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 2013-07-12

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.

©2012 - 2013 Diaspar Software Services Inc.

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

Android is a Trademark of Google Inc.

About This Book

This book shows you how I use various techniques and principles to design an Android application from the ground up. More than that, it shows you how I use these techniques to learn the Android development environment and libraries quickly. I have included almost every micro-commit of work as they actually happened, although I have elided a few as the means to protect your sanity. I have edited the writing to make it more informative, but I have not edited much of the work. This means at least two important things: this book represents the way I actually work on "real projects", rather than an idealised view of responsible design; and you will encounter messy, disorderly progress at times. I have chosen to depict the work as it happened, rather than as I wish it had happened, so if your temperament demands an orderly sequence of neat concepts culminating in a perfect result, then stop reading now; but if you want to experience real, in-the-moment learning and decision making, applying the key principles of responsible design, then dive in–I hope you like it.

I have chosen a direct, sometimes terse, style for this book. I did this simply to get to the point and avoid making the book any longer than necessary. I hope that you appreciate this, rather than find it dry and unappealing.

About The Rough Edges

This book will have plenty of rough edges in its production value for a while. If you want to suggest ways to improve this aspect of the book, then don't keep them to yourself. If you just want to complain about the rough edges, then please do keep that to yourself. I intend to pretty the book up the best I can as time rolls on.

Read this section before telling me about the code samples. I have tried to make the code samples easier to read, but Java code samples with a 38-character-wide screen just don't work. If you're reading this in two-column mode on a wide screen, such as the Kindle app on a computer monitor, then switch to one-column code to read the code samples. If you're reading this on a narrow screen, like an e-book reader, then try the PDF format in landscape mode. The code samples simply will not look good on a narrow screen in portrait mode, no matter what I do. To compensate, I will add links to the source code online. Where possible, I will describe the code or changes in the text, and you'll have to look at the code in some other format. I really don't know what else to do, and I really apologise. If you know how to make this situation better, then please tell me.

Some of the code samples are better describes with **diffs**, however, the built-in rendering of the diff format is a little tough to read. I'm looking for ways to make this better, but I don't want to hand-tune these diffs while finishing the first draft, because it will only delay the first draft further. If you find a diff particularly tough to follow, then I strongly suggest going to the corresponding snapshot tag at http://www.github.com/jbrains/TrackEveryPenny/tags[1], where you can explore the changes in a more flexible, easy-to-read format.

Read this section before telling me about the class diagrams. I wanted the class diagrams to appear transparent, but the .mobi format insists on turning transparency into a black background, so I have had to insert a white background instead. If you read this book on a gray background, then I apologise for the white background in those diagrams. **Also**, it appears that the Kindle reader cuts off the right side of images in two-column mode, but I really don't want to make those images any smaller, so please consider reading this book in one-column mode. I apologise again, but I don't know how to fix this problem.

Reading this book on an iPad. I have experimented a little with the settings on iPad. You'll probably have the best experience with the medium-to-large fonts, widest margins and the one-column layout.

[1] http://www.github.com/jbrains/TrackEveryPenny/tags

About The Author

J. B. Rainsberger (http://www.jbrains.ca[2], http://blog.thecodewhisperer.com[3], http://www.myagiletutor.com[4]) helps software companies better satsify their customers and the businesses they support. Over the years, he has learned to write valuable software, overcome many of his social deficiencies, and built a life that he loves. He travels the world sharing what he's learned, hoping to help other people get what they want out of work and out of their lives. He lives in Atlantic Canada with his wife Sarah and their three cats.

J. B.'s first book, *JUnit Recipes*, described how to use JUnit in detail to test Java applications of all kinds. He wishes he had written *Growing Object-Oriented Systems: Guided by Tests*, but his friends beat him to it, so he wrote this one instead.

J. B. hangs out on Twitter as `@jbrains`[5]

[2] http://www.jbrains.ca

[3] http://blog.thecodewhisperer.com

[4] http://www.myagiletutor.com

[5] http://www.twitter.com/jbrains

J. B. Rainsberger

1 So, where do we start?

I like to start any application with a feature that *tells* the user something, then work my way back to the feature that lets the user provide information. This tends to get me to a useful release sooner, as it discourages me from adding more than I need to deliver valuable output to the user.

About the app

I want users of this app to understand their personal spending better, as a step towards personal financial freedom. When I started doing this about ten years ago, I tracked every penny coming in and going out for three months, after that I analysed the flow of money and judged how much I truly valued each category of expenditure. The underlying philosophy–spend on what you value, not what you can afford–played a key role in helping me retire at 34, and I hope that this app will help me share that philosophy with a wide audience.[1]

We have simple, powerful tools to help us analyse our spending, namely online banking and the trusty spreadsheet. I don't aim to replace these tools, but rather to complement them. While we can download our credit card and debit card transactions from our banks, cash transactions can fall through the cracks, so I intend to start there.

For the first release on this app, I want to be able to track cash transactions as I make them, then be able to combine these cash transactions with the transactions that I can download from my banks. This sounds big enough to merit releasing, but small enough to limit the investment of time in finding users. I like to think of it as an inbox[2] for transactions that might otherwise fall through the cracks of our memory, which the user would process periodically. This already gives me a few interesting ideas for future features, such as reminding the user to process the inbox!

Product Sashimi

I want to slice this product thinly to make it easier to deliver something of value sooner. I have already identified the first *feature area*: tracking and exporting cash transactions. Now I need to work from the output back to the input to identify what to build.

What do I mean by "analyse the flow of money" in this context? When I read *Your Money or Your Life*, it suggested tracking every penny in and out in various categories of income and expenditure for a period of at least three months. I have in mind a spreadsheet that looks like this:

[1] I learned this reading *Your Money or Your Life* by Vicki Robin and Joe Dominguez. Although I read the first edition, the second edition eliminates much of the obsolete advice about investing in bonds. Interest rates haven't been the same for savers since the 1980s.

[2] I've used the term "inbox" the way David Allen uses it in *Getting Things Done*, meaning a place to quickly collect potential future tasks before forgetting them.

	A	B	C	D
1	Money Flow for 3 months ending December 31, 2011			
2				
3	**Category**	**Total**	**How much do you value this?**	**Comments**
4	Phone	$251.63	5	
5	Internet	$418.11	5	
6	Eating Out	$302.10	2	If I wasn't so tired, I'd cook more.
7	Rent	$5,820.00	3	I hardly take advantage of living in this city.
8	Home Insurance	$97.50	5	
9	Groceries	$2,780.00	4	I wish we could buy more directly from farmers.

How I imagine one might analyse their expenditures

Of course, this spreadsheet summarises the data, but doesn't tell me what data it summarises. What does this spreadsheet summarise and what constraints does it use to do that? It summarises **amounts** by **category** within a specific **time period**, so it seems I need to track the **date**, **category** and **amount** for each expenditure. That means that I want the app to export data that looks like this:

	A	B	C	D
1	**Date**	**Description**	**Category**	**Amount**
2	11/14/2011	Mobile phone	Phone	$118.98
3	11/17/2011	Home internet service	Internet	$127.80
4	11/17/2011	Dinner with friends	Eating Out	$24.90
5	11/18/2011	Groceries	Groceries	$109.50
6	11/19/2011	Taxi to work	Transportation	$14.00

How I imagine one might want their expenditures exported for analysis

That leads me to the first feature: **export expenditures as comma-separated values suitable for import into a spreadsheet**. I don't have to mess around with a data entry screen to make this work: I can simply hardcode some expenditures inside the app, then export them to a file. That sounds like a great place to start: I can focus on learning about the Android framework while building a feature that I understand very well, so I limit the number of things I have to worry about, which usually means a greater probability of success.

2 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.

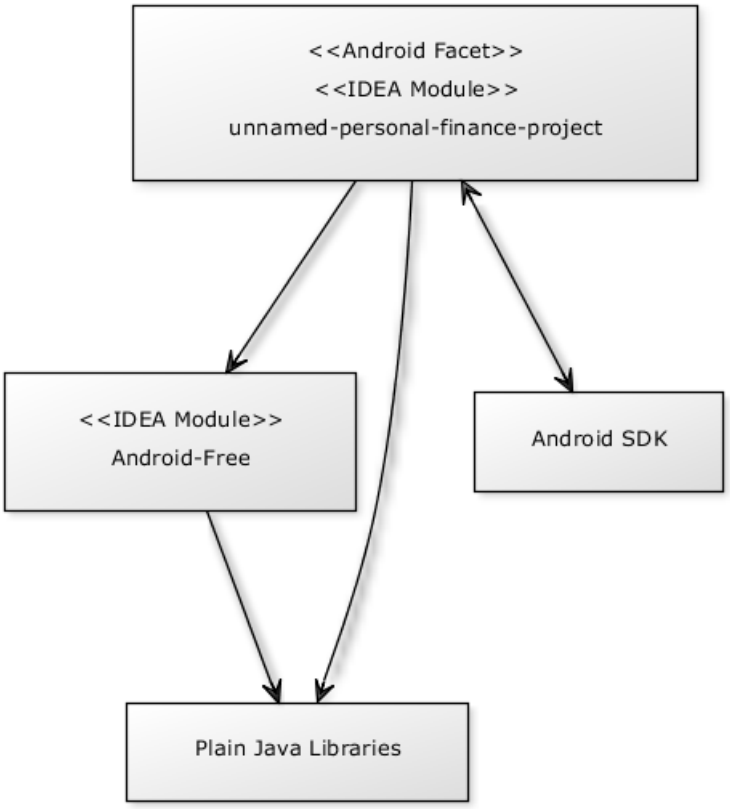
View or `View`?

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[3]
2. Follow the instructions here[4] 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.

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 github[5] and this particular version is tagged `snapshot_10`. I will introduce the next code sample more simply with the heading "Snapshot 20".

[3] http://developer.android.com/sdk/index.html#ExistingIDE

[4] http://developer.android.com/sdk/installing/adding-packages.html

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

Snapshot 10: the first test

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

import org.jmock.*;
import org.jmock.integration.junit4.JMock;
import org.junit.Test;
import org.junit.runner.RunWith;

@RunWith(JMock.class)
public class RenderYouHaveNTransactionsOnBrowseTransactionsScreenTest {
  private Mockery mockery = new Mockery();

  @Test
  public void zero() throws Exception {
    final BrowseTransactionsModel model
        = mockery.mock(BrowseTransactionsModel.class);
    final BrowseTransactionsView view
        = mockery.mock(BrowseTransactionsView.class);
    final BrowseTransactionsPresenter presenter
        = new BrowseTransactionsPresenter(model, view);

    mockery.checking(
        new Expectations() {{
          allowing(model).countTransactions();
          will(returnValue(0));

          oneOf(view).setNumberOfTransactions(0);
        }});

    presenter.render();
  }
}
```

The implementations offer no surprises.

Snapshot 10: give the Model to the View

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

public class BrowseTransactionsPresenter {
  private final BrowseTransactionsModel model;
  private final BrowseTransactionsView view;

  public BrowseTransactionsPresenter(
      BrowseTransactionsModel model,
      BrowseTransactionsView view
  ) {
    this.model = model;
    this.view = view;
  }

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

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

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

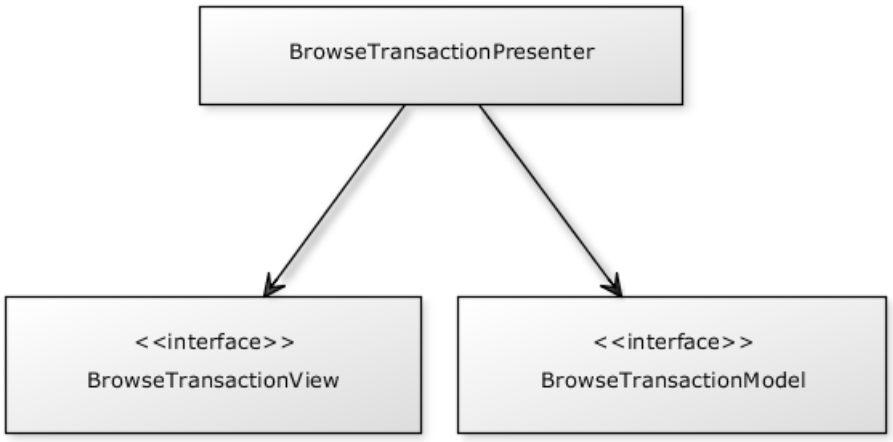
```
package ca.jbrains.upfp.mvp;

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.

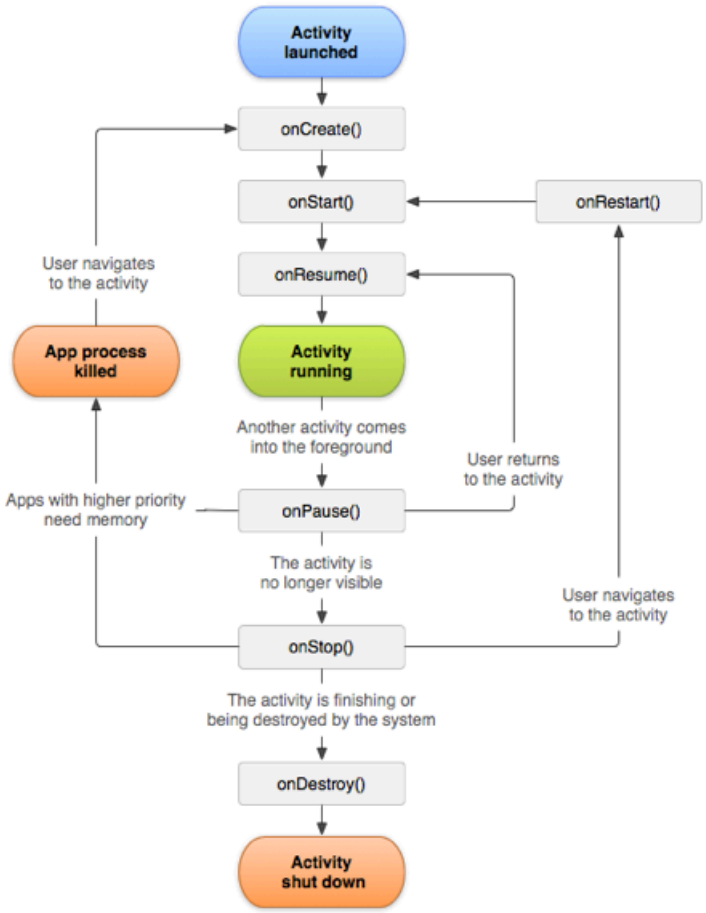
A snapshot of the design



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.

3 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);
```

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

```
23
24      final TextView transactionsCountView
25          = (TextView) browseTransactionsActivity
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.

In order to alert you to the changes between this and the previous version, I've rendered this code sample as a **diff**. You might find it hard to read a raw diff, and I don't blame you, but while I'm completing the first draft, I need a way to extract the changes from the code base and highlight them for you, and the diff format fits best. As the book nears completion, I will find a prettier way to render these changes. In the meantime, consider going to http://www.github.com/jbrains/TrackEveryPenny/tags[2] and find the tag corresponding to this snapshot. This way, you can explore the code more freely, and you'll almost certainly find the diff easier to read there than here.

[2] http://www.github.com/jbrains/TrackEveryPenny/tags

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[3]. 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.

[3] 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"[4]: 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.

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
```

[4] http://link.jbrains.ca/V9znH8

```
10     ) {
11 +     if (transactionCount < 0)
12 +       throw new ProgrammerMistake(
13 +           new IllegalArgumentException(
14 +               String.format(
15 +                   "number of transactions can't be negative, but it's %1$d\
16 ",
17 +                   transactionCount)));
18 +
19       final TextView transactionsCountView
20           = (TextView) findViewById(R.id.transactionsCount);
21
```

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

```
1  diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/mvp/BrowseTransactio\
2  nsView.java b/AndroidFree/src/test/java/ca/jbrains/upfp/mvp/BrowseTransacti\
3  onsView.java
4  index a3a7ae8..500be98 100644
5  --- a/AndroidFree/src/test/java/ca/jbrains/upfp/mvp/BrowseTransactionsView.\
6  java
7  +++ b/AndroidFree/src/test/java/ca/jbrains/upfp/mvp/BrowseTransactionsView.\
8  java
9  @@ -1,5 +1,7 @@
10  package ca.jbrains.upfp.mvp;
11
12  public interface BrowseTransactionsView {
13 -  void setNumberOfTransactions(int numberOfTransactions);
14 +  void displayNumberOfTransactions(
15 +      int numberOfTransactions
16 +  );
17  }
```

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

```
1   diff --git a/src/test/java/ca/jbrains/upfp/view/android/test/DisplayNumberO\
2   fTransactionsTest.java b/src/test/java/ca/jbrains/upfp/view/android/test/Di\
3   splayNumberOfTransactionsTest.java
4   index 45ae1a2..d49e8ec 100644
5   --- a/src/test/java/ca/jbrains/upfp/view/android/test/DisplayNumberOfTransa\
6   ctionsTest.java
7   +++ b/src/test/java/ca/jbrains/upfp/view/android/test/DisplayNumberOfTransa\
8   ctionsTest.java
9   @@ -1,16 +1,19 @@
10   package ca.jbrains.upfp.view.android.test;
11
12   import android.widget.TextView;
13  -import ca.jbrains.toolkit.ProgrammerMistake;
14   import ca.jbrains.upfp.*;
15  +import ca.jbrains.upfp.mvp.BrowseTransactionsView;
16  +import ca.jbrains.upfp.view.test.BrowseTransactionsViewContract;
17   import com.xtremelabs.robolectric.RobolectricTestRunner;
18   import org.junit.Test;
19   import org.junit.runner.RunWith;
20
21  -import static org.junit.Assert.*;
22  +import static org.junit.Assert.assertEquals;
23
24   @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 {
+  @Override
+  protected BrowseTransactionsView initializeView() {
     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);
-    }
+    return browseTransactionsActivity;
   }
 }
```

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.

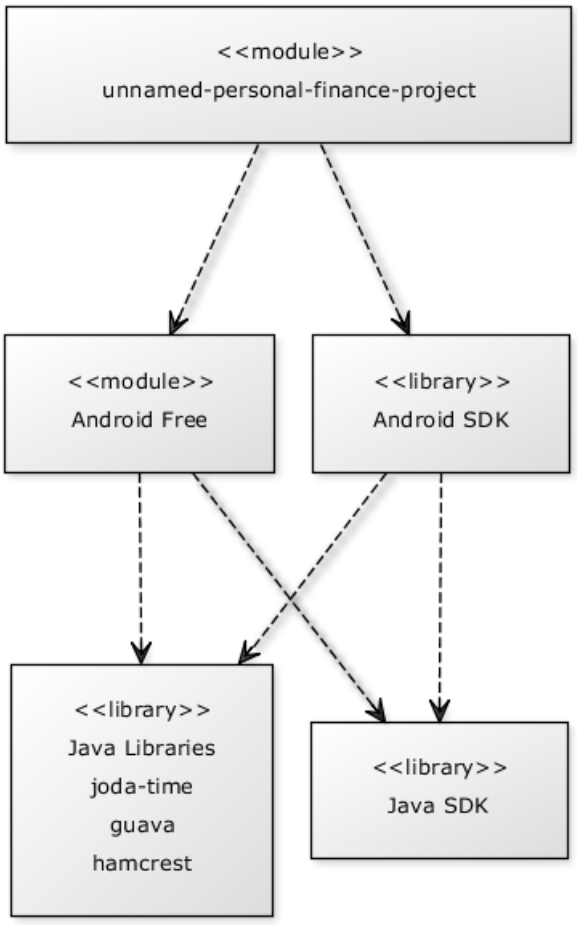
What's done

- 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.

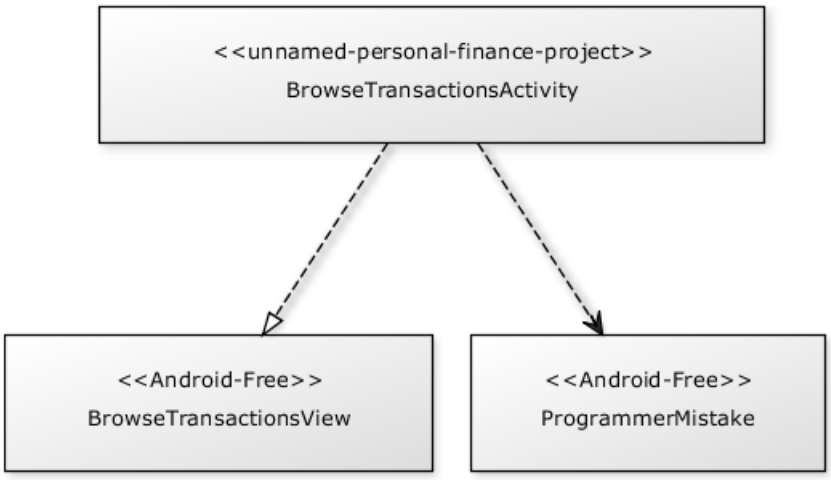
What's left to do

- 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.

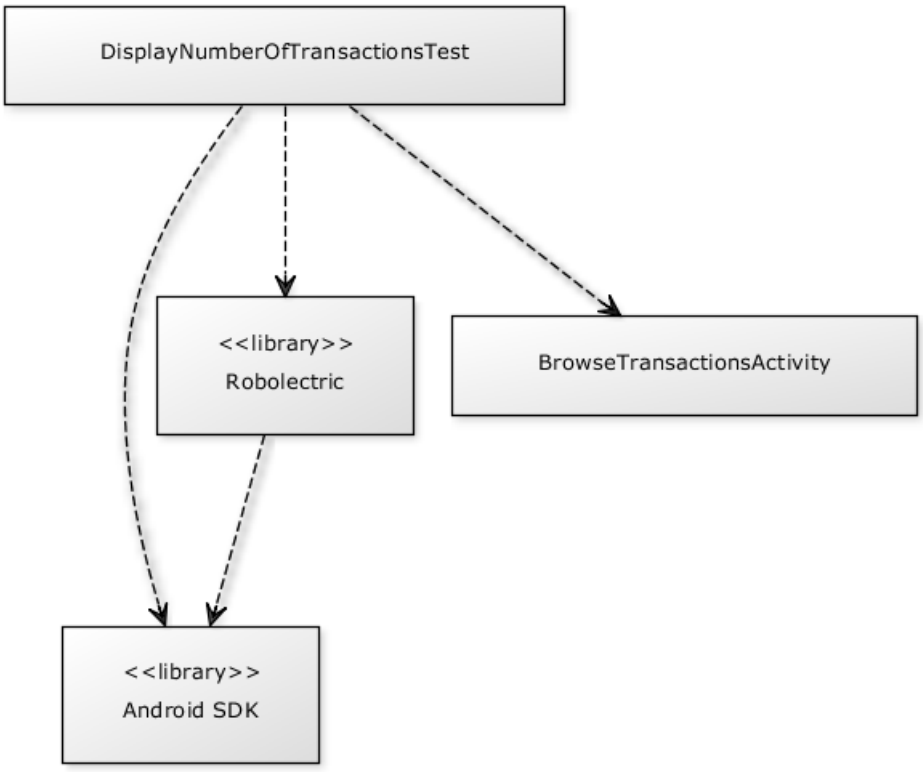
A recap of the current design



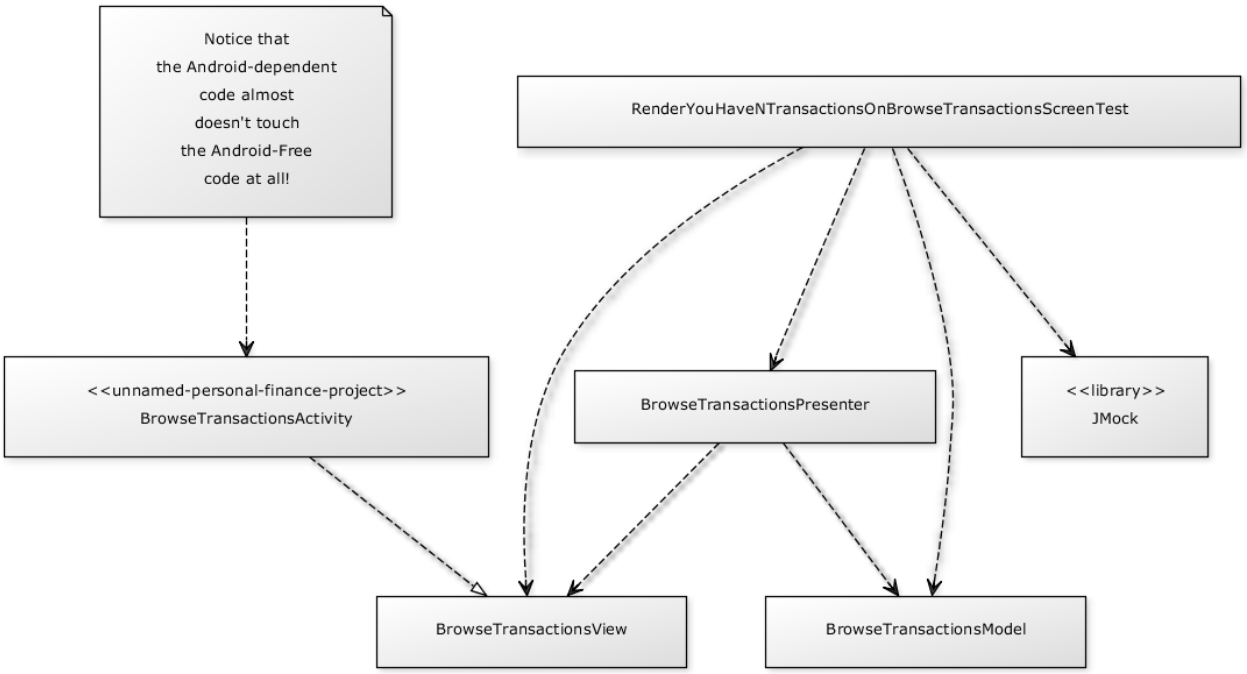
An "architecture" view



Production classes



The Activity's tests



The Presenter's tests

4 Activity Renders the View

Ordinarily, I'd rather complete the MVP triad before connecting it to its client, but since I need to learn more urgently about the Android SDK, I choose to connect the Activity to the Presenter right away. Since I've already implemented the Presenter and View, and I'd planned to fake out the Model for now, this should cause no significant problem.

According to the Activity lifecycle diagram, `onResume()` should ask the Presenter to perform its default rendering, so I add that to my test list. It feels a little artificial to extract an interface for the Presenter, but I strongly want to avoid duplicating behavior in tests for the Activity and the Presenter, so I follow my mechnical rule–mock only interfaces–and hope that it works out.

If you're not sure what I mean by duplicating behavior in the tests for the Activity and the Presenter, try writing tests for the Activity and the Presenter without an interface between them. Compare the two sets of tests and you'll likely see what I mean.

Snapshot 70: The new test

```
1  package ca.jbrains.upfp.controller.android.test;
2
3  import ca.jbrains.upfp.BrowseTransactionsActivity;
4  import com.xtremelabs.robolectric.RobolectricTestRunner;
5  import org.jmock.*;
6  import org.junit.Test;
7  import org.junit.runner.RunWith;
8
9  // I can't run the Activity without Robolectric, and I
10 // can't run with both Robolectric and JMock, so I choose
11 // to run with Robolectric and do the JMock stuff by hand.
12 @RunWith(RobolectricTestRunner.class)
13 public class RenderBrowseTransactionsScreenTest {
14   private Mockery mockery = new Mockery();
15
16   @Test
17   public void askToRenderOnResume() throws Exception {
18     final RendersView rendersView = mockery.mock(
```

```
19          RendersView.class);
20
21      mockery.checking(
22          new Expectations() {{
23            oneOf(rendersView).render();
24          }});
25
26      new BrowseTransactionsActivity(rendersView) {
27        // Make this method visible so that I can invoke it
28        // in the test.
29        //
30        // REFACTOR Move to ShadowActivity
31        @Override
32        public void onResume() {
33          super.onResume();
34        }
35      }.onResume();
36
37      mockery.assertIsSatisfied();
38    }
39  }
```

You'll notice a few annoying details. First, I'd like to use both the Robolectric and JMock test runners at the same time, but that doesn't work out of the box, and I don't feel like trying to write a custom test runner that decorates both of these. I choose instead to duplicate the work of the JMock test runner in the test, since I know JMock more thoroughly than I know Robolectric. Once this causes a problem, I'll have to consider a more robust solution.

Next, notice that the Android framework's authors have decided to make the Activity lifecycle methods non-`public`. I consider this a kind of misguided attempt at encapsulation, preferring instead to Hide Methods with Interfaces, not Visibility. Encapsulation should shield the reader from distracting details, but this design choice blocks me–poorly at that–from a detail that I find essential right now. In order to test-drive my implementation of part of the Activity's lifecycle, I have to promote its visibility to `public` in an anonymous subclass. The Android framework's authors could have avoided this bureaucratic and easily-swept-aside roadblock by extracting an interface for the Activity's lifecycle, as the principle Prefer Interfaces to Abstract Classes suggests.

Otherwise, I find the test quite straightforward: `onResume()` renders the View.

I don't like some of the names I've chosen here: I find them vague and overly-structural. I don't like the duplication in `RendersView.render()`. As I write more code in other Presenters, I trust that I will gather better ideas about how to name these methods, classes, and interfaces, but I don't let that stop me from progressing now. Find my model for improving names at The Code Whisperer[1].

Snapshot 70: Activity now renders View

```
diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
index 52a11bc..0595db6 100644
--- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
+++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
@@ -6,6 +6,7 @@ import android.util.Log;
 import android.view.View;
 import android.widget.*;
 import ca.jbrains.toolkit.ProgrammerMistake;
+import ca.jbrains.upfp.controller.android.test.RendersView;
 import ca.jbrains.upfp.domain.*;
 import ca.jbrains.upfp.mvp.BrowseTransactionsView;
 import com.google.common.collect.Lists;
@@ -47,6 +48,24 @@ public class BrowseTransactionsActivity extends Activity
                 new LocalDate(2012, 11, 17),
                 "Groceries", -1350)
         );
+  private final RendersView rendersView;
+
+  public BrowseTransactionsActivity() {
+    this(null);
+  }
+
+  public BrowseTransactionsActivity(
+      RendersView rendersView
+  ) {
+    this.rendersView = rendersView;
+  }
+
+  @Override
```

[1] http://blog.thecodewhisperer.com/2011/06/15/a-model-for-improving-names/

```
31 +  protected void onResume() {
32 +    super.onResume();
33 +    // Arbitrarily, I assume that I should do my work after the superclass\
34 , but I don't really know.
35 +    rendersView.render();
36 +  }
37
38    /**
39     * Called when the activity is first created.
```

In the default constructor for `BrowseTransactionsActivity` I intentionally connect the Activity to a `null` Presenter, because no test invokes the default constructor. That comes next.

Fire up the simulator

I suppose I could write an automated integrated test connecting the Activity to the real Presenter and so on, but I'd rather focus my energy on testing and designing the pieces in isolation from each other. For now, for such a simple app with such a simple feature, I will inspect things manually. This involves firing up the simulator and running the app, faking out the Model, then looking for "You have 12 transactions" on the screen. I expect to find this very satisfying.

First, I need `BrowseTransactionsPresenter` to implement `RendersView` so that the default constructor in `BrowseTransactionActivity` can instantiate the Browse Transactions MVP triad and pass it to the other constructor. Since a prodution client now wants to use these classes, I move them from the test source tree and package to the production source tree and package, then connect the wires.

```
1  AndroidFree/src/{test => main}/java/ca/jbrains/upfp/mvp/BrowseTransactions\
2 Model.java     | 0
3  AndroidFree/src/{test => main}/java/ca/jbrains/upfp/mvp/BrowseTransactions\
4 Presenter.java | 0
5  2 files changed, 0 insertions(+), 0 deletions(-)
```

Now that I can connect the wires without creating a cyclic dependency, I do. I move `RendersView` to a more appropriate production package.

```
 AndroidFree/src/main/java/ca/jbrains/upfp/mvp/BrowseTransactionsPresenter.\
java                                               |  3 ++-
 {src/main/java/ca/jbrains/upfp/controller/android/test => AndroidFree/src/\
main/java/ca/jbrains/upfp/mvp}/RendersView.java |  2 +-
 src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java            \
                                                   | 13 ++++++++++---
 src/test/java/ca/jbrains/upfp/controller/android/test/RenderBrowseTransact\
ionsScreenTest.java                                |  1 +
 4 files changed, 14 insertions(+), 5 deletions(-)
```

Snapshot 90: Presenter now implements `RendersView`

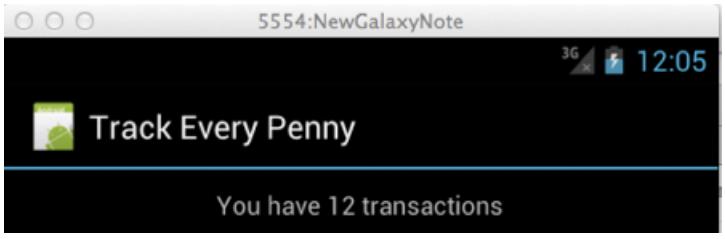
```
diff --git a/AndroidFree/src/main/java/ca/jbrains/upfp/mvp/BrowseTransactio\
nsPresenter.java b/AndroidFree/src/main/java/ca/jbrains/upfp/mvp/BrowseTran\
sactionsPresenter.java
index e001829..31c4224 100644
--- a/AndroidFree/src/main/java/ca/jbrains/upfp/mvp/BrowseTransactionsPrese\
nter.java
+++ b/AndroidFree/src/main/java/ca/jbrains/upfp/mvp/BrowseTransactionsPrese\
nter.java
@@ -1,6 +1,7 @@
 package ca.jbrains.upfp.mvp;

-public class BrowseTransactionsPresenter {
+public class BrowseTransactionsPresenter
+    implements RendersView {
    private final BrowseTransactionsModel model;
    private final BrowseTransactionsView view;

```

Snapshot 90: Activity now instantiates a Presenter

```
diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
index 0595db6..fac35fd 100644
--- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
+++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
@@ -6,9 +6,8 @@ import android.util.Log;
 import android.view.View;
```

```
 import android.widget.*;
 import ca.jbrains.toolkit.ProgrammerMistake;
-import ca.jbrains.upfp.controller.android.test.RendersView;
 import ca.jbrains.upfp.domain.*;
-import ca.jbrains.upfp.mvp.BrowseTransactionsView;
+import ca.jbrains.upfp.mvp.*;
 import com.google.common.collect.Lists;
 import org.joda.time.LocalDate;
 import org.joda.time.format.*;
@@ -51,7 +50,15 @@ public class BrowseTransactionsActivity extends Activity
   private final RendersView rendersView;

   public BrowseTransactionsActivity() {
-    this(null);
+    // We can't chain the constructor, because the instance in the process\
 of being created is itself the view.
+    // We have to wait for super() to be (implicitly) invoked.
+    this.rendersView = new BrowseTransactionsPresenter(
+        new BrowseTransactionsModel() {
+          @Override
+          public int countTransactions() {
+            return 12;
+          }
+        }, this);
   }

   public BrowseTransactionsActivity(
```

I have faked out the Model for now, although I don't want to leave it like this very long. With the MVP triad connected to the Activity, I fire up the simulator and receive satisfaction.



It works! No integrated tests. This still amazes me.

Cleaning up

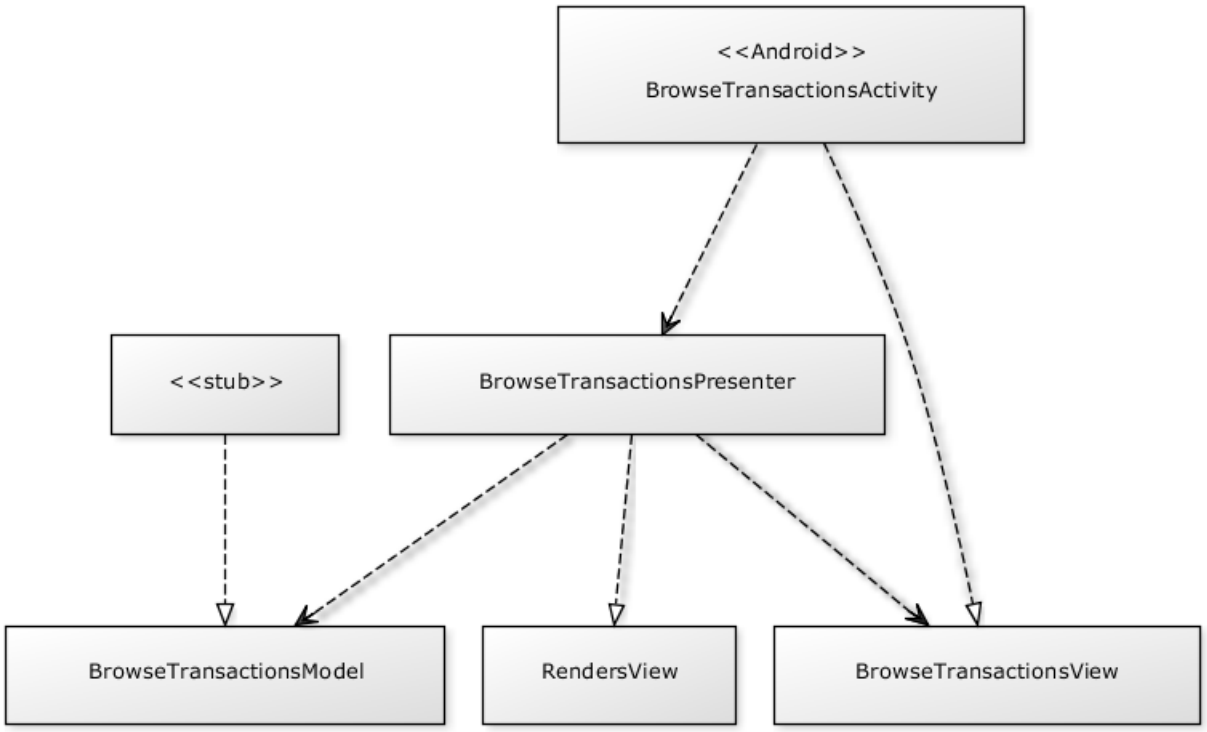
Before finishing this task, I remove some now-obsolete code that I'd written when I'd explored the Android SDK and hacked some stuff together–the kind of reckless coding that some people would try to justify as a "spike". Fortunately, this involves removing code from a lifecycle method, which simplifies the design, limiting the amount of code directly implemented in the framework's extension points.

Snapshot 100: Removed obsolete code

```
diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
index fac35fd..ddec922 100644
--- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
+++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
@@ -81,9 +81,6 @@ public class BrowseTransactionsActivity extends Activity
   public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);
-    final TextView transactionsCountView
-        = (TextView) findViewById(R.id.transactionsCount);
-    transactionsCountView.setText(String.valueOf(1));
   }

   public void exportTransactionsToCsv(View clicked) {
```

Design Snapshot

I have kept the Android-dependent and Android-free modules strictly separated, so only the former depends on the latter. Although I introduced the `RendersView` interface to avoid an integrated test, it has helped me avoid a hardwired cyclic dependency between `BrowseTransactionsActivity` and `BrowseTransactionsPresenter`.



The Android-Free side doesn't know about the Android-dependent side

Design Review

Should the Activity implement the View directly? I have a bad feeling about this. I chose this design due to expediency, but it encourages putting unrelated responsibilities in the same place, which lowers cohesion. Specifically, it increases the chance of accidentally changing the wrong thing later. If, in the process of adding a feature, I feel uneasy about changing code in this neighborhood, then I will separate the responsibilities then. I'll use that unease as a *transition indicator*, signaling the moment to deploy the aforementioned *mitigation* tactic of separating responsibilities.[2]

What's done

- Snapshot 70: BrowseTransactionsActivity asks to render its screen on 'resume'.
- Snapshot 80: Moved production code from test source tree to production source tree in preparation for implementing an interface.
- Snapshot 90: Wired the Activity to a real Presenter, in preparation for a manual test.
- Snapshot 100: Removed obsolete code.

[2] I use the terms *transition indicator* and *mitigation* as Tom DeMarco and Tim Lister use them in their classic Waltzing with Bears

What’s left to do

- Decide how Activity will handle Presenter throwing exceptions to it. Normal operation or should Presenter not throw exceptions?
- Add to Model contract: `countTransactions() >= 0`

5 Interlude: Avoiding Premature Generalisation

Both the remaining tasks on my list appear to force me to generalise the design before I feel ready to do that. Allow me to explain.

Too soon to handle exceptions uniformly

Since I've implemented only one Presenter and one Activity, deciding how an Activity would handle exceptions from a Presenter *in general* feels premature. Of the options I've considered, this one that bothers me least: pull the exception handling up into an `ExceptionHandlingActivity` where each lifecycle method implements a Template Method[1] to handle exceptions in its subclasses' lifecycle methods. As an example:

```
1  public abstract class ExceptionHandlingActivity
2    extends Activity {
3
4    @Override
5    public void onCreate(Bundle instance) {
6      super.onCreate(instance);
7      try {
8        saferOnCreate(instance);
9      } catch (Exception logged) {
10       Log.e(...);
11     }
12   }
13 }
14
15 public class MainActivity extends BaseActivity {
16   @Override
17   public void saferOnCreate(Bundle instance) {
18     setContentView(R.layout.main_activity);
19     // ...
20   }
21 }
```

[1] One of the originally-named Design Patterns.

This seems good, but feels heavyweight, so I'd rather wait until I duplicate the exception handling pattern a few more times before making any significant changes. I see where this might lead, and I don't want to travel that road until I've had a chance to find simpler options.

A strong signal: mocking part of an object to test the rest of it

Sometimes I find myself wanting to mock part of an object in order to test another part of it, as I would do to implement `ExceptionHandlingActivity`. I'd want to write tests like this one.

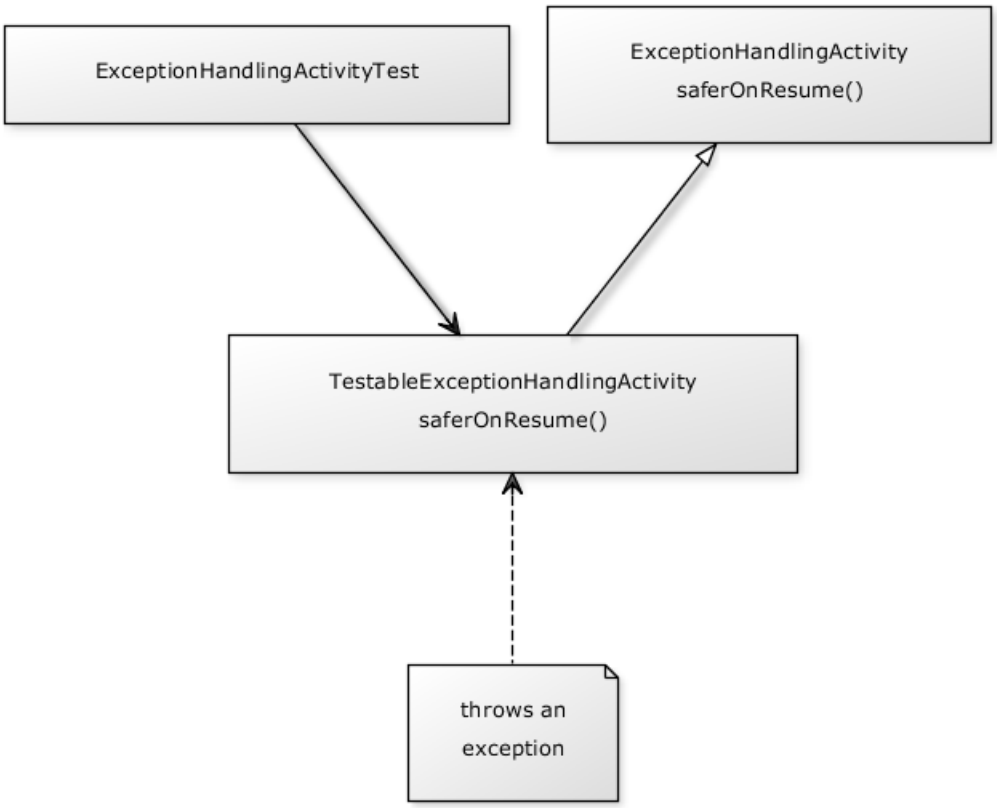
1. stub the subclass's implementation of `saferOnResume()` to throw `RuntimeException`
2. invoke `ExceptionHandlingActivity.onResume()`
3. check the result of `onResume()`'s generic exception handler–for example, logging the right text at the right loglevel

I don't need to know–and therefore don't *want* to know–what the subclass did to throw an exception; I only care that it throws one. Try to write that test and you'll see what makes it tricky to write correctly: I need the Subclass to Test pattern to do it.

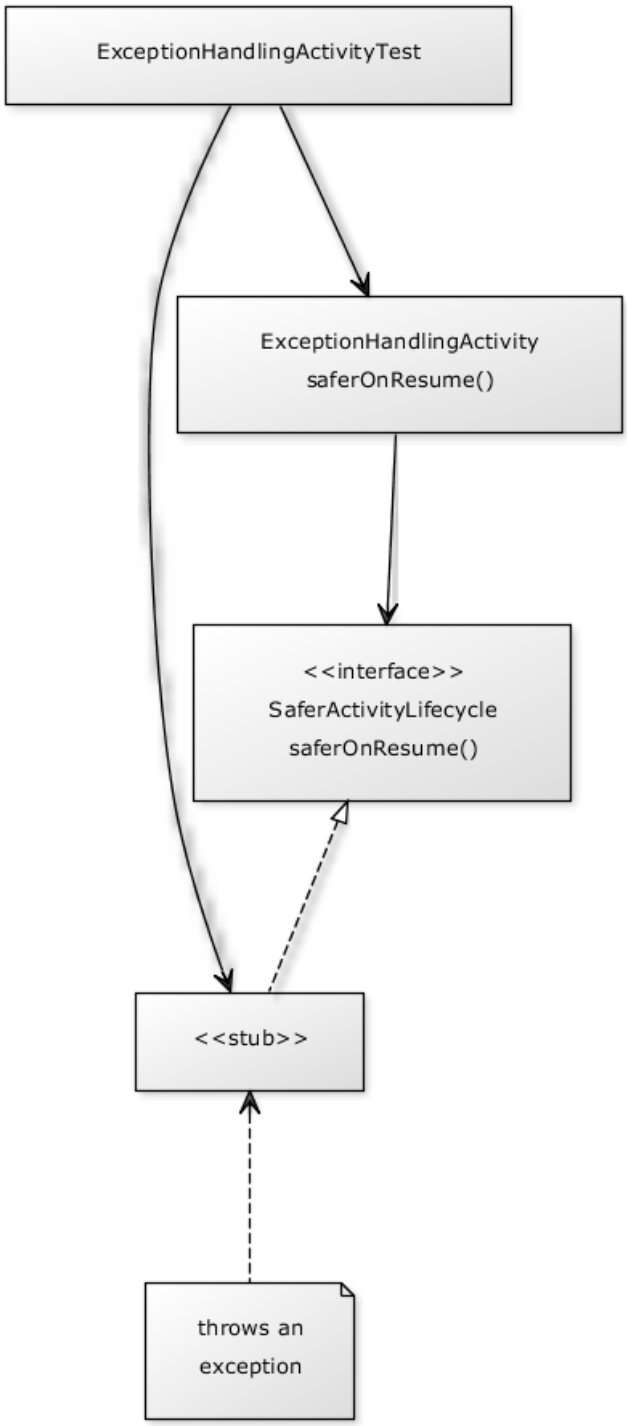
Good news: when I try to write the test, my first principles push to towards a better design.

1. Subclass `ExceptionHandlingActivity` in order to test it. Override `saferOnResume()` to throw an exception.
2. Replace Inheritance with Delegation[2], although in reverse, so that the soon-to-be-former superclass delegates to the soon-to-be-former subclass. This moves `saferOnResume()` onto a new interface. Tentatively call this `SaferActivityLifecycle`. Now each `ExceptionHandlingActivity` needs a `SaferActivityLifecycle`.
3. When adding new Activity objects, implement `SaferActivityLifecycle`, rather than extending `Activity`, and get uniform exception handling free.

[2] Refactoring, page 352.



Before, notice that the test doesn't actually use the class I want to test



After, the test uses the right class, and it avoids irrelevant details

You might recognise the mechanical principle behind this transformation: Prefer Interfaces to Abstract Classes.

You might also recognise some of the intuitive principles guiding this transformation. Referring again to Refactoring, page 352:

> A subclass uses only part of a superclass's interface or does not want to inherit data.

This superclass sounds like it has too big an interface–violating the Single Responsibility Principle–and the subclass wants to narrow that interface–applying the Interface Segregation Principle.

I've almost talked myself into applying this refactoring now, but I'd rather gather more evidence for it. Retaining the option on this refactoring costs me almost nothing and the option only expires if the code changes in a way to obviate the need for this particular improvement[3].

Too soon to implement the Model

It also feels too soon to implement the Model, nor even to write a contract test for the one method we have, so I add a CONTRACT comment to the interface in order to remind me.

Snapshot 105: Documenting an aspect of a contract

```
diff --git a/AndroidFree/src/main/java/ca/jbrains/upfp/mvp/BrowseTransactio\
nsModel.java b/AndroidFree/src/main/java/ca/jbrains/upfp/mvp/BrowseTransact\
ionsModel.java
index 2dc4889..e4aaaa9 100644
--- a/AndroidFree/src/main/java/ca/jbrains/upfp/mvp/BrowseTransactionsModel\
.java
+++ b/AndroidFree/src/main/java/ca/jbrains/upfp/mvp/BrowseTransactionsModel\
.java
@@ -1,5 +1,6 @@
 package ca.jbrains.upfp.mvp;

 public interface BrowseTransactionsModel {
+  // CONTRACT: result >= 0
   int countTransactions();
 }
```

Something new

So it's time to do something new. What would help this app become worthy of a public release?

[3] If you have never explored Real Options, then I suggest starting with Commitment: A Novel About Managing Project Risk. If nothing else, you'll find it easier to discuss design tradeoffs with business people, a skill that could come in handy.

- Export transactions to a spreadsheet-friendly format, like CSV, so that I analyse them.
- Enter transactions quickly and easily.

If the app did these two things, then I could use the app every day to record cash transactions, then send them to a spreadsheet and combine them with the transaction CSV files I download from my banks and analyse my spending however I like. This passes the Laugh Test for me, so I forge ahead.

What's done

- Snapshot 105: Added a note about the contract of an interface method.

What's left to do

- Export transactions to CSV, but where do we store the file?
- Enter a transaction quickly and easily
- Decide how Activity will handle Presenter throwing exceptions to it. Normal operation or should Presenter not throw exceptions?
- Write contract tests for Model, View once they stabilise

6 A New Feature

I like to explore new feature ideas before building them. Specifically, I want to check how much value I see in them and that I don't intend to build significantly more than I need to deliver that value. To check the value I see in a feature, I use the Magic Wand technique.[1]

This technique consists of asking a single question repeatedly: "You have it; what will you do with it?" In this case, "it" refers to the feature in question.

A One-Act Play

> **Me**: I think I want to export all my transactions to a comma-separated values (CSV) file.
>
> **Alter Ego (*waves magic wand, making a strange buzzing sound*)**: You have it; what will you do with it?
>
> **Me**: I'll import the transactions into a spreadsheet.
>
> **Alter Ego (*waves magic wand, making that buzzing sound again*)**: You have it; what will you do with it?
>
> **Me**: I'll summarise the transactions by category of income/expenditure and label each expense category with "I value spending this money", "I don't value spending this money", or "I have no choice but to spend this money".
>
> **Alter Ego (*waves magic wand, buzzing sound even more annoying*)**: You have it; what will you do with it?
>
> **Me**: For starters, I'll stop spending in the areas that I don't value! I'll be rich!

That sounds good enough to me to stop. The feature provides enough value to me that I probably won't feel like building it wasted my time. Now, could I make the feature simpler? Have I found the kernel of the feature?[2]

---

[1] One of the techniques that I teach as part of Product Sashimi.

[2] I use the term "kernel" here to mean the smallest, most essential part of a feature which, if we build it first, would allow us to build the rest of the feature in any sequence. I also teach this as part of Product Sashimi as a way to minimise the negative effects that dependent events have on a project's schedule. Dependent events, meaning activities that must happen in sequence, rather than in parallel, represent a significant risk in any project.

The Kernel of a Feature

When I look for the kernel of a feature, I use a technique called "Contract, then Expand".[3] I start with a single scenario for the feature, then simplify it ruthlessly (contracting) to arrive at the kernel, before building up interesting and useful variations (expanding) that I can consider building in virtually any sequence. This leads to a plan that allows me to change my mind at almost any time with inexpensive consequences. For exporting transactions to CSV, I start with this simple scenario.

Scenario: Export All Transactions to CSV

I say, "Export all my transactions to a CSV file."
I see, "Transactions exported to `/path/to/transactions.csv`"
I see a file at `/path/to/transactions.csv` with all my transactions in CSV format.

I don't know how to make this feature any simpler, so I think I've done enough here. The user can only export all their transactions, not a subset of them, and the application writes the file to only one place on the file system, not a place of the user's choosing. I don't see anything to simplify, not matter how ruthlessly I try to do it. I can think of many interesting variations, however.

- Apply a filter to the transactions before exporting.
- Choose where to write the file.
- Share the file, rather than writing it to the file system.

I don't want to lose these ideas, but I also don't want them to distract me, so I put them away for now. I believe I have reached the kernel of this feature. I can't think of a simpler file format, nor can I think of any simpler mechanism for exporting than to a file. With that done, I can turn my attention to more technical considerations.

What do I need to know about Android to build this?

I need to know what constraints Android will put on me before I build this feature. I can identify a few things that I need to learn about.

- How/where to store a file on the phone that the user can grab. SD card?
- How to wire up an "Export to CSV" button on the Browse Transactions Screen to invoke an event handler.

As always, I wonder how quickly can I get Android out of the picture? What's the minimum Android API that I have to use? After some digging, I decide that I need to save the CSV file to what Android

[3] Another technique that I teach as part of Product Sashimi.

calls “external storage”, and I figure out how to get the file off the “SD card” using an emulator. It’s actually pretty straightforward, although it took me a while to figure it out. Since I appear to need the emulator for this, that means some manual testing, which I’d like to minimise.[4]

1. Start the emulator (2-3 minutes).
2. Press the “Export” button.
3. Stop the emulator, because it appears to lock the SD Card disk image, at least on my computer.
4. Find the file sdcard.img in my emulator on disk and mount it.
5. Look in the right folder of the disk image for the CSV file and examine it.

The whole flow takes up to five minutes, and I find that very slow and tedious, so I’d like to minimise the number of times I have to do this. This encourages me to write as much of this as possible in the Android-Free module, which reinforces my earlier questions, “How quickly can I get Android out of the picture?”

Using Android External Storage

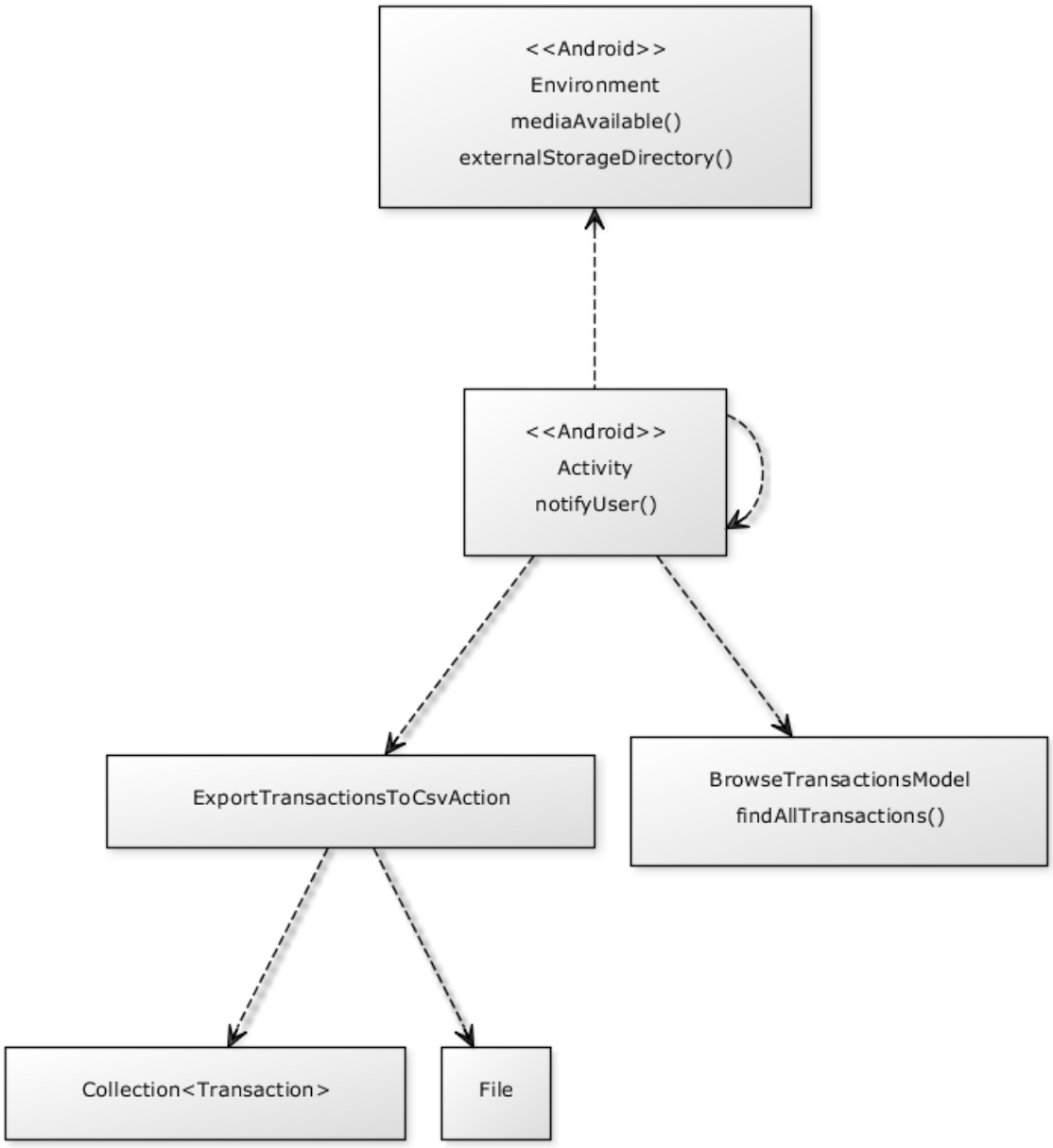
It seems simple enough:

1. Check that the media is available (it exists and I can write to it), so I have to handle the case where it isn’t.
2. Ask Android for a directory on the external storage. Arbitrarily, I choose the “Downloads” directory.
3. Write the file, which uses Java’s standard `File` API. Android-Free code!

I think I’ll roll my own CSV solution for now, because it doesn’t require much effort, but I’ll add a task to evaluate a CSV library and integrate it later. This will encourage me to isolate “format transactions as CSV” from “write text to file”, two responsibilities that have almost nothing to do with each other, anyway.

I feel like sketching this before building it. Once again, notice that the Android stuff uses the Android-Free stuff, but not the reverse.

[4] To clarify, I want to minimise *routine* manual testing–what Michael Bolton calls “checking”. If I were exploring the app, I’d have no problem doing so manually.



At the bottom, the Happy Zone; at the top, the Ugly Outside World

That's all the design I need to start writing some code. I start with the client, so that I know more about what I need to put in the Response.

Test list for the Activity, draft 1

- Happy path: media available, model returns transactions, export action succeeds
- Media not available
- Media not writable
- Model blows up

- Export action returns failure response
- Export action blows up

Before I write all these tests, I want to think through the contract between the Activity, the Model and the Export Action, to decide which failures I should treat as beyond the programmer's control and which ones I should treat as `ProgrammerMistakes`. I've summarised them in this handy table.

Classifying failures related to exporting transactions

Beyond programmer's control	Programmer Mistakes
media not available	Model returns unexpected value for transactions (example: null – the only example?)
media not writable	Data format problem trying to format transactions as CSV (example: how'd a null get in there?)
Model blows up due to underlying data storage problem	
I/O problem while writing text to file	

I do this to identify which cases to test-drive. I choose to handle the cases "beyond the programmer's control" and simply throw the `ProgrammerMistakes` up into the Activity and let it do its generic error handling, since those are mistakes that require a programmer to fix them. I revise my test list accordingly.

Test list for the Activity, draft 2

- Happy path: media available, model returns transactions, export action succeeds
  - 0 transactions
  - 1 transaction
  - a few transactions
  - many, many transactions (sanity check for stress)
- Media not available
- Media not writable
- Model throws exception
- Export Action throws exception

As an experiment, I choose to use checked exceptions, because I hate checked exceptions somewhat irrationally, and now feels like a safe time to challenge that irrational opinion. Eight tests; it shouldn't hurt too much.

I add a button to `BrowseTransactionsActivity` so that I can establish the contract between the screen's layout and the Activity: the layout expects a method on the Activity named `exportAllTransactions()`. I don't see any other constraint on the Activity's tests.

A “trivial” test

For the sake of safety, I start with a View-level test that presses the button and confirms that the Activity doesn’t blow up. It would blow up, for example, if I didn’t wire the button correctly to an Activity method.

Snapshot 110: Pressing the export button shouldn’t blow up

```
diff --git a/src/test/java/ca/jbrains/upfp/controller/android/test/RenderBr\
owseTransactionsScreenTest.java b/src/test/java/ca/jbrains/upfp/controller/\
android/test/RenderBrowseTransactionsScreenTest.java
index be20a1b..23c0d92 100644
--- a/src/test/java/ca/jbrains/upfp/controller/android/test/RenderBrowseTra\
nsactionsScreenTest.java
+++ b/src/test/java/ca/jbrains/upfp/controller/android/test/RenderBrowseTra\
nsactionsScreenTest.java
@@ -1,6 +1,7 @@
 package ca.jbrains.upfp.controller.android.test;

-import ca.jbrains.upfp.BrowseTransactionsActivity;
+import android.widget.Button;
+import ca.jbrains.upfp.*;
 import ca.jbrains.upfp.mvp.RendersView;
 import com.xtremelabs.robolectric.RobolectricTestRunner;
 import org.jmock.*;
@@ -37,4 +38,21 @@ public class RenderBrowseTransactionsScreenTest {

     mockery.assertIsSatisfied();
   }
+
+  @Test
+  public void exportAllTransactionsButtonDoesNotBlowUp()
+      throws Exception {
+    final BrowseTransactionsActivity
+        browseTransactionsActivity
+        = new BrowseTransactionsActivity();
+    browseTransactionsActivity.onCreate(null);
+
+    final Button exportAllTransactionsButton
+        = (Button) browseTransactionsActivity.findViewById(
+        R.id
+            .exportAllTransactionsButton);
```

```
+    exportAllTransactionsButton.performClick();
+
+    // don't blow up
+  }
 }
```

It seems like a simple test, just for the sake of safety, and yet it shows me something: I can't decide between the wordings "export transactions" and "export all transactions", as I use one wording for the Activity method name and one for the button ID. I notice the different when I write the test. That's why I do it. I prefer "export all transactions" for its precision, so I fix the inconsistency in the names.

Snapshot 110: Renaming the button

```
diff --git a/res/layout/main.xml b/res/layout/main.xml
index e05fa7e..986274d 100644
--- a/res/layout/main.xml
+++ b/res/layout/main.xml
@@ -46,7 +46,7 @@
             android:layout_alignParentBottom="true"
             android:layout_marginTop="10dp"
             >
-        <Button android:id="@+id/exportTransactionsButton"
+        <Button android:id="@+id/exportAllTransactionsButton"
                 android:layout_width="fill_parent"
                 android:layout_height="wrap_content"
                 android:layout_gravity="right"
```

Snapshot 110: Addng an event handler for the new button

```
diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
index a26882c..6efe40d 100644
--- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
+++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
@@ -2,6 +2,7 @@ package ca.jbrains.upfp;

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

```
+import android.view.View;
 import android.widget.TextView;
 import ca.jbrains.toolkit.ProgrammerMistake;
 import ca.jbrains.upfp.mvp.*;
@@ -61,4 +62,8 @@ public class BrowseTransactionsActivity extends Activity
         String.format(
             "%1$d", transactionCount));
   }
+
+  public void exportAllTransactions(View clicked) {
+    // Nothing yet
+  }
 }
```

Now on to the Activity tests. I don't have to worry about the details here; I'll worry about formatting transactions and writing text to file when I implement the Model and the Export Action.

Something's wrong.

As I try to write the first test I realise that I've conflated two sets of tests. Which ones? How did I discover this? I wrote the assertion first.

What would I check for the case of happy path with no transactions? I'd expect a CSV file with only headers and no transaction rows, but **how would I know**?! The Export Action takes care of that, so unless I stubbed the Export Action to do that, I couldn't check it in this test. Worse, if I did that, *I'd have tested a stub*!

Don’t test your stubs

When I teach programmers about test doubles, I generally warn them that one day they will write a test that looks like this:

```
// blah blah blah...
stub method foo() to return 23
// 27 lines of stuff...
assertEquals(23, foo())
```

This test only checks the mocking library, and **not your code**, so why do people do this? I can think of one key few force that leads otherwise thoughtful, intelligent people to write this test.

The more complicated our design, the more collaborating objects we have, and the more test doubles we need. The more test doubles we need, the bigger our tests become, and the more easily we lose track of which object we’ve implemented in the test. Somewhat ironically, the more aggressively I use test doubles, the less likely this problem occurs, because I have only one real implementation per test, so I find it easy to point to, even in a crowd.

I believe that this “mistake” represents a rite of passage for designers, rather than a sign of carelessness or stupidity. I believe that making this “mistake” represents progress: I interpret it as a sign that the programmer has tried to use test doubles aggressively, and can now see the signals that they try to broadcast–this object has too many collaborators, the interaction between these two objects requires too many messages, this object has too many responsibilities. Test doubles help me not just see these problems, but more appropriate ways to distribute responsibilities among objects.

To avoid checking a stub, this test can’t check the content of the exported file, so what can it check? What’s left to check? It could check that we’ve notified the user that exporting the transactions has worked. I find that much easier. The test could expect a nice, long `Toast` telling the user, “Exported all transactions to file BlahBlahBlah.csv”? That sounds great, and simplifies the test list considerably.

Test list for Activity, draft 3

- Happy path: media available, model returns transactions, export action succeeds
- Media not available
- Media not writable
- Model throws exception
- Export Action throws exception

Of course, I'd better put the 0, 1, many, lots tests on the test list for the Export Action, otherwise I might forget.

Test list for Export All Transactions to CSV File Action

- 0 transactions
- 1 transaction
- a few transactions
- very, very many transactions

Now, back to the Activity, now that I know what assertion to write.

Happy Path

Snapshot 120: The Happy Path test

```
1  package ca.jbrains.upfp.controller.android.test;
2
3  import ca.jbrains.upfp.*;
4  import ca.jbrains.upfp.controller.ExportAllTransactionsAction;
5  import ca.jbrains.upfp.mvp.BrowseTransactionsModel;
6  import com.google.common.collect.Lists;
7  import com.xtremelabs.robolectric.RobolectricTestRunner;
8  import com.xtremelabs.robolectric.shadows.*;
9  import org.jmock.*;
10 import org.junit.Test;
11 import org.junit.runner.RunWith;
12
13 import java.util.Collection;
14 import java.util.regex.Pattern;
15
16 import static ca.jbrains.hamcrest.RegexMatcher.matches;
17 import static org.junit.Assert.assertThat;
18
19 @RunWith(RobolectricTestRunner.class)
20 public class HandleExportAllTransactionsTest {
21   private Mockery mockery = new Mockery();
22
23   @Test
24   public void happyPath() throws Exception {
```

```
25      final BrowseTransactionsModel browseTransactionsModel
26          = mockery.mock(BrowseTransactionsModel.class);
27      final ExportAllTransactionsAction
28          exportAllTransactionsAction = mockery.mock(
29          ExportAllTransactionsAction.class);
30
31      final Collection<Object>
32          anyValidNonTrivialCollectionOfTransactions = Lists
33          .newArrayList(
34              new Object(), new Object(),
35              new Object());
36
37      mockery.checking(
38          new Expectations() {{
39            allowing(browseTransactionsModel)
40                .findAllTransactions();
41            will(
42                returnValue(
43                    anyValidNonTrivialCollectionOfTransactions));
44
45            allowing(exportAllTransactionsAction).execute();
46            // succeeds by not throwing an exception
47          }});
48
49      final BrowseTransactionsActivity
50          browseTransactionsActivity
51          = new BrowseTransactionsActivity
52          (null, exportAllTransactionsAction);
53      browseTransactionsActivity.onCreate(null);
54
55      browseTransactionsActivity.exportAllTransactions(
56          browseTransactionsActivity.findViewById(
57              R.id.exportAllTransactionsButton));
58
59      ShadowHandler.idleMainLooper();
60      assertThat(
61          ShadowToast.getTextOfLatestToast(),
62          matches(
63              Pattern.compile(
64                  "Exported all transactions to (.+)\\.csv")));
65    }
66  }
```

This test looks quite long, but I expect that from tests that involve the Android libraries, which further supports my desire to avoid them as much as I can. In short: when exporting transactions succeeds, the Activity should notify the user that it has succeeded.

In retrospect, I made this test far too complicated. Rather than checking directly for the `Toast`, I could have expected something like `notifyUser("Exported all transactions...")` on another View interface that an Android-dependent class would implement to display a `Toast`. While I lament an opportunity lost, as soon as I want to do the same thing from multiple parts of the system, I'll refactor towards that design, anyway.

I make this test pass by simply displaying the `Toast`, although I also have to add a new constructor and delegate the old one to it.

Snapshot 120: Display a `Toast`, and add some wiring

```
diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
index 99c8ec0..7cd5975 100644
--- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
+++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
@@ -3,9 +3,13 @@ package ca.jbrains.upfp;
 import android.app.Activity;
 import android.os.Bundle;
 import android.view.View;
-import android.widget.TextView;
+import android.widget.*;
 import ca.jbrains.toolkit.ProgrammerMistake;
+import ca.jbrains.upfp.controller.ExportAllTransactionsAction;
 import ca.jbrains.upfp.mvp.*;
+import com.google.common.collect.Lists;
+
+import java.util.Collection;

 public class BrowseTransactionsActivity extends Activity
     implements BrowseTransactionsView {
@@ -21,12 +25,28 @@ public class BrowseTransactionsActivity extends Activit\
y
           public int countTransactions() {
             return 12;
```

```
25              }
26  +
27  +           @Override
28  +           public Collection<Object> findAllTransactions() {
29  +             return Lists.newArrayList();
30  +           }
31            }, this);
32      }
33
34  +   /**
35  +    * @deprecated
36  +    */
37      public BrowseTransactionsActivity(
38          RendersView rendersView
39      ) {
40  +     this(rendersView, null);
41  +   }
42  +
43  +   public BrowseTransactionsActivity(
44  +       RendersView rendersView, ExportAllTransactionsAction
45  +       exportAllTransactionsAction
46  +   ) {
47  +
48        this.rendersView = rendersView;
49      }
50
51  @@ -66,6 +86,9 @@ public class BrowseTransactionsActivity extends Activity
52      }
53
54      public void exportAllTransactions(View clicked) {
55  -     // Nothing yet
56  +     Toast.makeText(
57  +         getApplicationContext(),
58  +         "Exported all transactions to /mnt/sdcard/TrackEveryPenny.csv",
59  +         Toast.LENGTH_LONG).show();
60      }
61   }
```

At the same time, I create one interface and enhance another.

Snapshot 120: A new interface for the export action

```
package ca.jbrains.upfp.controller;

public interface ExportAllTransactionsAction {
  void execute();
}
```

Snapshot 120: Now the Model can find all the transactions

```
diff --git a/AndroidFree/src/main/java/ca/jbrains/upfp/mvp/BrowseTransactio\
nsModel.java b/AndroidFree/src/main/java/ca/jbrains/upfp/mvp/BrowseTransact\
ionsModel.java
index e4aaaa9..9dedad7 100644
--- a/AndroidFree/src/main/java/ca/jbrains/upfp/mvp/BrowseTransactionsModel\
.java
+++ b/AndroidFree/src/main/java/ca/jbrains/upfp/mvp/BrowseTransactionsModel\
.java
@@ -1,6 +1,10 @@
 package ca.jbrains.upfp.mvp;

+import java.util.Collection;
+
 public interface BrowseTransactionsModel {
   // CONTRACT: result >= 0
   int countTransactions();
+
+  Collection<Object> findAllTransactions();
 }
```

That takes care of the first test.

One down, four to go

- DONE. Happy path: media available, model returns transactions, export action succeeds
- Media not available
- Media not writable
- Model throws exception

- Export Action throws exception

Now for the cases where the media is not available or not writable, how far should I go? We have to use the Android Environment, and I have no idea how to set up `Environment` to make the external storage media either unavailable or available-but-not-writable, so I stub it with an interface with the ugly name `AndroidDeviceFileSystemGateway`.[5] This is a highly structural name, rather than an intention-revealing name, so I've chosen a precise name.[6] I expect to implement this with an `EnvironmentBasedAndroidDeviceFileSystemGateway`. These names feel weird, and it feels especially weird putting the word "Android" in the name of an interface, but I can easily change that later as the lines between "Android device" and "Generic mobile device" become more clear, as I strongly hope they will.

I encounter this particular kind of design tension quite often when adding tests to legacy code, as I'd label the Android SDK. When a class presents low cohesion from too many responsibilities in one place, I find applying the Interface Segregation Principle to it awkward at first. I don't quite know what to name things. I start by extracting too many or too few interfaces, though over time, the design tends to stabilise to something sensible. This stability tends to emerge when I follow these steps.

1. Extract interface from the Horrible Gelatinous Class.
2. Write collaboration tests that stub and set expectations on the new interface.
3. Eventually try to inject the Horrible Gelatinous Class into itself through the constructor, which Java helpfully doesn't allow
4. Extract the implementation of this interface into a new, smaller class, to which the Horrible Gelatinous Class delegates.

I love this trick for one key reason: I don't have to worry about *when* to split the Horrible Gelatinous Class, because the compiler forces me to do it at what would otherwise be the First Irresponsible Moment. This trick applies Real Options to design.

Now that I look at the code again, when I chose to make `BrowseTransactionsActivity` implement `BrowseTransactionsView`, I created exactly this situation. I add an item to the backlog to extract the View from the Activity.

[5] I use the term "Gateway" the way Martin Fowler did in Principles of Enterprise Application Architecture

[6] These terms refer to a model I use for improving the names of things, like classes, methods and variables. You can read more here.

A reader has alerted me to the notion that an Activity *has* Views, rather than *being* a View, and that he knows this as a "best practice" of Android development. I interpret this as a weak signal that extracting the View from the Activity represents a positive step. Some of you might interpret this as an idiot stumbling in the dark. I'd like to address that here.

Remember that I'm learning the Android SDK as I go, and I *want* not to know about so-called "best practices" in the Android community… yet. I want the chance to discover for myself the design issues related to integrating with Android. If you have experience with Android, then you might find this frustrating at times, but please go with me. If I eventually refactor towards a pattern that you know well, then that encourages me to pay attention to that pattern, and having a name for it will almost certainly help me. On the other hand, if I don't need some pattern that you consider essential to good Android design, then I have wonder what makes it essential. Certainly it could mean that I haven't tried to build something that you've tried to build, so I simply haven't yet needed the pattern, but it could also point to that pattern as unnecessary, at least under certain conditions.

I prefer to discover these things on my own. This fits better with the way I learn, and particularly it helps me understand these things much more deeply.

Before writing the next test, I notice that this test doesn't need the Activity to pass the transactions-to-be-exported into the Export Transactions Action. Yes, that looks sloppy, and yes, that feels weird, and yes, I like it, anyway. If no test forces me to connect those two things together, then I've made a mistake somewhere, and this helps me see that quite clearly.

The "media not available" case

As I write the next test, I decide that I now *like* the name of the Gateway interface. The duplication between the interface name and the method name tells me that I have segregated the interface to the maximum, which I favor for new interfaces.[7]

In the process of making this new test pass, the first test fails, because it doesn't stub `androidDevicePublicStorageGa`
I don't like having to add such a detail to a test, so I mark this as a `SMELL`[8]. I interpret this as a weak signal that `exportAllTransactions()` already has too many responsibilities. I wait for more evidence before changing anything, so I stub `androidDevicePublicStorageGateway.findPublicExternalStorageDirectory`
to not throw an exception in the happy path test.

[7] I think this amounts to personal style. I prefer to build tiny things, then combine them later, whereas other people prefer to build bigger things, then split them later. I haven't found a universal, objective argument in favor of either approach.

[8] Some people sprinkle their code with `FIXME` or `TODO` comments. I use `SMELL` and `REFACTOR` in a similar fashion. If I find or write code that I don't like, but don't know how to improve it, I mark it with `SMELL`. If I find or write code that I know how to refactor, but don't want to do it right now, I mark it with `REFACTOR`. I keep an eye on the number of these in a code base, and don't let them grow out of hand.

Snapshot 130: Extracting fixture objects and adding a new test

```
diff --git a/src/test/java/ca/jbrains/upfp/controller/android/test/HandleEx\
portAllTransactionsTest.java b/src/test/java/ca/jbrains/upfp/controller/and\
roid/test/HandleExportAllTransactionsTest.java
index 034b01e..a6f022d 100644
--- a/src/test/java/ca/jbrains/upfp/controller/android/test/HandleExportAll\
TransactionsTest.java
+++ b/src/test/java/ca/jbrains/upfp/controller/android/test/HandleExportAll\
TransactionsTest.java
@@ -2,12 +2,13 @@ package ca.jbrains.upfp.controller.android.test;

 import ca.jbrains.upfp.*;
 import ca.jbrains.upfp.controller.ExportAllTransactionsAction;
+import ca.jbrains.upfp.controller.android.*;
 import ca.jbrains.upfp.mvp.BrowseTransactionsModel;
 import com.google.common.collect.Lists;
 import com.xtremelabs.robolectric.RobolectricTestRunner;
 import com.xtremelabs.robolectric.shadows.*;
 import org.jmock.*;
-import org.junit.Test;
+import org.junit.*;
 import org.junit.runner.RunWith;

 import java.util.Collection;
@@ -19,15 +20,30 @@ import static org.junit.Assert.assertThat;
 @RunWith(RobolectricTestRunner.class)
 public class HandleExportAllTransactionsTest {
   private Mockery mockery = new Mockery();
+  private final BrowseTransactionsModel
+      browseTransactionsModel = mockery.mock(
+      BrowseTransactionsModel.class);
+  private final ExportAllTransactionsAction
+      exportAllTransactionsAction = mockery.mock(
+      ExportAllTransactionsAction.class);
+  private final AndroidDevicePublicStorageGateway
+      androidDevicePublicStorageGateway = mockery.mock(
+      AndroidDevicePublicStorageGateway.class);
+
+  private final BrowseTransactionsActivity
+      browseTransactionsActivity
+      = new BrowseTransactionsActivity(
```

```
+      null, exportAllTransactionsAction,
+      androidDevicePublicStorageGateway
+  );
+
+  @Before
+  public void initializeActivity() {
+    browseTransactionsActivity.onCreate(null);
+  }

   @Test
   public void happyPath() throws Exception {
-    final BrowseTransactionsModel browseTransactionsModel
-        = mockery.mock(BrowseTransactionsModel.class);
-    final ExportAllTransactionsAction
-        exportAllTransactionsAction = mockery.mock(
-        ExportAllTransactionsAction.class);
-
     final Collection<Object>
         anyValidNonTrivialCollectionOfTransactions = Lists
         .newArrayList(
@@ -42,25 +58,60 @@ public class HandleExportAllTransactionsTest {
                 returnValue(
                     anyValidNonTrivialCollectionOfTransactions));

+          // SMELL Irrelevant detail
+          ignoring(androidDevicePublicStorageGateway)
+              .findPublicExternalStorageDirectory();
+
           allowing(exportAllTransactionsAction).execute();
           // succeeds by not throwing an exception
         }});

-    final BrowseTransactionsActivity
-        browseTransactionsActivity
-        = new BrowseTransactionsActivity
-        (null, exportAllTransactionsAction);
-    browseTransactionsActivity.onCreate(null);
+    pressExportAllTransactionsButton(
+        browseTransactionsActivity);
+
+    assertLastToastMatchesRegex(
+        "Exported all transactions to (.+)\\.csv");
```

```
+  }
+
+  @Test
+  public void mediaNotAvailable() throws Exception {
+    mockery.checking(
+        new Expectations() {{
+          allowing(browseTransactionsModel)
+              .findAllTransactions();
+
+          allowing(androidDevicePublicStorageGateway)
+              .findPublicExternalStorageDirectory();
+          will(
+              throwException(
+                  new PublicStorageMediaNotAvailableException()));
+
+          never(exportAllTransactionsAction);
+        }});
+
+    pressExportAllTransactionsButton(
+        browseTransactionsActivity);
+
+    assertLastToastMatchesRegex(
+        "No place to which to export the transactions. " +
+        "Insert an SD card or connect an " +
+        "external storage device and try again.");
+  }
+
+  private void pressExportAllTransactionsButton(
+      BrowseTransactionsActivity browseTransactionsActivity
+  ) {
     browseTransactionsActivity.exportAllTransactions(
         browseTransactionsActivity.findViewById(
             R.id.exportAllTransactionsButton));
+  }

+  public static void assertLastToastMatchesRegex(
+      String patternText
+  ) {
     ShadowHandler.idleMainLooper();
     assertThat(
         ShadowToast.getTextOfLatestToast(),
-        matches(
```

```
-            Pattern.compile(
-                "Exported all transactions to (.+)\\.csv")));
+        matches(Pattern.compile(patternText)));
    }
 }
```

Also, in the process of making this new test pass, I encounter a `NullPointerException` in `RenderBrowseTransactionsScreenTest.exportAllTransactionsButtonDoesNotBlowUp()` because I hadn't set the field `androidDevicePublicStorageGateway` in the Activity. This "upvotes" the need to delegate the Activity's `BrowseTransactionsView` behavior, which would let me chain Activity's constructors, which in turn would fix this problem. For now, I add a `SMELL` comment.

Finally, after making this new test pass, I write the "media not available" error to Android's log. YAGNI?[9] Maybe. As a novice Android programmer, I err on the side of caution here.

Snapshot 130: Handling the case where media is not available

```
diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
index 7cd5975..62094b9 100644
--- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
+++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
@@ -2,23 +2,30 @@ package ca.jbrains.upfp;

 import android.app.Activity;
 import android.os.Bundle;
+import android.util.Log;
 import android.view.View;
 import android.widget.*;
 import ca.jbrains.toolkit.ProgrammerMistake;
 import ca.jbrains.upfp.controller.ExportAllTransactionsAction;
+import ca.jbrains.upfp.controller.android.*;
 import ca.jbrains.upfp.mvp.*;
 import com.google.common.collect.Lists;

+import java.io.File;
 import java.util.Collection;

 public class BrowseTransactionsActivity extends Activity
     implements BrowseTransactionsView {
```

[9] "You aren't gonna need it", a central design principle favoring simplicity.

```
24     private final RendersView rendersView;
25   + private final AndroidDevicePublicStorageGateway
26   +     androidDevicePublicStorageGateway;
27
28     public BrowseTransactionsActivity() {
29       // We can't chain the constructor, because the instance
30       // in the process of being created is itself the view.
31       // We have to wait for super() to be (implicitly) invoked.
32   +
33   +   // REFACTOR Delegate BrowseTransactionsView behavior to a new class
34       this.rendersView = new BrowseTransactionsPresenter(
35           new BrowseTransactionsModel() {
36             @Override
37   @@ -31,6 +38,18 @@ public class BrowseTransactionsActivity extends Activity
38               return Lists.newArrayList();
39             }
40           }, this);
41   +
42   +   // SMELL I have to initialize this because I can't use
43   +   // constructor chaining yet. This has to be anything
44   +   // that won't throw a stupid exception.
45   +   this.androidDevicePublicStorageGateway
46   +       = new AndroidDevicePublicStorageGateway() {
47   +     @Override
48   +     public File findPublicExternalStorageDirectory()
49   +         throws PublicStorageMediaNotAvailableException {
50   +       return new File(".");
51   +     }
52   +   };
53     }
54
55     /**
56   @@ -39,15 +58,29 @@ public class BrowseTransactionsActivity extends Activit\
57   y
58     public BrowseTransactionsActivity(
59         RendersView rendersView
60     ) {
61   -   this(rendersView, null);
62   +   this(rendersView, null, null);
63     }
64
65   + /**
```

```
66  +   * @deprecated
67  +   */
68      public BrowseTransactionsActivity(
69          RendersView rendersView, ExportAllTransactionsAction
70          exportAllTransactionsAction
71      ) {
72
73  +     this(rendersView, exportAllTransactionsAction, null);
74  +   }
75  +
76  +   public BrowseTransactionsActivity(
77  +       RendersView rendersView,
78  +       ExportAllTransactionsAction exportAllTransactionsAction,
79  +       AndroidDevicePublicStorageGateway androidDevicePublicStorageGateway
80  +   ) {
81  +
82        this.rendersView = rendersView;
83  +     this.androidDevicePublicStorageGateway
84  +         = androidDevicePublicStorageGateway;
85      }
86
87      @Override
88  @@ -86,9 +119,23 @@ public class BrowseTransactionsActivity extends Activit\
89  y
90      }
91
92      public void exportAllTransactions(View clicked) {
93  -     Toast.makeText(
94  -         getApplicationContext(),
95  -         "Exported all transactions to /mnt/sdcard/TrackEveryPenny.csv",
96  -         Toast.LENGTH_LONG).show();
97  +     try {
98  +       androidDevicePublicStorageGateway
99  +           .findPublicExternalStorageDirectory();
100 +       Toast.makeText(
101 +           getApplicationContext(),
102 +           "Exported all transactions to /mnt/sdcard/TrackEveryPenny.csv",
103 +           Toast.LENGTH_LONG).show();
104 +     } catch (PublicStorageMediaNotAvailableException reported) {
105 +       Log.e(
106 +           "TrackEveryPenny",
107 +           "Couldn't save a file to public storage; media not available",
```

```
+          reported);
+      Toast.makeText(
+          getApplicationContext(),
+          "No place to which to export the transactions. Insert an SD card\
 or connect an " +
+          "external storage device and try again.",
+          Toast.LENGTH_LONG).show();
+    }
   }
 }
```

Test list for the Activity

- **DONE.** Happy path: media available, model returns transactions, export action succeeds
- **DONE.** Media not available
- Media not writable
- Model throws exception
- Export Action throws exception

The “media not writable” case

Making the third test pass results in some duplication, which makes me wince[10], but doesn’t bother me enough to refactor it away. I log the error exactly as before, putting a little more pressure on the decision to handle exceptions inside the Activity.

Snapshot 140: Similar to the ‘media not available’ case

```
diff --git a/src/test/java/ca/jbrains/upfp/controller/android/test/HandleEx\
portAllTransactionsTest.java b/src/test/java/ca/jbrains/upfp/controller/and\
roid/test/HandleExportAllTransactionsTest.java
index a6f022d..83326a5 100644
--- a/src/test/java/ca/jbrains/upfp/controller/android/test/HandleExportAll\
TransactionsTest.java
+++ b/src/test/java/ca/jbrains/upfp/controller/android/test/HandleExportAll\
TransactionsTest.java
@@ -11,6 +11,7 @@ import org.jmock.*;
 import org.junit.*;
```

[10] Don Roberts gave Martin Fowler this guideline, which he describes in Refactoring: Improving the Design of Existing Code on page 58: “The first time you do something, you just do it. The second time you do something similar, you wince at the dupication, but you do the duplicate thing anyway. The third time you do something similar, you refactor.”

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

+import java.io.File;
 import java.util.Collection;
 import java.util.regex.Pattern;

@@ -98,6 +99,32 @@ public class HandleExportAllTransactionsTest {
         "external storage device and try again.");
   }

+  @Test
+  public void mediaNotWritable() throws Exception {
+    mockery.checking(
+        new Expectations() {{
+          allowing(browseTransactionsModel)
+              .findAllTransactions();
+
+          allowing(androidDevicePublicStorageGateway)
+              .findPublicExternalStorageDirectory();
+          will(
+              throwException(
+                  new PublicStorageMediaNotWritableException(
+                      new File(
+                          "/mnt/sdcard/TrackEveryPenny.csv"))));
+
+          never(exportAllTransactionsAction);
+        }});
+
+    pressExportAllTransactionsButton(
+        browseTransactionsActivity);
+
+    assertLastToastMatchesRegex(
+        "Permission denied trying to export the transactions to file " +
+        "/mnt/sdcard/TrackEveryPenny.csv");
+  }
+
   private void pressExportAllTransactionsButton(
       BrowseTransactionsActivity browseTransactionsActivity
   ) {
```

Test list for the Activity

- **DONE**. Happy path: media available, model returns transactions, export action succeeds
- **DONE**. Media not available
- **DONE**. Media not writable
- Model throws exception
- Export Action throws exception

The “Model throws exception” case

Making the fourth test pass leads me to consider what, other than a `ProgrammerMistake`, would cause the Model to throw an exception. I find only one reasonable cause: an error in the underlying storage mechanism. Since I haven’t implemented the model yet, this can’t happen, and so I could invoke YAGNI here and defer handling this case, but it will probably cost me more to track this task, then remember to do it at the appropriate time than it will cost to build it now, so I build it. I feel somewhat ill at ease making a decision based on knowledge of an intended implementation of an interface, namely that the Model will use internal storage on some kind of disk. On the other hand, I would feel perhaps more ill at ease pretending that reading important user state (from somewhere) will never fail, so perhaps I’ve managed to avoid forgetting something important, even though I don’t feel entirely comfortable with how I got here.

Snapshot 150: The new, passing test

```
diff --git a/src/test/java/ca/jbrains/upfp/controller/android/test/HandleEx\
portAllTransactionsTest.java b/src/test/java/ca/jbrains/upfp/controller/and\
roid/test/HandleExportAllTransactionsTest.java
index 83326a5..6f04526 100644
--- a/src/test/java/ca/jbrains/upfp/controller/android/test/HandleExportAll\
TransactionsTest.java
+++ b/src/test/java/ca/jbrains/upfp/controller/android/test/HandleExportAll\
TransactionsTest.java
@@ -35,7 +35,8 @@ public class HandleExportAllTransactionsTest {
        browseTransactionsActivity
        = new BrowseTransactionsActivity(
        null, exportAllTransactionsAction,
-       androidDevicePublicStorageGateway
+       androidDevicePublicStorageGateway,
+       browseTransactionsModel
    );

    @Before
```

```
@@ -125,6 +126,27 @@ public class HandleExportAllTransactionsTest {
          "/mnt/sdcard/TrackEveryPenny.csv");
    }

+  @Test
+  public void modelBlowsUpInAnUnavoidableWay()
+      throws Exception {
+    mockery.checking(
+        new Expectations() {{
+          allowing(browseTransactionsModel)
+              .findAllTransactions();
+          will(
+              throwException(
+                  new InternalStorageException()));
+        }});
+
+    pressExportAllTransactionsButton(
+        browseTransactionsActivity);
+
+    assertLastToastMatchesRegex(
+        "Something strange just happened. Try again. You might need to rei\
nstall the " +
+        "application. I feel embarrassed and ashamed.");
+  }
+
+
    private void pressExportAllTransactionsButton(
        BrowseTransactionsActivity browseTransactionsActivity
    ) {
```

Snapshot 150: `BrowseTransactionsActivity` **also now needs** `BrowseTransactionsModel` **to do its work**

```
diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
index 183a6bf..e55d1f8 100644
--- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
+++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
@@ -19,6 +19,8 @@ public class BrowseTransactionsActivity extends Activity
    private final RendersView rendersView;
    private final AndroidDevicePublicStorageGateway
        androidDevicePublicStorageGateway;
```

```
+  private final BrowseTransactionsModel
+      browseTransactionsModel;

   public BrowseTransactionsActivity() {
     // We can't chain the constructor, because the instance
@@ -42,6 +44,19 @@ public class BrowseTransactionsActivity extends Activity
     // SMELL I have to initialize this because I can't use
     // constructor chaining yet. This has to be anything
     // that won't throw a stupid exception.
+    this.browseTransactionsModel
+        = new BrowseTransactionsModel() {
+      @Override
+      public int countTransactions() {
+        return 0;
+      }
+
+      @Override
+      public Collection<Object> findAllTransactions() {
+        return null;  //To change body of implemented methods use File | S\
ettings | File Templates.
+      }
+    };
+
     this.androidDevicePublicStorageGateway
         = new AndroidDevicePublicStorageGateway() {
       @Override
@@ -58,29 +73,20 @@ public class BrowseTransactionsActivity extends Activit\
y
   public BrowseTransactionsActivity(
       RendersView rendersView
   ) {
-    this(rendersView, null, null);
-  }
-
-  /**
-   * @deprecated
-   */
-  public BrowseTransactionsActivity(
-      RendersView rendersView, ExportAllTransactionsAction
-      exportAllTransactionsAction
-  ) {
-
```

```
-    this(rendersView, exportAllTransactionsAction, null);
+    this(rendersView, null, null, null);
   }

   public BrowseTransactionsActivity(
       RendersView rendersView,
       ExportAllTransactionsAction exportAllTransactionsAction,
-      AndroidDevicePublicStorageGateway androidDevicePublicStorageGateway
+      AndroidDevicePublicStorageGateway androidDevicePublicStorageGateway,
+      BrowseTransactionsModel browseTransactionsModel
   ) {

     this.rendersView = rendersView;
     this.androidDevicePublicStorageGateway
         = androidDevicePublicStorageGateway;
+    this.browseTransactionsModel = browseTransactionsModel;
   }

   @Override
@@ -120,12 +126,20 @@ public class BrowseTransactionsActivity extends Activ\
ity

   public void exportAllTransactions(View clicked) {
     try {
+      browseTransactionsModel.findAllTransactions();
       androidDevicePublicStorageGateway
           .findPublicExternalStorageDirectory();
       Toast.makeText(
           getApplicationContext(),
           "Exported all transactions to /mnt/sdcard/TrackEveryPenny.csv",
           Toast.LENGTH_LONG).show();
+    } catch (InternalStorageException reported) {
+      Log.wtf("TrackEveryPenny", reported);
+      Toast.makeText(
+          getApplicationContext(),
+          "Something strange just happened. Try again. You might need to "\
 +
+          "reinstall the application. I feel embarrassed and ashamed.",
+          Toast.LENGTH_LONG).show();
     } catch (PublicStorageMediaNotAvailableException reported) {
       Log.e(
           "TrackEveryPenny",
```

I notice that I don't log the errors consistently, so I fix that.

Snapshot 160: Handling errors more consistently

```
diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
index e55d1f8..c39466b 100644
--- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
+++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
@@ -155,8 +155,8 @@ public class BrowseTransactionsActivity extends Activit\
y
                .getPathNotWritable().getAbsolutePath();
        Log.e(
                "TrackEveryPenny", String.format(
-               "Path %1$s not writable",
-               pathNotWritableAsText));
+               "Path %1$s not writable", pathNotWritableAsText),
+               reported);
        Toast.makeText(
                getApplicationContext(),
                String.format(
```

Test list for the Activity

- **DONE**. Happy path: media available, model returns transactions, export action succeeds
- **DONE**. Media not available
- **DONE**. Media not writable
- **DONE**. Model throws exception
- Export Action throws exception

The "Export Action throws exception" case

Writing this test gives me a tingle of doubt. How can this Export Action fail? I can see I/O failures writing text to the external storage, because that uses the `File` API, so it makes perfect sense to anticipate `IOExceptions`. I like to use standard library classes when I can, but this violates one of my irrationally-held principle: don't use checked exceptions.[11] In addition, if I let the Export Action throw an `IOException` here, then I feel like I expose a dependency on the `File` API, and

[11] I don't push this principle, but I tend to follow it. If you and I work on a team and you insist on using checked exceptions, I won't press the matter. I've seen checked exceptions lead to incredible messes, because checked exceptions create hardwired dependencies on concrete classes from every method that might throw one. That said, I see value in *occasionally* forcing clients to handle exceptions.

I'd rather not, although as I write these words, they don't convince me: `IOException` does not necessarily imply the `File` API and it seems sufficiently abstract for my purposes here. Even so, given my aversion to checked exceptions, I let Export Action throw an `InternalStorageException` when it fails, matching the way the Model already fails. If I find `IOException` more appropriate later, then I can change this decision then. Moreover, this challenges my design: if changing a method to throw a checked exception introduces significant ripple-through changes, then either that affirms my dislike of checked exceptions, or it highlights a serious problem with the design. Either way, I think I win. As a way of rationalising my choice, I notice that catching `IOException` in `exportAllTransactions()` would introduce mixed levels of abstraction, so I feel a little better about using `InternalStorageException`.

Snapshot 170: The new test

```
diff --git a/src/test/java/ca/jbrains/upfp/controller/android/test/HandleEx\
portAllTransactionsTest.java b/src/test/java/ca/jbrains/upfp/controller/and\
roid/test/HandleExportAllTransactionsTest.java
index 6f04526..759a2af 100644
--- a/src/test/java/ca/jbrains/upfp/controller/android/test/HandleExportAll\
TransactionsTest.java
+++ b/src/test/java/ca/jbrains/upfp/controller/android/test/HandleExportAll\
TransactionsTest.java
@@ -146,6 +146,30 @@ public class HandleExportAllTransactionsTest {
          "application. I feel embarrassed and ashamed.");
    }

+  @Test
+  public void exportActionBlowsUpInAnUnavoidableWay()
+      throws Exception {
+    mockery.checking(
+        new Expectations() {{
+          // SMELL How can I ignore these irrelevant details?
+          ignoring(browseTransactionsModel);
+          ignoring(androidDevicePublicStorageGateway);
+
+          allowing(exportAllTransactionsAction)
+              .execute();
+          will(
+              throwException(
+                  new InternalStorageException()));
+        }});
+
+    pressExportAllTransactionsButton(
```

```
+        browseTransactionsActivity);
+
+    assertLastToastMatchesRegex(
+        "Something strange just happened. Try again. You might need to rei\
nstall the " +
+        "application. I feel embarrassed and ashamed.");
+  }
+

   private void pressExportAllTransactionsButton(
       BrowseTransactionsActivity browseTransactionsActivity
```

To make this test pass, I introduce in `BrowseTransactionsActivity` a new constructor parameter for the `ExportAllTransactionsAction` instance. This forces me to implement the Export Action with a no-op in order to retain the no-argument constructor that Android will invoke, but avoid a potentially annoying `NullPointerException` in the future.

Snapshot 170: BrowseTransactionsActivity now uses ExportAllTransactionsAction

```
diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
index c39466b..fca7f17 100644
--- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
+++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
@@ -17,6 +17,8 @@ import java.util.Collection;
 public class BrowseTransactionsActivity extends Activity
     implements BrowseTransactionsView {
   private final RendersView rendersView;
+  private final ExportAllTransactionsAction
+      exportAllTransactionsAction;
   private final AndroidDevicePublicStorageGateway
       androidDevicePublicStorageGateway;
   private final BrowseTransactionsModel
@@ -41,6 +43,14 @@ public class BrowseTransactionsActivity extends Activity
           }
         }, this);

+    this.exportAllTransactionsAction
+        = new ExportAllTransactionsAction() {
+      @Override
+      public void execute() {
```

```
+        // Do nothing, for now
+      }
+    };
+
     // SMELL I have to initialize this because I can't use
     // constructor chaining yet. This has to be anything
     // that won't throw a stupid exception.
@@ -84,6 +94,8 @@ public class BrowseTransactionsActivity extends Activity
    ) {

     this.rendersView = rendersView;
+    this.exportAllTransactionsAction
+        = exportAllTransactionsAction;
     this.androidDevicePublicStorageGateway
         = androidDevicePublicStorageGateway;
     this.browseTransactionsModel = browseTransactionsModel;
@@ -129,6 +141,7 @@ public class BrowseTransactionsActivit\
y
       browseTransactionsModel.findAllTransactions();
       androidDevicePublicStorageGateway
           .findPublicExternalStorageDirectory();
+      exportAllTransactionsAction.execute();
       Toast.makeText(
           getApplicationContext(),
           "Exported all transactions to /mnt/sdcard/TrackEveryPenny.csv",
```

In making this test pass, I've introduced quite a lot of duplication. Each exception handler logs the error and notifies the user with some text. I can't see a great way to remove the duplication, so I'll try to mechanically apply the fundamental refactorings I know to see where that leads the design. I'll set a timer for 15 minutes, and if I haven't discovered anything interesting in that time, I'll roll the code back and move on.

Refactoring the exception handlers

First, I remove duplication in the logging statements by extracting methods `logError()` and `wtf()`.

Snapshot 180: Removing duplication of 'TrackEveryPenny' in logging statements

```
diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
index fca7f17..1beff5a 100644
--- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
+++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
@@ -147,15 +147,14 @@ public class BrowseTransactionsActivity extends Activ\
ity
            "Exported all transactions to /mnt/sdcard/TrackEveryPenny.csv",
            Toast.LENGTH_LONG).show();
      } catch (InternalStorageException reported) {
-       Log.wtf("TrackEveryPenny", reported);
+       wtf(reported);
        Toast.makeText(
            getApplicationContext(),
            "Something strange just happened. Try again. You might need to "\
 +
            "reinstall the application. I feel embarrassed and ashamed.",
            Toast.LENGTH_LONG).show();
      } catch (PublicStorageMediaNotAvailableException reported) {
-       Log.e(
-           "TrackEveryPenny",
+       logError(
            "Couldn't save a file to public storage; media not available",
            reported);
        Toast.makeText(
@@ -166,10 +165,10 @@ public class BrowseTransactionsActivity extends Activ\
ity
      } catch (PublicStorageMediaNotWritableException reported) {
        final String pathNotWritableAsText = reported
            .getPathNotWritable().getAbsolutePath();
-       Log.e(
-           "TrackEveryPenny", String.format(
-           "Path %1$s not writable", pathNotWritableAsText),
-           reported);
+       logError(
+           String.format(
+               "Path %1$s not writable",
+               pathNotWritableAsText), reported);
        Toast.makeText(
            getApplicationContext(),
```

```
41              String.format(
42  @@ -178,4 +177,14 @@ public class BrowseTransactionsActivity extends Activi\
43  ty
44          ).show();
45        }
46      }
47  +
48  +   private void logError(
49  +       String message, Throwable reported
50  +   ) {
51  +     Log.e("TrackEveryPenny", message, reported);
52  +   }
53  +
54  +   private void wtf(Throwable reported) {
55  +     Log.wtf("TrackEveryPenny", reported);
56  +   }
57   }
```

Next, I remove duplication in displaying `Toasts` by extracting method `notifyUser()`, which sounds like a candidate to go onto a View-level interface.

Snapshot 190: Removing duplicate uses of Toast

```
1   diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
2   b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
3   index 1beff5a..b3b81ee 100644
4   --- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
5   +++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
6   @@ -142,26 +142,20 @@ public class BrowseTransactionsActivity extends Activ\
7   ity
8           androidDevicePublicStorageGateway
9               .findPublicExternalStorageDirectory();
10          exportAllTransactionsAction.execute();
11  -       Toast.makeText(
12  -           getApplicationContext(),
13  -           "Exported all transactions to /mnt/sdcard/TrackEveryPenny.csv",
14  -           Toast.LENGTH_LONG).show();
15  +       notifyUser(
16  +           "Exported all transactions to /mnt/sdcard/TrackEveryPenny.csv");
17      } catch (InternalStorageException reported) {
18          wtf(reported);
```

```
-      Toast.makeText(
-          getApplicationContext(),
+      notifyUser(
           "Something strange just happened. Try again. You might need to "\
 +
-          "reinstall the application. I feel embarrassed and ashamed.",
-          Toast.LENGTH_LONG).show();
+          "reinstall the application. I feel embarrassed and ashamed.");
     } catch (PublicStorageMediaNotAvailableException reported) {
       logError(
           "Couldn't save a file to public storage; media not available",
           reported);
-      Toast.makeText(
-          getApplicationContext(),
+      notifyUser(
           "No place to which to export the transactions. Insert an SD card\
 or connect an " +
-          "external storage device and try again.",
-          Toast.LENGTH_LONG).show();
+          "external storage device and try again.");
     } catch (PublicStorageMediaNotWritableException reported) {
       final String pathNotWritableAsText = reported
           .getPathNotWritable().getAbsolutePath();
@@ -169,15 +163,19 @@ public class BrowseTransactionsActivity extends Activ\
ity
           String.format(
               "Path %1$s not writable",
               pathNotWritableAsText), reported);
-      Toast.makeText(
-          getApplicationContext(),
+      notifyUser(
           String.format(
               "Permission denied trying to export the transactions to file\
 %1$s",
-              pathNotWritableAsText), Toast.LENGTH_LONG
-      ).show();
+              pathNotWritableAsText));
     }
   }

+  private void notifyUser(String message) {
+    Toast.makeText(
```

```
+        getApplicationContext(), message, Toast.LENGTH_LONG)
+        .show();
+  }
+
   private void logError(
       String message, Throwable reported
   ) {
```

When removing duplication, I try to make similar things as similar as possible, so I'd like to use `logError()` instead of `wtf()`. I can't think of a good reason not to do this, so I do it.

Snapshot 200: Removing wtf()

```
diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
index b3b81ee..398b47d 100644
--- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
+++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
@@ -145,7 +145,9 @@ public class BrowseTransactionsActivity extends Activit\
y
       notifyUser(
           "Exported all transactions to /mnt/sdcard/TrackEveryPenny.csv");
     } catch (InternalStorageException reported) {
-      wtf(reported);
+      logError(
+          "Couldn't read data from internal storage",
+          reported);
       notifyUser(
           "Something strange just happened. Try again. You might need to "\
 +
           "reinstall the application. I feel embarrassed and ashamed.");
@@ -181,8 +183,4 @@ public class BrowseTransactionsActivity extends Activit\
y
   ) {
     Log.e("TrackEveryPenny", message, reported);
   }
-
-  private void wtf(Throwable reported) {
-    Log.wtf("TrackEveryPenny", reported);
-  }
 }
```

These changes make higher-level duplication easier to spot. Instead of invoking `logError()`, then `notifyUser()` three times, I extract method `handleError()` for that. I don't really believe this refactoring, but I trust that the rules will lead me somewhere good.

Snapshot 210: Removing duplication in exception handlers

```
diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
index 398b47d..8e0ef54 100644
--- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
+++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
@@ -145,33 +145,39 @@ public class BrowseTransactionsActivity extends Activ\
ity
        notifyUser(
            "Exported all transactions to /mnt/sdcard/TrackEveryPenny.csv");
      } catch (InternalStorageException reported) {
-       logError(
+       handleError(
            "Couldn't read data from internal storage",
-           reported);
-       notifyUser(
            "Something strange just happened. Try again. You might need to "\
 +
-           "reinstall the application. I feel embarrassed and ashamed.");
+           "reinstall the application. I feel embarrassed and ashamed.",
+           reported);
      } catch (PublicStorageMediaNotAvailableException reported) {
-       logError(
+       handleError(
            "Couldn't save a file to public storage; media not available",
-           reported);
-       notifyUser(
            "No place to which to export the transactions. Insert an SD card\
 or connect an " +
-           "external storage device and try again.");
+           "external storage device and try again.",
+           reported);
      } catch (PublicStorageMediaNotWritableException reported) {
        final String pathNotWritableAsText = reported
            .getPathNotWritable().getAbsolutePath();
-       logError(
+       handleError(
```

```
37              String.format(
38                  "Path %1$s not writable",
39  -               pathNotWritableAsText), reported);
40  -       notifyUser(
41  +               pathNotWritableAsText),
42              String.format(
43                  "Permission denied trying to export the transactions to file\
44   %1$s",
45  -               pathNotWritableAsText));
46  +               pathNotWritableAsText),
47  +           reported);
48        }
49      }
50
51  +   private void handleError(
52  +       String internalMessage, String userVisibleMessage,
53  +       Throwable cause
54  +   ) {
55  +     logError(internalMessage, cause);
56  +     notifyUser(userVisibleMessage);
57  +   }
58  +
59      private void notifyUser(String message) {
60        Toast.makeText(
61            getApplicationContext(), message, Toast.LENGTH_LONG)
```

Next, I reformat the code–I won't bore you with the diff–then mark the newly-extracted methods with the ways I think I can now reuse them. "Anywhere in this app" really means anywhere in the *Android-dependent* module of this app.

Snapshot 220: Marked the reusability of some methods

```
1  diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
2  b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
3  index 8e0ef54..9c74c02 100644
4  --- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
5  +++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
6  @@ -170,6 +170,7 @@ public class BrowseTransactionsActivity extends Activit\
7  y
8        }
9      }
```

```
10
11 +  // REUSE Any Activity
12    private void handleError(
13        String internalMessage, String userVisibleMessage,
14        Throwable cause
15 @@ -178,12 +179,14 @@ public class BrowseTransactionsActivity extends Activ\
16 ity
17      notifyUser(userVisibleMessage);
18    }
19
20 +  // REUSE Any Activity
21    private void notifyUser(String message) {
22      Toast.makeText(
23          getApplicationContext(), message, Toast.LENGTH_LONG)
24          .show();
25    }
26
27 +  // REUSE Anywhere in this app
28    private void logError(
29        String message, Throwable reported
30    ) {
```

This refactoring hasn't led to anything significant yet, but when I add the REUSE markers, I realise that I need to write another 2-3 Activity classes before I can benefit much from the duplication I've just removed. Moreover, since Android chose implementation inheritance as their framework extension mechanism–which already violates my principle Avoid Inheriting Implementation–my first step to reuse this behavior will probably involve extracting a superclass, which violates this principle enough more. In order to avoid going prematuraly down a potential blind alley, I fight the urge to immediately replace inheritance with delegation, since I don't know enough about the built-in behavior of an Activity to do that confidently.

Also, looking at the implementation of `BrowseTransactionsActivity.exportAllTransactions()`, it looks strange to invoke these three methods in succession, ignore their return values, and not use the output from the first two to invoke the third.

Snapshot 220: A strange implementation of exportAllTransactions()

```
package ca.jbrains.upfp;

public class BrowseTransactionsActivity
    extends Activity
    implements BrowseTransactionsView {

  // ...
  public void exportAllTransactions(View clicked) {
    try {
      browseTransactionsModel.findAllTransactions();
      androidDevicePublicStorageGateway
          .findPublicExternalStorageDirectory();
      exportAllTransactionsAction.execute();
      notifyUser(
          "Exported all transactions to /mnt/sdcard/TrackEveryPenny.csv");
    } catch // ...and so on
    }
  }
  // ...
}
```

I can get away with this *so far* because the tests describe the Activity's behavior in a highly abstract way. This looks sloppy, but I find it freeing, even though I needed time to get used to it. Implementing these intefaces will force these methods to demand input, and that will force `exportAllTransactions()` to wire its three collaborating methods together. This kind of tactic annoys some programmers, who worry that practising TDD will encourage them to forget what they know and hack rather than think. I disagree. This roundabout technique helps me verify my assumptions about the design I need. Suppose implementing those interfaces *doesn't* force me to wire these three collaborating methods together the way I'd expected. That, friends, is information I want to have. For my minor transgressions against YAGNI, I hope you see that I'm following it here where it figures more prominently.

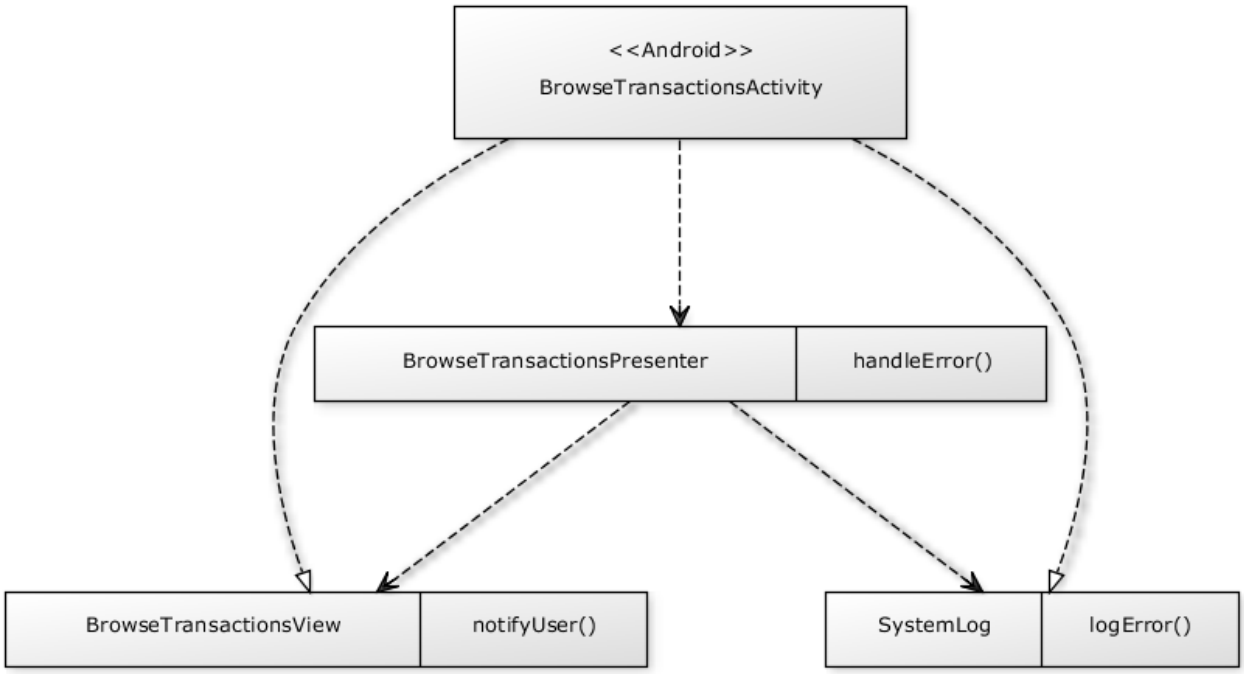
Wait just a minute!

As I rearranged items on the backlog to prepare for the next session, I realised something rather important: I had implemented Presenter behavior directly in the Activity, clearly violating the Single Responsibility Principle. Oddly enough, Robolectric made it *easier* for me to make this mistake, which makes me wonder how much of a mistake it really is. It felt really quite natural to write the code this way, and I can't justify feeling bad about that, except as some abstract notion of correctness, so I'm going to leave it for now.

Thinking more, though, I see that the methods I've recently extracted – `notifyUser()`, `logError()` and `handleError()` – could all end up on useful interfaces that my Android adapter layer would implement. This would remove a significant amount of code from my Activity classes, and thinner framework adapter layers tend to reduce the cost of changing that code over time. I don't feel ready to change this yet, but I'll remember this, and the moment it starts to get in my way, I'll change it then. In the meantime, I add a `REFACTOR` comment and draw a diagram to describe the design I have in mind.

Snapshot 230: A feeling about separating unrelated behavior

```
diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
index 9c74c02..1baec62 100644
--- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
+++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
@@ -138,6 +138,7 @@ public class BrowseTransactionsActivity extends Activit\
y

   public void exportAllTransactions(View clicked) {
     try {
+      // REFACTOR Shouldn't a presenter be doing this?
       browseTransactionsModel.findAllTransactions();
       androidDevicePublicStorageGateway
           .findPublicExternalStorageDirectory();
```



How I intend to separate Activity and Presenter from each other

What's done

It's a lot.

- Snapshot 110: Added an 'export all transactions' button that does nothing useful yet.
- Snapshot 120: The Activity notifies the user of success when everything succeeds.
- Snapshot 130: The Activity handles the case where the public external storage media is not available.
- Snapshot 140: Activity now gracefully handles the external storage media not being writable.
- Snapshot 150: The Activity now gracefully handles an internal storage problem when reading from the model.
- Snapshot 160: Added a parameter to make exception handlers more consistent with each other.
- Snapshot 170: The Activity now gracefully handles the export action failing.
- Snapshot 180: Removed duplication of "TrackEveryPenny" in logging statements.
- Snapshot 190: Introduced method notifyUser() to remove duplication in displaying Toast messages to the user.
- Snapshot 200: Stopped using wtf() where it wasn't appropriate.
- Snapshot 210: Removed duplication in exception handlers.
- Snapshot 220: Marked methods with REUSE levels.
- Snapshot 230: Marked code in the Activity that should perhaps be in the Presenter.

What's left to do

- (2 upvotes) Delegate the Activity's BrowseTransactionsView implementation to a new class
- Implement Model
  - Add to Model contract: countTransactions() >= 0
- Implement ExportAllTransactionsAction
- Export transactions to CSV, but where do we store the file?
- Write contract tests for Model, View once they stabilise
- Enter a transaction quickly and easily

7 Implementing the Model

Now feels like a good time for something a little easier, so I want to implement the Model. I expect nothing interesting to come from this, but Java has presented me with a conundrum having to do with the contract for the Model interface. I collect the contract for an interface by looking for all the stubs and expectations on that interface in my tests.

- `findAllTransactions()`
  - returns a non-null, non-empty collection of `Object` ("valid" meant non-null and "non-trivial" meant non-empty)
  - throws `InternalStorageException`
- `countTransactions()`
  - returns any number (nothing was special about `0`)

I infer a contract for the Model from these examples. Also, I've previously mentioned that `countTransactions()` must return a non-negative number, so I add that to the contract. I also notice an invariant to add to the contract: `countTransactions()` needs to answer the size of whatever `findAllTransactions()` answers.[^abstract-class] Before adding this invariant to the contract, I check for stubs that more violate it. I find none. I could have stubbed `countTransactions()` to return `-1` in a test, but I didn't, so I can safely add this invariant to the contract.

Contract for `BrowseTransactionsModel` (draft version 1)

- `findAllTransactions()` does exactly one of these:
  - returns a non-null collection of `Object` (eventually this will be a "transaction", but I'm not there yet)
  - throws `InternalStorageException`
- `countTransactions() >= 0`
- `findAllTransactions().size() == countTransactions()`

A simple contract, completely intuitive, so why spend so much time writing about it? I see a hole–the kind of hole that leads to a mistake deep in code that eventually hijacks a team for the hours or days it takes to track the mistake down. Specifically: what should `countTransactions()` do when `findAllTransactions()` throws `InternalStorageException`? If a client can't rely on `findAllTransactions()` to return a value, but tries to rely on `countTransactions()` to return a value, there will be problems.

Law of Demeter or Poka-yoke?

The Law of Demeter[1] nudges our design in the direction of objects knowing less about their collaborators. The principle of Poka-yoke[2] encourages us to design things to eliminate failure. In this case, these principles clash. Clients of the Model that use `countTransactions()` don't have to know about `findAllTransactions()`, but on the other hand, a poorly implemented Model could make these methods behave inconsistently, creating a potential failure where otherwise none would exist. Should I remove `countTransactions()` or add to the contract that it must fail exactly as `findAllTransactions()` fails whenever the latter fails? I can't find an objective basis on which to decide.

Arbitrarily, I choose to retain `countTransactions()` and make it fail exactly as `findAllTransactions()`.

Contract for `BrowseTransactionsModel` (draft version 2)

- `findAllTransactions()` does exactly one of these:
  - returns a non-null collection of `Object` (eventually this will be a "transaction", but I'm not there yet)
  - throws `InternalStorageException`
- `countTransactions() >= 0`
- `findAllTransactions().size() == countTransactions()`
- `countTransactions()` throws the same exceptions in the same conditions as `findAllTransactions()`

This contract feels sufficient to implement. This provides a simple–you might think it's "trivial", a word that I wish programmers would throw around less easily–example of writing contract tests. Since I don't already have contract tests for this interface, I test-drive the implementation, then extract the contract tests into an abstract superclass. This documents the contract well and gives future implementers a clear place to start.

[1] Read more at http://www.ccs.neu.edu/home/lieber/LoD.html
[2] Read more at http://facultyweb.berry.edu/jgrout/pokasoft.html

Snapshot 240: The tests for an in-memory model

```
package ca.jbrains.upfp.model.test;

import ca.jbrains.upfp.model.InternalStorageException;
import com.google.common.collect.Lists;
import org.junit.Test;

import java.util.*;

import static org.junit.Assert.*;

public class InMemoryBrowseTransactionsModelTest {
  @Test
  public void zeroTransactions() throws Exception {
    final InMemoryBrowseTransactionsModel model
        = new InMemoryBrowseTransactionsModel(
        Lists.newArrayList());
    assertEquals(
        Collections.emptyList(),
        model.findAllTransactions());
    assertEquals(0, model.countTransactions());
  }

  @Test
  public void manyTransactions() throws Exception {
    final Collection<Object> transactions = Lists
        .newArrayList(
            new Object(), new Object(), new Object());
    final InMemoryBrowseTransactionsModel model
        = new InMemoryBrowseTransactionsModel(transactions);
    assertEquals(transactions, model.findAllTransactions());
    assertEquals(3, model.countTransactions());
  }

  @Test
  public void findAllTransactionsFails() throws Exception {
    final InMemoryBrowseTransactionsModel model
        = new InMemoryBrowseTransactionsModel(null) {
      @Override
      public Collection<Object> findAllTransactions() {
        throw new InternalStorageException();
```

```
41        }
42      };
43
44      try {
45        model.countTransactions();
46        fail(
47            "How did you count the transactions when you can't find them?!");
48      } catch (InternalStorageException success) {
49      }
50    }
51  }
```

Snapshot 240: The implementation that passes these tests

```
1   package ca.jbrains.upfp.model.test;
2
3   import ca.jbrains.upfp.mvp.BrowseTransactionsModel;
4
5   import java.util.Collection;
6
7   public class InMemoryBrowseTransactionsModel
8       implements BrowseTransactionsModel {
9     private final Collection<Object> transactions;
10
11    public InMemoryBrowseTransactionsModel(
12        Collection<Object> transactions
13    ) {
14      this.transactions = transactions;
15    }
16
17    @Override
18    public int countTransactions() {
19      // Invoking findAllTransactions() instead of using the
20      // field directly ensures that the model throws
21      // exceptions consistently when it fails.
22      return findAllTransactions().size();
23    }
24
25    @Override
26    public Collection<Object> findAllTransactions() {
27      return transactions;
```

```
28     }
29 }
```

I also move the class `InternalStorageException` into the `model` package, since the Model now uses it, and I won't let the (Android-Free) Model depend on the (Android-Dependent) Controller. Of course, I could move `ExportAllTransactionsAction` into the Android-Free module's controller package, but then within that module Model and Controller would depend on each other. Since Controller naturally depends on Model, I move `InternalStorageException` directly into the Model. Besides–this class describes a Model exception, so it belongs either in the Model or in a package upon which the Model depends.[3]

Extracting contract tests

After test-driving the implementation, I extract contract tests by pulling up into a new abstract superclass any test that describes a part of the contract. I generally follow these steps:

1. Choose a test that describes an aspect of the contract.
2. Generalise the type of the subject of the test so that I can store it as a reference to the interface, rather than as a reference to the implementation. This usually requires introducing a method to create the instance in the desired state. By this point, the test should make no reference to the implementation it tests, but rather only to the interface whose contract it helps describe.
3. Pull the test method up into an `abstract` superclass, declaring as `abstract` any creation method I have introduced. Marking the class `abstract` prevents the test runner from trying to instantiate it, which would lead to a runtime error.

This time, I chose to execute the steps for the entire test class, rather than test by test. I did this to avoid burying you in code samples. If you paired with me, I'd insist on doing this test by test.

Snapshot 250: Generalising the type of `InMemoryBrowseTransactionsModel`

```
1  diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/InMemoryB\
2  rowseTransactionsModelTest.java b/AndroidFree/src/test/java/ca/jbrains/upfp\
3  /model/test/InMemoryBrowseTransactionsModelTest.java
4  index 72f94ec..3e8464d 100644
5  --- a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/InMemoryBrowseTr\
6  ansactionsModelTest.java
7  +++ b/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/InMemoryBrowseTr\
```

[3] Whenever I can justify a decision with both mechanics (following low-level rules) and intuition (following high-level principles or hand-waving logic), I feel better. When mechanics and intuition support each other this way, that points to a solid decision. Read more at http://www.jbrains.ca/permalink/becoming-an-accomplished-software-designer

```
ansactionsModelTest.java
@@ -1,6 +1,7 @@
 package ca.jbrains.upfp.model.test;

 import ca.jbrains.upfp.model.InternalStorageException;
+import ca.jbrains.upfp.mvp.BrowseTransactionsModel;
 import com.google.common.collect.Lists;
 import org.junit.Test;

@@ -11,8 +12,8 @@ import static org.junit.Assert.*;
 public class InMemoryBrowseTransactionsModelTest {
   @Test
   public void zeroTransactions() throws Exception {
-    final InMemoryBrowseTransactionsModel model
-        = new InMemoryBrowseTransactionsModel(
+    final BrowseTransactionsModel model
+        = createBrowseTransactionModelWith(
         Lists.newArrayList());
     assertEquals(
         Collections.emptyList(),
@@ -26,20 +27,16 @@ public class InMemoryBrowseTransactionsModelTest {
         .newArrayList(
             new Object(), new Object(), new Object());
     final InMemoryBrowseTransactionsModel model
-        = new InMemoryBrowseTransactionsModel(transactions);
+        = createBrowseTransactionModelWith(transactions);
     assertEquals(transactions, model.findAllTransactions());
     assertEquals(3, model.countTransactions());
   }

   @Test
   public void findAllTransactionsFails() throws Exception {
-    final InMemoryBrowseTransactionsModel model
-        = new InMemoryBrowseTransactionsModel(null) {
-      @Override
-      public Collection<Object> findAllTransactions() {
-        throw new InternalStorageException();
-      }
-    };
+    final BrowseTransactionsModel model
+        = createFailingBrowseTransactionsModel(
+        new InternalStorageException());
```

```
50
51      try {
52        model.countTransactions();
53  @@ -48,4 +45,23 @@ public class InMemoryBrowseTransactionsModelTest {
54      } catch (InternalStorageException success) {
55      }
56    }
57  +
58  +  private InMemoryBrowseTransactionsModel createBrowseTransactionModelWith\
59  (
60  +      Collection<Object> transactions
61  +  ) {
62  +    return new InMemoryBrowseTransactionsModel(
63  +        transactions);
64  +  }
65  +
66  +  private InMemoryBrowseTransactionsModel createFailingBrowseTransactionsM\
67  odel(
68  +      final RuntimeException intentionalException
69  +  ) {
70  +
71  +    return new InMemoryBrowseTransactionsModel(null) {
72  +      @Override
73  +      public Collection<Object> findAllTransactions() {
74  +        throw intentionalException;
75  +      }
76  +    };
77  +  }
78   }
```

Snapshot 260: The new contract tests

```
1  package ca.jbrains.upfp.model.test;
2
3  import ca.jbrains.upfp.model.InternalStorageException;
4  import ca.jbrains.upfp.mvp.BrowseTransactionsModel;
5  import com.google.common.collect.Lists;
6  import org.junit.Test;
7
8  import java.util.*;
9
```

```
10  import static org.junit.Assert.*;
11
12  public abstract class BrowseTransactionsModelContract {
13    @Test
14    public void zeroTransactions() throws Exception {
15      final BrowseTransactionsModel model
16          = createBrowseTransactionModelWith(
17          Lists.newArrayList());
18      assertEquals(
19          Collections.emptyList(),
20          model.findAllTransactions());
21      assertEquals(0, model.countTransactions());
22    }
23
24    @Test
25    public void manyTransactions() throws Exception {
26      final Collection<Object> transactions = Lists
27          .newArrayList(
28              new Object(), new Object(), new Object());
29      final InMemoryBrowseTransactionsModel model
30          = createBrowseTransactionModelWith(transactions);
31      assertEquals(transactions, model.findAllTransactions());
32      assertEquals(3, model.countTransactions());
33    }
34
35    @Test
36    public void findAllTransactionsFails() throws Exception {
37      final BrowseTransactionsModel model
38          = createFailingBrowseTransactionsModel(
39          new InternalStorageException());
40
41      try {
42        model.countTransactions();
43        fail(
44            "How did you count the transactions when you can't find them?!");
45      } catch (InternalStorageException success) {
46      }
47    }
48
49    protected abstract InMemoryBrowseTransactionsModel
50    createBrowseTransactionModelWith(
51        Collection<Object> transactions
```

```
52    );
53
54    protected abstract InMemoryBrowseTransactionsModel
55    createFailingBrowseTransactionsModel(
56        RuntimeException intentionalException
57    );
58  }
```

Snapshot 260: What's left in the implementation details tests

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/InMemoryB\
rowseTransactionsModelTest.java b/AndroidFree/src/test/java/ca/jbrains/upfp\
/model/test/InMemoryBrowseTransactionsModelTest.java
index 3e8464d..ba9cc1a 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/InMemoryBrowseTr\
ansactionsModelTest.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/InMemoryBrowseTr\
ansactionsModelTest.java
@@ -1,59 +1,19 @@
 package ca.jbrains.upfp.model.test;

-import ca.jbrains.upfp.model.InternalStorageException;
-import ca.jbrains.upfp.mvp.BrowseTransactionsModel;
-import com.google.common.collect.Lists;
-import org.junit.Test;
+import java.util.Collection;

-import java.util.*;
-
-import static org.junit.Assert.*;
-
-public class InMemoryBrowseTransactionsModelTest {
-  @Test
-  public void zeroTransactions() throws Exception {
-    final BrowseTransactionsModel model
-        = createBrowseTransactionModelWith(
-        Lists.newArrayList());
-    assertEquals(
-        Collections.emptyList(),
-        model.findAllTransactions());
-    assertEquals(0, model.countTransactions());
```

```
32  -  }
33  -
34  -  @Test
35  -  public void manyTransactions() throws Exception {
36  -    final Collection<Object> transactions = Lists
37  -        .newArrayList(
38  -            new Object(), new Object(), new Object());
39  -    final InMemoryBrowseTransactionsModel model
40  -        = createBrowseTransactionModelWith(transactions);
41  -    assertEquals(transactions, model.findAllTransactions());
42  -    assertEquals(3, model.countTransactions());
43  -  }
44  -
45  -  @Test
46  -  public void findAllTransactionsFails() throws Exception {
47  -    final BrowseTransactionsModel model
48  -        = createFailingBrowseTransactionsModel(
49  -        new InternalStorageException());
50  -
51  -    try {
52  -      model.countTransactions();
53  -      fail(
54  -          "How did you count the transactions when you can't find them?!")\
55  ;
56  -    } catch (InternalStorageException success) {
57  -    }
58  -  }
59  -
60  -  private InMemoryBrowseTransactionsModel createBrowseTransactionModelWith\
61  (
62  +public class InMemoryBrowseTransactionsModelTest
63  +    extends BrowseTransactionsModelContract {
64  +  @Override
65  +  protected InMemoryBrowseTransactionsModel createBrowseTransactionModelWi\
66  th(
67         Collection<Object> transactions
68      ) {
69        return new InMemoryBrowseTransactionsModel(
70            transactions);
71      }
72
73  -  private InMemoryBrowseTransactionsModel createFailingBrowseTransactionsM\
```

```
74 odel(
75 +  @Override
76 +  protected InMemoryBrowseTransactionsModel createFailingBrowseTransaction\
77 sModel(
78         final RuntimeException intentionalException
79     ) {
80
```

While editing this chapter, I noticed that `InMemoryBrowseTransactionsModelTest` still has some references to `InMemoryBrowseTransactionsModel`. Oops. I should have generalised the return type to `BrowseTransactionsModel`. I add that to my list of clean-up work and move on. I apologise for that oversight. If you'd paired with me on this, we probably wouldn't have missed it.

When I look at the contract tests again, I don't immediately understand the meaning of `createFailingBrowseTransa` so I rename it to

```
createBrowseTransactionsModelWhereFindAllTransactionsFailsWith(exception)
```

This makes the name longer, but that doesn't bother me. A long name that I immediately understand saves me more time than a shorter name that I have to think about.

Fixing a dependency problem

Before declaring this task completed, I check that types ended up in sensible packages. I stumble open problems with classes unrelated to the work from this session, then fix them. This follows my principle Fix Problems Before They Become Serious. I made these changes:

- Split package `ca.jbrains.upfp.mvp` into `ca.jbrains.upfp.model`, `ca.jbrains.upfp.view` and `ca.jbrains.upfp.presenter`
- Merge package `ca.jbrains.upfp.controller` into `ca.jbrains.upfp.presenter`
- Move `RenderYouHaveNTransactionsOnBrowseTransactionsScreenTest` into a test package—at least I'd put it in the test source tree!

What's done

- Snapshot 240:
  - BrowseTransactionsModel works only with 0 transactions.
  - BrowseTransactionsModel now handles any number of transactions.
  - BrowseTransactionsModel now handles failure consistently.
- Snapshot 250: Removed references from the test methods to the implementation of BrowseTransactionsModel.
- Snapshot 260: Extracted contract tests for BrowseTransactionsModel.
- Cleaning up:
  - Clarified a name.
  - Moved classes into more sensible packages. I'd worked a bit sloppily.

What's left to do

- Implement ExportAllTransactionsAction
- Export transactions to CSV, but where do we store the file?
- Write contract tests for View once they stabilise
- Enter a transaction quickly and easily
- (2 upvotes) Delegate the Activity's BrowseTransactionsView implementation to a new class

8 Implementing the Export All Transactions Action

Now for the fun part: implementing the Export All Transactions action interface with as little knowledge of the existence of Android as possible. I expect this to drive out some interfaces that I will find useful in adding more features, but I promise not to force it. The design goes where the design goes.

Exploring the feature with tests

First, what makes this implementation special? Two things: CSV format and saving content to file. Next, which tests will I probably need?

Test list for Export All Transactions To CSV on File (draft version 1)

Usually, the first draft of a test list amounts to simply getting everything out of my head, so it have some rough edges.

- 0 transactions
- 1 transaction
- several transactions
- many, many transactions
- disk space problems?
- generic failure to write the file (IOException)
- invalid transaction somewhere
  - null
  - missing a key property

This looks like a good start, but when I make this more precise, some duplication stands out.

Test list for Export All Transactions To CSV on File (draft version 2)

- 0 transactions, write file successfully
- 1 transaction, write file successfully
- several transactions, write file successfully

- many, many transactions, write file successfully (how slow is it?)
- many, many transactions, fail to write file
  - not enough disk space
  - generic I/O failure
- invalid transaction somewhere, can't even try writing the file
  - null
  - missing a key property

Scribbling these into a table gave me two ideas:

- a separate set of tests for the case where a transaction is invalid (null or missing a key property)
- splitting the remaining tests in two dimensions:
  - 0, 1, a few, many transactions
  - write file OK, not enough space on disk, generic I/O failure

As for the case "many, many transactions", I find nothing special about it any more. Whether the file writes successfully or not does not depend on how many transactions are in the collection, although it might depend on how much storage remains available on the device. The two points seems mostly orthogonal, so it definitely makes sense to test-drive them separately. All together now, I see three sets of tests.

Test list for Export All Transactions To CSV on File (draft version 3)

- Writing the contents out to file
  - happy path
  - not enough space on disk
  - generic I/O failure
- Typical collection tests
  - 0 transactions
  - 1 transaction
  - a few transactions
  - many transactions (stress test?)
- Invalid data
  - null transaction somewhere in the collection
  - transactions somewhere in the collection missing a key property

I find this easier to understand, but some questions remain. I've identified some cases that I need to test, but for *which behavior* do I need to check those cases? Clearly I have multiple responsibilities in this test list. What could possibly go wrong if we have a collection of 5 objects and the third one is null? I suppose converting that object into a row of text would be the problem: we don't want a blank line in the middle of the CSV file, as that would likely confuse anyone using the resulting

data. The same goes for a transaction with a key missing property in it, like the date of transaction or its amount. A blank value for the amount of a transaction defeats the whole purpose of the feature: analysing spending habits. That analysis requires three key pieces of information for every transaction:

- amount, to have something to count
- date, to enable summarising spending by date
- category, to enable summarising spending by category

Test list for Export All Transactions To CSV on File (draft version 4)

Now I present the list, complete with each interesting separate behavior and the special cases for each of those.

- Writing the contents out to file
  - happy path
  - not enough space on disk
  - generic I/O failure
- Formatting transactions as CSV file
  - 0 transactions
  - 1 transaction
  - a few transactions
  - many transactions (stress test?)
- Formatting a transaction as a CSV row
  - happy path
  - null
  - missing a key property

Well… there is one more thing. This test list encourages me to build the pieces, but not to put them together. One more time!

Test list for Export All Transactions To CSV on File (draft version 5)

- Writing the contents out to file
  - happy path
  - not enough space on disk
  - generic I/O failure
- Formatting transactions as CSV file
  - 0 transactions
  - 1 transaction
  - a few transactions

    - many transactions (stress test?)
- Formatting a transaction as a CSV row
    - happy path
    - null
    - missing a key property
- Putting it all together
    - happy path
    - formatting a transaction fails
    - formatting the entire text fails, probably by running out of memory
    - writing the text to file fails

This looks like enough to get started, and quite possibly enough to finish. Maybe I won't run into anything unexpected this time. I'm going to build this one from the parts to the whole, which will force me to flesh out the flow of data when I put it all together.

Formatting a Transaction as CSV

Arbitrarily, I start with CSV. Normally I split this into two parts: test-drive the loop and stub/mock formatting a single row, then test-drive formatting a single row. This works, but feels very procedural and is therefore no fun. Instead, I'd like to do this with some functional flair, so once I can format a single row, I think I shall `join()` the rows together with a header to form the document. For this reason, I choose to start by formatting a single transaction as a row of CSV.

Test list for Format Transaction as Row of CSV (draft version 1)

- Happy path
- Date missing
- Amount missing
- Category missing

Hm. That seems wrong somehow. If these three properties *define* a transaction, then I shouldn't be able to create a Transaction object without them. This means that by the time I format a Transaction object as CSV, there will only be happy paths, but that creating a Transaction might fail.

Test list for Format Transaction as Row of CSV (draft version 2)

- Happy path

Test list for new Transaction (draft version 1)

- All essential values provided (Happy Path)
- Date missing
- Amount missing
- Category missing

With my first test I have a mixture of Primitive Obsession[1] and Whole Value[2]. I could absolutely store the transaction's amount in cents, but I already know where that goes[3], so I save myself the side-trip. I could do the same for the transaction's category, but for now I treat it as nothing but text… oh, but wait! The category name can neither be null nor empty, so it's already time to model it as a Whole Value. Nevermind, then.

Snapshot 270: Creating a valid Transaction

```
package ca.jbrains.upfp.model.test;

import org.joda.time.LocalDate;
import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class CreateTransactionTest {
  @Test
  public void provideAllKeyInformation() throws Exception {
    final LocalDate anyNonNullDate = new LocalDate();
    final Category anyNonNullCategory = new Category(
        "irrelevant category name");
    final Amount anyNonNullAmount = Amount.cents(20);

    final Transaction transaction = new Transaction(
        anyNonNullDate, anyNonNullCategory,
        anyNonNullAmount);

    // Construction doesn't blow up
    assertEquals(anyNonNullDate, transaction.getDate());
```

[1] http://link.jbrains.ca/12F96Ug

[2] http://c2.com/ppr/checks.html#1

[3] Since Java doesn't let us attach functions to primitives, shortly after using a primitive to represent a value, I find I've extracted 3-5 utility functions that operate on that value. These end up strewn about the code base, or on my better days, gathered in a "utility class". Eventually I want to stub one of these operations, which encourages me to introduce an interface implemented by a class that delegates its instance methods to the utility functions… that's what I mean. At the absolute latest, after introducing the third utility function, I just introduce a Whole Value class to get there sooner.

```
    assertEquals(
        anyNonNullCategory, transaction.getCategory());
    assertEquals(anyNonNullAmount, transaction.getAmount());
  }
}
```

Snapshot 270: Transaction

```
package ca.jbrains.upfp.model.test;

import org.joda.time.LocalDate;

public class Transaction {
  private final LocalDate date;
  private final Category category;
  private final Amount amount;

  public Transaction(
      LocalDate date, Category category, Amount amount
  ) {
    this.date = date;
    this.category = category;
    this.amount = amount;
  }

  public LocalDate getDate() {
    return date;
  }

  public Category getCategory() {
    return category;
  }

  public Amount getAmount() {
    return amount;
  }
}
```

Snapshot 270: Category

```
package ca.jbrains.upfp.model.test;

public class Category {
  public Category(String name) {
  }
}
```

Snapshot 270: Amount

```
package ca.jbrains.upfp.model.test;

public class Amount {
  public Amount(int cents) {
  }

  static Amount cents(int cents) {
    return new Amount(cents);
  }
}
```

Now, I want the smaller pieces to behave like values[4], so I'll do that now.

Snapshot 280: Category needs to behave like a Value Object

```
package ca.jbrains.upfp.model.test;

import org.junit.Test;

import static org.junit.Assert.*;

public class CategoryValueObjectBehaviorTest {
  @Test
  public void reflexive() throws Exception {
    final Category category = new Category(
        "any valid name");
```

[4] http://link.jbrains.ca/pocv48

```
12      assertEquals(category, category);
13    }
14
15    @Test
16    public void symmetric() throws Exception {
17      final Category equalCategory1 = new Category(
18          "the same name");
19      final Category equalCategory2 = new Category(
20          "the same name");
21      final Category unequalCategory = new Category(
22          "any other name");
23
24      // both must be true; equivalent to iff
25      assertEquals(equalCategory1, equalCategory2);
26      assertEquals(equalCategory2, equalCategory1);
27
28      // both must be true; equivalent to iff
29      assertFalse(equalCategory1.equals(unequalCategory));
30      assertFalse(unequalCategory.equals(equalCategory1));
31
32      // if we get here, then we know equalCategory2 does not
33      // equal unequalCategory
34    }
35
36    @Test
37    public void transitive() throws Exception {
38      final Category equalCategory1 = new Category(
39          "the same name");
40      final Category equalCategory2 = new Category(
41          "the same name");
42      final Category equalCategory3 = new Category(
43          "the same name");
44      final Category unequalCategory = new Category(
45          "any other name");
46
47      // iff again
48      assertEquals(equalCategory1, equalCategory2);
49      assertEquals(equalCategory2, equalCategory3);
50      assertEquals(equalCategory1, equalCategory3);
51
52      // un != ec1 && ec1 == ec2 (as above) => un != ec2
53      assertFalse(unequalCategory.equals(equalCategory1));
```

```
54      assertFalse(unequalCategory.equals(equalCategory2));
55
56      // symmetry of && means + the above means
57      // ec1 == ec2 && ec2 != un => ec1 != un
58      // no need to test
59    }
60
61    @Test
62    public void nothingEqualsNull() throws Exception {
63      assertFalse(
64          new Category("irrelevant name").equals(
65              null));
66    }
67
68    // I'm willing to forgo testing for consistency, as long
69    // as Category remains immutable.
70  }
```

Snapshot 280: Category now behaves like a Value Object

```
1  diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Category.\
2  java b/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Category.java
3  index 8ba3915..e196022 100644
4  --- a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Category.java
5  +++ b/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Category.java
6  @@ -1,6 +1,28 @@
7   package ca.jbrains.upfp.model.test;
8
9  -public class Category {
10 +public final class Category {
11 +  private final String name;
12 +
13    public Category(String name) {
14 +    this.name = name;
15 +  }
16 +
17 +  @Override
18 +  public boolean equals(Object other) {
19 +    if (other instanceof Category) {
20 +      final Category that = (Category) other;
21 +      return this.name.equals(that.name);
```

```
+    }
+    return false;
+  }
+
+  @Override
+  public int hashCode() {
+    return name.hashCode();
+  }
+
+  @Override
+  public String toString() {
+    return name;
    }
 }
```

I also want Category to require its name.

Snapshot 290: Category needs to require its name

```
package ca.jbrains.upfp.model.test;

import org.junit.Test;

import static org.junit.Assert.fail;

public class CreateCategoryTest {
  @Test
  public void requireKeyInformation() throws Exception {
    try {
      new Category(null);
      fail("Why did you create a Category with no name?!");
    } catch (IllegalArgumentException success) {
    }
  }
}
```

Snapshot 290: Category now requires its name

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Category.\
java b/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Category.java
index e196022..917820e 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Category.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Category.java
@@ -4,6 +4,10 @@ public final class Category {
   private final String name;

   public Category(String name) {
+    if (name == null)
+      throw new IllegalArgumentException(
+          "name can't be null");
+
     this.name = name;
   }

```

Now, to do this again for Amount.

Snapshot 300: Amount needs to behave like a Value Object

```
package ca.jbrains.upfp.model.test;

import org.junit.Test;

import static org.junit.Assert.*;

public class AmountValueObjectBehaviorTest {
  @Test
  public void reflexive() throws Exception {
    final Amount amount = Amount.cents(12);

    assertEquals(amount, amount);
  }

  @Test
  public void symmetric() throws Exception {
    final Amount equalAmount = Amount.cents(20);
```

```
18      final Amount anotherEqualAmount = Amount.cents(20);
19      final Amount unequalAmount = Amount.cents(21);
20
21      assertEquals(equalAmount, anotherEqualAmount);
22      assertEquals(anotherEqualAmount, equalAmount);
23
24      assertFalse(equalAmount.equals(unequalAmount));
25      assertFalse(unequalAmount.equals(equalAmount));
26    }
27
28    @Test
29    public void transitive() throws Exception {
30      final Amount equalAmount1 = Amount.cents(32);
31      final Amount equalAmount2 = Amount.cents(32);
32      final Amount equalAmount3 = Amount.cents(32);
33      final Amount unequalAmount = Amount.cents(31);
34
35      assertEquals(equalAmount1, equalAmount2);
36      assertEquals(equalAmount2, equalAmount3);
37      assertEquals(equalAmount1, equalAmount3);
38
39      assertFalse(unequalAmount.equals(equalAmount1));
40      assertFalse(unequalAmount.equals(equalAmount2));
41    }
42
43    @Test
44    public void nothingEqualsNull() throws Exception {
45      assertFalse(Amount.cents(0).equals(null));
46    }
47
48    // I won't bother testing for consistency as long as
49    // Amount is immutable and final.
50  }
```

Snapshot 300: Amount now behaves like a Value Object

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Amount.ja\
va b/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Amount.java
index a1b1d36..3e9cc0d 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Amount.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Amount.java
@@ -1,10 +1,32 @@
 package ca.jbrains.upfp.model.test;

-public class Amount {
+public final class Amount {
+  private final int cents;
+
   public Amount(int cents) {
+    this.cents = cents;
   }

-  static Amount cents(int cents) {
+  public static Amount cents(int cents) {
     return new Amount(cents);
   }
+
+  @Override
+  public boolean equals(Object other) {
+    if (other instanceof Amount) {
+      final Amount that = (Amount) other;
+      return this.cents == that.cents;
+    }
+    return false;
+  }
+
+  @Override
+  public int hashCode() {
+    return cents;
+  }
+
+  @Override
+  public String toString() {
+    return String.format("%1d cents", cents);
+  }
 }
```

Fortunately, I can't give Amount a null value for cents and negative values make sense, so I only need to implement the Whole Value behavior. I wish for a way to remove duplication between Amount and Category that goes beyond the typical "equals builder" trick[5]. If I have to build a much larger object, then I'll consider bringing an equals builder into the project.

Now, when I look at `Transaction`, it behaves like an Entity[6] using its hash code as its identity. I won't worry about giving it a consistent, synthetic key until I need to, and I don't foresee it becoming a problem any time soon. Now I want to verify that I can create only sane `Transaction` objects by trying to create insane ones in tests.

Snapshot 310: Transaction needs to require its essential values

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/CreateTra\
nsactionTest.java b/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Cr\
eateTransactionTest.java
index 50f39f8..faa8d4f 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/CreateTransactio\
nTest.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/CreateTransactio\
nTest.java
@@ -3,16 +3,17 @@ package ca.jbrains.upfp.model.test;
 import org.joda.time.LocalDate;
 import org.junit.Test;

-import static org.junit.Assert.assertEquals;
+import static ca.jbrains.hamcrest.RegexMatcher.matches;
+import static org.junit.Assert.*;

 public class CreateTransactionTest {
+  private final LocalDate anyNonNullDate = new LocalDate();
+  private final Category anyNonNullCategory = new Category(
+      "irrelevant category name");
+  private final Amount anyNonNullAmount = Amount.cents(20);
+
   @Test
   public void provideAllKeyInformation() throws Exception {
-    final LocalDate anyNonNullDate = new LocalDate();
-    final Category anyNonNullCategory = new Category(
```

[5] http://link.jbrains.ca/W54e4n

[6] http://link.jbrains.ca/12Fhcfx

```
-        "irrelevant category name");
-    final Amount anyNonNullAmount = Amount.cents(20);
-
     final Transaction transaction = new Transaction(
         anyNonNullDate, anyNonNullCategory,
         anyNonNullAmount);
@@ -23,4 +24,45 @@ public class CreateTransactionTest {
         anyNonNullCategory, transaction.getCategory());
     assertEquals(anyNonNullAmount, transaction.getAmount());
   }
+
+  @Test
+  public void missingDate() throws Exception {
+    try {
+      new Transaction(
+          null, anyNonNullCategory, anyNonNullAmount);
+      fail("Why can I create a transaction with no date?!");
+    } catch (IllegalArgumentException success) {
+      assertThat(
+          success.getMessage(), matches(
+          ".*A Transaction must have a date.*"));
+    }
+  }
+
+  @Test
+  public void missingCategory() throws Exception {
+    try {
+      new Transaction(
+          anyNonNullDate, null, anyNonNullAmount);
+      fail(
+          "Why can I create a transaction with no category?!");
+    } catch (IllegalArgumentException success) {
+      assertThat(
+          success.getMessage(), matches(
+          ".*A Transaction must have a category.*"));
+    }
+  }
+
+  @Test
+  public void missingAmount() throws Exception {
+    try {
+      new Transaction(
```

```
+          anyNonNullDate, anyNonNullCategory, null);
+      fail(
+          "Why can I create a transaction with no amount?!");
+    } catch (IllegalArgumentException success) {
+      assertThat(
+          success.getMessage(), matches(
+          ".*A Transaction must have an amount.*"));
+    }
+  }
 }
```

Snapshot 310: Transaction now requires its essential values

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Transacti\
on.java b/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Transaction.\
java
index a2863ed..2b1af39 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Transaction.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Transaction.java
@@ -10,6 +10,16 @@ public class Transaction {
   public Transaction(
       LocalDate date, Category category, Amount amount
   ) {
+    if (date == null)
+      throw new IllegalArgumentException(
+          "A Transaction must have a date.");
+    if (category == null)
+      throw new IllegalArgumentException(
+          "A Transaction must have a category.");
+    if (amount == null)
+      throw new IllegalArgumentException(
+          "A Transaction must have an amount.");
+
     this.date = date;
     this.category = category;
     this.amount = amount;
```

I'd like to remove the duplication I've created in the `Transaction` object as well as in the tests. The Rule of Three dictates that I improve the design now, but I don't know off the top of my head about libraries that provide a "not nullable" annotation or some easy way to do that. I could reach into

my very old bag of tricks and do what I did on my first Java projects: extract a utility function like `Assert.notNull()` which throws the exception and constructs a standard geek-oriented error message. Rather than do that right now, I've added an item to the backlog to spend 15 minutes finding a library that can help, and I promise to do something about this the next time I need to create a Value Object or Entity with mandatory (non-nullable) properties. Deal?

That feels like a lot for one session, so I'll stop here.

What's done

- Snapshot 270: We can create a valid Transaction from a LocalDate, Category and Amount.
- Snapshot 280: Category now behaves like a Whole Value.
- Snapshot 290: Category requires its key information.
- Snapshot 300: Amount behaves as an immutable Whole Value.
- Snapshot 310: We can no longer create an insane Transaction object.

What's left to do

- Implement ExportAllTransactionsAction
  - Formatting a transaction as a CSV row
    * happy path
    * null
  - Formatting transactions as CSV file
    * 0 transactions to join
    * a few transactions to join
    * stress test
  - Writing the contents out to file
    * happy path
    * not enough space on disk
    * generic I/O failure
  - Putting it all together
    * happy path
    * formatting a transaction fails
    * formatting the entire text fails, probably by running out of memory
    * writing the text to file fails
- Export transactions to CSV, but where do we store the file?
- Write contract tests for View once they stabilise
- Enter a transaction quickly and easily
- (2 upvotes) Delegate the Activity's BrowseTransactionsView implementation to a new class
- Find a library that provides an easy way to mark a property as "non-nullable"

9 Formatting a Transaction as a row of CSV text

Next on the list is formatting a Transaction as a row of CSV text. Since Transaction objects are guaranteed sane by their constructor, I can only think of two cases to try: a sane Transaction and a null Transaction reference.

In the process of making the first test pass, I once again feel the urge to split behavior apart. I first notice a signal just after I use Fake It 'til You Make It[1] to make the test pass. I find myself here:

Snapshot 320: Faked It, but haven't yet Made It

```
1  package ca.jbrains.upfp.view.test;
2
3  import ca.jbrains.upfp.model.test.*;
4  import com.google.common.base.*;
5  import com.google.common.collect.*;
6  import com.sun.istack.internal.Nullable;
7  import org.joda.time.LocalDate;
8  import org.junit.Test;
9
10 import java.util.regex.Pattern;
11
12 import static ca.jbrains.hamcrest.RegexMatcher.matches;
13 import static org.junit.Assert.assertThat;
14
15 public class FormatTransactionAsCsvRowTest {
16   @Test
17   public void happyPath() throws Exception {
18     final Transaction transaction = new Transaction(
19         new LocalDate(2012, 11, 14), new Category(
20         "Bowling Winnings"), Amount.cents(250));
21     final String rowText = formatTransactionAsCsvRow(
22         transaction);
23     assertThat(
24         rowText, matches(
```

[1] http://link.jbrains.ca/UtljZc

```
25          Pattern.compile(
26              "\\s*\"2012-11-14\",\\s*\"Bowling Winnings\"," +
27              "\\s*\"2.50\"\\s*")));
28    }
29
30    private String formatTransactionAsCsvRow(
31        Transaction transaction
32    ) {
33      return Joiner.on(",").join(
34          Collections2.transform(
35              Lists.newArrayList(
36                  "2012-11-14", "Bowling Winnings", "2.50"),
37              new Function<String, String>() {
38                @Override
39                public String apply(@Nullable String text) {
40                  return "\"" + text + "\"";
41                }
42              }));
43    }
44  }
```

I start to extract variables for the formatted date, formatted category name and formatted amount, then realise that these want to be functions, not variables. As I consider extracting `formatDate()`, `formatCategoryName()` and `formatAmount()`, I think about how to check each of them individually. They're not particularly complicated functions, but I want tests to help me remember how to use Java's date formatting library correctly as well as its number formatting library. Checking these by invoking the entire `formatTransactionAsCsvRow()` function seems excessive – specifically, it violates the principle of Context Independence. At the same time, if I check these smaller functions separately, then I'll duplicate some of that knowledge in the test I've just written: I'll have to know the correct format for a date to write the test, and perhaps the user wants to change that! Do I want to have to know the Locale in order to write this test? Not really. More to the point, I'd much rather write a test that expects the right CSV text without knowing those details. I describe it here using BNF notation[2].

```
Transaction-as-CSV-row ::= (whitespace)* '"' Formatted-Date '"' ,
                           (whitespace)* '"' Formatted-Category '"' ,
                           (whitespace)* '"' Formatted-Amount '"'
                           (whitespace)*
```

In other words, when I check formatting the entire Transaction, I don't want to burden the test with the details of the formats of the individual properties; I simply want to check that the code arranges

[2] http://link.jbrains.ca/VgXrEW

those formatted parts in the correct sequence with the correct delimiters. How to do this? I can think of two options:

- Accept the functions that format the properties as arguments
- Accept the already-formatted properties as arguments

As cool and functional as I find the first option, I prefer the second. I run the risk of the Chunnel Problem here, but I'll risk it and work through it. Since this takes well over a dozen commits, I don't include all the steps here, but I encourage you to read the microcommits and follow the steps.

Snapshot 330: Overview of the changes

```
 AndroidFree/src/test/java/ca/jbrains/upfp/view/test/AmountCsvFormat.java  \
                     | 10 ++++++++++
 AndroidFree/src/test/java/ca/jbrains/upfp/view/test/CategoryCsvFormat.java\
                     | 11 ++++++++++
 AndroidFree/src/test/java/ca/jbrains/upfp/view/test/CsvFormat.java        \
                     |  5 +++++
 AndroidFree/src/test/java/ca/jbrains/upfp/view/test/DateCsvFormat.java    \
                     | 10 ++++++++++
 AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatTransactionAsCsv\
RowTest.java         | 71 ++++++++++++++++++++++++++++++++++++++++++++++++++--\
----------------------
 AndroidFree/src/test/java/ca/jbrains/upfp/view/test/SurroundWithQuotes.jav\
a                    | 15 +++++++++++++++
 AndroidFree/src/test/java/ca/jbrains/upfp/view/test/TransactionCsvFormat.j\
ava                  | 45 +++++++++++++++++++++++++++++++++++++++++++++
 src/test/java/ca/jbrains/upfp/controller/android/test/RenderBrowseTransact\
ionsScreenTest.java | 78 ++++++++++++++++++++++++++++++++++++++++++-------\
----------------------------
 8 files changed, 187 insertions(+), 58 deletions(-)
```

It looks like `RenderBrowseTransactionsScreenTest` changed a lot, but only one line really changed, the rest representing a change in code formatting.

When I describe the format of a `Transaction` as a row of CSV text, you see that I don't refer at all to the format of the individual properties of a `Transaction`, which makes this test very resilient to insignificant changes, and therefore quite stable.

Snapshot 330: Describing the format of a Transaction as a row of CSV text

```
package ca.jbrains.upfp.view.test;

import ca.jbrains.upfp.model.test.*;
import org.jmock.*;
import org.jmock.integration.junit4.JMock;
import org.joda.time.LocalDate;
import org.junit.Test;
import org.junit.runner.RunWith;

import java.util.regex.Pattern;

import static ca.jbrains.hamcrest.RegexMatcher.matches;
import static org.junit.Assert.assertThat;

@RunWith(JMock.class)
public class FormatTransactionAsCsvRowTest {
  private final Mockery mockery = new Mockery();

  private final CsvFormat<LocalDate> dateFormat = mockery
      .mock(CsvFormat.class, "date format");
  private final CsvFormat<Category> categoryFormat = mockery
      .mock(CsvFormat.class, "category format");
  private final CsvFormat<Amount> amountFormat = mockery
      .mock(CsvFormat.class, "amount format");
  private final TransactionCsvFormat transactionCsvFormat
      = new TransactionCsvFormat(
      dateFormat, categoryFormat, amountFormat);

  private LocalDate anyNonNullDate = new LocalDate(
      2012, 11, 14);
  private Category anyNonNullCategory = new Category(
      "Bowling Winnings");
  private Amount anyNonNullAmount = Amount.cents(250);

  @Test
  public void happyPath() throws Exception {
    mockery.checking(
        new Expectations() {{
          allowing(dateFormat).format(
              with(
```

```
                    any(
                        LocalDate.class)));
            will(returnValue("::the date::"));

            allowing(categoryFormat).format(
                with(
                    any(
                        Category.class)));
            will(returnValue("::the category::"));

            allowing(amountFormat).format(
                with(
                    any(
                        Amount.class)));
            will(returnValue("::the amount::"));
        }});

    final Transaction transaction = new Transaction(
        anyNonNullDate, anyNonNullCategory,
        anyNonNullAmount);
    final String rowText = transactionCsvFormat.format(
        transaction);
    assertThat(
        rowText, matches(
        Pattern.compile(
            "\\s*\"::the date::\"," +
            "\\s*\"::the category::\",\\s*\"::the amount::\"\\s*")));
  }
}
```

Formatting a `Transaction` as a row of CSV text involves formatting the properties in the right sequence, surrounding them with quotes, and separating them with commas.

Snapshot 330: Formatting a Transaction as a row of CSV text

```
package ca.jbrains.upfp.view.test;

import ca.jbrains.upfp.model.test.*;
import com.google.common.base.Joiner;
import com.google.common.collect.*;
import org.joda.time.LocalDate;
```

```
7
8  import java.util.List;
9
10 public class TransactionCsvFormat
11     implements CsvFormat<Transaction> {
12   private final CsvFormat<LocalDate> dateCsvFormat;
13   private final CsvFormat<Category> categoryCsvFormat;
14   private final CsvFormat<Amount> amountCsvFormat;
15
16   public TransactionCsvFormat(
17       CsvFormat<LocalDate> dateCsvFormat,
18       CsvFormat<Category> categoryCsvFormat,
19       CsvFormat<Amount> amountCsvFormat
20   ) {
21     this.dateCsvFormat = dateCsvFormat;
22     this.categoryCsvFormat = categoryCsvFormat;
23     this.amountCsvFormat = amountCsvFormat;
24   }
25
26   public String format(Transaction transaction) {
27     final List<String> formattedPropertiesInCorrectSequence
28         = Lists.newArrayList(
29         dateCsvFormat.format(transaction.getDate()),
30         categoryCsvFormat.format(transaction.getCategory()),
31         amountCsvFormat.format(transaction.getAmount()));
32
33     return assembleIntoCsvRow(
34         formattedPropertiesInCorrectSequence);
35   }
36
37   private String assembleIntoCsvRow(
38       List<String> formattedPropertiesInCorrectSequence
39   ) {
40     return Joiner.on(",").join(
41         Collections2.transform(
42             formattedPropertiesInCorrectSequence,
43             SurroundWithQuotes.INSTANCE));
44   }
45 }
```

I've pushed the "Faking It" down from formatting a Transaction to formatting its properties. "Making" these will come next.

Snapshot 330: Choosing a Date format for CSV text

```
package ca.jbrains.upfp.view.test;

import org.joda.time.LocalDate;

public class DateCsvFormat implements CsvFormat<LocalDate> {
  @Override
  public String format(LocalDate date) {
    return "2012-11-14";
  }
}
```

Snapshot 330: Choosing a Category format for CSV text

```
package ca.jbrains.upfp.view.test;

import ca.jbrains.upfp.model.test.Category;

public class CategoryCsvFormat
    implements CsvFormat<Category> {
  @Override
  public String format(Category category) {
    return "Bowling Winnings";
  }
}
```

Snapshot 330: Choosing an Amount format for CSV text

```
package ca.jbrains.upfp.view.test;

import ca.jbrains.upfp.model.test.Amount;

public class AmountCsvFormat implements CsvFormat<Amount> {
  @Override
  public String format(Amount amount) {
    return "2.50";
  }
}
```

You might have noticed that all these formats now implement a common interface.

Snapshot 330: Formatting anything for CSV text

```
package ca.jbrains.upfp.view.test;

public interface CsvFormat<ValueType> {
  String format(ValueType value);
}
```

Merely for the sake of completeness, I show you the function that `TransactionCsvFormat` uses to surround each formatted property with quotes before joining them with commas.

Snapshot 330: Surround a String with quotes

```
package ca.jbrains.upfp.view.test;

import com.google.common.base.Function;
import com.sun.istack.internal.Nullable;

class SurroundWithQuotes
    implements Function<String, String> {
  public static final SurroundWithQuotes INSTANCE
      = new SurroundWithQuotes();

  @Override
  public String apply(@Nullable String text) {
    return "\"" + text + "\"";
  }
}
```

Before declaring this task complete, I need to consider formatting a null Transaction. I feel like receiving a null Transaction indicates a Programmer Mistake, so I decide to throw one. This makes me think of the bigger picture, in which I would consider most null parameters as a Programmer Mistake. I feel a little nervous, however, letting null references work their way through the code until some unfortunate object chokes on it, so against my preference to avoid defensive programming, I'll throw a `ProgrammerMistake` if someone gives me null. I hope I can find a better way to do this. In the meantime, I'll rely on my typical approach: push defensive programming measures up the call stack as far as possible.

Snapshot 340: We should treat formatting a null Transaction as a Programmer Mistake

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatTran\
sactionAsCsvRowTest.java b/AndroidFree/src/test/java/ca/jbrains/upfp/view/t\
est/FormatTransactionAsCsvRowTest.java
index ee61d09..bab0924 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatTransaction\
AsCsvRowTest.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatTransaction\
AsCsvRowTest.java
@@ -1,5 +1,6 @@
 package ca.jbrains.upfp.view.test;

+import ca.jbrains.toolkit.ProgrammerMistake;
 import ca.jbrains.upfp.model.test.*;
 import org.jmock.*;
 import org.jmock.integration.junit4.JMock;
@@ -10,7 +11,7 @@ import org.junit.runner.RunWith;
 import java.util.regex.Pattern;

 import static ca.jbrains.hamcrest.RegexMatcher.matches;
-import static org.junit.Assert.assertThat;
+import static org.junit.Assert.*;

 @RunWith(JMock.class)
 public class FormatTransactionAsCsvRowTest {
@@ -66,4 +67,14 @@ public class FormatTransactionAsCsvRowTest {
             "\\s*\"::the date::\"," +
             "\\s*\"::the category::\",\\s*\"::the amount::\"\\s*")));
   }
+
+  @Test
+  public void nullTransaction() throws Exception {
+    try {
+      transactionCsvFormat.format(null);
+      fail("How did you format a null transaction?!");
+    } catch (ProgrammerMistake success) {
+    }
+  }
+
 }
```

Snapshot 340: We now treat formatting a null Transaction as a Programmer Mistake

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/Transactio\
nCsvFormat.java b/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/Trans\
actionCsvFormat.java
index c155cce..0e448f3 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/TransactionCsvFor\
mat.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/TransactionCsvFor\
mat.java
@@ -1,5 +1,6 @@
 package ca.jbrains.upfp.view.test;

+import ca.jbrains.toolkit.ProgrammerMistake;
 import ca.jbrains.upfp.model.test.*;
 import com.google.common.base.Joiner;
 import com.google.common.collect.*;
@@ -24,6 +25,9 @@ public class TransactionCsvFormat
   }

   public String format(Transaction transaction) {
+    if (transaction == null) throw new ProgrammerMistake(
+        "Can't format a null transaction.");
+
     final List<String> formattedPropertiesInCorrectSequence
         = Lists.newArrayList(
         dateCsvFormat.format(transaction.getDate()),
```

Now that I think of it, I should make this condition a Programmer Mistake in the other cases that I've recently added. Moreover, whatever library I use to help me mark mandatory properties should give me the option of throwing a `ProgrammerMistake`, so I make a note of that in the backlog.

Snapshot 350: We should label a programmer mistake more clearly

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/CreateTra\
nsactionTest.java b/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Cr\
eateTransactionTest.java
index faa8d4f..be5f250 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/CreateTransactio\
nTest.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/CreateTransactio\
nTest.java
@@ -1,5 +1,6 @@
 package ca.jbrains.upfp.model.test;

+import ca.jbrains.toolkit.ProgrammerMistake;
 import org.joda.time.LocalDate;
 import org.junit.Test;

@@ -31,7 +32,7 @@ public class CreateTransactionTest {
         new Transaction(
             null, anyNonNullCategory, anyNonNullAmount);
         fail("Why can I create a transaction with no date?!");
-    } catch (IllegalArgumentException success) {
+    } catch (ProgrammerMistake success) {
         assertThat(
             success.getMessage(), matches(
             ".*A Transaction must have a date.*"));
@@ -44,8 +45,9 @@ public class CreateTransactionTest {
         new Transaction(
             anyNonNullDate, null, anyNonNullAmount);
         fail(
-            "Why can I create a transaction with no category?!");
-    } catch (IllegalArgumentException success) {
+            "Why can I create a transaction with no " +
+            "category?!");
+    } catch (ProgrammerMistake success) {
         assertThat(
             success.getMessage(), matches(
             ".*A Transaction must have a category.*"));
@@ -58,8 +60,9 @@ public class CreateTransactionTest {
         new Transaction(
             anyNonNullDate, anyNonNullCategory, null);
         fail(
```

```
-           "Why can I create a transaction with no amount?!");
-     } catch (IllegalArgumentException success) {
+           "Why can I create a transaction with no " +
+           "amount?!");
+     } catch (ProgrammerMistake success) {
        assertThat(
            success.getMessage(), matches(
            ".*A Transaction must have an amount.*"));
```

Snapshot 350: We now label a programmer mistake more clearly

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Transacti\
on.java b/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Transaction.\
java
index 2b1af39..746f230 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Transaction.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Transaction.java
@@ -1,5 +1,6 @@
 package ca.jbrains.upfp.model.test;

+import ca.jbrains.toolkit.ProgrammerMistake;
 import org.joda.time.LocalDate;

 public class Transaction {
@@ -10,15 +11,12 @@ public class Transaction {
   public Transaction(
       LocalDate date, Category category, Amount amount
   ) {
-    if (date == null)
-      throw new IllegalArgumentException(
-          "A Transaction must have a date.");
-    if (category == null)
-      throw new IllegalArgumentException(
-          "A Transaction must have a category.");
-    if (amount == null)
-      throw new IllegalArgumentException(
-          "A Transaction must have an amount.");
+    if (date == null) throw new ProgrammerMistake(
+        "A Transaction must have a date.");
+    if (category == null) throw new ProgrammerMistake(
+        "A Transaction must have a category.");
```

```
31 +    if (amount == null) throw new ProgrammerMistake(
32 +        "A Transaction must have an amount.");
33
34      this.date = date;
35      this.category = category;
```

Now I feel ready to format the individual Transaction properties as CSV text.

What's done

- Snapshot 320: Faked formatting a Transaction as a row of CSV text.
- Snapshot 330:
  - Separated formatting the properties from assembling them as a row of CSV text.
  - Extracted formatting as functions in preparation for assembling them in a TransactionCsvFormat class.
  - Extracted format*() "helper" methods into a new production code class.
  - Turned code into data so that I could pass them as parameters.
  - Turned the property format objects into fields so that I could inject them through the constructor.
  - Extracted a common interface from all the little format objects.
  - Improved a method name.
  - Removed an obsolete parameter.
  - Weakened the types of variables to prepare for injecting them into a constructor.
  - Prepared nested class to move them to top-level classes.
  - Turned nested classes into top-level classes.
  - Now when we check the Transaction CSV format, we don't care how formatting the date works.
  - Now when we check the Transaction CSV format, we don't care how formatting any of its properties works.
  - Cleaned up the test, including using the "metaconstant" idea from Midje[3] to draw attention to dummy variables. This is my new favorite way to avoid irrelevant details in tests.
  - A few more type enhancements.
  - Finally removed duplication between the production code and tests, so that I'm no longer faking it.
- Snapshot 340: Reject formatting a null Transaction as a Programmer Mistake.
- Snapshot 350: Reformat some code to match the current formatting conventions.
- Snapshot 360: When someone fails to provide a mandatory property to a Transaction, I now treat it as a Programmer Mistake.

[3] Brian Marick describes metaconstants in this article

What's left to do

With all the details on this list, I could never keep all this in my head.

- Implement ExportAllTransactionsAction
  - Formatting a date as a CSV element
    - happy path
  - Formatting a category as a CSV element
    - happy path
  - Formatting an amount as a CSV element
    - happy path
    - 1.00
    - 1.50
    - 0.50
    - 12.45
    - 1897.20
  - Formatting transactions as CSV file
    - 0 transactions to join
    - a few transactions to join
    - stress test
  - Writing the contents out to file
    - happy path
    - not enough space on disk
    - generic I/O failure
  - Putting it all together
    - happy path
    - formatting a transaction fails
    - formatting the entire text fails, probably by running out of memory
    - writing the text to file fails
- Export transactions to CSV, but where do we store the file?
- Write contract tests for View once they stabilise
- Enter a transaction quickly and easily
- (2 upvotes) Delegate the Activity's BrowseTransactionsView implementation to a new class
- Find a library that provides an easy way to mark a property as "non-nullable"
  - It needs to let me throw a Programmer Mistake when I want to.
  - Evaluate the @Nullable and @NotNull annotations.

10 Formatting the Properties of a Transaction

This session should probably not reveal any surprises, but I believe I've already written that once or twice here. I plan to implement formatting dates, categories and amounts. Since Category simply represents a string, formatting it is no real work, so I'll warm up with that.

Formatting a Category

Just now I notice that `Category` allows an empty name, which I should disallow, so I take a few moments to do that first.

Snapshot 360: Category should not allow a blank name

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/CreateCat\
egoryTest.java b/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Creat\
eCategoryTest.java
index c3c388b..bb2bc65 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/CreateCategoryTe\
st.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/CreateCategoryTe\
st.java
@@ -1,8 +1,12 @@
 package ca.jbrains.upfp.model.test;

+import ca.jbrains.toolkit.ProgrammerMistake;
 import org.junit.Test;

-import static org.junit.Assert.fail;
+import java.util.regex.Pattern;
+
+import static ca.jbrains.hamcrest.RegexMatcher.matches;
+import static org.junit.Assert.*;

 public class CreateCategoryTest {
   @Test
```

```
@@ -13,4 +17,36 @@ public class CreateCategoryTest {
     } catch (IllegalArgumentException success) {
     }
   }
+
+  @Test
+  public void nameCannotBeBlank() throws Exception {
+    try {
+      new Category("");
+      fail(
+          "Why did you create a Category with an empty " +
+          "name?!");
+    } catch (ProgrammerMistake success) {
+      assertThat(
+          success.getMessage(), matches(
+          Pattern.compile(
+              ".*Category name can't be blank.*")));
+    }
+  }
+
+  @Test
+  public void nameCannotBeOnlyWhitespace()
+      throws Exception {
+    try {
+      new Category("  \t \n   \r ");
+      fail(
+          "Why did you create a Category with a name made" +
+          " up of only whitespace?!");
+    } catch (ProgrammerMistake success) {
+      assertThat(
+          success.getMessage(), matches(
+          Pattern.compile(
+              ".*Category name can't be only whitespace.*")));
+    }
+  }
+
 }
```

Snapshot 360: Category no longer allows a blank name

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Category.\
java b/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Category.java
index 917820e..89f29a4 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Category.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Category.java
@@ -1,12 +1,17 @@
 package ca.jbrains.upfp.model.test;

+import ca.jbrains.toolkit.ProgrammerMistake;
+
 public final class Category {
   private final String name;

   public Category(String name) {
-    if (name == null)
-      throw new IllegalArgumentException(
-          "name can't be null");
+    if (name == null) throw new IllegalArgumentException(
+        "name can't be null");
+    if (name.isEmpty()) throw new ProgrammerMistake(
+        "Category name can't be blank.");
+    if (name.trim().isEmpty()) throw new ProgrammerMistake(
+        "Category name can't be only whitespace.");

     this.name = name;
   }
```

I can see that I might want to do a lot more of this kind of domain object validation, so I should look for a library to do it in a declarative style somewhat like Rails Validations does. I put that on the backlog. After dealing with this minor distraction, I return to formatting a `Category`.

Snapshot 370: We should format a valid Category correctly

```
package ca.jbrains.upfp.view.test;

import ca.jbrains.upfp.model.test.Category;
import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class FormatCategoryAsCsvTest {
  @Test
  public void happyPath() throws Exception {
    final Category category = new Category(
        "A valid category name");
    final CategoryCsvFormat categoryCsvFormat
        = new CategoryCsvFormat();
    assertEquals(
        "A valid category name", categoryCsvFormat.format(
        category));
  }
}
```

Snapshot 370: We now format a valid Category correctly

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/CategoryCs\
vFormat.java b/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/Category\
CsvFormat.java
index 554df02..8dbb457 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/CategoryCsvFormat\
.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/CategoryCsvFormat\
.java
@@ -6,6 +6,6 @@ public class CategoryCsvFormat
     implements CsvFormat<Category> {
   @Override
   public String format(Category category) {
-    return "Bowling Winnings";
+    return category.getName();
   }
 }
```

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Category.\
java b/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Category.java
index 89f29a4..7c5b71e 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Category.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Category.java
@@ -16,6 +16,10 @@ public final class Category {
     this.name = name;
   }

+  public String getName() {
+    return name;
+  }
+
   @Override
   public boolean equals(Object other) {
     if (other instanceof Category) {
```

In the process of making this simple test pass, I wonder about a `Category` where the name has leading or trailing whitespace. Since a user will enter this information, I need to consider how to validate or sanitise it.[1] On the one hand, handling it in the model ensures consistent behavior everywhere I use `Category`; but on the other hand, I prefer to fix problems closer to where the problem originates, which leads me to trim the category name in whatever controller that handles receiving a new category from the user. In this case, I choose not to program defensively, but rather defer the issue until I let the user create a category, whenever that happens.

I can't think of any examples of formatting a `Category` that might fail, so I move on to formatting dates.

Formatting a Date

I trust joda-time[2] to format dates correctly, so I write only a single test to check that I've chosen the format that I want.

[1] Read more about this in Ward Cunningham's paper, "The CHECKS Pattern Language of Information Integrity".

[2] http://link.jbrains.ca/U64wt2

Snapshot 380: We should format dates in ISO format

```
package ca.jbrains.upfp.view.test;

import org.joda.time.LocalDate;
import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class FormatLocalDateAsCsvTest {
  @Test
  public void happyPath() throws Exception {
    assertEquals(
        "1974-05-04", new DateCsvFormat().format(
        new LocalDate(1974, 5, 4)));
  }
}
```

Snapshot 380: We now format dates in ISO format

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/DateCsvFor\
mat.java b/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/DateCsvForma\
t.java
index 8002502..925aee7 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/DateCsvFormat.jav\
a
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/DateCsvFormat.jav\
a
@@ -1,10 +1,14 @@
 package ca.jbrains.upfp.view.test;

 import org.joda.time.LocalDate;
+import org.joda.time.format.*;

 public class DateCsvFormat implements CsvFormat<LocalDate> {
+  public static final DateTimeFormatter ISO_FORMAT
+      = DateTimeFormat.forPattern("yyyy-MM-dd");
+
   @Override
   public String format(LocalDate date) {
```

```
-    return "2012-11-14";
+    return ISO_FORMAT.print(date);
   }
 }
```

Again, I can't think of any examples of formatting a `LocalDate` that might fail, so I move on to formatting amounts.

Formatting an Amount

Although I use another trusted library here, I want to check a little more carefully, since I never remember how to format leading zeroes with `printf`.

Snapshot 390: We should format an Amount to look like dollars

```
package ca.jbrains.upfp.view.test;

import ca.jbrains.upfp.model.test.Amount;
import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class FormatAmountAsCsvTest {
  @Test
  public void noZeroes() throws Exception {
    assertEquals("12.87", formatCents(1287));
  }

  @Test
  public void oneTrailingZero() throws Exception {
    assertEquals("12.50", formatCents(1250));
  }

  @Test
  public void twoTrailingZeroes() throws Exception {
    assertEquals("89.00", formatCents(8900));
  }

  @Test
  public void lessThanOneDollar() throws Exception {
```

```
26      assertEquals("0.98", formatCents(98));
27    }
28
29    @Test
30    public void lessThanTenCents() throws Exception {
31      assertEquals("0.01", formatCents(1));
32    }
33
34    @Test
35    public void zero() throws Exception {
36      assertEquals("0.00", formatCents(0));
37    }
38
39    @Test
40    public void negative() throws Exception {
41      assertEquals("-0.07", formatCents(-7));
42    }
43
44    @Test
45    public void thousands() throws Exception {
46      assertEquals("1579.23", formatCents(157923));
47    }
48
49    private String formatCents(int cents) {
50      return new AmountCsvFormat().format(
51          Amount.cents(cents));
52    }
53  }
```

Snapshot 390: We now format an Amount to look like dollars

```
1   diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/AmountCsvF\
2   ormat.java b/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/AmountCsvF\
3   ormat.java
4   index 8785ba5..c9fd858 100644
5   --- a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/AmountCsvFormat.j\
6   ava
7   +++ b/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/AmountCsvFormat.j\
8   ava
9   @@ -5,6 +5,6 @@ import ca.jbrains.upfp.model.test.Amount;
10   public class AmountCsvFormat implements CsvFormat<Amount> {
```

```
11     @Override
12     public String format(Amount amount) {
13 -     return "2.50";
14 +     return String.format("%1$.2f", amount.inDollars());
15     }
16   }
```

```
1  diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Amount.ja\
2  va b/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Amount.java
3  index 3e9cc0d..5c357f3 100644
4  --- a/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Amount.java
5  +++ b/AndroidFree/src/test/java/ca/jbrains/upfp/model/test/Amount.java
6  @@ -29,4 +29,8 @@ public final class Amount {
7     public String toString() {
8       return String.format("%1d cents", cents);
9     }
10 +
11 +  public double inDollars() {
12 +    return cents / 100.0d;
13 +  }
14   }
```

Just before I recap and start a new session, something springs to mind: I've called these formats "CSV format" although they have started to feel like generic text formats instead. Should I rename these classes changing "CSV" to "text"? Perhaps, but I don't want to do it just yet, and for two reasons:

1. I don't quite know that it will buy me now.
2. I might format dates and amounts as text differently in different contexts, whereas I've chosen these specific formats for this specific CSV representation of Transactions.

If anything, the name "CSV format" now feels overly vague. I couldn't reuse the `DateCsvFormat` unchanged in another application. Truly, this class implements the specific CSV format for dates that I want to use in this application only. Even so, I can't think of a name with higher precision without sacrificing common sense, which tells me to do nothing for the moment. As I write more features, I gather more evidence and perceive more signals from the code, and they will tell me what to do.

What's done

- Snapshot 360:

    - Category name can no longer be blank.
    - Minor formatting change.
    - Category name can no longer be only whitespace.
- Snapshot 370: We can now format a Category as an element in a CSV row.
- Snapshot 380: We now format dates correctly for CSV.
- Snapshot 390: We can now format an Amount for CSV

What's left to do

- Implement ExportAllTransactionsAction
    - Formatting transactions as CSV file
        * 0 transactions to join
        * a few transactions to join
        * stress test
    - Writing the contents out to file
        * happy path
        * not enough space on disk
        * generic I/O failure
    - Putting it all together
        * happy path
        * formatting a transaction fails
        * formatting the entire text fails, probably by running out of memory
        * writing the text to file fails
- Export transactions to CSV, but where do we store the file?
- Write contract tests for View once they stabilise
- Enter a transaction quickly and easily
- (2 upvotes) Delegate the Activity's BrowseTransactionsView implementation to a new class
- Find a library that provides an easy way to mark a property as "non-nullable"
    - It needs to let me throw a Programmer Mistake when I want to.
    - Evaluate the @Nullable and @NotNull annotations.
- Find a library that provides Rails-style object validations.

11 Formatting a Collection of Transactions

I choose to finish the formatting part of this behavior before moving on to writing the contents to file, then putting it all together. This part looks pretty simple: we already know how to format individual `Transaction` objects as a row of CSV, so now we simply have to format a collection of them, each on its own line. I see only one mildly interesting part: make sure that the column appear in the header row in the same sequence as the elements in a row. This makes me think of the Abstract Factory pattern. I don't know whether I'll use it, but I have it in the back of my mind.

I plan to write tests for formatting a collection of `Transaction` objects in a similar style to the ones for formatting a single `Transaction`. I had specified the format of a single `Transaction` in terms of the formats of its properties without actually formatting those properties, stubbing the formatted properties as metaconstants. I plan to do the same here, stubbing the format of each line in order to focus on correctly joining the lines together. In order to make this safe, however, I will need at least one test to verify that the sequence of the column names in the header match the sequence of columns in the rows, for which I might need that Abstract Factory.

> I use Design Patterns in my work, but not to pull off the shelf and drop into my code. I use the vocabulary of Design Patterns to describe my designs more concisely to my colleagues. I also refactor towards and away from Design Patterns as I get the feeling I need to. If I think I need, for example, a Composite, and I can't refactor towards it, or if refactoring towards it starts to look or feel strange, I interpret that a signal telling me that I don't really need Composite.

As with any tests involving a collection, I follow the "0, 1, many, lots, oops" pattern. I consider five tests:

- empty collection, in case there's a special case here to consider
  - Example: show a message saying "No items found", rather than a table of search results.
- one item, in case there's a special case here to consider
  - Example: a singular noun instead of a plural.
- a few items, since that forces the algorithm to generalise
- a lot of items, just to look for stress points in the design
- blowing up, in case that can happen and I have to deal with it

For this feature, I don't need special behavior for the empty case: a file with a header and no body work for me. Similarly, I see no special behavior for the one-item case. A stress test seems sensible, given the lower RAM environment of a mobile device. As for blowing up, what might go wrong? I'm formatting a collection of objects in memory, all of which I've constructed to be sane, and formatting each object and each property never throws an exception. If this blows up, then it will be from a Programmer Mistake, which I will happily let bubble up the call stack. That leaves three tests:

- 0 transactions
- a few transactions
- 2500 transactions

I choose 2500 arbitrarily. One thing: testing 2500 transactions in plain Java running in a VM on a laptop will likely reveal nothing useful. I need to run this as an end-to-end test on the real hardware. Well, no… I can test it in the simulator using Robolectric and it will probably tell me something useful. I add this to the backlog. That brings me down to two tests, and I feel ready to write them.

Formatting an empty collection of Transactions results in just the header row. Here I do some TDD as if I meant it[1] and start building the production code directly inside the test class.

Snapshot 400: Formatting an empty collection of Transactions

```
1  package ca.jbrains.upfp.view.test;
2
3  import ca.jbrains.upfp.model.test.Transaction;
4  import com.google.common.base.Joiner;
5  import com.google.common.collect.*;
6  import org.junit.Test;
7
8  import java.io.*;
9  import java.util.*;
10
11 import static ca.jbrains.hamcrest.RegexMatcher.matches;
12 import static org.hamcrest.core.StringEndsWith.endsWith;
13 import static org.junit.Assert.assertThat;
14
15 public class FormatTransactionsAsCsvFileTest {
16   @Test
17   public void noTransactions() throws Exception {
18     final String text = formatTransactionsAsCsvFile(
19         Collections.<Transaction>emptyList());
```

[1] http://link.jbrains.ca/VmrbDp

```
    assertThat(
        text, matches(
        "\\s*\"Date\",\\s*\"Category\"," +
        "\\s*\"Amount\"\\s*"));
    assertThat(
        text, endsWith(
        System.getProperty(
            "line.separator")));
  }

  private String formatTransactionsAsCsvFile(
      List<Transaction> transactions
  ) {
    final StringWriter text = new StringWriter();
    final PrintWriter canvas = new PrintWriter(text);
    canvas.println(formatHeader());
    return text.toString();
  }

  private String formatHeader() {
    return Joiner.on(",").join(
        Collections2.transform(
            Lists.newArrayList(
                "Date", "Category", "Amount"),
            new SurroundWithQuotes()));
  }
}
```

To format a nonempty collection, I stub the format for a single `Transaction` using metaconstants.

Snapshot 410: Formatting a non-empty collection of Transactions

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatTran\
sactionsAsCsvFileTest.java b/AndroidFree/src/test/java/ca/jbrains/upfp/view\
/test/FormatTransactionsAsCsvFileTest.java
index 6dfbbcb..e8d2ca5 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatTransaction\
sAsCsvFileTest.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatTransaction\
sAsCsvFileTest.java
@@ -1,22 +1,33 @@
```

```
 package ca.jbrains.upfp.view.test;

-import ca.jbrains.upfp.model.test.Transaction;
+import ca.jbrains.upfp.model.test.*;
 import com.google.common.base.Joiner;
 import com.google.common.collect.*;
+import org.jmock.*;
+import org.jmock.integration.junit4.JMock;
+import org.joda.time.LocalDate;
 import org.junit.Test;
+import org.junit.runner.RunWith;

 import java.io.*;
 import java.util.*;

 import static ca.jbrains.hamcrest.RegexMatcher.matches;
 import static org.hamcrest.core.StringEndsWith.endsWith;
-import static org.junit.Assert.assertThat;
+import static org.junit.Assert.*;

+@RunWith(JMock.class)
 public class FormatTransactionsAsCsvFileTest {
+  private Mockery mockery = new Mockery();
+
+  private final CsvFormat<Transaction> transactionCsvFormat
+      = mockery.mock(CsvFormat.class, "transaction format");
+
   @Test
   public void noTransactions() throws Exception {
     final String text = formatTransactionsAsCsvFile(
-        Collections.<Transaction>emptyList());
+        Collections.<Transaction>emptyList(),
+        transactionCsvFormat);
     assertThat(
         text, matches(
         "\\s*\"Date\",\\s*\"Category\"," +
@@ -27,12 +38,63 @@ public class FormatTransactionsAsCsvFileTest {
             "line.separator")));
   }

+  @Test
+  public void aFewTransactions() throws Exception {
```

```
52  +    mockery.checking(
53  +        new Expectations() {{
54  +          allowing(transactionCsvFormat).format(
55  +              with(
56  +                  any(
57  +                      Transaction.class)));
58  +          will(
59  +              onConsecutiveCalls(
60  +                  returnValue("::row 1::"), returnValue(
61  +                  "::row 2::"), returnValue("::row 3::")));
62  +        }});
63  +
64  +    final String text = formatTransactionsAsCsvFile(
65  +        Lists.newArrayList(
66  +            createAnyNonNullTransaction(),
67  +            createAnyNonNullTransaction(),
68  +            createAnyNonNullTransaction()),
69  +        transactionCsvFormat);
70  +
71  +    final List<String> lines = Arrays.asList(
72  +        text.split(
73  +            System.getProperty("line.separator")));
74  +    assertEquals(4, lines.size());
75  +    assertThat(
76  +        lines.get(0), matches(
77  +        "\\s*\"Date\",\\s*\"Category\"," +
78  +        "\\s*\"Amount\"\\s*"));
79  +    assertThat(lines.get(1), matches("::row 1::"));
80  +    assertThat(lines.get(2), matches("::row 2::"));
81  +    assertThat(lines.get(3), matches("::row 3::"));
82  +    assertThat(
83  +        text, endsWith(
84  +        System.getProperty(
85  +            "line.separator")));
86  +  }
87  +
88  +  private Transaction createAnyNonNullTransaction() {
89  +    return new Transaction(
90  +        new LocalDate(2012, 8, 4), new Category(
91  +        "irrelevant category"), Amount.cents(
92  +        26123));
93  +  }
```

```
94  +
95     private String formatTransactionsAsCsvFile(
96  -      List<Transaction> transactions
97  +      List<Transaction> transactions,
98  +      CsvFormat<Transaction> transactionCsvFormat
99     ) {
100 +    // I'm not sure whether I prefer this to join on line
101 +    // .separator
102       final StringWriter text = new StringWriter();
103       final PrintWriter canvas = new PrintWriter(text);
104       canvas.println(formatHeader());
105 +     for (Transaction each : transactions) {
106 +       canvas.println(transactionCsvFormat.format(each));
107 +     }
108       return text.toString();
109    }
110
```

A Cohesion Problem

Notice a cohesion problem in the code. Specifically, the sequence of the columns in the header and in the row. Right now, `formatTransactionsAsCsvFile()` knows the right sequence and the class `TransactionCsvFormat` knows the right sequence. I don't want to repeat myself[2]. Moreover, I don't want to program by accident[3]. I want to put these two pieces of information together, and sure enough, I want an Abstract Factory.

A deeper look shows another problem: mixed levels of abstraction. Specifically, in the same code, we know the details of how to format the header row, but defer the details of formatting the body rows, even though those details need to match. That makes two symptoms of the same underlying problem. I find this signal more than strong enough to fix it right now. I intend to do it this way:

1. Push the details of formatting the header row out into a tiny object.
2. Extract an interface for this new object.
3. Stub the new interface using the very same metaconstant techniques I've already used here.
4. Create a little Abstract Factory to ensure that the formats of the header row and the body rows match.

I finish the first two steps relatively easily.

[2]Read about Don't Repeat Yourself and other key principles in The Pragmatic Programmer: From Journey to Master.

[3]Read about Programming by Accident (or by Coincidence) in, coincidentally, The Pragmatic Programmer: From Journey to Master.

Snapshot 420: Moving the details of formatting the header out of the test

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatTran\
sactionsAsCsvFileTest.java b/AndroidFree/src/test/java/ca/jbrains/upfp/view\
/test/FormatTransactionsAsCsvFileTest.java
index e8d2ca5..ee81d5c 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatTransaction\
sAsCsvFileTest.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatTransaction\
sAsCsvFileTest.java
@@ -1,8 +1,7 @@
 package ca.jbrains.upfp.view.test;

 import ca.jbrains.upfp.model.test.*;
-import com.google.common.base.Joiner;
-import com.google.common.collect.*;
+import com.google.common.collect.Lists;
 import org.jmock.*;
 import org.jmock.integration.junit4.JMock;
 import org.joda.time.LocalDate;
@@ -22,12 +21,14 @@ public class FormatTransactionsAsCsvFileTest {

   private final CsvFormat<Transaction> transactionCsvFormat
       = mockery.mock(CsvFormat.class, "transaction format");
+  private final CsvHeaderFormat csvHeaderFormat
+      = new TransactionsCsvHeader();

   @Test
   public void noTransactions() throws Exception {
     final String text = formatTransactionsAsCsvFile(
         Collections.<Transaction>emptyList(),
-        transactionCsvFormat);
+        csvHeaderFormat, transactionCsvFormat);
     assertThat(
         text, matches(
         "\\s*\"Date\",\\s*\"Category\"," +
@@ -56,7 +57,7 @@ public class FormatTransactionsAsCsvFileTest {
         Lists.newArrayList(
             createAnyNonNullTransaction(),
             createAnyNonNullTransaction(),
-            createAnyNonNullTransaction()),
+            createAnyNonNullTransaction()), csvHeaderFormat,
```

```
41          transactionCsvFormat);
42
43      final List<String> lines = Arrays.asList(
44  @@ -85,24 +86,17 @@ public class FormatTransactionsAsCsvFileTest {
45
46     private String formatTransactionsAsCsvFile(
47         List<Transaction> transactions,
48  +      CsvHeaderFormat csvHeaderFormat,
49         CsvFormat<Transaction> transactionCsvFormat
50     ) {
51       // I'm not sure whether I prefer this to join on line
52       // .separator
53       final StringWriter text = new StringWriter();
54       final PrintWriter canvas = new PrintWriter(text);
55  -    canvas.println(formatHeader());
56  +    canvas.println(csvHeaderFormat.formatHeader());
57       for (Transaction each : transactions) {
58         canvas.println(transactionCsvFormat.format(each));
59       }
60       return text.toString();
61     }
62  -
63  -  private String formatHeader() {
64  -    return Joiner.on(",").join(
65  -        Collections2.transform(
66  -            Lists.newArrayList(
67  -                "Date", "Category", "Amount"),
68  -            new SurroundWithQuotes()));
69  -  }
70   }
```

Snapshot 420: Introducing a class to format the header

```
1  package ca.jbrains.upfp.view.test;
2
3  import com.google.common.base.Joiner;
4  import com.google.common.collect.*;
5
6  import java.util.ArrayList;
7
8  // This has to match the implementation of
```

```
// CsvFormat<Transaction>
public class TransactionsCsvHeader
    implements CsvHeaderFormat {
  public static final ArrayList<String> COLUMN_NAMES = Lists
      .newArrayList("Date", "Category", "Amount");

  @Override
  public String formatHeader() {
    return Joiner.on(",").join(
        Collections2.transform(
            COLUMN_NAMES, new SurroundWithQuotes()));
  }
}
```

Snapshot 420: Introducing an interface to format the header

```
package ca.jbrains.upfp.view.test;

public interface CsvHeaderFormat {
  String formatHeader();
}
```

Now I see that I need to move the tests for the format of the header into a new fixture, so I'll call that step 2.5.

Snapshot 430: Checking the format of the header more directly

```
package ca.jbrains.upfp.view.test;

import org.junit.Test;

import static ca.jbrains.hamcrest.RegexMatcher.matches;
import static org.junit.Assert.assertThat;

public class FormatTransactionCsvHeaderTest {
  @Test
  public void noTransactions() throws Exception {
    final String text = new TransactionsCsvHeader()
        .formatHeader();
```

```
13      assertThat(
14          text, matches(
15          "\\s*\"Date\",\\s*\"Category\",\\s*\"Amount\"\\s*"));
16    }
17  }
```

I much prefer this. I never liked checking for a new line at the end of the file and the format of the header row in the same test. Now I can replace those details with the metaconstant `::header::`.

Snapshot 440: Moving towards a single level of abstraction

```
1  diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatTran\
2  sactionsAsCsvFileTest.java b/AndroidFree/src/test/java/ca/jbrains/upfp/view\
3  /test/FormatTransactionsAsCsvFileTest.java
4  index ee81d5c..883b0ca 100644
5  --- a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatTransaction\
6  sAsCsvFileTest.java
7  +++ b/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatTransaction\
8  sAsCsvFileTest.java
9  @@ -21,18 +21,25 @@ public class FormatTransactionsAsCsvFileTest {
10
11     private final CsvFormat<Transaction> transactionCsvFormat
12         = mockery.mock(CsvFormat.class, "transaction format");
13 -   private final CsvHeaderFormat csvHeaderFormat
14 -       = new TransactionsCsvHeader();
15 +   private final CsvHeaderFormat csvHeaderFormat = mockery
16 +       .mock(CsvHeaderFormat.class, "header format");
17
18     @Test
19     public void noTransactions() throws Exception {
20 +     mockery.checking(
21 +         new Expectations() {{
22 +           allowing(csvHeaderFormat).formatHeader();
23 +           will(returnValue("::header::"));
24 +         }});
25 +
26       final String text = formatTransactionsAsCsvFile(
27           Collections.<Transaction>emptyList(),
28           csvHeaderFormat, transactionCsvFormat);
29 -     assertThat(
30 -         text, matches(
```

```
-        "\\s*\"Date\",\\s*\"Category\"," +
-        "\\s*\"Amount\"\\s*"));
+    final List<String> lines = Arrays.asList(
+        text.split(
+            System.getProperty("line.separator")));
+    assertEquals(1, lines.size());
+    assertThat(lines.get(0), matches("::header::"));
     assertThat(
         text, endsWith(
         System.getProperty(
@@ -43,6 +50,8 @@ public class FormatTransactionsAsCsvFileTest {
   public void aFewTransactions() throws Exception {
     mockery.checking(
         new Expectations() {{
+          allowing(csvHeaderFormat).formatHeader();
+          will(returnValue("::header::"));
           allowing(transactionCsvFormat).format(
               with(
                   any(
@@ -64,10 +73,7 @@ public class FormatTransactionsAsCsvFileTest {
         text.split(
             System.getProperty("line.separator")));
     assertEquals(4, lines.size());
-    assertThat(
-        lines.get(0), matches(
-        "\\s*\"Date\",\\s*\"Category\"," +
-        "\\s*\"Amount\"\\s*"));
+    assertThat(lines.get(0), matches("::header::"));
     assertThat(lines.get(1), matches("::row 1::"));
     assertThat(lines.get(2), matches("::row 2::"));
     assertThat(lines.get(3), matches("::row 3::"));
```

I write a comment about the procedural implementation of `formatTransactionsAsCsvFile()`. I prefer this to the more functional `join()`, because I know how `PrintWriter.println()` handles newline characters on every platform, so I don't have to worry about `\r` versus `\n` versus `\n\r`, or whatever Windows, Mac OS and Linux all do. I don't want to know these things. Still, I do like the functional approach, and the code right now duplicates behavior in the `TransctionCsvFormat` while doing it a significantly different way. I can't leave it like that.

Snapshot 450: Replacing a procedure with a functional implementation

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatTran\
sactionsAsCsvFileTest.java b/AndroidFree/src/test/java/ca/jbrains/upfp/view\
/test/FormatTransactionsAsCsvFileTest.java
index 883b0ca..a4d8da4 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatTransaction\
sAsCsvFileTest.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatTransaction\
sAsCsvFileTest.java
@@ -1,17 +1,21 @@
 package ca.jbrains.upfp.view.test;

+import ca.jbrains.upfp.Conveniences;
 import ca.jbrains.upfp.model.test.*;
+import com.google.common.base.*;
 import com.google.common.collect.Lists;
+import com.sun.istack.internal.Nullable;
 import org.jmock.*;
 import org.jmock.integration.junit4.JMock;
 import org.joda.time.LocalDate;
 import org.junit.Test;
 import org.junit.runner.RunWith;

-import java.io.*;
 import java.util.*;

 import static ca.jbrains.hamcrest.RegexMatcher.matches;
+import static com.google.common.collect.Collections2
+    .transform;
 import static org.hamcrest.core.StringEndsWith.endsWith;
 import static org.junit.Assert.*;

@@ -22,7 +26,8 @@ public class FormatTransactionsAsCsvFileTest {
    private final CsvFormat<Transaction> transactionCsvFormat
        = mockery.mock(CsvFormat.class, "transaction format");
    private final CsvHeaderFormat csvHeaderFormat = mockery
-       .mock(CsvHeaderFormat.class, "header format");
+       .mock(
+           CsvHeaderFormat.class, "header format");

    @Test
```

```
   public void noTransactions() throws Exception {
@@ -37,13 +42,10 @@ public class FormatTransactionsAsCsvFileTest {
         csvHeaderFormat, transactionCsvFormat);
     final List<String> lines = Arrays.asList(
         text.split(
-            System.getProperty("line.separator")));
+            Conveniences.NEWLINE));
     assertEquals(1, lines.size());
     assertThat(lines.get(0), matches("::header::"));
-    assertThat(
-        text, endsWith(
-        System.getProperty(
-            "line.separator")));
+    assertThat(text, endsWith(Conveniences.NEWLINE));
   }

   @Test
@@ -71,16 +73,13 @@ public class FormatTransactionsAsCsvFileTest {

     final List<String> lines = Arrays.asList(
         text.split(
-            System.getProperty("line.separator")));
+            Conveniences.NEWLINE));
     assertEquals(4, lines.size());
     assertThat(lines.get(0), matches("::header::"));
     assertThat(lines.get(1), matches("::row 1::"));
     assertThat(lines.get(2), matches("::row 2::"));
     assertThat(lines.get(3), matches("::row 3::"));
-    assertThat(
-        text, endsWith(
-        System.getProperty(
-            "line.separator")));
+    assertThat(text, endsWith(Conveniences.NEWLINE));
   }

   private Transaction createAnyNonNullTransaction() {
@@ -90,19 +89,28 @@ public class FormatTransactionsAsCsvFileTest {
         26123));
   }

+  // REFACTOR Parameterise this in terms of Transaction
   private String formatTransactionsAsCsvFile(
```

```
83          List<Transaction> transactions,
84          CsvHeaderFormat csvHeaderFormat,
85  -       CsvFormat<Transaction> transactionCsvFormat
86  +       final CsvFormat<Transaction> transactionCsvFormat
87      ) {
88  -     // I'm not sure whether I prefer this to join on line
89  -     // .separator
90  -     final StringWriter text = new StringWriter();
91  -     final PrintWriter canvas = new PrintWriter(text);
92  -     canvas.println(csvHeaderFormat.formatHeader());
93  -     for (Transaction each : transactions) {
94  -       canvas.println(transactionCsvFormat.format(each));
95  -     }
96  -     return text.toString();
97  +     final List<String> lines = Lists.newArrayList(
98  +         csvHeaderFormat.formatHeader());
99  +     lines.addAll(
100 +         transform(
101 +             transactions,
102 +             new Function<Transaction, String>() {
103 +               @Override
104 +               public String apply(
105 +                   @Nullable Transaction transaction
106 +               ) {
107 +                 return transactionCsvFormat.format(
108 +                     transaction);
109 +               }
110 +             }));
111 +
112 +     return Joiner.on(Conveniences.NEWLINE).join(lines)
113 +         .concat(Conveniences.NEWLINE);
114     }
115  }
```

```
1  package ca.jbrains.upfp;
2
3  public class Conveniences {
4    public static final String NEWLINE = System.getProperty(
5        "line.separator");
6  }
```

I'd like to make this implementation even more generic, but I'd rather get to the finish line. I've made a note to go back and parameterise this implementation, since it knows nothing specific about `Transaction` objects.

The next step involves writing the CSV text out to file, but of course, the file doesn't care about the format of the text, except that it is text, so this code knows nothing about CSV. Now that I think more about it, I can't see writing a class for this behavior on its own. It seems too simple. I'd probably feel better writing it directly in the context of the code that will use it, then break it into a separate class later if I see the need. That means that I can skip directly to putting it all together, which I will do in the next session.

What's done

- Snapshot 400: We can format an empty collection of Transactions.
- Snapshot 410: We can format a collection of transactions as a CSV file, but I see a design problem.
- Snapshot 420: The Transactions CSV File format no longer needs to know the details of the format of the header row.
- Snapshot 430: We now check formatting the header of the CSV file on its own, and more directly.
- Snapshot 440: The code to format a collection of Transaction objects as CSV no longer mixes levels of abstraction.
- Snapshot 450: Replaced procedure with functional algorithm.

What's left to do

- Implement ExportAllTransactionsAction
  - Putting it all together
    - happy path
    - formatting a transaction fails
    - formatting the entire text fails, probably by running out of memory
    - writing the text to file fails
- Export transactions to CSV, but where do we store the file?
- Write contract tests for View once they stabilise
- Enter a transaction quickly and easily
- (2 upvotes) Delegate the Activity's BrowseTransactionsView implementation to a new class
- Find a library that provides an easy way to mark a property as "non-nullable"
  - It needs to let me throw a Programmer Mistake when I want to.
  - Evaluate the @Nullable and @NotNull annotations.
- Find a library that provides Rails-style object validations.

12 Putting Almost All of It Together

Now I get to see whether I have the Chunnel Problem that I expected to have when I decided to jump down and build from the bottom towards the top. First I'd like to check how I'm going to connect this code back to the Activity. Looking at `BrowseTransactionsActivity`, I remember that the Activity will provide a collection of `Transaction` objects and a `File` object whose path tells me where I can expect to safely write those transactions as CSV. I, playing the role of an `ExportAllTransactionsAction`, expect the Activity to have already verified that the path in question exists and that I have permission to write to it. The name `androidDevicePublicStorageGateway.findPublicExt` doesn't capture this more nuanced intent, so I add to the backlog an item to improve that. When implementing `ExportAllTransactionsAction`, I don't mind handling generic I/O failures, but I expect those up the call stack to take responsibility for setting the action up to succeed.

I start with the happy path, in which I write the file successfully to disk. In the process of writing this test, I see more clearly the pieces I need. The Export Action has one primary responsibility: to write to disk whatever the CSV file formatter says to write to disk. This makes everything really quite straightforward, but points out a naming problem in what I've built so far.

Improving Names

I have the interface `CsvFormat<ValueType>`, which represents a format for representing `ValueType` objects as a row in a CSV file. (Now that I write this, "Value Type" doesn't fit, either, since `Transaction` is an Entity class, not a Value Type, so I might have another naming problem to fix.) I also have the method `formatTransactionsAsCsvFile(transactions, headerFormat, rowFormat)`, still buried in `FormatTransactionsAsCsvFileTest`, which I've not yet extracted to its own production class. I can already see that when I do that, I'll want to extract an interface from it, so that I can stub it in the tests for `ExportAllTransactionsAction`. It, too, represents a CSV Format, but for an entire file, rather than for a single row. This creates a naming collision between `CsvFormat`, meaning for a row, and this about-to-be-born interface. Fortunately, I can fix these problems easily with some renaming, so I do that before I finish writing the new test.

After stashing my incomplete test – and I really like that `git` lets me do that – I fix the incorrect and imprecise names.

Or not. As I started renaming the interfaces, I noticed that although `CsvRowFormat` makes sense for `Transaction`, it doesn't fit `Date`, `Category`, nor `Amount`, which I'd classify as "elements". I feel a Composite pattern coming on, as I have a file (the whole) which consists of rows (the parts), and then even in each row (a small whole) I have elements (smaller parts). With Composite, I expect to use the same interface for the whole and the parts, so perhaps I can do that here. How could I make `formatTransactionsAsCsvFile()` implement `CsvFormat`? By implementing

`CsvFormat<Collection<Transaction>>` and moving the parameters `headerFormat` and `rowFormat` into the extracted class's constructor. That sounds like it would fit, so let me try it.

Snapshot 460: Moving production code to a new production class

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatTran\
sactionsAsCsvFileTest.java b/AndroidFree/src/test/java/ca/jbrains/upfp/view\
/test/FormatTransactionsAsCsvFileTest.java
index a4d8da4..7865357 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatTransaction\
sAsCsvFileTest.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatTransaction\
sAsCsvFileTest.java
@@ -2,9 +2,7 @@ package ca.jbrains.upfp.view.test;

 import ca.jbrains.upfp.Conveniences;
 import ca.jbrains.upfp.model.test.*;
-import com.google.common.base.*;
 import com.google.common.collect.Lists;
-import com.sun.istack.internal.Nullable;
 import org.jmock.*;
 import org.jmock.integration.junit4.JMock;
 import org.joda.time.LocalDate;
@@ -14,8 +12,6 @@ import org.junit.runner.RunWith;
 import java.util.*;

 import static ca.jbrains.hamcrest.RegexMatcher.matches;
-import static com.google.common.collect.Collections2
-    .transform;
 import static org.hamcrest.core.StringEndsWith.endsWith;
 import static org.junit.Assert.*;

@@ -28,6 +24,10 @@ public class FormatTransactionsAsCsvFileTest {
    private final CsvHeaderFormat csvHeaderFormat = mockery
        .mock(
            CsvHeaderFormat.class, "header format");
+  private final TransactionsCsvFileFormat
+      transactionsCsvFileFormat
+      = new TransactionsCsvFileFormat(
+      csvHeaderFormat, transactionCsvFormat);

    @Test
```

```
38     public void noTransactions() throws Exception {
39  @@ -37,9 +37,8 @@ public class FormatTransactionsAsCsvFileTest {
40             will(returnValue("::header::"));
41           }});
42
43  -    final String text = formatTransactionsAsCsvFile(
44  -        Collections.<Transaction>emptyList(),
45  -        csvHeaderFormat, transactionCsvFormat);
46  +    final String text = transactionsCsvFileFormat.format(
47  +        Collections.<Transaction>emptyList());
48       final List<String> lines = Arrays.asList(
49           text.split(
50               Conveniences.NEWLINE));
51  @@ -64,12 +63,11 @@ public class FormatTransactionsAsCsvFileTest {
52                     "::row 2::"), returnValue("::row 3::")));
53           }});
54
55  -    final String text = formatTransactionsAsCsvFile(
56  +    final String text = transactionsCsvFileFormat.format(
57           Lists.newArrayList(
58               createAnyNonNullTransaction(),
59               createAnyNonNullTransaction(),
60  -            createAnyNonNullTransaction()), csvHeaderFormat,
61  -        transactionCsvFormat);
62  +            createAnyNonNullTransaction()));
63
64       final List<String> lines = Arrays.asList(
65           text.split(
66  @@ -88,29 +86,4 @@ public class FormatTransactionsAsCsvFileTest {
67           "irrelevant category"), Amount.cents(
68           26123));
69     }
70  -
71  -  // REFACTOR Parameterise this in terms of Transaction
72  -  private String formatTransactionsAsCsvFile(
73  -      List<Transaction> transactions,
74  -      CsvHeaderFormat csvHeaderFormat,
75  -      final CsvFormat<Transaction> transactionCsvFormat
76  -  ) {
77  -    final List<String> lines = Lists.newArrayList(
78  -        csvHeaderFormat.formatHeader());
79  -    lines.addAll(
```

```
-        transform(
-            transactions,
-            new Function<Transaction, String>() {
-              @Override
-              public String apply(
-                  @Nullable Transaction transaction
-              ) {
-                return transactionCsvFormat.format(
-                    transaction);
-              }
-            }));
-
-    return Joiner.on(Conveniences.NEWLINE).join(lines)
-        .concat(Conveniences.NEWLINE);
-  }
 }
```

Snapshot 460: The File format consists of the Header format and the Body format

```
package ca.jbrains.upfp.view.test;

import ca.jbrains.upfp.Conveniences;
import ca.jbrains.upfp.model.test.Transaction;
import com.google.common.base.*;
import com.google.common.collect.*;
import com.sun.istack.internal.Nullable;

import java.util.List;

public class TransactionsCsvFileFormat
    implements CsvFormat<List<Transaction>> {
  private CsvHeaderFormat csvHeaderFormat;
  private CsvFormat<Transaction> transactionCsvFormat;

  public TransactionsCsvFileFormat(
      CsvHeaderFormat csvHeaderFormat,
      final CsvFormat<Transaction> transactionCsvFormat
  ) {
    this.csvHeaderFormat = csvHeaderFormat;
    this.transactionCsvFormat = transactionCsvFormat;
  }
```

```
23
24    @Override
25    public String format(List<Transaction> transactions) {
26      final List<String> lines = Lists.newArrayList(
27          csvHeaderFormat.formatHeader());
28      lines.addAll(
29          Collections2.transform(
30              transactions,
31              new Function<Transaction, String>() {
32                @Override
33                public String apply(
34                    @Nullable Transaction transaction
35                ) {
36                  return transactionCsvFormat.format(
37                      transaction);
38                }
39              }));
40
41      return Joiner.on(Conveniences.NEWLINE).join(lines)
42          .concat(Conveniences.NEWLINE);
43    }
44  }
```

Snapshot 460: Removing a misleading name, even if it means using a vague name

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/CsvFormat.\
java b/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/CsvFormat.java
index 0fe323b..d047488 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/CsvFormat.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/CsvFormat.java
@@ -1,5 +1,5 @@
 package ca.jbrains.upfp.view.test;

-public interface CsvFormat<ValueType> {
-  String format(ValueType value);
+public interface CsvFormat<T> {
+  String format(T t);
 }
```

In the end, the type parameter for `CsvFormat` is neither (only) an Entity nor a Value. Instead, it might be either. As a result, I replace a misleading-but-precise name with a vague-but-accurate name, since

precision without accuracy causes problems[1]. On the bright side, `TransactionsCsvFileFormat` now implements `CsvFormat<List<Transaction>>`, which means I can now implement `ExportAllTransactionsAction` without knowing any details of its collaborators.

Back to the Happy Path

Finally! I want to get back to this happy path test I've started writing. I don't want to have to check the contents of the file on disk, especially considering that I will stub the `CsvFormat` to give me placeholder text, so I hide this behind an interface. This feels a bit silly, and I have to prepare myself for the possibility that this *is* silly and I will undo it. I don't mind. I'd prefer to use someone else's library for this, but I took a look at Jakarta Commons VFS, and it goes far, far, far beyond what I need right now.

Snapshot 470: Export needs to write the correct format of a Transactions CSV file to disk

```
1  package ca.jbrains.upfp.presenter.test;
2
3  import ca.jbrains.upfp.model.test.Transaction;
4  import ca.jbrains.upfp.view.test.CsvFormat;
5  import com.google.common.collect.Lists;
6  import org.jmock.*;
7  import org.jmock.integration.junit4.JMock;
8  import org.junit.Test;
9  import org.junit.runner.RunWith;
10
11 import java.io.File;
12 import java.util.List;
13
14 @RunWith(JMock.class)
15 public class ExportAllTransactionsAsCsvToFileActionTest {
16   private final Mockery mockery = new Mockery();
17   private final CsvFormat<List<Transaction>>
18       transactionsFileFormat = mockery.mock(
19       CsvFormat.class, "transactions file format");
20   private final WriteTextToFileAction writeTextToFileAction
21       = mockery.mock(WriteTextToFileAction.class);
22
23   @Test
24   public void happyPath() throws Exception {
25     final String csvText = "::the transactions as CSV::";
```

[1] I've summarised my model for improving names in software systems in a handy diagram.

```
26      final File path = new File("/irrelevant/path");
27
28      mockery.checking(
29          new Expectations() {{
30            allowing(transactionsFileFormat).format(
31                with(
32                    any(
33                        List.class)));
34            will(returnValue(csvText));
35
36            oneOf(writeTextToFileAction).writeTextToFile(
37                csvText, path);
38          }});
39
40      final List<Transaction> transactions = Lists
41          .newArrayList();
42
43      exportAllTransactionsAsCsvToFileAction(
44          transactions, path);
45    }
46
47    private void exportAllTransactionsAsCsvToFileAction(
48        List<Transaction> transactions, File path
49    ) {
50      final String text = transactionsFileFormat.format(
51          transactions);
52      writeTextToFileAction.writeTextToFile(text, path);
53    }
54  }
```

Snapshot 470: Export now writes the correct format of a Transactions CSV file to disk

```
1  package ca.jbrains.upfp.presenter.test;
2
3  import java.io.File;
4
5  public interface WriteTextToFileAction {
6    void writeTextToFile(String csvText, File path);
7  }
```

The Other Cases

Now for the other cases. I look at the tests for formatting a list of `Transaction` objects and see no obvious failure cases, so I don't need to simulate failures in formatting, but writing to file can absolutely fail, so I'd better deal with that. First, I wonder whether this method, `exportAllTransactionsAsCsvToFileAction()` will handle different file-based errors differently, because if not, then I'd rather push those tests down into the implementation of `WriteTextToFile`. Looking back at the backlog, the potential failure cases involve running out of memory or disk space. I now realise that overwriting an existing file might merit its own special case test, but doesn't necessarily represent a failure case. I can't imagine recovering from either an out of memory or out of disk space error, and in any case, I don't know how to distinguish an out-of-disk-space error from any other garden-variety `IOException`, so I conclude that no, `exportAllTransactionsAsCsvToFileAction()` won't handle different I/O failures differently, so it can assume that `WriteTextToFile` might throw only a generic `IOException`. I see no reason to avoid `IOException` here, even though I would normally avoid a checked exception.

Back to the case where writing the text to file fails, how should the export action respond? I lean towards throwing an exception up to the Activity, which can then interpret the failure any way it wants. I can't think of another interpretation than "disk full" for an I/O exception while writing text to a file, but most importantly, if I plan to assume this interpretation, I'd rather do it further up the call stack, in order to decrease the export action's dependence on its context. Strangely enough, `IOException` would achieve that goal quite well, so I will try simply propagating the `IOException`, which I ordinarily dislike. I can always wrap it in a more descriptive exception class later, if the design signals that need to me.

Snapshot 480: Export now handles an IOException while writing to disk

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/Expor\
tAllTransactionsAsCsvToFileActionTest.java b/AndroidFree/src/test/java/ca/j\
brains/upfp/presenter/test/ExportAllTransactionsAsCsvToFileActionTest.java
index 9d08134..1262f61 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/ExportAllTra\
nsactionsAsCsvToFileActionTest.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/ExportAllTra\
nsactionsAsCsvToFileActionTest.java
@@ -8,9 +8,11 @@ import org.jmock.integration.junit4.JMock;
 import org.junit.Test;
 import org.junit.runner.RunWith;

-import java.io.File;
+import java.io.*;
 import java.util.List;
```

```
16
17 +import static org.junit.Assert.*;
18 +
19  @RunWith(JMock.class)
20  public class ExportAllTransactionsAsCsvToFileActionTest {
21    private final Mockery mockery = new Mockery();
22 @@ -44,9 +46,40 @@ public class ExportAllTransactionsAsCsvToFileActionTest \
23 {
24          transactions, path);
25    }
26
27 +  @Test
28 +  public void writingToFileFails() throws Exception {
29 +    final IOException ioFailure = new IOException(
30 +        "You probably ran out of disk space.");
31 +
32 +    mockery.checking(
33 +        new Expectations() {{
34 +          ignoring(transactionsFileFormat);
35 +
36 +          allowing(writeTextToFileAction).writeTextToFile(
37 +              with(any(String.class)), with(
38 +              any(
39 +                  File.class)));
40 +          will(throwException(ioFailure));
41 +        }});
42 +
43 +    final List<Transaction> irrelevantTransactions = Lists
44 +        .newArrayList();
45 +    try {
46 +      exportAllTransactionsAsCsvToFileAction(
47 +          irrelevantTransactions, new File(
48 +          "/irrelevant/path"));
49 +      fail(
50 +          "Writing text to disk failed, " +
51 +          "but you don't care?!");
52 +    } catch (IOException success) {
53 +      assertSame(ioFailure, success);
54 +    }
55 +  }
56 +
57 +
```

```
   private void exportAllTransactionsAsCsvToFileAction(
       List<Transaction> transactions, File path
-  ) {
+  ) throws IOException {
     final String text = transactionsFileFormat.format(
         transactions);
     writeTextToFileAction.writeTextToFile(text, path);
```

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/Write\
TextToFileAction.java b/AndroidFree/src/test/java/ca/jbrains/upfp/presenter\
/test/WriteTextToFileAction.java
index 7f6884c..e59c6ae 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/WriteTextToF\
ileAction.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/WriteTextToF\
ileAction.java
@@ -1,7 +1,8 @@
 package ca.jbrains.upfp.presenter.test;

-import java.io.File;
+import java.io.*;

 public interface WriteTextToFileAction {
-  void writeTextToFile(String csvText, File path);
+  void writeTextToFile(String csvText, File path)
+      throws IOException;
 }
```

It feels strange to me to write this test. You can see that it passes merely by adding a checked exception to the list that a method throws. It also feels like a long test to write just to check that `exportAllTransactionsAsCsvToFileAction()` *doesn't* handle an exception. Doing this once might not cause any problems. Doing this several times in the same layer represents a stronger signal that I need to separate behaviors differently. I'll keep an eye on this.

What happens if the file already exists? I don't think I can answer that until I try implementing `WriteTextToFileAction` and actually write the file to disk – I don't remember the details. I'll put that on the backlog and pick that up in the next session.

What's done

- Snapshot 460:
  - Extracted production code from test file.
  - Promoted non-interface-conforming method parameters to constructor parameters, in preparation for implementing `CsvFormat<List<Transaction>>`.
  - TransactionsCsvFileFormat now implements `CsvFormat<List<Transaction>>`.
  - Replaced misleading type name ValueType with vague-but-accurate type name T.
- Snapshot 470: We now ask to write a list of Transaction objects to file, as long as nothing goes wrong.
- Snapshot 480: Verified that the Export Action propagates IOExceptions write writing to file.

What's left to do

- Implement ExportAllTransactionsAction
  - Implement WriteTextToFile
    * happy path
    * I/O failure
    * file already exists, overwrite (exception?)
- Export transactions to CSV, but where do we store the file?
- Write contract tests for View once they stabilise
- Enter a transaction quickly and easily
- (2 upvotes) Delegate the Activity's BrowseTransactionsView implementation to a new class
- Find a library that provides an easy way to mark a property as "non-nullable"
  - It needs to let me throw a Programmer Mistake when I want to.
  - Evaluate the @Nullable and @NotNull annotations.
- Find a library that provides Rails-style object validations.
- Rename androidDevicePublicStorageGateway.findPublicExternalStorageDirectory() to something more like findSafePlaceToWriteAFile(), which better captures its intent.

13 Writing Text to File

Now I implement the interface that I thought, for a moment, too trivial to extract. I don't like such a concrete name as `WriteTextToFileAction` for an interface because, well, what do I name the class? The concreteness of `File` already appears in the name. Definitely not `WriteTextToFileActionImpl`, at least not for more than a few seconds. As I write this, I realise that the Export Action wants to write the CSV content to some kind of `TextCanvas`, and doesn't actually care whether the content ends up in a `File`. The Activity, on the other hand, wants to write that information to a `File`. This implies changing some of the work I've some completed.

Which should I do first: implement `WriteTextToFileAction` or refactor the related interfaces? I will probably find it easier to refactor after implementing the entire cluster of classes, because I will have two things: concrete evidence of inappropriate boundaries and rapid feedback about improving those boundaries. I risk, of course, doing work only to throw it away. Since I think I can implement `WriteTextToFileAction` in only about 15 minutes, I assume the risk of having to throw it away. Moreover, I intend to refactor, not rewrite, and so I won't throw anything away. This becomes my plan of attack:

1. Implement `WriteTextToFileAction`.
2. Refactor `WriteTextToFileAction`, `ExportAllTransactionsAction` and `BrowseTransactionActivity` to shift the focus away from `File`.
    1. Add a new interface and implementation that delegates to a `WriteTextToFileActionImpl`.
    2. Migrate clients away from `WriteTextToFile` towards the new interface (`TextWriter`? Interesting.)
    3. Retire the obsolete code.
    4. Resist the urge to marry the new code with the existing `java.io` API, because they don't use interfaces.

I expect to end up with a narrowing API that provides a gateway to `java.io`, which amounts to applying the Interface Segregation Principle, and that gives me a warm, fuzzy feeling.

In the process of making the first test pass, I flash back to 2000 and remember why I don't test-drive code that directly uses the `java.io` API nor the file system. Specifically, cleaning up the file system at the beginning of the tests. In 2012, `java.io` still doesn't make it easy to remove a directory tree. For this, I include Jakarta Commons IO, but only as a testing library. I don't know yet how much I need to worry about my application exploding in size from too many libraries.

Snapshot 490: Adding a library for testing

```
 .gitignore                                                                 \
          |   1 +
 .idea/libraries/commons_io_2_4.xml                                         \
          |   9 +++++++++
 AndroidFree/AndroidFree.iml                                                \
          |   1 +
 AndroidFree/src/libs/test/commons-io-2.4.jar                               \
          | Bin 0 -> 185140 bytes
 AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/WriteTextToFileAc\
tion.java |   2 +-
 AndroidFree/src/test/java/ca/jbrains/upfp/view/WriteTextToFileActionImpl.j\
ava       |  17 ++++++++++++++++
 AndroidFree/src/test/java/ca/jbrains/upfp/view/test/WriteTextToFileTest.ja\
va        |  38 ++++++++++++++++++++++++++++++++++++
 7 files changed, 67 insertions(+), 1 deletion(-)
```

Snapshot 490: We need to actually write text out to a file

```
package ca.jbrains.upfp.view.test;

import ca.jbrains.upfp.view.WriteTextToFileActionImpl;
import org.apache.commons.io.FileUtils;
import org.junit.*;

import java.io.File;

import static org.junit.Assert.*;

public class WriteTextToFileTest {
  private static final File testOutputDirectory = new File(
      "./test/output/WriteTextToFileTest/");

  @BeforeClass
  public static void initialiseTestOutputAreaOnFileSystem()
      throws Exception {
    FileUtils.deleteDirectory(testOutputDirectory);
    assertTrue(
        String.format(
```

```
            "Couldn't create test output directory at %1$s",
            testOutputDirectory.getAbsolutePath()),
        testOutputDirectory.mkdirs());
  }

  @Test
  public void happyPath() throws Exception {
    final File file = new File(
        testOutputDirectory, "happyPath.csv");

    new WriteTextToFileActionImpl().writeTextToFile(
        "::text::", file);

    assertEquals(
        "::text::", FileUtils.readFileToString(
        file));
  }
}
```

Snapshot 490: We need to keep the test output out of the code repository

```
diff --git a/.gitignore b/.gitignore
index 71aeeeb..df3b7ed 100644
--- a/.gitignore
+++ b/.gitignore
@@ -4,3 +4,4 @@ out
 tmp
 src/gen
 gen
+test/output
```

Snapshot 490: We now actually write text out to a file

```
package ca.jbrains.upfp.view;

import ca.jbrains.upfp.presenter.test.WriteTextToFileAction;

import java.io.*;

public class WriteTextToFileActionImpl
    implements WriteTextToFileAction {
  @Override
  public void writeTextToFile(String text, File path)
      throws IOException {
    final FileWriter fileWriter = new FileWriter(path);
    fileWriter.write(text);
    fileWriter.flush();
    fileWriter.close();
  }
}
```

Snapshot 490: A small, but significant, improvement in name

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/Write\
TextToFileAction.java b/AndroidFree/src/test/java/ca/jbrains/upfp/presenter\
/test/WriteTextToFileAction.java
index e59c6ae..81c9dee 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/WriteTextToF\
ileAction.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/WriteTextToF\
ileAction.java
@@ -3,6 +3,6 @@ package ca.jbrains.upfp.presenter.test;
 import java.io.*;

 public interface WriteTextToFileAction {
-  void writeTextToFile(String csvText, File path)
+  void writeTextToFile(String text, File path)
       throws IOException;
 }
```

In checking the case of an I/O failure, I resort to subclassing to test in order to substitute a crash test dummy `FileWriter`. I don't like to do this, and I'd ordinarily Replace Inheritance with Delegation next, but I don't think the resulting code would improve any, so I leave it as is.

Snapshot 500: Subclass to test in order to install a crash test dummy

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/WriteTextT\
oFileTest.java b/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/WriteT\
extToFileTest.java
index 97840ea..863711b 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/WriteTextToFileTe\
st.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/WriteTextToFileTe\
st.java
@@ -4,7 +4,7 @@ import ca.jbrains.upfp.view.WriteTextToFileActionImpl;
 import org.apache.commons.io.FileUtils;
 import org.junit.*;

-import java.io.File;
+import java.io.*;

 import static org.junit.Assert.*;

@@ -35,4 +35,30 @@ public class WriteTextToFileTest {
         "::text::", FileUtils.readFileToString(
         file));
   }
+
+  @Test
+  public void ioFailure() throws Exception {
+    final IOException ioFailure = new IOException(
+        "Simulating a failure writing to the file.");
+    try {
+      new WriteTextToFileActionImpl() {
+        @Override
+        protected FileWriter fileWriterOn(File path)
+            throws IOException {
+          return new FileWriter(path) {
+            @Override
+            public void write(String str, int off, int len)
+                throws IOException {
+              throw ioFailure;
```

```
+            }
+          };
+        }
+      }.writeTextToFile(
+          "::text::", new File(
+          testOutputDirectory, "anyWritableFile.txt"));
+      fail("How did you survive the I/O failure?!");
+    } catch (IOException success) {
+      if (success != ioFailure) throw success;
+    }
+  }
 }
```

Snapshot 500: We can override the FileWriter

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/view/WriteTextToFile\
ActionImpl.java b/AndroidFree/src/test/java/ca/jbrains/upfp/view/WriteTextT\
oFileActionImpl.java
index ce773e1..8c4fd39 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/view/WriteTextToFileActionI\
mpl.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/view/WriteTextToFileActionI\
mpl.java
@@ -9,9 +9,14 @@ public class WriteTextToFileActionImpl
   @Override
   public void writeTextToFile(String text, File path)
       throws IOException {
-    final FileWriter fileWriter = new FileWriter(path);
+    final FileWriter fileWriter = fileWriterOn(path);
     fileWriter.write(text);
     fileWriter.flush();
     fileWriter.close();
   }
+
+  protected FileWriter fileWriterOn(File path)
+      throws IOException {
+    return new FileWriter(path);
+  }
 }
```

Finally, I discover that, by default, `WriteTextToFileActionImpl` overwrites any file that might already exist. I don't mind that behavior, so I don't change it. I write a test to record this observation, a kind of Learning Test.

Snapshot 510: Documenting the behavior when the output file already exists

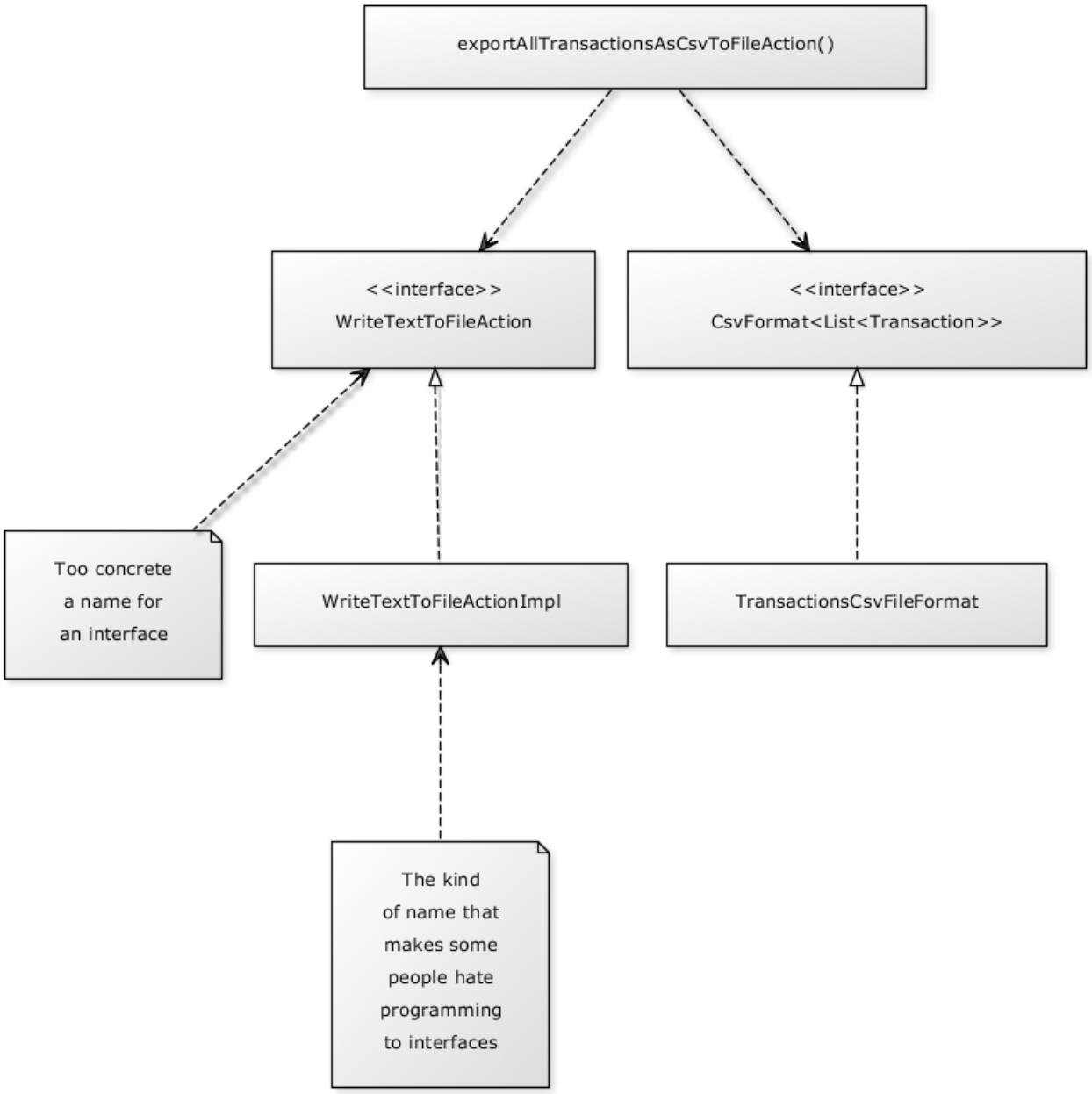
```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/WriteTextT\
oFileTest.java b/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/WriteT\
extToFileTest.java
index 863711b..c8ffb8e 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/WriteTextToFileTe\
st.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/view/test/WriteTextToFileTe\
st.java
@@ -61,4 +61,19 @@ public class WriteTextToFileTest {
       if (success != ioFailure) throw success;
     }
   }
+
+  @Test
+  public void fileAlreadyExists() throws Exception {
+    final File file = new File(
+        testOutputDirectory, "alreadyExists.txt");
+    FileUtils.write(
+        file, "There is already something here.");
+
+    new WriteTextToFileActionImpl().writeTextToFile(
+        "::text::", file);
+
+    assertEquals(
+        "::text::", FileUtils.readFileToString(
+        file));
+  }
 }
```

I can't think of any further tests for `WriteTextToFileActionImpl`, so I move on to the refactoring I intended to do.

Fixing the Strange Interface Boundaries

Diagrams like these make designers like me look like idiots, in particular because of the `Impl`. This triggers skeptical designers to doubt the value of programming to interfaces, because we do it "only

for testing" and it "needlessly doubles the number of types in the system".



Questionable, at best; idiotic, at worst

When I choose names that lead me to a design like this, I interpret that as a strong signal that I've drawn the boundaries between interface and implementation incorrectly. It almost always means that I've violated the Dependency Inversion Principle by allowing concrete implementation details to leak up the call stack. In this case, I've unnecessarily burdened clients with the implementation detail of writing the CSV content to a **file**. That doesn't sound like much, nor does it sound unusual, but nevertheless, an implementation detail leaks into the client, and I'd rather fix that before it

becomes a serious problem.

I fix this by working from the implementation backward. When I name an implementation class with `Impl`, it means that I haven't differentiated the implementation from the interface enough. Writing to **file** makes this implementation special, so `File` should remain in the name of the implementation class, but not in the name of the interface. This leaves `WriteTextAction`, which sounds good enough to me for now. It also sounds delightfully generic and therefore reusable.

I introduce the new interface, `WriteTextAction`, which completely encapsulates the client's interactions with `java.io.FileWriter`. Next, I change `exportAllTransactionsAsCsvToFileAction()` to use the new `WriteTextAction`.

Remember: add the new interface, migrate the clients, then remove the old interface.

Snapshot 520: The new interface for writing text

```
package ca.jbrains.upfp.presenter.test;

import java.io.IOException;

public interface WriteTextAction {
  void writeText(String text) throws IOException;
}
```

Snapshot 530: The first step in migrating clients to the new interface

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/Expor\
tAllTransactionsAsCsvToFileActionTest.java b/AndroidFree/src/test/java/ca/j\
brains/upfp/presenter/test/ExportAllTransactionsAsCsvToFileActionTest.java
index 1262f61..3080d8d 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/ExportAllTra\
nsactionsAsCsvToFileActionTest.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/ExportAllTra\
nsactionsAsCsvToFileActionTest.java
@@ -78,10 +78,16 @@ public class ExportAllTransactionsAsCsvToFileActionTest\
 {

```

```
12
13     private void exportAllTransactionsAsCsvToFileAction(
14  -      List<Transaction> transactions, File path
15  +      List<Transaction> transactions, final File path
16     ) throws IOException {
17       final String text = transactionsFileFormat.format(
18           transactions);
19  -    writeTextToFileAction.writeTextToFile(text, path);
20  +    new WriteTextAction() {
21  +      @Override
22  +      public void writeText(String text)
23  +          throws IOException {
24  +        writeTextToFileAction.writeTextToFile(text, path);
25  +      }
26  +    }.writeText(text);
27     }
28   }
```

Unfortunately, I don't see how to go any further without a minor distraction. I want to abstract away the path of the file to which to write the text, but `exportAllTransactionsAsCsvToFileAction()` takes it as a parameter. See how this innocuous leak has already got in the way?! What I really want is a partial function `exportAllTransactionsAsCsv(transactions)` that has already decided where to write the transactions, namely to a previously-designated file. When using objects, I do this by moving the method parameter into the constructor of, in this case, a new class.

First, I introduce the new class.

Snapshot 540: Moving production code into a new class

```
1  diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/Expor\
2  tAllTransactionsAsCsvToFileActionTest.java b/AndroidFree/src/test/java/ca/j\
3  brains/upfp/presenter/test/ExportAllTransactionsAsCsvToFileActionTest.java
4  index 3080d8d..463d17e 100644
5  --- a/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/ExportAllTra\
6  nsactionsAsCsvToFileActionTest.java
7  +++ b/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/ExportAllTra\
8  nsactionsAsCsvToFileActionTest.java
9  @@ -21,6 +21,10 @@ public class ExportAllTransactionsAsCsvToFileActionTest \
10 {
11         CsvFormat.class, "transactions file format");
12     private final WriteTextToFileAction writeTextToFileAction
13         = mockery.mock(WriteTextToFileAction.class);
```

```
+  private final ExportAllTransactionsAsCsvToFileAction
+      exportAllTransactionsAsCsvToFileAction
+      = new ExportAllTransactionsAsCsvToFileAction(
+      transactionsFileFormat, writeTextToFileAction);

   @Test
   public void happyPath() throws Exception {
@@ -42,8 +46,9 @@ public class ExportAllTransactionsAsCsvToFileActionTest {
     final List<Transaction> transactions = Lists
         .newArrayList();

-    exportAllTransactionsAsCsvToFileAction(
-        transactions, path);
+    exportAllTransactionsAsCsvToFileAction
+        .exportAllTransactionsAsCsvToFileAction(
+            transactions, path);
   }

   @Test
@@ -65,9 +70,10 @@ public class ExportAllTransactionsAsCsvToFileActionTest \
{
     final List<Transaction> irrelevantTransactions = Lists
         .newArrayList();
     try {
-      exportAllTransactionsAsCsvToFileAction(
-          irrelevantTransactions, new File(
-          "/irrelevant/path"));
+      exportAllTransactionsAsCsvToFileAction
+          .exportAllTransactionsAsCsvToFileAction(
+              irrelevantTransactions, new File(
+              "/irrelevant/path"));
       fail(
           "Writing text to disk failed, " +
           "but you don't care?!");
@@ -75,19 +81,4 @@ public class ExportAllTransactionsAsCsvToFileActionTest \
{
       assertSame(ioFailure, success);
     }
   }
-
-
-  private void exportAllTransactionsAsCsvToFileAction(
```

```
-      List<Transaction> transactions, final File path
-  ) throws IOException {
-    final String text = transactionsFileFormat.format(
-        transactions);
-    new WriteTextAction() {
-      @Override
-      public void writeText(String text)
-          throws IOException {
-        writeTextToFileAction.writeTextToFile(text, path);
-      }
-    }.writeText(text);
-  }
 }
```

Snapshot 540: The new class in question

```
package ca.jbrains.upfp.presenter.test;

import ca.jbrains.upfp.model.test.Transaction;
import ca.jbrains.upfp.view.test.CsvFormat;

import java.io.*;
import java.util.List;

public class ExportAllTransactionsAsCsvToFileAction {
  private final CsvFormat<List<Transaction>>
      transactionsFileFormat;
  private final WriteTextToFileAction writeTextToFileAction;

  public ExportAllTransactionsAsCsvToFileAction(
      CsvFormat<List<Transaction>> transactionsFileFormat,
      WriteTextToFileAction writeTextToFileAction
  ) {
    this.transactionsFileFormat = transactionsFileFormat;
    this.writeTextToFileAction = writeTextToFileAction;
  }

  public void exportAllTransactionsAsCsvToFileAction(
      List<Transaction> transactions, final File path
  ) throws IOException {
    final String text = transactionsFileFormat.format(
```

```
        transactions);
    new WriteTextAction() {
      @Override
      public void writeText(String text)
          throws IOException {
        writeTextToFileAction.writeTextToFile(text, path);
      }
    }.writeText(text);
  }
}
```

Next, I move the method parameter to the new class's constructor.

Snapshot 550: The body of the test shouldn't care about the file parameter

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/Expor\
tAllTransactionsAsCsvToFileActionTest.java b/AndroidFree/src/test/java/ca/j\
brains/upfp/presenter/test/ExportAllTransactionsAsCsvToFileActionTest.java
index 463d17e..62efc58 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/ExportAllTra\
nsactionsAsCsvToFileActionTest.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/ExportAllTra\
nsactionsAsCsvToFileActionTest.java
@@ -24,7 +24,8 @@ public class ExportAllTransactionsAsCsvToFileActionTest {
   private final ExportAllTransactionsAsCsvToFileAction
       exportAllTransactionsAsCsvToFileAction
       = new ExportAllTransactionsAsCsvToFileAction(
-      transactionsFileFormat, writeTextToFileAction);
+      transactionsFileFormat, writeTextToFileAction,
+      new File("/irrelevant/path"));

   @Test
   public void happyPath() throws Exception {
@@ -48,7 +49,7 @@ public class ExportAllTransactionsAsCsvToFileActionTest {

     exportAllTransactionsAsCsvToFileAction
         .exportAllTransactionsAsCsvToFileAction(
-            transactions, path);
+            transactions);
   }

```

```
   @Test
@@ -72,8 +73,7 @@ public class ExportAllTransactionsAsCsvToFileActionTest {
     try {
       exportAllTransactionsAsCsvToFileAction
           .exportAllTransactionsAsCsvToFileAction(
-              irrelevantTransactions, new File(
-              "/irrelevant/path"));
+              irrelevantTransactions);
       fail(
           "Writing text to disk failed, " +
           "but you don't care?!");
```

Snapshot 550: Making the file parameter easier to encapsulate

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/Expor\
tAllTransactionsAsCsvToFileAction.java b/AndroidFree/src/test/java/ca/jbrai\
ns/upfp/presenter/test/ExportAllTransactionsAsCsvToFileAction.java
index cba9f23..4022358 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/ExportAllTra\
nsactionsAsCsvToFileAction.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/ExportAllTra\
nsactionsAsCsvToFileAction.java
@@ -10,17 +10,20 @@ public class ExportAllTransactionsAsCsvToFileAction {
   private final CsvFormat<List<Transaction>>
       transactionsFileFormat;
   private final WriteTextToFileAction writeTextToFileAction;
+  private final File destinationFile;

   public ExportAllTransactionsAsCsvToFileAction(
       CsvFormat<List<Transaction>> transactionsFileFormat,
-      WriteTextToFileAction writeTextToFileAction
+      WriteTextToFileAction writeTextToFileAction,
+      File destinationFile
   ) {
     this.transactionsFileFormat = transactionsFileFormat;
     this.writeTextToFileAction = writeTextToFileAction;
+    this.destinationFile = destinationFile;
   }

   public void exportAllTransactionsAsCsvToFileAction(
-      List<Transaction> transactions, final File path
```

```
+      List<Transaction> transactions
   ) throws IOException {
     final String text = transactionsFileFormat.format(
         transactions);
@@ -28,7 +31,8 @@ public class ExportAllTransactionsAsCsvToFileAction {
       @Override
       public void writeText(String text)
           throws IOException {
-        writeTextToFileAction.writeTextToFile(text, path);
+        writeTextToFileAction.writeTextToFile(
+            text, destinationFile);
       }
     }.writeText(text);
   }
```

Now when I look at the constructor for `ExportAllTransactionsAsCsvToFileAction`, I see that I can combine the `WriteTextToFileAction` and the `File` parameters, because, well, they belong together! I can then hide them behind a `WriteTextAction`. I've condensed the microsteps here, but you can follow them in the code repository.

Snapshot 560: The test no longer knows anything about a file path

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/Expor\
tAllTransactionsAsCsvToFileActionTest.java b/AndroidFree/src/test/java/ca/j\
brains/upfp/presenter/test/ExportAllTransactionsAsCsvToFileActionTest.java
index 62efc58..0ee8669 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/ExportAllTra\
nsactionsAsCsvToFileActionTest.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/ExportAllTra\
nsactionsAsCsvToFileActionTest.java
@@ -19,13 +19,12 @@ public class ExportAllTransactionsAsCsvToFileActionTest\
 {
   private final CsvFormat<List<Transaction>>
       transactionsFileFormat = mockery.mock(
       CsvFormat.class, "transactions file format");
-  private final WriteTextToFileAction writeTextToFileAction
-      = mockery.mock(WriteTextToFileAction.class);
+  private final WriteTextAction writeTextAction = mockery
+      .mock(WriteTextAction.class);
   private final ExportAllTransactionsAsCsvToFileAction
       exportAllTransactionsAsCsvToFileAction
```

```
20          = new ExportAllTransactionsAsCsvToFileAction(
21  -       transactionsFileFormat, writeTextToFileAction,
22  -       new File("/irrelevant/path"));
23  +       transactionsFileFormat, writeTextAction);
24
25      @Test
26      public void happyPath() throws Exception {
27  @@ -40,8 +39,7 @@ public class ExportAllTransactionsAsCsvToFileActionTest {
28                          List.class)));
29              will(returnValue(csvText));
30
31  -           oneOf(writeTextToFileAction).writeTextToFile(
32  -               csvText, path);
33  +           oneOf(writeTextAction).writeText(csvText);
34          }});
35
36      final List<Transaction> transactions = Lists
37  @@ -61,10 +59,10 @@ public class ExportAllTransactionsAsCsvToFileActionTest\
38   {
39          new Expectations() {{
40              ignoring(transactionsFileFormat);
41
42  -           allowing(writeTextToFileAction).writeTextToFile(
43  -               with(any(String.class)), with(
44  -               any(
45  -                   File.class)));
46  +           allowing(writeTextAction).writeText(
47  +               with(
48  +                   any(
49  +                       String.class)));
50              will(throwException(ioFailure));
51          }});
52
```

Snapshot 560: The export action no longer knows anything about a file path

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/Expor\
tAllTransactionsAsCsvToFileAction.java b/AndroidFree/src/test/java/ca/jbrai\
ns/upfp/presenter/test/ExportAllTransactionsAsCsvToFileAction.java
index 4022358..8ac7a32 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/ExportAllTra\
nsactionsAsCsvToFileAction.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/ExportAllTra\
nsactionsAsCsvToFileAction.java
@@ -3,23 +3,20 @@ package ca.jbrains.upfp.presenter.test;
 import ca.jbrains.upfp.model.test.Transaction;
 import ca.jbrains.upfp.view.test.CsvFormat;

-import java.io.*;
+import java.io.IOException;
 import java.util.List;

 public class ExportAllTransactionsAsCsvToFileAction {
   private final CsvFormat<List<Transaction>>
       transactionsFileFormat;
-  private final WriteTextToFileAction writeTextToFileAction;
-  private final File destinationFile;
+  private WriteTextAction writeTextAction;

   public ExportAllTransactionsAsCsvToFileAction(
       CsvFormat<List<Transaction>> transactionsFileFormat,
-      WriteTextToFileAction writeTextToFileAction,
-      File destinationFile
+      WriteTextAction writeTextAction
   ) {
     this.transactionsFileFormat = transactionsFileFormat;
-    this.writeTextToFileAction = writeTextToFileAction;
-    this.destinationFile = destinationFile;
+    this.writeTextAction = writeTextAction;
   }

   public void exportAllTransactionsAsCsvToFileAction(
@@ -27,13 +24,6 @@ public class ExportAllTransactionsAsCsvToFileAction {
   ) throws IOException {
     final String text = transactionsFileFormat.format(
         transactions);
```

```
-    new WriteTextAction() {
-      @Override
-      public void writeText(String text)
-          throws IOException {
-        writeTextToFileAction.writeTextToFile(
-            text, destinationFile);
-      }
-    }.writeText(text);
+    writeTextAction.writeText(text);
   }
 }
```

Now I migrate `WriteTextToFileActionImpl` away from implementing `WriteTextToFileAction` towards implementing `WriteTextAction` and clean things up. Again, I've condensed the microsteps to avoid a 700-page book.

Snapshot 570: Renaming the dreaded Impl class

```
 AndroidFree/src/test/java/ca/jbrains/upfp/view/{WriteTextToFileActionImpl.\
java => WriteTextToFileAction.java} | 28 ++++++++++++++++++-----------
 AndroidFree/src/test/java/ca/jbrains/upfp/view/test/WriteTextToFileTest.ja\
va                                  | 20 +++++++++-----------
 2 files changed, 26 insertions(+), 22 deletions(-)
```

Snapshot 570: The test reads more sensibly now

```
package ca.jbrains.upfp.view.test;

import ca.jbrains.upfp.view.WriteTextToFileAction;
import org.apache.commons.io.FileUtils;
import org.junit.*;

import java.io.*;

import static org.junit.Assert.*;

public class WriteTextToFileTest {
  private static final File testOutputDirectory = new File(
      "./test/output/WriteTextToFileTest/");
```

```
14
15    @BeforeClass
16    public static void initialiseTestOutputAreaOnFileSystem()
17        throws Exception {
18      FileUtils.deleteDirectory(testOutputDirectory);
19      assertTrue(
20          String.format(
21              "Couldn't create test output directory at %1$s",
22              testOutputDirectory.getAbsolutePath()),
23          testOutputDirectory.mkdirs());
24    }
25
26    @Test
27    public void happyPath() throws Exception {
28      final File file = new File(
29          testOutputDirectory, "happyPath.csv");
30
31      new WriteTextToFileAction(file).writeText("::text::");
32
33      assertEquals(
34          "::text::", FileUtils.readFileToString(
35          file));
36    }
37
38    @Test
39    public void ioFailure() throws Exception {
40      final IOException ioFailure = new IOException(
41          "Simulating a failure writing to the file.");
42      final File file = new File(
43          testOutputDirectory, "anyWritableFile.txt");
44      try {
45        new WriteTextToFileAction(file) {
46          @Override
47          protected FileWriter fileWriterOnDestinationFile()
48              throws IOException {
49            return new FileWriter(file) {
50              @Override
51              public void write(String str, int off, int len)
52                  throws IOException {
53                throw ioFailure;
54              }
55            };
```

```
56          }
57        }.writeText("::text::");
58        fail("How did you survive the I/O failure?!");
59      } catch (IOException success) {
60        if (success != ioFailure) throw success;
61      }
62    }
63
64    @Test
65    public void fileAlreadyExists() throws Exception {
66      final File file = new File(
67          testOutputDirectory, "alreadyExists.txt");
68      FileUtils.write(
69          file, "There is already something here.");
70
71      new WriteTextToFileAction(file).writeText("::text::");
72
73      assertEquals(
74          "::text::", FileUtils.readFileToString(
75          file));
76    }
77  }
```

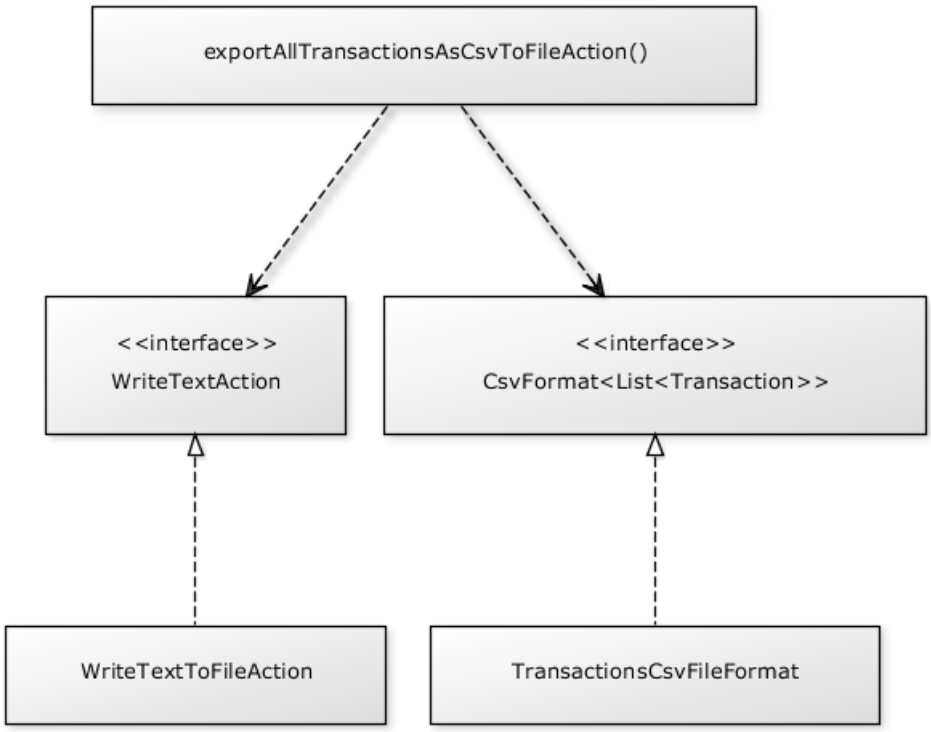
Snapshot 570: The production class still has a lingering smell

```
1   package ca.jbrains.upfp.view;
2
3   import ca.jbrains.upfp.presenter.test.WriteTextAction;
4
5   import java.io.*;
6
7   public class WriteTextToFileAction
8       implements WriteTextAction {
9     private final File destinationFile;
10
11    public WriteTextToFileAction(File destinationFile) {
12      this.destinationFile = destinationFile;
13    }
14
15    protected FileWriter fileWriterOnDestinationFile()
16        throws IOException {
```

```
17      return new FileWriter(destinationFile);
18    }
19
20    @Override
21    public void writeText(String text) throws IOException {
22      final FileWriter fileWriter
23          = fileWriterOnDestinationFile();
24      fileWriter.write(text);
25      fileWriter.flush();
26      fileWriter.close();
27    }
28  }
```

In spite of these improvements, the Action still makes the `FileWriter` available to the test, which smells. I don't fix it now, but eventually I will move the `FileWriter` into the Action's constructor, then add a creation method to the Action that takes a `File` and initialises the Action with a new `FileWriter`. This would give me the best of both worlds. In retrospect, I should have left behind a REFACTOR comment, but I didn't do that at the time, and I don't know why not.

It doesn't look like much, but I've taken the design one step further.



Much better

Now that I've implemented all the pieces, I expect to be able to invoke them from the Activity. I'll save that for the next session.

What's done

- Snapshot 490: We can write text to a file, if everything goes OK.
- Snapshot 500: Writing text to file handles an IOException during the writing.
- Snapshot 510: WriteTextToFileActionImpl overwrites any existing file by default, which works for me.
- Snapshot 520: Introduced interface for writing text that does not refer to a file. (I know, it's not much different than the abstract class Writer.)
- Snapshot 530: Export action now invokes a WriteTextAction that delegates to the existing WriteTextToFileAction, encapsulating the file-ness.
- Snapshot 540: Extracted class for the Export Action.
- Snapshot 550: Moved path parameter up to the constructor.
- Snapshot 560:
  - Moved initialising the WriteTextAction into the constructor.
  - Add WriteTextAction as a parameter to Export…Action.
  - Removed obsolete parameters.

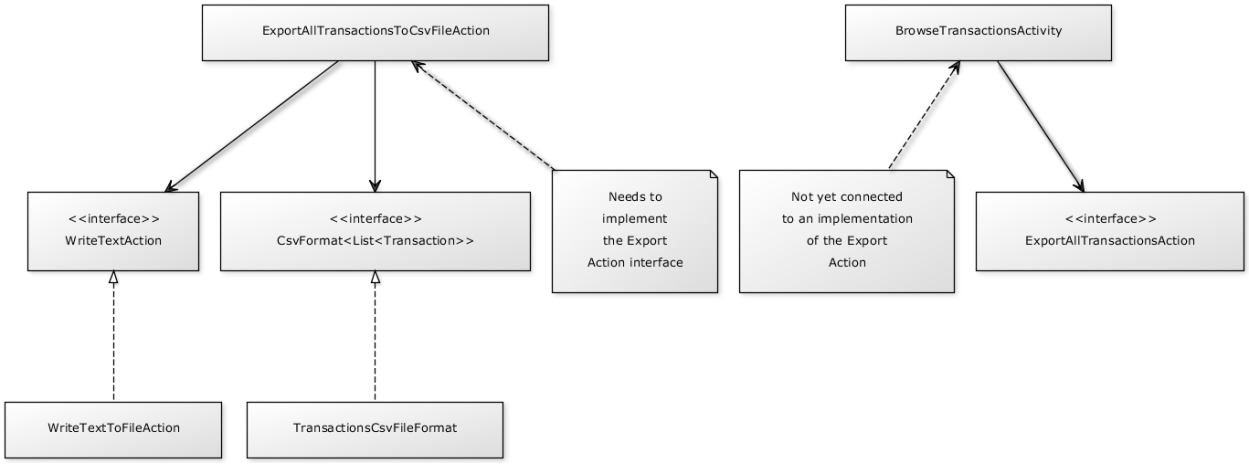
  - Export…ActionTest now expects to invoke a WriteTextAction, and no longer a WriteTextToFileAction.
  - Removed obsolete fixture object.
  - Minor code formatting.
- Snapshot 570:
  - WriteTextToFileActionImpl now implements both Write…Action interfaces, preparing to ditch the old one.
  - Removed obsolete interface.
  - Reverse delegation inside WriteTextToFileActionImpl in preparation for inlining the old writeTextToFile() method.
  - Inlined method to remove duplication in passing a parameter both to the constructor and a method.
  - Improved names.
  - Last, but not least, no more Impl.

What's left to do

- Wire BrowseTransactionsActivity to ExportAllTransactionsAsCsvToFileAction.
- Write contract tests for View once they stabilise
- Enter a transaction quickly and easily
- (2 upvotes) Delegate the Activity's BrowseTransactionsView implementation to a new class
- Find a library that provides an easy way to mark a property as "non-nullable"
  - It needs to let me throw a Programmer Mistake when I want to.
  - Evaluate the @Nullable and @NotNull annotations.
- Find a library that provides Rails-style object validations.
- Rename androidDevicePublicStorageGateway.findPublicExternalStorageDirectory() to something more like findSafePlaceToWriteAFile(), which better captures its intent.

14 Exposing a Chunnel Problem

It feels like ages since I looked at `BrowseTransactionsActivity`, so I'd better review the situation before I start changing anything. How will I connect the Activity to the Export Action and all the rest? I imagine this will force me to move a bunch of code into new packages, and from test source trees into production source trees. I often have to do this kind of wrangling when I work from the pieces towards the whole. I also want to know just how serious a Chunnel Problem I've created by working this way.



Time to connect the Activity to the Export Action

Here, I've drawn attention to `BrowseTransactionsActivity`'s missing link to `ExportAllTransactionsAction`, and ignored its other collaborators. From this diagram and a cursory glance of the code, I have identified these items left to do.

1. `ExportAllTransactionsToCsvFileAction` needs to implement `ExportAllTransactionsAction`.
2. `BrowseTransactionsActivity` needs to instantiate an `ExportAllTransactionsToCsvFileAction` with all the pieces it needs, which I plan to do in the default constructor.
    1. For this to work, the contract between `BrowseTransactionsActivity` and `ExportAllTransactionsActio` must be crystal clear, since this is the key integration point, and I'd rather not rely on integrated tests.

After this, I expect to do some overall cleanup, then move on to the next feature.

First, I have a slight Chunnel Problem between `ExportAllTransactionsAction` and `ExportAllTransactionsToCsvF` – the method signatures don't match – so I fix that. This causes me to move a handful of classes from test packages into production packages, because a production class wants to use them for the first time.

Snapshot 580: Moving production code from test packages to production packages

```
 AndroidFree/src/{test/java/ca/jbrains/upfp/model/test => main/java/ca/jbra\
ins/upfp/model}/Amount.java      |  2 +-
 AndroidFree/src/{test/java/ca/jbrains/upfp/model/test => main/java/ca/jbra\
ins/upfp/model}/Category.java    |  2 +-
 AndroidFree/src/{test/java/ca/jbrains/upfp/model/test => main/java/ca/jbra\
ins/upfp/model}/Transaction.java |  2 +-
 AndroidFree/src/main/java/ca/jbrains/upfp/presenter/ExportAllTransactionsA\
ction.java                       |  6 +++++-
 AndroidFree/src/test/java/ca/jbrains/upfp/model/test/AmountValueObjectBeha\
viorTest.java                    |  1 +
 AndroidFree/src/test/java/ca/jbrains/upfp/model/test/CategoryValueObjectBe\
haviorTest.java                  |  1 +
 AndroidFree/src/test/java/ca/jbrains/upfp/model/test/CreateCategoryTest.ja\
va                               |  1 +
 AndroidFree/src/test/java/ca/jbrains/upfp/model/test/CreateTransactionTest\
.java                            |  3 +++
 AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/ExportAllTransact\
ionsAsCsvToFileAction.java       |  7 +++----
 AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/ExportAllTransact\
ionsAsCsvToFileActionTest.java   | 12 +++++++-------
 AndroidFree/src/test/java/ca/jbrains/upfp/view/test/AmountCsvFormat.java  \
                                 |  2 +-
 AndroidFree/src/test/java/ca/jbrains/upfp/view/test/CategoryCsvFormat.java\
                                 |  2 +-
 AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatAmountAsCsvTest.\
java                             |  2 +-
 AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatCategoryAsCsvTes\
t.java                           |  2 +-
 AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatTransactionAsCsv\
RowTest.java                     |  2 +-
 AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatTransactionsAsCs\
vFileTest.java                   |  2 +-
 AndroidFree/src/test/java/ca/jbrains/upfp/view/test/TransactionCsvFormat.j\
ava                              |  2 +-
 AndroidFree/src/test/java/ca/jbrains/upfp/view/test/TransactionsCsvFileFor\
mat.java                         |  2 +-
 src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java             \
                                 |  7 ++++---
 src/test/java/ca/jbrains/upfp/controller/android/test/HandleExportAllTrans\
```

```
actionsTest.java                  |  9 +++++----
 20 files changed, 39 insertions(+), 30 deletions(-)
```

This large diff indicates working from new code towards the client, rather than from the client towards new code. If I had worked systematically from the client towards each layer below, then I would have moved classes and interfaces from test into production more incrementally. While I generally prefer to work in an orderly fashion from the client towards its collaborators, I've learned–though I don't remember where–that the brain generally doesn't work like that. Often, in order to explore a problem, we jump back and forth between higher-level, more abstract thinking and lower-level, more concrete thinking. Sometimes we find it easier to write code working up the call stack, and sometimes we find it easy enough to write code working down the call stack. We risk the Chunnel Problem to avoid writer's block.

In most of these files, I've only changed `import` statements; the others depend directly on the change in method signature in `ExportAllTransactionsAction`.

Snapshot 580: Changing the Export Action method signature

```
diff --git a/AndroidFree/src/main/java/ca/jbrains/upfp/presenter/ExportAllT\
ransactionsAction.java b/AndroidFree/src/main/java/ca/jbrains/upfp/presente\
r/ExportAllTransactionsAction.java
index ba6368a..8dc7d43 100644
--- a/AndroidFree/src/main/java/ca/jbrains/upfp/presenter/ExportAllTransact\
ionsAction.java
+++ b/AndroidFree/src/main/java/ca/jbrains/upfp/presenter/ExportAllTransact\
ionsAction.java
@@ -1,5 +1,9 @@
 package ca.jbrains.upfp.presenter;

+import ca.jbrains.upfp.model.Transaction;
+
+import java.util.List;
+
 public interface ExportAllTransactionsAction {
-  void execute();
+  void execute(List<Transaction> transactions);
 }
```

Snapshot 580: Changing the Export To File Action method signature

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/Expor\
tAllTransactionsAsCsvToFileActionTest.java b/AndroidFree/src/test/java/ca/j\
brains/upfp/presenter/test/ExportAllTransactionsAsCsvToFileActionTest.java
index 04b7909..751cb90 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/ExportAllTra\
nsactionsAsCsvToFileActionTest.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/ExportAllTra\
nsactionsAsCsvToFileActionTest.java
@@ -1,6 +1,6 @@
 package ca.jbrains.upfp.presenter.test;

-import ca.jbrains.upfp.model.test.Transaction;
+import ca.jbrains.upfp.model.Transaction;
 import ca.jbrains.upfp.view.test.CsvFormat;
 import com.google.common.collect.Lists;
 import org.jmock.*;
@@ -45,9 +45,8 @@ public class ExportAllTransactionsAsCsvToFileActionTest {
     final List<Transaction> transactions = Lists
         .newArrayList();

-    exportAllTransactionsAsCsvToFileAction
-        .exportAllTransactionsAsCsvToFileAction(
-            transactions);
+    exportAllTransactionsAsCsvToFileAction.execute(
+        transactions);
   }

   @Test
@@ -69,9 +68,8 @@ public class ExportAllTransactionsAsCsvToFileActionTest {
     final List<Transaction> irrelevantTransactions = Lists
         .newArrayList();
     try {
-      exportAllTransactionsAsCsvToFileAction
-          .exportAllTransactionsAsCsvToFileAction(
-              irrelevantTransactions);
+      exportAllTransactionsAsCsvToFileAction.execute(
+          irrelevantTransactions);
       fail(
           "Writing text to disk failed, but you don't care?!");
     } catch (IOException success) {
```

```
diff --git a/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/Expor\
tAllTransactionsAsCsvToFileAction.java b/AndroidFree/src/test/java/ca/jbrai\
ns/upfp/presenter/test/ExportAllTransactionsAsCsvToFileAction.java
index 8ac7a32..6fbacae 100644
--- a/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/ExportAllTra\
nsactionsAsCsvToFileAction.java
+++ b/AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/ExportAllTra\
nsactionsAsCsvToFileAction.java
@@ -1,6 +1,6 @@
 package ca.jbrains.upfp.presenter.test;

-import ca.jbrains.upfp.model.test.Transaction;
+import ca.jbrains.upfp.model.Transaction;
 import ca.jbrains.upfp.view.test.CsvFormat;

 import java.io.IOException;
@@ -19,9 +19,8 @@ public class ExportAllTransactionsAsCsvToFileAction {
     this.writeTextAction = writeTextAction;
   }

-  public void exportAllTransactionsAsCsvToFileAction(
-      List<Transaction> transactions
-  ) throws IOException {
+  public void execute(List<Transaction> transactions)
+      throws IOException {
     final String text = transactionsFileFormat.format(
         transactions);
     writeTextAction.writeText(text);
```

Snapshot 580: Changing the clients of the Export Action

```
diff --git a/src/test/java/ca/jbrains/upfp/controller/android/test/HandleEx\
portAllTransactionsTest.java b/src/test/java/ca/jbrains/upfp/controller/and\
roid/test/HandleExportAllTransactionsTest.java
index ef37ef9..bad6e83 100644
--- a/src/test/java/ca/jbrains/upfp/controller/android/test/HandleExportAll\
TransactionsTest.java
+++ b/src/test/java/ca/jbrains/upfp/controller/android/test/HandleExportAll\
```

```
TransactionsTest.java
@@ -12,7 +12,7 @@ import org.junit.*;
 import org.junit.runner.RunWith;

 import java.io.File;
-import java.util.Collection;
+import java.util.*;
 import java.util.regex.Pattern;

 import static ca.jbrains.hamcrest.RegexMatcher.matches;
@@ -64,7 +64,8 @@ public class HandleExportAllTransactionsTest {
             ignoring(androidDevicePublicStorageGateway)
                 .findPublicExternalStorageDirectory();

-            allowing(exportAllTransactionsAction).execute();
+            allowing(exportAllTransactionsAction).execute(
+                with(any(List.class)));
             // succeeds by not throwing an exception
           }});

@@ -155,8 +156,8 @@ public class HandleExportAllTransactionsTest {
             ignoring(browseTransactionsModel);
             ignoring(androidDevicePublicStorageGateway);

-            allowing(exportAllTransactionsAction)
-                .execute();
+            allowing(exportAllTransactionsAction).execute(
+                with(any(List.class)));
             will(
                 throwException(
                     new InternalStorageException()));
```

```
diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
index 165a203..45432c5 100644
--- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
+++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
@@ -13,7 +13,7 @@ import ca.jbrains.upfp.view.BrowseTransactionsView;
 import com.google.common.collect.Lists;

 import java.io.File;
```

```
-import java.util.Collection;
+import java.util.*;

 public class BrowseTransactionsActivity extends Activity
     implements BrowseTransactionsView {
@@ -47,7 +47,7 @@ public class BrowseTransactionsActivity extends Activity
     this.exportAllTransactionsAction
         = new ExportAllTransactionsAction() {
       @Override
-      public void execute() {
+      public void execute(List<Transaction> transactions) {
         // Do nothing, for now
       }
     };
@@ -143,7 +143,8 @@ public class BrowseTransactionsActivity extends Activit\
y
       browseTransactionsModel.findAllTransactions();
       androidDevicePublicStorageGateway
           .findPublicExternalStorageDirectory();
-      exportAllTransactionsAction.execute();
+      exportAllTransactionsAction.execute(
+          Lists.<Transaction>newArrayList());
       notifyUser(
           "Exported all transactions to /mnt/sdcard/TrackEveryPenny.csv");
     } catch (InternalStorageException reported) {
```

This finally exposes the missing connexion in `BrowseTransactionsActivity.exportAllTransactions()` that had made me uncomfortable when I first implemented it. I'd invoked three methods and completely ignored their return values. Now I need to connect two of them: the export action needs a list of `Transaction` objects, and the model returns that list two lines earlier. I would like a test to force me to make this connexion.

Snapshot 590: We should export the proper Transactions

```
diff --git a/src/test/java/ca/jbrains/upfp/controller/android/test/HandleEx\
portAllTransactionsTest.java b/src/test/java/ca/jbrains/upfp/controller/and\
roid/test/HandleExportAllTransactionsTest.java
index bad6e83..47fdbde 100644
--- a/src/test/java/ca/jbrains/upfp/controller/android/test/HandleExportAll\
TransactionsTest.java
+++ b/src/test/java/ca/jbrains/upfp/controller/android/test/HandleExportAll\
TransactionsTest.java
```

```
@@ -4,6 +4,7 @@ import ca.jbrains.upfp.*;
 import ca.jbrains.upfp.controller.android.*;
 import ca.jbrains.upfp.model.*;
 import ca.jbrains.upfp.presenter.ExportAllTransactionsAction;
+import ca.jbrains.upfp.test.ObjectMother;
 import com.google.common.collect.Lists;
 import com.xtremelabs.robolectric.RobolectricTestRunner;
 import com.xtremelabs.robolectric.shadows.*;
@@ -171,6 +172,29 @@ public class HandleExportAllTransactionsTest {
         "application. I feel embarrassed and ashamed.");
   }

+  @Test
+  public void exportWhateverTheModelProvides()
+      throws Exception {
+    final List<Transaction> anyNonEmptyListOfTransactions
+        = ObjectMother.anyNonEmptyListOfTransactions();
+
+    mockery.checking(
+        new Expectations() {{
+          allowing(browseTransactionsModel)
+              .findAllTransactions();
+          will(returnValue(anyNonEmptyListOfTransactions));
+
+          ignoring(androidDevicePublicStorageGateway);
+
+          oneOf(exportAllTransactionsAction).execute(
+              with(
+                  anyNonEmptyListOfTransactions));
+        }});
+
+    pressExportAllTransactionsButton(
+        browseTransactionsActivity);
+  }
+

   private void pressExportAllTransactionsButton(
       BrowseTransactionsActivity browseTransactionsActivity
```

Snapshot 590: We now export the proper Transactions

```
diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
index 45432c5..a85166b 100644
--- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
+++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
@@ -13,7 +13,7 @@ import ca.jbrains.upfp.view.BrowseTransactionsView;
 import com.google.common.collect.Lists;

 import java.io.File;
-import java.util.*;
+import java.util.List;

 public class BrowseTransactionsActivity extends Activity
     implements BrowseTransactionsView {
@@ -39,8 +39,8 @@ public class BrowseTransactionsActivity extends Activity
           }

           @Override
-          public Collection<Object> findAllTransactions() {
-            return Lists.newArrayList();
+          public List<Transaction> findAllTransactions() {
+            return Lists.<Transaction>newArrayList();
           }
         }, this);

@@ -63,8 +63,9 @@ public class BrowseTransactionsActivity extends Activity
       }

       @Override
-      public Collection<Object> findAllTransactions() {
-        return null;  //To change body of implemented methods use File | S\
ettings | File Templates.
+      public List<Transaction> findAllTransactions() {
+        return null;  //To change body of implemented
+        // methods use File | Settings | File Templates.
       }
     };

@@ -140,11 +141,11 @@ public class BrowseTransactionsActivity extends Activ\
ity
```

```
41     public void exportAllTransactions(View clicked) {
42       try {
43         // REFACTOR Shouldn't a presenter be doing this?
44 -       browseTransactionsModel.findAllTransactions();
45 +       final List<Transaction> transactions
46 +           = browseTransactionsModel.findAllTransactions();
47         androidDevicePublicStorageGateway
48             .findPublicExternalStorageDirectory();
49 -       exportAllTransactionsAction.execute(
50 -           Lists.<Transaction>newArrayList());
51 +       exportAllTransactionsAction.execute(transactions);
52         notifyUser(
53             "Exported all transactions to /mnt/sdcard/TrackEveryPenny.csv");
54       } catch (InternalStorageException reported) {
```

This change forces me to make the contract of `BrowseTransactionsModel` more precise.

Snapshot 590: The Model now specifies the type of 'all transactions'

```
1  diff --git a/AndroidFree/src/main/java/ca/jbrains/upfp/model/BrowseTransact\
2  ionsModel.java b/AndroidFree/src/main/java/ca/jbrains/upfp/model/BrowseTran\
3  sactionsModel.java
4  index c1da7bf..e81c2ba 100644
5  --- a/AndroidFree/src/main/java/ca/jbrains/upfp/model/BrowseTransactionsMod\
6  el.java
7  +++ b/AndroidFree/src/main/java/ca/jbrains/upfp/model/BrowseTransactionsMod\
8  el.java
9  @@ -1,10 +1,10 @@
10  package ca.jbrains.upfp.model;
11
12 -import java.util.Collection;
13 +import java.util.List;
14
15  public interface BrowseTransactionsModel {
16    // CONTRACT: result >= 0
17    int countTransactions();
18
19 -  Collection<Object> findAllTransactions();
20 +  List<Transaction> findAllTransactions();
21  }
```

This leads to a number of small incidental changes, which you can browse in the code repository at your leisure.

In retrospect, I could have used the type `Collection<Transaction>`, but I have grown so accustomed to using `List` as a default collection type that I forgot to apply the principle of Use The Weakest Type Possible. In my defence, IDEA omits one key refactoring that Eclipse provides, called "Generalize Declared Type", which looks at the code to determine the weakest type (farthest up the inheritance hierarchy) that one could declare for a variable. I rely on that so much when I use Eclipse that, when using IDEA, I will probably declare some variables with an unnecessarily precise type.

Next, I need `BrowseTransactionsActivity` to export those `Transaction` objects to the correct `File` using the correct implementation of `ExportAllTransactionsAction`. This part will require more precise surgery, so I will save it for the next session.

What's Done

- Snapshot 580:
  - Renamed method to match interface.
  - Changed the interface to match the implementation's method signature, which necessitated moving a handful of classes to new packages and source trees.
- Snapshot 590: `BrowseTransactionsActivity` now exports the correct list of `Transaction` objects.

What's Left to Do

- Wire `BrowseTransactionsActivity` to `ExportAllTransactionsAsCsvToFileAction`.
  - `BrowseTransactionsActivity` needs to choose the right `WriteTextAction` with the right `File`.
- Write contract tests for View once they stabilise
- Enter a transaction quickly and easily
- (2 upvotes) Delegate the Activity's `BrowseTransactionsView` implementation to a new class
- Find a library that provides an easy way to mark a property as "non-nullable"
  - It needs to let me throw a Programmer Mistake when I want to.
  - Evaluate the `@Nullable` and `@NotNull` annotations.
- Find a library that provides Rails-style object validations.
- Rename `androidDevicePublicStorageGateway.findPublicExternalStorageDirectory()` to something more like `findSafePlaceToWriteAFile()`, which better captures its intent.

15 Putting It All Together?

I *really* want to connect the `BrowseTransactionsActivity` to the CSV File-based implementation of `ExportAllTransactionsAction`, but looking at the Activity, I notice duplicate model instances in its no-argument constructor, so I have inconsistent data, albeit only entirely in memory. To avoid heartache and head-scratching, I need to fix this.

Fixing a Harmful Dependency

First, since I haven't looked at this code in a relatively long time, I clean it up a little. In the process, I run into the obstacle that prevents me from chaining constructors, which I really want to do. I will need to extract the `BrowseTransactionsView` behavior from the Activity before I go much further.

Snapshot 600: Paving the way to chain constructors

```
diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
index 2b07e43..6753f0b 100644
--- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
+++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
@@ -32,20 +32,6 @@ public class BrowseTransactionsActivity extends Activity
     // We have to wait for super() to be (implicitly)
     // invoked.

-    // REFACTOR Delegate BrowseTransactionsView behavior
-    // to a new class
-    this.rendersView = new BrowseTransactionsPresenter(
-        new BrowseTransactionsModel() {
-          @Override
-          public int countTransactions() {
-            return 12;
-          }
-
-          @Override
-          public List<Transaction> findAllTransactions() {
-            return Lists.<Transaction>newArrayList();
-          }
```

```
-        }, this);
 
     this.exportAllTransactionsAction
         = new ExportAllTransactionsAction() {
@@ -63,16 +49,20 @@ public class BrowseTransactionsActivity extends Activit\
y
         = new BrowseTransactionsModel() {
       @Override
       public int countTransactions() {
-        return 0;
+        return findAllTransactions().size();
       }
 
       @Override
       public List<Transaction> findAllTransactions() {
-        return null;  //To change body of implemented
-        // methods use File | Settings | File Templates.
+        return Lists.newArrayList();
       }
     };
 
+    // REFACTOR Delegate BrowseTransactionsView behavior
+    // to a new class
+    this.rendersView = new BrowseTransactionsPresenter(
+        this.browseTransactionsModel, this);
+
     this.androidDevicePublicStorageGateway
         = new AndroidDevicePublicStorageGateway() {
       @Override
@@ -83,15 +73,6 @@ public class BrowseTransactionsActivity extends Activity
     };
   }
 
-  /**
-   * @deprecated
-   */
-  public BrowseTransactionsActivity(
-      RendersView rendersView
-  ) {
-    this(rendersView, null, null, null);
-  }
-
```

```
65     public BrowseTransactionsActivity(
66         RendersView rendersView,
67         ExportAllTransactionsAction
68 @@ -126,6 +107,7 @@ public class BrowseTransactionsActivity extends Activit\
69 y
70       setContentView(R.layout.main);
71     }
72
73 +   // REFACTOR Marked to move to an extracted class
74     public void displayNumberOfTransactions(
75         int transactionCount
76     ) {
```

Next, I intend to move the `BrowseTransactionsView` behavior into a smaller class, but shortly after starting, I realise that I have made a fundamentally incorrect assumption in designing the Activity. I expected the Activity to instantiate a Presenter, Model, and View, then wire them together, and all inside the Activity's constructor – where else, really? As long as the Activity implements the View, this works, but as soon as I want to separate these two behaviors, I uncover a coupling problem: the Android implementation of the View needs to talk to the `TextView` rendering "You have *n* transactions", and the Android runtime doesn't create that widget until it finishes invoking the Activity's `onCreate()`.

> This all sounds so familiar... I experience a jarring, painful flashback to 1999 and Enterprise Java Beans. In that framework, or at least before version 3 of it, since I've lost touch with it over the years, I would implement `SessionBean` with a class like `MySessionBean`, and the EJB container would invoke lifecycle methods like `onCreate()`, `onDestroy()`, `onActivate()` and `onPassivate()`. I came to think of `onCreate()` like a constructor and `onDestroy()` like a destructor, so rather than write my code directly into the `SessionBean`, I'd delegate it to a class that didn't depend on the EJB libraries, and `MySessionBean` would instantiate that class in `onCreate()` and release it in `onDestroy()`. It sounds like I will need to do the same thing here.

Unfortunately, this requires a substantial change to the existing code. Fortunately, it will result in less code that has to run in the Android simulator. I accept that exchange, and since I have version control, I can experiment for a while, and if I get it wrong, I can throw the experiment away and try something else. I have nothing to lose but a little time.

Snapshot 610: Moving View behavior from the Activity into a View class

```
diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
index 6753f0b..ffde6cf 100644
--- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
+++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
@@ -4,35 +4,69 @@ import android.app.Activity;
 import android.os.Bundle;
 import android.util.Log;
 import android.view.View;
-import android.widget.*;
-import ca.jbrains.toolkit.ProgrammerMistake;
+import android.widget.Toast;
 import ca.jbrains.upfp.controller.android.*;
 import ca.jbrains.upfp.model.*;
 import ca.jbrains.upfp.presenter.*;
 import ca.jbrains.upfp.view.BrowseTransactionsView;
+import ca.jbrains.upfp.view.android
+    .AndroidBrowseTransactionsView;
 import com.google.common.collect.Lists;

 import java.io.File;
 import java.util.List;

-public class BrowseTransactionsActivity extends Activity
-    implements BrowseTransactionsView {
-  private final RendersView rendersView;
-  private final ExportAllTransactionsAction
+public class BrowseTransactionsActivity extends Activity {
+  private RendersView rendersView;
+  private ExportAllTransactionsAction
       exportAllTransactionsAction;
-  private final AndroidDevicePublicStorageGateway
+  private AndroidDevicePublicStorageGateway
       androidDevicePublicStorageGateway;
-  private final BrowseTransactionsModel
-      browseTransactionsModel;
+  private BrowseTransactionsModel browseTransactionsModel;
+  private BrowseTransactionsView browseTransactionsView;

   public BrowseTransactionsActivity() {
```

```
41  -    // We can't chain the constructor,
42  -    // because the instance in the process of being
43  -    // created is itself the view.
44  -    // We have to wait for super() to be (implicitly)
45  -    // invoked.
46  +    // Do all this work in onCreate()
47  +  }
48  +
49  +  // REFACTOR Move this constructor into the "business
50  +  // delegate"
51  +  public BrowseTransactionsActivity(
52  +      RendersView rendersView,
53  +      ExportAllTransactionsAction
54  +          exportAllTransactionsAction,
55  +      AndroidDevicePublicStorageGateway
56  +          androidDevicePublicStorageGateway,
57  +      BrowseTransactionsModel browseTransactionsModel
58  +  ) {
59  +
60  +    this.rendersView = rendersView;
61  +    this.exportAllTransactionsAction
62  +        = exportAllTransactionsAction;
63  +    this.androidDevicePublicStorageGateway
64  +        = androidDevicePublicStorageGateway;
65  +    this.browseTransactionsModel = browseTransactionsModel;
66  +  }
67  +
68  +  @Override
69  +  protected void onResume() {
70  +    super.onResume();
71  +    // Arbitrarily, I assume that I should do my work
72  +    // after the superclass, but I don't really know.
73  +    // REFACTOR Delegate to businessDelegate.renderView()
74  +    rendersView.render();
75  +  }
76
77  +  /**
78  +   * Called when the activity is first created.
79  +   */
80  +  @Override
81  +  public void onCreate(Bundle savedInstanceState) {
82  +    super.onCreate(savedInstanceState);
```

```
+    setContentView(R.layout.main);

+    // This seems like a more logical place to initialise
+    // the View, anyway.
     this.exportAllTransactionsAction
         = new ExportAllTransactionsAction() {
       @Override
@@ -58,10 +92,14 @@ public class BrowseTransactionsActivity extends Activit\
y
       }
     };

+    this.browseTransactionsView
+        = new AndroidBrowseTransactionsView(this);
+
     // REFACTOR Delegate BrowseTransactionsView behavior
     // to a new class
     this.rendersView = new BrowseTransactionsPresenter(
-        this.browseTransactionsModel, this);
+        this.browseTransactionsModel,
+        browseTransactionsView);

     this.androidDevicePublicStorageGateway
         = new AndroidDevicePublicStorageGateway() {
@@ -71,63 +109,20 @@ public class BrowseTransactionsActivity extends Activi\
ty
         return new File(".");
       }
     };
-  }
-
-  public BrowseTransactionsActivity(
-      RendersView rendersView,
-      ExportAllTransactionsAction
-          exportAllTransactionsAction,
-      AndroidDevicePublicStorageGateway
-          androidDevicePublicStorageGateway,
-      BrowseTransactionsModel browseTransactionsModel
-  ) {
-
-    this.rendersView = rendersView;
-    this.exportAllTransactionsAction
```

```
125 -        = exportAllTransactionsAction;
126 -    this.androidDevicePublicStorageGateway
127 -        = androidDevicePublicStorageGateway;
128 -    this.browseTransactionsModel = browseTransactionsModel;
129 -  }
130 -
131 -  @Override
132 -  protected void onResume() {
133 -    super.onResume();
134 -    // Arbitrarily, I assume that I should do my work
135 -    // after the superclass, but I don't really know.
136 -    rendersView.render();
137 -  }
138
139 -  /**
140 -   * Called when the activity is first created.
141 -   */
142 -  @Override
143 -  public void onCreate(Bundle savedInstanceState) {
144 -    super.onCreate(savedInstanceState);
145 -    setContentView(R.layout.main);
146    }
147
148 -  // REFACTOR Marked to move to an extracted class
149 +  // REFACTOR Move to businessDelegate?
150    public void displayNumberOfTransactions(
151        int transactionCount
152    ) {
153 -    if (transactionCount < 0) throw new ProgrammerMistake(
154 -        new IllegalArgumentException(
155 -            String.format(
156 -                "number of transactions can't be " +
157 -                "negative, but it's %1$d",
158 -                transactionCount)));
159 -
160 -    final TextView transactionsCountView
161 -        = (TextView) findViewById(R.id.transactionsCount);
162 -    transactionsCountView.setText(
163 -        String.format(
164 -            "%1$d", transactionCount));
165 +    browseTransactionsView.displayNumberOfTransactions(
166 +        transactionCount);
```

```
167     }
168
169     public void exportAllTransactions(View clicked) {
170 +     // I'm not entirely sure where this will end up
171       try {
172 -       // REFACTOR Shouldn't a presenter be doing this?
173         final List<Transaction> transactions
174             = browseTransactionsModel.findAllTransactions();
175         androidDevicePublicStorageGateway
176 @@ -188,4 +183,21 @@ public class BrowseTransactionsActivity extends Activi\
177 ty
178     ) {
179       Log.e("TrackEveryPenny", message, reported);
180     }
181 +
182 +   // SMELL Programming by accident
183 +   // This should disappear after moving other behavior
184 +   // into the businessDelegate.
185 +   public void setCollaborators(
186 +       ExportAllTransactionsAction
187 +           exportAllTransactionsAction,
188 +       AndroidDevicePublicStorageGateway
189 +           androidDevicePublicStorageGateway,
190 +       BrowseTransactionsModel browseTransactionsModel
191 +   ) {
192 +     this.exportAllTransactionsAction
193 +         = exportAllTransactionsAction;
194 +     this.androidDevicePublicStorageGateway
195 +         = androidDevicePublicStorageGateway;
196 +     this.browseTransactionsModel = browseTransactionsModel;
197 +   }
198   }
```

Snapshot 610: The new View class

```
package ca.jbrains.upfp.view.android;

import android.widget.TextView;
import ca.jbrains.toolkit.ProgrammerMistake;
import ca.jbrains.upfp.*;
import ca.jbrains.upfp.view.BrowseTransactionsView;

public class AndroidBrowseTransactionsView
    implements BrowseTransactionsView {
  private BrowseTransactionsActivity
      browseTransactionsActivity;

  public AndroidBrowseTransactionsView(
      BrowseTransactionsActivity browseTransactionsActivity
  ) {
    this.browseTransactionsActivity
        = browseTransactionsActivity;
  }

  @Override
  public void displayNumberOfTransactions(
      int numberOfTransactions
  ) {
    if (numberOfTransactions < 0)
      throw new ProgrammerMistake(
          new IllegalArgumentException(
              String.format(
                  "number of transactions can't be " +
                  "negative, but it's %1$d",
                  numberOfTransactions)));

    final TextView transactionsCountView
        = (TextView) browseTransactionsActivity
        .findViewById(
            R.id.transactionsCount);
    transactionsCountView.setText(
        String.format(
            "%1$d", numberOfTransactions));
  }
}
```

Snapshot 610: Checking View behavior more directly

```
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 5a364b4..cb5b3aa 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
@@ -3,6 +3,8 @@ package ca.jbrains.upfp.view.android.test;
 import android.widget.TextView;
 import ca.jbrains.upfp.*;
 import ca.jbrains.upfp.view.BrowseTransactionsView;
+import ca.jbrains.upfp.view.android
+    .AndroidBrowseTransactionsView;
 import ca.jbrains.upfp.view.test
     .BrowseTransactionsViewContract;
 import com.xtremelabs.robolectric.RobolectricTestRunner;
@@ -19,10 +21,14 @@ public class DisplayNumberOfTransactionsTest
     final BrowseTransactionsActivity
         browseTransactionsActivity
         = new BrowseTransactionsActivity();
+    final AndroidBrowseTransactionsView
+        androidBrowseTransactionsView
+        = new AndroidBrowseTransactionsView(
+        browseTransactionsActivity);
     browseTransactionsActivity.onCreate(null);

-    browseTransactionsActivity.displayNumberOfTransactions(
-        12);
+    androidBrowseTransactionsView
+        .displayNumberOfTransactions(12);

     final TextView transactionsCountView
         = (TextView) browseTransactionsActivity
@@ -31,12 +37,17 @@ public class DisplayNumberOfTransactionsTest
         "12", transactionsCountView.getText().toString());
   }
```

```
38
39 +
40     @Override
41     protected BrowseTransactionsView initializeView() {
42       final BrowseTransactionsActivity
43           browseTransactionsActivity
44           = new BrowseTransactionsActivity();
45       browseTransactionsActivity.onCreate(null);
46 -     return browseTransactionsActivity;
47 +     final AndroidBrowseTransactionsView
48 +         androidBrowseTransactionsView
49 +         = new AndroidBrowseTransactionsView(
50 +         browseTransactionsActivity);
51 +     return androidBrowseTransactionsView;
52     }
53  }
```

I have made two major changes:

- `BrowseTransactionsActivity` now initialises everything in `onCreate()`.
- The new `AndroidBrowseTransactionsView` now handles all the `BrowseTransactionsView` behavior.

A Moment of Doubt

Strangely, the result feels like a step backwards. Moving initialisation code from the constructor into `onCreate()` seems fine, but introducing `AndroidBrowseTransactionsView` has complicated the design, adding indirection that so far doesn't feel helpful. It will probably feel this way until I eliminate all references from the View back to the Activity. So far, the View depends on the Activity only to instantiate UI widgets, so I suppose I could add a constructor the View to accept those widgets, and a creation method to the Activity to wire together the widgets that the Android runtime creates from the layout files. This sounds like putting the dependency in the proper direction.

Unfortunately, I've relied so far on the Android runtime to instantiate the View widgets, so until I explore that more, I'll have some annoyingly indirect code in the View tests: I'd have to create the Activity just to create the widgets for the View. This depends unnecessarily on the context, and motivates a very common design problem: giving too much of oneself to one's collaborator. You can categorise this design problem a few ways:

- Violates the Dependency Inversion principle by making the View depend on the (concrete implementation) Activity, rather than an interface for providing `TextViews`.

- Creates a cyclic dependency between Activity and View.
- Exacerbates a violation of the Single Responsibility Principle by making more code depend on an Activity class with too many responsibilities.
- Forces unnecessary implementation details into tests, because the View tests need to know to use the Activity to instantiate the `TextViews` for the View.

If you think about it for a few minutes, you can problem think of a few more specific violations of good design principles. Clearly, I need to change this. I start by reducing the "integration surface" between the Activity and the View, applying a variation of the Interface Segregation Principle – instead of the Activity giving itself entirely to the View, it provides to the View only the piece of itself that the View expressly needs.

Snapshot 620: Reducing the amount of the Activity upon which the View depends

```
diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
index ffde6cf..058c83b 100644
--- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
+++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
@@ -4,7 +4,7 @@ import android.app.Activity;
 import android.os.Bundle;
 import android.util.Log;
 import android.view.View;
-import android.widget.Toast;
+import android.widget.*;
 import ca.jbrains.upfp.controller.android.*;
 import ca.jbrains.upfp.model.*;
 import ca.jbrains.upfp.presenter.*;
@@ -93,7 +93,8 @@ public class BrowseTransactionsActivity extends Activity \
{
     };

     this.browseTransactionsView
-        = new AndroidBrowseTransactionsView(this);
+        = new AndroidBrowseTransactionsView(
+        transactionsCountView());

     // REFACTOR Delegate BrowseTransactionsView behavior
     // to a new class
@@ -112,6 +113,10 @@ public class BrowseTransactionsActivity extends Activi\
ty {

```

```
29      }
30
31  +   private TextView transactionsCountView() {
32  +     return (TextView) findViewById(R.id.transactionsCount);
33  +   }
34  +
35      // REFACTOR Move to businessDelegate?
36      public void displayNumberOfTransactions(
37          int transactionCount
```

Snapshot 620: The View depends on a UI widget, not the entire Activity

```
1   diff --git a/src/main/java/ca/jbrains/upfp/view/android/AndroidBrowseTransa\
2   ctionsView.java b/src/main/java/ca/jbrains/upfp/view/android/AndroidBrowseT\
3   ransactionsView.java
4   index aea9fb5..9b2b18b 100644
5   --- a/src/main/java/ca/jbrains/upfp/view/android/AndroidBrowseTransactionsV\
6   iew.java
7   +++ b/src/main/java/ca/jbrains/upfp/view/android/AndroidBrowseTransactionsV\
8   iew.java
9   @@ -2,19 +2,16 @@ package ca.jbrains.upfp.view.android;
10
11   import android.widget.TextView;
12   import ca.jbrains.toolkit.ProgrammerMistake;
13  -import ca.jbrains.upfp.*;
14   import ca.jbrains.upfp.view.BrowseTransactionsView;
15
16   public class AndroidBrowseTransactionsView
17       implements BrowseTransactionsView {
18  -  private BrowseTransactionsActivity
19  -      browseTransactionsActivity;
20  +  private final TextView transactionsCountView;
21
22      public AndroidBrowseTransactionsView(
23  -       BrowseTransactionsActivity browseTransactionsActivity
24  +       TextView transactionsCountView
25      ) {
26  -     this.browseTransactionsActivity
27  -         = browseTransactionsActivity;
28  +     this.transactionsCountView = transactionsCountView;
29      }
```

```
30
31     @Override
32  @@ -29,10 +26,6 @@ public class AndroidBrowseTransactionsView
33                     "negative, but it's %1$d",
34                     numberOfTransactions)));
35
36  -      final TextView transactionsCountView
37  -          = (TextView) browseTransactionsActivity
38  -          .findViewById(
39  -              R.id.transactionsCount);
40         transactionsCountView.setText(
41             String.format(
42                 "%1$d", numberOfTransactions));
```

Snapshot 620: The test still uses the Activity to instantiate the right widget for the View

```
1   diff --git a/src/test/java/ca/jbrains/upfp/view/android/test/DisplayNumberO\
2   fTransactionsTest.java b/src/test/java/ca/jbrains/upfp/view/android/test/Di\
3   splayNumberOfTransactionsTest.java
4   index cb5b3aa..290e143 100644
5   --- a/src/test/java/ca/jbrains/upfp/view/android/test/DisplayNumberOfTransa\
6   ctionsTest.java
7   +++ b/src/test/java/ca/jbrains/upfp/view/android/test/DisplayNumberOfTransa\
8   ctionsTest.java
9   @@ -18,36 +18,43 @@ public class DisplayNumberOfTransactionsTest
10        extends BrowseTransactionsViewContract {
11      @Test
12      public void happyPath() throws Exception {
13  +     // The duplication between this test setup and
14  +     // initializeView() is incidental; don't remove it
15        final BrowseTransactionsActivity
16            browseTransactionsActivity
17            = new BrowseTransactionsActivity();
18  +     browseTransactionsActivity.onCreate(null);
19  +     final TextView transactionsCountView
20  +         = (TextView) browseTransactionsActivity
21  +         .findViewById(R.id.transactionsCount);
22        final AndroidBrowseTransactionsView
23            androidBrowseTransactionsView
24            = new AndroidBrowseTransactionsView(
25  -         browseTransactionsActivity);
```

```
-    browseTransactionsActivity.onCreate(null);
+        transactionsCountView);

     androidBrowseTransactionsView
         .displayNumberOfTransactions(12);

-    final TextView transactionsCountView
-        = (TextView) browseTransactionsActivity
-        .findViewById(R.id.transactionsCount);
     assertEquals(
         "12", transactionsCountView.getText().toString());
   }

-
+  // The duplication between initializeView() and the
+  // setup for the other tests is incidental; don't
+  // remove it
   @Override
   protected BrowseTransactionsView initializeView() {
     final BrowseTransactionsActivity
         browseTransactionsActivity
         = new BrowseTransactionsActivity();
     browseTransactionsActivity.onCreate(null);
+    final TextView transactionsCountView
+        = (TextView) browseTransactionsActivity
+        .findViewById(R.id.transactionsCount);
     final AndroidBrowseTransactionsView
         androidBrowseTransactionsView
         = new AndroidBrowseTransactionsView(
-        browseTransactionsActivity);
+        transactionsCountView);
     return androidBrowseTransactionsView;
   }
 }
```

I'd like to explain the "incidental duplication" comment I made in `DisplayNumberOfTransactionsTest`. Both the contract for `BrowseTransactionsView` and the implementation details tests for displaying the number of transactions both initialise the layout in the Activity in order to have access to the UI widgets they use. They happen to do this the same way, but while `DisplayNumberOfTransactionsTest` needs references to both the `AndroidBrowseTransactionsView` and the `TextView` that renders the transaction count, the contract tests only need the `AndroidBrowseTransactionsView` instance. I tried to make `initializeView()` work for both cases, but every path led to convoluting the design of the

contract, the implementation details tests, or both, so I added those comments instead.

Then something wonderful happens.

While writing these words, I realise that `initializeView()` might not need to initialise the transactions count `TextView` at all, because nothing in the contract touches the UI widgets in the view. I test my theory by passing a null `TextView` into the `AndroidBrowseTransactionsView`, whereupon the test passes. I delete the now-obsolete code, which makes my comments misleading, so I delete the comments. That makes me happy.

Snapshot 630: Drastically simplifying a 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 290e143..1a73c01 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
@@ -18,8 +18,6 @@ public class DisplayNumberOfTransactionsTest
     extends BrowseTransactionsViewContract {
   @Test
   public void happyPath() throws Exception {
-    // The duplication between this test setup and
-    // initializeView() is incidental; don't remove it
     final BrowseTransactionsActivity
         browseTransactionsActivity
         = new BrowseTransactionsActivity();
@@ -39,22 +37,8 @@ public class DisplayNumberOfTransactionsTest
         "12", transactionsCountView.getText().toString());
   }

-  // The duplication between initializeView() and the
-  // setup for the other tests is incidental; don't
-  // remove it
   @Override
   protected BrowseTransactionsView initializeView() {
-    final BrowseTransactionsActivity
-        browseTransactionsActivity
-        = new BrowseTransactionsActivity();
-    browseTransactionsActivity.onCreate(null);
-    final TextView transactionsCountView
```

```
-        = (TextView) browseTransactionsActivity
-        .findViewById(R.id.transactionsCount);
-    final AndroidBrowseTransactionsView
-        androidBrowseTransactionsView
-        = new AndroidBrowseTransactionsView(
-        transactionsCountView);
-    return androidBrowseTransactionsView;
+    return new AndroidBrowseTransactionsView(null);
   }
 }
```

As I edit these words, I realise that I missed an opportunity to take this one important step further: in `DisplayNumberOfTransactionsTest` I could simply instantiate a `TextView` myself and give it directly to the View without involving the Activity at all. When I look at the resulting code, it becomes obvious that I should have done it this way all along, but I find it comforting to know that even when my intuition has let me down by not leading me here more directly, the mechanics of refactoring and applying my trusted design principles has led me here. Either way, I've made it. I add a note to the backlog to clean this up.

I'd like to end this session, but first I need to unwind my mental stack a little. I've separated the `BrowseTransactionsView` from the `BrowseTransactionsActivity` as much as I can for now, and made another Activity method obsolete, so I'd better remove that now.

Snapshot 640: Removing obsolete code and an unnecessary task

```
diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
index 058c83b..a5d2fa2 100644
--- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
+++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
@@ -117,14 +117,6 @@ public class BrowseTransactionsActivity extends Activi\
ty {
     return (TextView) findViewById(R.id.transactionsCount);
   }

-  // REFACTOR Move to businessDelegate?
-  public void displayNumberOfTransactions(
-      int transactionCount
```

```
14 -  ) {
15 -    browseTransactionsView.displayNumberOfTransactions(
16 -        transactionCount);
17 -  }
18 -
19    public void exportAllTransactions(View clicked) {
20      // I'm not entirely sure where this will end up
21      try {
```

I still need to deal with the `BrowseTransactionsActivity` constructor that I use in the Activity's tests. I imagine that in light of the simplifications I've just made, I can greatly simplify those tests, which in turn improves the separation between the Activity and the "business delegate" that I've started introducing at the beginning of this session. In the next session, I'll review how this relates to wiring the Activity to the `ExportAllTransactionsAction`, and finish that job for good. If the resulting design matches what I have in my head, then it will all have been worth the wait, and adding the next feature will feel easy by comparison.

What's done

- Snapshot 600:
  - Minor code formatting changes.
  - Removed obsolete constructor.
  - Removed duplicate model instances in constructor.
  - Marked the methods I expected to extract into a smaller class that implements BrowseTransactionsView.
- Snapshot 610:
  - Formatting changes.
  - Marked the changes that I think I'll need to make.
  - Moved default constructor behavior into onCreate() and patched things up to match. It gets a little uglier.
  - Moved BrowseTransactionsView behavior into a little view class, preparing to remove it entirely from the Activity.
  - Copied the lone test for the Android implementation of BrowseTransactionsView, so that I could test the new view implementation directly. I'm not sure that it's better.
  - Removed a test for the old implementation of BrowseTransactionsView.
  - AndroidBrowseTransactionsView now respects the BrowseTransactionsView contract.
  - BrowseTransactionsActivity no longer implements BrowseTransactionsView, but I don't think the resulting design is really any better.
- Snapshot 620:
  - AndroidBrowseTrasactionView now depends on less Android stuff.

    - Marked some duplication as incidental so that I don't tear my hair out trying to remove it.
- Snapshot 630:
    - Verified my theory that I don't need the UI widgets to check the contract for BrowseTransactionsView.
    - Removed obsolete code.
    - Removed obsolete comments.
- Snapshot 640: Removed obsolete code.

What's left to do

- Wire `BrowseTransactionsActivity` to `ExportAllTransactionsAsCsvToFileAction`.
    - `BrowseTransactionsActivity` needs to choose the right `WriteTextAction` with the right `File`.
        * Move the remaining Activity code into a "business delegate".
- Write contract tests for View once they stabilise
- Enter a transaction quickly and easily
- (2 upvotes) Delegate the Activity's `BrowseTransactionsView` implementation to a new class
- Find a library that provides an easy way to mark a property as "non-nullable"
    - It needs to let me throw a Programmer Mistake when I want to.
    - Evaluate the `@Nullable` and `@NotNull` annotations.
- Find a library that provides Rails-style object validations.
- Rename `androidDevicePublicStorageGateway.findPublicExternalStorageDirectory()` to something more like `findSafePlaceToWriteAFile()`, which better captures its intent.

16 At Last, Gratification

While I liked the improvements that I made in the previous session, I feel like I got a little off course, which I do from time to time. I timebox work attentively in part to ensure that I don't do this for too long, but also so that I feel like I have some latitude to try things as I think of them. Think of brakes on a car: they exist not to let you stop, but to let you go fast. Now I slam on the brakes and look at what it would take to wire `BrowseTransactionActivity` to `ExportAllTransactionsAsCsvToFileAction`.

It doesn't take long to spot where to do the work. The method `onCreate()` instantiates a do-nothing export action, which I can replace with the export-to-CSV action.

Snapshot 640: Finding a spot for the export action

```
package ca.jbrains.upfp;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.*;
import ca.jbrains.upfp.controller.android.*;
import ca.jbrains.upfp.model.*;
import ca.jbrains.upfp.presenter.*;
import ca.jbrains.upfp.view.BrowseTransactionsView;
import ca.jbrains.upfp.view.android
    .AndroidBrowseTransactionsView;
import com.google.common.collect.Lists;

import java.io.File;
import java.util.List;

public class BrowseTransactionsActivity extends Activity {
  private RendersView rendersView;
  private ExportAllTransactionsAction
      exportAllTransactionsAction;
  private AndroidDevicePublicStorageGateway
      androidDevicePublicStorageGateway;
  private BrowseTransactionsModel browseTransactionsModel;
  private BrowseTransactionsView browseTransactionsView;
```

```
27
28    public BrowseTransactionsActivity() {
29      // Do all this work in onCreate()
30    }
31
32    // REFACTOR Move this constructor into the "business
33    // delegate"
34    public BrowseTransactionsActivity(
35        RendersView rendersView,
36        ExportAllTransactionsAction
37            exportAllTransactionsAction,
38        AndroidDevicePublicStorageGateway
39            androidDevicePublicStorageGateway,
40        BrowseTransactionsModel browseTransactionsModel
41    ) {
42
43      this.rendersView = rendersView;
44      this.exportAllTransactionsAction
45          = exportAllTransactionsAction;
46      this.androidDevicePublicStorageGateway
47          = androidDevicePublicStorageGateway;
48      this.browseTransactionsModel = browseTransactionsModel;
49    }
50
51    @Override
52    protected void onResume() {
53      super.onResume();
54      // Arbitrarily, I assume that I should do my work
55      // after the superclass, but I don't really know.
56      // REFACTOR Delegate to businessDelegate.renderView()
57      rendersView.render();
58    }
59
60    /**
61     * Called when the activity is first created.
62     */
63    @Override
64    public void onCreate(Bundle savedInstanceState) {
65      super.onCreate(savedInstanceState);
66      setContentView(R.layout.main);
67
68      // This seems like a more logical place to initialise
```

```
    // the View, anyway.
    this.exportAllTransactionsAction
        = new ExportAllTransactionsAction() {
      @Override
      public void execute(List<Transaction> transactions) {
        // Do nothing, for now
      }
    };

    // SMELL I have to initialize this because I can't
    // use constructor chaining yet.
    // This has to be anything that won't throw a stupid
    // exception.
    this.browseTransactionsModel
        = new BrowseTransactionsModel() {
      @Override
      public int countTransactions() {
        return findAllTransactions().size();
      }

      @Override
      public List<Transaction> findAllTransactions() {
        return Lists.newArrayList();
      }
    };

    this.browseTransactionsView
        = new AndroidBrowseTransactionsView(
        transactionsCountView());

    // REFACTOR Delegate BrowseTransactionsView behavior
    // to a new class
    this.rendersView = new BrowseTransactionsPresenter(
        this.browseTransactionsModel,
        browseTransactionsView);

    this.androidDevicePublicStorageGateway
        = new AndroidDevicePublicStorageGateway() {
      @Override
      public File findPublicExternalStorageDirectory()
          throws PublicStorageMediaNotAvailableException {
        return new File(".");
```

```
111         }
112       };
113
114     }
115
116     private TextView transactionsCountView() {
117       return (TextView) findViewById(R.id.transactionsCount);
118     }
119
120     public void exportAllTransactions(View clicked) {
121       // I'm not entirely sure where this will end up
122       try {
123         final List<Transaction> transactions
124             = browseTransactionsModel.findAllTransactions();
125         androidDevicePublicStorageGateway
126             .findPublicExternalStorageDirectory();
127         exportAllTransactionsAction.execute(transactions);
128         notifyUser(
129             "Exported all transactions to " +
130             "/mnt/sdcard/TrackEveryPenny.csv");
131       } catch (InternalStorageException reported) {
132         handleError(
133             "Couldn't read data from internal storage",
134             "Something strange just happened. Try again. " +
135             "You might need to reinstall the application. I" +
136             " feel embarrassed and ashamed.",
137             reported);
138       } catch (PublicStorageMediaNotAvailableException
139           reported) {
140         handleError(
141             "Couldn't save a file to public storage; media " +
142             "not available",
143             "No place to which to export the transactions. " +
144             "Insert an SD card or connect an external " +
145             "storage device and try again.",
146             reported);
147       } catch (PublicStorageMediaNotWritableException
148           reported) {
149         final String pathNotWritableAsText = reported
150             .getPathNotWritable().getAbsolutePath();
151         handleError(
152             String.format(
```

```
153                 "Path %1$s not writable",
154                 pathNotWritableAsText), String.format(
155             "Permission denied trying to export the " +
156             "transactions to file %1$s",
157             pathNotWritableAsText), reported);
158       }
159     }
160
161     // REUSE Any Activity
162     private void handleError(
163         String internalMessage, String userVisibleMessage,
164         Throwable cause
165     ) {
166       logError(internalMessage, cause);
167       notifyUser(userVisibleMessage);
168     }
169
170     // REUSE Any Activity
171     private void notifyUser(String message) {
172       Toast.makeText(
173           getApplicationContext(), message, Toast.LENGTH_LONG)
174           .show();
175     }
176
177     // REUSE Anywhere in this app
178     private void logError(
179         String message, Throwable reported
180     ) {
181       Log.e("TrackEveryPenny", message, reported);
182     }
183
184     // SMELL Programming by accident
185     // This should disappear after moving other behavior
186     // into the businessDelegate.
187     public void setCollaborators(
188         ExportAllTransactionsAction
189             exportAllTransactionsAction,
190         AndroidDevicePublicStorageGateway
191             androidDevicePublicStorageGateway,
192         BrowseTransactionsModel browseTransactionsModel
193     ) {
194       this.exportAllTransactionsAction
```

```
195         = exportAllTransactionsAction;
196     this.androidDevicePublicStorageGateway
197         = androidDevicePublicStorageGateway;
198     this.browseTransactionsModel = browseTransactionsModel;
199   }
200 }
```

If I create an `ExportAllTransactionsAsCsvToFileAction` object here, that should finish the job. Of course, in order to do that, I need to move more classes from test packages to production packages.

Snapshot 650: Preparing code for its first production client

```
 AndroidFree/src/{test/java/ca/jbrains/upfp/presenter/test => main/java/ca/\
jbrains/upfp/presenter}/ExportAllTransactionsAsCsvToFileAction.java | 4 ++-\
-
 AndroidFree/src/{test/java/ca/jbrains/upfp/presenter/test => main/java/ca/\
jbrains/upfp/presenter}/WriteTextAction.java                        | 2 +-
 AndroidFree/src/{test/java/ca/jbrains/upfp/view/test => main/java/ca/jbrai\
ns/upfp/view}/AmountCsvFormat.java                                  | 3 ++-
 AndroidFree/src/{test/java/ca/jbrains/upfp/view/test => main/java/ca/jbrai\
ns/upfp/view}/CategoryCsvFormat.java                                | 3 ++-
 AndroidFree/src/{test/java/ca/jbrains/upfp/view/test => main/java/ca/jbrai\
ns/upfp/view}/CsvFormat.java                                        | 2 +-
 AndroidFree/src/{test/java/ca/jbrains/upfp/view/test => main/java/ca/jbrai\
ns/upfp/view}/CsvHeaderFormat.java                                  | 2 +-
 AndroidFree/src/{test/java/ca/jbrains/upfp/view/test => main/java/ca/jbrai\
ns/upfp/view}/DateCsvFormat.java                                    | 3 ++-
 AndroidFree/src/{test/java/ca/jbrains/upfp/view/test => main/java/ca/jbrai\
ns/upfp/view}/SurroundWithQuotes.java                               | 4 ++-\
-
 AndroidFree/src/{test/java/ca/jbrains/upfp/view/test => main/java/ca/jbrai\
ns/upfp/view}/TransactionCsvFormat.java                             | 2 +-
 AndroidFree/src/{test/java/ca/jbrains/upfp/view/test => main/java/ca/jbrai\
ns/upfp/view}/TransactionsCsvFileFormat.java                        | 2 +-
 AndroidFree/src/{test/java/ca/jbrains/upfp/view/test => main/java/ca/jbrai\
ns/upfp/view}/TransactionsCsvHeader.java                            | 4 +++\
-
 AndroidFree/src/{test => main}/java/ca/jbrains/upfp/view/WriteTextToFileAc\
tion.java                                                           | 2 +-
 AndroidFree/src/test/java/ca/jbrains/upfp/presenter/test/ExportAllTransact\
ionsAsCsvToFileActionTest.java                                      | 4 +++\
```

```
-
 AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatAmountAsCsvTest.\
java                                                                | 1 +
 AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatCategoryAsCsvTes\
t.java                                                              | 1 +
 AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatLocalDateAsCsvTe\
st.java                                                             | 1 +
 AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatTransactionAsCsv\
RowTest.java                                                        | 1 +
 AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatTransactionCsvHe\
aderTest.java                                                       | 1 +
 AndroidFree/src/test/java/ca/jbrains/upfp/view/test/FormatTransactionsAsCs\
vFileTest.java                                                      | 3 +++
 19 files changed, 30 insertions(+), 15 deletions(-)
```

Yet Another Chunnel Problem

While writing the one line of code to wire everything together, I discovered a Chunnel Problem. In order to instantiate `WriteTextToFileAction()`, I need to know the path where I intend to write the CSV file. So far, the code doesn't compute that until it handles the "export all transactions" event. This kind of thing happens to me sometimes when I design from the bottom up. It seems strange to choose the CSV filename at the application scope. Yes, I can get away with it for now, but it sounds like a request-scope kind of detail. I feel like this will come back to haunt me as I add more features, but for now, I simply take advantage of the fact that I don't yet let the user choose the path for the exported transactions. I really dislike this choice, however, because it forces the `Activity` to instantiate the `AndroidDevicePublicStorageGateway` before instantiating the `WriteTextToFileAction`.

But wait! There's a clue in that last sentence. Perhaps I can combine the *public storage gateway* with writing text to a file. They seem to go together: if I write to something other than an SD Card, then I might write text to something other than a file. I will explore that later. Until then, I introduce the temporal coupling and make a note of it.

I had to move some code around which, combined with a quirk in the diff algorithm, makes this diff particularly hard to read. I'm really sorry about that, and I will fix it in a later draft.

I've added SMELL comments to highlight the temporal coupling I've introduced. Initialising the Public Storage Gateway and the Export All Transactions actions have switched position. The Public Storage Gateway code looks like it has changed, but it hasn't. I've added code to compute the target path for exporting the transactions before initialising the Export All Transactions action.

Snapshot 660: Grudgingly adding temporal coupling to the Activity

```
diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
index a5d2fa2..58e4035 100644
--- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
+++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
@@ -8,12 +8,12 @@ import android.widget.*;
 import ca.jbrains.upfp.controller.android.*;
 import ca.jbrains.upfp.model.*;
 import ca.jbrains.upfp.presenter.*;
-import ca.jbrains.upfp.view.BrowseTransactionsView;
+import ca.jbrains.upfp.view.*;
 import ca.jbrains.upfp.view.android
     .AndroidBrowseTransactionsView;
 import com.google.common.collect.Lists;

-import java.io.File;
+import java.io.*;
 import java.util.List;

 public class BrowseTransactionsActivity extends Activity {
@@ -67,14 +67,62 @@ public class BrowseTransactionsActivity extends Activit\
y {

     // This seems like a more logical place to initialise
     // the View, anyway.
-    this.exportAllTransactionsAction
-        = new ExportAllTransactionsAction() {
+
+    // SMELL This has to happen before instantiating the
+    // WriteTextToFileAction
+    this.androidDevicePublicStorageGateway
+        = new AndroidDevicePublicStorageGateway() {
       @Override
-      public void execute(List<Transaction> transactions) {
-        // Do nothing, for now
+      public File findPublicExternalStorageDirectory()
+          throws PublicStorageMediaNotAvailableException {
+        return new File(".");
       }
     };
```

```
41
42 +    // SMELL This needs the
43 +    // AndroidDevicePublicStorageGateway,
44 +    // but doesn't depend directly on it
45 +    final File exportedTransactionsPath;
46 +    try {
47 +      exportedTransactionsPath = new File(
48 +          androidDevicePublicStorageGateway
49 +              .findPublicExternalStorageDirectory(),
50 +          "TrackEveryPenny.csv");
51 +    } catch (PublicStorageMediaNotAvailableException
52 +        reported) {
53 +      handleError(
54 +          "Couldn't save a file to public storage; media " +
55 +          "not available",
56 +          "No place to which to export the transactions. " +
57 +          "Insert an SD card or connect an external " +
58 +          "storage device and try again.",
59 +          reported);
60 +      return;
61 +    } catch (PublicStorageMediaNotWritableException
62 +        reported) {
63 +      final String pathNotWritableAsText = reported
64 +          .getPathNotWritable().getAbsolutePath();
65 +      handleError(
66 +          String.format(
67 +              "Path %1$s not writable",
68 +              pathNotWritableAsText), String.format(
69 +          "Permission denied trying to export the " +
70 +          "transactions to file %1$s",
71 +          pathNotWritableAsText), reported);
72 +      return;
73 +    }
74 +
75 +    this.exportAllTransactionsAction
76 +        = new ExportAllTransactionsAsCsvToFileAction(
77 +        new TransactionsCsvFileFormat(
78 +            new TransactionsCsvHeader(),
79 +            new TransactionCsvFormat(
80 +                new DateCsvFormat(),
81 +                new CategoryCsvFormat(),
82 +                new AmountCsvFormat())),
```

```
+        new WriteTextToFileAction(
+            exportedTransactionsPath));
+
     // SMELL I have to initialize this because I can't
     // use constructor chaining yet.
     // This has to be anything that won't throw a stupid
@@ -101,16 +149,6 @@ public class BrowseTransactionsActivity extends Activi\
ty {
     this.rendersView = new BrowseTransactionsPresenter(
         this.browseTransactionsModel,
         browseTransactionsView);
-
-    this.androidDevicePublicStorageGateway
-        = new AndroidDevicePublicStorageGateway() {
-      @Override
-      public File findPublicExternalStorageDirectory()
-          throws PublicStorageMediaNotAvailableException {
-        return new File(".");
-      }
-    };
-
   }

   private TextView transactionsCountView() {
@@ -155,6 +193,8 @@ public class BrowseTransactionsActivity extends Activit\
y {
           "Permission denied trying to export the " +
           "transactions to file %1$s",
           pathNotWritableAsText), reported);
+    } catch (IOException unhandled) {
+      throw new RuntimeException(unhandled);
     }
   }

@@ -187,8 +227,7 @@ public class BrowseTransactionsActivity extends Activit\
y {
   public void setCollaborators(
       ExportAllTransactionsAction
           exportAllTransactionsAction,
-      AndroidDevicePublicStorageGateway
-          androidDevicePublicStorageGateway,
+      AndroidDevicePublicStorageGateway androidDevicePublicStorageGateway,
```

```
125         BrowseTransactionsModel browseTransactionsModel
126     ) {
127       this.exportAllTransactionsAction
128 @@ -197,4 +236,5 @@ public class BrowseTransactionsActivity extends Activit\
129 y {
130           = androidDevicePublicStorageGateway;
131       this.browseTransactionsModel = browseTransactionsModel;
132     }
133 +
134  }
```

In the process of wiring this code together, the class that exports transactions to a CSV file now needs to implement `ExportAllTransactionsAction`. This exposes another typical Chunnel Problem annoyance: an unhandled checked exception. This came about when I implemented `WriteTextAction`, where it makes perfect sense for `writeText()` to throw an `IOException`. If writing to text can fail due to an I/O problem, then certainly exporting a set of transactions can fail for the same reasons, so letting the checked exception bubble up makes sense to me. Unfortunately, this means adding `IOException` to the contract of `ExportAllTransactionsAction`, which I don't like in general, but can reasonably easily justify in this particular case. If it turns out to cause bigger problems, then I can always wrap the `IOException` in some kind of more generic `ExportException`, although as I typed that name, I immediately hated it. All the more reason to feel comfortable renaming things later!

Snapshot 660: Fine-tuning the Export Transactions Action to fit things together

```
1  diff --git a/AndroidFree/src/main/java/ca/jbrains/upfp/presenter/ExportAllT\
2  ransactionsAsCsvToFileAction.java b/AndroidFree/src/main/java/ca/jbrains/up\
3  fp/presenter/ExportAllTransactionsAsCsvToFileAction.java
4  index 5ab96e7..41b5e58 100644
5  --- a/AndroidFree/src/main/java/ca/jbrains/upfp/presenter/ExportAllTransact\
6  ionsAsCsvToFileAction.java
7  +++ b/AndroidFree/src/main/java/ca/jbrains/upfp/presenter/ExportAllTransact\
8  ionsAsCsvToFileAction.java
9  @@ -6,7 +6,8 @@ import ca.jbrains.upfp.view.CsvFormat;
10  import java.io.IOException;
11  import java.util.List;
12
13 -public class ExportAllTransactionsAsCsvToFileAction {
14 +public class ExportAllTransactionsAsCsvToFileAction
15 +    implements ExportAllTransactionsAction {
16     private final CsvFormat<List<Transaction>>
```

```
17          transactionsFileFormat;
18      private WriteTextAction writeTextAction;
```

```
diff --git a/AndroidFree/src/main/java/ca/jbrains/upfp/presenter/ExportAllT\
ransactionsAction.java b/AndroidFree/src/main/java/ca/jbrains/upfp/presente\
r/ExportAllTransactionsAction.java
index 8dc7d43..35860fa 100644
--- a/AndroidFree/src/main/java/ca/jbrains/upfp/presenter/ExportAllTransact\
ionsAction.java
+++ b/AndroidFree/src/main/java/ca/jbrains/upfp/presenter/ExportAllTransact\
ionsAction.java
@@ -2,8 +2,10 @@ package ca.jbrains.upfp.presenter;

 import ca.jbrains.upfp.model.Transaction;

+import java.io.IOException;
 import java.util.List;

 public interface ExportAllTransactionsAction {
-  void execute(List<Transaction> transactions);
+  void execute(List<Transaction> transactions)
+      throws IOException;
 }
```

I noticed something unexpected in the last test run: the file `TrackEveryPenny.csv`. I've ignored it for now, but I don't expect any of my current tests actually to create that file on disk. Good news: exporting no transactions works from end to end, at least running in the Robolectric simulated Android framework. Even so, this signals quite strongly to me, along with the temporal coupling that I've just introduced to put the remaining pieces together, that `BrowseTransactionActivity` does too much. But first, let me explain the origin of the unexpected file.

Early on I wrote the test `exportAllTransactionsButtonDoesNotBlowUp()` (in `RenderBrowseTransactionsScreenTes` when I wanted to verify that I'd wired an event handler to the "export all transactions" button in the layout. This test instantiates `BrowseTransactionActivity` using the production constructor, rather than by injecting mock collaborators. It has acted as an integrated test all along, but I didn't notice it until now, because until now that didn't create any problems, and suddenly it has. I can fix this quickly, but by introducing yet more temporal coupling and programming by accident. I *will* clean this up, I promise.

Snapshot 670: Turning an integrated test into an isolated object test

```
diff --git a/src/test/java/ca/jbrains/upfp/controller/android/test/RenderBr\
owseTransactionsScreenTest.java b/src/test/java/ca/jbrains/upfp/controller/\
android/test/RenderBrowseTransactionsScreenTest.java
index d56412b..ae0c2b3 100644
--- a/src/test/java/ca/jbrains/upfp/controller/android/test/RenderBrowseTra\
nsactionsScreenTest.java
+++ b/src/test/java/ca/jbrains/upfp/controller/android/test/RenderBrowseTra\
nsactionsScreenTest.java
@@ -2,7 +2,10 @@ package ca.jbrains.upfp.controller.android.test;

 import android.widget.Button;
 import ca.jbrains.upfp.*;
-import ca.jbrains.upfp.presenter.RendersView;
+import ca.jbrains.upfp.controller.android
+    .AndroidDevicePublicStorageGateway;
+import ca.jbrains.upfp.model.BrowseTransactionsModel;
+import ca.jbrains.upfp.presenter.*;
 import com.xtremelabs.robolectric.RobolectricTestRunner;
 import org.jmock.*;
 import org.junit.Test;
@@ -43,10 +46,30 @@ public class RenderBrowseTransactionsScreenTest {
   @Test
   public void exportAllTransactionsButtonDoesNotBlowUp()
       throws Exception {
+    final ExportAllTransactionsAction
+        exportAllTransactionsAction = mockery.mock(
+        ExportAllTransactionsAction.class);
+    final AndroidDevicePublicStorageGateway
+        androidDevicePublicStorageGateway = mockery.mock(
+        AndroidDevicePublicStorageGateway.class);
+    final BrowseTransactionsModel browseTransactionsModel
+        = mockery.mock(BrowseTransactionsModel.class);
+
     final BrowseTransactionsActivity
         browseTransactionsActivity
         = new BrowseTransactionsActivity();
     browseTransactionsActivity.onCreate(null);
+    browseTransactionsActivity.setCollaborators(
+        exportAllTransactionsAction,
+        androidDevicePublicStorageGateway,
```

```
41 +        browseTransactionsModel);
42 +
43 +    mockery.checking(
44 +        new Expectations() {{
45 +          ignoring(exportAllTransactionsAction);
46 +          ignoring(androidDevicePublicStorageGateway);
47 +          ignoring(browseTransactionsModel);
48 +        }});
49
50      final Button exportAllTransactionsButton
51          = (Button) browseTransactionsActivity.findViewById(
```

Mocks Doing Their Job

Here I've used so-called "setter injection", which allows me to change the Activity's collaborators after instantiating it. Unfortunately, I also have to know to inject these collaborators *after* simulating the Activity's `create` event, which adds to the programming-by-accident that I already had to do. This provides another, stronger, signal to extract behavior from the Activity into another object that doesn't have to concern itself with the Activity event lifecycle of `create`, `destroy`, `reload`, and so on.

Look at the mocks I had to add. Three of them, and I ignore them all! This causes people to **hate** mocks, but at times like these, **I absolutely love them**. Let me explain.

In order to test something as simple as "I have wired a button to an event handler", not only do I have to ignore all the Activity's collaborating interfaces, but I have nothing useful to check at the end. I've written tests before that pass only if the action doesn't blow up, so this felt normal and didn't signal me that something might need to change. On the other hand, instantiating three mocks just to ignore them all makes for a pretty clear signal: **ABSTRACTION MISSING**. This further convinces me that `BrowseTransactionsActivity` does too many things, and that I would benefit from separating the two responsibilities of wiring event handlers to UI widgets and handling those events.

These accumulating signals lead me to a major design improvement, but I don't want that to distract me from my current direction. Clearly `BrowseTransactionsActivity` has turned into a real mess, but I want – no, **need** – to push towards a working export feature, and there remain two missing pieces:

- A working model, which can simply wrap an in-memory list of `Transaction` objects.
- A working `AndroidDevicePublicStorageGateway`, which can simply implement the example I read in the Android Developer's Guide at http://link.jbrains.ca/Ugc6AQ[1]

[1] http://link.jbrains.ca/Ugc6AQ

In the interest of seeing this feature work, I will implement these two items simply and directly, tolerating all the design problems around me, then return to those design problems in the next session.

I choose the `AndroidDevicePublicStorageGateway` first, so that I can end with the simplest task, one that could not possibly result in a tangent or uncover a design problem. According to the Android Developer's Guide, in consultation with the contract for `AndroidDevicePublicStorageGateway`, we need to check three cases: happy path, media not available, and media not writable. The contract for `AndroidDevicePublicStorageGateway` says to throw an exception for each of the failure cases, otherwise return a `File` whose path is the root of the writable, external public storage. It sounds simple enough, and so three tests ought to suffice.

In order to simplify matters, I will test-drive a function to evaluate the result of `Environment.getExternalStorageSt` then write one line of untested code that passes `Environment.getExternalStorageState()` into that function. The one line of untested won't kill me, and I can decide later whether to figure out how to test it. I might never bother.

Once again, due to my strange choice of interface name, I implement an Impl. I will fix this. I tried to introduce an interface to hide Environment, but I ended up chasing my own tail, because I wanted to name that interface something very close to `PublicExternalStorageGateway`. Since that didn't work out, I'll subclass to test.

Snapshot 680: Subclassing the storage gateway to test

```
1  package ca.jbrains.upfp.model.android.test;
2
3  import android.os.Environment;
4  import ca.jbrains.upfp.controller.android.*;
5  import ca.jbrains.upfp.model.android
6      .AndroidDevicePublicStorageGatewayImpl;
7  import org.junit.Test;
8
9  import java.io.File;
10
11 import static org.junit.Assert.*;
12
13 public class InitialiseExternalStorageTest {
14   @Test
15   public void happyPath() throws Exception {
16     final File externalStoragePublicDirectory = new File(
17         "/irrelevant/path");
18
19     assertEquals(
20         externalStoragePublicDirectory,
```

```
        new AndroidDevicePublicStorageGatewayImpl() {
          @Override
          public String getExternalStorageState() {
            return Environment.MEDIA_MOUNTED;
          }

          @Override
          public File getExternalStoragePublicDirectory() {
            return externalStoragePublicDirectory;
          }
        }.findPublicExternalStorageDirectory());
  }

  @Test
  public void mediaMountedInReadOnlyMode()
      throws Exception {
    final File externalStoragePublicDirectory = new File(
        "/irrelevant/path");

    try {
      new AndroidDevicePublicStorageGatewayImpl() {
        @Override
        public String getExternalStorageState() {
          return Environment.MEDIA_MOUNTED_READ_ONLY;
        }

        @Override
        public File getExternalStoragePublicDirectory() {
          return externalStoragePublicDirectory;
        }
      }.findPublicExternalStorageDirectory();
      fail(
          "Why did you give the client a directory that " +
          "they can't write to?!");
    } catch (PublicStorageMediaNotWritableException
        success) {
      assertEquals(
          externalStoragePublicDirectory,
          success.getPathNotWritable());
    }
  }

```

```
  @Test
  public void mediaNotAvailable() throws Exception {
    try {
      new AndroidDevicePublicStorageGatewayImpl() {
        @Override
        public String getExternalStorageState() {
          // Anything except (MEDIA_MOUNTED or
          // MEDIA_MOUNTED_READ_ONLY)
          return Environment.MEDIA_UNMOUNTABLE;
        }
      }.findPublicExternalStorageDirectory();
      fail(
          "Why did you give the client a directory that " +
          "they can't write to?!");
    } catch (PublicStorageMediaNotAvailableException success) {
    }
  }
}
```

The resulting class provides a narrowing API for `Environment`, exposing only the behavior related to public external storage. It shows how to apply both the Single Responsibility Principle and the Interface Segregation Principle to legacy code.

Snapshot 680: Narrowing the interface of Environment

```
package ca.jbrains.upfp.model.android;

import android.os.Environment;
import ca.jbrains.upfp.controller.android.*;

import java.io.File;

public class AndroidDevicePublicStorageGatewayImpl
    implements AndroidDevicePublicStorageGateway {
  public String getExternalStorageState() {
    return Environment.getExternalStorageState();
  }

  public File getExternalStoragePublicDirectory() {
    return Environment.getExternalStoragePublicDirectory(
        Environment.DIRECTORY_DOWNLOADS);
```

```
  }

  @Override
  public File findPublicExternalStorageDirectory()
      throws PublicStorageMediaNotAvailableException,
             PublicStorageMediaNotWritableException {
    if (Environment.MEDIA_MOUNTED.equals(
        getExternalStorageState()))
      return getExternalStoragePublicDirectory();

    if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(
        getExternalStorageState()))
      throw new PublicStorageMediaNotWritableException(
          getExternalStoragePublicDirectory());

    throw new PublicStorageMediaNotAvailableException();
  }
}
```

Finally, the Activity wants to use the new storage gateway.

Snapshot 680: Wiring the storage gateway to the Activity

```
diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
index 58e4035..d32381f 100644
--- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
+++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
@@ -7,6 +7,8 @@ import android.view.View;
 import android.widget.*;
 import ca.jbrains.upfp.controller.android.*;
 import ca.jbrains.upfp.model.*;
+import ca.jbrains.upfp.model.android
+    .AndroidDevicePublicStorageGatewayImpl;
 import ca.jbrains.upfp.presenter.*;
 import ca.jbrains.upfp.view.*;
 import ca.jbrains.upfp.view.android
@@ -71,13 +73,7 @@ public class BrowseTransactionsActivity extends Activity\
 {
     // SMELL This has to happen before instantiating the
     // WriteTextToFileAction
```

```
19       this.androidDevicePublicStorageGateway
20  -        = new AndroidDevicePublicStorageGateway() {
21  -      @Override
22  -      public File findPublicExternalStorageDirectory()
23  -          throws PublicStorageMediaNotAvailableException {
24  -        return new File(".");
25  -      }
26  -    };
27  +        = new AndroidDevicePublicStorageGatewayImpl();
28
29       // SMELL This needs the
30       // AndroidDevicePublicStorageGateway,
```

The new storage gateway implementation has two responsibilities: the narrowing interface for `Environment` and connecting to external public storage. I'd like to split these apart, and in the process, would probably introduce an interface with a name similar to `ExternalPublicStorageGateway`, but that part of the code needs more generous cleaning up, so I add that to the backlog.

Finally – I hope – I hardcode some transactions in the `BrowseTransactionsModel` instance inside the `BrowseTransactionsActivity`, fire up the application, and try exporting those transactions to CSV.

Snapshot 690: Hardcoding transactions to poke around with the app

```
1  diff --git a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java \
2  b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
3  index d32381f..3612f70 100644
4  --- a/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
5  +++ b/src/main/java/ca/jbrains/upfp/BrowseTransactionsActivity.java
6  @@ -14,6 +14,7 @@ import ca.jbrains.upfp.view.*;
7   import ca.jbrains.upfp.view.android
8       .AndroidBrowseTransactionsView;
9   import com.google.common.collect.Lists;
10 +import org.joda.time.LocalDate;
11
12  import java.io.*;
13  import java.util.List;
14 @@ -132,7 +133,16 @@ public class BrowseTransactionsActivity extends Activi\
15 ty {
16
17        @Override
18        public List<Transaction> findAllTransactions() {
```

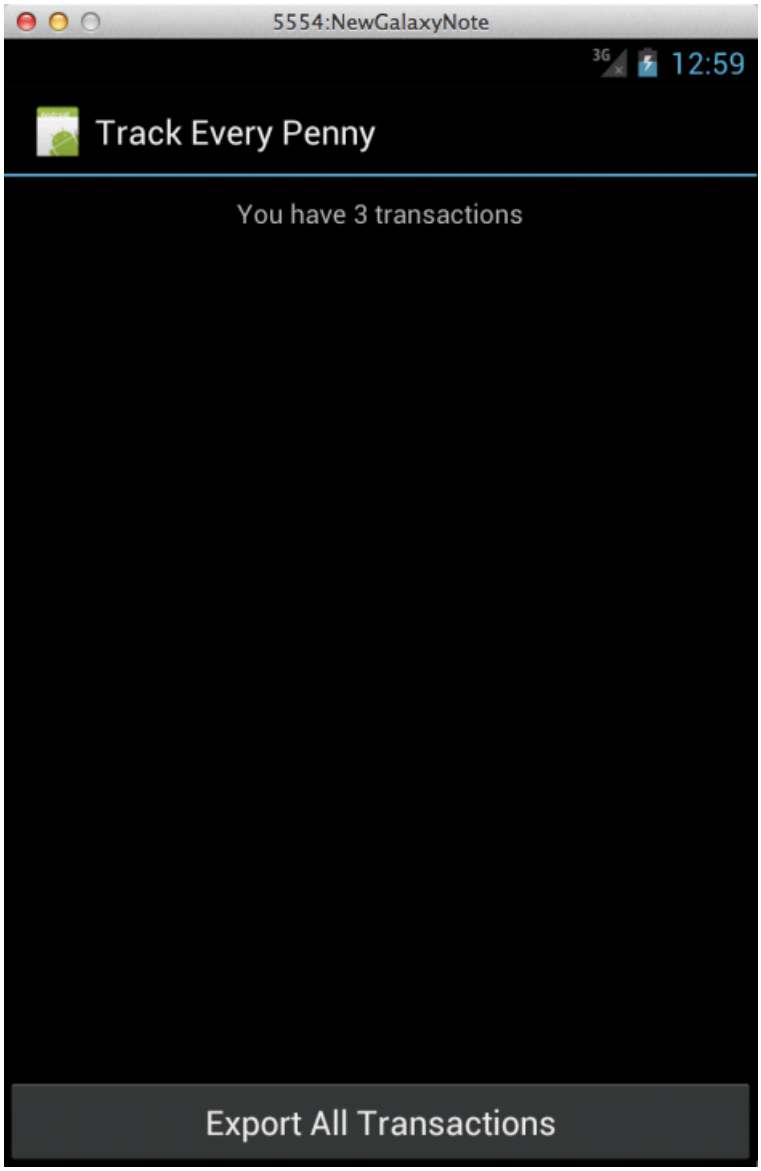
```
-         return Lists.newArrayList();
+         return Lists.newArrayList(
+             new Transaction(
+                 new LocalDate(2012, 11, 19), new Category(
+                 "Groceries"), Amount.cents(
+                 -11748)), new Transaction(
+             new LocalDate(
+                 2012, 11, 19), new Category("Books"),
+             Amount.cents(-799)), new Transaction(
+             new LocalDate(2012, 11, 19), new Category(
+             "Utilities"), Amount.cents(-21034)));
        }
      };

```

The Thrilling Conclusion

Next, I fire up the simulator.

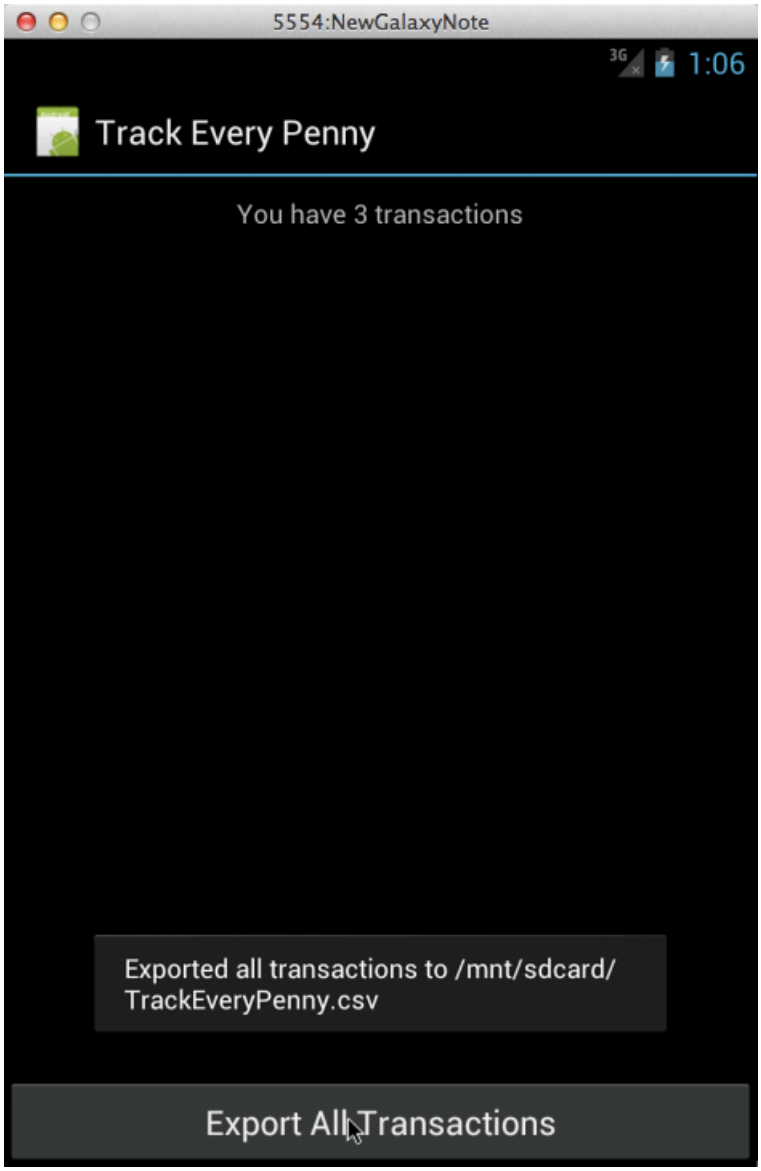
> I generally consider it a good sign when I decide to fire up the simulator and have to re-learn how to do it.

I ask IDEA to run the application and see the familiar opening screen, now with **3 transactions**.



Before pressing the magic button

I press the “Export All Transactions” button.



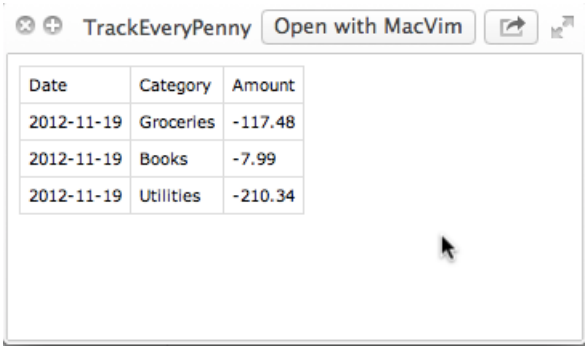
It claims to have done something

The message looks fine, so now I need to check the output. This requires opening the SD Card disk image, which took me rather a long time to learn to do.

Android gives you access to the SD Card as a disk image (at least on Mac OS), but you have to know where to find it. I found mine at `$HOME/.android/avd/NewGalaxyNote.avd/sdcard.img`. When I'd installed the Android simulator, I'd accepted all the defaults, so if you did the same, then you should look in the same place. I'd named my Android Virtual Device (`avd`) "NewGalaxyNote", so you should look for a folder matching the name you gave to your AVD. When I opened the disk image, it had the typical default Mac OS label `NO NAME`.

Don't leave the disk image open while running the simulator. I learned that the hard way, too. Open the disk image, inspect it, then close it, so that the simulator can continue to write to it. I think I managed to limit my wasted time to about an hour learning this little tip.

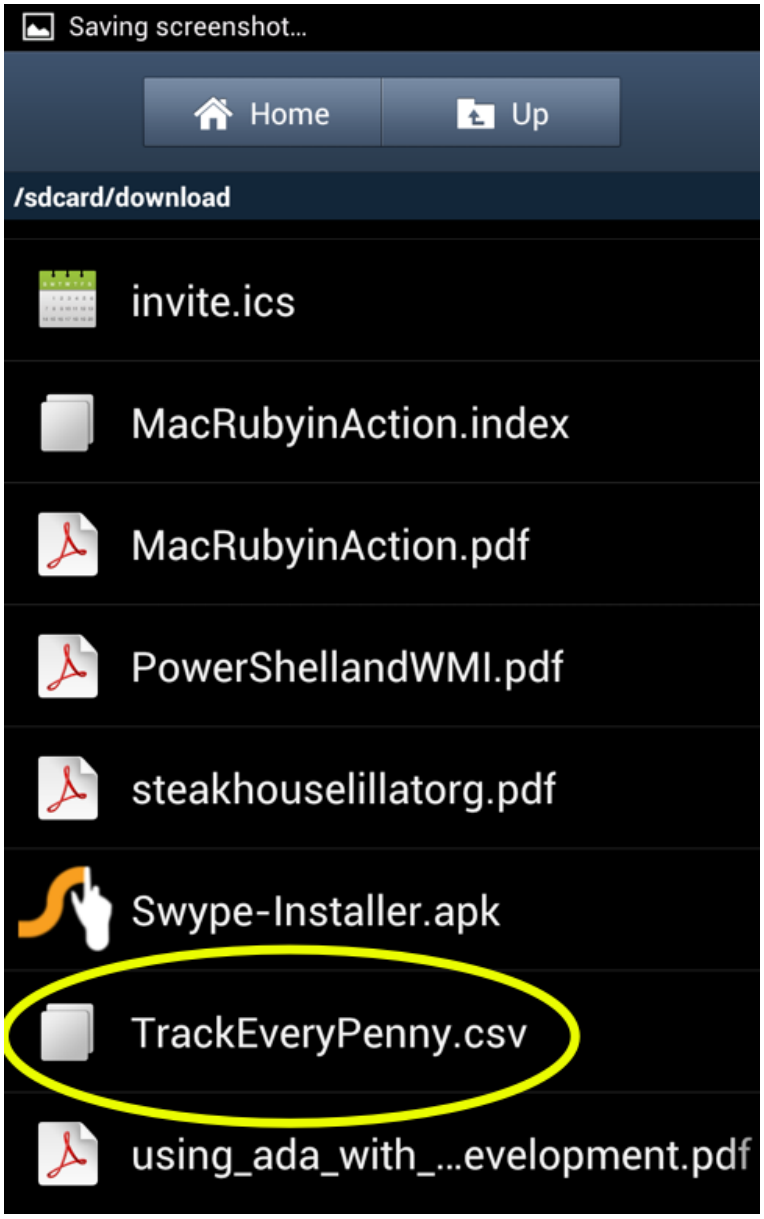
I located the transactions file inside the disk image at `Download/TrackEveryPenny.csv`. A quick peek showed me some familiar transactions.



We have transactions!

Again, But Even More Real

Since I saw it behaving as expected in the simulator, I wanted to see it on my phone. Pressing the button looks exactly the same, so I won't bother showing that to you again.



I see a transaction file on my SD Card

I had to install a CSV viewer app to truly appreciate the contents of this file.

No.	Date	Category	Amount
1	2012-11-19	Groceries	-117.48
2	2012-11-19	Books	-7.99
3	2012-11-19	Utilities	-210.34

We have transactions for real!

Is That All?!

I admit that this feels like a slight anticlimax. I've written a lot of code, refactored a lot, considered a lot of design decisions, even deferred some much-needed improvements, and the app doesn't do much yet. On the other hand, in only two days of disciplined hacking, I've seen this system evolve towards a sustainable, clean architecture that I believe I can use to develop all the remaining features I need. Sure, I'll learn some more details along the way, but I've gone from no knowledge of Android development to a Walking Skeleton with a solid architecture minimising the code that has to run in the simulator. I have 59 passing tests that run fairly quickly.[2] With a few more hours of refactoring, I will have a pattern that I can follow to implement more features.

Join me.

What's done

- Snapshot 640: Removed obsolete code.
- Snapshot 650: Moved classes into production packages to make them available to their first production client.
- Snapshot 660: BrowseTransactionsActivity now uses the real ExportAllTransactionsAction in production, but unfortunately, now also in some of its tests.
- Snapshot 670: Turned an integrated test into an isolated object test, using an ugly hack that makes me want to change things.
- Snapshot 680:
  - AndroidDevicePublicStorageGatewayImpl handles the happy path, but without integrating with Environment.

[2] Java startup time still dominates the run, which averages 3 seconds elapsed time on my machine, but IDEA reports the test run portion at about 500 ms on average (118 tests per second). Not blazing fast, but only 2/3 of the tests so far run Android free. For the next few features, I imagine I'll write at least 3/4 Android-free tests. If I do, then 200 tests will run in under 1.5 second (137 tests per second) and 1000 would run in about 6.9 seconds (145 tests per second). Still not blazing fast, but I can write a lot of good code with 1000 tests and my total test run would take less than 10 seconds. If I don't deliver a valuable app in under 1000 tests, then I've done something wrong.

  - AndroidDevicePublicStorageGatewayImpl signals when the media is not writable.
  - AndroidDevicePublicStorageGatewayImpl handles the case where the media is not available.
  - Wired AndroidDevicePublicStorageGatewayImpl to the Android Environment class that connects to external public storage.
  - Wired AndroidDevicePublicStorageGatewayImpl together with BrowseTransactionsActivity.
- Snapshot 690: Hardcoded some sample transactions for manual testing.

What's left to do

- Try to combine AndroidDevicePublicStorageGateway and WriteTextToFileAction to remove the temporal coupling inside BrowseTransactionActivity.onCreate().
- Extract narrowing interface from AndroidDevicePublicStorageGatewayImpl that hides Environment.
  - Clean up the surrounding code. Figure out where the dividing line is between Android-Free and Android-Dependent code.
- Write contract tests for View once they stabilise
- Enter a transaction quickly and easily
- (2 upvotes) Delegate the Activity's BrowseTransactionsView implementation to a new class
- Find a library that provides an easy way to mark a property as "non-nullable"
  - It needs to let me throw a Programmer Mistake when I want to.
  - Evaluate the @Nullable and @NotNull annotations.
- Find a library that provides Rails-style object validations.
- Rename androidDevicePublicStorageGateway.findPublicExternalStorageDirectory() to something more like findSafePlaceToWriteAFile(), which better captures its intent.

17 Interlude: I Learned Some Things

So that felt like a long and winding road, as it often does when I work with an unfamiliar platform, framework and libraries. I end up mixing the two predominant styles of design somewhat haphazardly. In some places I strive to keep things separate, designing in what some people call the "London school", guided by the principles of Presenter-First Design that I first learned from "Presenter First: Organizing Complex GUI Applications for Test-Driven Development"[1]. In other places I throw everything into the same spot and use duplication and naming problems to guide my refactoring, designing in what some people call the "Detroit school", guided more directly by the Four Elements of Simple Design. We use the terms "London school" and "Detroit school" to recognise the people from whom we've learned the underlying principles. The book *Growing Object-Oriented Software Guided by Tests*, written by distinguished London programmers Steve Freeman and Nat Pryce, illustrates the "London school" approach centered on strict application of the Dependency Inversion and Interface Segregation Principles, which involves heavy and highly-effective use of test doubles. Detroit-adjacent programmers Ron Jeffries and Chet Hendrickson (along with several non-Detroit-adjacent programmers like Joshua Kerievsky, James Shore and Arlo Belshee) exemplify the "Detroit school" approach which centers on a more intuitive and direct application of the Four Elements of Simple Design, in which they avoid test doubles and use the resulting tension to guide them towards designs that better reflect the Single Responsibility Principle. Many commentators see this as schism, but I don't. As you've seen in this book so far, I use and value both approaches, although I don't agree with some of my colleagues' disdain for test doubles. I prefer not to make this an issue of "to mock or not to mock", since that takes focus away from where I think it belongs.

I distinguish the London and Detroit schools a different way: the Detroit school approach favors those who like to build big things, then break them apart as they become unwieldy; but the London school approach favors those who like to build little things, then put them together as they exhibit properties of belonging together. While I see many authors[2] claiming that either approach objectively outperforms the other, I can't agree. As with most such debates, I find I have better results from knowing and practising both approaches when they seem to fit the situation. Broadly speaking, I use the London school approach when I feel confident in making one or two up-front high-level design ("architecture") decisions, such as "I will use MVC/MVP to design this part of the system" or "I will model these interactions as publish/subscribe events instead of direct method invocations"; I use the Detroit school approach when I don't feel confident about making those decisions. The London school approach feels more like methodical, safe, steady design, whereas the Detroit school approach feels more exploratory or meandering (in the *good* sense, like wandering through a garden, unworried about the path, trusting that I'll get where I need to go). Some critics of the Detroit school dislike the feeling of having to forget what they already know about design and re-discover

[1] http://link.jbrains.ca/presenter-first-agile-2006

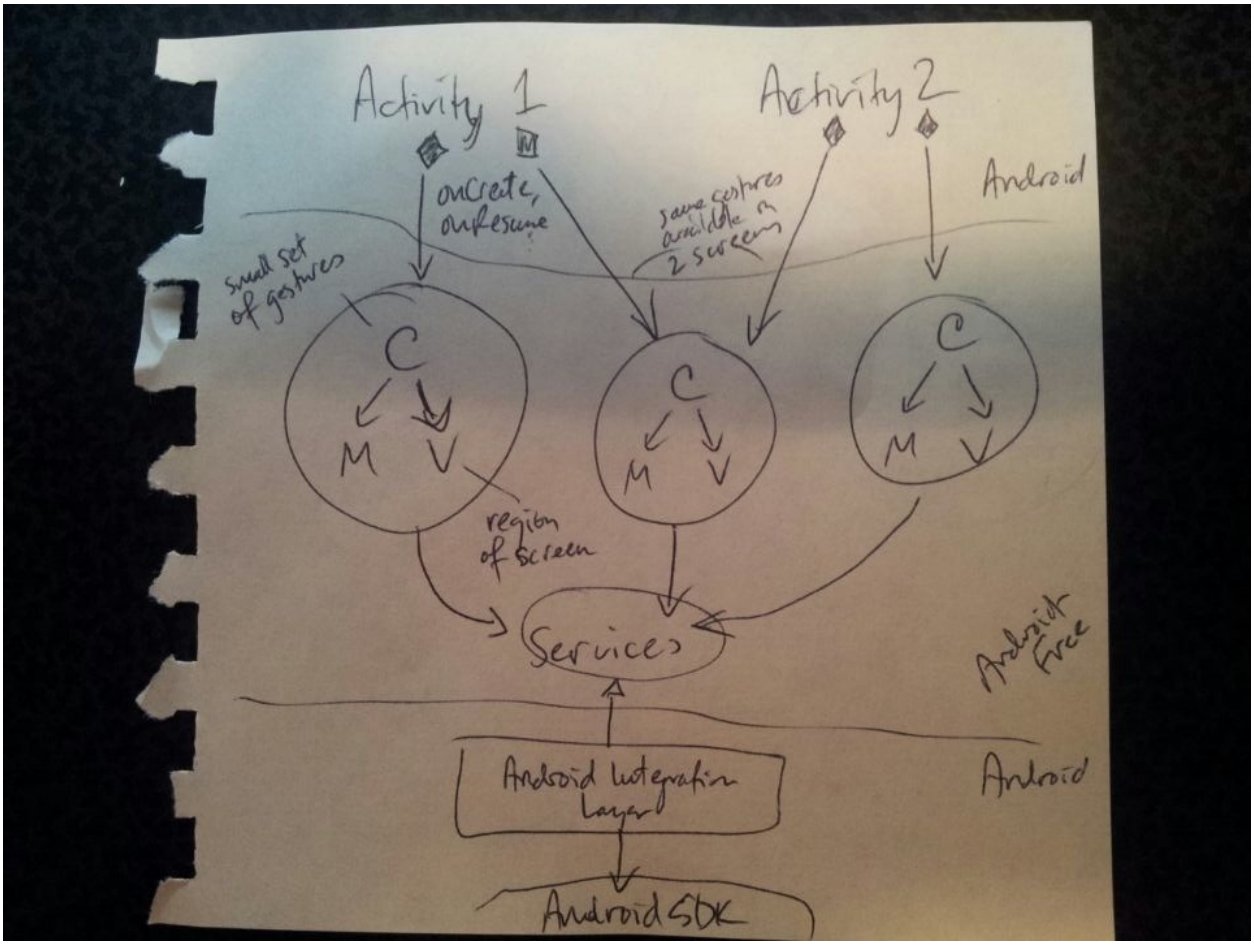
[2] For example: Arlo Belshee's "The No Mocks Book", chromatic's "Mock Objects Despoil Your Tests"…

everything from first principles; some critics of the London school feel that test doubles lock their design in and resist change. I see it another way: the Detroit school allows me to meander safely when I have a general idea of the destination, but an unclear picture of the route; the London school encourages me to build tiny things and put them together, and when I find that a piece needs to change shape, I can throw that piece away and design a new one in its place without paying a prohibitive cost for it. I think of the London school approach as a shortcut that I can take when I think I recognise what I need to build and the Detroit school approach on which I can fall back when I just don't know what to do next. Thus, I benefit the most from practising both, feeling comfortable with both, and even feeling comfortable refactoring from one style of design to the other. Please don't limit this question in your mind to "mocks suck" or "mocks rock", no matter how pithy you find those two slogans.

With that out of the way, I'd like to move on to a new feature, but I don't intend to write another painstaking line-by-line account of delivering it. Instead, I will take what I've learned from the preceding several chapters, use that to guide my design more carefully, then share the highlights with you. If you want the gory details, then you can read the micro-commits directly. I will continue to apply the London school approach to the feature's high-level design ("architecture"), particularly to maintain as much Android-free code as I can, while applying the Detroit school approach inside the adapter layer that exposes Android library services, like external storage, to better discover the appropriate Android-free API to expose to the Android-free module. I think we can agree that I did a poor job of guessing at the boundary between Android's external storage and the Android-free module, resulting in code that didn't depend directly on Android's libraries, but that nonetheless had Android-specific (or at least Android-leaning) names in the code. This provides a perfect example of a Leaky Abstraction[3], so I'd like not to repeat those choices. When I find that I can improve design elements from the first feature to make them fit the choices I make in the next few features, then I will refactor the old code, but *I will mostly leave the old code alone until that happens*. I believe that this embodies the "responsible" aspect of Responsible Design: rather than squeeze every last bit of cleanliness out of this code and defer delivery of new features, I will use what I've learned to deliver the next feature with a cleaner design, improving old code only when I *need* to touch it. Rather than try to over-polish what I've built, I'll use the Boy Scout Rule as a guideline – leave the (legacy) design in better condition than I found it. Yes: writing legacy code *can be* this easy! Of course, as I refine my design practice over the years, I find that I now consider "legacy code" designs that I'd have considered "clean" only a few years ago.

Of the things I've learned so far, one idea dominates: the two primary integration points between the Android SDK and my Plain Old Java code. In short, the Android Activity manages the lifecycle of an MVC triad, which uses services – such as persistence or logging – that an object in the Android Integration Layer might implement using the Android SDK. My current conception of Responsible Android Application Architecture looks like this.

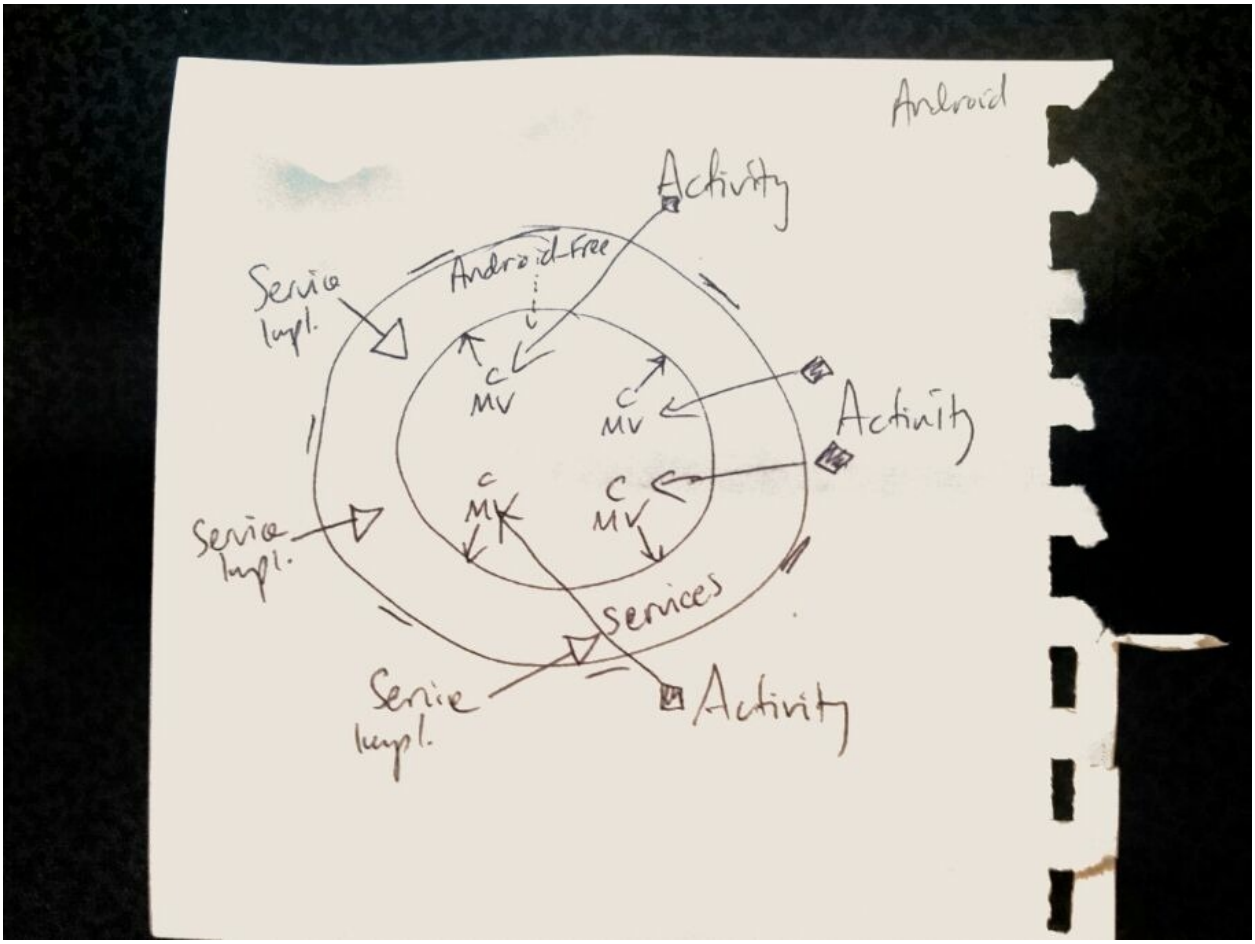
[3]Read the Wikipedia page for "Leaky Abstraction" for the general principles and my older article "The Leaky View" for a more specific example.



Keeping Android-Free code free of Android!

Android SDK uses my Android-free MVC triads (Dependency Inversion Principle, Hollywood Principle) and my Android-free MVC triads use the Android libraries hidden behind appropriate interfaces (Dependency Inversion Principle, Open/Closed Principle, Single Responsibility Principle). This way, the Android-free code never uses Android-aware code directly: in the worst case, an Android-aware integration point connects Android-free code to an Android-aware implementation of an Android-free interface. If you prefer the Ports and Adapters view[4], this diagram shows the relationships.

[4]Read Alistair Cockburn's "Hexagonal Architecture, The Pattern: Ports and Adapters", The C2.com Wiki's "Ports and Adapters Architecture" and Tony Baker's "Improve Your Software Architecture with Ports and Adapters"



A Ports-and-Adapters or Hexagonal Architure View

I now think of the MVC triads as the primary unit of development, each corresponding roughly to a tiny, independent, cohesive set of user gestures (often just one), and of Android Activities as ways to arrange these MVC triads into user-friendly combinations that flow well between each other. I can test this notion by building an MVC triad entirely ignoring the Android Activity, then wiring it into an Activity just before deploying the feature. This would force me to write the MVC triads as cohesive, independent, stateless services that an Android app could wire together in any desired UI flow. This would allow me, for example, to move a feature from one screen to another as the user interface becomes crowded, or as I learn how to otherwise improve the look and feel of the app. This style of development would allow discourage me from letting view details leak into the Android-free module, a common source of problems in almost every development project I've seen in the past decade. Before I move on, readers have mentioned Document Fragments to me, and I've intentionally not explored them yet. If I reach a point where I am essentially re-implementing them, then I feel more confident that they will help my design, rather than hinder it. I used this same approach ten years ago to decide which of the many competing Java web frameworks worked best with a test-driven, evolutionary design approach. I chose Spring Web-MVC at the time, primarily because it required controllers implement an interface rather than inherit implementation (subclass),

which made it much less likely that an MVC triad would depend unintentionally and inappropriately on the Spring framework.

Now I think we can move on to the next feature. As before, I'd like to start with some simple Product Sashimi work, exploring the feature in detail, trying to identify the kernel of it, then implementing that kernel quickly and cleanly. I'll do something a bit controversial: I'll throw away all the items on the backlog. If I find any of them particularly important, then I'll put them back on the backlog, but I don't want those tasks hanging around my neck weighing me down. If this makes you uncomfortable, then I invite you to feel the fear and do it anyway.[5]

[5]Dr. Susan Jeffers, Feel the Fearâ€¦ and Do It Anyway

18 The Next Feature

Now I'd like to explore a candidate for the next feature to build. I could easily justify simply moving on to building a CRUD-style "new transaction" feature, but I'd really much rather build something that I, and my users, would find more valuable. To do this, I return to the one of the core reasons to build this app at all: to avoid misplacing cash transactions, which I find very easy to lose track of. You know the kind: you spend $2 on a quick coffee or give $4 to a food vendor on the street. These casual, usually cash, transactions have a habit of falling through the cracks and as a result, we can't really know by how much we're underestimating our cash spending in a year. It could be hundreds or thousands of dollars.

I don't need to paint such a dramatic picture. Even if I remember every cash transaction with perfect accuracy, I have to keep this information in my head until I have the chance to write it down. I used to carry a paper notebook with me for just this purpose, but you wouldn't buy a book on how to track cash spending in a paper notebook! As part of writing this book, I want to experiment with using a mobile app to track these easy-to-lose transactions, to compare the experience with carrying around a paper notebook and pen, or keeping this stuff in my head for between minutes and hours before I can enter them on some kind of computer. Accordingly, I need to balance the strengths of a computer-based system with the strengths of a paper-based system.

With a paper-based system, I can quickly write my transaction information down. I can use any shorthand I want to speed things up even more. I can write the information on any piece of paper I have with me. The overwhelming benefit of the paper system is **how quickly I can record the transaction**.

With a computer-based system, I don't have to worry about losing little scraps of paper with important cash transactions written on them. Even when I don't lose the paper, I have to transcribe the information into a spreadsheet. This makes it all too easy to let recorded-but-unfiled transactions pile up, creating a very tedious data entry task that I will probably put off for a long time. I feel better when I can process a transaction completely, then forget about it. The benefits of the computer system are **durability** and **immediacy**.

So I want a feature that lets me record transactions quickly, easily, and completely. Building a general-purpose data-entry form with a handful of fields that forces me to type words on a mobile device virtual keyboard creates tedious work that I will resent. I can picture this vividly from taking daily taxis to and from a client, paying exactly $12 each way, and having to record that over 120 times, matching each trip to the correpsonding receipt in an expense report. I put up with this tedium only because my client reimbursed me for each of those trips, but do I trust myself to treat my own personal expenses with such diligence? Probably not. I want this app to help me with these transactions, not become another bit of annoying software to deal with.

I had this vague notion of a "template" for a new transaction, which remembers some key information about frequent or recurring transactions. It would pre-fill some of the fields for me, like

the category, the vendor, or the amount. Before building a general-purpose "expenditure template" feature – which admittedly sounds like a great feature to put in the architecture document of some enterprise project – I wanted to explore how I'd use it to see how well it would do the job. I started by writing a scenario in which I'd use the feature, which looks like a use case scenario or acceptance test.

The general-purpose scenario

My wife and I buy most of our groceries weekly at the farmers' market. When we visit the market on Saturday mornings, I walk around the market, buy groceries from different vendors, keep a running total of how much we're spending, then capture that with Evernote. I don't mind keeping track of the total in my head while walking around the market, but I hate trying to remember that number while walking back home. I'd like to get that out of my head as soon as I'm ready to leave the market. When I enter this information into Evernote, there's no structure to the information. It behaves like an electronic version of little slips of paper. On the other hand, if I only had a general-purpose data-entry form for a new transaction, then the scenario would look like this:

- Joe says, "I just bought $119.60 worth of Groceries at Summerside Farmers' Market using cash"
- Joe sees, "Got it! You spent $119.60 in cash on Groceries at Summerside Farmers' Market on May 16, 2013 at 10:31"
- …given that the current time is 10:31 on May 16, 2013

If you've seen BDD scenarios before, then this will look familiar, but oddly different. I didn't write this in the given-when-then format, although you can easily identify the steps as "when, then, given". I find that when I explore new scenarios, I focus on the *when* and *then*, and later notice all the *given*s, including one or two that I hadn't expected. If you want to rearrange these points into the given-when-then order, then feel free to do that.

You'll also notice the strange "Joe says" and "Joe sees" format. I use this to hide user interaction details from my scenarios. I admit that it looks gimmicky and weird, and I think it works well for precisely that reason. It encourages me to think about the feature differently. It discourages me from locking myself in to a specific user interface design. It seems to encourage others the same way, and it opens up two key possibilities: perhaps we can build more value with a simpler user interface, and perhaps we can imagine much more valuable features than what a typical user interface would support. One lowers the cost to build, the other finds potentially higher value to deliver. Everyone wins.

By the way, this style of writing a scenario is nothing new. Remember *use cases*? Thoughtful proponents of this technique, like Ivar Jacobson and Alistair Cockburn, encourage writing use case scenarios without describing the detailed mechanics of the interactions between the user and the system. Just because "we're agile" doesn't mean that we turn our backs on ideas found in books whose titles don't include our buzzwords!

In spite of this desire to avoid user interaction details, I do need to consider the user interface, because in this case, it represents a significant part of the value of the feature. That the user has to type in “Groceries” and “Summerside Farmers’ Market” makes this feature slow and tedious to use, and I want to minimise that. For this reason, some interaction details will need to creep in to other scenarios. I will have to work hard to avoid making references to user interface controls.

Remember What I Buy Here

In the case of the Summerside Farmers’ Market, I mostly buy groceries there, so I would find it convenient not to have to say “on groceries” when I say “at Summerside Farmers’ Market”. This leads me to a new scenario.

- Given that the system already knows that I buy Groceries at Summerside Farmers’ Market
- Joe says, “I just spent $86 at Summerside Farmers’ Market”
- Joe sees, “Got it! $86 at Summerside Farmers’ Market for Groceries”

In this scenario, I didn’t bother to include the timestamp, so I’d better make a note to write at least one scenario that focuses on checking the timestamp for a new transaction. To simplify things, I’ll let the system timestamp all transactions as “now”, meaning when I enter them. I don’t want to duplicate this detail in all my scenarios, not only because of the tedium, but also to avoid having to change them all in the future.

I completely forgot about the method of payment: cash. This makes sense to me, given that tracking easy-to-lose cash transactions forms the motivation for this entire feature. Since none of these scenarios treat the method of payment as a significant detail, I’ll just note that all scenarios assume the payment method of “cash”. I will, however, need to write at least one scenario that focuses on what changes when I pay with a card.

> Paying with a card almost always means a receipt, so perhaps I can start by photographing the receipt, then adding details afterwards. I’ll just note that for now, then come back to that later.

In writing this scenario, another improvement struck me.

Don’t Make Me Type

Get your phone. Type “Summerside Farmers’ Market”. Do it a few times. Could you type it the exact same way each time? I couldn’t. Since I visit the farmers’ market almost every week, it’ll only take a few visits before typing that phrase correctly and consistently becomes the bottleneck. It

sounds crazy, but you know it will happen. What if I could abbreviate "Summerside Farmers' Market" somehow? I'll use the metaphor of a big button marked "SFM", although I don't have to implement that as a user interface button control. I'm treading dangerous ground here, as user interface details have started creeping into my thoughts here. I'll just have to keep an eye (a brain?) on it. I could write that scenario like this:

- Given that the system already knows that "SFM" means the vendor "Summerside Farmers' Market" and the category "Groceries"
- Joe says, "I just spent $86 at 'SFM'"
- Joe sees, "Got it! $86 at Summerside Farmers' Market for Groceries"

Just as a quick check, let me visualise the user interaction here, to judge whether I'll find it as easy as it sounds.

1. Press the [SFM] button.
2. Enter 86 on the virtual numeric keypad.
3. Press some kind of OK button.

Yes. That sounds great. I feel comfortable whipping out my phone at the market just before leaving, then pressing only a few buttons to get our groceries spending out of my head. It sounds fantastic, but what about other conveniences?

The Cost Remains the Same

We have a wonderful taxi system where I live: the taxis don't use meters, but rather have "zoned" fares, like some metro systems do. Within the city limits, they charge a fixed price, and outside the city limits, there's a price chart. This means that when we take a taxi to run errands, we almost always pay exactly the same amount, including the tip. Right now, that amount is $7, so when I take a taxi back from the bank, I'd like to simply say "I took a taxi" and let the system remember that a taxi costs $7 cash. Wait… that sounded like a scenario to me.

- Given that the system remembers that a Taxi Trip means paying $7 to Courtesy Cab Company.
- Joe says, "I just took a 'Taxi Trip'"
- Joe sees, "Got it! $7 to Courtesy Cab Company for Taxi Service".

Now sometimes I want to explain why I needed a taxi.

- Given that the system remembers that a Taxi Trip means paying $7 to Courtesy Cab Company.
- Joe says, "I just took a 'Taxi Trip' to shop for household supplies"

- Joe sees, “Got it! $7 to Courtesy Cab Company for Taxi Service in order to ‘shop for household supplies’”.

This implies some way of entering free-form text related to the purpose of the transaction. I should now augment my general-purpose scenario to include a comment of some kind, so let’s just assume that I’ve done that.

This Time, It’s Different

We buy coffee beans at the farmers’ market, and I like to track our expenditures on coffee beans separately from other groceries. (Don’t ask why.) This means that not all transactions at Summerside Farmers’ Market are necessarily for Groceries. Similarly, sometimes we ask a taxi to deliver huge bags of cat food directly from the pet supply shop to our house, and when we do, we tip the driver more for the convenience of not putting on pants. This means that not all transactions for Taxi Service are $7. I don’t want to have to go back to the general-purpose data-entry form to deal with these exceptional-but-still-common occurrences.

I can envision implementing this by pressing the “SFM” button, then popping up the data-entry form with “Summerside Farmers’ Market” and “Groceries” filled in, but letting the user change those fields as much as desired, but I don’t want to lock myself in to that design. Some user experience architect wizard might find a better way, so I write the scenario more abstractly.

- Given that the system remembers that I usually buy Groceries at Summerside Farmers’ Market, known as “SFM”
- Joe says, “I spent $30 at SFM, but on Coffee Beans”
- Joe sees, “Got it! $30 at Summerside Farmers’ Market for Coffee Beans”

The “but” in Joe’s command implies overriding the details that the system remembers for me, while not committing to any particular user interaction. (I come back to this question towards the end of this exploration.) Of course, I could just tell the system to also remember “SFMCB” meaning “Coffee Beans at Summerside Farmers’ Market”, but I wouldn’t want to force the user to choose between not creating a template and creating 12 templates for different categories with the same vendor. I’d like to do this more flexibly.

Multiple Categories/One Vendor

Wow, this section title sounds like a programmer wrote it. I need to crawl back into the user’s headspace immediately.

We often buy both groceries and coffee beans in the same trip to the farmers’ market. Now if I have already implemented the “SFM” preset, then I can simply enter two transactions. It would annoy me, but I could live with it. Even so, I could do better, so let me try.

- Given that the system remembers that "SFM" means "Groceries at Summerside Farmers' Market"
- Joe says, "I spent $210 at 'SFM', but on a few different things."
  - Joe says, "$30 on Coffee Beans. $180 on Groceries. That's it."
- Joe sees, "Got it! $30 at Summerside Farmers' Market on Coffee Beans, and $180 at Summerside Farmers' Market on Groceries."

Sometimes I want to label these transactions as "on the same receipt", such as when I buy both household supplies and office supplies on the same trip to Staples. In that case, I want to add that detail.

- Given that the system remembers that "SFM" means "Groceries at Summerside Farmers' Market"
- Joe says, "I spent $210 at 'SFM', but on a few different things."
  - Joe says, "$30 on Coffee Beans. $180 on Groceries. That's it."
- Joe sees, "Got it! $30 at Summerside Farmers' Market on Coffee Beans, and $180 at Summerside Farmers' Market on Groceries. All this on receipt XYZ123."
- …given that the system generates a "next receipt" ID of "XYZ123".

I'm not going to go back to all my previous scenarios and add a dummy receipt ID. In all those cases, we can assume that the receipt ID doesn't matter, because I don't check it. If in another scenario, I need to check the receipt ID expressly, then I will.

Let's Go Shopping!

I worked through these scenarios with my wife, Sarah, who hit upon an idea that I really liked, but even she – not a programmer – proposed it the way a programmer would think of it. She mused, "What about going into a kind of *mode*? You put the app in *Farmers' Market mode*, then you enter a few transactions, all marked *Farmers' Market*, then you turn the mode off." Even ostensibly non-technical people can jump to solutions. To test this idea, we struggled for a few minutes to write a scenario that took advantage of this behavior, and then it came to us.

- Joe says, "Assume that everything I buy falls under Souvenirs"
- Joe says, "I just spent $20 at Avonlea Book Shop"
- Joe sees, "Got it! $20 at Avonlea Book Shop for Souvenirs"
- Joe says, "I just spent $150 at Island Gifts"
- Joe sees, "Got it! $150 at Island Gifts for Souvenirs"
- Joe says, "I'm done shopping for Souvenirs"
- Joe says, "I just spent $75 at Shipyard Restaurant"
- Joe sees, "Got it! $75 at Shipyard Restaurant for Unknown Category"

In the process of writing this scenario, I realised that I had quietly decided that I'd rather allow the user to enter incomplete information and complete it later, than force the user to specify a category.[1] I intend to use this feature by collecting cash transactions in CSV values, then import them into a spreadsheet, where I can manipulate the data any way I need. I don't want to turn this modest app into an industrial-strength personal finance analysis tool. This makes me wonder which information, if any, I want to make mandatory. I decide to go the easiest route: as long as the user specifies at least one part of the transaction, I don't care which, the app will accept it. We can write scenarios for that in a table, which I suppose you can already picture in your head, so I won't bother writing them out in full here.

I find this idea of tolerating incomplete data interesting. What kinds of incomplete data could become troublesome? What if I need to split an expenditure into multiple categories, but don't want to do it now? (Of course, in order to do this, I'd have to capture at least a receipt to which I could refer later, and even though I'm working on receipt-less transactions now, I don't want to lose the idea, so I write a simple scenario.)

- Joe says, "Here's a picture of a receipt. I need to split this into multiple categories, later."
- Joe sees, "Got it! Receipt XYZ123 which Needs Splitting"
- …given that the system generates the receipt ID "XYZ123"

I've capitalised "Needs Splitting" to emphasise that it's a term with special meaning to the app. I might put the words "Needs splitting" in the comments field; I might create a new yes/no field; I haven't decided, and I don't have to decide just yet.

The Power of Modes

Sarah's idea changed my conception of at least one old scenario. Instead of splitting a $210 expenditure at the Farmers' Market, I could think of it this way:

- Given that the system remembers that "SFM" means "Summerside Farmers' Market"
- Joe says, "I'm shopping at SFM until further notice"
- Joe says, "$30 on Coffee Beans"
- Joe sees, "Got it! $30 at Summerside Farmers' Market for Coffee Beans"
- Joe says, "$180 on Groceries"
- Joe sees, "Got it! $180 at Summerside Farmers' Market for Groceries"
- Joe says, "I'm done shopping at SFM"
- Joe says, "$150 on Groceries"

[1] Among the many reasons that I use scenarios to explore features: writing them out in detail brings unstated and potentially troubling assumptions to the surface. When I work with others, this technique pushes those assumptions into the open, where I often learn that other people assumed the exact opposite. It can annoy us a lot to deal with those questions in the moment, but I prefer that to a rude surprise when I try to deploy a finished feature.

- Joe sees, “Got it! $150 at Unknown Vendor for Groceries”

Now that I think of it more, this *modes* idea together with the one-click presets becomes a powerful system. It seems like it might handle every case that I care about.

Industrial-Strength Scenarios

Now I think about complicated, real-life scenarios, and wonder how these elements would fit together to implement those scenarios. For example, we don’t get all our groceries at the farmers’ market; sometimes we have to shop at the chain grocery store about 3 km from our house. For that, we usually take a taxi.

- Joe says, “[Round Trip Taxi]”
- Joe says, “$137.48 at [Superstore]”
- Joe sees, “Got it! $14 at Courtesy Cab Company for Taxi paid by cash. $137.48 at Superstore for Groceries paid by Mastercard.”
- …given that the system remembers these presets:
  - [Round Trip Taxi] means $14 at Courtesy Cab Company for Taxi paid by cash
  - [Superstore] means Superstore for Groceries paid by Mastercard

This scenario even involves methods of payment, so I included that detail, even though I don’t intend to implement it yet. I’ll stick to cash transactions for now, but I like how easily these scenario elements appear to extend to handling different methods of payment. I don’t foresee having to make any fundamental design changes to support it.

A Quick Look at the Experience

Before forging ahead and building this feature, I want to think a little about how the user interactions. Again, I don’t want to lock myself into a particular user interface, but Alan Cooper will kill me if after all these scenarios I end up with a typically horrific user interface.

Let’s return to the scenario in which I buy $30 worth of coffee beans at the farmers’ market, where I usually only buy groceries.

Given a preset labeled “SFM” that means “Groceries at Summerside Farmers’ Market” –

1. Press [SFM].
2. Enter 30 on the keypad.
3. Press [OK].

> Joe sees, "$30 at Summerside Farmers' Market for Groceries".

For example, the bottom half of the screen shows the most-recently-entered transactions, so this new transaction moves to the top of that list. It's just one way to quickly review what I've entered for basic correctness.

What about getting rid of that last step? Perhaps the preset "SFM" can assume that we only need to add an amount. If the user needs to change some the default "Groceries" to "Coffee Beans", then tapping the newly-created transaction could open the general-purpose data-entry form, where the user could change any of the information. Remember the goal of speed of data entry. We can certainly experiment with this idea when we implement the View for this feature.

What We've Learned

In exploring this feature, Sarah and I settled on a few things.

- Preset buttons, which pre-fill (or auto-suggest) some of the information about a transaction.
- One-click presets, which complete a transaction as quickly as possible, such as "120, [SFM]" meaning "$120 at Summerside Farmers' Market for Groceries".
- Modal presets, which the user *turns on* for a number of transactions, then *turns off* when finished, to use for example on a multi-store trip to buy various items all for Kitchen Renovation Project.
- Auto-suggest the timestamp as *now* and the payment method as *cash*.

We still have some questions.

- Force the user to press OK? or assume the transaction is complete as soon as possible and force the user to re-open it to edit it?
- Force the user to enter at least one field, or allow a completely empty transaction?

We can't resist making some tentative decisions about the user interactions.

- Short press for regular preset; long press to toggle the *mode*.
- Enable the numeric keypad to enter the transaction amount, and not the regular keyboard. (Android 4's Swype keyboard put the decimal point on a different keyboard layer from the digits!)
- For now, list the transactions in reverse order on the main screen, either ordered by most recently created or most recently changed.

I feel like I have enough to start building the feature. I get the sense that I'll need to start building the feature in order to answer the questions we've raised here, anyway. I'll have to write more comprehensive scenarios than these, but I don't want to fill the entire book with them, so I'll leave that for you to see in the project repository at your leisure.

(In case you're wondering about how the user will create these presets, don't. We'll do that later. I can hardcode a handful of presets for myself, then try using them, and if I don't find them useful, then probably nobody else will.)

Where Do We Start?

I know of two ways to approach this question. One approach suggests building the general-purpose data-entry form, because then at least the user can do everything with it, even if the user hates doing any of it. The other approach suggests building the most frequently-used scenario, whichever it is, because we want the user to start using it soon, giving us feedback, and generating ideas about how to improve it before we've built too much.

I don't see a clear answer, in general. I think it depends on the situation. That does, however, leave the question, *where shall **I** start **today** *?

Indeed.

Configuring IDEA For Your Protection

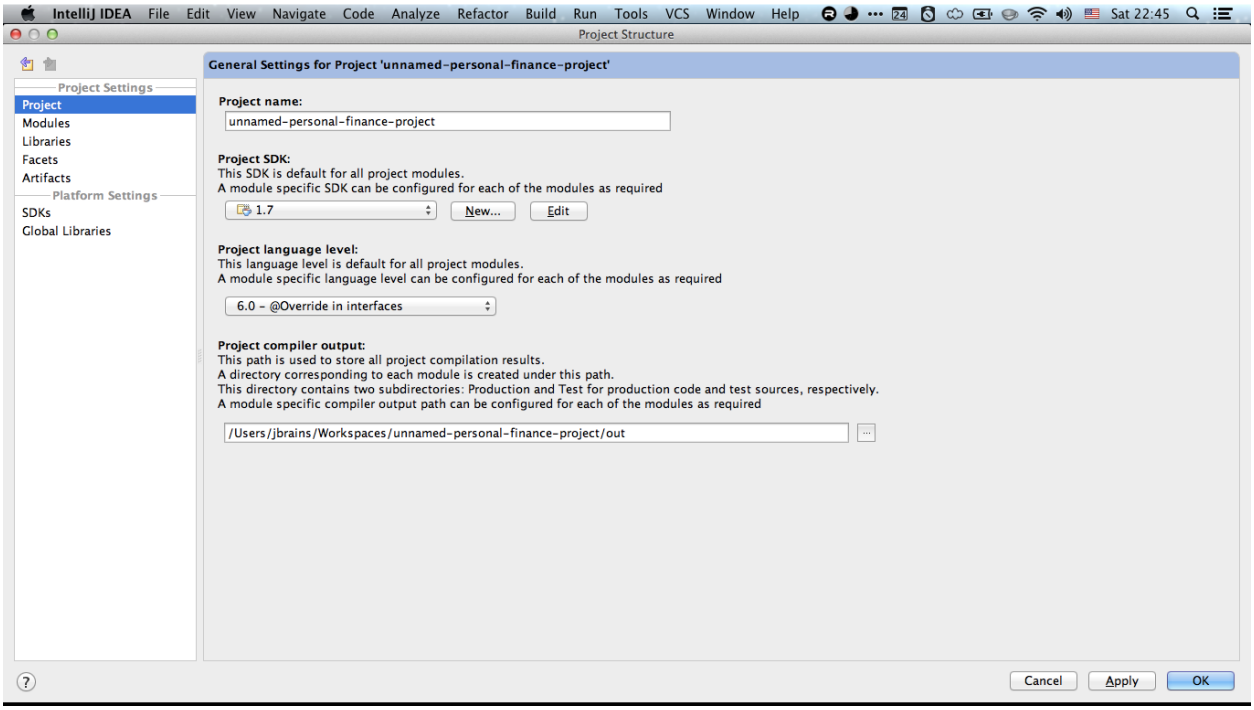
Maybe you haven't used IDEA before. Maybe you want to use Eclipse or NetBeans or vim or something. No matter the IDE, I want to avoid cyclic dependencies within modules. In Eclipse, for example, I'd have an Android project–which I imagine Eclipse would load up with all the cool Android tools for debugging and packaging–and an Android-Free project. By making them separate projects, I could ensure that Android-Free never depends on Android, which would avoid wasting buckets of time at some inopportune point in the future. Keeping module-level dependencies in a DAG[2] helps avoid much of the pain you find in typical legacy code.

I won't use Eclipse here, because I want to learn something new, and IDEA no longer costs $500. With IDEA, a *project* contains *modules* which I can arrange in a dependency graph. This means that I want a project with two modules that correspond to the (Eclipse) projects I mentioned above. Let me show you how I have configured my project.

Project Settings

Nothing special here. I include it for completeness.

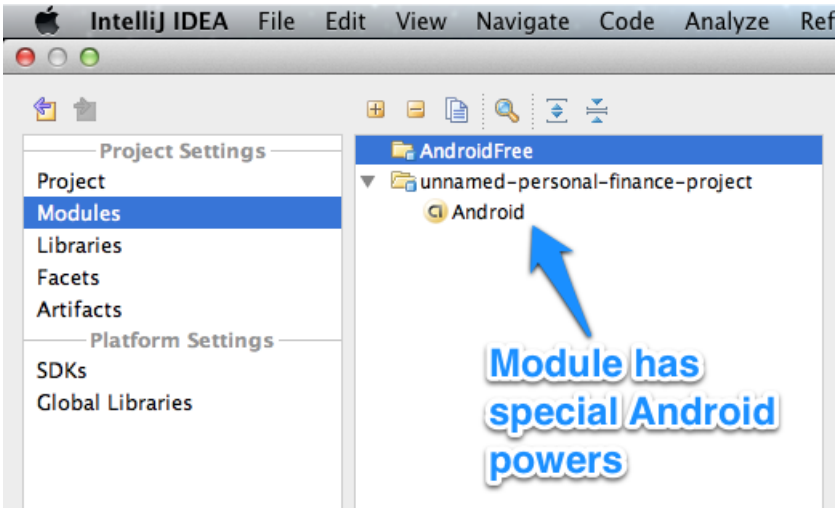
[2] directed, acyclic graph–look it up, if you don't know it



Project Settings

Modules Overview

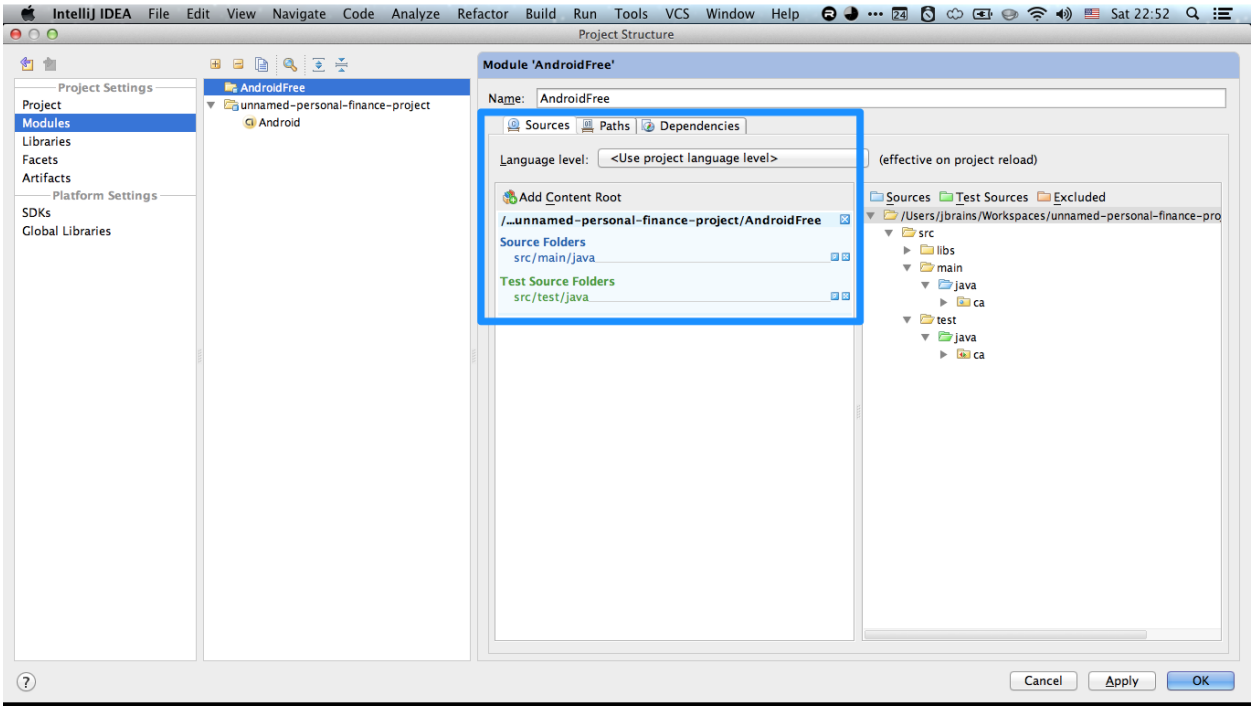
I have two modules: one named “AndroidFree” and one named after the project.



One module has special Android powers; the other not

Android-Free Module Details

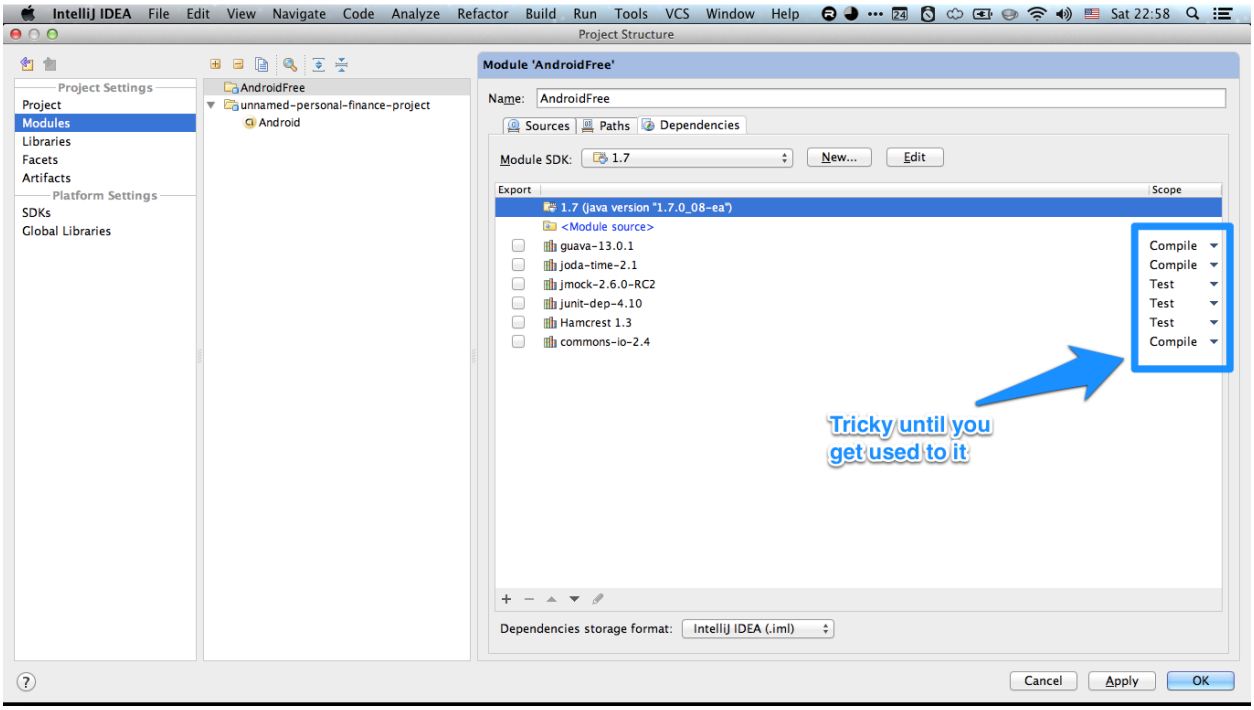
I configure this module like any regular Java module. I decided to use the prevailing conventions, naming them `src/main/java` and `src/test/java`, even though I really don't like them.[3]



IDEA magically (and sensibly) lets tests depend on production, but not the reverse

Since this module acts as a "leaf" in the tree, it only depends on the JDK and other libraries, but no other module.

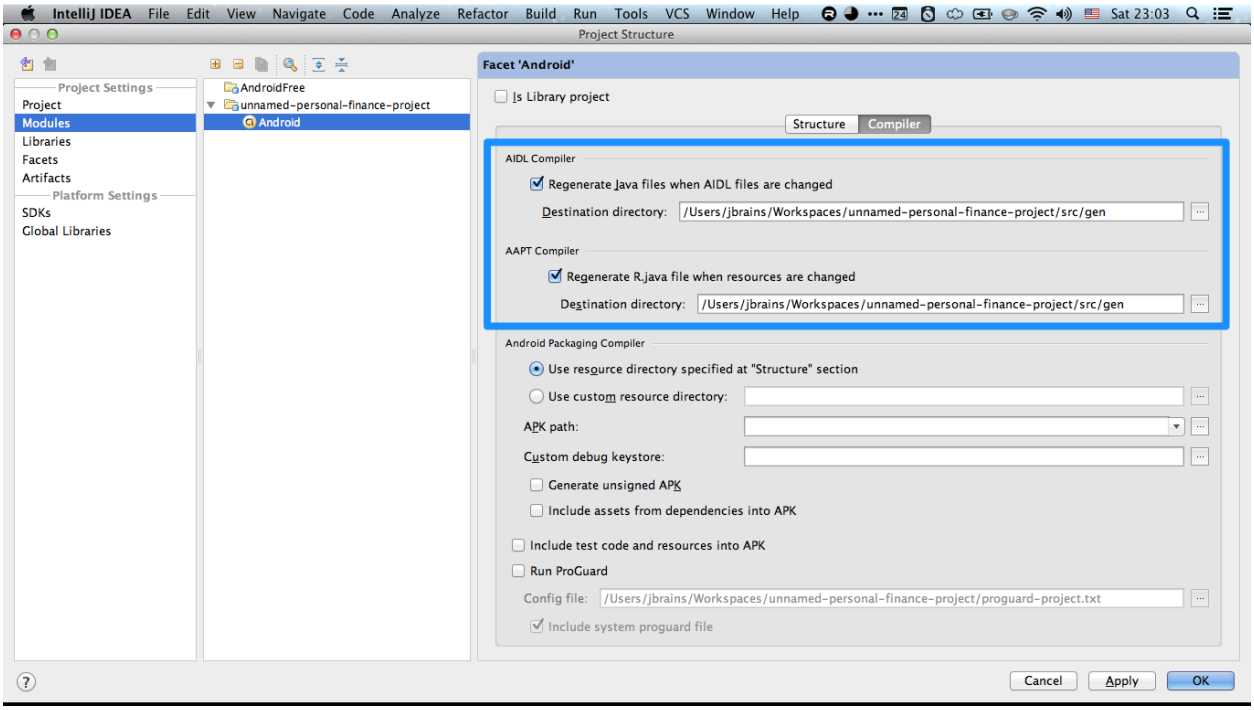
[3] I prefer to name the two source trees `production/source` and `test/source` because, you know, those words mean something in English.



Libraries start in “test scope”, then move to “compile scope” as needed

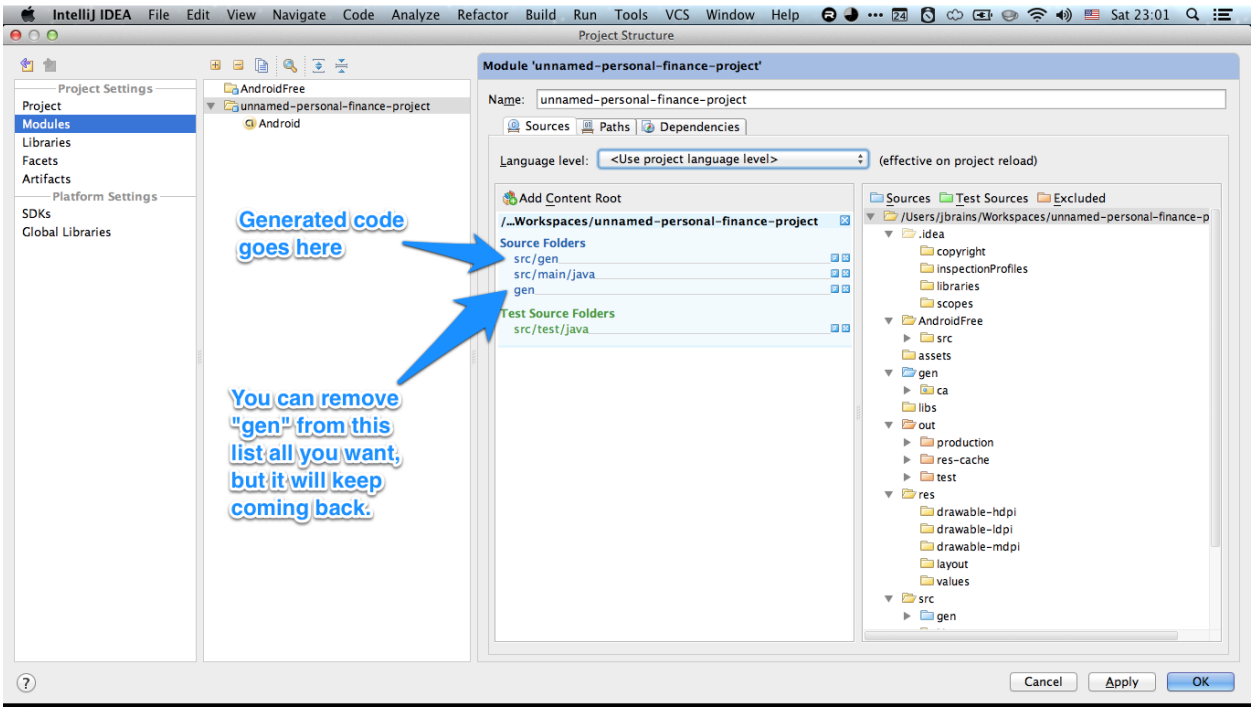
Android Module Details

I prefer Android’s compiler to put the code it generates under `src`, so I configure it that way. Source is source, no?



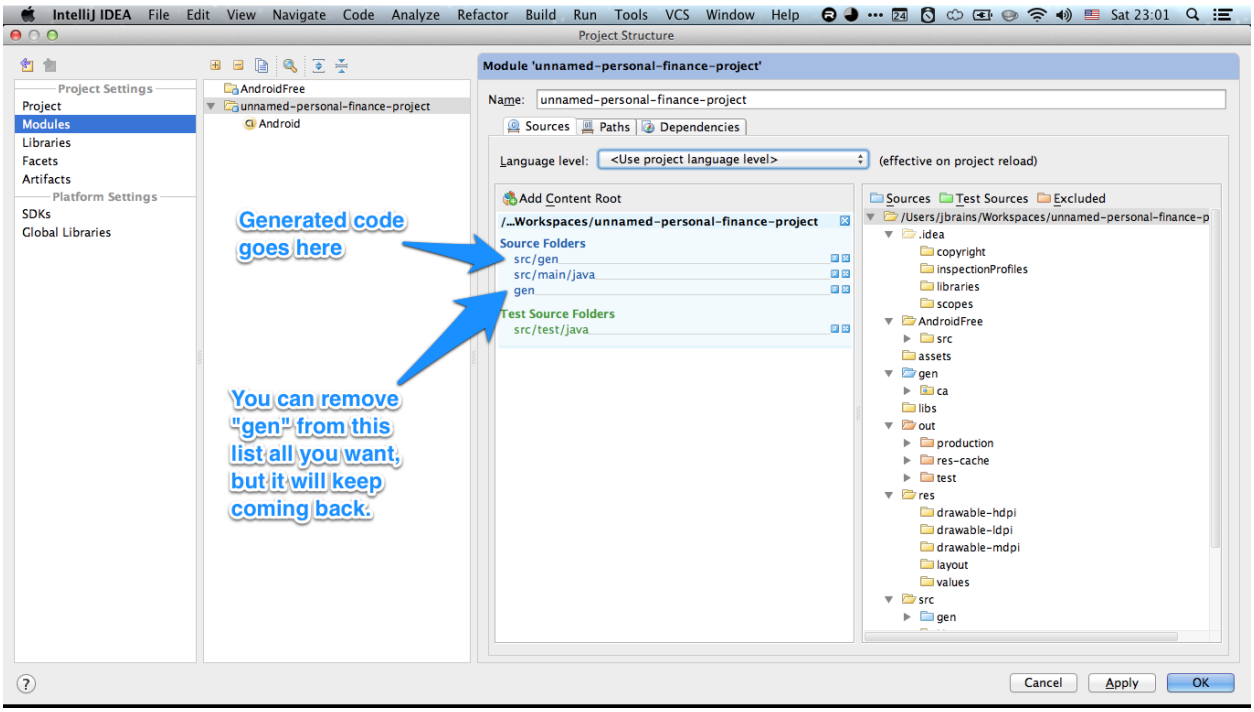
Source is source, no?

This forces me to specify `src/gen` as a source folder.



The first IDEA-related annoyance I've run into

Finally, the Android module depends on the Android-Free module. IDEA magically ensures that Android's tests can use Android-Free's production code *and* tests, which comes in handy for writing contract tests.



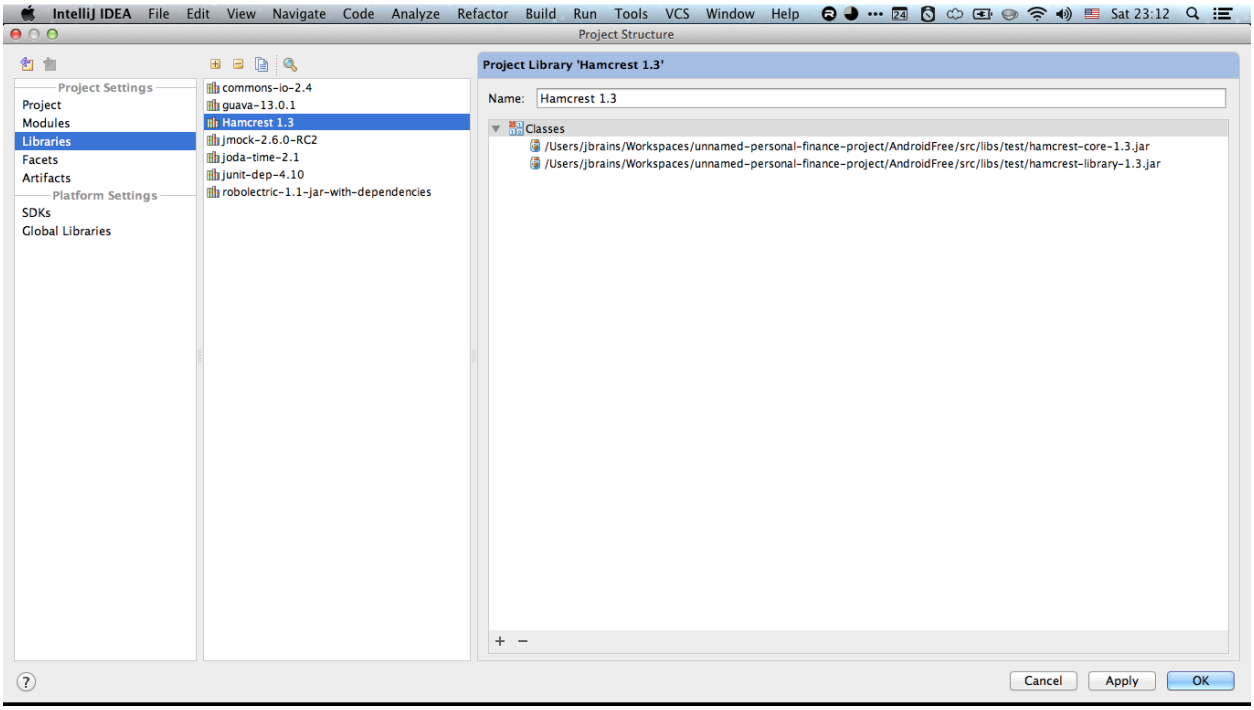
I almost have the hang of this!

Libraries

Make sure you put `<project-path>/.idea`[4] under version control, because if you don't, then you'll run into the same problem that I did: the link from modules to libraries will diverge from the link from libraries to files on disk.

Modules refer to Libraries, and Libraries describe a collection of classes or JARs. Eclipse has the same "library" concept, so I felt right at home adding libraries.

[4] You can safely keep `.idea/workspace.xml` out of version control, but you probably want to stash the rest of the directory in version control.



A library with multiple JARs

Why didn't I use Maven?

It makes me mad.

Go write code!

Go on...

Principles

I refer to a variety of design principles throughout this book. I've summarised them here and referred to additional reading where appropriate.

Four Elements of Simple Design

I describe this one first, because it forms the basis for so much of what follows. (* write more *)

Avoid Inheriting Implementation

We can inherit *implementation* or *interface*. When we extend a class, we (usually) inherit implementation, but when we implement an interface, we inherit (only) interface. I prefer not to inherit implementation, because I find it too easy to accidentally change the contract of my superclass, which violates one of the most fundamental modular design principles that makes inheritance work *at all*, known as the Liskov Substitution Principle.

Informally, this principle states that clients of a type (class or interface) must not know the difference among different implementations or subclasses of that type. You can read the technical details anywhere on the web–Wikipedia provides a good place to start[5]–but when a class violates this principle, then it causes clients to need to behave differently depending on which implementation or subclass they use at runtime. This simply breaks the late-binding polymorphism mechanism that makes objects work. Most often, this dependency problem results in unexpected behavior, which you probably call "bugs".

While interfaces no not eliminate this problem, I find that interfaces divide clients and suppliers more clearly, and this encourages me to consider the contract more carefully when I implement an interface. I have found it all too easy in the past to introduce a subclass and tack on a little extra behavior without regard to the consequences. In particular, I don't have to worry about when to call `super`'s implementation when I override a method. (Before? After? During? Not at all?)

Context Independence

The less a module (function, class, variable) knows about its context, the better. More precisely, minimise the amount of knowledge a module has about code above it in the call stack. The more a block of code knows about its callers, the more it depends on that specific set of callers and the

[5] http://link.jbrains.ca/VxG6H8

less easily one can use it elsewhere. If it seems that object-oriented programming never realised the promise of reusability, you can blame that on billions of lines of context-dependent code. Nowhere have I seen anyone better explain and illustrate this concept than in Steve Freeman and Nat Pryce's tour-de-force, Growing Object-Oriented Software: Guided by Tests

Dependency Inversion

The least-well-understood SOLID principle. Ignore the name for a moment. Concrete things should have the fewest clients possible, in order to avoid implementation details leaking up the call stack and polluting the system. This means that most of the parts of the system should depend on mostly abstract things. In Java, that usually means interfaces. Concrete things can depend on other concrete things, but typically only when those things describe small aspects of a single behavior. Abstract things should absolutely never depend on concrete things. In fact, if you find an abstract thing depending on a concrete thing, you should invert the dependency – thus the origin of the time.

Dependency injection, which some people confuse with Dependency Inversion, provides just one mechanism for avoiding having concrete things depend on other concrete things.

The vast majority of legacy code problems amount to dependencies in the wrong direction. I'll give you a moment to read that again, because most of what I do to tame legacy systems amounts to inverting dependencies.

Fix Problems Before They Become Serious

This sounds prudent, but boring. Let me illustrate with a photo.

She probably won't kill you… yet

If this cat swipes at you with her claws, it will hurt, but you'll live. Do you like your chances of surviving that same swipe from her a year from now? No, but so many programmers like their chances with legacy code, which I just don't understand.

Hide Methods With Interfaces, Not Visibility

Many people disagree with this principle, but I stand by it. I prefer to hide methods behind interfaces, rather than by marking them non-`public`. Marking these methods non-`public` restricts the programmer unnecessarily, whereas hiding the methods behind an interface encourages the programmer to depend on only the interface methods when possible, but allows the programmer access to the implementation details when the situation demands it.

When I've asked designers why they mark methods as `private`, they answer most commonly that they don't trust other programmers to use the object correctly. I find that interfaces provide adequate documentation to guide a programmer to use objects as I've intended. Moreover, I prefer to encourage mindful, not mindless, use of my objects.

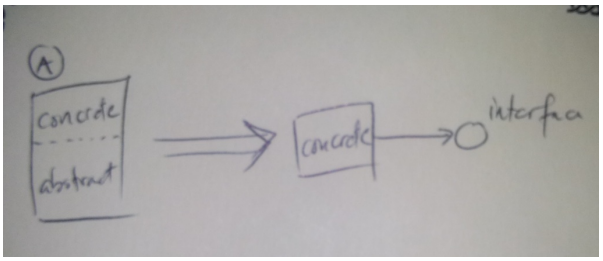
Other designers cite the abstract notion of encapsulation to justify their choice. In this situation, I ask them whether they've ever experienced wanting access to a hidden method in someone else's framework or library. They almost always answer "Yes". The conversation often successfully ends there.

With interfaces I can hide implementation details for safety without making them entirely inaccessible, so I prefer interfaces to method visibility controls.

Interface Segregation

One of the SOLID principles, and a consequence of the notion of designing objects to know as little as possible about their collaborators as possible. I prefer to design objects to depend on smaller, more cohesive interfaces, even when the implementing class has more varied behavior. Different clients might therefore use different "vies" of the same collaborating class, an indication that that class might have too many responsibilities.

Prefer Interfaces to Abstract Classes



Replace abstract class with concrete class and interface

Abstract classes annoy me because testing them requires the Subclass to Test pattern. Using this pattern means testing a potentially subtly different class than the class I intend to test, so I use this pattern with care. I typically remedy this situation by Replacing Inheritance with Delegation, separating the purely concrete part of the class–whose methods all have implementations–from the purely abstract part of the class–whose methods all have no implementation. This results in a concrete class with a reference to an interface, and I usually find the tests for these two objects simpler and more resilient to change than tests for the abstract class from which they'd evolved.

Single Responsibility

One of the SOLID principles, it relates strongly to cohesion. Intuitively, a module (such as an object or function) should have only one reason to change, meaning that if it has multiple parts, those parts tend to change all together or not at all. Violating the Single Responsibility Principle usually results in finding it difficult to find the code you need to change (you've buried the needle in a big haystack) or unintentionally changing something other than what you meant to change (too easy to create unintended dependencies within unrelated-but-neighboring code). We tend to fix violating by separating big things into smaller things, then perhaps rearranging them into more cohesive units.

Use the Weakest Type Possible

One can think of this principle as a special case of the Dependency Inversion Principle or Context Independence principle, or any of a number of other ones. Simply put: declare each variable (or

return value) with the weakest type possible. In this context, *weak* means *most abstract*. If you have a `List`, but don't use any of the special methods specific to a `List`, then declare it a `Collection` instead. Store references to interfaces, rather than a particular implementation, if you can. For each value in your system, refer to it with the type farthest up the inheritance hierarchy that the type-checking system will allow. This provides maximum flexibility to change the code. Sometimes it even alerts me to design problems: if a variable has a very abstract name, but a more concrete type, then perhaps an implementation detail has started leaking up the call stack, and I should fix it before it becomes a serious problem.

Glossary

Abstract Factory One of the standard Design Patterns. Imagine you have a GUI toolkit with two families of widgets, one with the Windows look and feel, and one with the Mac OS look and feel. You don't want to drew a Windows window inside a Mac OS window, but you also don't want clients to care about this distinction, since you have all these nice widget interfaces. You introduce an Abstract Factory that creates each kind of widget (panel, button, text field), subclassed by a `WindowsLookAndFeelWidgetFactory` that creates Windows widgets and a `MacOsLookAndFeelWidgetFactory` that creates Mac OS widgets. These both implement the `WidgetFactory` interface, which makes the factory abstract.

Chunnel Problem When I design two parts of a system, working towards each other, but then find that they don't quite fit together, I call this a *Chunnel Problem*. This refers to the way the two ends of the Chunnel (between England and France) didn't quite line up during construction, forcing the respective crews to adjust each end of the tunnel so that they would connect when they met. (Even if this didn't actually happen, it makes for a great image.)

Composite One of the standard Design Patterns. Think of files and folders on the file system. Some tools, like `cat`, care about the difference. Other tools, like `ls` don't. If files and folders implement a common interface, then tools like `ls` don't have to know whether something is a file or a folder – the fact that it has a name, permissions, and size gives it enough to do its job. When the "whole" and the "part" implement a common interface, we call them collectively a "composite".

Contract I usually refer to contracts for *interfaces*, rather than implementations. By "contract" I mean the set of methods, pre-conditions, post-conditions and invariants. The contract of an interface defines the *only* behavior upon which clients can safely depend. I write Contract Tests to specify contracts by example.

Contract Test A test for behavior on an interface, rather than a specific implementation. All implementations of the interface must pass its contract tests in order to safely replace one implementation of the interface with another[6]. I use Contract Tests extensively to specify and check the collaboration between neighboring objects, by which I can avoid relying on expensive, brittle integrated tests.

Crash Test Dummy When I stub a method to intentionally throw an exception, I call that a "crash test dummy", which term I learned from Kent Beck in Test-Driven Development: by Example, chapter 27.

[6]We know this statement as the Liskov Substitution Principle, one of the SOLID principles.

Laugh Test If I can consider an option without laughing out loud at the thought of it, then I say that it has "passed the Laugh Test".

Narrowing API When I need only a small part of a class, and especially if I want to be able to stub or set expectations on it, I apply the Interface Segregation Principle and extract an interface with only the parts that I need. Since this presents only a subset of the class's methods, it "narrows" the interface of the class to only the parts I need, hence "narrowing API".

Subclass to Test Pattern Sometimes the subject of a test doesn't provide direct access to the thing that I want to check. Sometimes creating a testable subclass of the subject allows me to gain access to that thing so that I can invoke it or check it. This allows me to write the test without changing the subject, but it runs the risk of testing something subtly–and perhaps some day significantly–different from the subject. This pattern plays a central role in rescuing legacy systems. When I apply this pattern, I usually intend to follow it with Replace Inheritance with Delegation. Read more at https://c2.com/cgi/wiki?SubclassToTest[7] and https://c2.com/cgi/wiki?SubclassToTestAntiPattern[8] as well as Working Effectively with Legacy Code[9].

Walking Skeleton One of the names for the first working version of a product. In building a Walking Skeleton, I want to verify some basic assumptions I've made about the technology I've chosen, deliver a working and nominally useful first feature, and, if I do it well, filling in the next several features amounts to "fleshing out" the skeleton. Different people call this idea by different names, and they all mean slightly different things. Whether you call it Walking Skeleton, Minimum Viable Product (MVP), Architectural Spike or something else, these all have similar core goals. I refer to this also as the "kernel" of the product: the part we build first upon which we can build most of the rest in virtally any sequence we like.

[7] https://c2.com/cgi/wiki?SubclassToTest
[8] https://c2.com/cgi/wiki?SubclassToTestAntiPattern
[9] http://link.jbrains.ca/U9NRlv

Additional References

Beck, Kent. *Test-Driven Development: By Example"[10] In spite of its age, I continue to recommend this as a first book on test-driven development to virtually everyone. The exercise of test-driving a testing framework illustrates how to push the envelope of incremental design.

Bolton, Michael, "Testing vs Checking"[11] At Agile 2009 I attended the session at which Michael began using the term "checking" to distinguish from "testing". To quote Michael, "Checking is something that we do with the motivation of confirming existing beliefs. [...] Testing is something that we do with the motivation of finding new information." I find the distinction worth making, although I tend to say "testing" out of habit.

Cockburn, Alistair. *Writing Effective Use Cases*[12] What makes for effective use cases? A lot of the same principles, guidelines and techniques that make for effective user stories and scenarios (or examples, or acceptance tests, whatever you prefer to call them).

Cunningham & Cunningham Wiki http://c2.com/cgi/wiki[13] The original wiki, first inhabited by the Portland Design Patterns group, and eventually hijacked by the early Extreme Programming community. It provides a rich reference in a variety of software-related topics, including in-depth articles and discussions about the finer points of design.

DeMarco, Tom and Tim Lister. *Waltzing With Bears: Managing Risk On Software Projects*[14] "Risk Management is Project Management for adults." If that statement doesn't pique your interest, then don't bother reading this classic work, which explains managing risk in a lively and simple manner. Among their conclusions, you'll find that incremental delivery, starting early and retaining options provide "the ultimate risk mitigation strategy".

Fowler, Martin. *Patterns of Enterprise Application Architecture*[15] As with *Design Patterns*, this book provides a rich vocabulary for programmers to talk simply about design options for their enterprise applications.

Fowler, Martin. *Refactoring: Improving the Design of Existing Code*[16] Highly-skilled, disciplined refactoring lies at the heart of Responsible Design. This book taught me the fundamentals of refactoring; I have practised what I've learned from it since the day I began reading it.

Freeman, Steve and Nat Pryce. *Growing Object-Oriented Software Guided by Tests*[17] Every few years, some very accomplished programmer writes a book that one could title "How I write software". This is Steve and Nat's book. At the centerpiece lies the concept of Context Independence, which gave me a concise way to describe what I'd been teaching for years about design, particularly in languages like Java and C# that declare interfaces.

Gamma, Erich and others. ***Design Patterns: Elements of Reusable Object-Oriented Software***[18] The classic work on the topic of reusable object-oriented design. This book gave me a vocabulary for discussing recurring design elements, as well as "targets" for refactorings. When I find myself refactoring in the direction of a pattern that I know well, I usually interpret that positively; although when I find myself refactoring away from a pattern that I think I'll need, I interpret that even more positively.

Hunt, Andrew and Dave Thomas. ***The Pragmatic Programmer: From Journey to Master***[19] Before I read about Extreme Programming and Test-Driven Development, this book taught me more useful programming tricks than any other. Even if you find Agile practices total nonsense, you'll learn at least ten things of great value from this book.

Jacobson, Ivar. ***Object Oriented Software Engineering: A Use Case Driven Approach***[20] The title doesn't exactly scream "lightweight development", but don't let that fool you. You will see in this book some of the roots of, or forerunners to, the agile-leaning ideas that you might take for granted today.

Keriesvky, Joshua. ***Refactoring to Patterns***[21] In the early days of TDD, we wondered how to combine it with Design Patterns. The very question belies the faulty notion that one ought to pull Design Patterns off the shelf and insert them into our programs. This book has done more to dispel that myth than any other I've read, and I consider it an indispensable follow-on work to Fowler's *Refactoring*.

Matts, Chris and Olav Maassen. ***Commitment***[22] A graphic novel about managing risks with Real Options, which focuses on answering the question, "When *must* we commit?" The authors had not released it when I wrote these words.

---

[10] http://link.jbrains.ca/UtljZc
[11] http://link.jbrains.ca/Yx450w
[12] http://link.jbrains.ca/115Q7Ev
[13] http://c2.com/cgi/wiki
[14] http://link.jbrains.ca/S2jyPY
[15] http://link.jbrains.ca/TEkD2M
[16] http://link.jbrains.ca/UiOGui
[17] http://link.jbrains.ca/10nrSjg
[18] http://link.jbrains.ca/11ATEqK
[19] http://link.jbrains.ca/12UrVlx
[20] http://link.jbrains.ca/1cAD91v
[21] http://link.jbrains.ca/12UpzmI
[22] http://link.jbrains.ca/VzcLy4