Mobile App Development

Exercises: Todo Fragments App

Whilst the Android XML view definition and view objects provide for seperation of the V in MVC, the view remains tightly coupled with the Activity class. This is problematic in principle. The V in MVC needs to be more decoupled and further abstracted so that it could be composed and recomposed as necessary at run time.

It is true that an Activities view may change at run time but the code for the change is inside the activity, hence the tight coupling. To decouple is to abstract the view control code out of the Activity and delegate it to another class. This is achieved in Android with *fragments*.

A `fragment` is a controller object that an activity can delegate view management tasks to. The `Activity's` own view can have a placeholder(s) defined to insert any framgent(s) view. This decoupling allows for views to be dynamically recomposed as the result of device or user requirements and events.

To clarify best practice in use of Android, the following model is implemented in several stages and at each stage we will consider any refactoring that may improve the code. Here is the Object diagram we will use as the case study.

`File > New Project`
Application name: `TodoFragments`
Company domain: `example.com`
`Next`
`Next`
`Empty Activity`
`Next`
leave the activity name and layout name to the default `MainActivity` *and* `activity_main`
`Finish`

`File > Project Structure`
`> Dependencies`

check the following google compatibility libraries are included and add them if necessary
`com.android.support.appcompat-v7:25.0.1`

`Build > Clean Project`

`Run`

The TextView with a default "Hello world" is displayed.

To implement fragments we need a generic container to host the fragments. We shall use the FrameLayout as it is completely generic and suitable for hosting fragments.

Edit `java/res/layout/activity_main.xml` and replace the view with the following:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
```

```
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:id="@+id/fragment_container"
      android:layout_height="match_parent"
      android:layout_width="match_parent"
      />
```

Check the view

`Build > Rebuild`

`Run`

You should see a blank empty `FrameLayout` container view object as intended. We will use this same layout to host other fragments. Currently, we have defined a single fragment, however we will develop this further and have the activity's layout with multiple container views as well as widgets of its own.

Next is the design of a UI fragment for a Todo item to include in the generic FrameLayout.

Create a Todo UI Fragment

The steps in creating a UI fragment are the same as creating an activity, nemely:

1. Crete a layout file to contain the UI widgets
2. Create a class to use the layout
3. Overide methods to inflate the view objects

1. Crete a layout file to contain the UI widgets

The Todo item could include the following attributes: `ID, Title, Detail, Date, and IsComplete` *(we shall return to theses attributes for defining the Model.) Lets define the attributes as a resource.*

Edit `java/res/values/strings.xml` and replace the resource with the following:

```
<resources>
  <string name="app_name">TodoFragments</string>
  <string name="todo_title">todo title</string>
  <string name="todo_title_hint">todo title hint</string>
  <string name="todo_title_label">One line description of the todo</string>
  <string name="todo_detail_label">detail of what to do</string>
  <string name="todo_complete_label">is it complete?</string>
  <string name="todo_date">Todo date</string>
</resources>
```

Next the Todo fragment view to include the identified attributes above.

`Right-mouse click` on `app/res/layout` and select `New > Layout resource file`
Name the file `fragment_todo.xml` and leave the Root element as the default `LinearLayout`
`OK`

Edit the newly created `app/res/layout/fragment_todo.xml`
(either add the widgets or replace the entire file with the following view definition.)

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="vertical"
              android:layout_width="match_parent"
```

```
                android:layout_height="match_parent"
                android:layout_margin="16dp"
                android:weightSum="1">

    <TextView
        android:id="@+id/todo_detail_label"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/todo_title_label"
        style="@string/todo_title_label"/>

    <EditText
        android:id="@+id/todo_title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/todo_title_hint"/>

    <TextView
        android:id="@+id/todo_detail"
        android:layout_width="match_parent"
        android:layout_height="@android:dimen/notification_large_icon_height"
        android:text="@string/todo_detail_label"/>

    <Button
        android:id="@+id/todo_date"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        />

    <CheckBox
        android:id="@+id/todo_complete"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/todo_complete_label"/>

</LinearLayout>
```

Check the *Design* view to see a preview of the todo fragment.

2. Create a class to use the layout

This a controller class for manupilating fragments in a layout and it needs to extend and be a subclass of the `Fragment` class.

`Right-mouse click` on `app/java/com.example.todofraga > New > Java Class`
Name: `TodoFragment`
Superclass: `Fragment`
leave other options to default

Edit the TodoFragment.java file and insert the following class definition under the package name.

```
import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.text.Editable;
import android.text.TextWatcher;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.CompoundButton.OnCheckedChangeListener;
import android.widget.EditText;

public class TodoFragment extends Fragment {
    private Todo mTodo;
    private EditText mEditTextTitle;
    private Button mButtonDate;
    private CheckBox mCheckBoxIsComplete;

    @Override
    public void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        mTodo = new Todo();
        // TODO: refactor
        mTodo.setTitle("Test title");
        mTodo.setIsComplete(true);
    }

    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater,
                             @Nullable ViewGroup container,
                             @Nullable Bundle savedInstanceState) {

        View view = inflater.inflate(R.layout.fragment_todo, container, false);

        mEditTextTitle = (EditText) view.findViewById(R.id.todo_title);
        mEditTextTitle.setText(mTodo.getTitle());
        mEditTextTitle.addTextChangedListener(new TextWatcher() {
            @Override
            public void beforeTextChanged(CharSequence s, int start, int count, int after) {
                // This line is intentionally left blank
            }

            @Override
            public void onTextChanged(CharSequence s, int start, int before, int count) {
                mTodo.setTitle(s.toString());
            }

            @Override
            public void afterTextChanged(Editable s) {
                // This line is intentionally left blank
            }
        });

        mButtonDate = (Button) view.findViewById(R.id.todo_date);
        mButtonDate.setText(mTodo.getDate().toString());
        mButtonDate.setEnabled(false);

        mCheckBoxIsComplete = (CheckBox) view.findViewById(R.id.todo_complete);
        mCheckBoxIsComplete.setOnCheckedChangeListener(new OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
                mTodo.setIsComplete(isChecked);
            }
        });

        return view;

    }
}
```

3. Overide methods to inflate the view objects

`Right-mouse click` in the disply window for the `TodoFragment` class and select `Folding > Collapse All`

Note the similarity and the differences to the Activity class:

- The Activity class had the onCreate method defined with protected scope whereas here the scope is public, this is so that the method can be called by any activity that is hosting the view.

- As in the Activity class the use of a Bundle to save state data
- View objects are inflated and a view object is returned to the hosting Activity but not from the onCreate method but the onCreateView method.

The Model

Being a controller class, the TodoFragment sits between the model and the view and supports the getter and setter methods for the data in the view. The TodoFragment onCreate method instantiates the Todo model. This is not yet defined and we shall do that next.

`Right-mouse click` on `app/java/com.example.todofraga > New > Java Class`
Name: `Todo`
leave other options to default and click `OK`

The model is currently a Plane Old Java Object(POJO) with the getter and setter methods for the data that represents a Todo. Here is the code:

```
import java.util.Date;
import java.util.UUID;

public class Todo {

    private UUID mId;
    private String mTitle;
    private String mDetail;
    private Date mDate;
    private boolean mIsComplete;

    public Todo() {
        mId = UUID.randomUUID();
        mDate = new Date();
    }

    public void setIsComplete(boolean todoIsComplete) {
        mIsComplete = todoIsComplete;
    }

    public boolean isIsComplete() {

        return mIsComplete;
    }

    public UUID getId() {
        return mId;
    }

    public String getTitle() {
        return mTitle;
    }

    public String getDetail() {
        return mDetail;
    }

    public Date getDate() {
        return mDate;
    }

    public void setId(UUID todoId) {
        mId = todoId;
    }

    public void setTitle(String title) {
        mTitle = title;
    }
    public void setDetail(String detail) {
        mDetail = detail;
    }

    public void setDate(Date todoDate) {
        mDate = todoDate;
    }
}
```

With the Todo model class defined, the reference to the Todo class in the TodoFragment is now resolved and the error message should have now disapeard.

Android treats the Fragment classes as controllers that can only have their views displayed when they are added to an Activity. This is acheived with the Fragment Manager class that keeps track of the Fragment and the Back Stack objects.

Fragment Manager adding UI Fragments to Activity

The Activity class uses a Fragment manager and Fragment manager transactions to keep track of the Back Stack and the Fragments. Here is the code for a Fragment Manager to add the TodoFragment to the TodoActivity

Edit the `MainActivity` and insert the following code under the package name.

```
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v7.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

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

        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragment_container);

        if (fragment == null){
            TodoFragment todoFragment = new TodoFragment();
            fm.beginTransaction()
                    .add(R.id.fragment_container, todoFragment)
                    .commit();
        }

    }
}
```

Note the `.add` method is adding the `todoFragment` to the calling Activity's view. That is, a fragment is inserted into `FrameLayout` view object of the MainActivity.

`Run`
You should see the Fragment displayed.

Note, the todoFragment is only instantiated if its null. A fragment is a subclass of Activity and may (or may not) have its own view, its life cycle however has more states than the parent Activity and it may well be in scope and exist during various state changes of its parent Activity, hence, the check for `null` to see if it already exists.

So far, the Activity and View close coupling has been refactored to a dynamically loaded fragment into the associated Activity's container view. We now have a Fragment manager that introduces new complexity but provides logical clarity and code that is more easily maintained.

Additional task: Reviewing the new code however begs the question of having a fragment manager repeated in each activity. Whenever there is a "concern", in that a method is shared amongst many classes, there is an opportunity for a refactor to simplify the code to a single instance of the method. You may consider improving the design with a *factory pattern* to a single instance of a Fragment Manager to be used accross Activity classes as an additional exercise.

Reflection and QA

What is a Fragment?

Which fragment lifecycle methods should normally be implemented?

Can a fragment display its view independantly of its parent Activity?

What are the three main steps to create a UI Fragment?