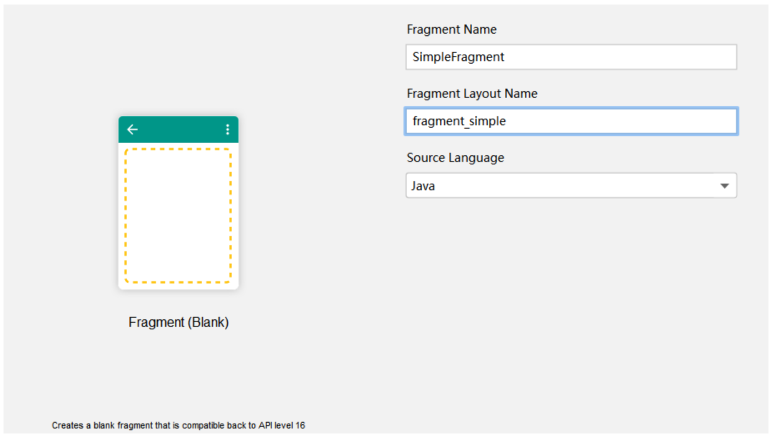


In the above figure:

1. The **Project: Android** pane shows the contents of the project.
2. The **Component Tree** pane for the activity_main.xml layout shows the android:id value of each UI element.
3. The layout includes image and text elements arranged to provide a space at the top for the app bar and a Fragment.

1.2 Add a Fragment

1. In **Project: Android** view, expand **app > java** and select **edu.cuhk.csci3310.favouritemeal**.
2. Choose **File > New > Fragment > Fragment (Blank)**.
3. In the **Configure Component** dialog, name the Fragment **SimpleFragment**. The Fragment layout will be filled in as fragment_simple.
4. Click **Finish** to create the Fragment.


Fragment (Blank)
Creates a blank fragment that is compatible back to API level 16

Fragment Name: SimpleFragment
Fragment Layout Name: fragment_simple
Source Language: Java

5. Open SimpleFragment, and inspect the code:

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

All subclasses of Fragment must include a public no-argument constructor as shown. The Android framework often re-instantiates a Fragment object when needed, in particular during state restore. The framework needs to be able to find this constructor so it can instantiate the Fragment.

The Fragment class uses callback methods that are similar to Activity callback methods. For example, onCreateView() provides a LayoutInflater to inflate the Fragment UI from the layout resource fragment_simple.

1.3 Edit the Fragment's layout

1. Open fragment_simple.xml. In the layout editor pane, click **Text** to view the XML.
2. Change the attributes for the "Hello blank fragment" TextView:

TextView attribute	Value
android:id	@+id/fragment_header
android:layout_width	wrap_content
android:layout_height	wrap_content
android:textAppearance	@style/Base.TextAppearance.AppCompat.Medium
android:padding	"4dp"
android:text	"@string/question_cuisine"

The @string/question_cuisine resource is defined in the strings.xml file in the starter app as "LIKE THE CUISINE?".

3. Change the FrameLayout root element to LinearLayout, and change the following attributes for the LinearLayout:

LinearLayout attribute	Value
android:layout_height	wrap_content
android:layout_width	wrap_content
android:background	@color/my_fragment_color
android:orientation	horizontal

The my_fragment_color value for the android:background attribute is already defined in the starter app in colors.xml.

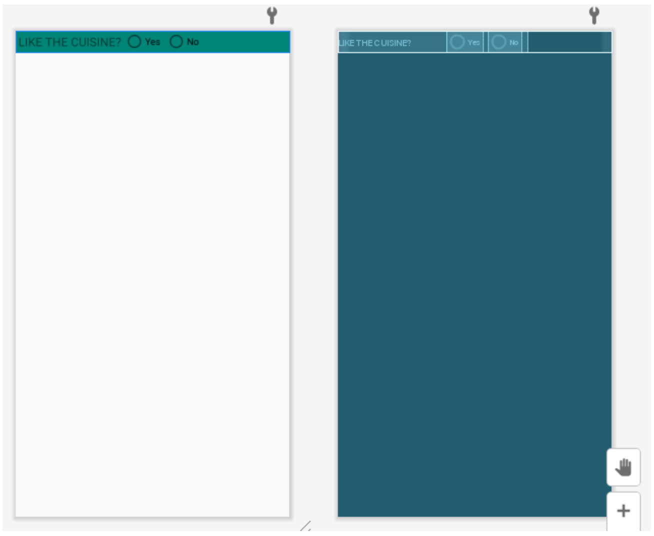
4. Within the LinearLayout, add the following RadioGroup element after the TextView element:

```
<RadioGroup
    android:id="@+id/radio_group"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:orientation="horizontal">
        <RadioButton
             android:id="@+id/radio_button_yes"
             android:layout_width="wrap_content"
             android:layout_height="wrap_content"
             android:layout_marginRight="8dp"
             android:text="@string/yes" />
        <RadioButton
            android:id="@+id/radio_button_no"
             android:layout_width="wrap_content"
             android:layout_height="wrap_content"
             android:layout_marginRight="8dp"
             android:text="@string/no" />
</RadioGroup>
```

The @string/yes and @string/no resources are defined in the strings.xml file in the starter app as "Yes" and "No". In addition, the following string resources are also defined in the strings.xml file:

```
<string name="yes_message">CUISINE: Like</string>
<string name="no_message">CUISINE: Thanks</string>
```

The layout preview for fragment_simple.xml should look like the following:



1.4 Add a listener for the radio buttons

Add the RadioGroup.OnCheckedChangeListener interface to define a callback to be invoked when a radio button is checked or unchecked:

1. Open SimpleFragment again.
2. Change the code in onCreateView() to the code below. This code sets up the View and the RadioGroup and returns the View (rootView). The View and RadioGroup are declared as final, which means they won't change. This is because you will need to access them from within an anonymous inner class (the OnCheckedChangeListener for the RadioGroup):

```
// Inflate the layout for this fragment.
final View rootView =
          inflater.inflate(R.layout.fragment_simple, container, false);
final RadioGroup radioGroup = rootView.findViewById(R.id.radio_group);
// TODO: Set the radioGroup onCheckedChanged listener.
// Return the View for the fragment's UI.
return rootView;
```

3. Add the following constants to the top of SimpleFragment to represent the two states of the radio button choice: 0 = yes and 1 = no:

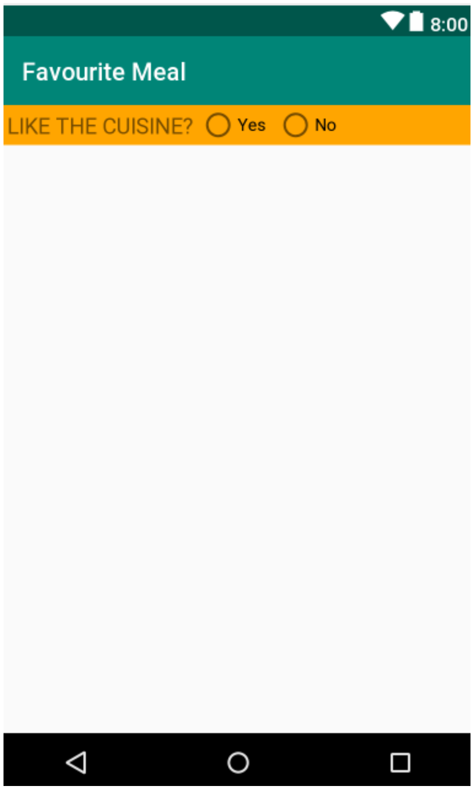
```
private static final int YES = 0;
private static final int NO = 1;
```

4. Replace the TODO comment inside the onCreateView() method with the following code to set the radio group listener and change the textView (the fragment_header in the layout) depending on the radio button choice:

```
radioGroup.setOnCheckedChangeListener(new
                             RadioGroup.OnCheckedChangeListener() {
    @Override
     public void onCheckedChanged(RadioGroup group, int checkedId) {
         View radioButton = radioGroup.findViewById(checkedId);
         int index = radioGroup.indexOfChild(radioButton);
         TextView textView =
                         rootView.findViewById(R.id.fragment_header);
```

```
            switch (index) {
                case YES: // User chose "Yes."
                    textView.setText(R.string.yes_message);
                   break;
                case NO: // User chose "No."
                    textView.setText(R.string.no_message);
                   break;
                default: // No choice made.
                   // Do nothing.
                   break;
            }
        }
    });
```

The onCheckedChanged() method must be implemented for RadioGroup.OnCheckedChangeListener; in this example, the index for the two radio buttons is 0 ("Yes") or 1 ("No"), which displays either the yes_message text or the no_message text.



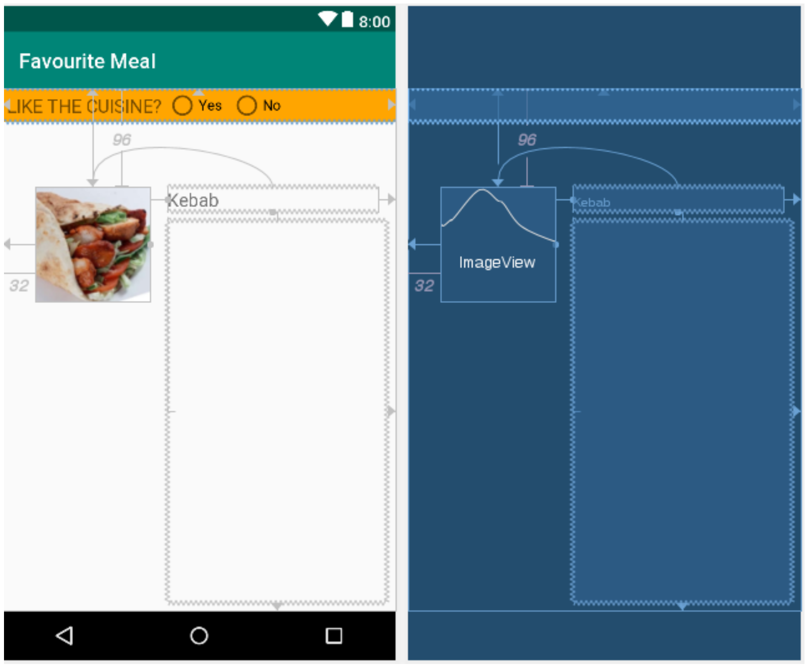
1.5 Add the Fragment statically to the Activity

1. Open activity_main.xml.
2. Add the following <fragment> element below the ScrollView within the ConstraintLayout root view. It refers to the SimpleFragment you just created:

```
<fragment
android:id="@+id/fragment"
android:name=" edu.cuhk.csci3310.favouritemeal.SimpleFragment"
```

```
android:layout_width="0dp"
android:layout_height="wrap_content"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"
tools:layout="@layout/fragment_simple" />
```

The figure below shows the layout preview, with SimpleFragment statically included in the activity layout:



A render error may appear below the preview, because a <fragment> element can dynamically include different layouts when the app runs, and the layout editor doesn't know what layout to use for a preview. To see the Fragment in the preview, click the **@layout/fragment_simple** link in the error message, and choose **SimpleFragment**.

3. Run the app.

The Fragment is now included in the MainActivity layout, as shown in the figures below.

1. Tap a radio button. The "LIKE THE CUISINE?" text is replaced by either the yes_message text ("CUISINE: Like") or the no_message text ("CUISINE: Thanks"), depending on which radio button is tapped.
2. After tapping a radio button, change the orientation of your device or emulator from portrait to landscape. Note that the "Yes" or "No" choice is still selected.

Switching the device orientation after choosing "No" demonstrates that a Fragment can retain an instance of its data after a configuration change (such as changing the orientation). This feature makes a Fragment useful as a UI component, as compared to using separate Views. While an Activity is destroyed and recreated when a device's configuration changes, a Fragment is not destroyed.

2. Add a Fragment to an Activity dynamically

In this task, you learn how to add the same Fragment to an Activity *dynamically*. The Fragment is added only if the user performs an interaction in the Activity—in this case, tapping a button.

You will change the FavouriteMeal app to manage the Fragment using FragmentManager and FragmentTransaction statements that can add, remove, and replace a Fragment.

2.1 Add a ViewGroup for the Fragment

At any time while your Activity is running, your code can add a Fragment to the Activity. All your Activity code needs is a ViewGroup in the layout as a placeholder for the Fragment, such as a FrameLayout. Change the <fragment> element to <FrameLayout> in the main layout. Follow these steps:

1. (Optional) Copy the FavouriteMeal project, and open the copy in Android Studio. Refactor and rename the project to FavouriteMeal2. (For help with copying projects and refactoring and renaming, see How to copy and rename a project.)

If a warning appears to remove the FavouriteMeal modules, click to remove them.

2. Open the activity_main.xml layout, and switch the layout editor to **Text** (XML) view.
3. Change the <fragment> element to <FrameLayout>, remove the android:name, and change the android:id to fragment_container as shown below:

```
<FrameLayout
     android:id="@+id/fragment_container"
```

2.2 Add a button to open and close the Fragment

Users need a way to open and close the Fragment from the Activity. To keep this example simple, add a Button to open the Fragment, and if the Fragment is open, the same Button can close the Fragment.

1. Open activity_main.xml, click the **Design** tab if it is not already selected, and add a Button under the imageView element.
2. Constrain the Button to the bottom of imageView and to the left side of the parent.
3. Open the **Attributes** pane, and add the ID open_button and the text "@string/open". The @string/open and @string/close resources are defined in the strings.xml file in the starter app as "OPEN" and "CLOSE".

Note that since you have changed <fragment> to <FrameLayout>, the Fragment (now called fragment_container) no longer appears in the design preview—but don't worry, it's still there!

4. Open MainActivity, and declare and initialize the Button. Also, add a private boolean to determine whether the Fragment is displayed:

```
private Button mButton;
private boolean isFragmentDisplayed = false;
```

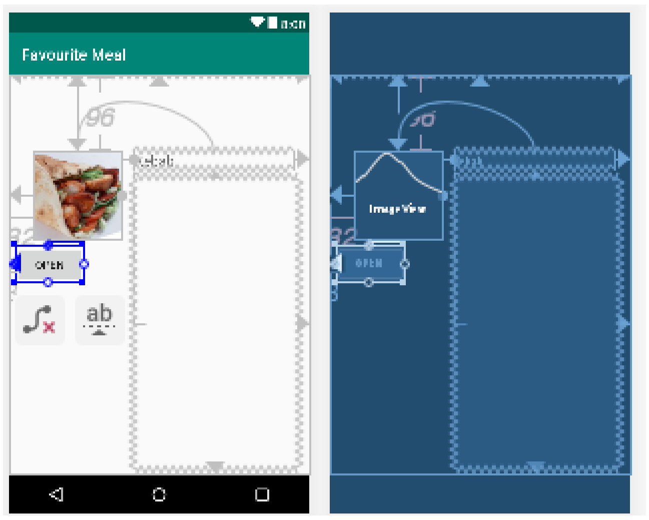
5. Add the following to the onCreate() method to initialize the mButton:

```
mButton = findViewById(R.id.open_button);
```

2.3 Use Fragment transactions

To manage a Fragment in your Activity, use FragmentManager. To perform a Fragment transaction in your Activity—such as adding, removing, or replacing a Fragment—use methods in FragmentTransaction.

The best practice for instantiating the Fragment in the Activity is to provide a newinstance() factory method in the Fragment. Follow these steps to add the newinstance() method to SimpleFragment and instantiate the Fragment in MainActivity. You will also add the displayFragment() and closeFragment() methods, and use Fragment transactions:



1. Open SimpleFragment, and add the following method for instantiating and returning the Fragment to the Activity:

```
public static SimpleFragment newInstance() {
        return new SimpleFragment();
}
```

Open MainActivity, and create the following displayFragment() method to instantiate and open SimpleFragment. It starts by creating an instance of simpleFragment by calling the newInstance() method in SimpleFragment:

```
public void displayFragment() {
    SimpleFragment simpleFragment = SimpleFragment.newInstance();
    // TODO: Get the FragmentManager and start a transaction.
    // TODO: Add the SimpleFragment.
}
```

2. Replace the TODO: Get the FragmentManager... comment in the above code with the following:

```
// Get the FragmentManager and start a transaction.
FragmentManager fragmentManager = getSupportFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
```

To start a transaction, get an instance of FragmentManager using getSupportFragmentManager(), and then get an instance of FragmentTransaction that uses beginTransaction(). Fragment operations are wrapped into a *transaction* (similar to a bank transaction) so that all the operations finish before the transaction is committed for the final result.

Use getSupportFragmentManager() (instead of getFragmentManager()) to instantiate the Fragment class so your app remains compatible with devices running system versions as low as Android 1.6. (The getSupportFragmentManager() method uses the Support Library.)

3. Replace the TODO: Add the SimpleFragment comment in the above code with the following:

```
// Add the SimpleFragment.
fragmentTransaction.add(R.id.fragment_container,
simpleFragment).addToBackStack(null).commit();
// Update the Button text.
mButton.setText(R.string.close);
// Set boolean flag to indicate fragment is open.
isFragmentDisplayed = true;
```

This code adds a new Fragment using the add() transaction method. The first argument passed to add() is the layout resource
(fragment_container) for the ViewGroup in which the Fragment should be placed. The second parameter is the Fragment (simpleFragment) to add.

In addition to the add() transaction, the code calls addToBackStack(null) in order to add the transaction to a back stack of Fragment transactions. This back stack is managed by the Activity. It allows the user to return to the previous Fragment state by pressing the Back button.

The code then calls commit() for the transaction to take effect.

The code also changes the text of the Button to "CLOSE" and sets the Boolean isFragmentDisplayed to true so that you can track the state of the Fragment.

5. To close the Fragment, add the following closeFragment() method to MainActivity:

```
public void closeFragment() {
    // Get the FragmentManager.
    FragmentManager fragmentManager = getSupportFragmentManager();
     // Check to see if the fragment is already showing.
    SimpleFragment simpleFragment = (SimpleFragment) fragmentManager
                .findFragmentById(R.id.fragment_container);
     if (simpleFragment != null) {
        // Create and commit the transaction to remove the fragment.
         FragmentTransaction fragmentTransaction =
                 fragmentManager.beginTransaction();
         fragmentTransaction.remove(simpleFragment).commit();
    }
    // Update the Button text.
    mButton.setText(R.string.open);
    // Set boolean flag to indicate fragment is closed.
     isFragmentDisplayed = false;
}
```

As with displayFragment(), the closeFragment() code snippet gets an instance of FragmentManager using getSupportFragmentManager(), uses beginTransaction() to start a series of transactions, and acquires a reference to the Fragment using the layout resource (fragment_container). It then uses the remove() transaction to remove the Fragment.

However, before creating this transaction, the code checks to see if the Fragment is displayed (not null). If the Fragment is not displayed, there's nothing to remove.

The code also changes the text of the Button to "OPEN" and sets the Boolean isFragmentDisplayed to false so that you can track the state of the Fragment.

2.4 Set the Button onClickListener

To take action when the user clicks the Button, implement an OnClickListener for the button in the onCreate() method of MainActivity in order to open and close the fragment based on the boolean value of isFragmentDisplayed. The onClick() method will call the displayFragment() method to open the Fragment if the Fragment is not already open; otherwise, it calls the closeFragment().

You will also add code to save the value of isFragmentDisplayed and use it if the configuration changes, such as if the user switches from portrait or landscape orientation.

1. Open MainActivity, and add the following to the onCreate() method to set the click listener for the Button:

```
// Set the click listener for the button.
mButton.setOnClickListener(new View.OnClickListener() {
    @Override
     public void onClick(View view) {
         if (!isFragmentDisplayed) {
              displayFragment();
        } else {
              closeFragment();
        }
    }
});
```

2. To save the boolean value representing the Fragment display state, define a key for the Fragment state to use in the savedInstanceState Bundle. Add this member variable to MainActivity:

```
static final String STATE_FRAGMENT = "state_of_fragment";
```

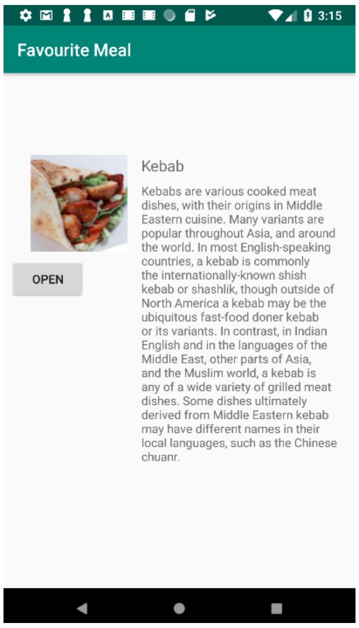
3. Add the following method to MainActivity to save the state of the Fragment if the configuration changes:

```
public void onSaveInstanceState(Bundle savedInstanceState) {
    // Save the state of the fragment (true=open, false=closed).
    savedInstanceState.putBoolean(STATE_FRAGMENT, isFragmentDisplayed);
     super.onSaveInstanceState(savedInstanceState);
}
```

4. Go back to the onCreate() method and add the following code. It checks to see if the instance state of the Activity was saved for some reason, such as a configuration change (the user switching from vertical to horizontal). If the saved instance was not saved, it would be null. If the saved instance is *not* null, the code retrieves the Fragment state from the saved instance, and sets the Button text:

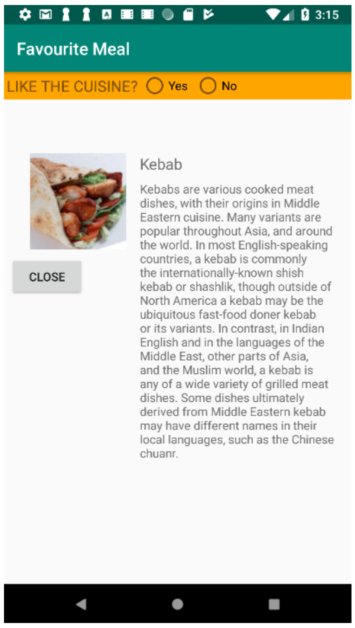
```
if (savedInstanceState != null) {
     isFragmentDisplayed =
                      savedInstanceState.getBoolean(STATE_FRAGMENT);
     if (isFragmentDisplayed) {
        // If the fragment is displayed, change button to "close".
        mButton.setText(R.string.close);
    }
}
```

5. Run the app. Tapping **Open** adds the Fragment and shows the **Close** text in the button. Tapping **Close** removes the Fragment and shows the **Open** text in the button. You can switch your device or emulator from vertical to a horizontal orientation to see that the buttons and Fragment work.



Kebab

Kebabs are various cooked meat dishes, with their origins in Middle Eastern cuisine. Many variants are popular throughout Asia, and around the world. In most English-speaking countries, a kebab is commonly the internationally-known shish kebab or shashlik, though outside of North America a kebab may be the ubiquitous fast-food doner kebab or its variants. In contrast, in Indian English and in the languages of the Middle East, other parts of Asia, and the Muslim world, a kebab is any of a wide variety of grilled meat dishes. Some dishes ultimately derived from Middle Eastern kebab may have different names in their local languages, such as the Chinese chuanr.



Kebab

Kebabs are various cooked meat dishes, with their origins in Middle Eastern cuisine. Many variants are popular throughout Asia, and around the world. In most English-speaking countries, a kebab is commonly the internationally-known shish kebab or shashlik, though outside of North America a kebab may be the ubiquitous fast-food doner kebab or its variants. In contrast, in Indian English and in the languages of the Middle East, other parts of Asia, and the Muslim world, a kebab is any of a wide variety of grilled meat dishes. Some dishes ultimately derived from Middle Eastern kebab may have different names in their local languages, such as the Chinese chuanr.

References

- Fragment
- Fragments
- FragmentManager
- FragmentTransaction
- Creating a Fragment
- Building a Flexible UI
- Building a Dynamic UI with Fragments