TDD for the newb Who Wants to Become an Apprentice

Howard Deiner

Agile Technical Practices and Process Coach

SolutionsIQ, Redmond WA, USA


Agenda

1. Get organized for the day
2. Validate IDE configurations with a TDD for “Hello World”.
3. Description of the day’s project
4. A CreditCard class (credit card number validation)
5. A Merchant class (UK postal code, UK phone, and email validation)
6. What in the world are test doubles?
7. A Merchant wants to go postal (test double via fake)
8. A Merchant actually goes postal (REST WS called)
9. A Customer class (refactor to reuse)
10. An Order class (distance computation)
11. Let’s talk about code coverage
12. Questions, Answers, Debate, Depart


Getting
Organized

A few ground rules:

- Be considerate of others.
- Don't be afraid to say, "I don't understand." No one is expected to understand this material yet.
- If you don't try, you won't learn. I can give you my implementations for code if we need to move on, but the effort you put in is proportional to the benefit you will receive.

This workshop will presume basic Java knowledge and use the basic Eclipse for Java Developers IDE. If you wish to use another IDE, that's fine.

And now, I'd like everyone to find a partner. Pairing works really well in this environment. Try it. You'll like it!

You will find this deck, as well as worked out code samples (but don't cheat during the workshop!) at

`https://github.com/hdeiner/DevWeek2016-TDD-for-the-Newb-Who-Wants-to-Become-an-Apprentice.git`


JUnit
"Hello World"

Write a class called “HelloWorld”.

This class will have a String method called “respondTo”, whichs takes one String parameter.

We want to have an object of the HelloWorld class respondTo “hello” with “world”.

And we want an object of the HelloWorld class respondTo “Bernie” with “Sanders”.

Let’s do this together, writing the code using the TDD 5 Step Cycle:

- Write a test
- Make sure the test fails for the right reason
- Write just enough code to make the test pass
- Make sure that all the tests still pass
- Refactor


What's in
store for
the rest of
the day?

The theme for the day's work revolves around a website that we intend to build.

Said website will allow a Customer to use one of their CreditCards to buy from a Merchant. (I know. A totally unique idea).

We will be constructing the three classes from above.

We will be doing a lot of field validation,and testing that the validations repond properly.

Later in the day, we will be doing some validation on UK postal codes using Web Service APIs provided by https://postcodes.io. We will also be collecting geolocation data.

And finally, our project will calculate distance between Customer and Merchant addresses. Mostly, so we have some more code to develop using TDD.


The
CreditCard
Class

We've all used credit cards for online purchases, right?

You tell the system your numbers, your expiration date, your CVV (card verification value), and off you go!

We're not going to actually be submitting any credit charges today, so we're not going to worry about the expiration date and CVV. But we will be going deeply into validating the credit card's numbers. They can tell us a real story, and we can actually make sure that the numbers given make sense before we ever try to use them!

For example, did you know that all credit card numbers actually have a checksum built into them? It's true! In fact, IBM scientist Hans Peter Luhn was granted a US Patent on his "Luhn Algorithm" back in 1960.

The Luhn Algorithm

- Drop the last digit from the number. The last digit is what we want to check against
- Reverse the numbers
- Multiply the digits in odd positions (1, 3, 5, etc.) by 2 and subtract 9 to all any result higher than 9
- Add all the numbers together
- The check digit (the last number of the card) is the amount that you would need to add to get a multiple of 10 (modulo 10).
- Modulo 10 of the sum of all the numbers plus the checkDigit should equal zero.

Example of Working the Luhn Algorithm

Step																Total
Original Number:	4	5	5	6	7	3	7	5	8	6	8	9	9	8	5	5	
Drop the last digit:	4	5	5	6	7	3	7	5	8	6	8	9	9	8	5		
Reverse the digits:	5	8	9	9	8	6	8	5	7	3	7	6	5	5	4		
Multiple odd digits by 2:	10	8	18	9	16	6	16	5	14	3	14	6	10	5	8		
Subtract 9 to numbers over 9:	1	8	9	9	7	6	7	5	5	3	5	6	1	5	8		
Add all numbers:	1	8	9	9	7	6	7	5	5	3	5	6	1	5	8		85

Mod 10: (85+5) modulo 10 = 0 (5 was the last digit of card)

Whatever else happens today, you've at least found a way to check for transcription errors when entering credit card numbers.

But wait, there's more to know about credit card numbers!

Various Credit Card Issuers Have Different Formats

Credit Card Issuer	Starts With	Length
American Express	34, 37	15
Diners Club - Carte Blanche	300, 301, 302, 303, 304, 305	14
Diners Club - International	36	14
Diners Club - USA & Canada	54	16
Discover	6011, 622126 to 622925, 644, 645, 646, 647, 648, 649, 65	16
InstaPayment	637, 638, 639	16
JCB	3528 to 3589	16
Laser	6304, 6706, 6771, 6709	16-19
Maestro	5018, 5020, 5038, 5893, 6304, 6759, 6761, 6762, 6763	16-19
MasterCard	51, 52, 53, 54, 55	16-19
Visa	4	13-16
Visa Electron	4026, 417500, 4508, 4844, 4913, 4917	16

For final testing, here are valid card numbers. You can make up your own bad numbers.

VISA	**MasterCard**	**American Express**
4485999990287897	5315571812010758	372354590612298
4916526029752812	5148442462496387	379786717424615
4539696491343569	5590456078838330	344545022650393
Discover	**JCB**	**Diners Club - North America**
6011571608164376	3158304464572924	5510050509751175
6011451808735426	3112184521307356	5557142064959709
6011686546895236	3158938116950200	5405084407199367
Diners Club - Carte Blanche	**Diners Club – International**	**Maestro**
30344023257021	36193482992253	6763709843268368
30091117396997	36314264739742	6759233278986184
30478764169001	36104261563793	6304537298512113
LASER	**Visa Electron**	**InstaPayment**
6771485405590909	4508823435239888	6395821636176587
6706323137698726	4175008932920428	6378944580148062
6304542517832193	4917994137916076	6376872922774166

Rules for CreditClass

1. The constructor takes in one String argument – the card number
2. An Exception must be thrown if the Luhn Algorithm shows a bad card number
3. An Exception must be thrown if none of the starts with and length rules match
4. Have the constructor figure out the card issuer as well and keep it as a private attribute.
5. Create a public String method called getCardIssuer, and have it return the card issurer.

Time to get to work!

I'll be roming around answering
questions and helping those who need it.

Don't be afraid to ask for help.
Don't allow yourself to get stuck!


The
Merchant
Class

Now it's time to consider the Merchant class. How hard could that posibly be?

Well, if you're as devious as I am, there could be plenty to do. Remember, our goal for these classes is to make the data that they collect as pure as we can make them by rigoursly keeping the data clean and free from entry flaws.

Merchants have:

- A name
- An address
- A postalcode
- A phone number
- An eMail address

And yes, there will be rules.

Write the Merchant class and specify all of the attributes in the constructor.

For now, any names that come in should be scrubbed to eliminate leading and trailing blanks. No errors, just trimming. And there must be a some name.

Likewise for addresses. We should silently trim away leading and trailing spaces. But so long as there is an address, we're fine. For now.

The other fields are a different story…

Postal Code Validation

According to wikipedia, at https://en.wikipedia.org/wiki/Postcodes_in_the_United_Kingdom, there is a definate format to a UK postalcode.

Format (A=letter,N=digit, mind the space)	Coverage	Example
AA9A 9AA	WC postcode area; EC1–EC4, NW1W, SE1P, SW1	EC1A 1BB
A9A 9AA	E1W, N1C, N1P	W1A 0AX
A9 9AA	B, E, G, L, M, N, S, W	M1 1AE
A99 9AA		B33 8TH
AA9 9AA	All other postcodes	CR2 6XH
AA99 9AA		DN55 1PT

UK Phone Number Validation

- UK phone numbers can be either 10 or 11 digits long (with the exception of some special numbers, but we're not going to worry about them).
- They consist of an area code followed by a local number.
- The area code varies in length between three and five digits, and the local portion of the number takes up the remaining length of the 10 or 11 digits.
- For all practical purposes, no-one ever quotes just the local portion of their number, so you can ignore the distinction now, except for how it affects formatting.
- They start with zero. Possibly with the country cod of +44 in front of it.
- The second digit can be anything. Currently no valid numbers start with 04 or 06, but there's nothing stopping these ranges coming into use in the future. (03 has recently been brought into use).
- They can be formatted with a set of brackets for the area code and with spaces (one or more, in varying positions), but those are all entirely optional.

eMail Validation

The format of email addresses is local-part@domain where the local-part may be up to 64 characters long and the domain name may have a maximum of 255 characters. See https://en.wikipedia.org/wiki/Email_address

Local part

The local-part of the email address may use any of these ASCII characters:

- Uppercase and lowercase Latin letters (A–Z, a–z) and digits 0 to 9
- Specified special characters: !#$%&'*+-/=?^_`{|}~
- Character . (dot, period, full stop), provided that it is not the first or last character, and provided also that it does not appear consecutively (e.g. John..Doe@example.com is not allowed).
- Space and "(),:;<>@[\] characters are allowed with restrictions (they are only allowed inside a quoted string, as described in the paragraph below, and in addition, a backslash or double-quote must be preceded by a backslash).
- Comments are allowed with parentheses at either end of the local part; e.g. john.smith(comment)@example.com and (comment)john.smith@example.com are both equivalent to john.smith@example.com.

eMail Validation - continued

Local part – continued

- A quoted string may exist as a dot separated entity within the local-part, or it may exist when the outermost quotes are the outermost characters of the local-part (e.g., abc."defghi".xyz@example.com or "abcdefghixyz"@example.com are allowed.
- Conversely, abc"defghi"xyz@example.com is not; neither is abc\"def\"ghi@example.com).
- Quoted strings and characters however, are not commonly used.
- RFC 5321 also warns that "a host that expects to receive mail SHOULD avoid defining mailboxes where the Local-part requires (or uses) the Quoted-string form".
- The local-part postmaster is treated specially–it is case-insensitive, and should be forwarded to the domain email administrator.
- Technically all other local-parts are case-sensitive, therefore jsmith@example.com and JSmith@example.com specify different mailboxes; however, many organizations treat uppercase and lowercase letters as equivalent.

eMail Validation - continued

Domain part

- The domain name part of an email address has to conform to strict guidelines: it must match the requirements for a hostname, consisting of letters, digits, hyphens and dots.
- In addition, the domain part may be an IP address literal, surrounded by square brackets, such as jsmith@[192.168.2.1] or jsmith@[IPv6:2001:db8::1], although this is rarely seen except in email spam.
- Comments are allowed in the domain part as well as in the local part; for example, john.smith@(comment)example.com and john.smith@example.com(comment) are equivalent to john.smith@example.com..

eMail Validation – test data for valid eMail addresses

`prettyandsimple@example.com`

`very.common@example.com`

`disposable.style.email.with+symbol@example.com`

`other.email-with-dash@example.com`

eMail Validation – test data for invalid eMail addresses

`Abc.example.com` (no @ character)

`A@b@c@example.com` (only one @ is allowed outside quotation marks)

`a"b(c)d,e:f;g<h>i[j\k]l@example.com` (none of the special characters in this local part are allowed outside quotation marks)

`just"not"right@example.com` (quoted strings must be dot separated or the only element making up the local-part)

`this is"not\allowed@example.com` (spaces, quotes, and backslashes may only exist when within quoted strings and preceded by a backslash)

`john.doe@example..com` (double dot after @)

a valid address with a leading spacea

valid address with a trailing space

Time to get to work!

I'll be roming around answering
questions and helping those who need it.

Don't be afraid to ask for help.
Don't allow yourself to get stuck!


What in the World
Are Test Doubles?

What in the World?

Many classes need collaborators in order to do their work.

When you're unit testing a particular class, you might be able to use its real collaborators, but in some cases that's not convenient.

You can replace the real collaborators with various kinds of "test doubles".

Think of a "test double" as being like a "stunt double".

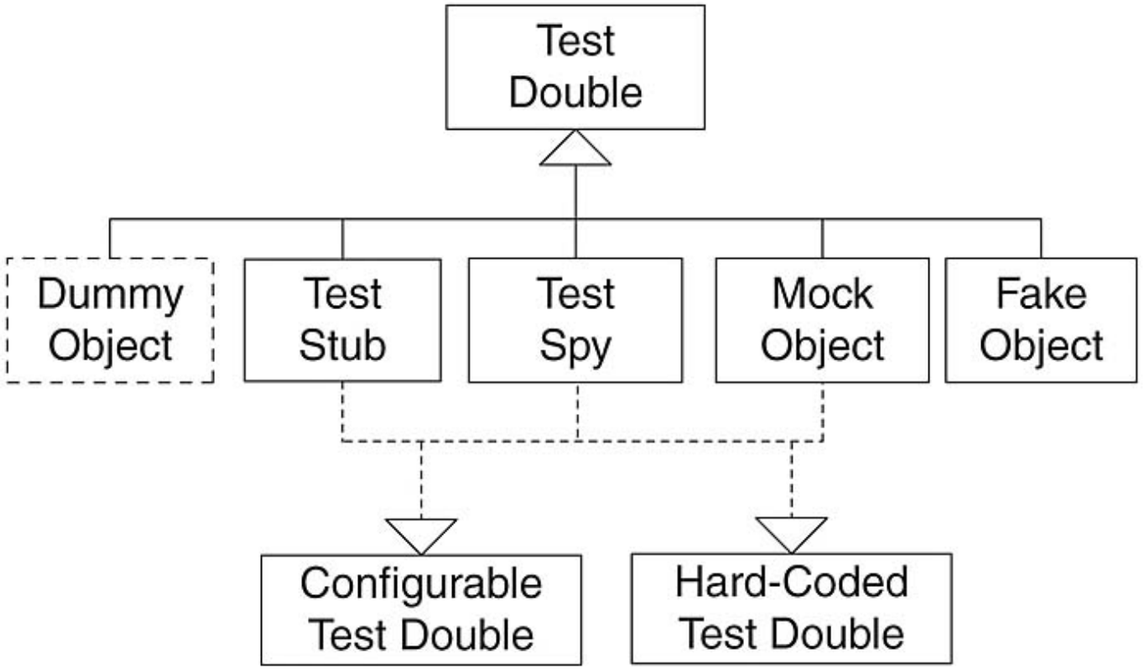
When you're filming a movie you don't want the real actor jumping out of airplanes or jumping out of skyscrapers on to motorcyles in motion, you get someone more expendable!

Gerard Meszaros came up with the umbrella term "test double" in his book "xUnit Test Patterns" to try to clear up the terminology.

Say Again?

Many people say “mock” when they mean “stub” or “fake”, and it can be quite confusing. Here’s the picture. What’s here?

- Dummy objects are used when a parameter is needed for the tested method but without actually needing to use the parameter.
- Test stubs are used for providing the tested code with "indirect input”.
- Test spys are used for verifying "indirect output" of the tested code, by asserting the expectations afterwards, without having defined the expectations before the tested code is executed.
- Mock objects are used for verifying "indirect output" of the tested code, by first defining the expectations before the tested code is executed.
- Fake objecta are simpler implementations, like using an a hashmap for database instead of doing real database access.



Test Doubles Are Really, Really Useful Because They…

- Allow for rapid execution where depended upon code is slow.
- Allow for parallel development of code that is depended upon. Don't plan for the best way to develop depended upon code. Instead, plan on how you can break those dependencies.
- Allow execution of the nooks and crannies in code. How can you test the code that recovers from the power cord being broken?
- Allows for testing of nondeterministic code.
- Makes the development of your code easier, since any issues from the code you are depending on are not present when you are still developing the code.
- This implies that integration goes smoother as well.


A Merchant Wants to Go Postal

Remember how I said before that we can grab UK postal codes using Web Service APIs provided by https://postcodes.io and get geoSpacial data?

Well… We're going to do that. Real soon now. Honest.

But first, let's consider how to go about this.

We have a component that we are about to depend on. And that component can slow us down. We have issues to deal with regarding the speed of the iternet connection to the servers. Latency. Throughput. And all sort of nasties. Potentially.

We want to use this oppurtunity to do the right thing when it comes to testing depended upon components. We want a test double.

First, let's talk about how that test double gets put into the system.

Essentially, we have an iterface to the depended upon component that can be implemented in one of two ways. That's called inheritence.

Our mindset always has to be "depend on interfaces, not implementations."

So, as we go about implementing some GeoLocationUtilities, let's make that class an interface, and implement that class now in GeoLocationUtilitiesFake.

Normally, we would use some sort of Dependency Inversion at this point, such as Spring to "wire" in the implementation of the interface that we want to use. We'll just keep that in our test environment for now.

Let's start with the tests. Always start with the tests. And, I have some tests in mind. I'd like to get the latitudes and longitudes for The Stafford Hotel, Central Hall Westminster, and Oxford University. Something like this:

TestGeoLocationUtilities.java

```
1  package test;
2
3  import static org.junit.Assert.*;
9
10 public class TestGeoLocationUtilities {
11
12     private GeoLocationUtilitiesInterface geoLocationUtilities = new GeoLocationUtilitiesFake();
13
14     @Test
15     public void testLatLongForTheStafford() {
16         double[] geoTheStafford = new double[2];
17         geoTheStafford = geoLocationUtilities.getLatLongFromPostalcode("SW1A 1NJ");
18         assertArrayEquals(new double[] {51.5059089138853, -0.140379565359194}, geoTheStafford, 0.1);
19     }
20
21     @Test
22     public void testLatLongForCentralHallWestminster() {
23         double[] geoCentralHallWestminster = geoLocationUtilities.getLatLongFromPostalcode("SW1H 9NH");
24         assertArrayEquals(new double[] {51.4999994919247, -0.130116786932411}, geoCentralHallWestminster, 0.1);
25     }
26
27     @Test
28     public void testLatLongForOxfordUniversity() {
29         double[] geoOxfordUniversity = geoLocationUtilities.getLatLongFromPostalcode("OX1 2JD");
30         assertArrayEquals(new double[] {51.7580400229174, -1.26201026426438}, geoOxfordUniversity, 0.1);
31     }
32 }
```

So, we're looking at something like the following for an interface:

GeoLocationUtilitiesInterface.java

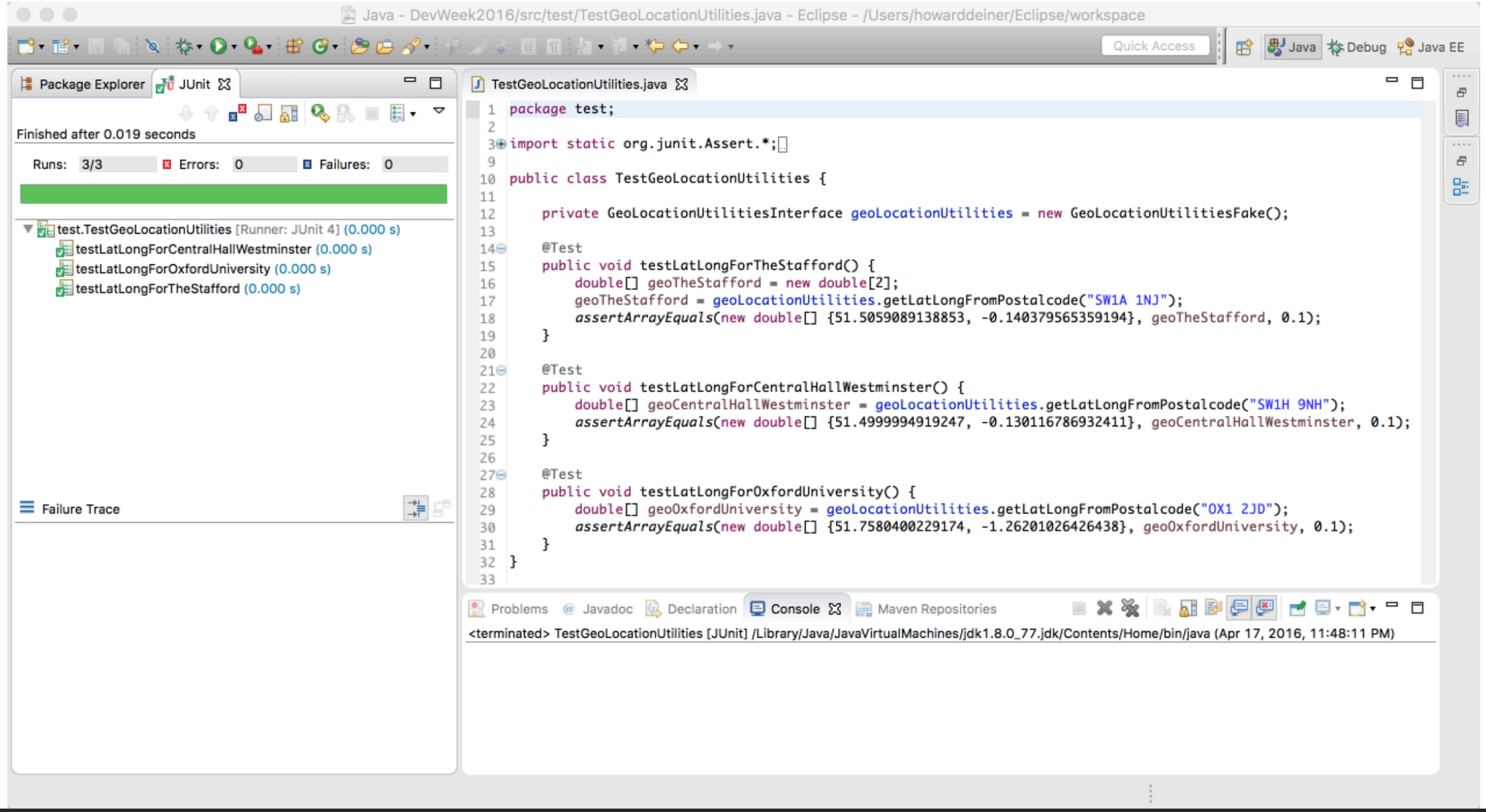
```
1  package main;
2
3  public interface GeoLocationUtilitiesInterface {
4      public double[] getLatLongFromPostalcode(String postalCode);
5  }
```

The fake implementation for that interface is a piece of cake:

GeoLocationUtilitiesFake.java

```
1  package main;
2
3  public class GeoLocationUtilitiesFake implements GeoLocationUtilitiesInterface {
4
5      public double[] getLatLongFromPostalcode(String postalCode) {
6          if (postalCode.equals("SW1A 1NJ")) {
7              return new double[] {51.5059089138853, -0.140379565359194};
8          }
9          if (postalCode.equals("SW1H 9NH")) {
10             return new double[] {51.4999994919247, -0.130116786932411};
11         }
12         if (postalCode.equals("OX1 2JD")) {
13             return new double[] {51.7580400229174, -1.26201026426438};
14         }
15         return null;
16     }
17 }
```

And, of course, the tests work. First time. Every time. And quickly.

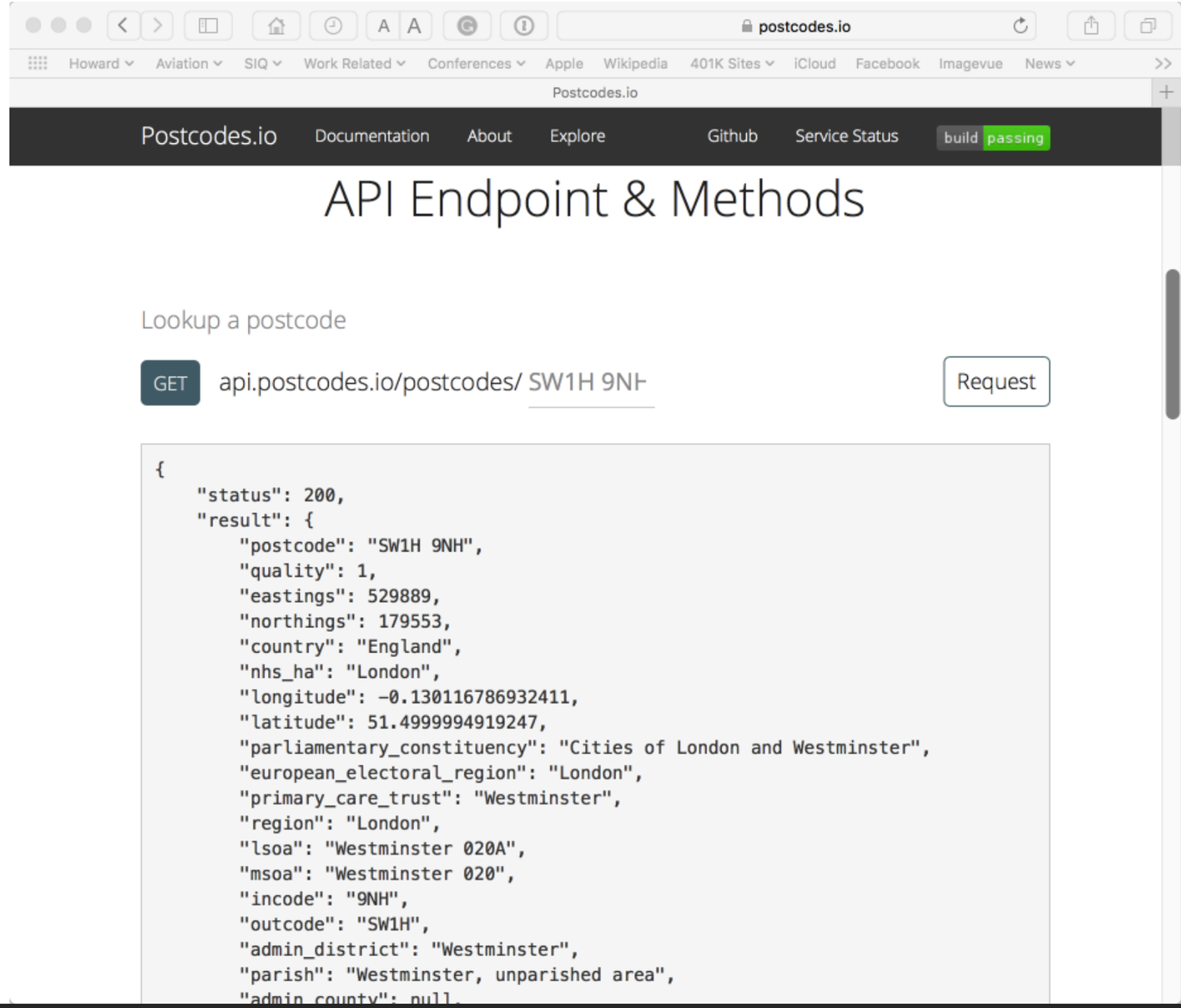



A Merchant
Actually Goes
Postal

To actually go postal, we're going to need to know a few things.

- Exactly how does the Postcode & Geolocation API for the UK work?
- How will we easily consume the web service from our Java code?
- And what changes will have to occur to code a GeoLocationsUtilitiesImplementation and wire it into the unit testing?

Not too terrible. Standard REST get with JSON output.



And to do the REST in Java? Try Resty!

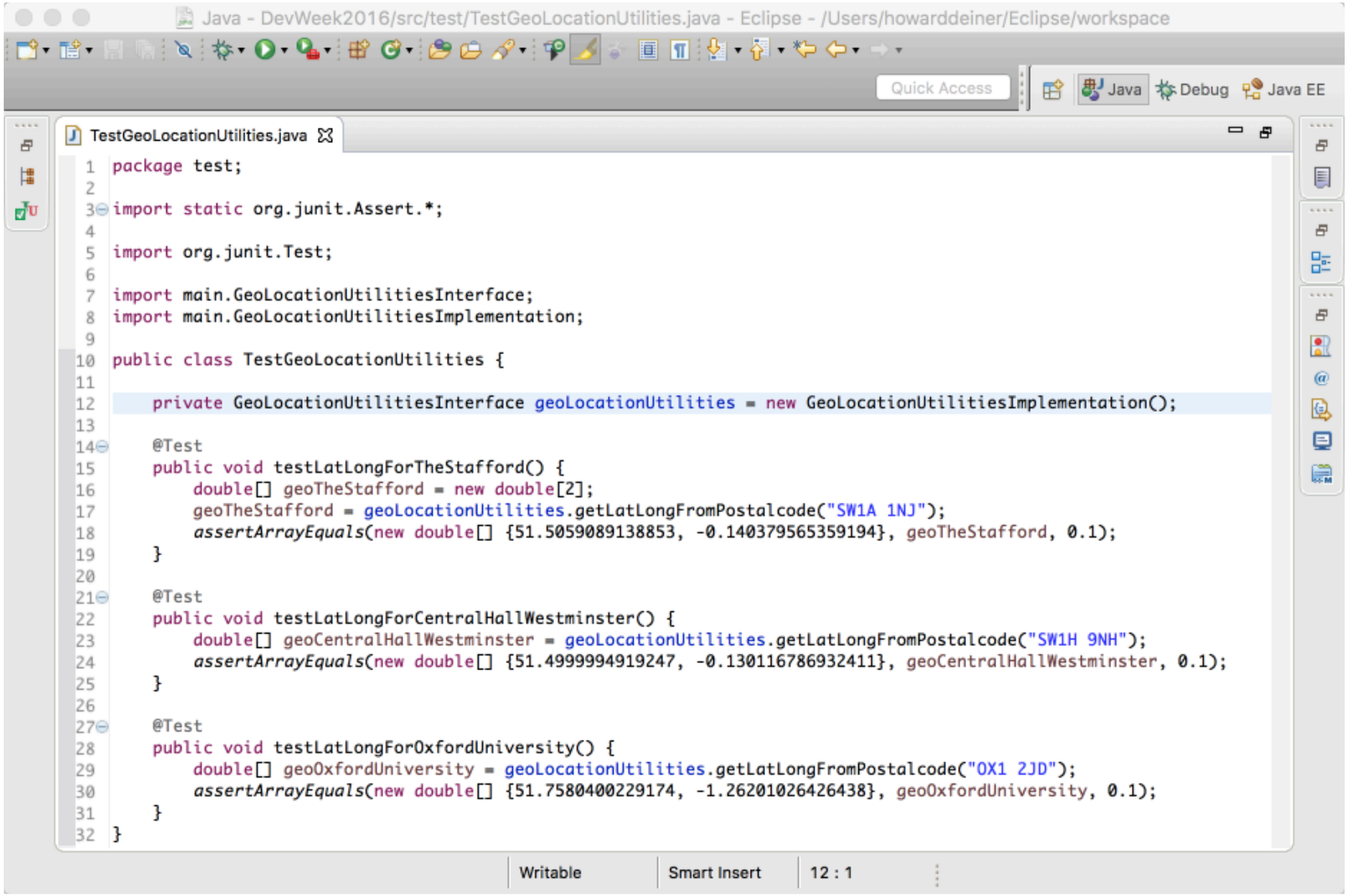
Java - DevWeek2016/src/main/GeoLocationUtilitiesImplementation.java - Eclipse - /Users/howarddeiner/Eclipse/workspace

GeoLocationUtilitiesImplementation.java

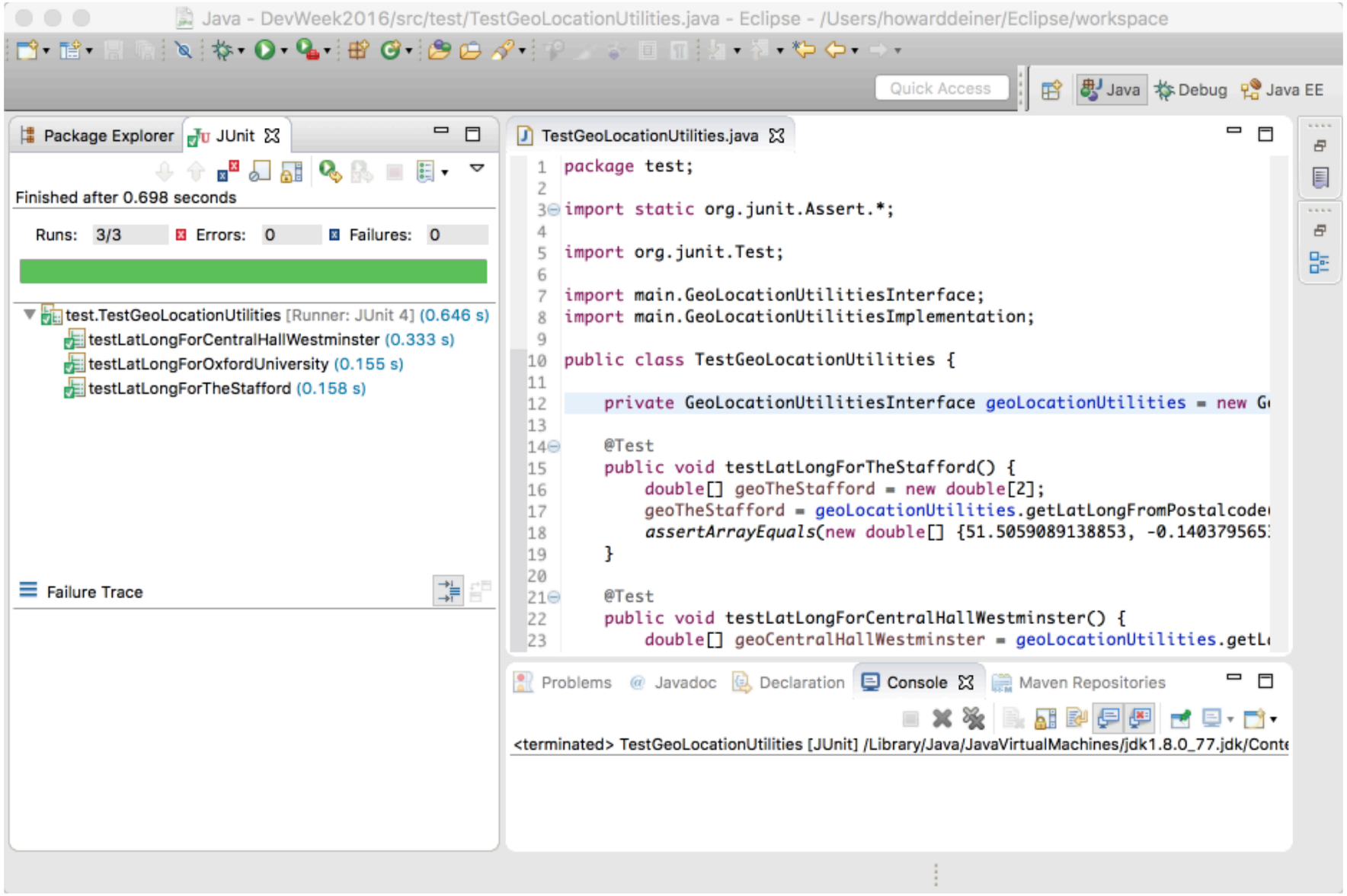
```
1  package main;
2
3  import java.net.URI;
4  import java.net.URL;
5
6  import us.monoid.web.JSONResource;
7  import us.monoid.web.Resty;
8
9  public class GeoLocationUtilitiesImplementation implements GeoLocationUtilitiesInterface {
10     public double[] getLatLongFromPostalcode(String postalCode) {
11         double[] results = new double[2];
12         Resty r = new Resty();
13         try {
14             URL url = new URL("http://api.postcodes.io/postcodes/" + postalCode);
15             URI uri = new URI(url.getProtocol(), url.getUserInfo(), url.getHost(), url.getPort(), url.getPath(), url.getQuery(), url.getRef());
16             JSONResource response = r.json(uri);
17             results[0] = Double.parseDouble(response.get("result.latitude").toString());
18             results[1] = Double.parseDouble(response.get("result.longitude").toString());
19         } catch (Exception e) {
20             e.printStackTrace();
21         }
22         return results;
23     }
24
25 }
26
```

Writable | Smart Insert | 26 : 1

Now, rewire the GeoLocationsUtilitiesInterface to use GeoLocationsUtilitiesImplementation…



And, without doing anything else, all the test still run. Just a little slower. ☺ (0.6 versus 0.0 sec)




A Customer
Class

Finally, we're getting somewhere to start our online store. Guess we're going to need customers, right?

Remember to keep your data clean. I happen to like having all minimum attributes passed into the constructor, to make sure that a "minimum object" is created.

Customers have:

- A name
- An address
- A postalcode
- A phone number
- An eMail address
- A credit card

All standing orders to remain in effect. Don't forget to refactor!

Time to get to work!

I'll be roming around answering
questions and helping those who need it.

Don't be afraid to ask for help.
Don't allow yourself to get stuck!


An Order
Class

OK. This is the moment we've been waiting for. We finally get to have an order.

Same rules here. Clean data. Clean code. Test anywhere you feel fear.

Orders have:

- A merchant
- A customer
- A distance from merchant to customer
- A collecton of line items (extra credit)

Off you go, then. Make us proud!

Oh. That bit about computing the distance from merhcant to customer. You may want this information….

The haversine formula is an equation important in navigation, giving great-circle distances between two points on a sphere from their longitudes and latitudes.

It is a special case of a more general formula in spherical trigonometry, the law of haversines, relating the sides and angles of spherical triangles.

The first table of haversines in English was published by James Andrew in 1805.

No help yet? Next slide.

For any two points on a sphere, the haversine of the central angle between them is given by

$$\operatorname{hav}\left(\frac{d}{r}\right) = \operatorname{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)\operatorname{hav}(\lambda_2 - \lambda_1)$$

where

- hav is the haversine function:

$$\operatorname{hav}(\theta) = \sin^2\left(\frac{\theta}{2}\right) = \frac{1 - \cos(\theta)}{2}$$

- d is the distance between the two points (along a great circle of the sphere; see spherical distance),
- r is the radius of the sphere,
- φ_1, φ_2: latitude of point 1 and latitude of point 2, in radians
- λ_1, λ_2: longitude of point 1 and longitude of point 2, in radians

Solve for d by applying the inverse haversine (if available) or by using the arcsine (inverse sine) function:

$$d = r\,\mathrm{hav}^{-1}(h) = 2r\arcsin\left(\sqrt{h}\right)$$

where h is $\mathrm{hav}(\frac{d}{r})$, or more explicitly:

$$d = 2r\arcsin\left(\sqrt{\mathrm{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)\,\mathrm{hav}(\lambda_2 - \lambda_1)}\right)$$

$$= 2r\arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

Is this helping or hurting? Try the next slide…

No more helping. You're on your own now…

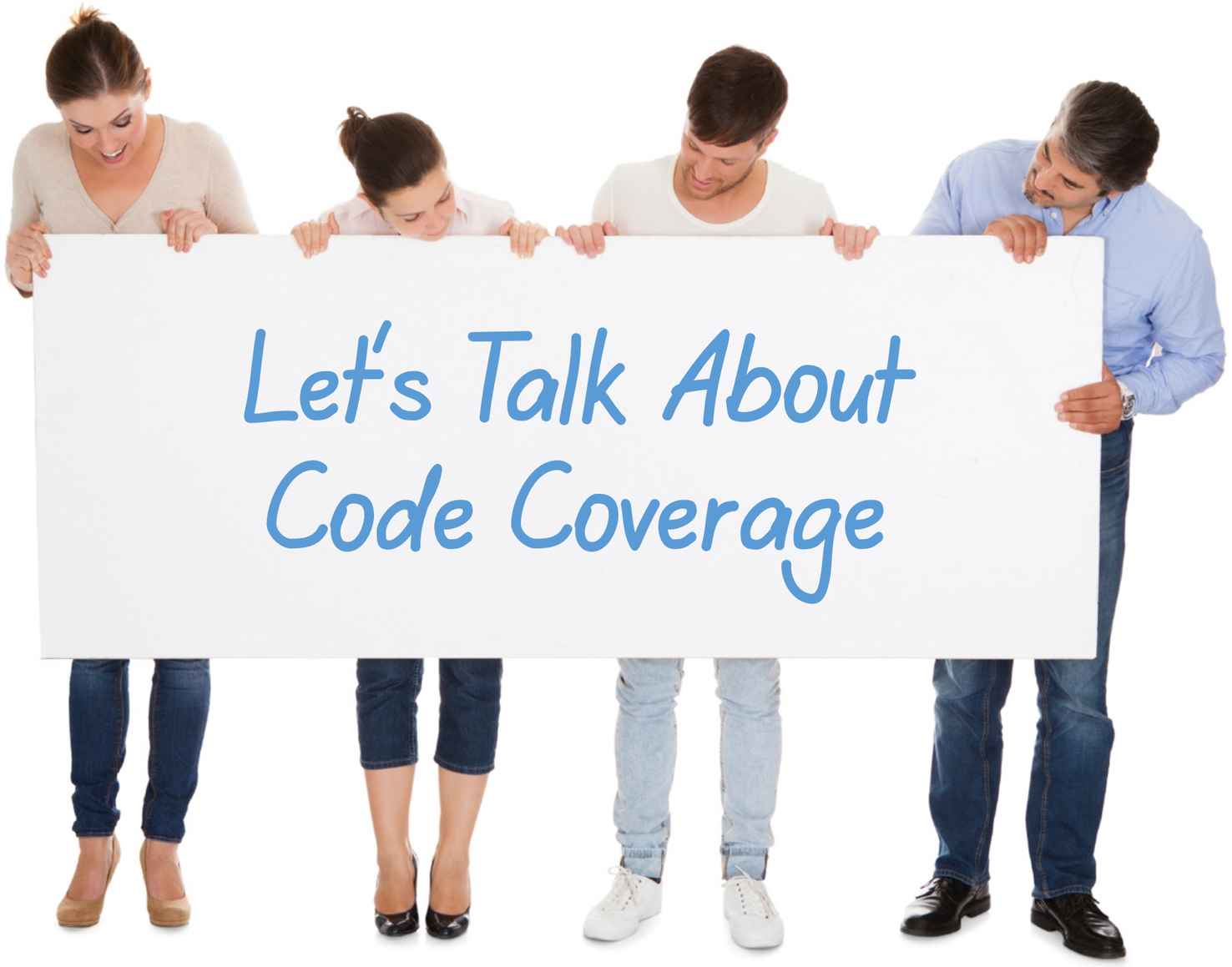
*Haversine.java

```java
package main;

public class Haversine {
    public double getDistanceBetweenTwoLatLongs(double[] latLong1, double[] latLong2) {
        double radiusOfEarthInMiles = 3959.0;

        double distance = 0.0;

        double latitude1 = latLong1[0];
        double longitude1 = latLong1[1];
        double latitude2 = latLong2[0];
        double longitude2 = latLong2[1];

        double deltaLatitude = Math.toRadians(latitude2 - latitude1);
        double deltaLongitude = Math.toRadians(longitude2 - longitude1);

        double latitude1Radians = Math.toRadians(latitude1);
        double latitude2Radians = Math.toRadians(latitude2);

        double a = Math.pow(Math.sin(deltaLatitude/2.0), 2.0)
                 + Math.pow(Math.sin(deltaLongitude/2.0), 2.0) * Math.cos(latitude1Radians) * Math.cos(latitude2Radians);
        double c = 2 * Math.asin(Math.sqrt(a));

        distance = radiusOfEarthInMiles * c;

        return distance;
    }
}
```

Time to get to work!

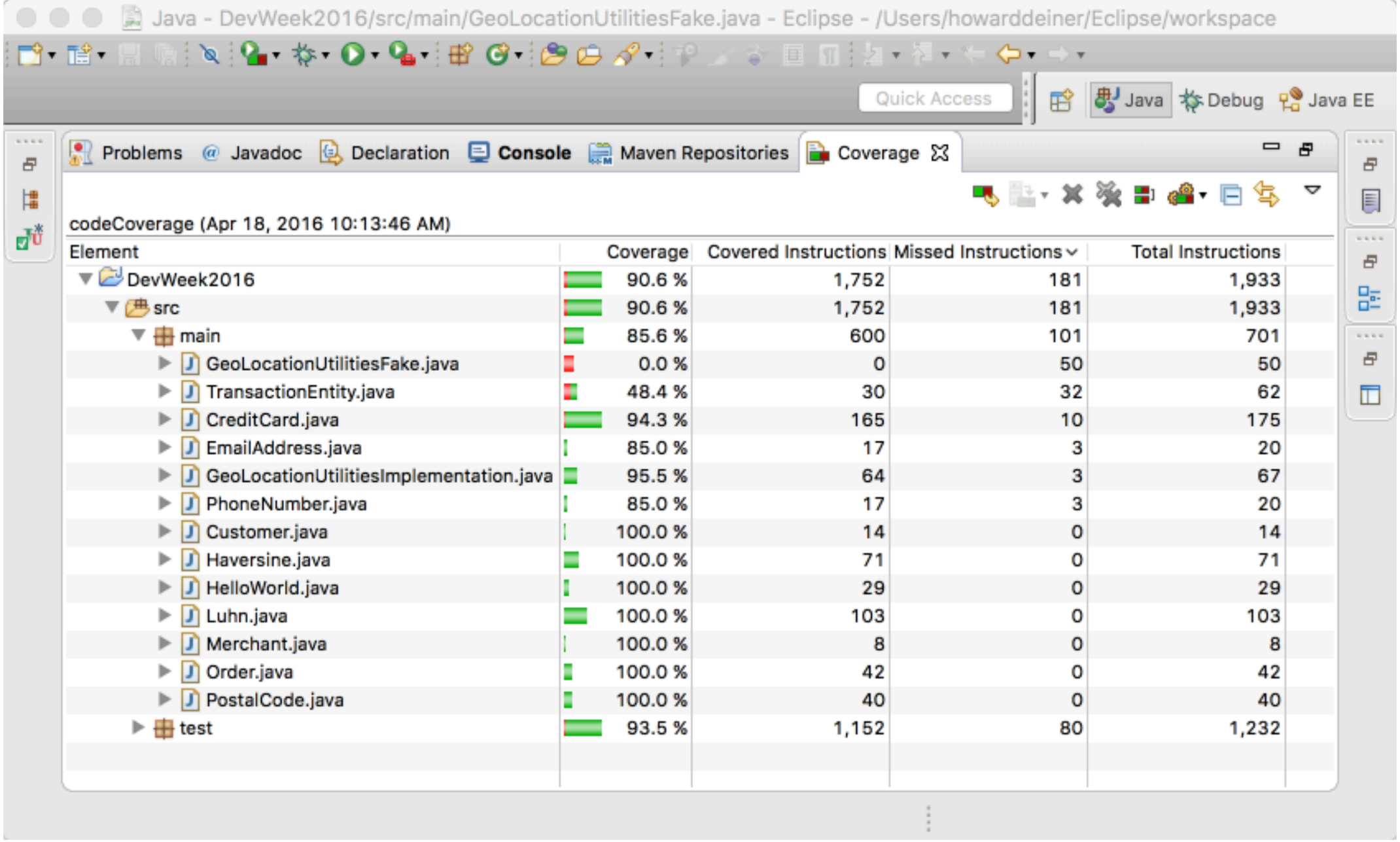
I'll be roming around answering
questions and helping those who need it.

Don't be afraid to ask for help.
Don't allow yourself to get stuck!


Let's Talk About
Code Coverage

Great code coverage will make your boss really happy. He'll be overjoyed that everything is really tested and battle hardened.

First of all, if you develop using Test Driven Development, your code coverage will be extremely high. We didn't do too bad.



Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
DevWeek2016	90.6 %	1,752	181	1,933
src	90.6 %	1,752	181	1,933
main	85.6 %	600	101	701
GeoLocationUtilitiesFake.java	0.0 %	0	50	50
TransactionEntity.java	48.4 %	30	32	62
CreditCard.java	94.3 %	165	10	175
EmailAddress.java	85.0 %	17	3	20
GeoLocationUtilitiesImplementation.java	95.5 %	64	3	67
PhoneNumber.java	85.0 %	17	3	20
Customer.java	100.0 %	14	0	14
Haversine.java	100.0 %	71	0	71
HelloWorld.java	100.0 %	29	0	29
Luhn.java	100.0 %	103	0	103
Merchant.java	100.0 %	8	0	8
Order.java	100.0 %	42	0	42
PostalCode.java	100.0 %	40	0	40
test	93.5 %	1,152	80	1,232

HOWEVER, the code coverage tools only tell us what sections of code were executed.

They don't tell us anything about the quality of the code.

They are easy to game. If you want great code coverage, don't bother writing unit tests. After all the code seems done, run a couple of really intense functional tests under Junit (which isn't such a bad idea). Your code coverage should be pretty good. But you probably didn't test (or even write) tests to handle difficult edge conditions.

That's where Mutation Testing comes in.

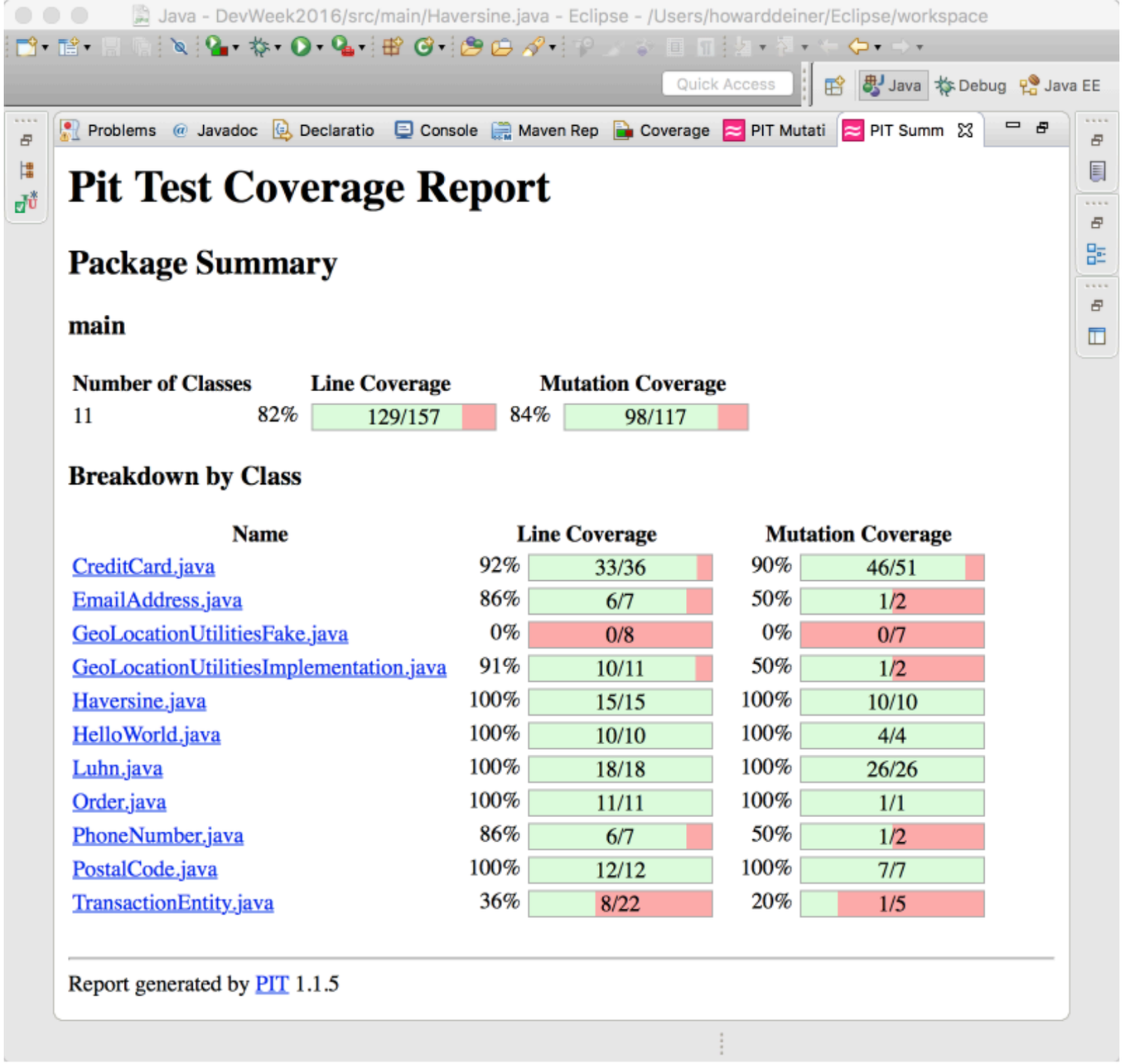
Mutation Testing assumes that you have 100% code coverage. Then it messes with your code. Inverts conditions, returns nulls instead of values, etc. If your tests still run with the same result, then a mutation lives, and your code is missing some unit tests.

So, how did we do?

Not bad.

I actually wanted more issues to show you, but because we did Test Driven Development, our coverage was quite high.

Remember this. Metrics in general are most useful when we look at trends, and use them as a jumping off point to have a conversation. If you fixate on them, you are bound to be looking at the wrong things.



Pit Test Coverage Report

Package Summary

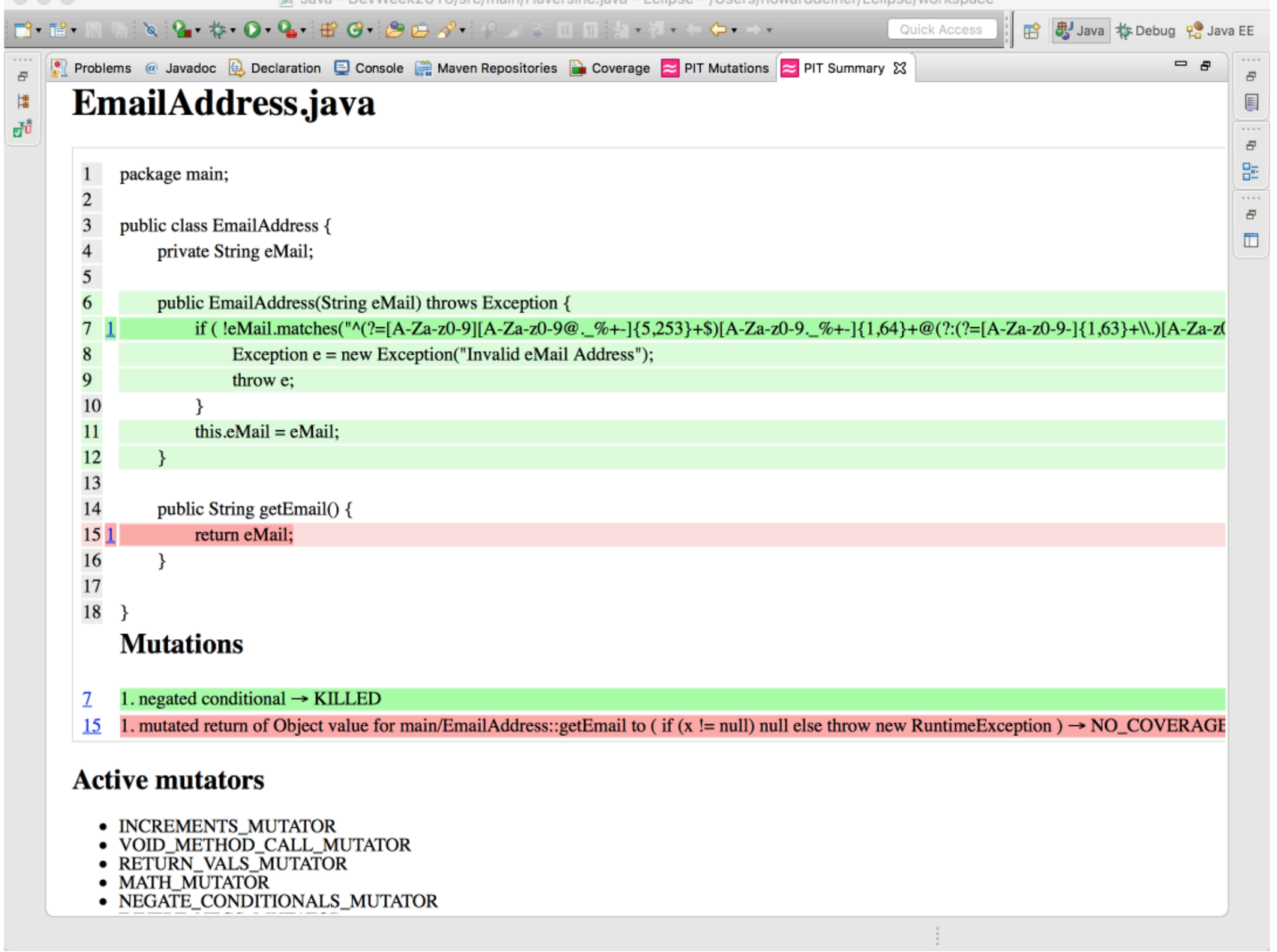
main

Number of Classes	Line Coverage		Mutation Coverage	
11	82%	129/157	84%	98/117

Breakdown by Class

Name	Line Coverage		Mutation Coverage	
CreditCard.java	92%	33/36	90%	46/51
EmailAddress.java	86%	6/7	50%	1/2
GeoLocationUtilitiesFake.java	0%	0/8	0%	0/7
GeoLocationUtilitiesImplementation.java	91%	10/11	50%	1/2
Haversine.java	100%	15/15	100%	10/10
HelloWorld.java	100%	10/10	100%	4/4
Luhn.java	100%	18/18	100%	26/26
Order.java	100%	11/11	100%	1/1
PhoneNumber.java	86%	6/7	50%	1/2
PostalCode.java	100%	12/12	100%	7/7
TransactionEntity.java	36%	8/22	20%	1/5

Report generated by PIT 1.1.5


Quick Access
Java
Debug
Java EE
Problems
Javadoc
Declaration
Console
Maven Repositories
Coverage
PIT Mutations
PIT Summary
EmailAddress.java
1 package main;
2
3 public class EmailAddress {
4 private String eMail;
5
6 public EmailAddress(String eMail) throws Exception {
7 1 if (!eMail.matches("^(?=[A-Za-z0-9][A-Za-z0-9@._%+-]{5,253}+$)[A-Za-z0-9._%+-]{1,64}+@(?:(?=[A-Za-z0-9-]{1,63}+\\.)[A-Za-z(
8 Exception e = new Exception("Invalid eMail Address");
9 throw e;
10 }
11 this.eMail = eMail;
12 }
13
14 public String getEmail() {
15 1 return eMail;
16 }
17
18 }
Mutations
7 1. negated conditional → KILLED
15 1. mutated return of Object value for main/EmailAddress::getEmail to (if (x != null) null else throw new RuntimeException) → NO_COVERAGE
Active mutators
INCREMENTS_MUTATOR
VOID_METHOD_CALL_MUTATOR
RETURN_VALS_MUTATOR
MATH_MUTATOR
NEGATE_CONDITIONALS_MUTATOR

Yes.

We are looking at the end.

Many thanks to all who contributed and persevered during this long day.

Be sure to contact me when issues arise. Even if it's just to chat.

Time for questions, answers, debate, and then depart.

TDD for the newb Who Wants to Become an Apprentice

Howard Deiner

Agile Technical Practices and Process Coach

SolutionsIQ, Redmond WA, USA