Responsible Design

for Android™

Part 1: Dancing with the SDK

J. B. Rainsberger

Responsible Design for Android

Part 1: Dancing with the SDK

J. B. Rainsberger

This book is for sale at http://leanpub.com/ResponsibleDesignAndroid-Part1

This version was published on 2013-01-17

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process.

Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

To learn more about Lean Publishing, go to http://leanpub.com/manifesto.

To learn more about Leanpub, go to http://leanpub.com.

Tweet This Book!

Please help J. B. Rainsberger by spreading the word about this book on Twitter!

The suggested hashtag for this book is #rdandroid.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search/#rdandroid

Contents

1 A Tiny Slice

I chose a tiny slice of a feature to start: displaying **You have *n* transactions** on the app's opening screen. This feature carves a thin, but deep slice through the entire system, and allows me to focus on displaying dynamic information on the screen. It also provides a relatively simple, quick win.

The test list

I can only think of two tests: `1` transaction and any other number, with the singular/plural of "transaction" the only difference. I think I'll ignore this minor difference for the moment, leaving a "test list" with a single test on it.

Presenter-First Design

I expect to use the typical presenter-first (or controller-first[1]) approach to building any feature for this app. This approach tends to fit perfectly for any GUI-based application. If you're not familiar with Presenter-First Design, then I recommend you read a few articles on the topic.[2] In the meantime, I'll summarise the key points.

- Test-drive the Presenter by stubbing/mocking the Model and View.
- While implementing the Presenter, design the contracts of the Model and View.
- Keep the Presenter thin; move behavior into the Model or View.
- Specify the Presenter's behavior in terms of a purely abstract UI.
- The Presenter cannot refer to its context, meaning its caller.

When I design presenter-first, code tends to flow from the Presenter into the Model or View.

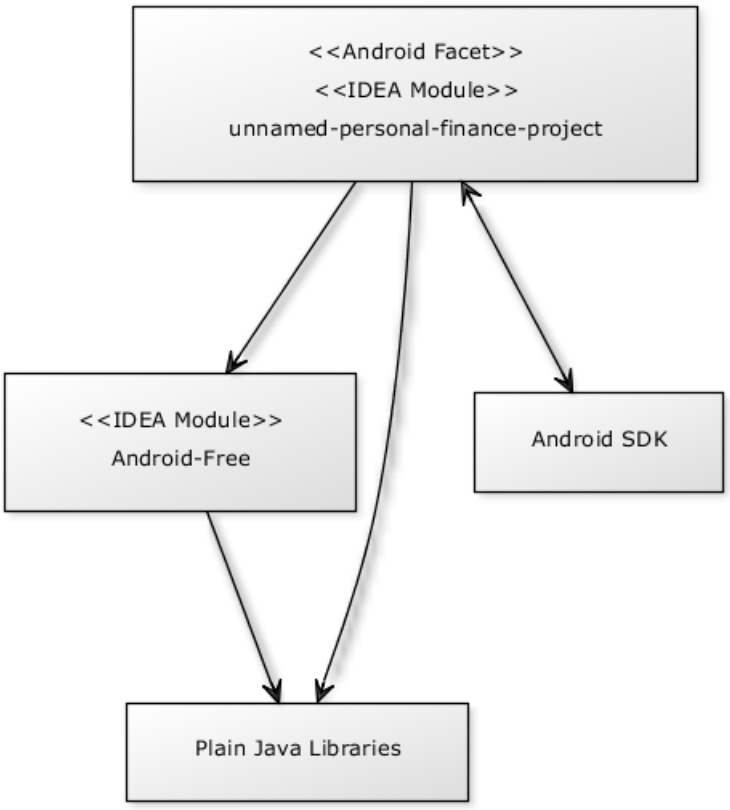
View or `View`?

Android has a class named `View`, which you might easily confuse with the generic concept of a View. This makes it tricky for both of us. Whenever you see "View", you can assume that I mean the generic concept. If I want to refer to the class `View`, then I will fall all over myself to clarify that. I hope that helps.

[1] If you feel more comfortable referring the Presenter as the "Controller", then please do. I make no significant distinction between the two ideas.
[2] http://www.atomicobject.com/pages/Presenter+First

Setting the project up for success

Before writing a single line of code, I create two IDE modules: an Android module and an Android-Free module. I can save myself hours of grief by configuring these modules to avoid cyclic dependencies by following one simple rule: the Android module may refer to the Android-Free module, but not the reverse.



I always look for code in the Android module that can move to the Android-Free module

If you want to see how I did this in IDEA, see this appendix.

If you've never worked with Android before, or simply don't have the Android tools installed, then you'll need to do that, too. I intend to replace this note with another appendix, but for now, follow these basic instructions.

1. Download the SDK tools[a]
2. Follow the instructions here[b] to install the right packages (click on tools and 4.0.3)

I've used v4.0.3 of the Android Tools for this project, and I hope that's not horribly

out of date by the time you read this.

[a] http://developer.android.com/sdk/index.html#ExistingIDE
[b] http://developer.android.com/sdk/installing/adding-packages.htm

Ignoring Android for the moment

Now that I've created this warm, dry place where I can write code without worrying about Android, I choose to start here. I will build a Model/View/Presenter (MVP) triad triggered by launching the app. Following the Presenter-First Design approach, I start by implementing the Presenter in order to discover the contracts of the Model and View. I choose the conventional name `render()` for the Presenter method that responds to visiting the screen, as opposed to pressing a button or tapping a widget. When rendering the screen, then, the View should display "You have *n* transactions", so I add a method to the contract of the View. Stepping into the role of the Presenter, I need something to give me *n*, and that sounds like a Model responsibility, so I add a method to the contract of the Model to tell me how many transactions it has. This gives me enough to write the first test.

The first code sample: Snapshot 10

The following code is from Snapshot 10 of the code base. You can find the code base at http://github.com/jbrains/TrackEveryPenny[3] and this particular version is tagged `snapshot_10`. I will introduce the next code sample more simply with the heading "Snapshot 20".

Snapshot 10: the first test

```
1  package ca.jbrains.upfp.mvp;
2
3  import org.jmock.*;
4  import org.jmock.integration.junit4.JMock;
5  import org.junit.Test;
6  import org.junit.runner.RunWith;
7
8  @RunWith(JMock.class)
9  public class RenderYouHaveNTransactionsOnBrowseTransactionsScreenTest {
10   private Mockery mockery = new Mockery();
11
12   @Test
13   public void zero() throws Exception {
```

[3] http://github.com/jbrains/TrackEveryPenny

```
14      final BrowseTransactionsModel model
15          = mockery.mock(BrowseTransactionsModel.class);
16      final BrowseTransactionsView view
17          = mockery.mock(BrowseTransactionsView.class);
18      final BrowseTransactionsPresenter presenter
19          = new BrowseTransactionsPresenter(model, view);
20
21      mockery.checking(
22          new Expectations() {{
23            allowing(model).countTransactions();
24            will(returnValue(0));
25
26            oneOf(view).setNumberOfTransactions(0);
27          }});
28
29      presenter.render();
30    }
31  }
```

The implementations offer no surprises.

Snapshot 10: give the Model to the View

```
1   package ca.jbrains.upfp.mvp;
2
3   public class BrowseTransactionsPresenter {
4     private final BrowseTransactionsModel model;
5     private final BrowseTransactionsView view;
6
7     public BrowseTransactionsPresenter(
8         BrowseTransactionsModel model,
9         BrowseTransactionsView view
10    ) {
11      this.model = model;
12      this.view = view;
13    }
14
15    public void render() {
16      view.setNumberOfTransactions(model.countTransactions());
17    }
18  }
```

```
package ca.jbrains.upfp.mvp;

public interface BrowseTransactionsModel {
  int countTransactions();
}
```

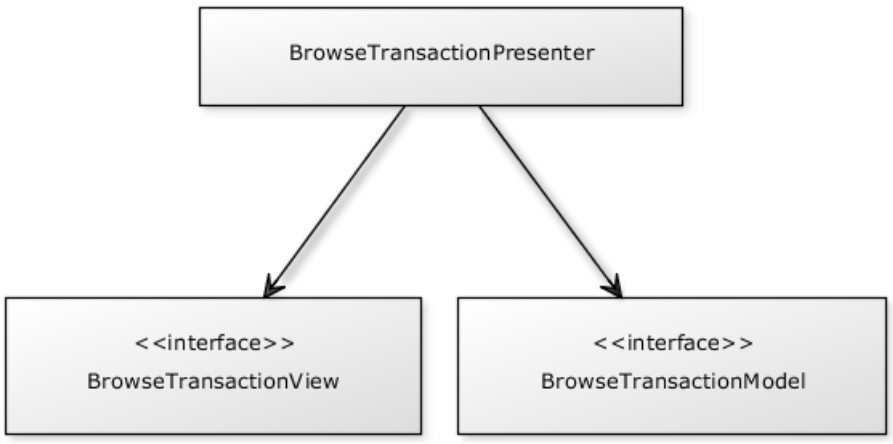
```
package ca.jbrains.upfp.mvp;

public interface BrowseTransactionsView {
  void setNumberOfTransactions(int numberOfTransactions);
}
```

I have named this micro-feature "Browse Transactions", with an eye towards displaying not only the number of transactions, but also a summary of transactions, probably filtered by the date. This represents slightly speculative design, but given how easily I can rename types, I don't feel worried about the name.

In retrospect, however, I don't like including the words "Browse Transaction Screen" in the test. At the time I wrote the test, I expected to display "You have *n* transactions" on that screen, but neither the test nor the production code depend at all on this decision, **and indeed, that is the point!** Even though I've successfully structured the code to avoid depending on its context, I've let the context sneak in to the name, something I will fix before putting this feature away.

At first, I thought I'd ignored the poor grammar of "You have 1 transactions" in this test, but I later realised that formatting "1 transaction" instead of "1 transactions" belongs to the View, and I haven't implemented it yet. I note this and add it to the test list for implementing the View, which I will do next.

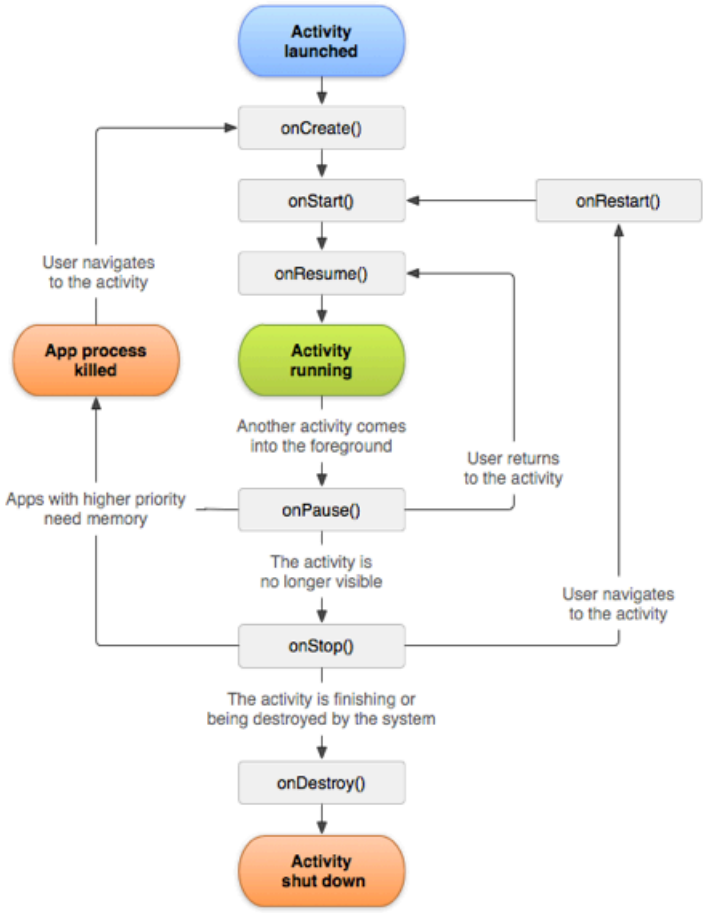
A snapshot of the design



So far, so good

Exploring the Android Activity lifecycle

As part of the hacking I did back when I was finding my bearings, I learned a little about the lifecycle of an Activity, which appears to represent a screen. I dug into the Android development documentation and found a handy diagram that summarises the lifecycle.



Smells like EJB to me

This diagram tells me that `onResume()` should invoke `presenter.render()`. If this causes any problems, then I'll walk "up the stack" on the lifecycle diagram, first to `onStart()`, then to `onCreate()`.

Do I need an error case for `presenter.render()`? I don't think so, and I hope not. Right now, the Presenter will throw any `RuntimeExceptions` up the stack and probably crash the app, so I need to develop a strategy for handling exceptions inside the Activity. I add this to my backlog.

What's done

- Snapshot 10: Presenter handles displaying 0 transactions.

What's left to do

- Decide how the Activity will handle Presenter throwing exceptions to it. Normal operation or should Presenter not throw exceptions?
- Implement Android View.
  - Check singular/plural text when displaying "1 transaction"/"3 transactions"
- Fake out Model in Android Activity for now, since we can only read it.
- Do a little manual inspection to make sure we haven't messed anything up.

2 Implementing the View

I want to implement the View, meaning interface `BrowseTransactionsView`, to display text on an Android screen. I decide to have the Activity implement the View directly, since whatever implementation I choose will have to depend on the Android SDK, and the Activity already does that. This gives me on opportunity to learn some Robolectric[1] basics.

Presenter implements the right View behavior

I started with a simple happy path test, checking that the Activity displays the correct number of transactions.

Snapshot 20: the new test

```
1  package ca.jbrains.upfp.view.android.test;
2
3  import android.widget.TextView;
4  import ca.jbrains.upfp.*;
5  import com.xtremelabs.robolectric.RobolectricTestRunner;
6  import org.junit.Test;
7  import org.junit.runner.RunWith;
8
9  import static org.junit.Assert.assertEquals;
10
11 @RunWith(RobolectricTestRunner.class)
12 public class DisplayNumberOfTransactionsTest {
13   @Test
14   public void happyPath() throws Exception {
15     final BrowseTransactionsActivity
16         browseTransactionsActivity
17         = new BrowseTransactionsActivity();
18
19     browseTransactionsActivity.onCreate(null);
20
21     browseTransactionsActivity.displayNumberOfTransactions(
22         12);
23
24     final TextView transactionsCountView
25         = (TextView) browseTransactionsActivity
```

[1] A library for test-driving Android SDK client code, which you can find at http://pivotal.github.com/robolectric/index.html

```
26          .findViewById(R.id.transactionsCount);
27
28      assertEquals(
29          "12", transactionsCountView.getText().toString()
30      );
31    }
32  }
```

Now I wish I had checked the entire text "You have 12 transactions", as it would have reminded me to check the case "You have 1 transaction". At the time, I assumed that I would turn this text into an Android resource, but I still haven't done that as of the moment I wrote this sentence. Since I don't know exactly how to do this right now, I add this task to my list.

Snapshot 20: a naive implementation

```
diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
index 985b08d..50fa839 100644
--- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
+++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
@@ -246,4 +246,15 @@ public class BrowseTransactionsActivity extends Activi\
ty {
    public TextView categoryView() {
      return (TextView) findViewById(R.id.category);
    }
+
+  public void displayNumberOfTransactions(
+      int transactionCount
+  ) {
+    final TextView transactionsCountView
+        = (TextView) findViewById(R.id.transactionsCount);
+
+    transactionsCountView.setText(
+        String.format(
+            "%1$d", transactionCount));
+  }
 }
```

I had hacked around before starting this book in order to get a basic feel for the Android SDK, but I made the mistake of refactoring away from my spike, rather than throwing it away and starting again. It will take me a couple of days to fix up all the code samples to make it look like I'd started from scratch, rather than refactor away from the spike, and I plan to do that *after* I've completed the first draft of the book.

Rather than flood you with a very long code sample, click here to view the layout file[a]. Look for the field `transactionsCount`, which `DisplayNumberOfTransactionsTest` would have forced me to add, if I hadn't already created it.

So how did I know to check `R.id.transactionsCount` in my test? I learned about the relationship between the layout XML document and the Android-generated class `R` when I hacked together my first screen. If I hadn't done that, then I'd have written a Learning Test to discover how to gain access to a layout element in the Activity.

[a] https://raw.github.com/jbrains/TrackEveryPenny/snapshot_20/res/layout/main.xml

For a moment I considered whether to handle a negative number of transactions, only because Java's primitives force me to. I don't know whether I should consider such a mistake likely or not, and I can handle it easily, so I decide to handle this case.

Snapshot 30: the new test

```
diff --git a/src/test/java/ca/jbrains/upfp/view/android/test/DisplayNumberO\
fTransactionsTest.java b/src/test/java/ca/jbrains/upfp/view/android/test/Di\
splayNumberOfTransactionsTest.java
index 4145d8d..45ae1a2 100644
--- a/src/test/java/ca/jbrains/upfp/view/android/test/DisplayNumberOfTransa\
ctionsTest.java
+++ b/src/test/java/ca/jbrains/upfp/view/android/test/DisplayNumberOfTransa\
ctionsTest.java
@@ -1,12 +1,13 @@
 package ca.jbrains.upfp.view.android.test;

 import android.widget.TextView;
+import ca.jbrains.toolkit.ProgrammerMistake;
 import ca.jbrains.upfp.*;
 import com.xtremelabs.robolectric.RobolectricTestRunner;
```

```
 import org.junit.Test;
 import org.junit.runner.RunWith;

-import static org.junit.Assert.assertEquals;
+import static org.junit.Assert.*;

 @RunWith(RobolectricTestRunner.class)
 public class DisplayNumberOfTransactionsTest {
@@ -29,4 +30,25 @@ public class DisplayNumberOfTransactionsTest {
         "12", transactionsCountView.getText().toString()
     );
   }
+
+  @Test
+  public void rejectNegativeNumber() throws Exception {
+    final BrowseTransactionsActivity
+        browseTransactionsActivity
+        = new BrowseTransactionsActivity();
+
+    browseTransactionsActivity.onCreate(null);
+
+    try {
+      browseTransactionsActivity
+          .displayNumberOfTransactions(-1);
+      fail(
+          "Why did you display a negative number of transactions?! " +
+          "That's crazy talk!");
+    } catch (ProgrammerMistake success) {
+      assertTrue(
+          success.getCause()
+              instanceof IllegalArgumentException);
+    }
+  }
 }
```

Here I use a trick that I learned from Nat Pryce's article "Throw Defect"[a]: I have created an unchecked exception class named `ProgrammerMistake` that I use whenever I want to signal an occurrence of something that ought not to happen. I have used it here to signal receiving a negative number of transactions, since no sane calculation involving a collection of transactions could answer a

> negative number. I have designed `ProgrammerMistake` as a trivial subclass of `RuntimeException` adding the constructors I need as I need them.
>
> [a] http://link.jbrains.ca/V9znH8

By throwing an exception, I put more pressure on the Activity to decide how to handle exceptions, which sounds like a fundamental design decision that I can't defer much longer.

The implementation does nothing surprising.

Snapshot 30: a Guard Clause

```
diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
index 50fa839..cf2fceb 100644
--- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
+++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
@@ -250,6 +250,13 @@ public class BrowseTransactionsActivity extends Activi\
ty {
   public void displayNumberOfTransactions(
       int transactionCount
   ) {
+    if (transactionCount < 0)
+      throw new ProgrammerMistake(
+          new IllegalArgumentException(
+              String.format(
+                  "number of transactions can't be negative, but it's %1$d\
",
+                  transactionCount)));
+
     final TextView transactionsCountView
         = (TextView) findViewById(R.id.transactionsCount);

```

Presenter implements the View interface

I'd intended for the Activity to implement the View, but I haven't done that yet. Instead, I've created a small Chunnel Problem in which the method signature in the Activity does not match the one in the View. I fix that by renaming the View method.

Snapshot 40: changing the View interface

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/mvp/BrowseTransactio\
nsView.java b/AndroidFree/src/test/java/ca/jbrains/upfp/mvp/BrowseTransacti\
onsView.java
index a3a7ae8..500be98 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/mvp/BrowseTransactionsView.\
java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/mvp/BrowseTransactionsView.\
java
@@ -1,5 +1,7 @@
 package ca.jbrains.upfp.mvp;

 public interface BrowseTransactionsView {
-  void setNumberOfTransactions(int numberOfTransactions);
+  void displayNumberOfTransactions(
+      int numberOfTransactions
+  );
 }
```

Snapshot 40: changing the View's clients to match

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/mvp/BrowseTransactio\
nsPresenter.java b/AndroidFree/src/test/java/ca/jbrains/upfp/mvp/BrowseTran\
sactionsPresenter.java
index aaa1dcb..e001829 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/mvp/BrowseTransactionsPrese\
nter.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/mvp/BrowseTransactionsPrese\
nter.java
@@ -13,6 +13,7 @@ public class BrowseTransactionsPresenter {
   }

   public void render() {
-    view.setNumberOfTransactions(model.countTransactions());
+    view.displayNumberOfTransactions(
+        model.countTransactions());
   }
 }
```

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/mvp/RenderYouHaveNTr\
ansactionsOnBrowseTransactionsScreenTest.java b/AndroidFree/src/test/java/c\
a/jbrains/upfp/mvp/RenderYouHaveNTransactionsOnBrowseTransactionsScreenTest\
.java
index de519f3..4e29ad6 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/mvp/RenderYouHaveNTransacti\
onsOnBrowseTransactionsScreenTest.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/mvp/RenderYouHaveNTransacti\
onsOnBrowseTransactionsScreenTest.java
@@ -23,7 +23,7 @@ public class RenderYouHaveNTransactionsOnBrowseTransactio\
nsScreenTest {
           allowing(model).countTransactions();
           will(returnValue(0));

-          oneOf(view).setNumberOfTransactions(0);
+          oneOf(view).displayNumberOfTransactions(0);
         }});

     presenter.render();
```

Now to make the Activity implement the View, I promote the View from the test source tree to a production source tree, since a production class–the Activity–now wants to use it.

Snapshot 50: moving the interface from test to production

```
 AndroidFree/src/{test => main}/java/ca/jbrains/upfp/mvp/BrowseTransactions\
View.java | 0
 src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java             \
          | 4 +++-
 2 files changed, 3 insertions(+), 1 deletion(-)
```

Snapshot 50: Activity implements View

```
diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
index cf2fceb..52a11bc 100644
--- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
+++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
@@ -7,6 +7,7 @@ import android.view.View;
 import android.widget.*;
 import ca.jbrains.toolkit.ProgrammerMistake;
 import ca.jbrains.upfp.domain.*;
+import ca.jbrains.upfp.mvp.BrowseTransactionsView;
 import com.google.common.collect.Lists;
 import org.joda.time.LocalDate;
 import org.joda.time.format.*;
@@ -15,7 +16,8 @@ import java.io.*;
 import java.text.*;
 import java.util.ArrayList;

-public class BrowseTransactionsActivity extends Activity {
+public class BrowseTransactionsActivity extends Activity
+    implements BrowseTransactionsView {
   public static final DateTimeFormatter DATE_TIME_FORMATTER
       = DateTimeFormat.forPattern("yyyy-MM-dd");
   // REFACTOR Move down into Android-free model layer
```

Since the Activity now implements the View, I try to extract a more precise contract for the View, avoiding a mismatch between the Presenter's assumptions about the View and the implementation of the View. Rather than simply extract contract tests at random, I look at the Presenter's tests, scanning for which View methods it replaces with stubs or expectations. The Presenter sets only a single expectation on a single View method: `displayNumberOfTransactions()`. Expectations in one layer must correspond to actions in the next, meaning that since Presenter expects the View to `displayNumberOfTransactions(0)`, I need to check what happens in the View implementation when I ask it to `displayNumberOfTransactions(0)`. It so happens that I already have such a test, although it checks the value `12` instead of `0`. I see no fundamental difference between displaying "You have `0` transactions" and "You have `12` transactions", so the expectation that the Presenter sets on the View matches the action in a test for the View. This View test, then, makes a good candidate for a contract test for the view.

What a shame, then, that I didn't actually do this the first time around.

Instead, I extracted a contract test for the negative number of transactions case, since it seemed sensible to clarify what the View should do in that case.

Extracting Contract Tests

A Contract Test differs from an ordinary test in only one, crucial way: the Contract Test makes no reference to a particular implementation of the interface whose contract it describes.

In this example, I separate initialising `BrowseTransactionsActivity` from the rest of the methods `happyPath()` and `rejectNegativeNumber()` on `DisplayNumberOfTransactionsTest`, leaving behind in those tests only code that refers to the interface `BrowseTransactionsView`.

Snapshot 60: A contract test for the View

```
1  package ca.jbrains.upfp.view.test;
2
3  import ca.jbrains.toolkit.ProgrammerMistake;
4  import ca.jbrains.upfp.mvp.BrowseTransactionsView;
5  import org.junit.Test;
6
7  import static org.junit.Assert.*;
8
9  public abstract class BrowseTransactionsViewContract {
10   @Test
11   public void rejectNegativeNumber() throws Exception {
12     final BrowseTransactionsView browseTransactionsView
13         = initializeView();
14
15     try {
16       browseTransactionsView
17           .displayNumberOfTransactions(-1);
18
19       fail(
20           "Why did you display a negative number of " +
21           "transactions?! That's crazy talk!");
22     } catch (ProgrammerMistake success) {
23       assertTrue(
24           success.getCause()
25               instanceof IllegalArgumentException);
26     }
27   }
28
```

```
29    protected abstract BrowseTransactionsView initializeView();
30  }
```

Snapshot 60: Pull the contract up from the test

```
diff --git a/src/test/java/ca/jbrains/upfp/view/android/test/DisplayNumberO\
fTransactionsTest.java b/src/test/java/ca/jbrains/upfp/view/android/test/Di\
splayNumberOfTransactionsTest.java
index 45ae1a2..d49e8ec 100644
--- a/src/test/java/ca/jbrains/upfp/view/android/test/DisplayNumberOfTransa\
ctionsTest.java
+++ b/src/test/java/ca/jbrains/upfp/view/android/test/DisplayNumberOfTransa\
ctionsTest.java
@@ -1,16 +1,19 @@
 package ca.jbrains.upfp.view.android.test;

 import android.widget.TextView;
-import ca.jbrains.toolkit.ProgrammerMistake;
 import ca.jbrains.upfp.*;
+import ca.jbrains.upfp.mvp.BrowseTransactionsView;
+import ca.jbrains.upfp.view.test.BrowseTransactionsViewContract;
 import com.xtremelabs.robolectric.RobolectricTestRunner;
 import org.junit.Test;
 import org.junit.runner.RunWith;

-import static org.junit.Assert.*;
+import static org.junit.Assert.assertEquals;

 @RunWith(RobolectricTestRunner.class)
-public class DisplayNumberOfTransactionsTest {
+public class DisplayNumberOfTransactionsTest
+    extends BrowseTransactionsViewContract {
+
   @Test
   public void happyPath() throws Exception {
     final BrowseTransactionsActivity
@@ -31,24 +34,14 @@ public class DisplayNumberOfTransactionsTest {
     );
   }

-  @Test
-  public void rejectNegativeNumber() throws Exception {
```

```
38  +  @Override
39  +  protected BrowseTransactionsView initializeView() {
40       final BrowseTransactionsActivity
41           browseTransactionsActivity
42           = new BrowseTransactionsActivity();
43
44       browseTransactionsActivity.onCreate(null);
45
46  -    try {
47  -      browseTransactionsActivity
48  -          .displayNumberOfTransactions(-1);
49  -      fail(
50  -          "Why did you display a negative number of transactions?! " +
51  -          "That's crazy talk!");
52  -    } catch (ProgrammerMistake success) {
53  -      assertTrue(
54  -          success.getCause()
55  -              instanceof IllegalArgumentException);
56  -    }
57  +    return browseTransactionsActivity;
58     }
59   }
```

By inheriting the implementation of `BrowseTransactionsViewContract`, `DisplayNumberOfTransactionsTest` signals that the implementation it checks—`BrowseTransactionsActivity`—acts like a `BrowseTransactionsView`. When I connect the Presenter to the Activity, the two will just work together.

Since I now realise that I ought to have extracted the happy path as a contract test, I add this task to my list.

Looking more closely at the contract of the View, I notice that the Presenter does nothing to prevent invoking `displayNumberOfTransactions()` with a negative number, which would result in a `ProgrammerMistake`. This presents me with two options: test that the Presenter propagates the `ProgrammerMistake` when it tries to display "You have -1 transactions" or add a semantic constraint to the Model so that `countTransactions()` always returns 0 or higher. The latter option seems more intuitively sensible *and* avoids writing a fairly trivial test, so I add a task to implementing the Model to extract a contract test that checks that `countTransactions()` is never negative.

What's done

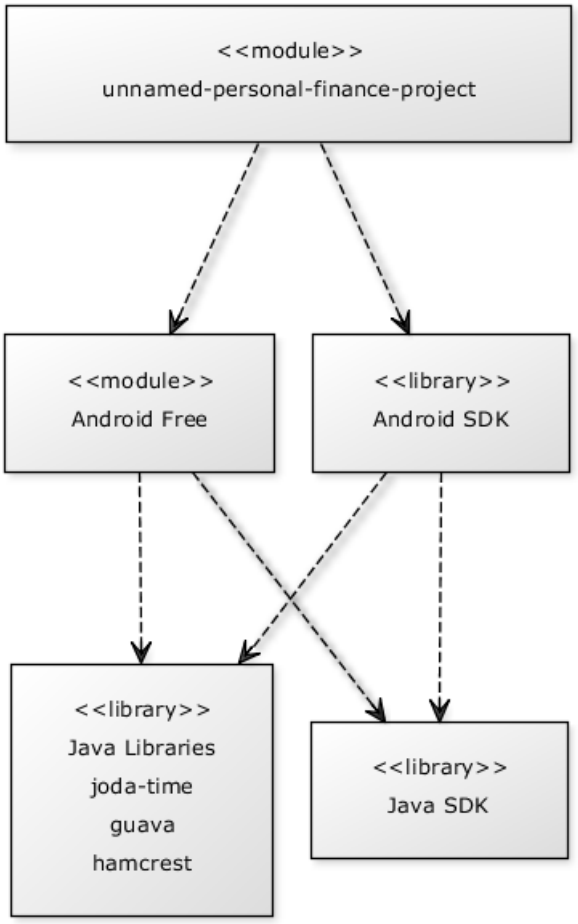
- Snapshot 20: Test-drove displaying the current transaction count in memory.
- Snapshot 30: We gracefully handle a negative number of transactions.

- Snapshot 40: Fixed the "Chunnel" problem between a class and the interface it's about to implement.
- Snapshot 50: BrowseTransactionsActivity now acts like a BrowseTransactionsView.
- Snapshot 60: Extracted a contract for BrowseTransactionsView from DisplayNumberOfTransactionsTest.

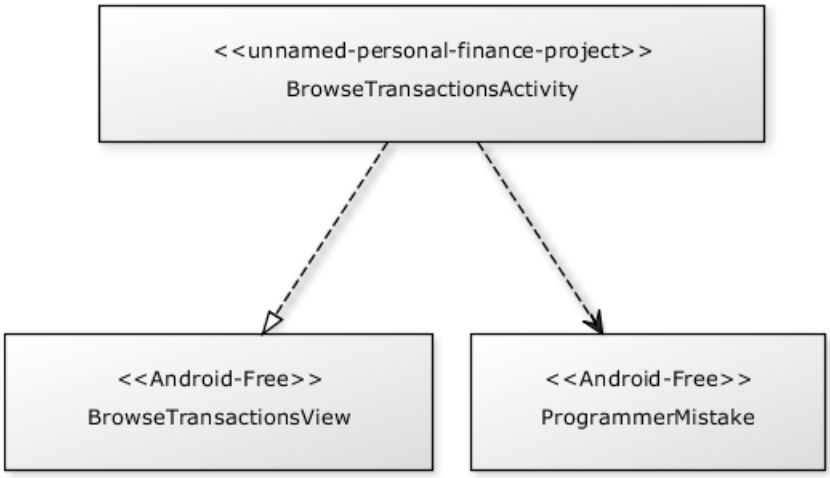
What's left to do

- Decide how Activity will handle Presenter throwing exceptions to it. Normal operation or should Presenter not throw exceptions?
- Fake out Model in Android Activity for now, since nothing can write to it.
- Do a little manual inspection to make sure we haven't messed anything up.
- Add to Model contract: `countTransactions() >= 0`; now View can safely blow up with ProgrammerMistake if it receives a negative number, so…
- Make Activity invoke the Presenter at the right lifecycle moments.

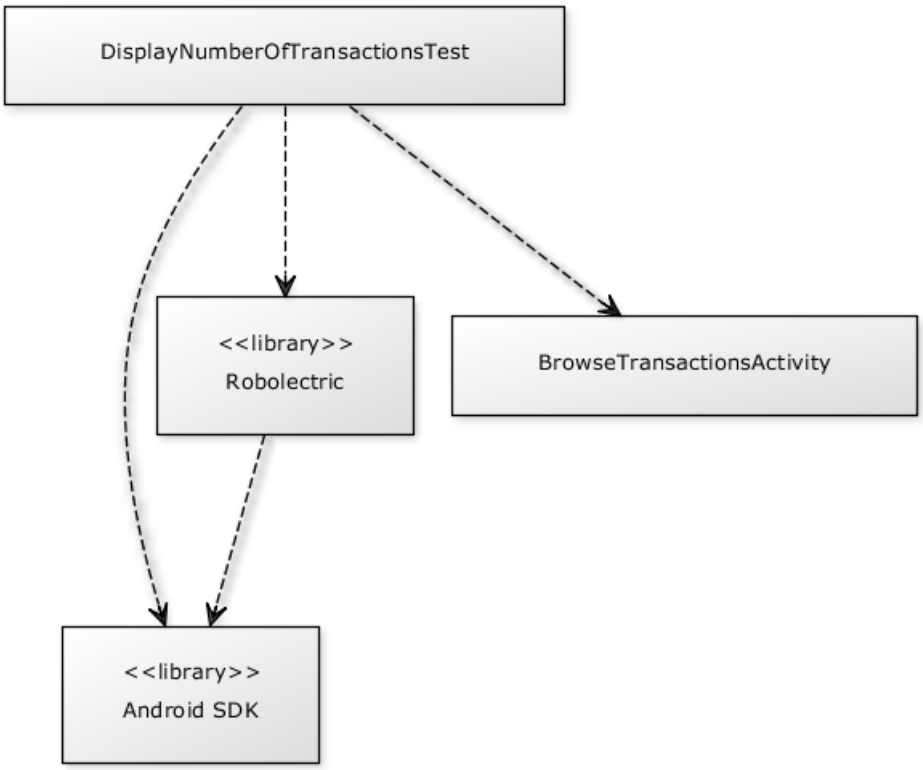
A recap of the current design



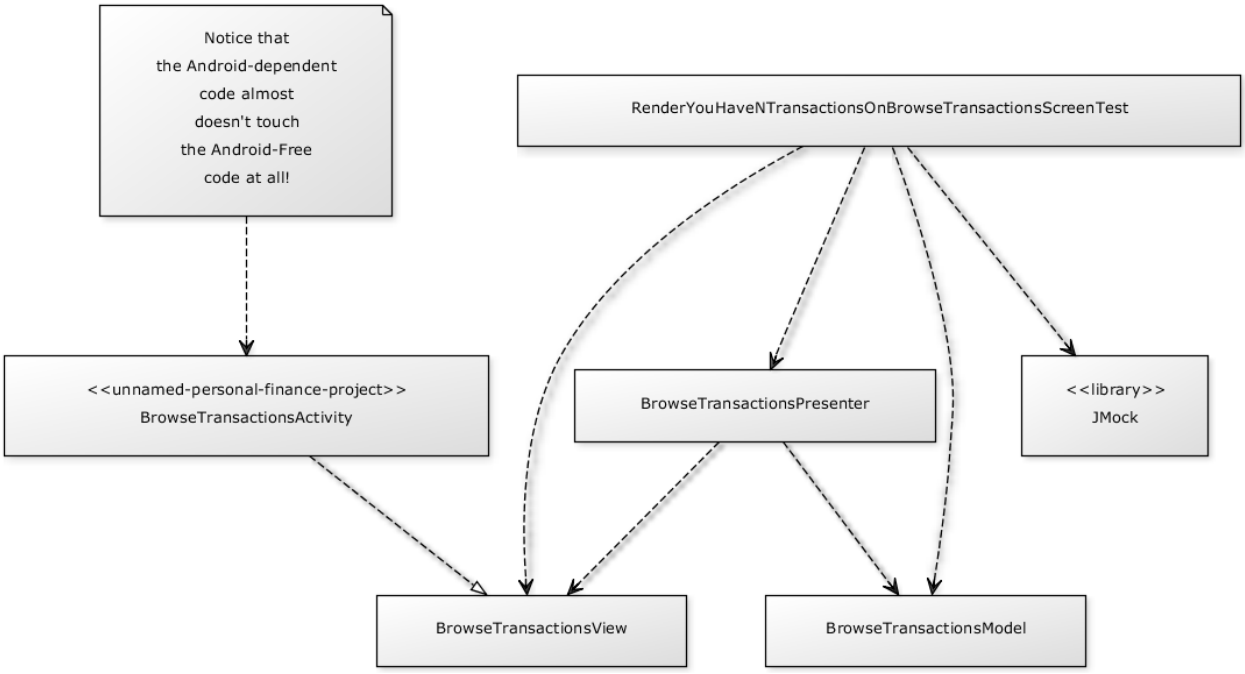
An ”architecture” view



Production classes



The Activity's tests



The Presenter's tests