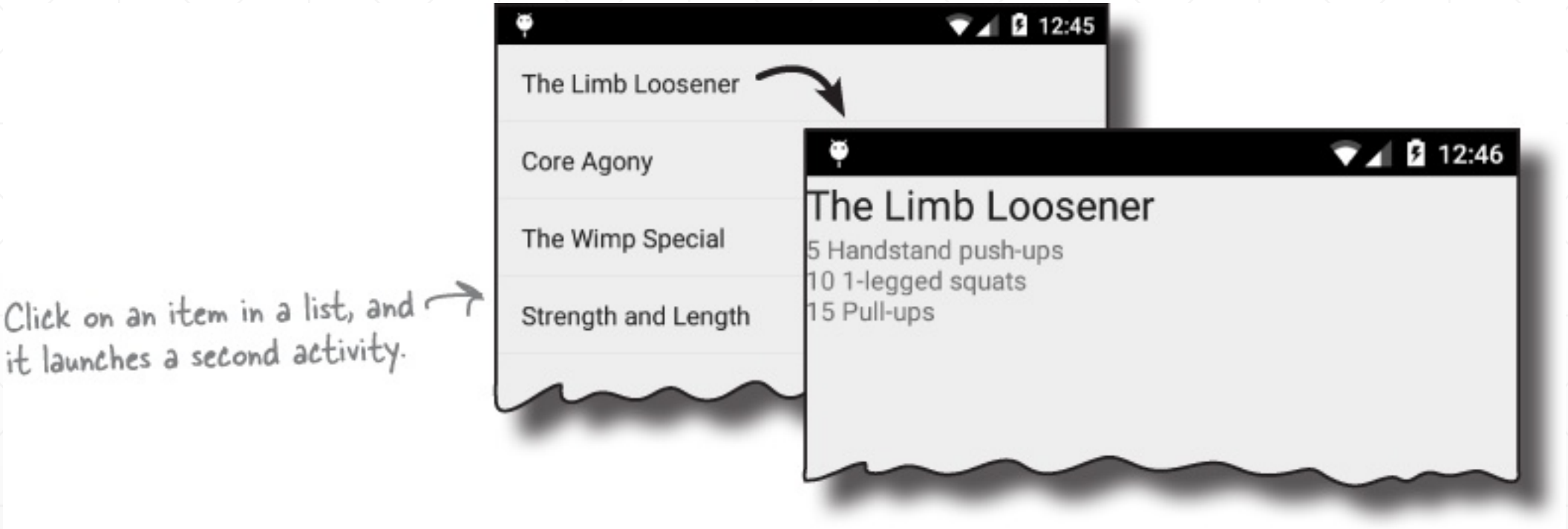
Bullet Points

- A fragment is used to control part of a screen. It can be reused across multiple activities.
- A fragment has an associated layout.
- A fragment is a subclass of the android.app.Fragment class.
- The onCreateView() method gets called each time Android needs the fragment's layout.
- Add a fragment to an activity's layout using the <fragment> element and adding a classattribute.
- The fragment lifecycle methods tie in with the states of the activity that contains the fragment.

Bullet Points

- The Fragment class doesn’t extend the Activity class or implement the Context class.
- Fragments don’t have a findViewById() method. Instead, use the getView() method to get a reference to the root view, then call the view’s findViewById() method.
- A list fragment is a fragment that comes complete with a ListView. You create one by subclassing ListFragment.
- If you need to get a fragment to respond to changes in the user interface, use the<FrameLayout> element.
- Use fragment transactions to make a set of changes to an existing fragment and add to the back stack.
- Make apps look different on different devices by putting separate layouts in device-appropriate folders.

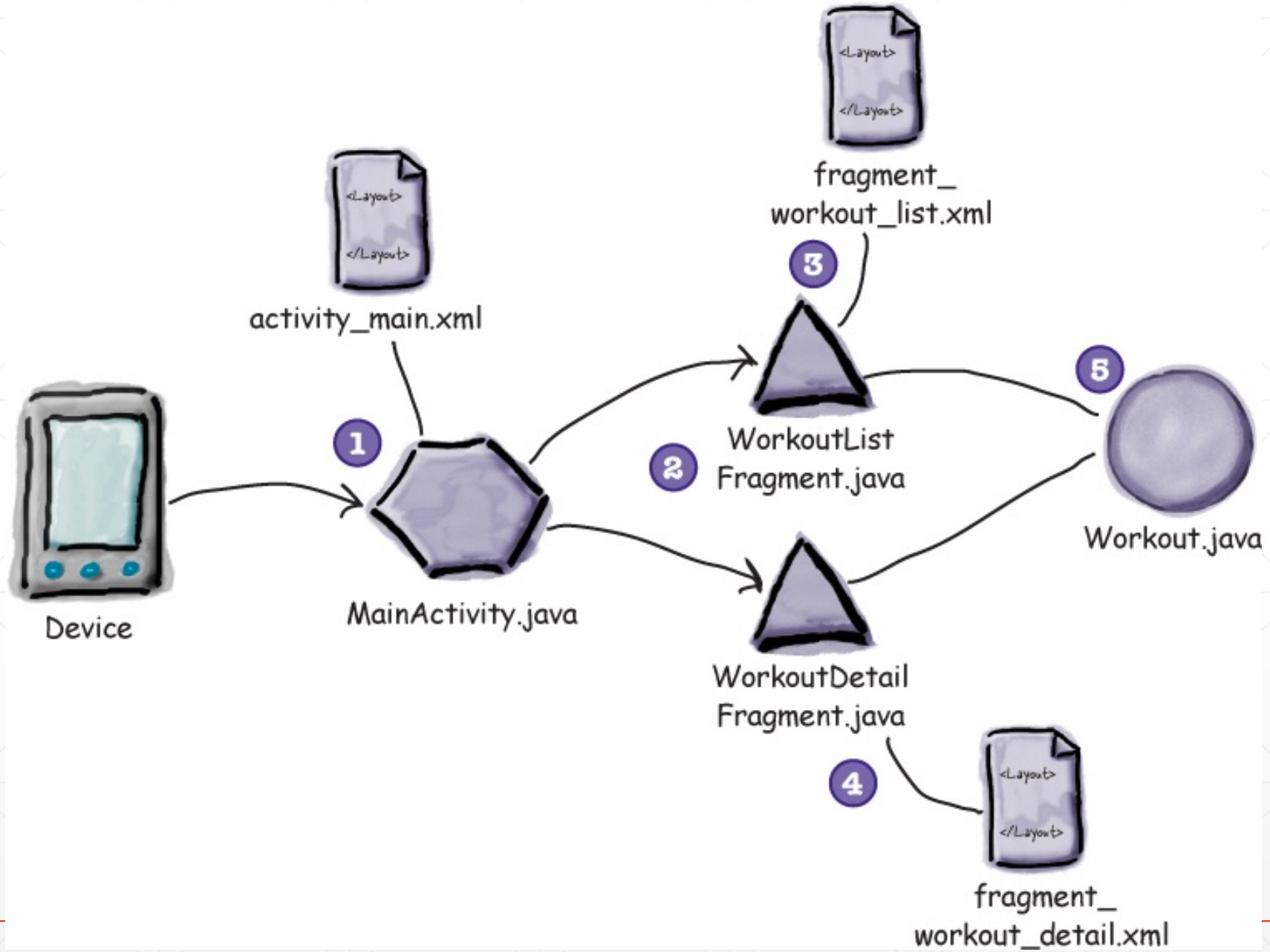
On a phone


12:45
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
12:46
The Limb Loosener
5 Handstand push-ups
10 1-legged squats
15 Pull-ups
Click on an item in a list, and it launches a second activity.

On a tablet


14:48
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
The Limb Loosener
5 Handstand push-ups
10 1-legged squats
15 Pull-ups
There's a lot more space on a tablet, so we can use the space in a different way.


17:19
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
If we use a fragment for the list of workouts, we can reuse it in multiple activities.
We can use a separate fragment for the workout details.
17:19
Core Agony
100 Pull-ups
100 Push-ups
100 Sit-ups
100 Squats
14:51
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
Core Agony
100 Pull-ups
100 Push-ups
100 Sit-ups
100 Squats


<Layout>
</Layout>
fragment_
workout_list.xml
activity_main.xml
3
1
2
5
WorkoutList
Fragment.java
Workout.java
Device
MainActivity.java
WorkoutDetail
Fragment.java
4
fragment_
workout_detail.xml


14:48
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
The Limb Loosener
5 Handstand push-ups
10 1-legged squats
15 Pull-ups
14:49
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
The Wimp Special
5 Pull-ups
10 Push-ups
15 Squats
12:46
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
The Wimp Special
5 Pull-ups
10 Push-ups
15 Squats

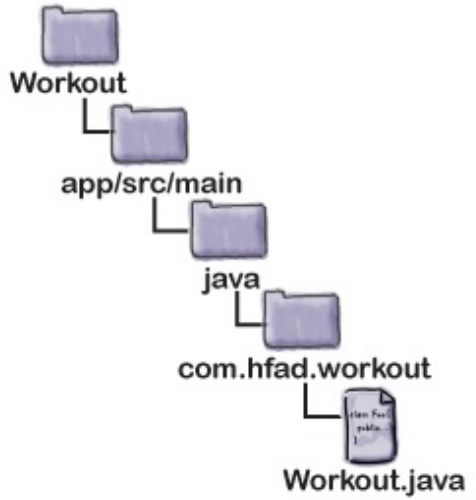
```
package com.hfad.workout;

public class Workout {
    private String name;
    private String description;

    public static final Workout[] workouts = {
            new Workout("The Limb Loosener",
                    "5 Handstand push-ups\n10 1-legged squats\n15 Pull-ups"),
            new Workout("Core Agony",
                    "100 Pull-ups\n100 Push-ups\n100 Sit-ups\n100 Squats"),
            new Workout("The Wimp Special",
                    "5 Pull-ups\n10 Push-ups\n15 Squats"),
            new Workout("Strength and Length",
                    "500 meter run\n21 x 1.5 pood kettleball swing\n21 x pull-ups")
    };

    //Each Workout has a name and description
    private Workout(String name, String description) {
        this.name = name;
        this.description = description;
    }

    public String getDescription() {
        return description;
    }

    public String getName() {
        return name;
    }

    public String toString() {
        return this.name;
    }
}
```

Each Workout has a name and description.

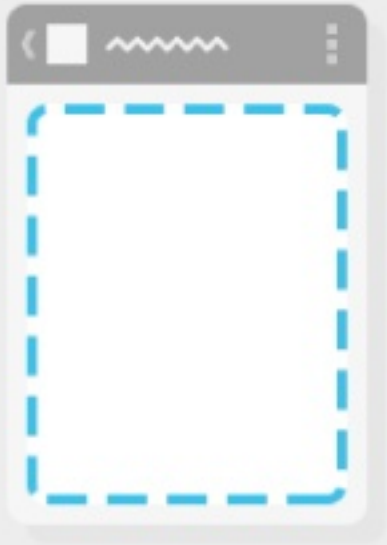
workouts is an array of four Workouts.

These are getters for the private variables.

The String representation of a Workout is its name.



Choose options for your new file



Fragment (Blank)

We're creating a blank fragment.

Creates a blank fragment that is compatible back to API level 4.

This is the name of the fragment.

Fragment Name: WorkoutDetailFragment

☑ Create layout XML?

Fragment Layout Name: fragment_workout_detail

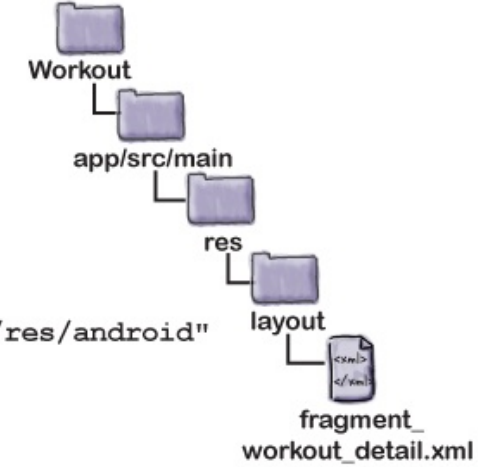
This is the name of the fragment layout.

☐ Include fragment factory methods? ☐ Include interface callbacks?

We don't want Android Studio creating a load of extra code for us, so we're unticking these options.

Generate event callbacks for communication with an Activity or other fragments

Cancel | Previous | Next | Finish

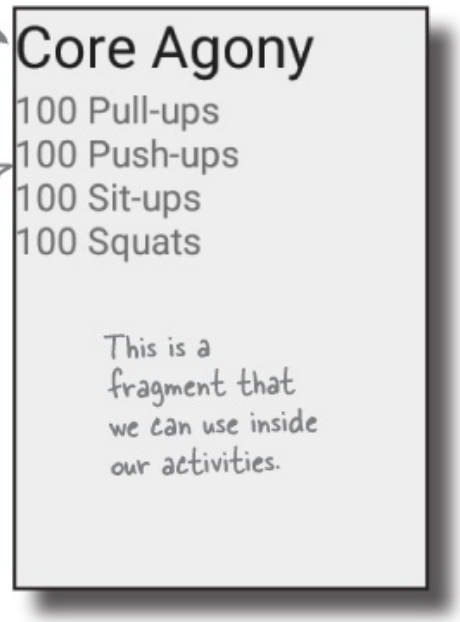



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    android:orientation="vertical">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text=""
        android:id="@+id/textTitle" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=""
        android:id="@+id/textDescription" />

</LinearLayout>
```

We're displaying the workout name and description in two separate TextViews.

This is the workout name.

This is the workout description.

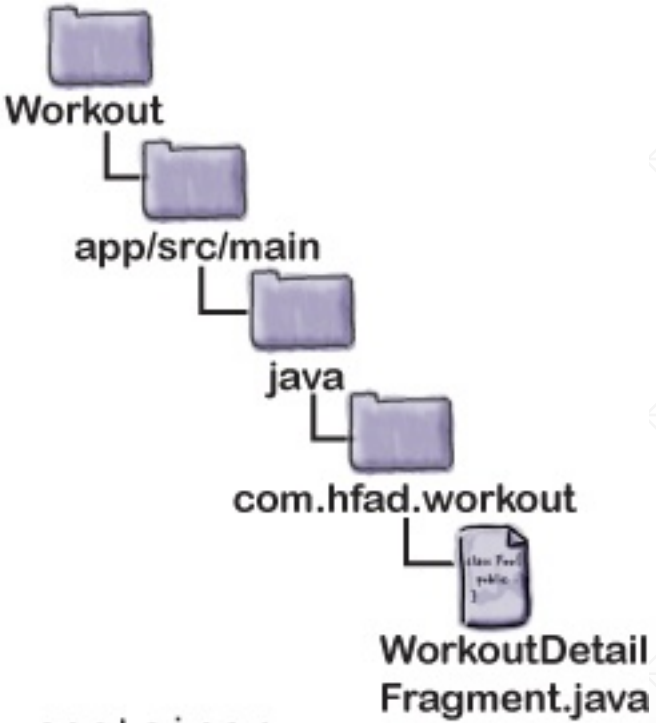


```
package com.hfad.workout;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class WorkoutDetailFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_workout_detail, container, false);
    }
}
```

The class extends the Android Fragment class.

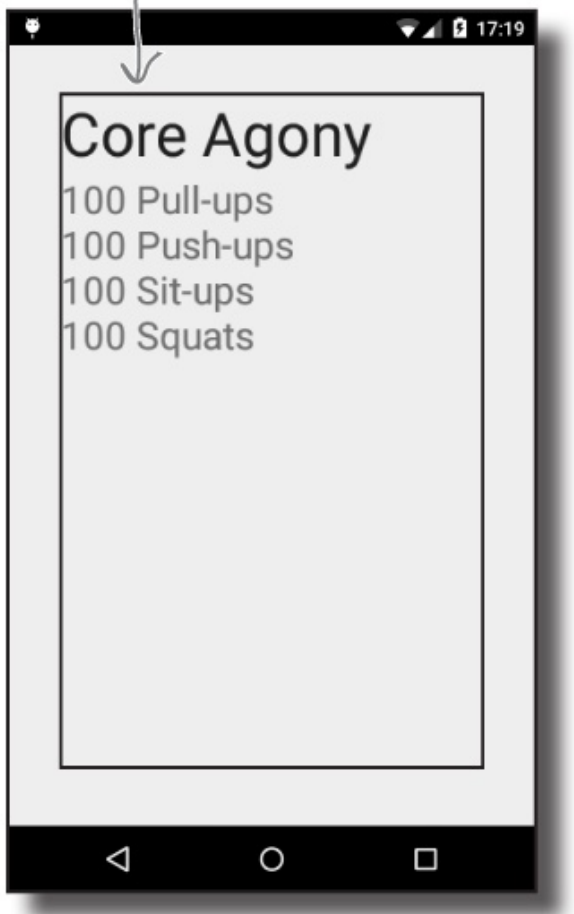
This is the onCreateView() method. It's called when Android needs the fragment's layout.

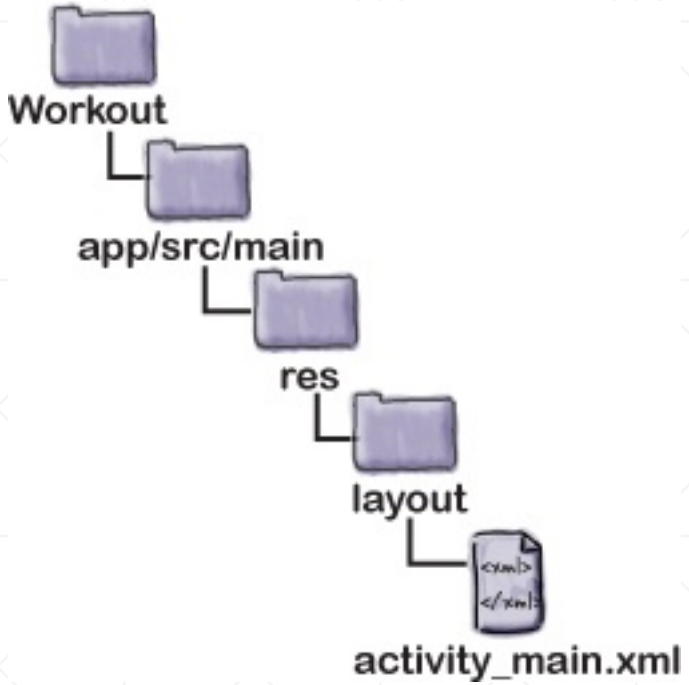
This tells Android which layout the fragment uses (in this case, it's fragment_workout_detail).



```
<?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
        class="com.hfad.workout.WorkoutDetailFragment"
        android:id="@+id/detail_frag"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```

This adds the fragment WorkoutDetailFragment to the activity's layout.

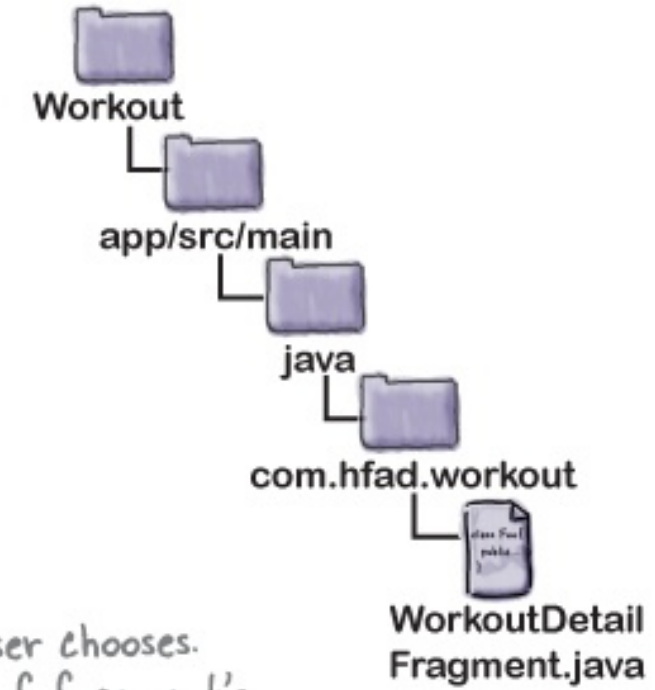




```
package com.hfad.workout;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class WorkoutDetailFragment extends Fragment {
    private long workoutId;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_workout_detail, container, false);
    }

    public void setWorkout(long id) {
        this.workoutId = id;
    }
}
```

This is the ID of the workout the user chooses. Later, we'll use it to set the values of fragment's views with the workout details.

This is a setter method for the workout ID. The activity will use this method to set the value of the workout ID.

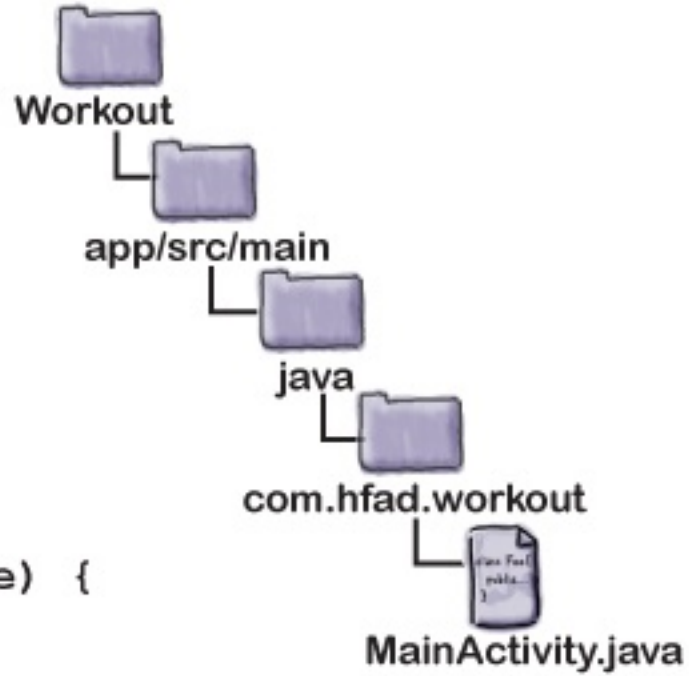


```
getFragmentManager().findFragmentById(R.id.fragment_id)
```

This is the ID of the fragment in the activity's layout.

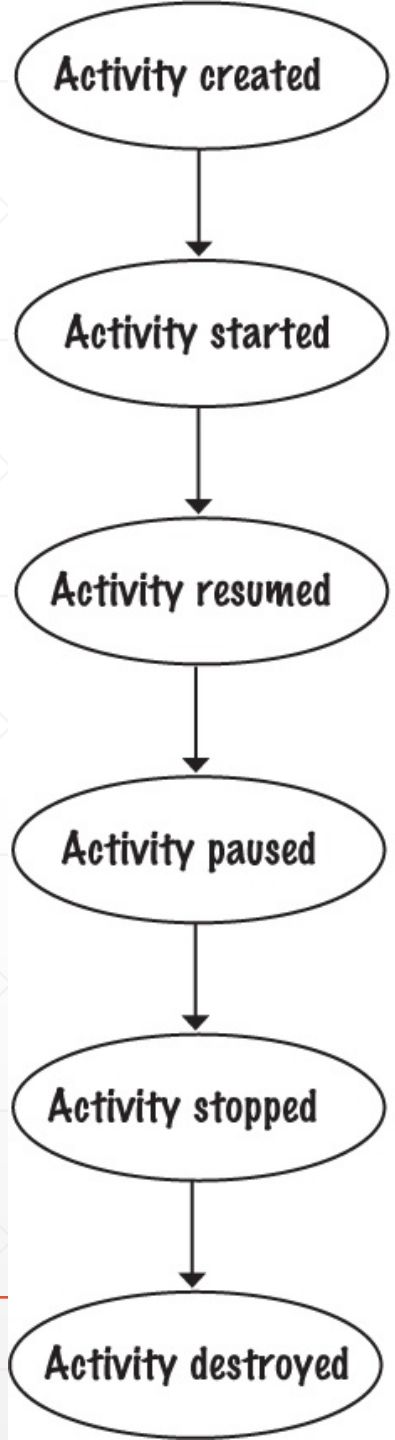
findFragmentById() is a bit like findViewById() except you use it to get a reference to a fragment.

```
package com.hfad.workout;

import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        WorkoutDetailFragment frag = (WorkoutDetailFragment)
                getFragmentManager().findFragmentById(R.id.detail_frag);
        frag.setWorkout(1);
    }
}
```



We're going to get WorkoutDetailFragment to display details of a workout here to check it's working.

This gets us a reference to WorkoutDetailFragment. Its id in the activity's layout is detail_frag.



The activity is created when its onCreate() method runs.
At this point, the activity is initialized, but isn't visible.

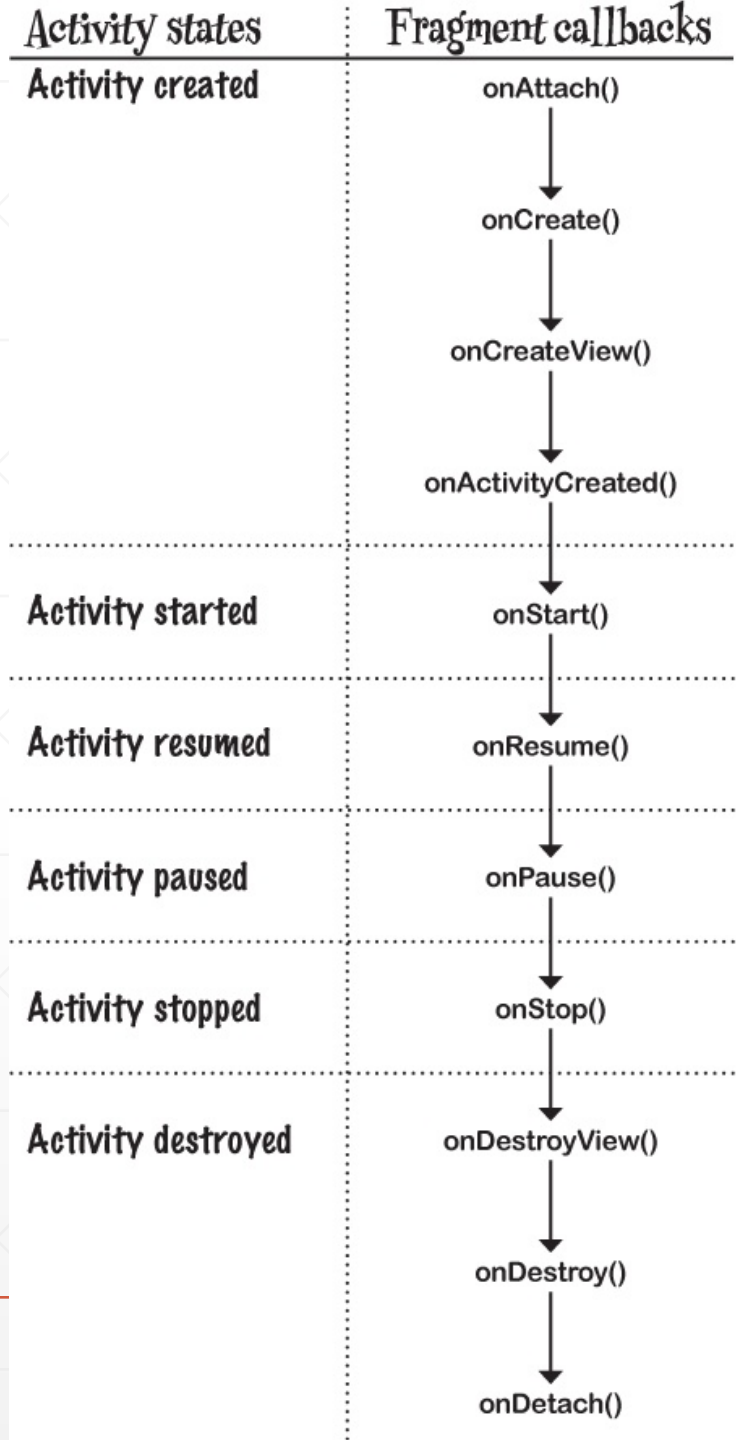
The activity is started when its onStart() method runs.
The activity is visible, but doesn't have the focus.

The activity is resumed when its onResume() method runs.
The activity is visible, and has the focus.

The activity is paused when its onPause() method runs.
The activity is still visible, but no longer has the focus.

The activity is stopped when its onStop() method runs.
The activity is no longer visible, but still exists.

The activity is destroyed when its onDestroy() method runs.
The activity no longer exists.



onAttach(Activity)
This happens when the fragment is associated with an activity.

onCreate(Bundle)
This is very similar to the activity's `onCreate()` method. It can be used to do the initial setup of the fragment.

onCreateView(LayoutInflater, ViewGroup, Bundle)
Fragments use a layout inflater to create their view at this stage.

onActivityCreated(Bundle)
This method is called when the `onCreate()` method of the activity has completed.

onStart()
The `onStart()` method is called when the fragment is about to become visible.

onResume()
Called when the fragment is visible and actively running.

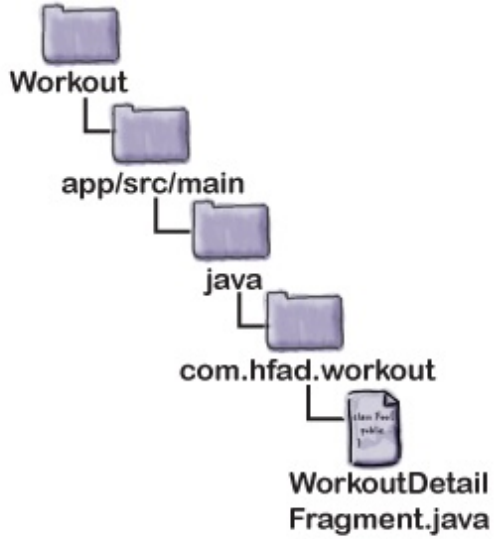
onPause()
Called when the fragment is no longer interacting with the user.

onStop()
Called when the fragment is no longer visible to the user.

onDestroyView()
Gives the fragment the chance to clear away any resources that were associated with its view.

onDestroy()
In this method, the fragment can clear away any other resources it created.

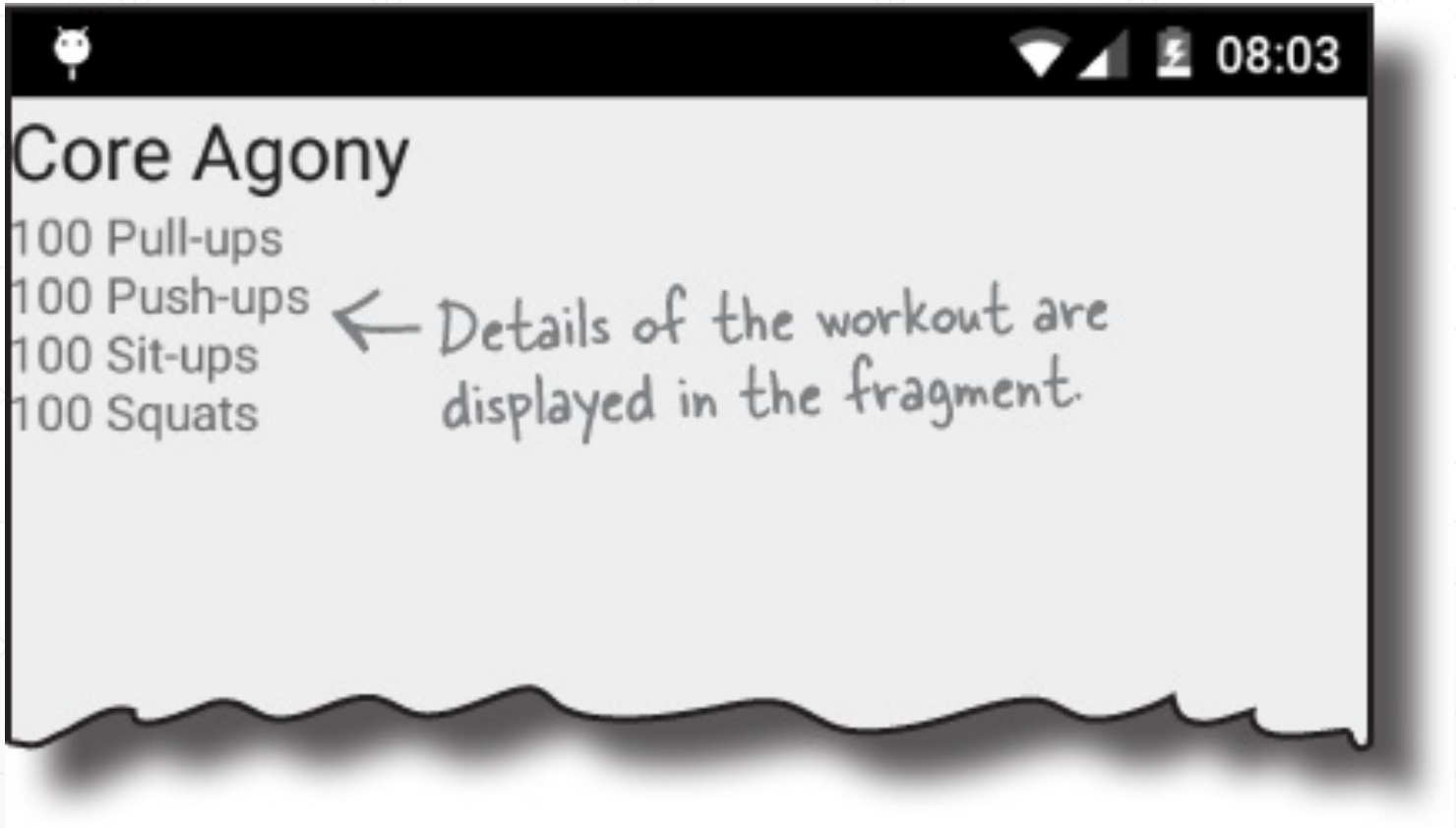
onDetach()
When the fragment finally loses contact with the activity.

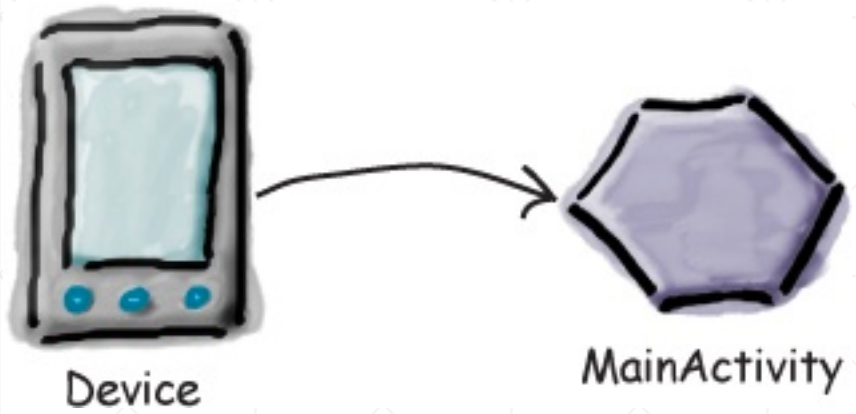



```
package com.hfad.workout;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class WorkoutDetailFragment extends Fragment {
    private long workoutId;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_workout_detail, container, false);
    }

    @Override
    public void onStart() {
        super.onStart();
        View view = getView();
        if (view != null) {
            TextView title = (TextView) view.findViewById(R.id.textTitle);
            Workout workout = Workout.workouts[(int) workoutId];
            title.setText(workout.getName());
            TextView description = (TextView) view.findViewById(R.id.textDescription);
            description.setText(workout.getDescription());
        }
    }
    public void setWorkout(long id) {
        this.workoutId = id;
    }
}
```

We're using this class in the onStart() method.

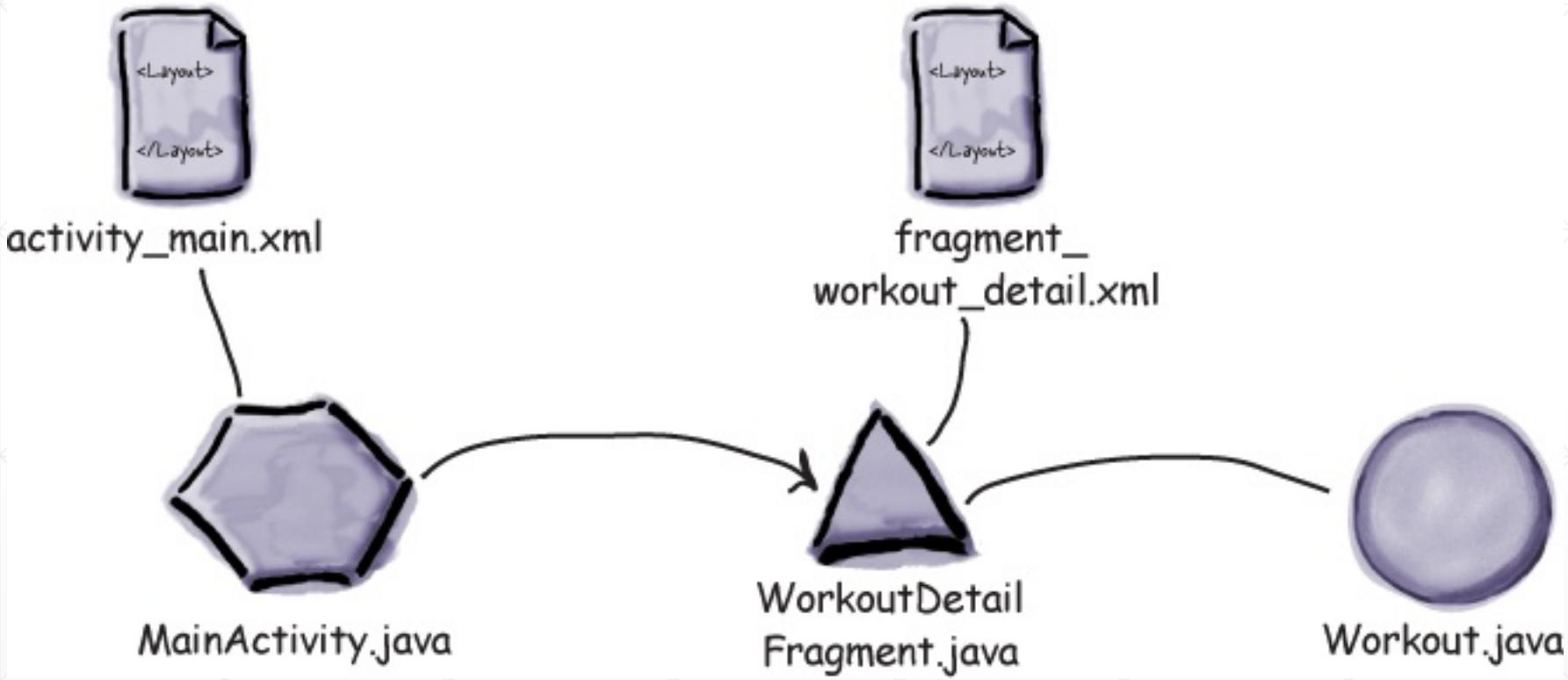
The getView() method gets the fragment's root View. We can then use this to get references to the workout title and description text views.

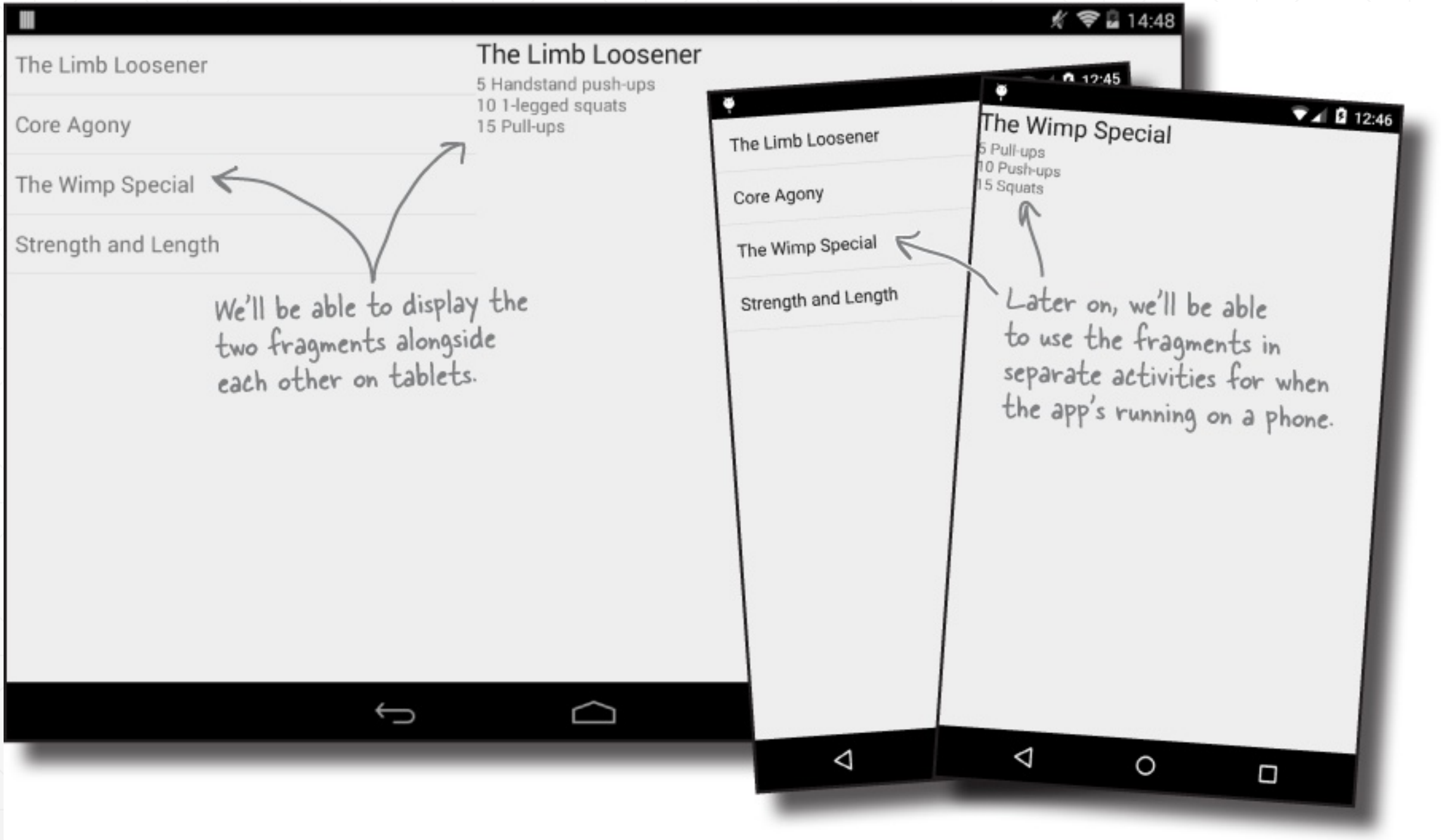

08:03
Core Agony
100 Pull-ups
100 Push-ups
100 Sit-ups
100 Squats
Details of the workout are displayed in the fragment.


Device
MainActivity

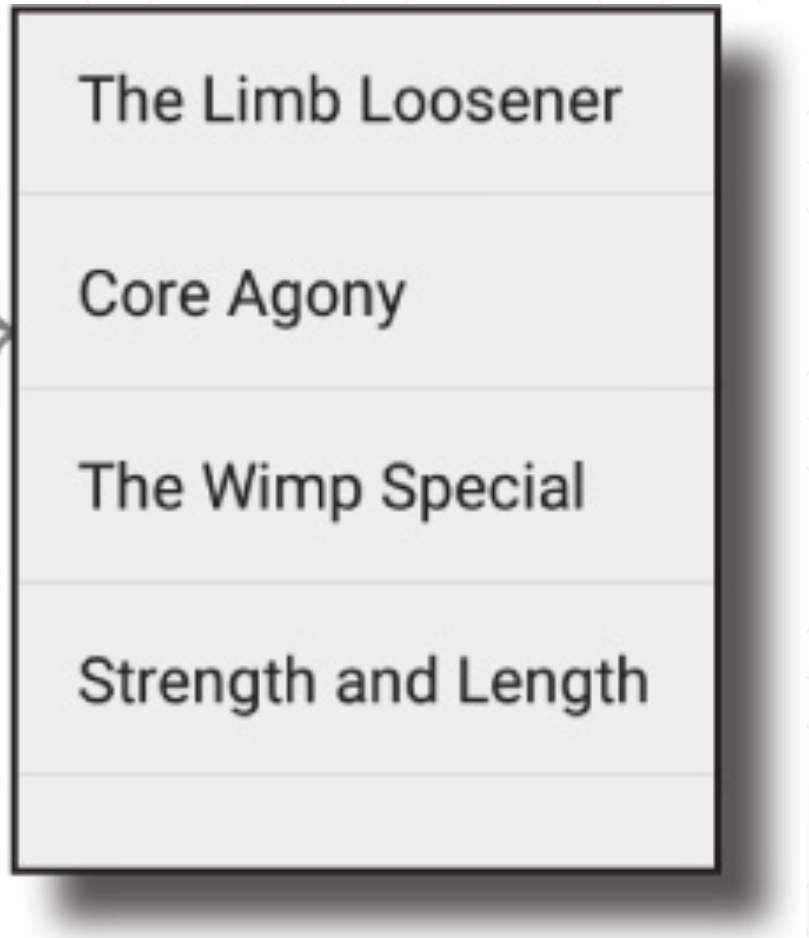

setWorkout(1)
MainActivity
WorkoutList
Fragment


textTitle: CoreAgony
textDescription: 100 Pull ups
100 Push-ups
100 Sit ups
100 Squats
MainActivity
WorkoutList
Fragment


<Layout>
</Layout>
activity_main.xml
<Layout>
</Layout>
fragment_
workout_detail.xml
MainActivity.java
WorkoutDetail
Fragment.java
Workout.java

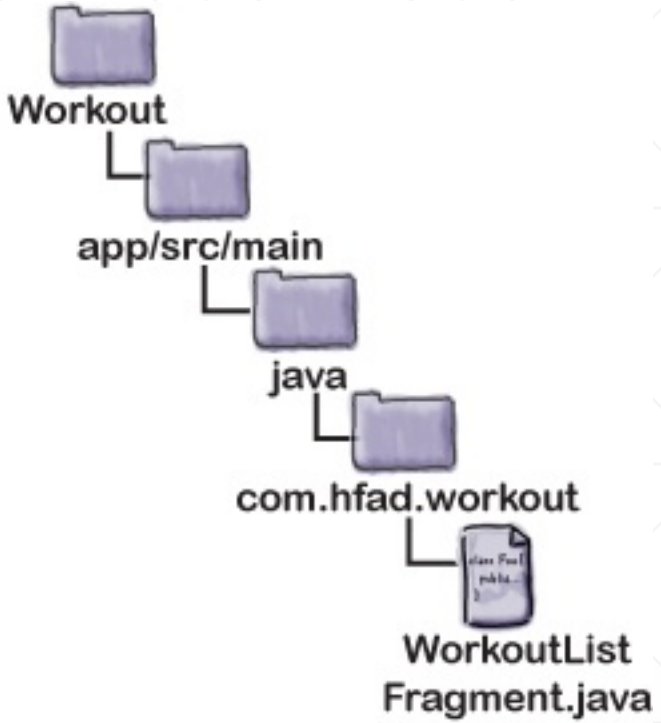

14:48
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
The Limb Loosener
5 Handstand push-ups
10 1-legged squats
15 Pull-ups
We'll be able to display the two fragments alongside each other on tablets.
12:45
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
12:46
The Wimp Special
5 Pull-ups
10 Push-ups
15 Squats
Later on, we'll be able to use the fragments in separate activities for when the app's running on a phone.

A list fragment comes complete with its own list view so you don't need to add it yourself. You just need to provide it with data.

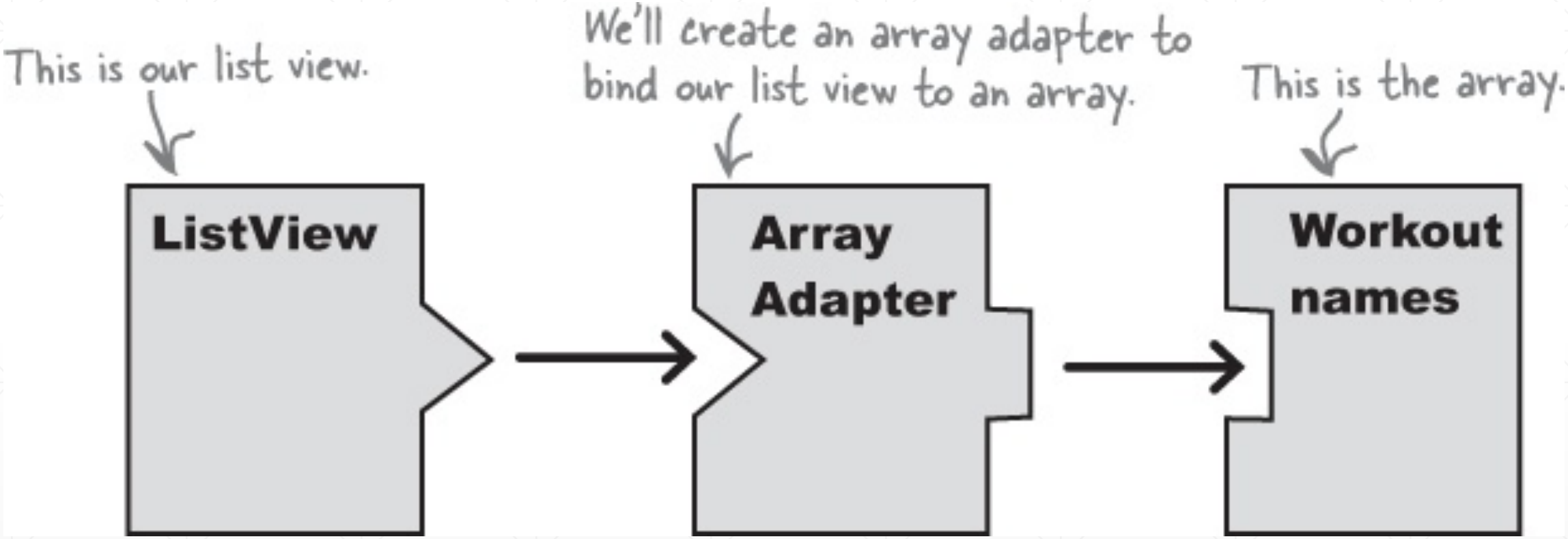


```
package com.hfad.workout;

import android.os.Bundle;
import android.app.ListFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class WorkoutListFragment extends ListFragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return super.onCreateView(inflater, container, savedInstanceState);
    }
}
```

The activity needs to extend ListFragment, not Fragment.



Calling the superclass onCreateView() method gives you the default layout for the ListFragment.

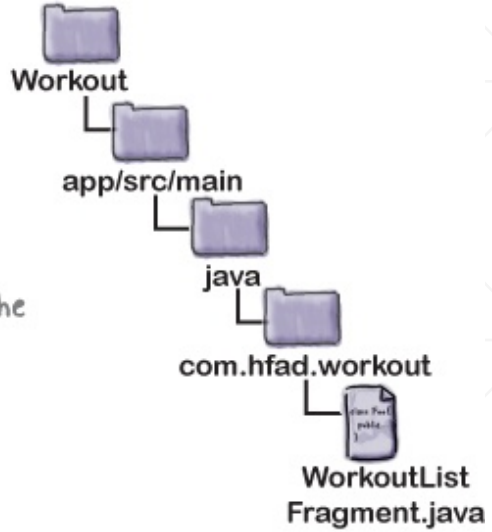

This is our list view.
We'll create an array adapter to bind our list view to an array.
This is the array.
ListView
Array Adapter
Workout names

```
ArrayAdapter<DataType> listAdapter = new ArrayAdapter<DataType>(
                context, android.R.layout.simple_list_item_1, array);
```

```
ArrayAdapter<DataType> listAdapter = new ArrayAdapter<DataType>(
    inflator.getContext(), android.R.layout.simple_list_item_1, array);
```

This gets you the current context.

```
setListAdapter(listAdapter);
```




```
package com.hfad.workout;

import android.os.Bundle;
import android.app.ListFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;

public class WorkoutListFragment extends ListFragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        String[] names = new String[Workout.workouts.length];
        for (int i = 0; i < names.length; i++) {
            names[i] = Workout.workouts[i].getName();
        }
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
                inflater.getContext(), android.R.layout.simple_list_item_1,
                names);
        setListAdapter(adapter);

        return super.onCreateView(inflater, container, savedInstanceState);
    }
}
```

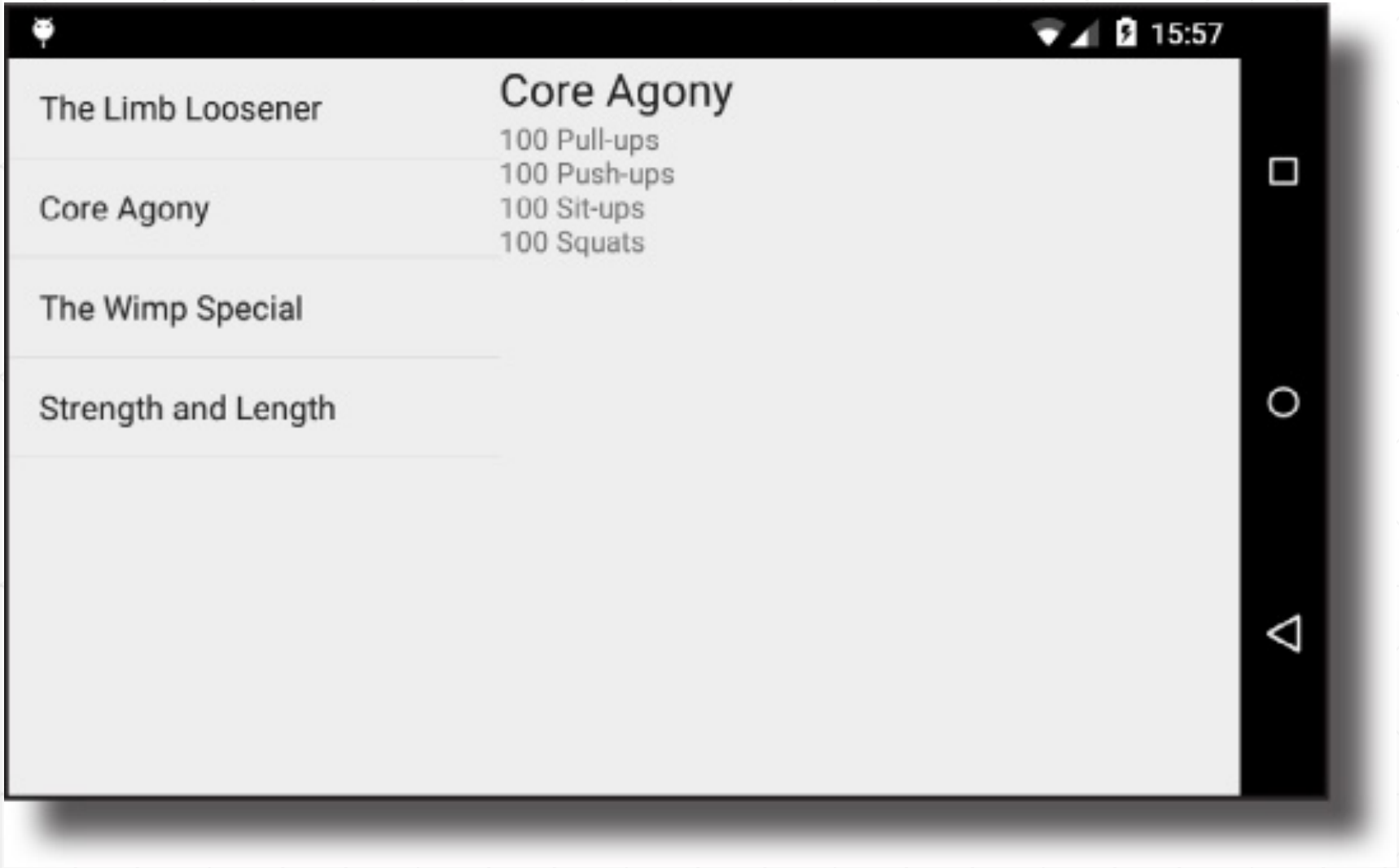
We're using this class in the onCreateView() method.

Create a String array of the workout names.

Create an array adapter.

Get the context from the layout inflater.

Bind the array adapter to the list view.

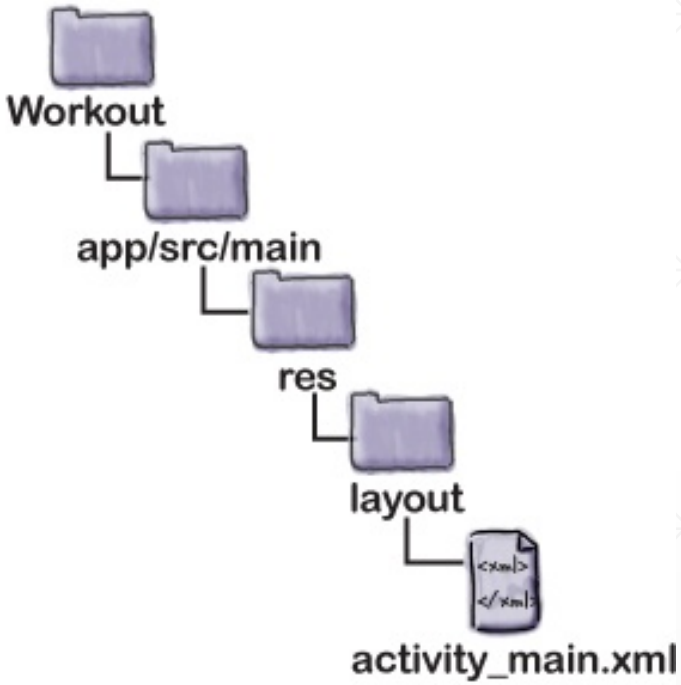

15:57
The Limb Loosener
Core Agony
Core Agony
100 Pull-ups
100 Push-ups
100 Sit-ups
100 Squats
The Wimp Special
Strength and Length

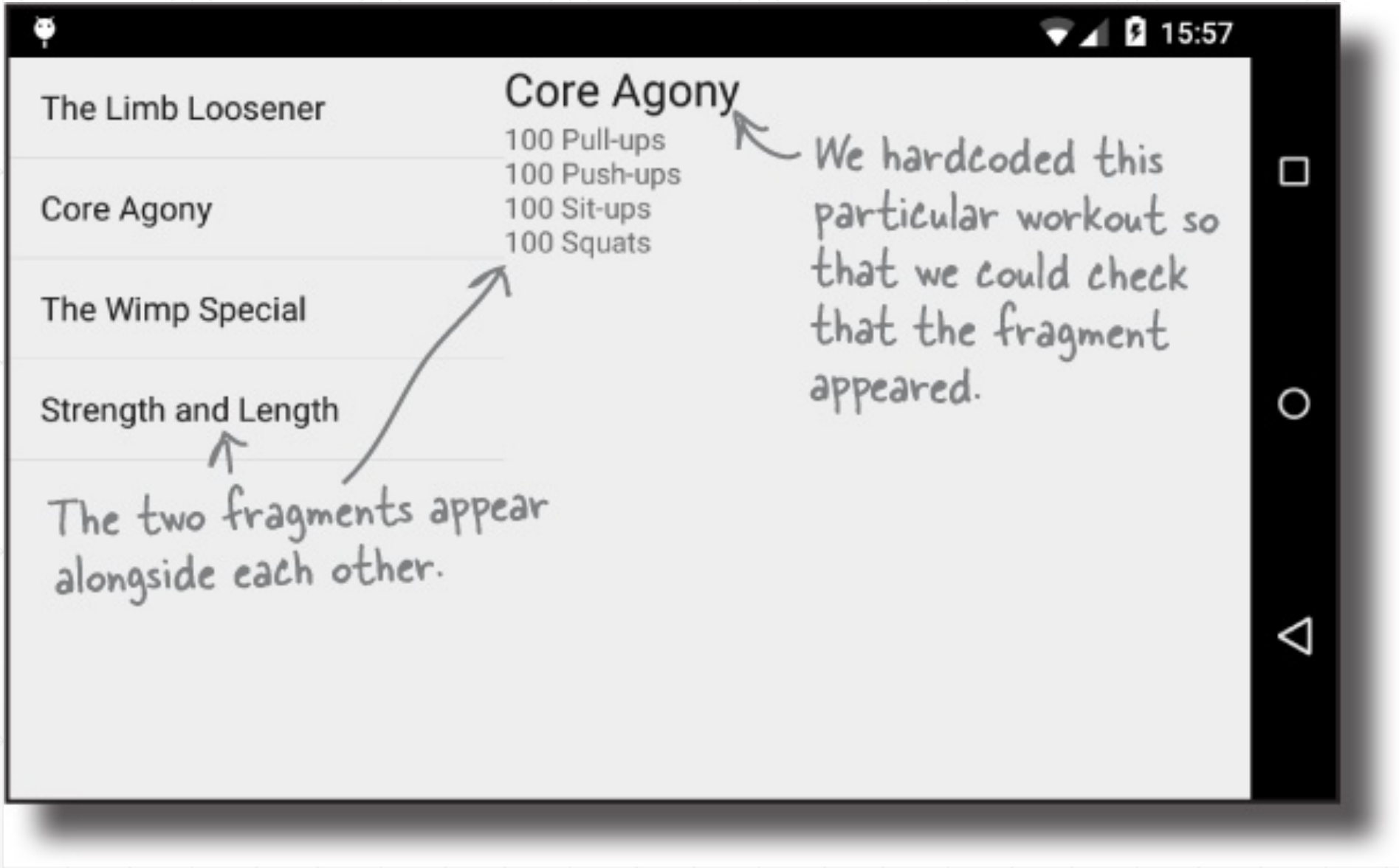
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment
        class="com.hfad.workout.WorkoutListFragment"
        android:id="@+id/list_frag"
        android:layout_width="0dp"
        android:layout_weight="2"
        android:layout_height="match_parent"/>

    <fragment
        class="com.hfad.workout.WorkoutDetailFragment"
        android:id="@+id/detail_frag"
        android:layout_width="0dp"
        android:layout_weight="3"
        android:layout_height="match_parent" />

</LinearLayout>
```

Display the list of workouts first.

Then display the workout details.

We're using layout_weight to control how much space each fragment should take up.

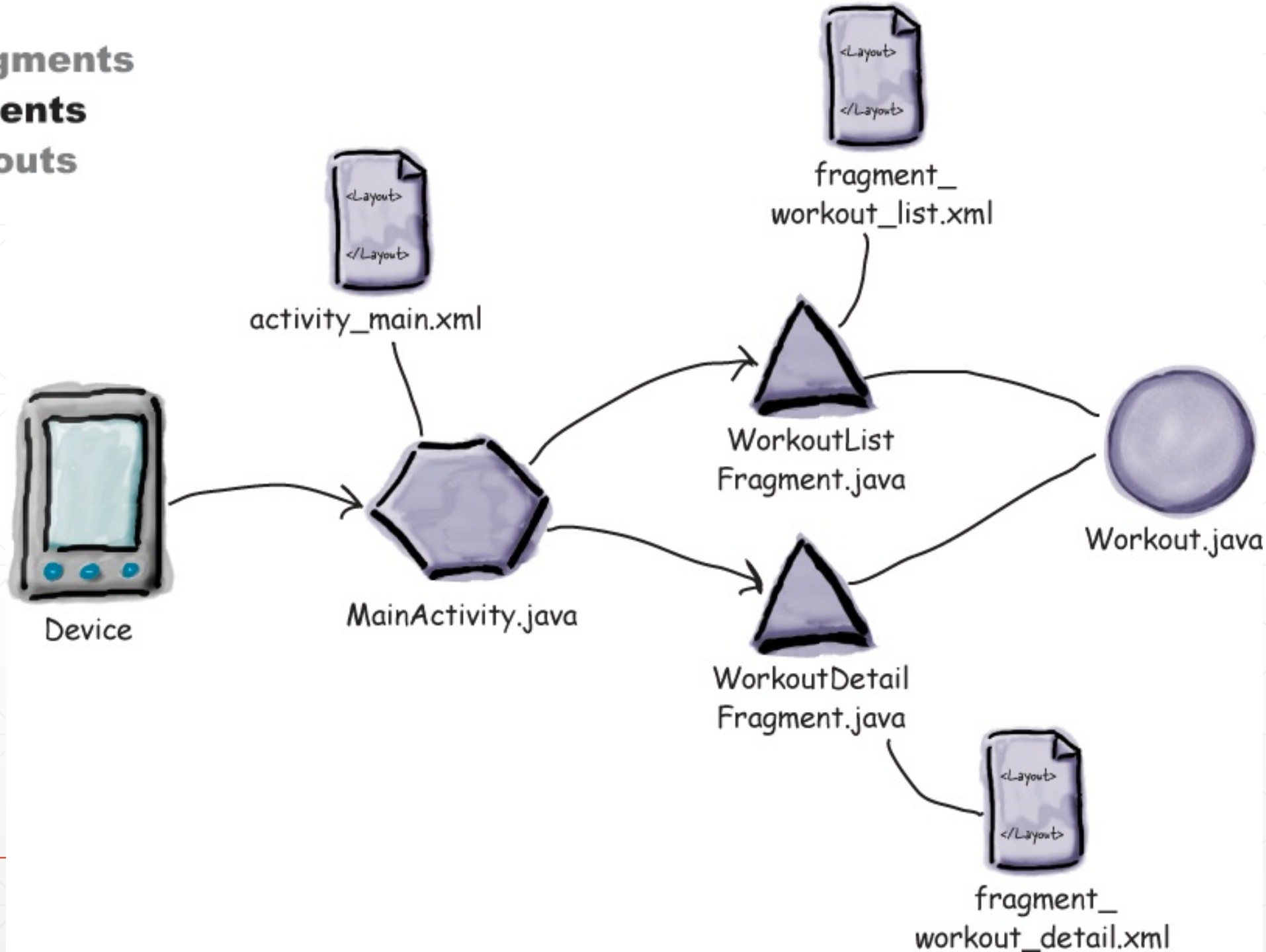



15:57
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
Core Agony
100 Pull-ups
100 Push-ups
100 Sit-ups
100 Squats
We hardcoded this particular workout so that we could check that the fragment appeared.
The two fragments appear alongside each other.

Create fragments

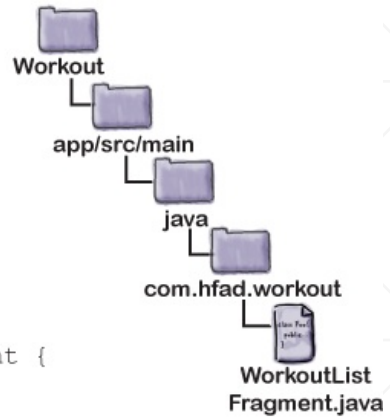
Link fragments

Device layouts



```
interface WorkoutListListener {
    void itemClicked(long id);
};
```

```
@Override
public void onAttach(Activity activity) {
    ...
}
```




```
package com.hfad.workout;

import android.os.Bundle;
import android.app.ListFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.app.Activity;
import android.widget.ListView;

public class WorkoutListFragment extends ListFragment {

    static interface WorkoutListListener {
        void itemClicked(long id);
    };

    private WorkoutListListener listener;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        String[] names = new String[Workout.workouts.length];
        for (int i = 0; i < names.length; i++) {
            names[i] = Workout.workouts[i].getName();
        }
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
                inflater.getContext(), android.R.layout.simple_list_item_1,
                names);
        setListAdapter(adapter);
        return super.onCreateView(inflater, container, savedInstanceState);
    }

    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        this.listener = (WorkoutListListener)activity;
    }

    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        if (listener != null) {
            listener.itemClicked(id);
        }
    }
}
```

Import these classes.

Add the listener to the fragment.

This is called when the fragment gets attached to the activity.

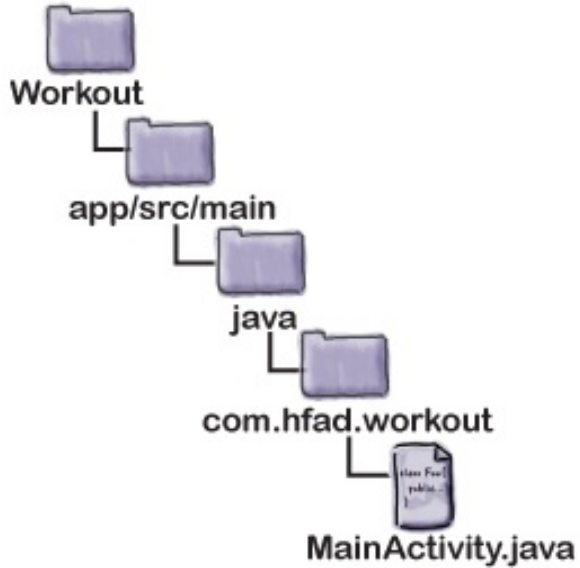
Tell the listener when an item in the ListView is clicked.

```
package com.hfad.workout;

import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity
                          implements WorkoutListFragment.WorkoutListListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        WorkoutDetailFragment frag = (WorkoutDetailFragment)
                                getFragmentManager().findFragmentById(R.id.detail_frag);
        frag.setWorkout(1);
    }

    @Override
    public void itemClicked(long id) {
        //The code to set the detail will go here
    }
}
```

Implement the listener defined in WorkoutListFragment.

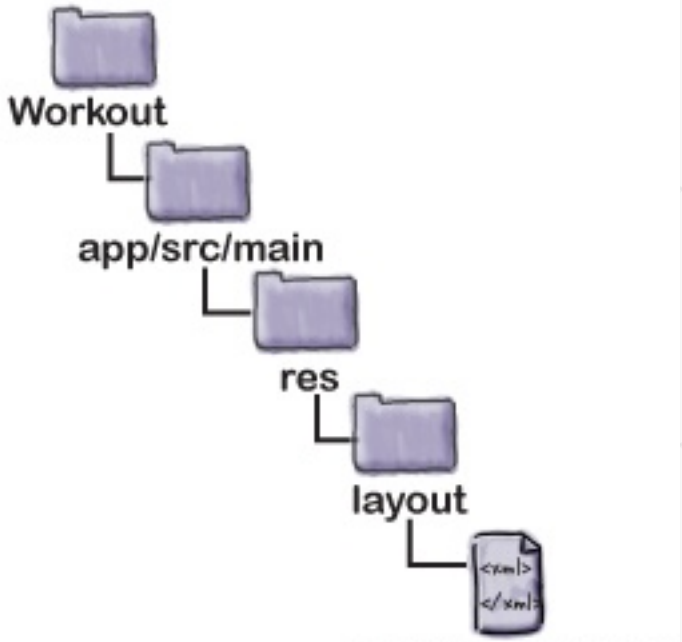
We'll remove these lines as we're no longer hardcoding which workout to display.

This method is defined in the listener.




11:34
The Limb Loosener
5 Handstand push-ups
10 1-legged squats
15 Pull-ups
The user clicks on the Limb Loosener workout.
11:35
The Wimp Special
5 Pull-ups
10 Push-ups
15 Squats
They then click on the Wimp Special.
11:36
The Limb Loosener
5 Handstand push-ups
10 1-legged squats
15 Pull-ups
When the user clicks on the device back button, we want the app to go back to the Limb Loosener workout.

```
<?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
        class="com.hfad.workout.WorkoutListFragment"
        android:id="@+id/list_frag"
        android:layout_width="0dp"
        android:layout_weight="2"
        android:layout_height="match_parent"/>

    <FrameLayout
        android:id="@+id/fragment_container"
        android:layout_width="0dp"
        android:layout_weight="3"
        android:layout_height="match_parent" />
</LinearLayout>
```



We're going to display the fragment inside a FrameLayout.

We'll add the fragment to the frame layout programmatically.

```
WorkoutDetailFragment fragment = new WorkoutDetailFragment();
FragmentTransaction transaction = getFragmentManager().beginTransaction();
```

The start of the fragment transaction

```
transaction.replace(R.id.fragment_container, fragment);
```

This replaces the fragment held in the fragment container.

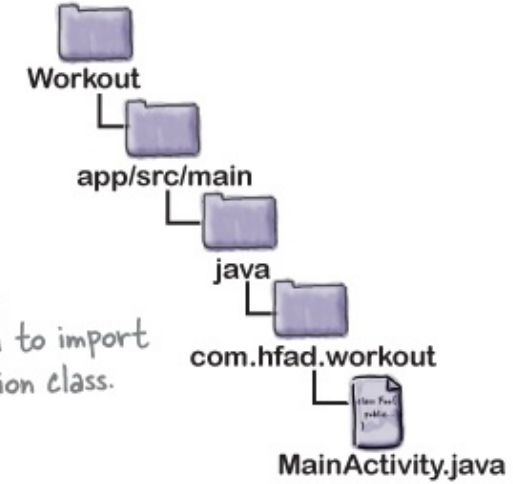
```
transaction.add(R.id.fragment_container, fragment);
transaction.remove(fragment);
```

You can add or remove fragments if you want. In our example, we don't need to.

```
transaction.setTransition(transition);
```
← You don't have to set a transition.

```
transaction.addToBackStack(null);
```
← Most of the time you won't need to retrieve the transaction, so it can be set to null.

```
transaction.commit();
```




```
package com.hfad.workout;

import android.app.Activity;
import android.os.Bundle;
import android.app.FragmentTransaction;

public class MainActivity extends Activity
                          implements WorkoutListFragment.WorkoutListListener {

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

    @Override
    public void itemClicked(long id) {
        WorkoutDetailFragment details = new WorkoutDetailFragment();
        FragmentTransaction ft = getFragmentManager().beginTransaction();
        details.setWorkout(id);
        ft.replace(R.id.fragment_container, details);
        ft.addToBackStack(null);
        ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
        ft.commit();
    }
}
```

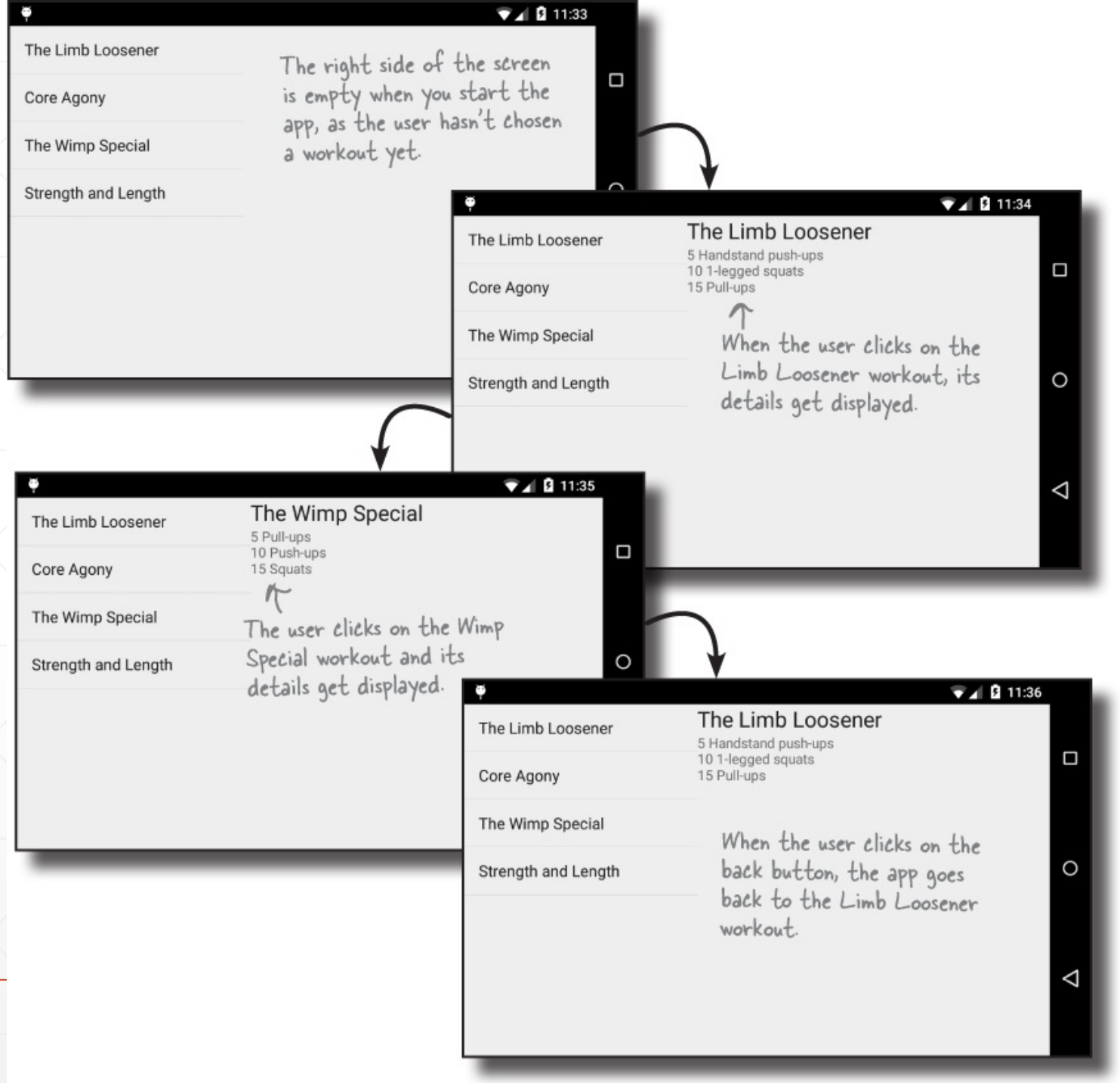
We're using a fragment transaction, so we need to import the FragmentTransaction class.

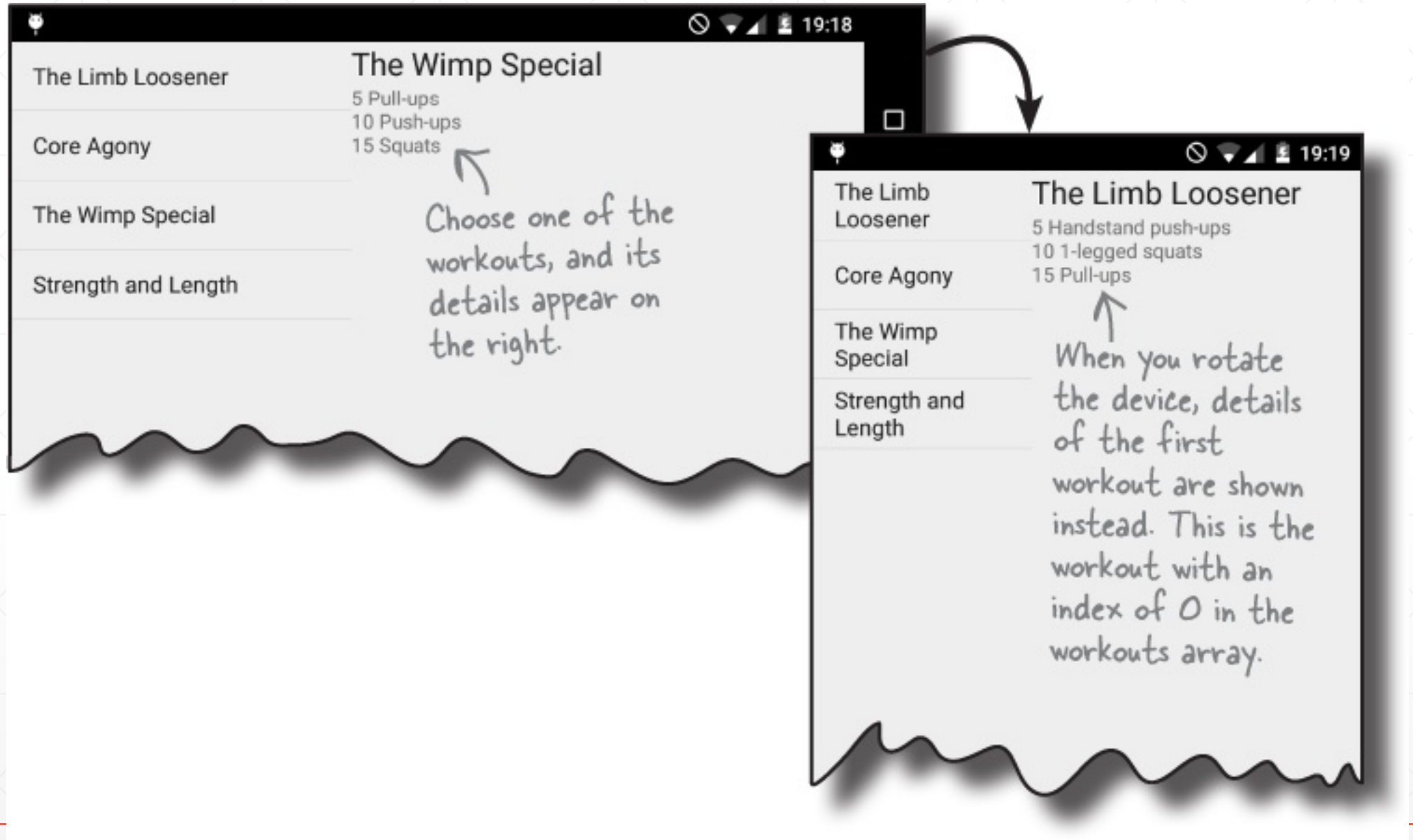
Start the fragment transaction.

Replace the fragment and add it to the back stack.

Get the new and old fragments to fade in and out

Commit the transaction.


11:33
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
The right side of the screen is empty when you start the app, as the user hasn't chosen a workout yet.
11:34
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
The Limb Loosener
5 Handstand push-ups
10 1-legged squats
15 Pull-ups
When the user clicks on the Limb Loosener workout, its details get displayed.
11:35
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
The Wimp Special
5 Pull-ups
10 Push-ups
15 Squats
The user clicks on the Wimp Special workout and its details get displayed.
11:36
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
The Limb Loosener
5 Handstand push-ups
10 1-legged squats
15 Pull-ups
When the user clicks on the back button, the app goes back to the Limb Loosener workout.

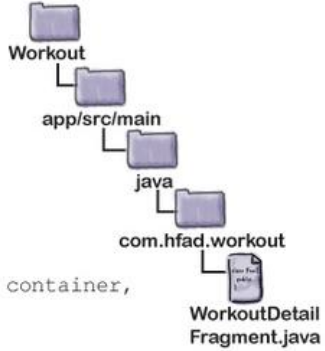

19:18
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
The Wimp Special
5 Pull-ups
10 Push-ups
15 Squats
Choose one of the workouts, and its details appear on the right.
19:19
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
The Limb Loosener
5 Handstand push-ups
10 1-legged squats
15 Pull-ups
When you rotate the device, details of the first workout are shown instead. This is the workout with an index of 0 in the workouts array.

```
public void onSaveInstanceState(Bundle savedInstanceState) {
    savedInstanceState.putLong("workoutId", workoutId);
}
```

The onSaveInstanceState() method gets called before the fragment is destroyed.

```
if (savedInstanceState != null) {
    workoutId = savedInstanceState.getLong("workoutId");
}
```

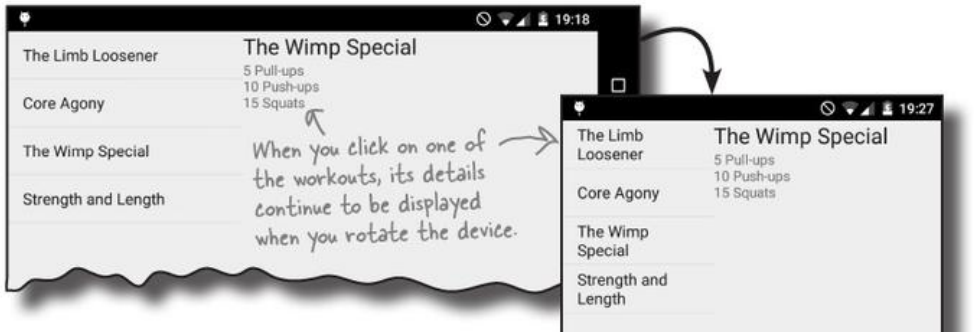
We can use it to get the previous state of the workoutId variable.




```
package com.hfad.workout;

... ← No new imports are required so we've skipped them.

public class WorkoutDetailFragment extends Fragment {
    private long workoutId;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        if (savedInstanceState != null) {
            workoutId = savedInstanceState.getLong("workoutId");
        }
        return inflater.inflate(R.layout.fragment_workout_detail, container, false);
    }

    @Override
    public void onStart() {
        super.onStart();
        View view = getView();
        if (view != null) {
            TextView title = (TextView) view.findViewById(R.id.textTitle);
            Workout workout = Workout.workouts[(int) workoutId];
            title.setText(workout.getName());
            TextView description = (TextView) view.findViewById(R.id.textDescription);
            description.setText(workout.getDescription());
        }
    }

    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        savedInstanceState.putLong("workoutId", workoutId);
    }

    public void setWorkout(long id) {
        this.workoutId = id;
    }
}
```

← Set the value of the workoutId.

Save the value of the workoutId in the savedInstanceState Bundle before the fragment gets destroyed. We're retrieving it in the onCreateView() method.

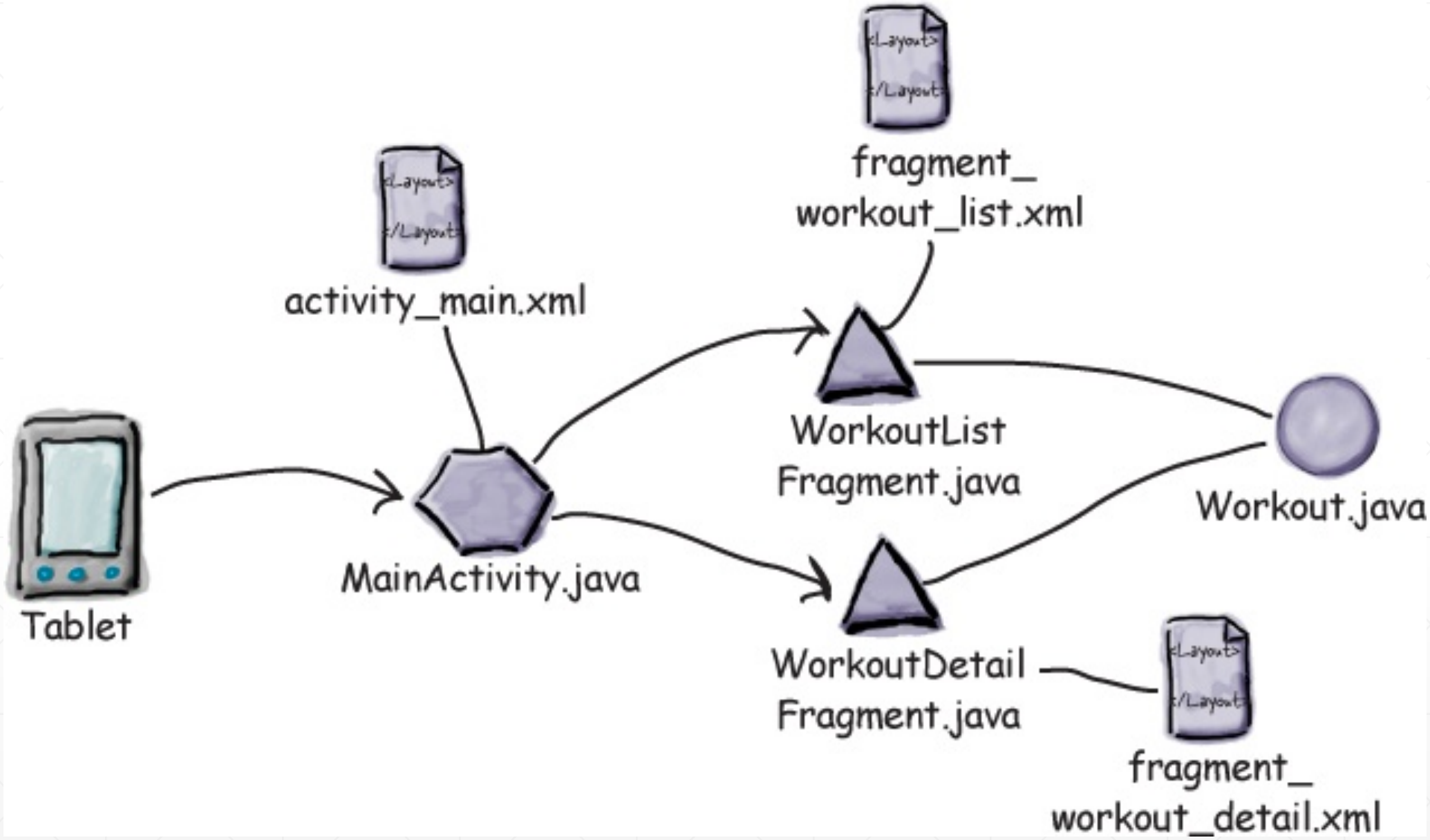


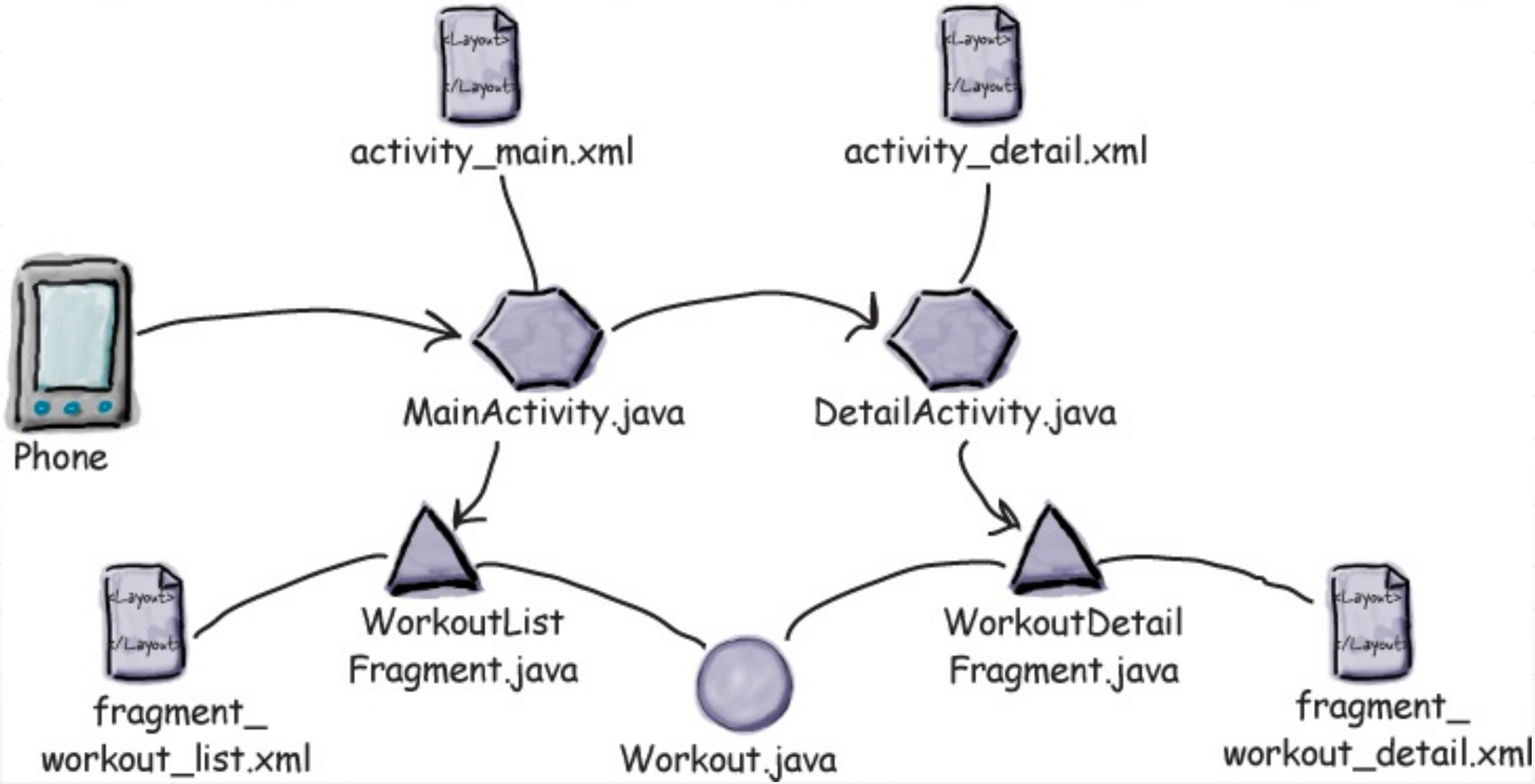
Create fragments

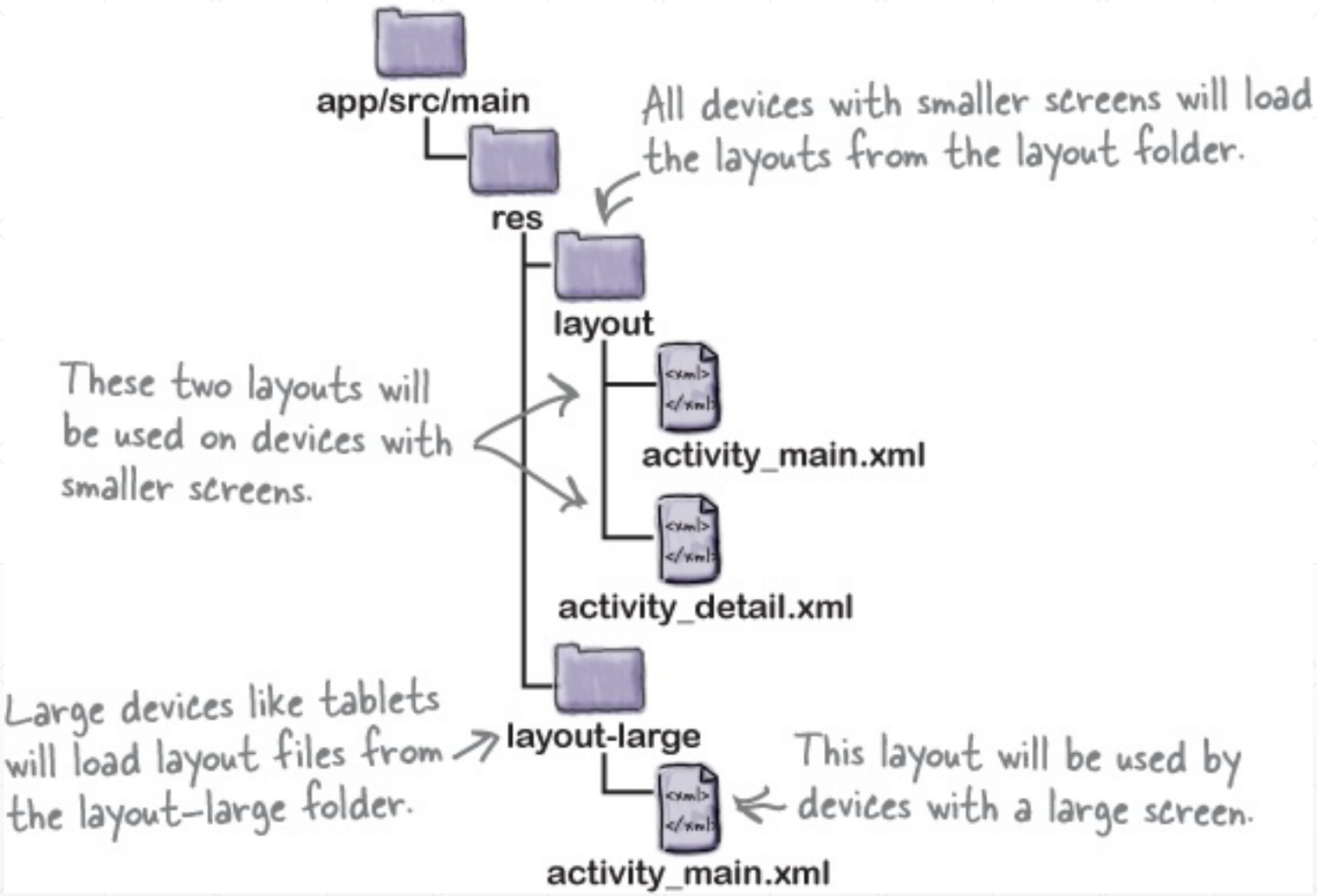
Link fragments

Device layouts


12:45
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
These are two different activities, each one displaying a different fragment.
12:46
The Wimp Special
5 Pull-ups
10 Push-ups
15 Squats


fragment_
workout_list.xml
activity_main.xml
WorkoutList
Fragment.java
Workout.java
Tablet
MainActivity.java
WorkoutDetail
Fragment.java
fragment_
workout_detail.xml


activity_main.xml
activity_detail.xml
Phone
MainActivity.java
DetailActivity.java
WorkoutList
Fragment.java
WorkoutDetail
Fragment.java
fragment_
workout_list.xml
Workout.java
fragment_
workout_detail.xml


app/src/main
res
All devices with smaller screens will load the layouts from the layout folder.
layout
These two layouts will be used on devices with smaller screens.
activity_main.xml
activity_detail.xml
Large devices like tablets will load layout files from the layout-large folder.
layout-large
This layout will be used by devices with a large screen.
activity_main.xml

You must specify a resource type.

Screen density is based on dots per inch.

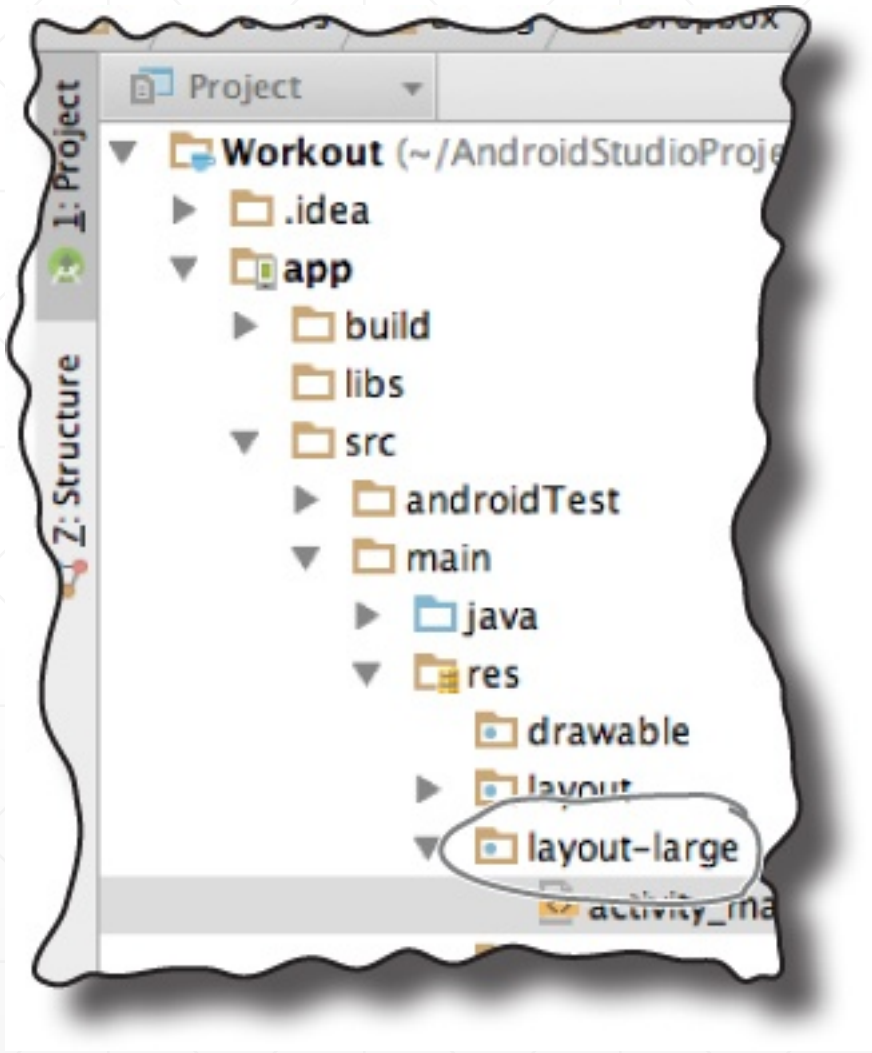
Resource type	Screen size	Screen density	Orientation	Aspect ratio
drawable	-small	-ldpi	-land	-long
layout	-normal	-mdpi	-port	-notlong
menu	-large	-hdpi		
mipmap	-xlarge	-xhdpi		
values		-xxhdpi		
		-xxxhdpi		
		-nodpi		
		-tvdpi		

long is for screens that have a very high value for height.

A mipmap resource is used for application icons. Older versions of Android Studio use drawables instead.

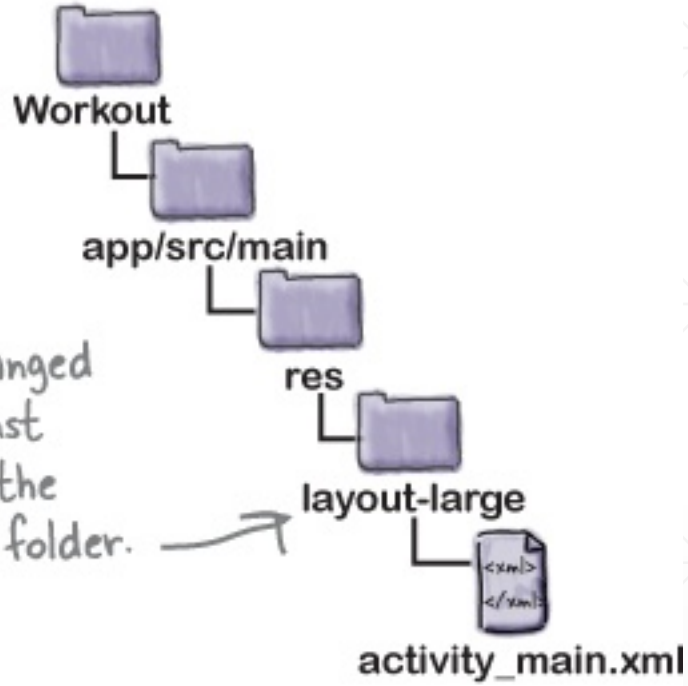
This is for density-independent resources. Use -nodpi for any image resources you don't want to scale (e.g., a folder called drawable-nodpi).

```
<supports-screens android:smallScreens="false"/>
```


Project
1: Project
2: Structure
Workout (~/AndroidStudioProje
.idea
app
build
libs
src
androidTest
main
java
res
drawable
layout-large

```
<?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
        class="com.hfad.workout.WorkoutListFragment"
        android:id="@+id/list_frag"
        android:layout_width="0dp"
        android:layout_weight="2"
        android:layout_height="match_parent"/>

    <FrameLayout
        android:id="@+id/fragment_container"
        android:layout_width="0dp"
        android:layout_weight="3"
        android:layout_height="match_parent" />
</LinearLayout>
```

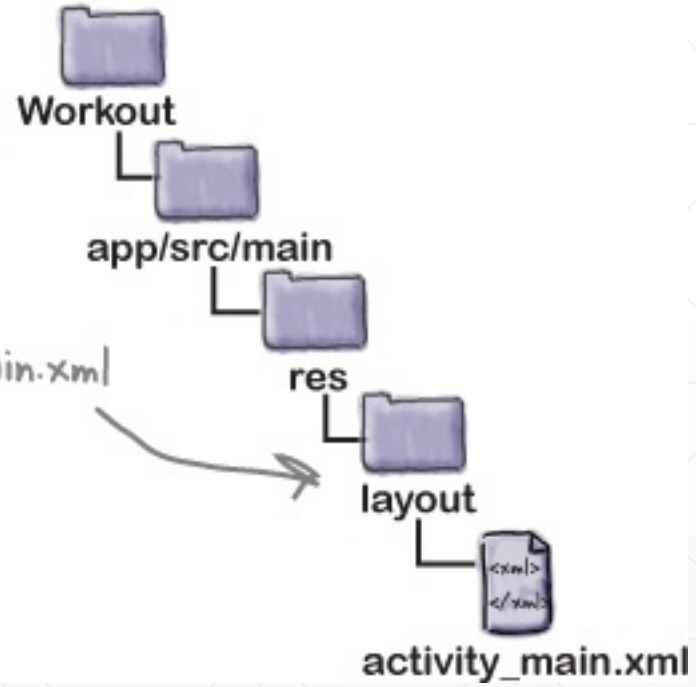
We've not changed the layout, just copied it to the layout-large folder.

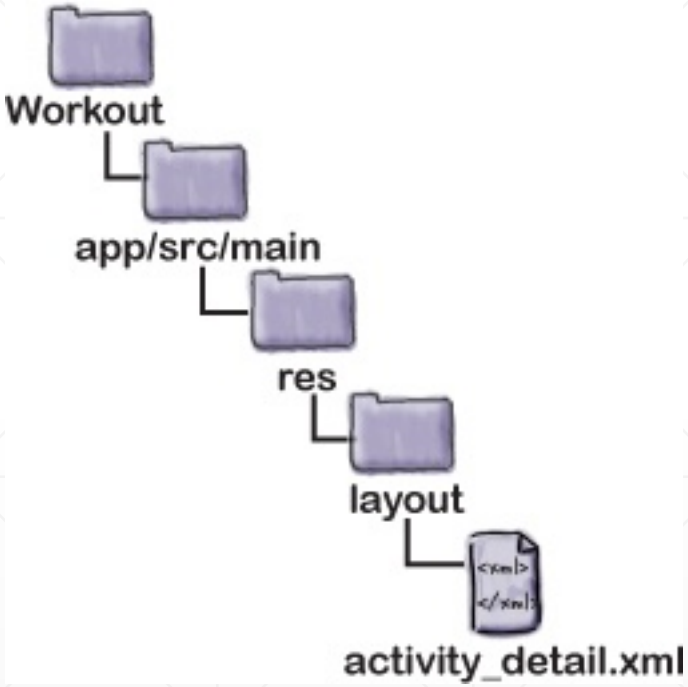



12:45
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
We just want MainActivity to display WorkoutListFragment on a phone.

```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    class="com.hfad.workout.WorkoutListFragment"
    android:id="@+id/list_frag"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

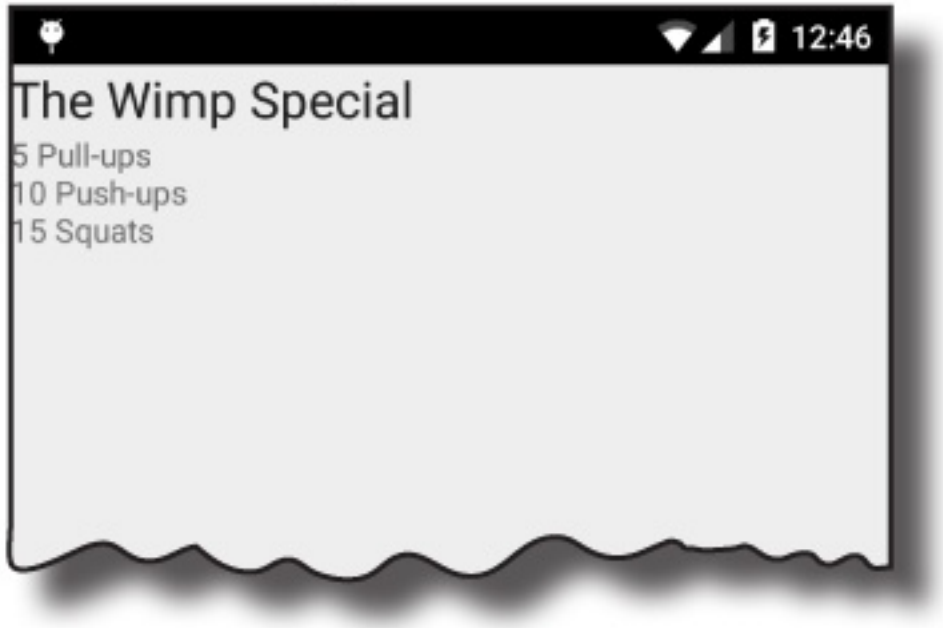
Make sure you edit activity_main.xml in the layout folder.

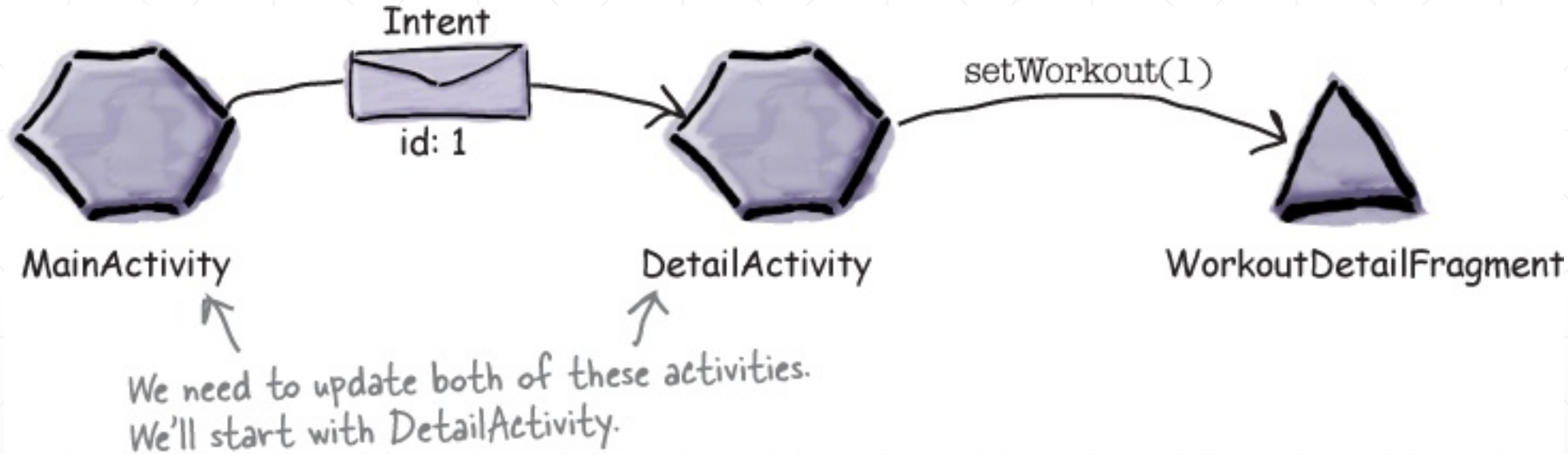



Workout
app/src/main
res
layout
activity_detail.xml

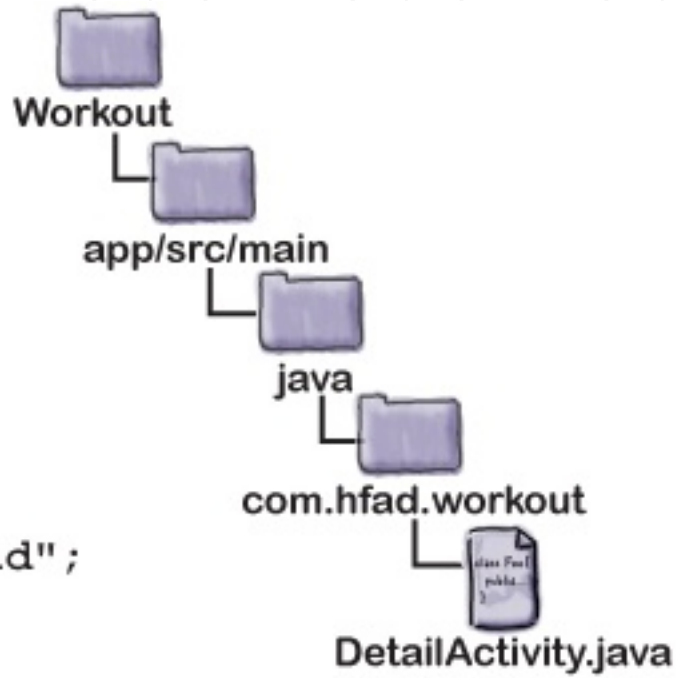
```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    class="com.hfad.workout.WorkoutDetailFragment"
    android:id="@+id/detail_frag"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

DetailActivity will just display WorkoutDetailFragment.




Intent
id: 1
setWorkout(1)
MainActivity
DetailActivity
WorkoutDetailFragment
We need to update both of these activities.
We'll start with DetailActivity.

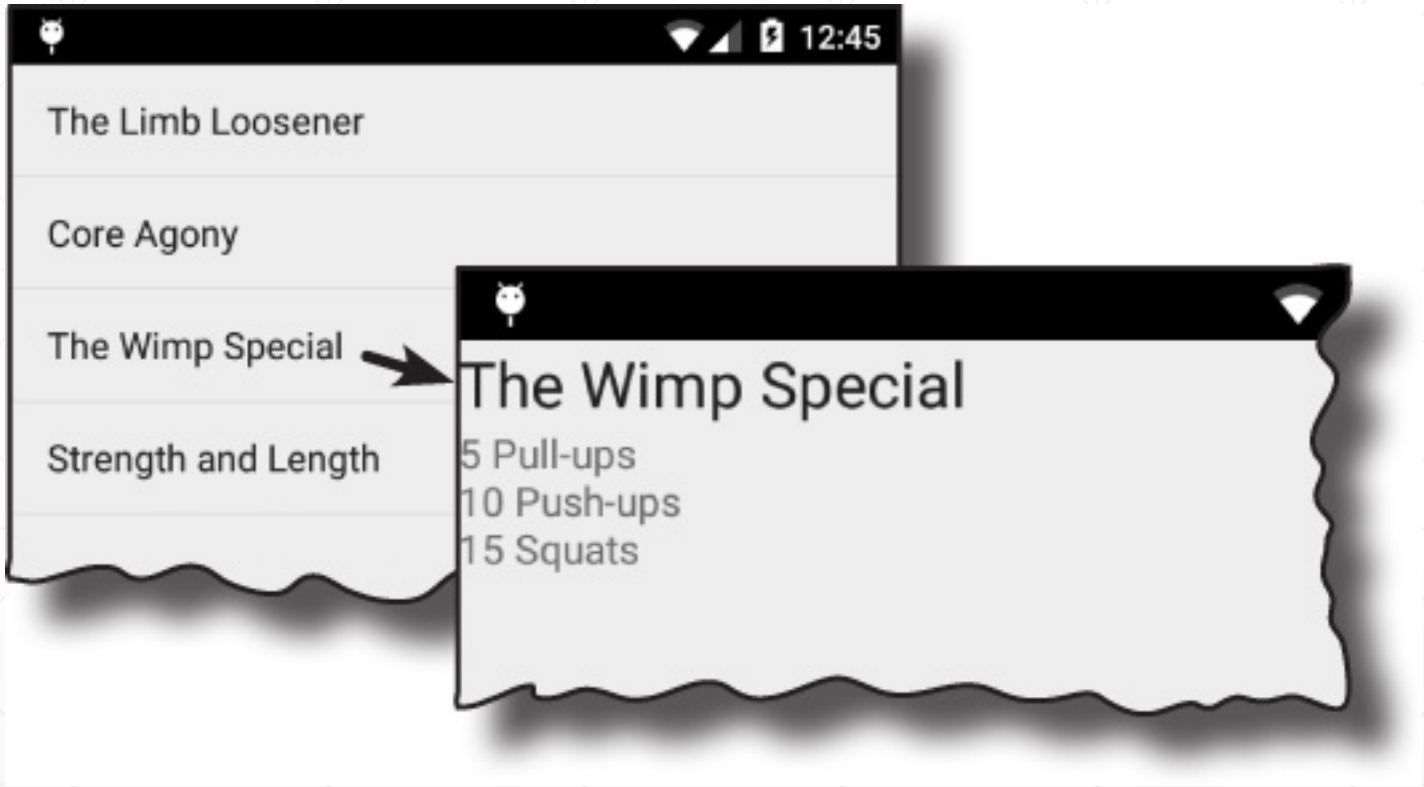
```
package com.hfad.workout;

import android.app.Activity;
import android.os.Bundle;

public class DetailActivity extends Activity {
    public static final String EXTRA_WORKOUT_ID = "id";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_detail);
        WorkoutDetailFragment workoutDetailFragment = (WorkoutDetailFragment)
                getFragmentManager().findFragmentById(R.id.detail_frag);
        int workoutId = (int) getIntent().getExtras().get(EXTRA_WORKOUT_ID);
        workoutDetailFragment.setWorkout(workoutId);
    }
}
```

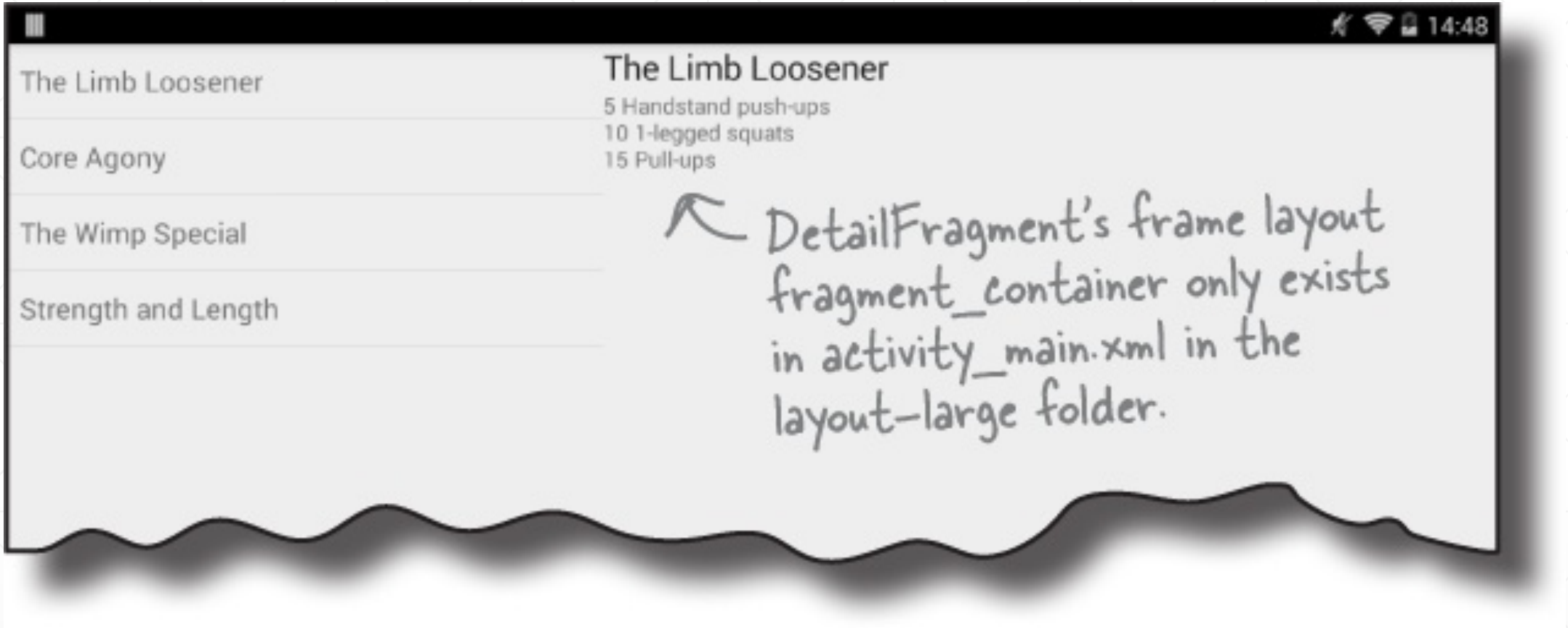


Get a reference to the fragment.

Get the ID of the workout the user clicked on from the intent.

Pass the workout ID to the fragment.


12:45
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
The Wimp Special
5 Pull-ups
10 Push-ups
15 Squats


14:48
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
The Limb Loosener
5 Handstand push-ups
10 1-legged squats
15 Pull-ups
DetailFragment's frame layout fragment_container only exists in activity_main.xml in the layout-large folder.

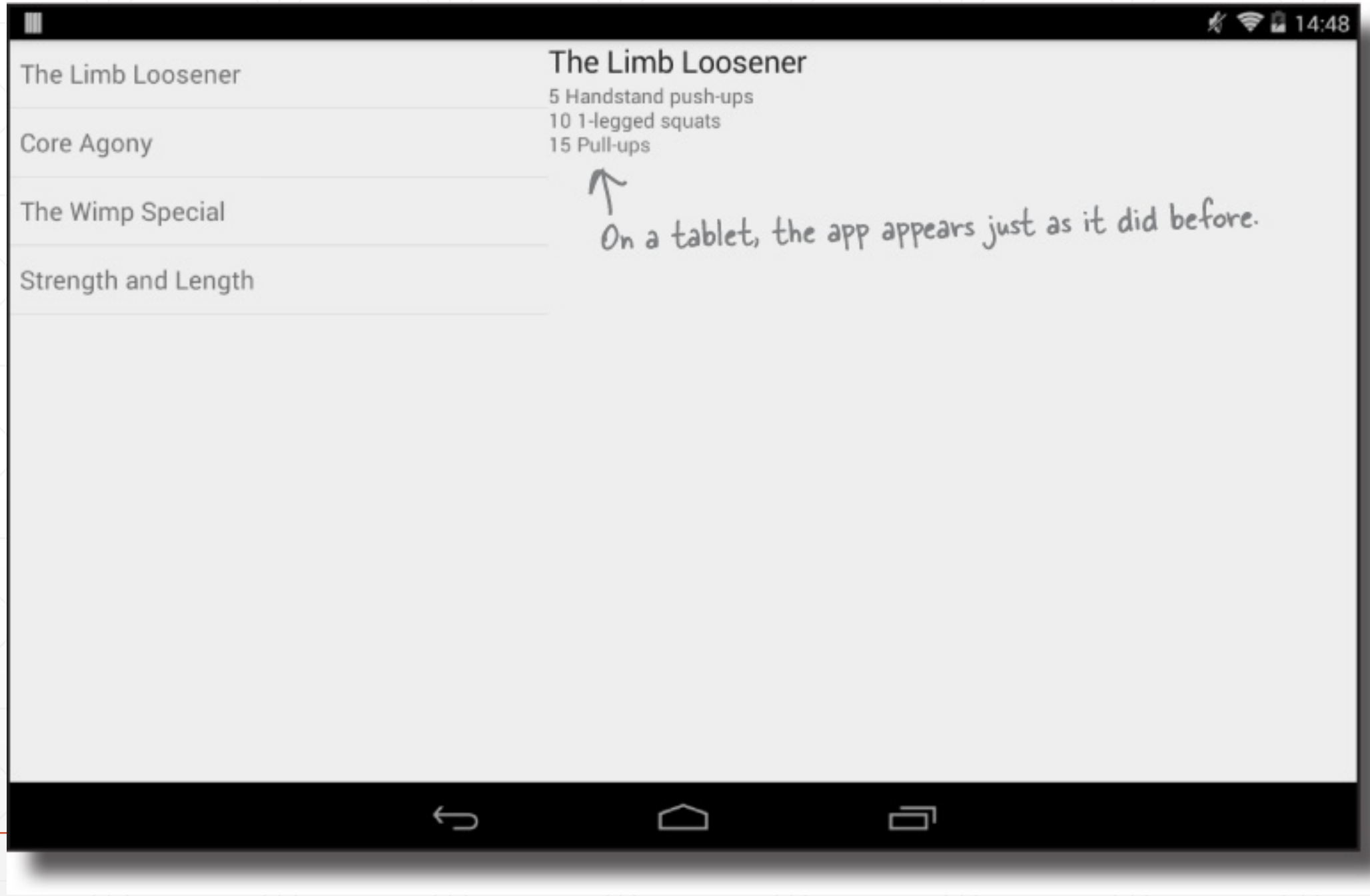
```
package com.hfad.workout;

import android.app.Activity;
import android.app.FragmentTransaction;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends Activity
                  implements WorkoutListFragment.WorkoutListListener {

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

    @Override
    public void itemClicked(long id) {
        View fragmentContainer = findViewById(R.id.fragment_container);
        if (fragmentContainer != null) {
            WorkoutDetailFragment details = new WorkoutDetailFragment();
            FragmentTransaction ft = getFragmentManager().beginTransaction();
            details.setWorkout(id);
            ft.replace(R.id.fragment_container, details);
            ft.addToBackStack(null);
            ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
            ft.commit();
        } else {
            Intent intent = new Intent(this, DetailActivity.class);
            intent.putExtra(DetailActivity.EXTRA_WORKOUT_ID, (int)id);
            startActivity(intent);
        }
    }
}
```

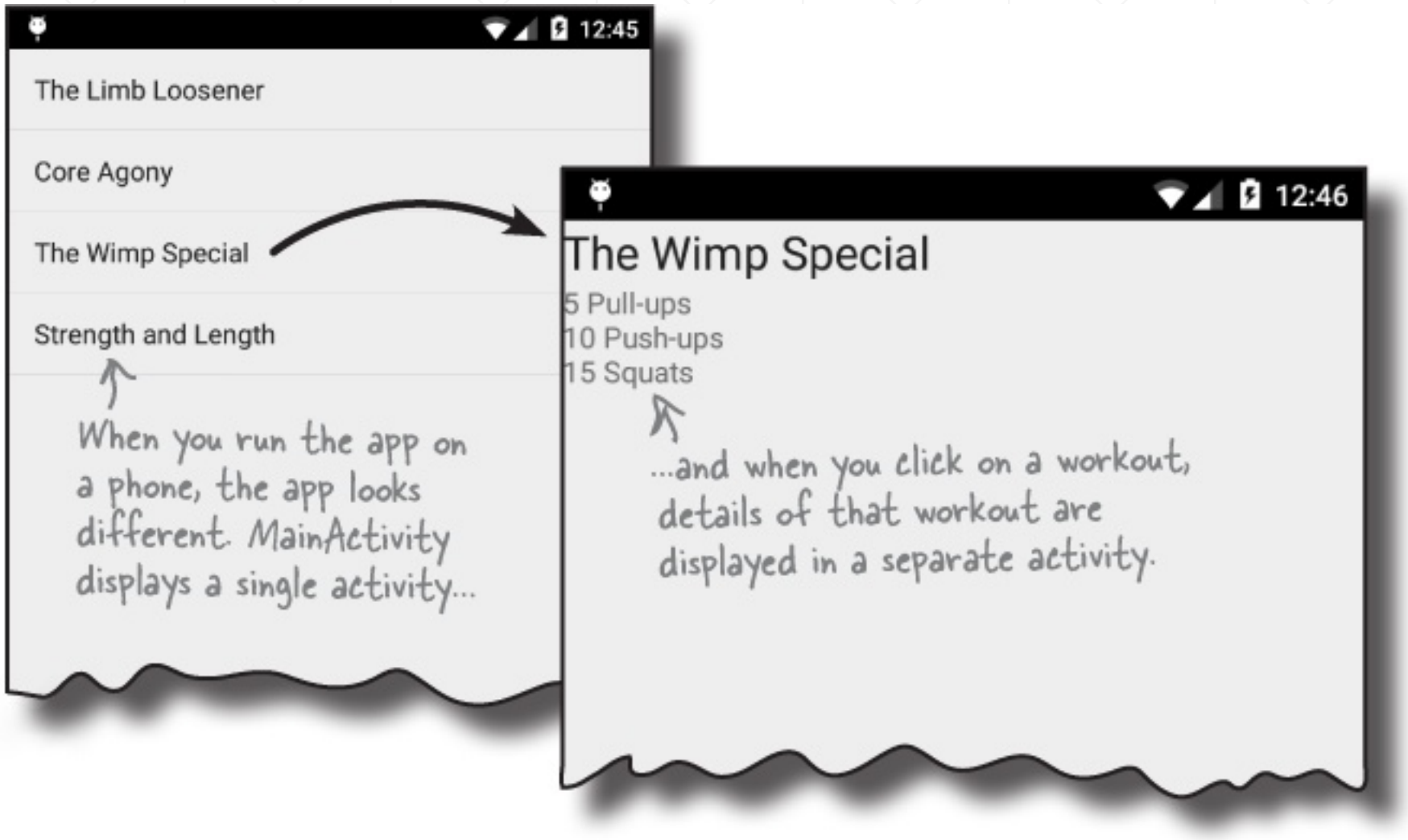
We're using these extra classes in the itemClicked() method.

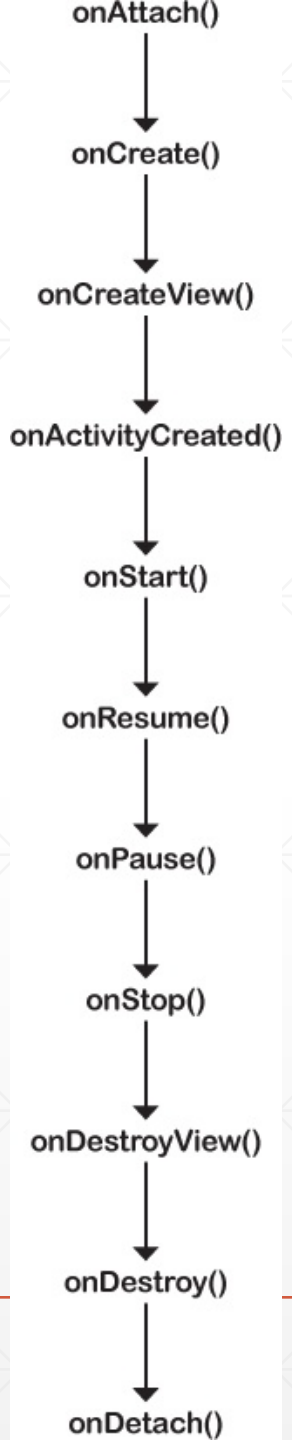
Get a reference to the frame layout that contains WorkoutDetailFragment. This will exist if the app is being run on a device with a large screen.

We only need to run this code if the frame layout is there.

If the frame layout isn't there, the app must be running on a device with a smaller screen. Start DetailActivity, passing it the ID of the workout.


14:48
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
The Limb Loosener
5 Handstand push-ups
10 1-legged squats
15 Pull-ups
On a tablet, the app appears just as it did before.


12:45
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
When you run the app on a phone, the app looks different. MainActivity displays a single activity...
12:46
The Wimp Special
5 Pull-ups
10 Push-ups
15 Squats
...and when you click on a workout, details of that workout are displayed in a separate activity.

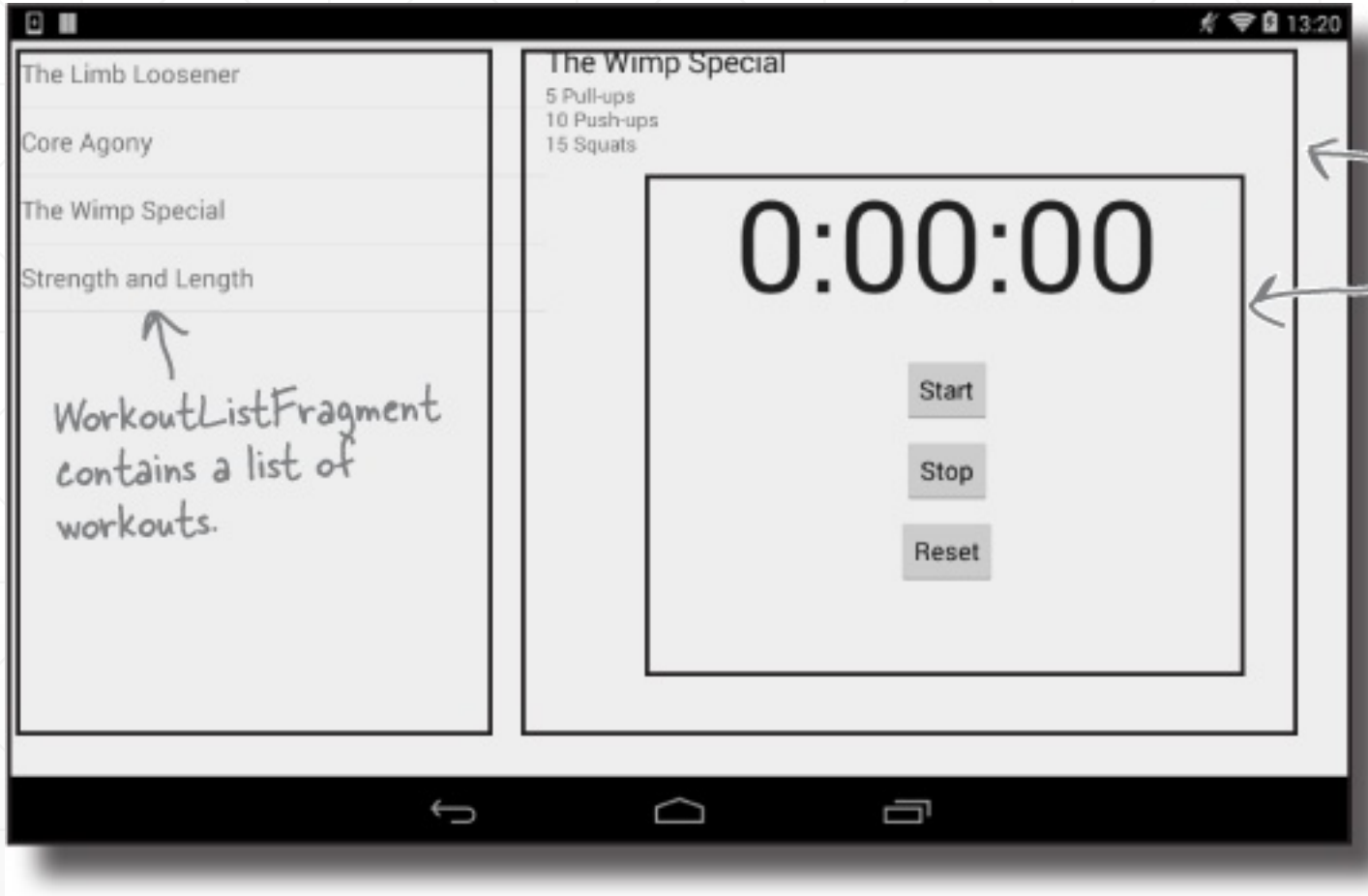

onAttach()
onCreate()
onCreateView()
onActivityCreated()
onStart()
onResume()
onPause()
onStop()
onDestroyView()
onDestroy()
onDetach()

Bullet Points

- A fragment is used to control part of a screen. It can be reused across multiple activities.
- A fragment has an associated layout.
- A fragment is a subclass of the android.app.Fragment class.
- The onCreateView() method gets called each time Android needs the fragment's layout.
- Add a fragment to an activity's layout using the <fragment> element and adding a classattribute.
- The fragment lifecycle methods tie in with the states of the activity that contains the fragment.

Bullet Points

- The Fragment class doesn’t extend the Activity class or implement the Context class.
- Fragments don’t have a findViewById() method. Instead, use the getView() method to get a reference to the root view, then call the view’s findViewById() method.
- A list fragment is a fragment that comes complete with a ListView. You create one by subclassing ListFragment.
- If you need to get a fragment to respond to changes in the user interface, use the<FrameLayout> element.
- Use fragment transactions to make a set of changes to an existing fragment and add to the back stack.
- Make apps look different on different devices by putting separate layouts in device-appropriate folders.



WorkoutListFragment contains a list of workouts.

WorkoutDetailFragment displays details of the workout the user clicks on.

We're going to add a stopwatch fragment to WorkoutDetailFragment.

Bullet Points

- Fragments can contain other fragments.
- If you're nesting a fragment in another fragment, you need to add the nested fragment programmatically in Java code.
- When you perform transactions on a nested fragment, use getChildFragmentManager() to create the transaction.
- If you use the android:onClick attribute in a fragment, Android will look for a method of that name in the fragment's parent activity.
- Instead of using the android:onClick attribute in a fragment, make the fragment implement the View.OnClickListener interface and implement its onClick() method..

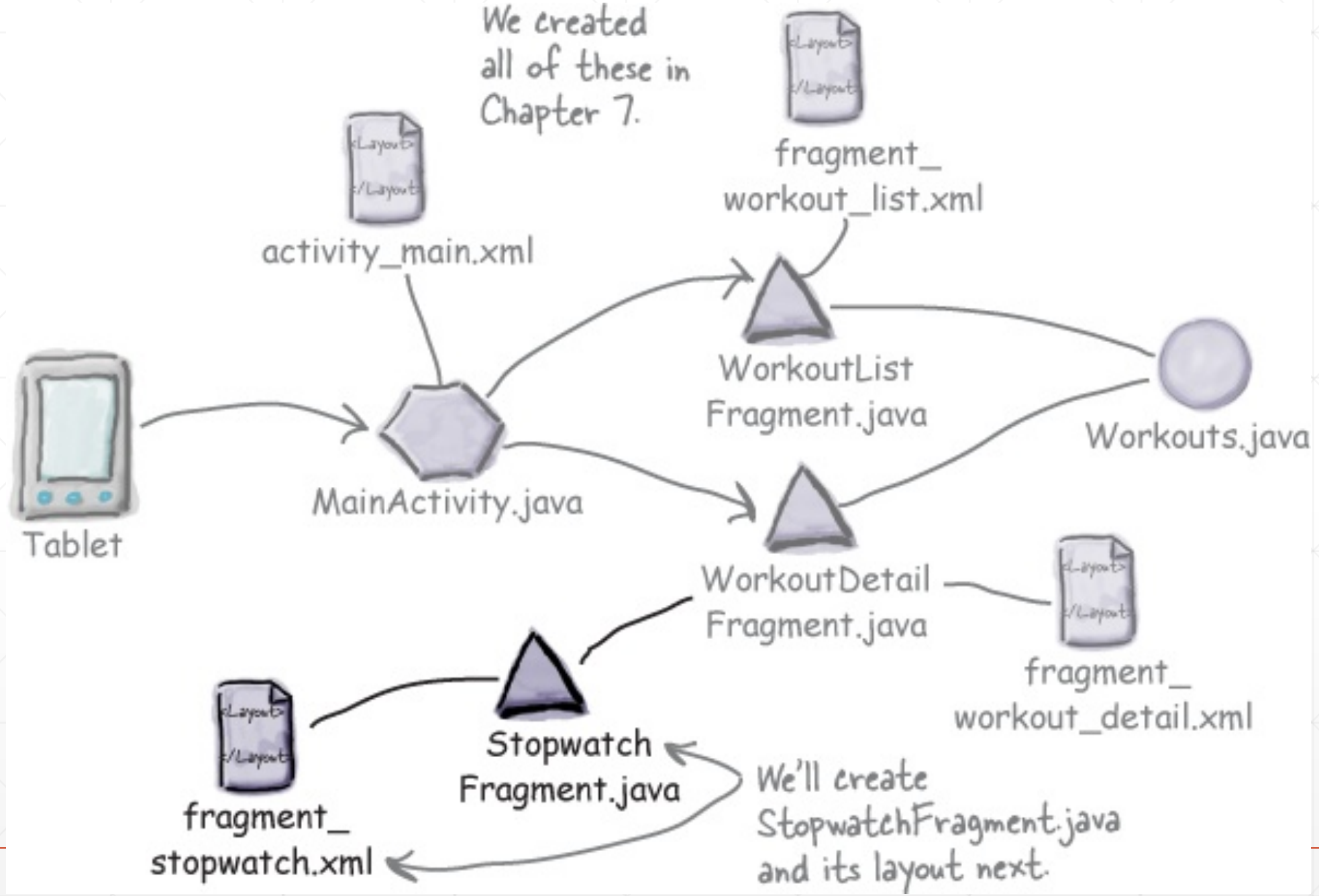
Bullet Points

- When the device configuration changes, the activity and its fragments get destroyed. When the activity is re-created, it replays its fragment transactions in the onCreate() method's call to setContentView().
- The fragment's onCreateView() method runs after the activity has replayed its fragment transactions.

WorkoutListFragment contains a list of workouts.

WorkoutDetailFragment displays details of the workout the user clicks on.

We're going to add a stopwatch fragment to WorkoutDetailFragment.


We created all of these in Chapter 7.
activity_main.xml
fragment_ workout_list.xml
Tablet
MainActivity.java
WorkoutList Fragment.java
Workouts.java
WorkoutDetail Fragment.java
fragment_ workout_detail.xml
Stopwatch Fragment.java
fragment_ stopwatch.xml
We'll create StopwatchFragment.java and its layout next.

Lifecycle Method	Activity?	Fragment?
onAttach()		X
onCreate()	X	X
onCreateView()		X
onActivityCreated()		X
onStart()	X	X
onPause()	X	X
onResume()	X	X
onStop()	X	X
onDestroyView()		X
onRestart()	X	
onDestroy()	X	X
onDetach()		X

```
public class StopwatchActivity extends Activity {
    //Number of seconds displayed on the stopwatch.
    private int seconds = 0;
    //Is the stopwatch running?
    private boolean running;
    private boolean wasRunning;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_stopwatch);
        if (savedInstanceState != null) {
            seconds = savedInstanceState.getInt("seconds");
            running = savedInstanceState.getBoolean("running");
            wasRunning = savedInstanceState.getBoolean("wasRunning");
            if (wasRunning) {
                running = true;
            }
        }
        runTimer();
    }

    @Override
    protected void onPause() {
        super.onPause();
        wasRunning = running;
        running = false;
    }
```

The number of seconds that have passed.

running says whether the stopwatch is running.

wasRunning says whether the stopwatch was running before the stopwatch was paused.

If the activity was destroyed and re-created, restore the state of the variables from the savedInstanceState Bundle.

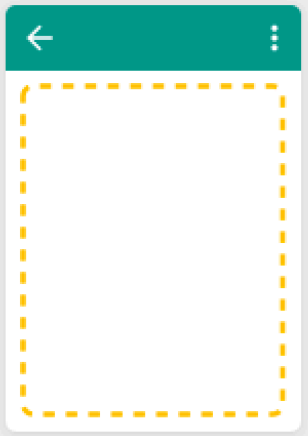
Start the runTimer() method.

Stop the stopwatch if the activity is paused.

New Android Component

Configure Component

Android Studio



Creates a blank fragment that is compatible back to API level 4.

Fragment Name: StopWatchFragment

☑ Create layout XML?

Fragment Layout Name: fragment_stopwatch

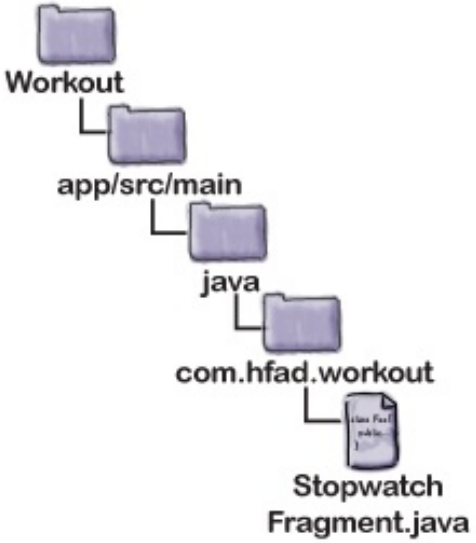
☐ Include fragment factory methods?

☐ Include interface callbacks?

Generate event callbacks for communication with an Activity or other fragments

⚠ Fragment Name must be unique

Cancel | Previous | Next | Finish




```
package com.hfad.workout;

import android.os.Bundle;
import android.os.Handler;
import android.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class StopwatchFragment extends Fragment {
    //Number of seconds displayed on the stopwatch.
    private int seconds = 0;
    //Is the stopwatch running?
    private boolean running;
    private boolean wasRunning;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (savedInstanceState != null) {
            seconds = savedInstanceState.getInt("seconds");
            running = savedInstanceState.getBoolean("running");
            wasRunning = savedInstanceState.getBoolean("wasRunning");
            if (wasRunning) {
                running = true;
            }
        }
    }
```

The number of seconds that have passed.

running says whether the stopwatch is running.
wasRunning says whether the stopwatch was running before the stopwatch was paused.

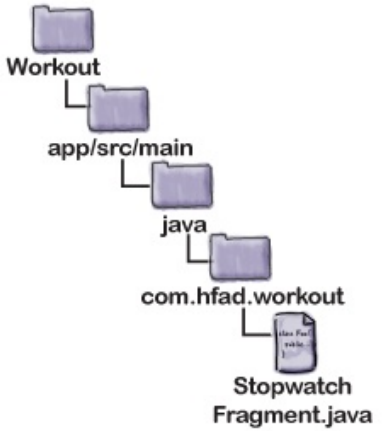
Restore the state of the variables from the savedInstanceState Bundle.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    View layout = inflater.inflate(R.layout.fragment_stopwatch, container, false);
    runTimer(layout);
    return layout;
}
```

Set the fragment's layout, and start the runTimer() method passing in the layout.

```
@Override
public void onPause() {
    super.onPause();
    wasRunning = running;
    running = false;
}
```

If the fragment's paused, record whether the stopwatch was running and stop it.




```
@Override
public void onResume() {
    super.onResume();
    if (wasRunning) {
        running = true;
    }
}
```

If the stopwatch was running before it was paused, set it running again.

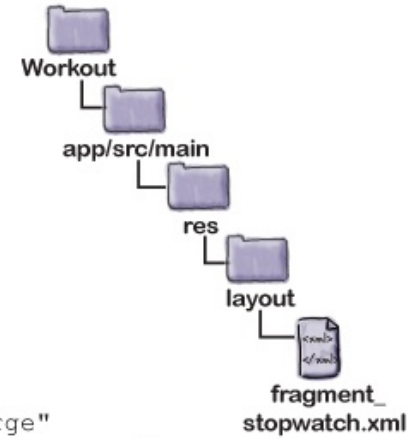
```
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    savedInstanceState.putInt("seconds", seconds);
    savedInstanceState.putBoolean("running", running);
    savedInstanceState.putBoolean("wasRunning", wasRunning);
}
```

Put the values of the variables in the Bundle before the activity is destroyed. These are used when the user turns the device.

```
public void onClickStart(View view) {
    running = true;
}
```

This code needs to run when the user clicks on the Start button.

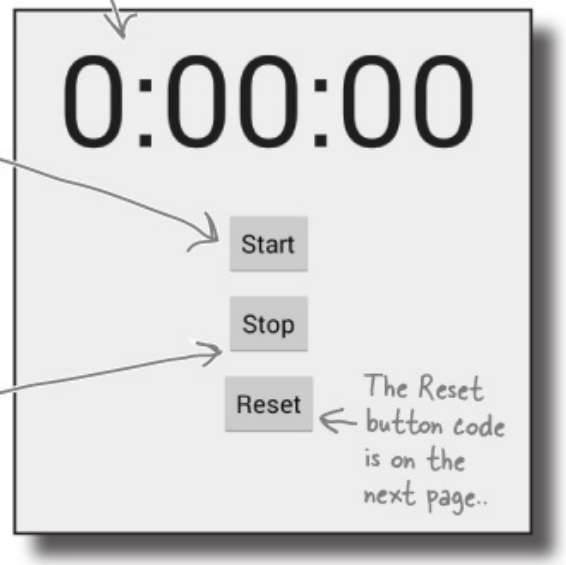
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/time_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="0dp"
        android:text=""
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:textSize="92sp" />

    <Button
        android:id="@+id/start_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/time_view"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp"
        android:onClick="onClickStart"
        android:text="@string/start" />

    <Button
        android:id="@+id/stop_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/start_button"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"
        android:onClick="onClickStop"
        android:text="@string/stop" />
```



The number of hours, minutes, and seconds that have passed.

The Start button

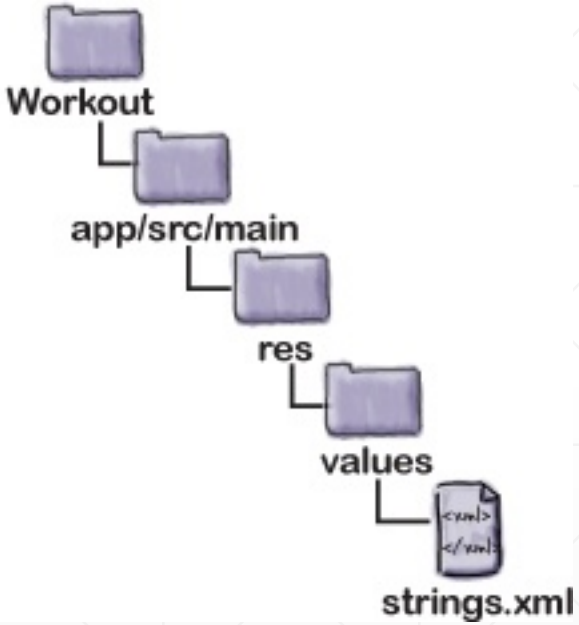
The Stop button



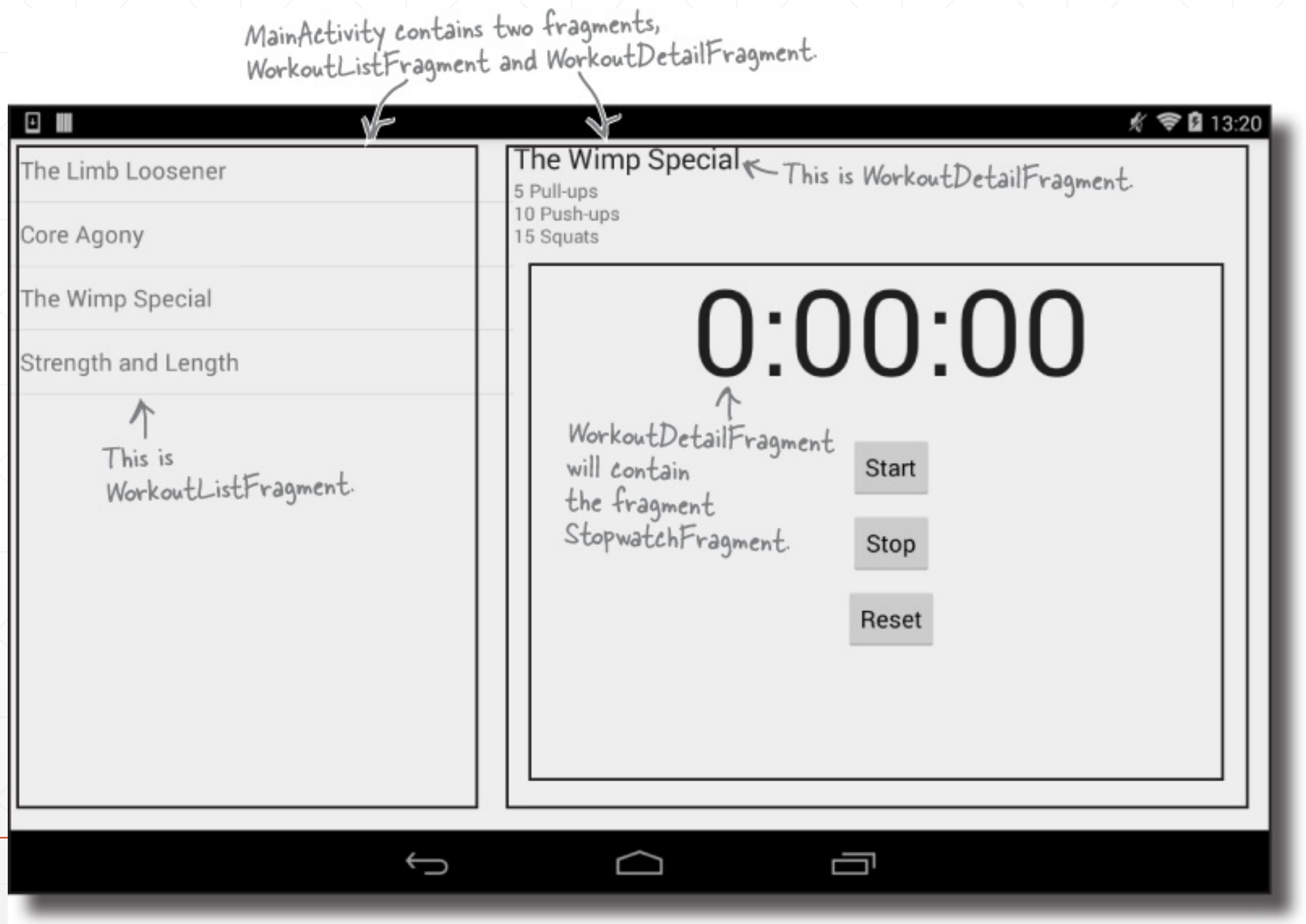
The Reset button code is on the next page..

```
...
    <string name="start">Start</string>
    <string name="stop">Stop</string>
    <string name="reset">Reset</string>
...
```

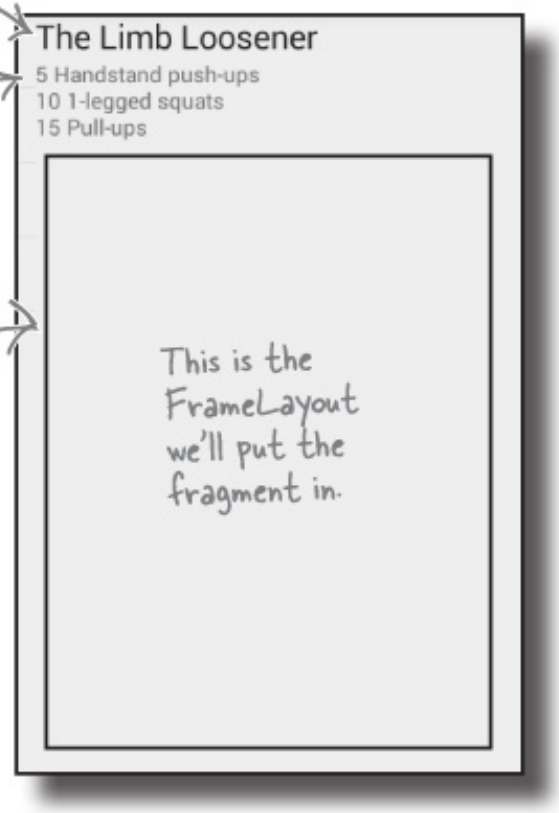
These are the button labels.

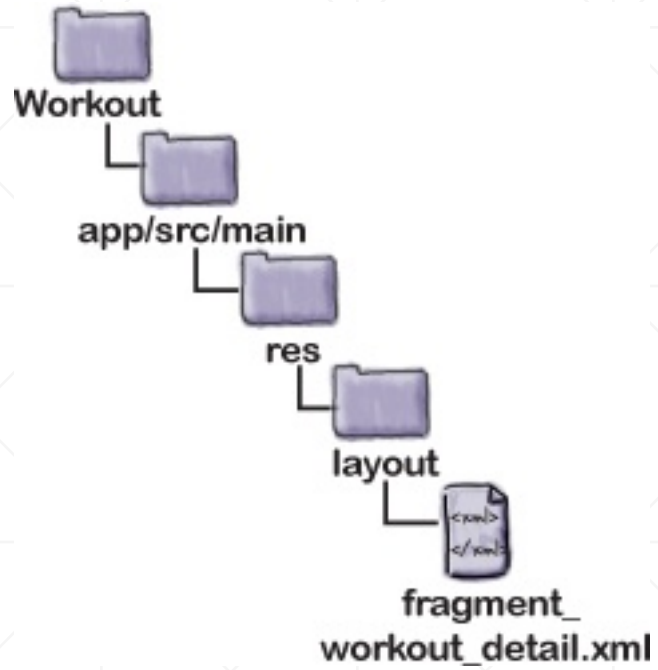


The stopwatch looks the same as it did when it was an activity. Because it's now a fragment, we can reuse it in different activities and fragments.


MainActivity contains two fragments, WorkoutListFragment and WorkoutDetailFragment.
13:20
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
This is WorkoutListFragment.
The Wimp Special
This is WorkoutDetailFragment.
5 Pull-ups
10 Push-ups
15 Squats
0:00:00
WorkoutDetailFragment will contain the fragment StopwatchFragment.
Start
Stop
Reset

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text=""
        android:id="@+id/textTitle" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=""
        android:id="@+id/textDescription" />

    <FrameLayout
        android:id="@+id/stopwatch_container"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```





```
WorkoutDetailFragment details = new WorkoutDetailFragment();
FragmentTransaction ft = getFragmentManager().beginTransaction();
ft.replace(R.id.fragment_container, details);
ft.addToBackStack(null);
ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
ft.commit();
```

Create a new instance of the fragment you wish to display.

Start the fragment transaction.

Replace the fragment and add it to the back stack.

Get the new and old fragments to fade in and out

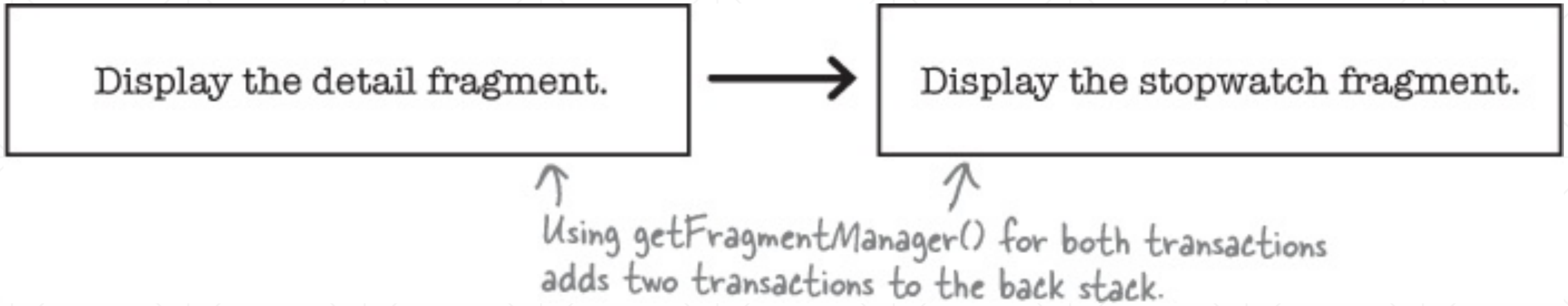
Commit the transaction.

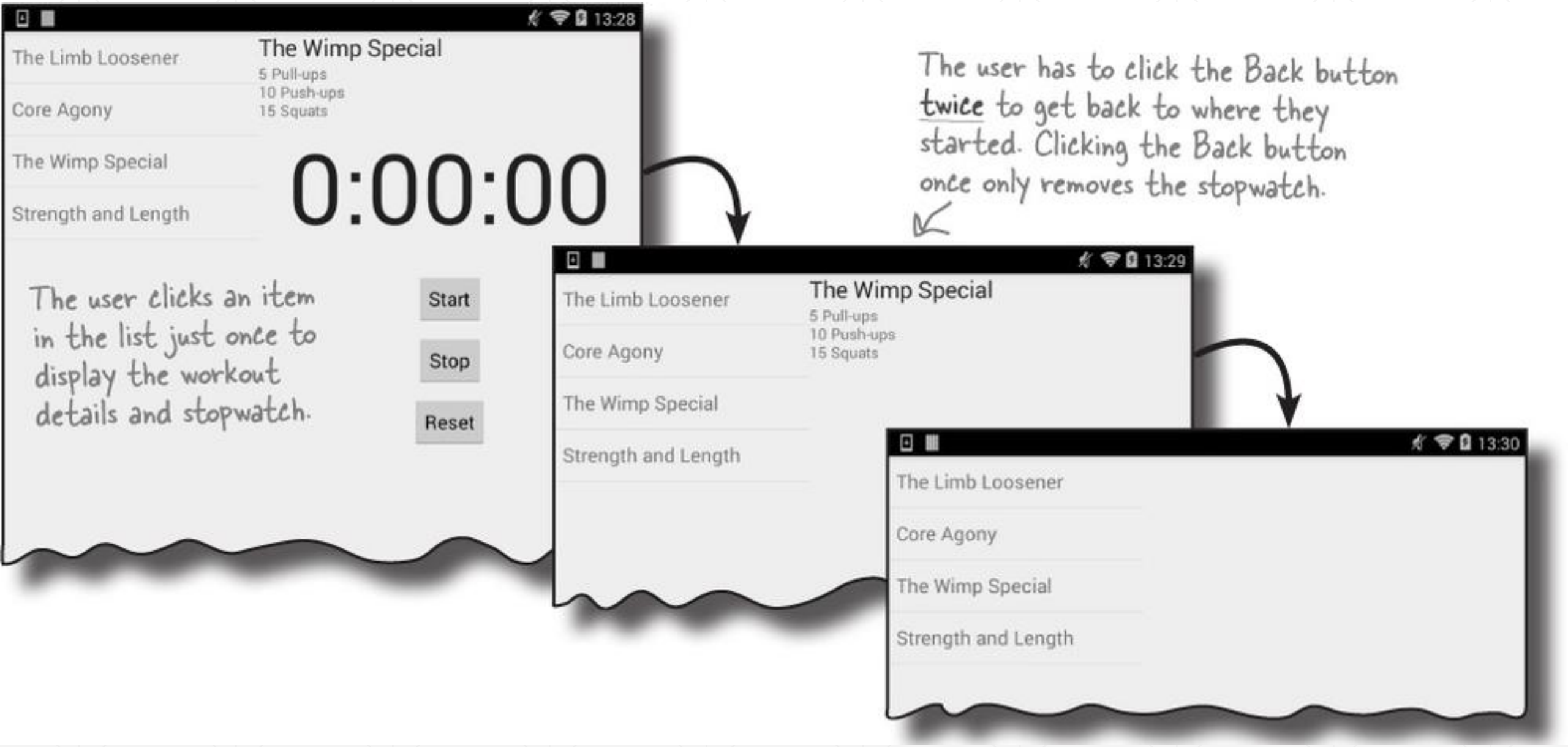
This gets a reference to the activity's fragment manager.

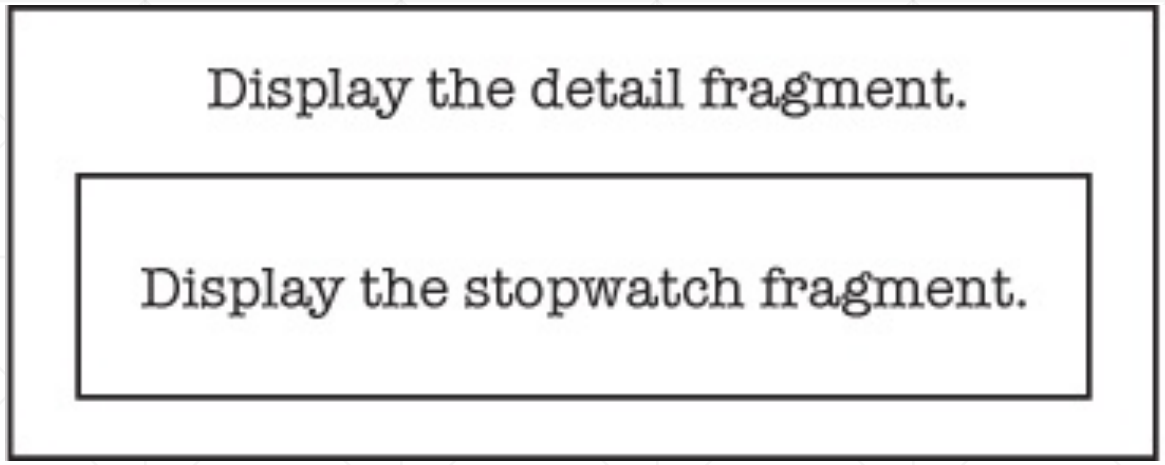
```
FragmentTransaction ft = getFragmentManager().beginTransaction();
```

This gets a reference to the fragment's fragment manager.

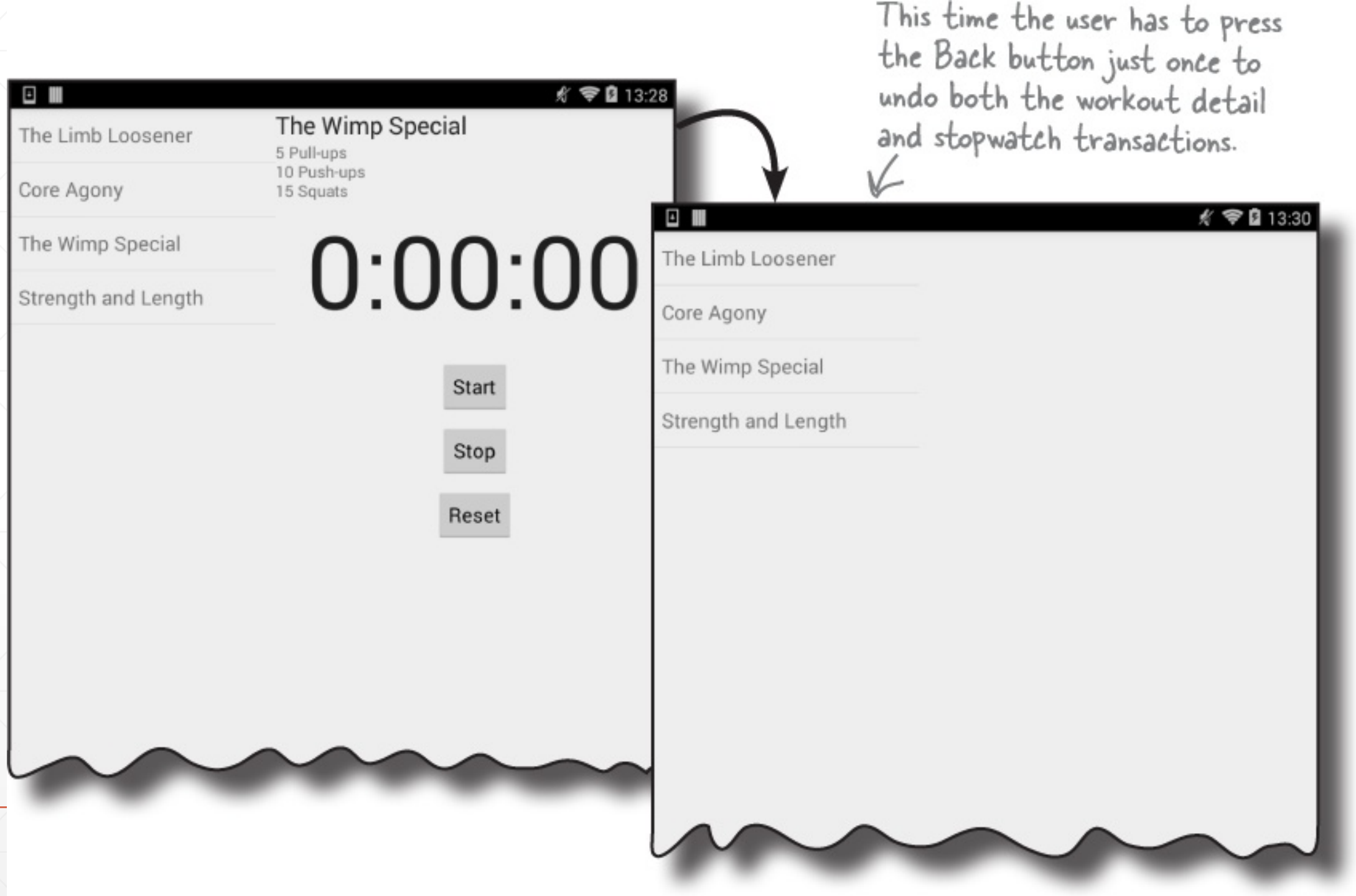
```
FragmentTransaction ft = getChildFragmentManager().beginTransaction();
```

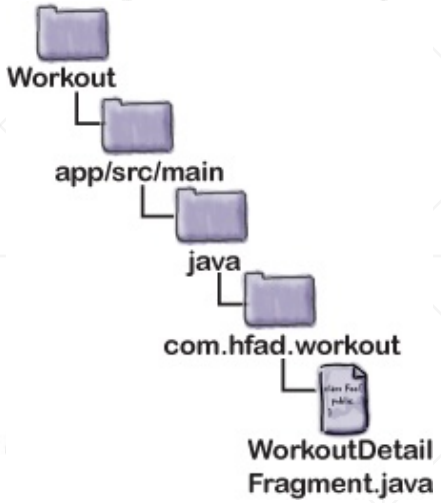

Display the detail fragment.
Display the stopwatch fragment.
Using getFragmentManager() for both transactions
adds two transactions to the back stack.


13:28
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
The Wimp Special
5 Pull-ups
10 Push-ups
15 Squats
0:00:00
Start
Stop
Reset
The user clicks an item in the list just once to display the workout details and stopwatch.
The user has to click the Back button twice to get back to where they started. Clicking the Back button once only removes the stopwatch.
13:29
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
The Wimp Special
5 Pull-ups
10 Push-ups
15 Squats
13:30
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length



Using getChildFragmentManager() to display the stopwatch means that the transaction to display the stopwatch is nested inside the transaction to display the detail fragment.


13:28
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
The Wimp Special
5 Pull-ups
10 Push-ups
15 Squats
0:00:00
Start
Stop
Reset
This time the user has to press the Back button just once to undo both the workout detail and stopwatch transactions.
13:30
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length




```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container
                         Bundle savedInstanceState) {
    if (savedInstanceState != null) {
        workoutId = savedInstanceState.getLong("workoutId");
    }
    FragmentTransaction ft = getChildFragmentManager().beginTransaction();
    StopwatchFragment stopwatchFragment = new StopwatchFragment();
    ft.replace(R.id.stopwatch_container, stopwatchFragment);
    ft.addToBackStack(null);
    ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
    ft.commit();
    return inflater.inflate(R.layout.detail, container, false);
}
```

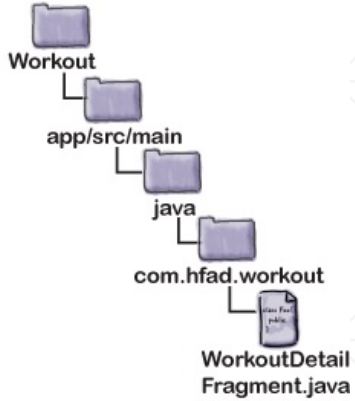
Start the transaction.

Add the transaction to the back stack.

Replace the fragment in the frame layout.

Set the transition animation style.

Commit the transaction.

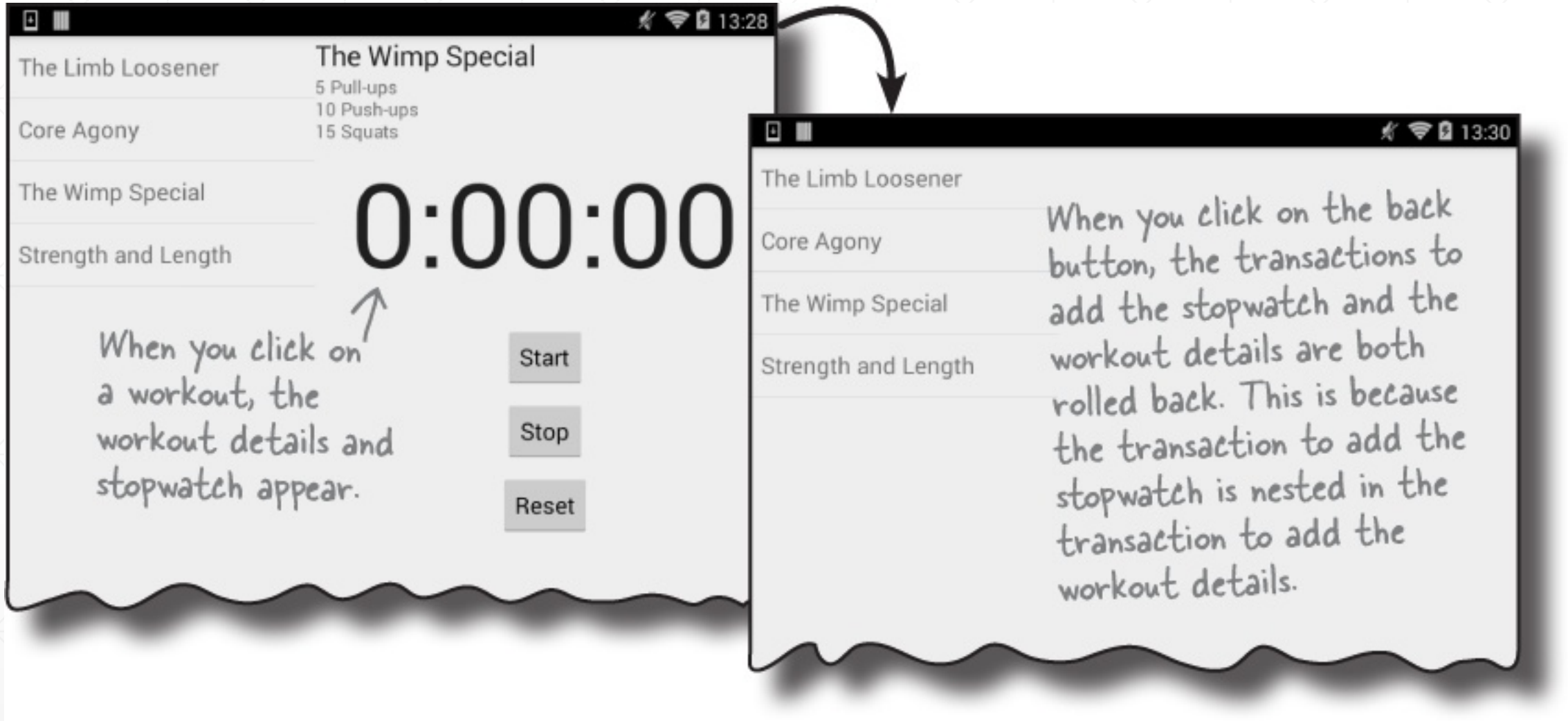



```
package com.hfad.workout;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import android.app.FragmentTransaction;

public class WorkoutDetailFragment extends Fragment {
    private long workoutId;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        if (savedInstanceState != null) {
            workoutId = savedInstanceState.getLong("workoutId");
        }
        FragmentTransaction ft = getChildFragmentManager().beginTransaction();
        StopwatchFragment stopwatchFragment = new StopwatchFragment();
        ft.replace(R.id.stopwatch_container, stopwatchFragment);
        ft.addToBackStack(null);
        ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
        ft.commit();
        return inflater.inflate(R.layout.fragment_workout_detail, container, false);
    }

    @Override
    public void onStart() {
        super.onStart();
        View view = getView();
        if (view != null) {
            TextView title = (TextView) view.findViewById(R.id.textTitle);
            Workout workout = Workout.workouts[(int) workoutId];
            title.setText(workout.getName());
            TextView description = (TextView) view.findViewById(R.id.textDescription);
            description.setText(workout.getDescription());
        }
    }

    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        savedInstanceState.putLong("workoutId", workoutId);
    }

    public void setWorkout(long id) {
        this.workoutId = id;
    }
}
```

We're using the FragmentTransaction class, so we're importing it.

Use a fragment transaction to add the stopwatch fragment to the frame layout.

These methods don't need to change.


13:28
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
The Wimp Special
5 Pull-ups
10 Push-ups
15 Squats
0:00:00
Start
Stop
Reset
When you click on a workout, the workout details and stopwatch appear.
13:30
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
When you click on the back button, the transactions to add the stopwatch and the workout details are both rolled back. This is because the transaction to add the stopwatch is nested in the transaction to add the workout details.

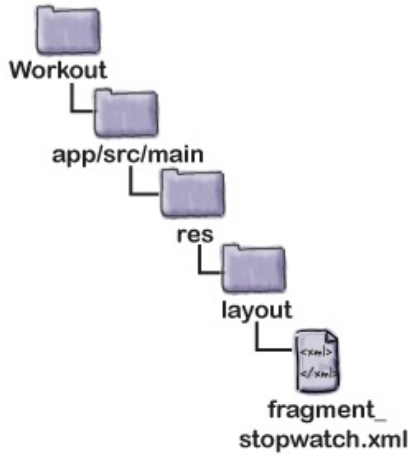

13:28
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
The Wimp Special
5 Pull-ups
10 Push-ups
15 Squats
0:00:00
Start
Stop
Reset
Click on one of the buttons in the stopwatch...
...and the app crashes.
Unfortunately, Workout has stopped.
OK

Yikes.

```
01-24 17:37:00.326    2400-2400/com.hfad.fraghack E/AndroidRuntime: FATAL EXCEPTION: main
    Process: com.hfad.fraghack, PID: 2400
    java.lang.IllegalStateException: Could not find a method onClickStart(View) in the activity
    class com.hfad.fraghack.MainActivity for onClick handler on view class android.widget.
    Button with id 'start_button'
            at android.view.View$1.onClick(View.java:3994)
            at android.view.View.performClick(View.java:4756)
            at android.view.View$PerformClick.run(View.java:19749)
            at android.os.Handler.handleCallback(Handler.java:739)
            at android.os.Handler.dispatchMessage(Handler.java:95)
            at android.os.Looper.loop(Looper.java:135)
            at android.app.ActivityThread.main(ActivityThread.java:5221)
            at java.lang.reflect.Method.invoke(Native Method)
            at java.lang.reflect.Method.invoke(Method.java:372)
            at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:899)
            at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:694)
     Caused by: java.lang.NoSuchMethodException: onClickStart [class android.view.View]
            at java.lang.Class.getMethod(Class.java:664)
            at java.lang.Class.getMethod(Class.java:643)
            at android.view.View$1.onClick(View.java:3987)
            at android.view.View.performClick(View.java:4756)
            at android.view.View$PerformClick.run(View.java:19749)
            at android.os.Handler.handleCallback(Handler.java:739)
            at android.os.Handler.dispatchMessage(Handler.java:95)
            at android.os.Looper.loop(Looper.java:135)
            at android.app.ActivityThread.main(ActivityThread.java:5221)
            at java.lang.reflect.Method.invoke(Native Method)
            at java.lang.reflect.Method.invoke(Method.java:372)
            at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:899)
            at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:694)
```

We're using the same layout for the stopwatch now that it's a fragment as we did when it was an activity.

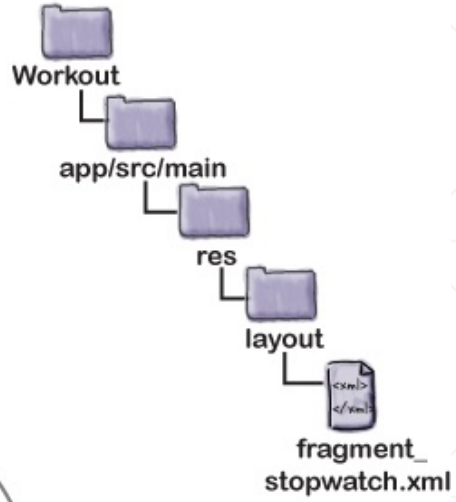
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    <Button
        android:id="@+id/start_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/time_view"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp"
        android:onClick="onClickStart"
        android:text="@string/start" />

    <Button
        android:id="@+id/stop_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/start_button"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"
        android:onClick="onClickStop"
        android:text="@string/stop" />

    <Button
        android:id="@+id/reset_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/stop_button"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"
        android:onClick="onClickReset"
        android:text="@string/reset" />
</RelativeLayout>
```



We're using the android:onClick attributes in the layout to say which methods should be called when each button is clicked.


Whenever I see android:onClick, I assume it's all about me. My methods run, not the fragment's.
Activity

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    <Button
        android:id="@+id/start_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/time_view"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp"
        android:onClick="onClickStart"
        android:text="@string/start" />

    <Button
        android:id="@+id/stop_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/start_button"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"
        android:onClick="onClickStop"
        android:text="@string/stop" />

    <Button
        android:id="@+id/reset_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/stop_button"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"
        android:onClick="onClickReset"
        android:text="@string/reset" />
</RelativeLayout>
```



Remove the onClick attributes for each of the buttons in the stopwatch.

This turns the fragment into an OnClickListener.

```
public class StopwatchFragment extends Fragment implements View.OnClickListener {
    ...
}
```

```
@Override
public void onClick(View v) {
    ...
}
```

You must override the onClick() method in your fragment code.

```
@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.start_button:
            onClickStart(v);
            break;
        case R.id.stop_button:
            onClickStop(v);
            break;
        case R.id.reset_button:
            onClickReset(v);
            break;
    }
}
```

This is the View the user clicked on.

Check which View was clicked.

If the Start button was clicked, call the onClickStart() method.

If the Stop button was clicked, call the onClickStop() method.

If the Reset button was clicked, call the onClickReset() method.

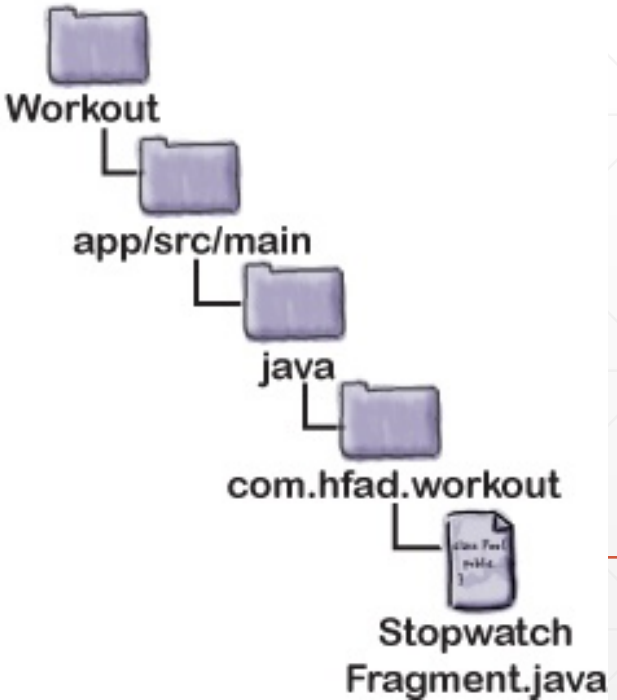
Get a reference to the button.

```
Button startButton = (Button) layout.findViewById(R.id.start_button);
startButton.setOnClickListener(this);
```

Attach the listener to the button.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    View layout = inflater.inflate(R.layout.stopwatch, container, false);
    runTimer(layout);
    Button startButton = (Button)layout.findViewById(R.id.start_button);
    startButton.setOnClickListener(this);
    Button stopButton = (Button)layout.findViewById(R.id.stop_button);
    stopButton.setOnClickListener(this);
    Button resetButton = (Button)layout.findViewById(R.id.reset_button);
    resetButton.setOnClickListener(this);
    return layout;
}
```

This attaches the listener to each of the buttons.



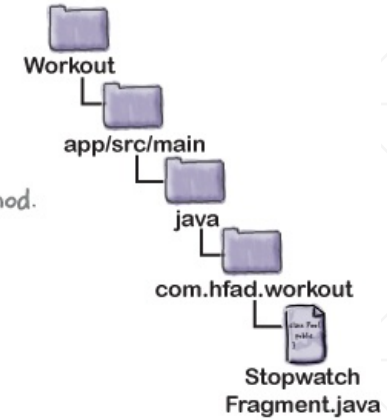
```
package com.hfad.workout;
```

We're using the Button class, so we'll import it.

```
...
import android.widget.Button;
```

The fragment needs to implement the View.OnClickListener interface.

```
public class StopwatchFragment extends Fragment implements View.OnClickListener {
    //Number of seconds displayed on the stopwatch.
    private int seconds = 0;
    //Is the stopwatch running?
    private boolean running;
    private boolean wasRunning;
```



We're not changing the onCreate() method.

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (savedInstanceState != null) {
            seconds = savedInstanceState.getInt("seconds");
            running = savedInstanceState.getBoolean("running");
            wasRunning = savedInstanceState.getBoolean("wasRunning");
            if (wasRunning) {
                running = true;
            }
        }
    }
```

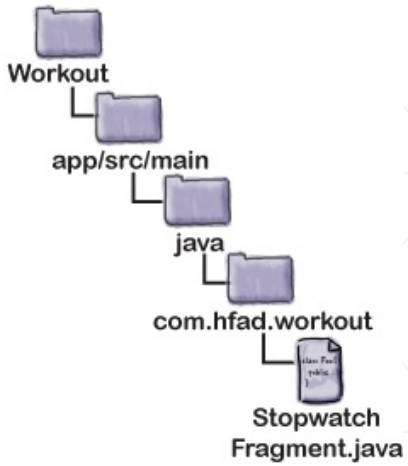
Update the onCreateView() method to attach the listener to the buttons.

```
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View layout = inflater.inflate(R.layout.stopwatch, container, false);
        runTimer(layout);
        Button startButton = (Button) layout.findViewById(R.id.start_button);
        startButton.setOnClickListener(this);
        Button stopButton = (Button) layout.findViewById(R.id.stop_button);
        stopButton.setOnClickListener(this);
        Button resetButton = (Button) layout.findViewById(R.id.reset_button);
        resetButton.setOnClickListener(this);
        return layout;
    }
```

```
@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.start_button:
            onClickStart(v);
            break;
        case R.id.stop_button:
            onClickStop(v);
            break;
        case R.id.reset_button:
            onClickReset(v);
            break;
    }
}

...

public void onClickStart(View view) {
    running = true;
}

public void onClickStop(View view) {
    running = false;
}

public void onClickReset(View view) {
    running = false;
    seconds = 0;
}
...
}
```

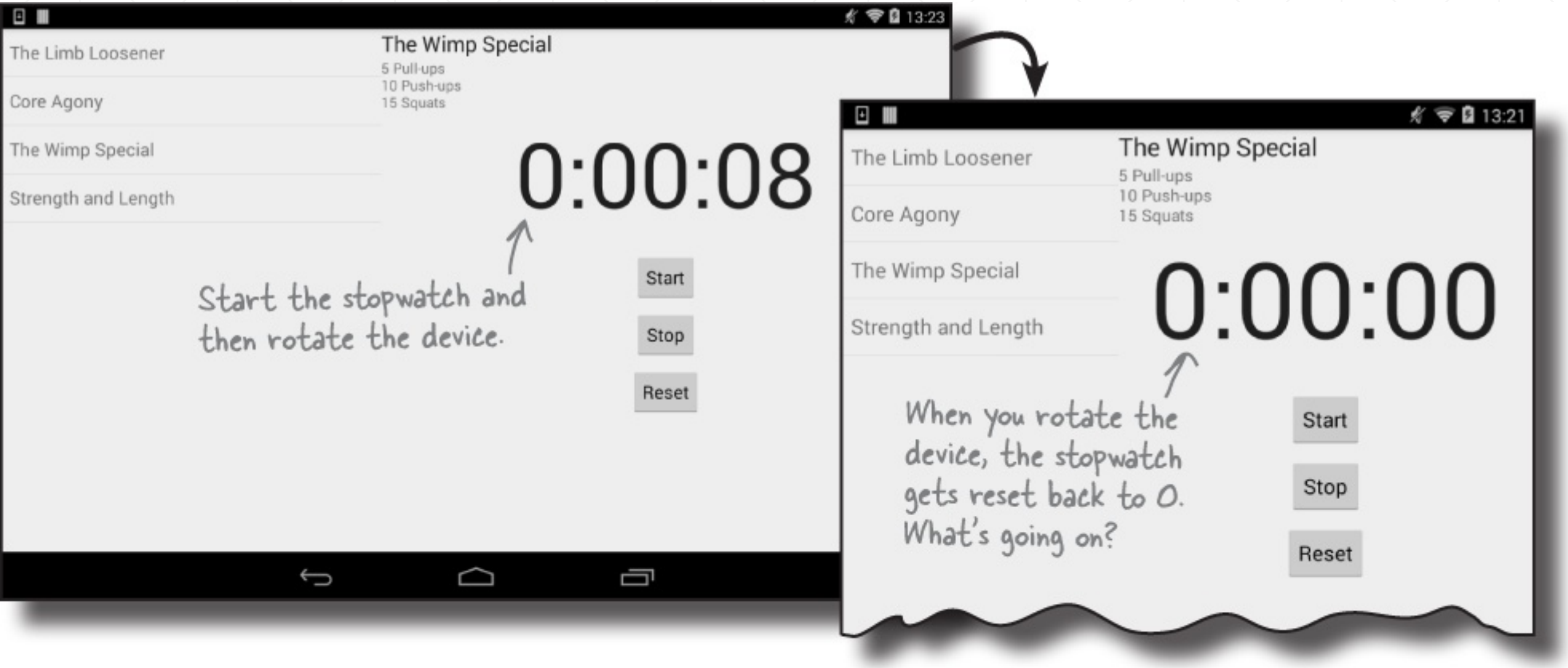
As we're implementing the OnClickListener interface, we need to override the onClick() method.

Call the appropriate method in the fragment for the button that was clicked.

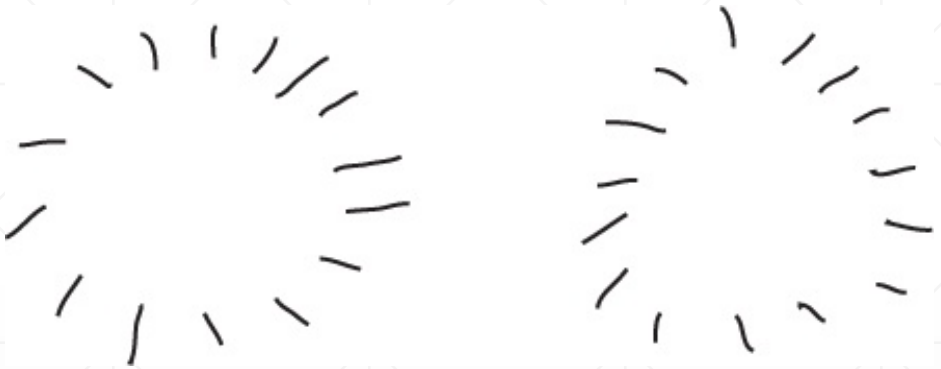


These are the same methods that we had before. They'll get called when the buttons are clicked.


13:23
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
The Wimp Special
5 Pull-ups
10 Push-ups
15 Squats
0:00:08
When you click on the Start button, the stopwatch starts. The Stop and Reset buttons work as you'd expect too.
Start
Stop
Reset


13:23
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
The Wimp Special
5 Pull-ups
10 Push-ups
15 Squats
0:00:08
Start
Stop
Reset
Start the stopwatch and then rotate the device.
13:21
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
The Wimp Special
5 Pull-ups
10 Push-ups
15 Squats
0:00:00
Start
Stop
Reset
When you rotate the device, the stopwatch gets reset back to 0. What's going on?


Activity
Fragment



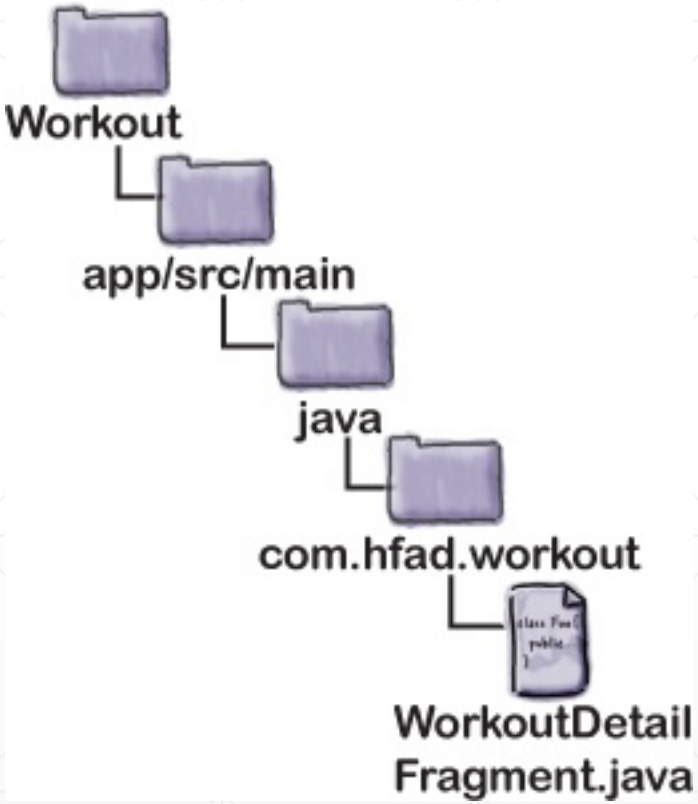

onCreate()
Activity

onCreateView() method in the fragment runs after the activity has replayed all of its fragment transactions.

```
...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        if (savedInstanceState != null) {
            workoutId = savedInstanceState.getLong("workoutId");
        }
        FragmentTransaction ft = getChildFragmentManager().beginTransaction();
        StopwatchFragment stopwatchFragment = new StopwatchFragment();
        ft.replace(R.id.stopwatch_container, stopwatchFragment);
        ft.addToBackStack(null);
        ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
        ft.commit();
        return inflater.inflate(R.layout.fragment_workout_detail, container, false);
    }
...
```

This runs if WorkoutDetailFragment has saved its state prior to being destroyed.

This replaces the stopwatch fragment with a brand-new one.


Workout
app/src/main
java
com.hfad.workout
WorkoutDetail
Fragment.java

```
package com.hfad.workout;

...

public class WorkoutDetailFragment extends Fragment {
    private long workoutId;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        if (savedInstanceState != null) {
            workoutId = savedInstanceState.getLong("workoutId");
        } else {
            FragmentTransaction ft = getChildFragmentManager().beginTransaction();
            StopwatchFragment stopwatchFragment = new StopwatchFragment();
            ft.replace(R.id.stopwatch_container, stopwatchFragment);
            ft.addToBackStack(null);
            ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
            ft.commit();
        }
        return inflater.inflate(R.layout.fragment_workout_detail, container, false);
    }

    @Override
    public void onStart() {
        super.onStart();
        View view = getView();
        if (view != null) {
            TextView title = (TextView) view.findViewById(R.id.textTitle);
            Workout workout = Workout.workouts[(int) workoutId];
            title.setText(workout.getName());
            TextView description = (TextView) view.findViewById(R.id.textDescription);
            description.setText(workout.getDescription());
        }
    }

    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        savedInstanceState.putLong("workoutId", workoutId);
    }

    public void setWorkout(long id) {
        this.workoutId = id;
    }
}
```

The only change we need to make is to put the transaction in an else statement. The transaction will only run if savedInstanceState is null.


13:22
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
The Wimp Special
5 Pull-ups
10 Push-ups
15 Squats
0:00:05
Start
Stop
Reset
Start the stopwatch and then rotate the device.
13:23
The Limb Loosener
Core Agony
The Wimp Special
Strength and Length
The Wimp Special
5 Pull-ups
10 Push-ups
15 Squats
0:00:08
Start
Stop
Reset
When you rotate the device, the stopwatch keeps on going. Its previous state is maintained.

Bullet Points

- Fragments can contain other fragments.
- If you're nesting a fragment in another fragment, you need to add the nested fragment programmatically in Java code.
- When you perform transactions on a nested fragment, use getChildFragmentManager() to create the transaction.
- If you use the android:onClick attribute in a fragment, Android will look for a method of that name in the fragment's parent activity.
- Instead of using the android:onClick attribute in a fragment, make the fragment implement the View.OnClickListener interface and implement its onClick() method..

Bullet Points

- When the device configuration changes, the activity and its fragments get destroyed. When the activity is re-created, it replays its fragment transactions in the onCreate() method's call to setContentView().
- The fragment's onCreateView() method runs after the activity has replayed its fragment transactions.