TEST DRIVEN DEVELOPMENT

Red, Green, and Refactor to Rule them All

THE DARK LORD SAURON HAS A PROBLEM!

- Now that all those mean old elves, dwarves and humans have rings of power, they feel like they can run around doing whatever they want.
- That's not very fair! Why can't Sauron rule the world for a change?

BUT THEN, HE HAS AN IDEA!

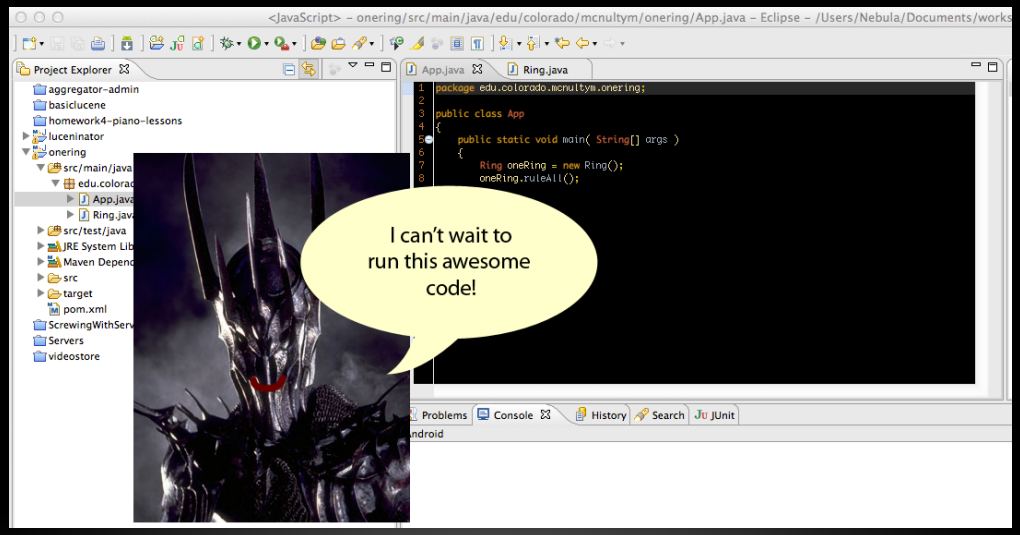
- Why not create…

ONE RING TO RULE THEM ALL!

- Bet you couldn't see that one coming
- But how to build it?

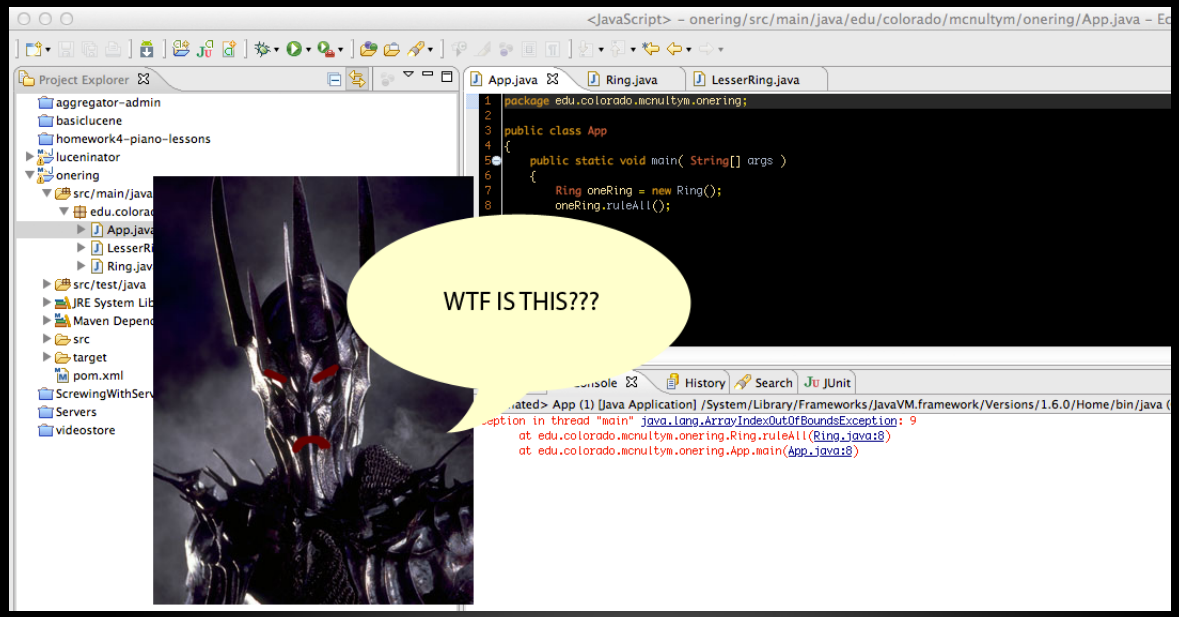
JUMPING RIGHT IN!

- Never being the most patient of dark lords, Sauron decides to get coding straight away.
- Taking out his laptop of ultimate doom, he codes well into the night, until he over nine thousand classes and eleventy billion functions.
- Finally, he is ready to test his code. With eager anticipation, he presses the play button in his trusty IDE of Ultimate Doom.



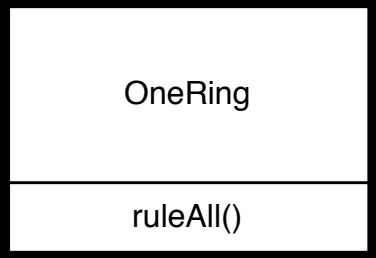
BUT THERE'S A PROBLEM!

- Instead of ruling them all as anticipated, Sauron's ruleAll() function spews out an angry red exception!



UNFORTUNATELY, SAURON HATES UML DIAGRAMS

- Sauron starts making some diagrams, but he finds the process tedious and bureaucratic
- He gets so frustrated he sets fire to an innocent hobbit village

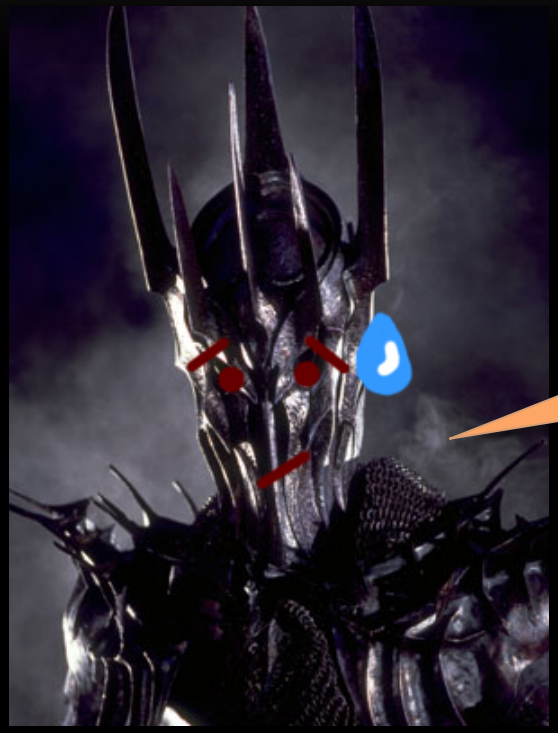


SUDDENLY, GANDALF APPEARS!

I know we are normally foes, but I cannot stand by and let you burn down any more hobbit villages, so I'll help you with your plight.

Do not despair, there is a better way!
Why not write the tests before you write the code?

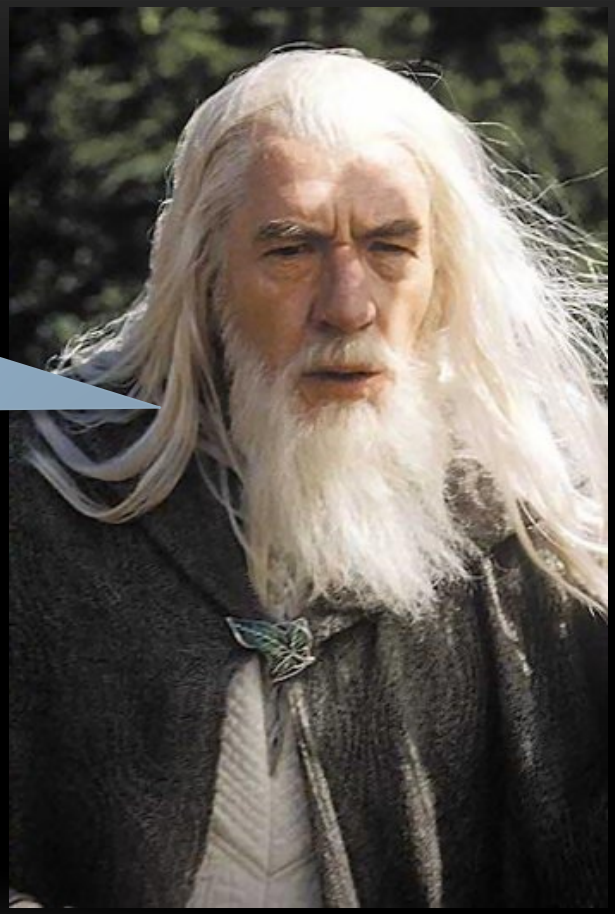
HUH?



I appreciate the advice, but how on Middle Earth am I supposed to write tests for code I haven't even written yet?

THAT'S EASY!

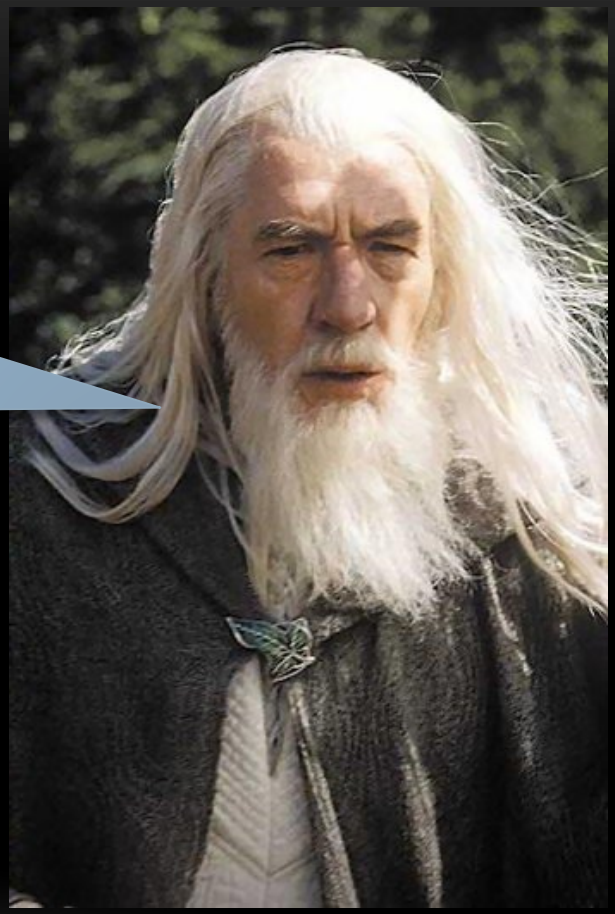
That's easy! You just have to use
…
Test Driven Development!



SURE, WHY NOT?

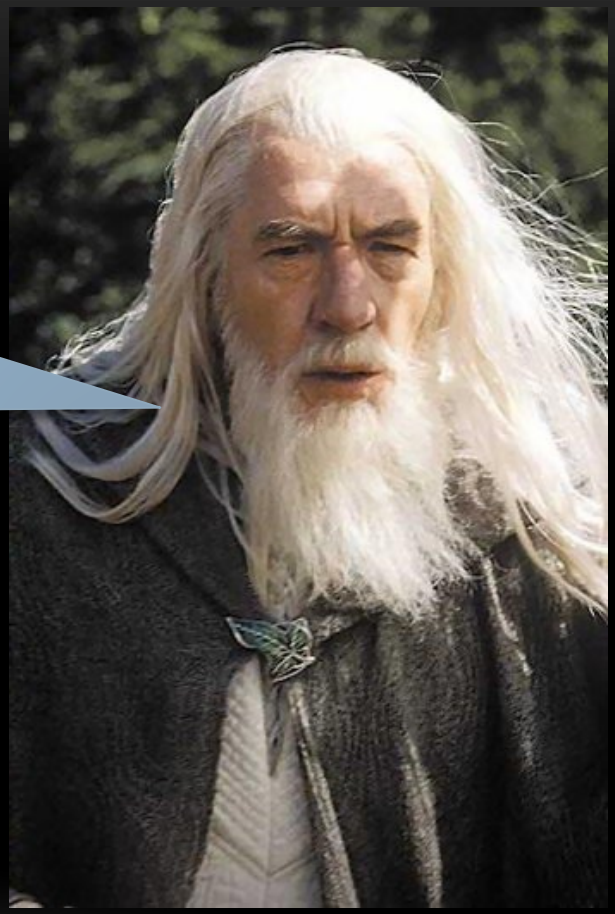
Well, if it will help me gain supreme dominance over Middle Earth, I suppose it's worth a shot.

What's that?



Nothing, nothing. Shall we get started?

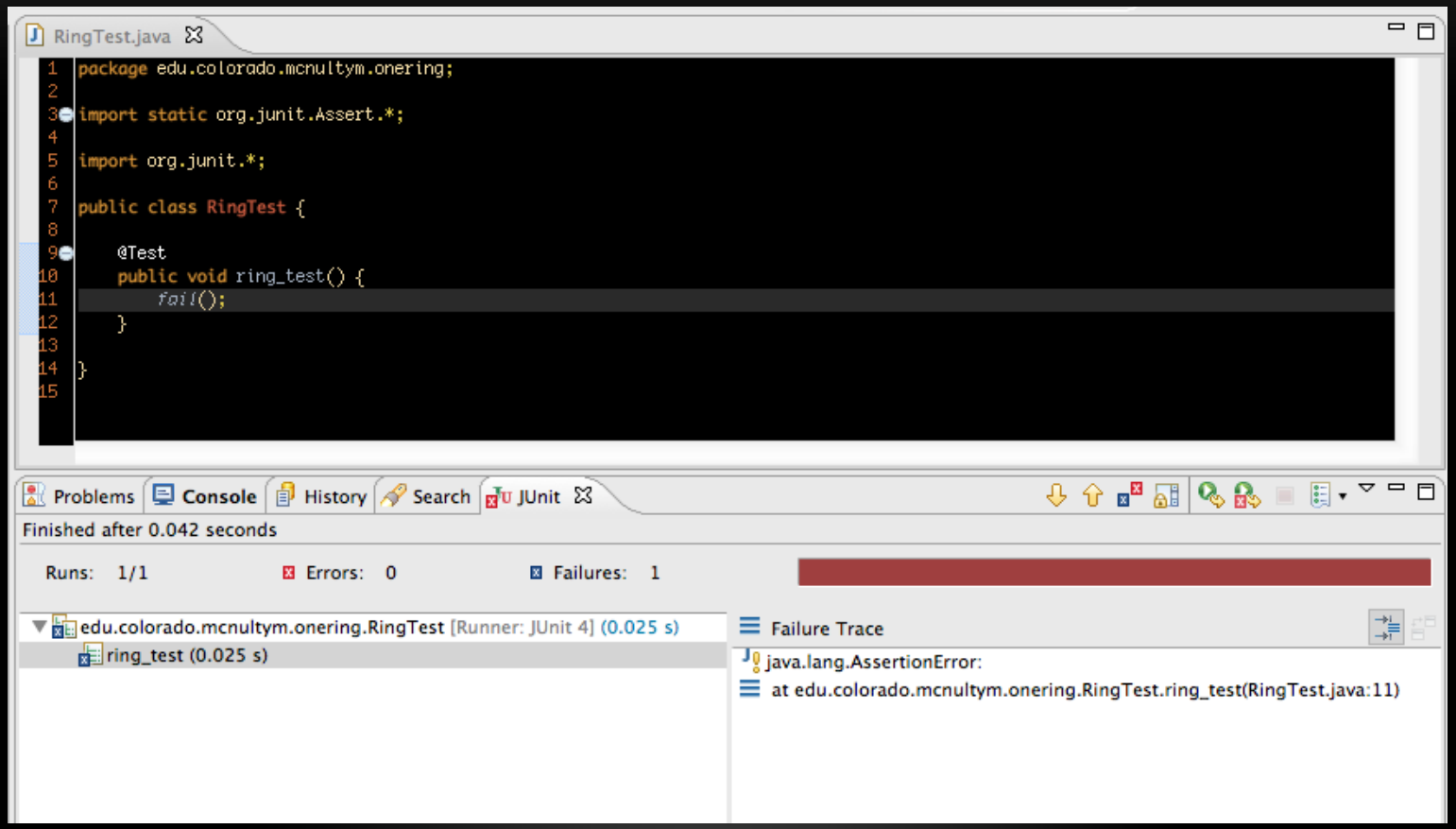
Very well then. Let’s begin.



TEST DRIVEN DEVELOPMENT

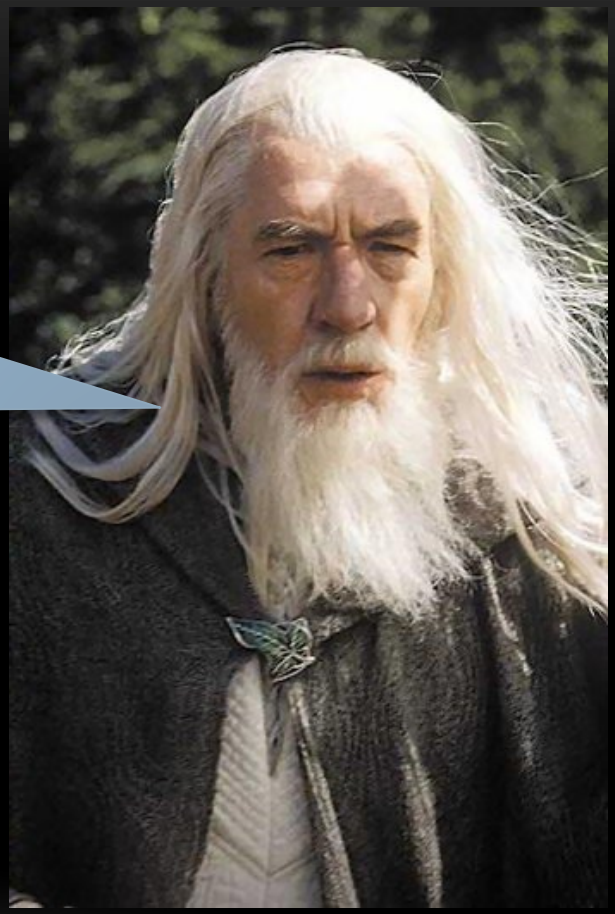
- Test driven development means understanding what you want your code to do before you actually write it
- Write tests for the functionality you want your code to have, not the code you have already.
- There are many languages and frameworks that support test driven development, but for this demonstration, we will be using Elvish Java with Numenorian J-Unit.
- The first thing you want to do is make a failing test.

A SIMPLE FAILING TEST



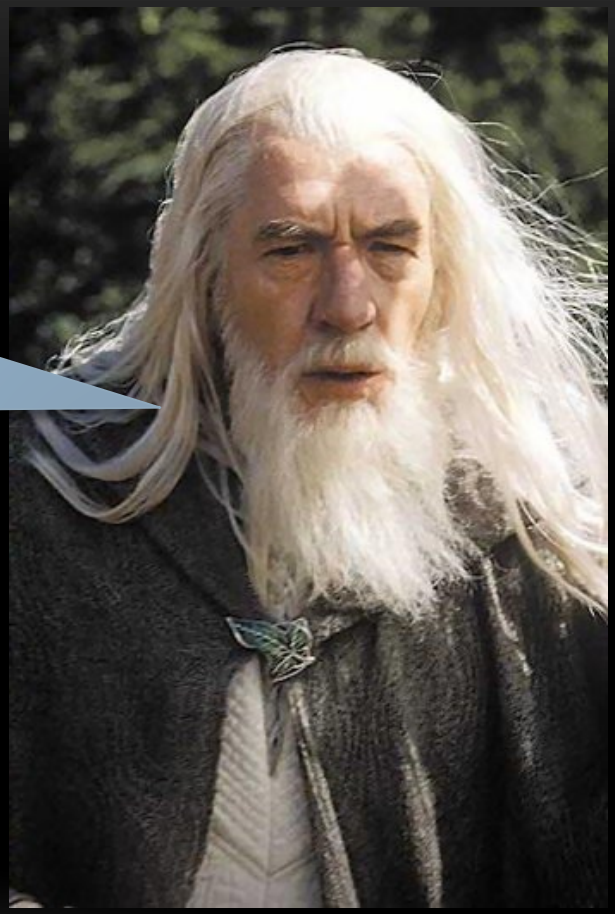
Wait a second … why would we want to do that? Don't we want to start with a test that passes?

YOU SHALL NOT PASS!!!



*Disclaimer: Sorry, I had to.

… Yet. The initial failing test is just to confirm that the test is being run properly. But now we need to make sure it's actually testing something.

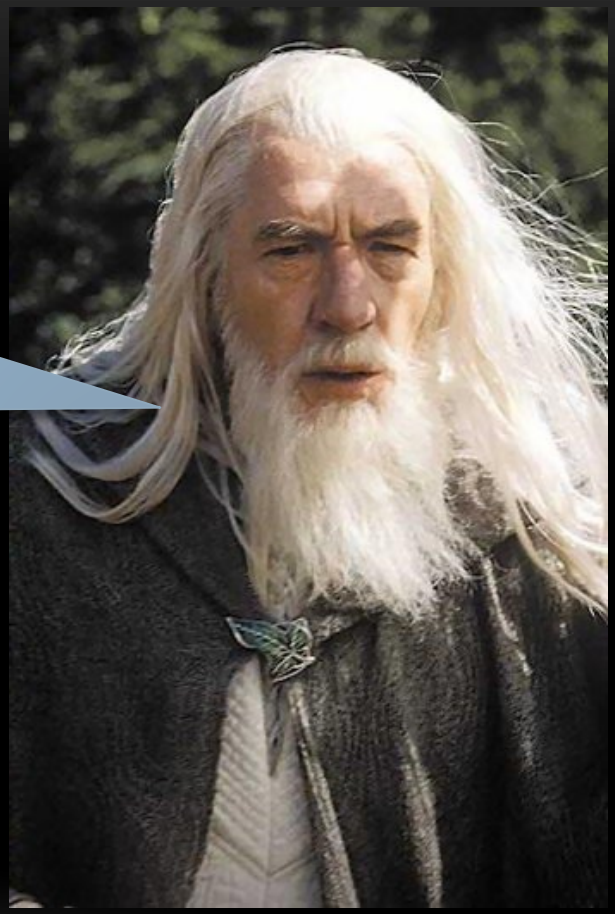


THE TDD CYCLE

- To understand TDD, you must understand the following cycle:
  - Write a small, failing test
  - Make the smallest possible change to make the test pass
  - Refactor mercilessly

I like to do things mercilessly!

… Sigh. I know you do, Sauron.
Let’s continue, shall we?



SO WHERE TO START?

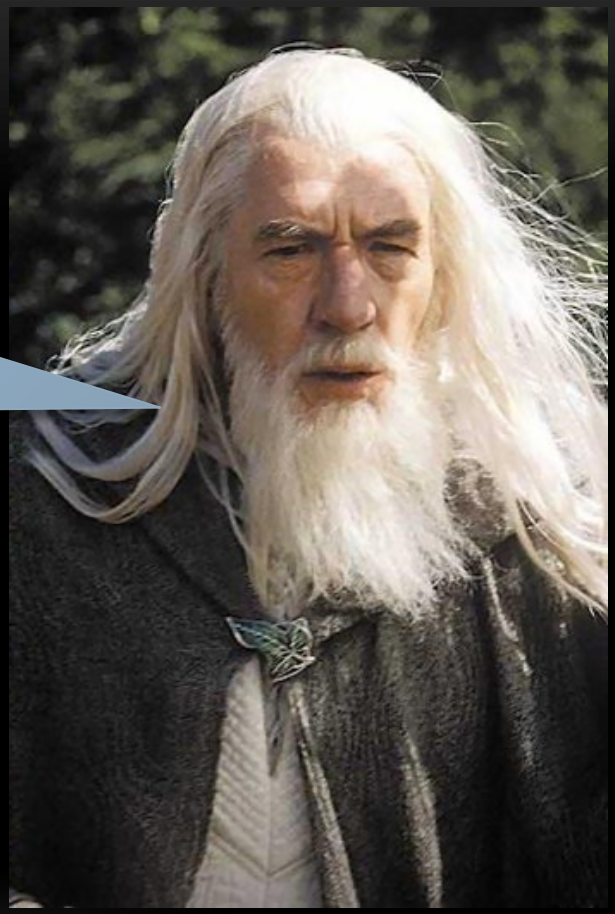
- The most obvious first step is to make a list of the things we want our code to accomplish. In our case:
  - The One Ring shall bear the following inscription:

```
4 Three Rings for the Elven-kings under the sky,
5 Seven for the Dwarf-lords in their halls of stone,
6 Nine for Mortal Men doomed to die,
7 One for the Dark Lord on his dark throne
8 In the Land of Mordor where the Shadows lie.
9 One Ring to rule them all, One Ring to find them,
10 One Ring to bring them all and in the darkness bind them
11 In the Land of Mordor where the Shadows lie.
```

- The One Ring shall have control of the following:
  - The Three Elven Rings
  - The Seven Dwarven Rings
  - The Nine Human Rings

That looks a whole lot like a Requirements Document.

Patience. It is acceptance criteria. We will be writing actual code soon.



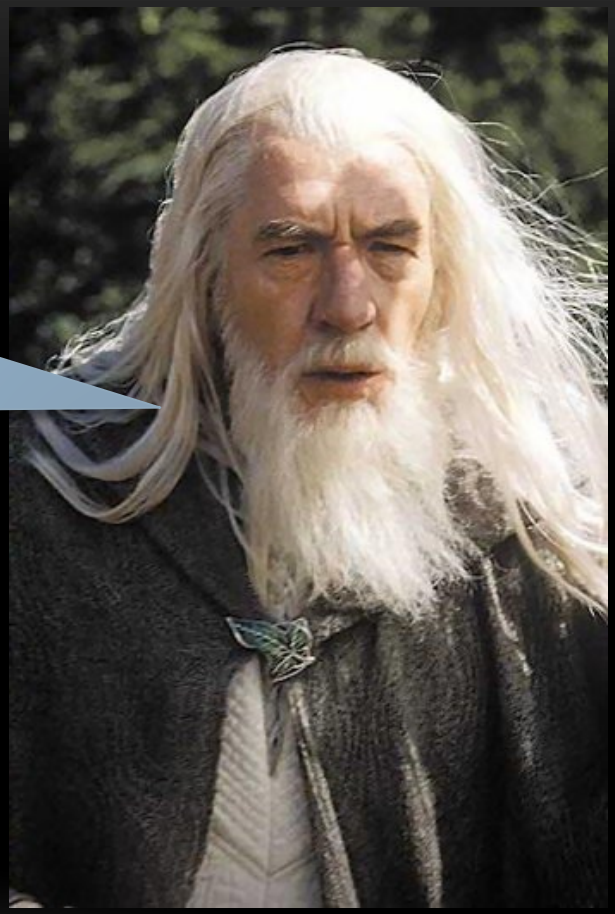
OUR FIRST (REAL) TEST

- First let's test that the code has the inscription

```
7  public class RingTest {
8
9      @Test
10     public void ring_should_have_correct_inscription() {
11         OneRing ring = new OneRing();
12         String inscription = ring.getInscription();
13         assertTrue(inscription.contains("One Ring to rule them all"));
14     }
15
16 }
17
```

Hey, that code doesn't even compile!

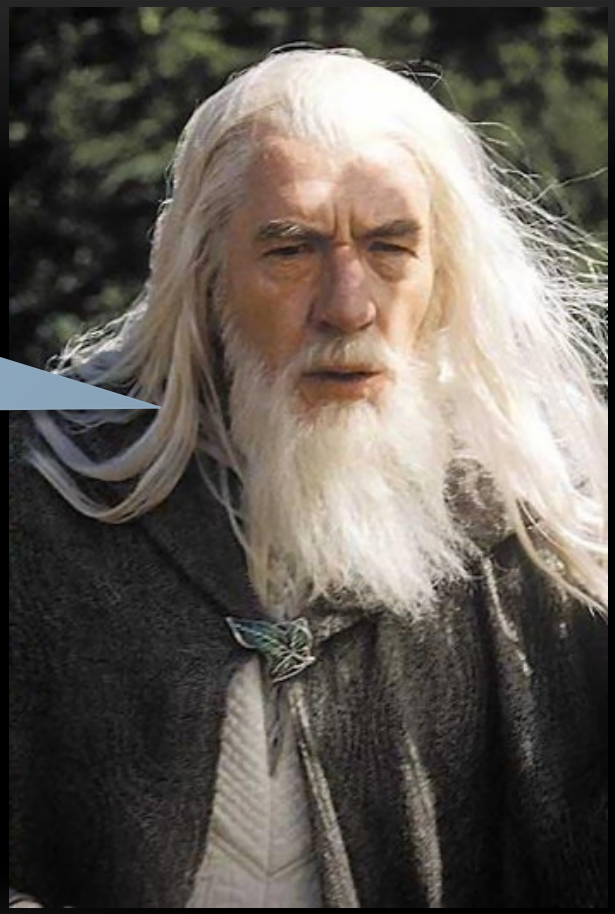
You are correct. The next step is to write the minimum amount of code necessary to make it compile.



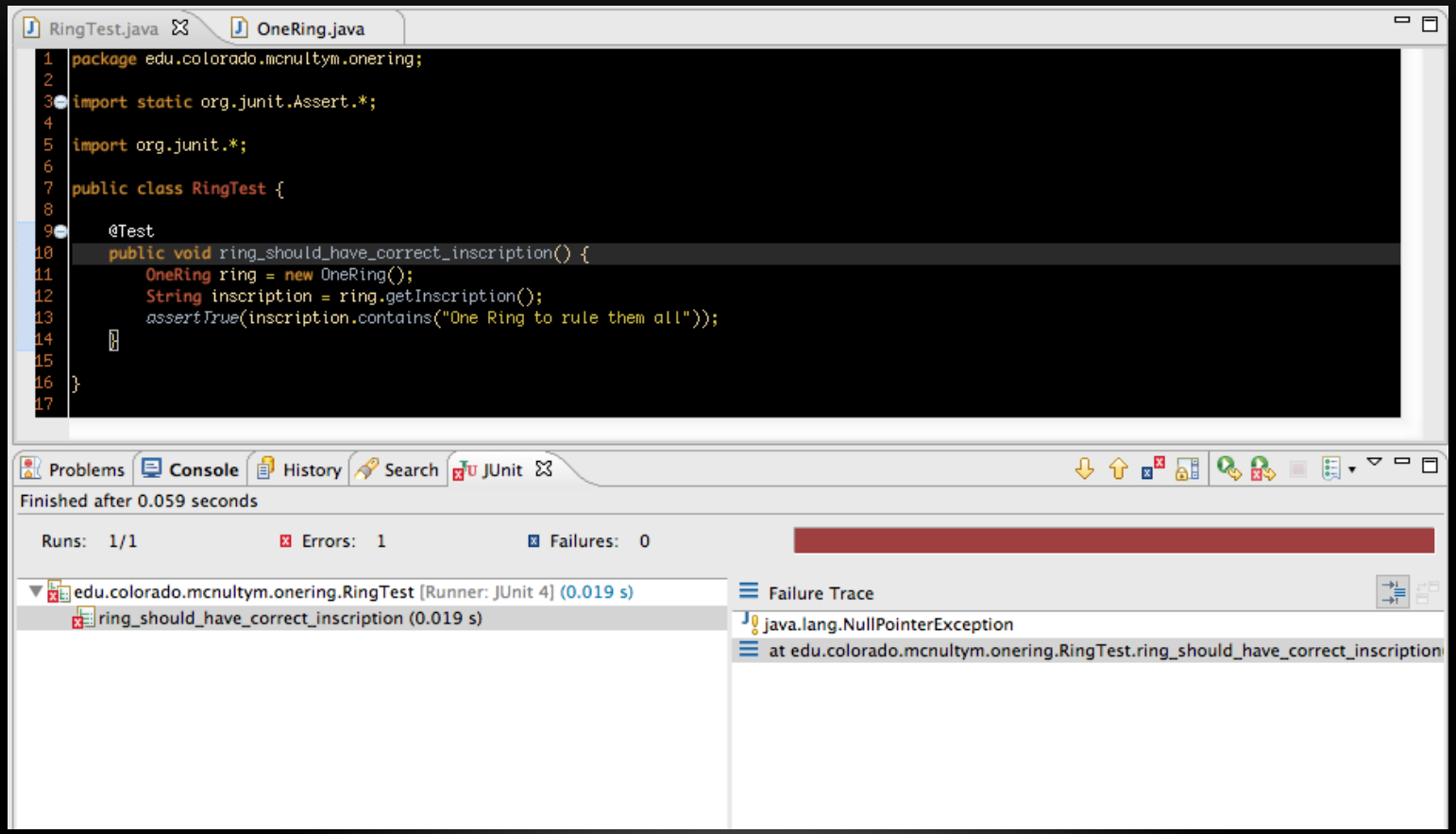
A RUDIMENTARY RING CLASS

```
1  package edu.colorado.mcnultym.onering;
2
3  public class OneRing {
4
5      String inscription;
6
7      public String getInscription() {
8          return inscription;
9      }
10
11 }
```

Now that we run the test, it should fail.

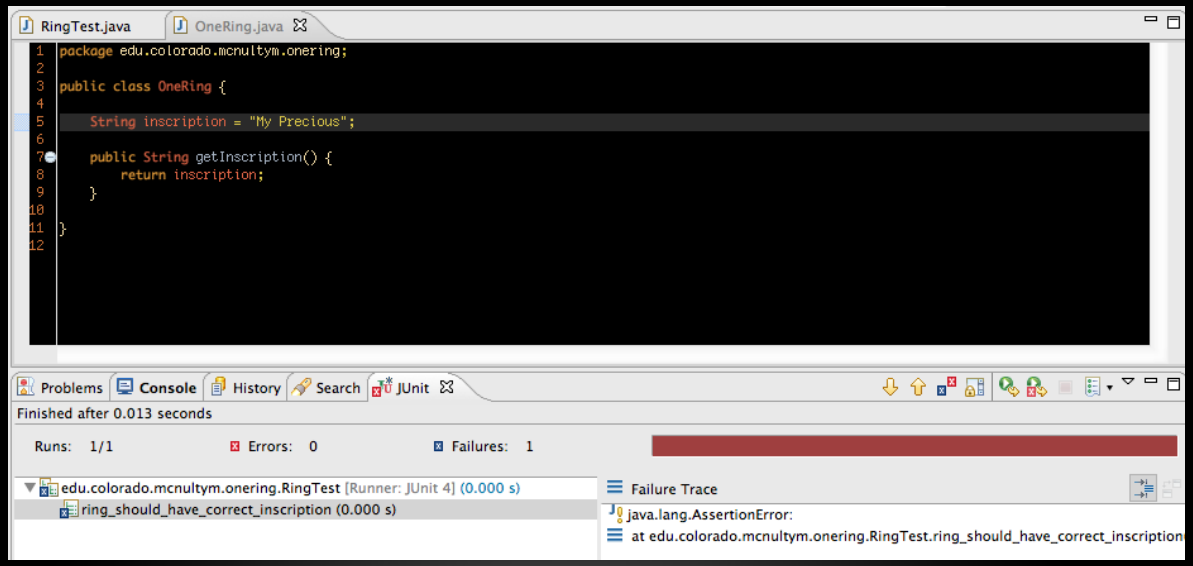


FAILING TEST

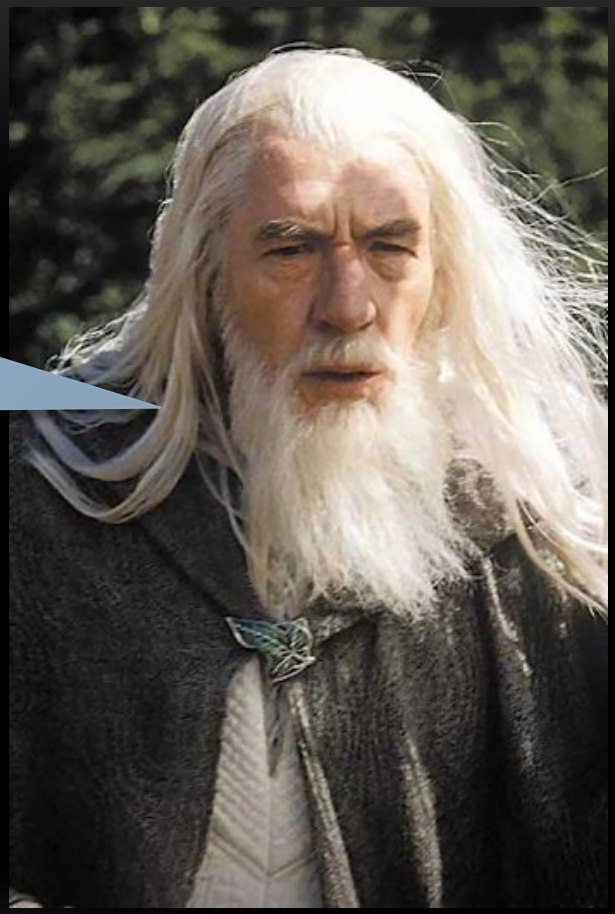


ASSERTIONS

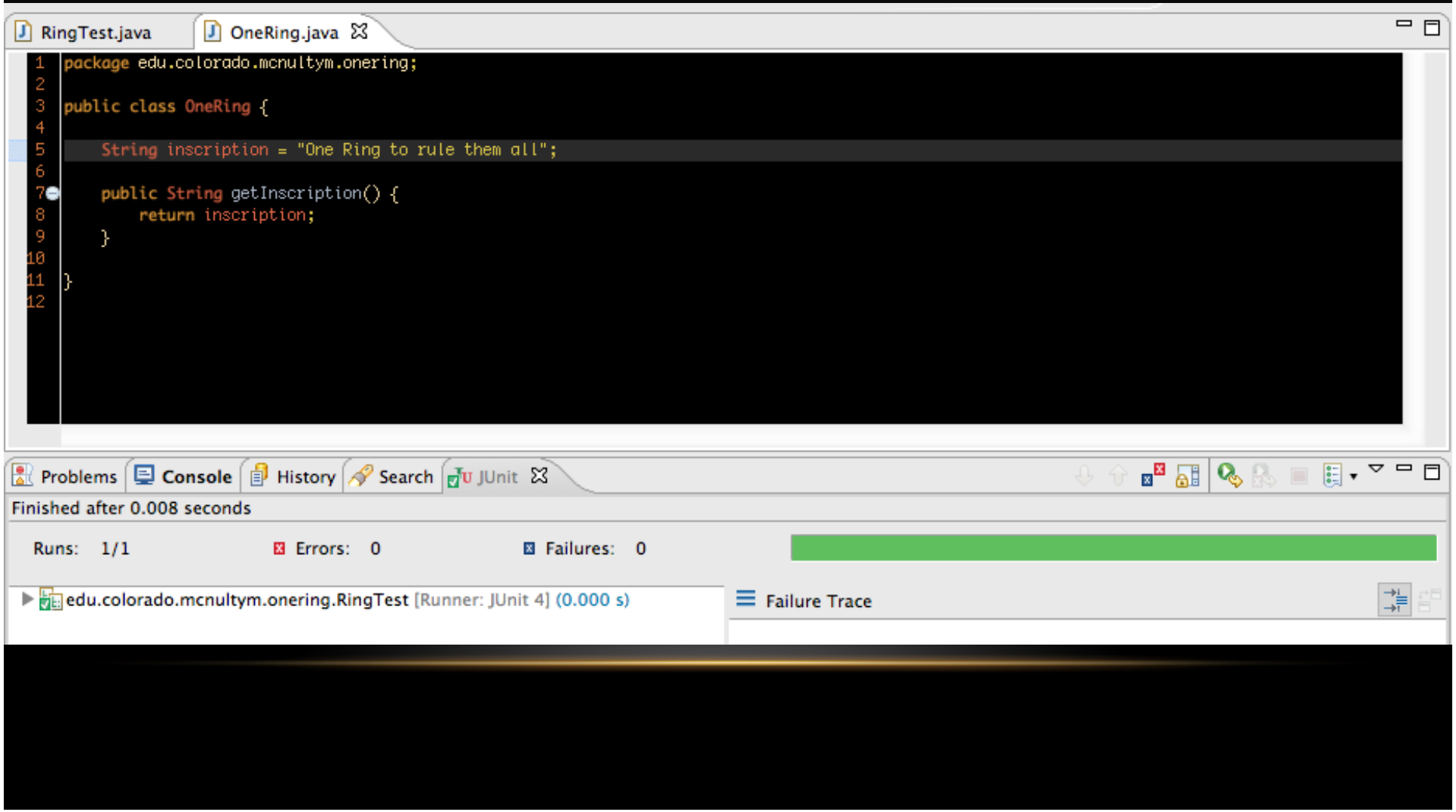
- Assertions, as the name suggests assert whether a given condition holds or not
- Our assertion makes sure that the inscription contains the text “One Ring to Rule them All”
- Right now our assertion fails because inscription is null. If we fill inscription with a garbage variable, we’ll get a more legitimate failure:



Now let's get that test to pass!

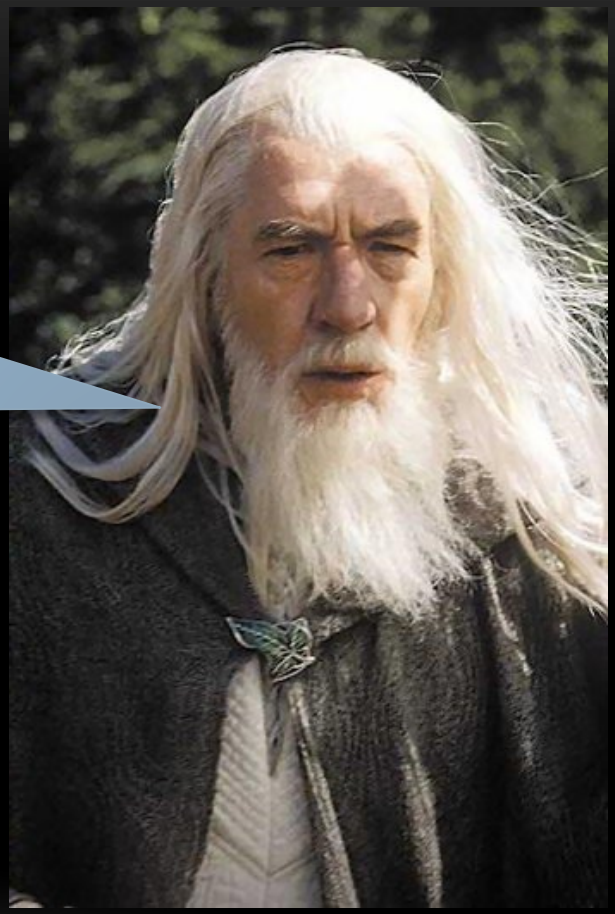


A PASSING TEST

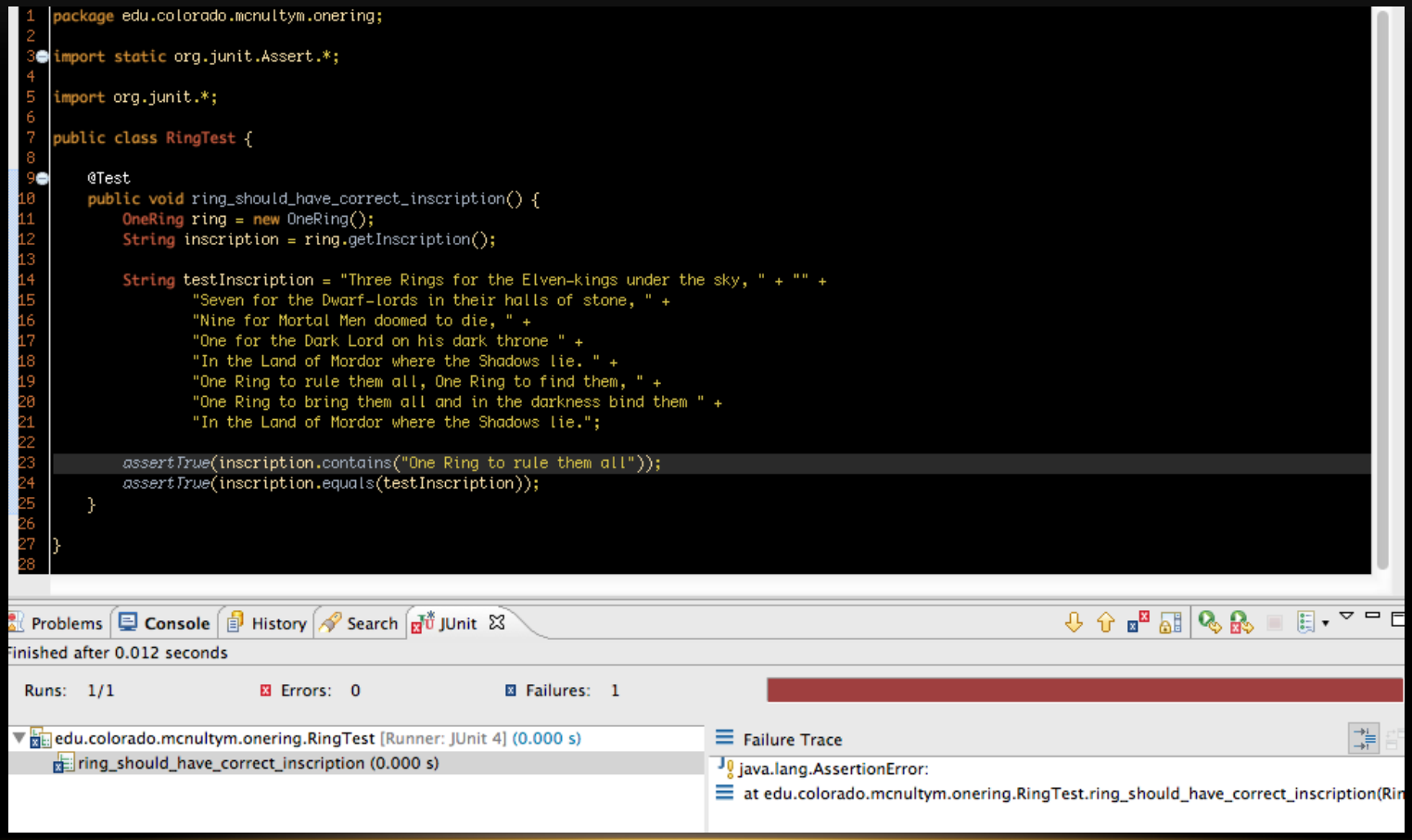


Well sure, the test passes, but it doesn't do what I wanted it to do. I want the ring to have the whole inscription!

Of course. We're getting to that. The next step is to refactor and remove duplicate code. We don't have any duplicate code yet, but our test could use some more assertions.

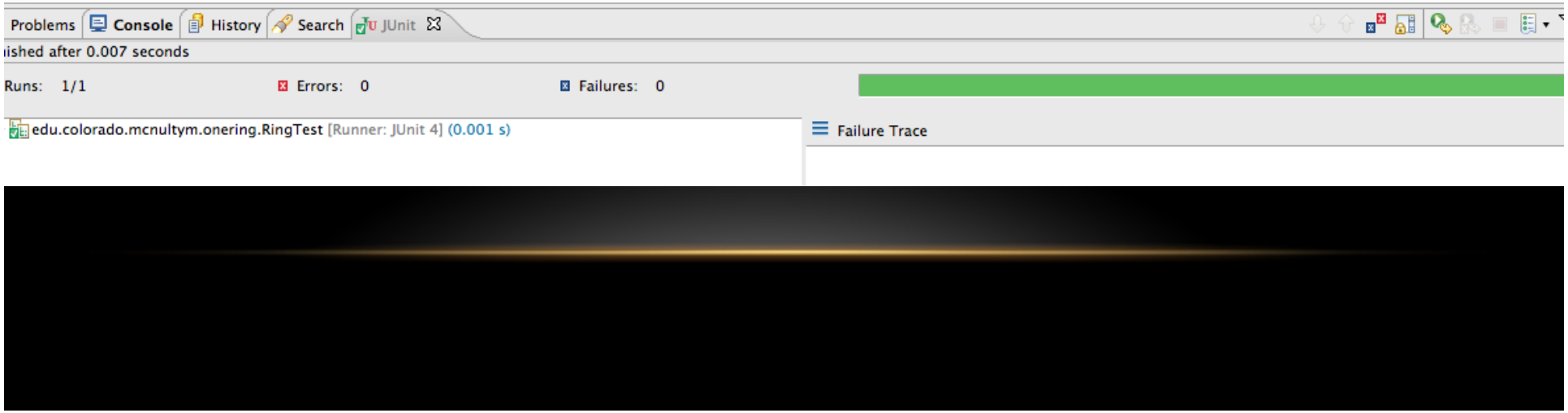


FAILING ONCE MORE



NOW BACK TO GREEN

```java
package edu.colorado.mcnultym.onering;

public class OneRing {

    String inscription;

    public OneRing() {

        this. inscription = "Three Rings for the Elven-kings under the sky, " + "" +
            "Seven for the Dwarf-lords in their halls of stone, " +
            "Nine for Mortal Men doomed to die, " +
            "One for the Dark Lord on his dark throne " +
            "In the Land of Mordor where the Shadows lie. " +
            "One Ring to rule them all, One Ring to find them, " +
            "One Ring to bring them all and in the darkness bind them " +
            "In the Land of Mordor where the Shadows lie.";
    }

    public String getInscription() {
        return inscription;
    }

}
```

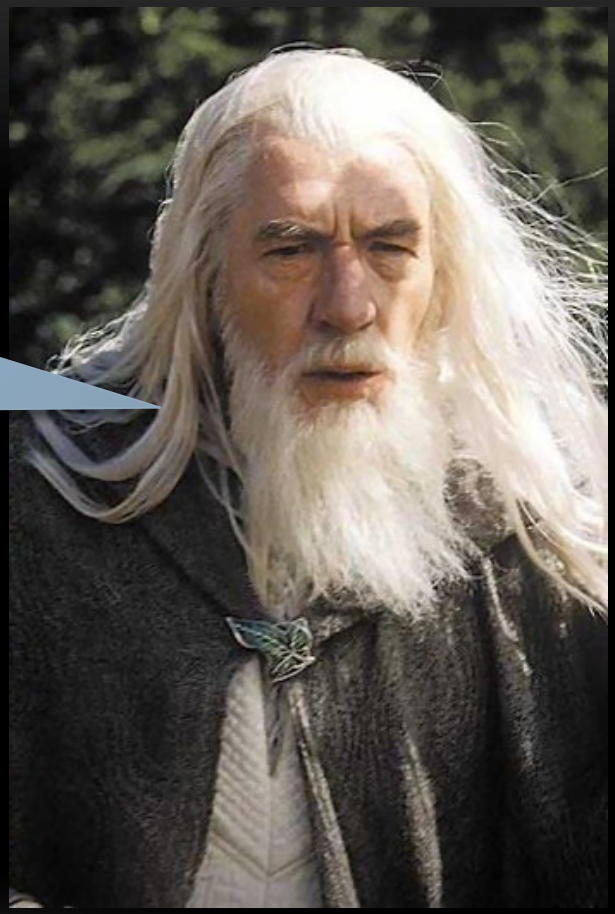


REFACTOR MERCILESSLY!

- Now that we're testing for the whole screen, we don't really need the first assertion statement, so we'll remove it.

```
@Test
public void ring_should_have_correct_inscription() {
    OneRing ring = new OneRing();
    String inscription = ring.getInscription();

    String testInscription = "Three Rings for the Elven-kings under the sky, " + "" +
            "Seven for the Dwarf-lords in their halls of stone, " +
            "Nine for Mortal Men doomed to die, " +
            "One for the Dark Lord on his dark throne " +
            "In the Land of Mordor where the Shadows lie. " +
            "One Ring to rule them all, One Ring to find them, " +
            "One Ring to bring them all and in the darkness bind them " +
            "In the Land of Mordor where the Shadows lie.";

    assertTrue(inscription.equals(testInscription));
}
```

And so the cycle continues.



WRITE A TEST AND GET IT TO COMPILE!

```java
package edu.colorado.mcnultym.onering;

import java.util.List;

public class OneRing {

    String inscription;
    List<ElvenRing> elvenRings;

    public OneRing() {

        this. inscription = "Three Rings for the Elven-kings under the sky, " + "" +
            "Seven for the Dwarf-lords in their halls of stone, " +
            "Nine for Mortal Men doomed to die, " +
            "One for the Dark Lord on his dark throne " +
            "In the Land of Mordor where the Shadows lie. " +
            "One Ring to rule them all, One Ring to find them, " +
            "One Ring to bring them all and in the darkness bind them " +
            "In the Land of Mordor where the Shadows lie.";
    }

    public String getInscription() {
        return inscription;
    }

    public List<ElvenRing> getElvenRings() {
        return elvenRings;
    }

}
```

```java
@Test
public void ring_should_control_three_elven_rings() {
    OneRing ring = new OneRing();
    List<ElvenRing> elvenRings = ring.getElvenRings();
    assertTrue(elvenRings.size() == 3);
}
```

```java
package edu.colorado.mcnultym.onering;

public class ElvenRing {

    public void printMessage() {
        System.out.println("I'm an elvish ring! I'm pretty and fair!");
        System.out.println("Sauron rules me!");
    }

}
```

WATCH IT FAIL

RingTest.java | OneRing.java | ElvenRing.java

```
1  package edu.colorado.mcnultym.onering;
2
3  import static org.junit.Assert.*;
4
5  import java.util.List;
6
7  import org.junit.*;
8
9  public class RingTest {
10
12     public void ring_should_have_correct_inscription() {
27
28     @Test
29     public void ring_should_control_three_elven_rings() {
30         OneRing ring = new OneRing();
31         List<ElvenRing> elvenRings = ring.getElvenRings();
32         assertTrue(elvenRings.size() == 3);
33     }
34
35 }
36
```

Problems | Console | History | Search | JUnit

Finished after 0.039 seconds

Runs: 2/2 Errors: 1 Failures: 0

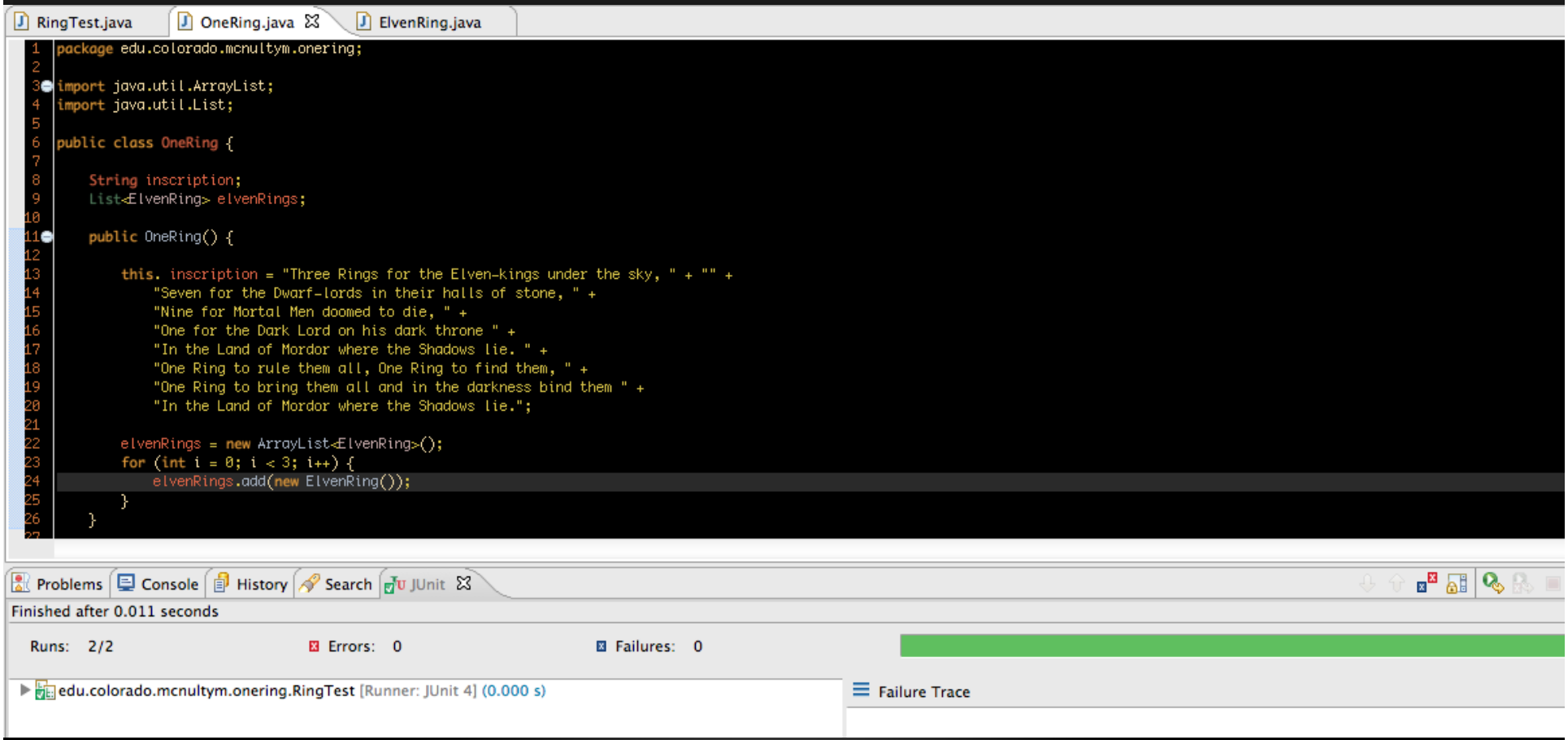
edu.colorado.mcnultym.onering.RingTest [Runner: JUnit 4] (0.000 s)
- ring_should_have_correct_inscription (0.000 s)
- ring_should_control_three_elven_rings (0.000 s)

Failure Trace

java.lang.NullPointerException

at edu.colorado.mcnultym.onering.RingTest.ring_should_control_three_elven_rings(RingTest.java:32)

MAKE IT PASS


RingTest.java
OneRing.java
ElvenRing.java
package edu.colorado.mcnultym.onering;
import java.util.ArrayList;
import java.util.List;
public class OneRing {
String inscription;
List<ElvenRing> elvenRings;
public OneRing() {
this. inscription = "Three Rings for the Elven-kings under the sky, " + "" +
"Seven for the Dwarf-lords in their halls of stone, " +
"Nine for Mortal Men doomed to die, " +
"One for the Dark Lord on his dark throne " +
"In the Land of Mordor where the Shadows lie. " +
"One Ring to rule them all, One Ring to find them, " +
"One Ring to bring them all and in the darkness bind them " +
"In the Land of Mordor where the Shadows lie.";
elvenRings = new ArrayList<ElvenRing>();
for (int i = 0; i < 3; i++) {
elvenRings.add(new ElvenRing());
}
}
Problems
Console
History
Search
JUnit
Finished after 0.011 seconds
Runs: 2/2
Errors: 0
Failures: 0
edu.colorado.mcnultym.onering.RingTest [Runner: JUnit 4] (0.000 s)
Failure Trace

REFACTOR MERCILESSLY

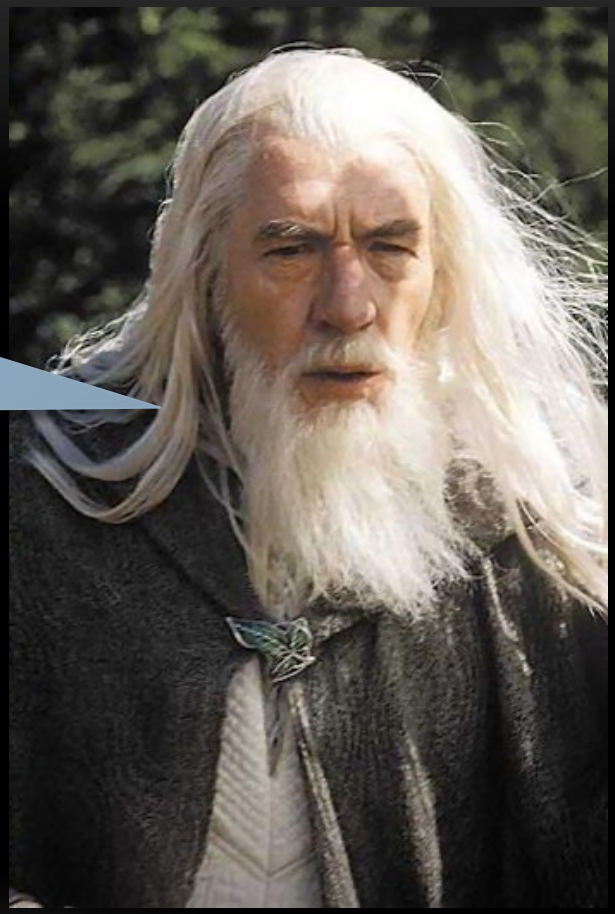
- Notice that both of the tests start by instantiating a new OneRing object
- Not only is this problematic, because it is the One Ring, not the Many Rings, but it also looks a lot like duplicate code
- Fear not, however, because the @Before annotation comes to the rescue
- This method creates a new OneRing at the beginning of the test that can be used for all the subsequent tests

SETUP METHOD WITH @BEFORE ANNOTATION

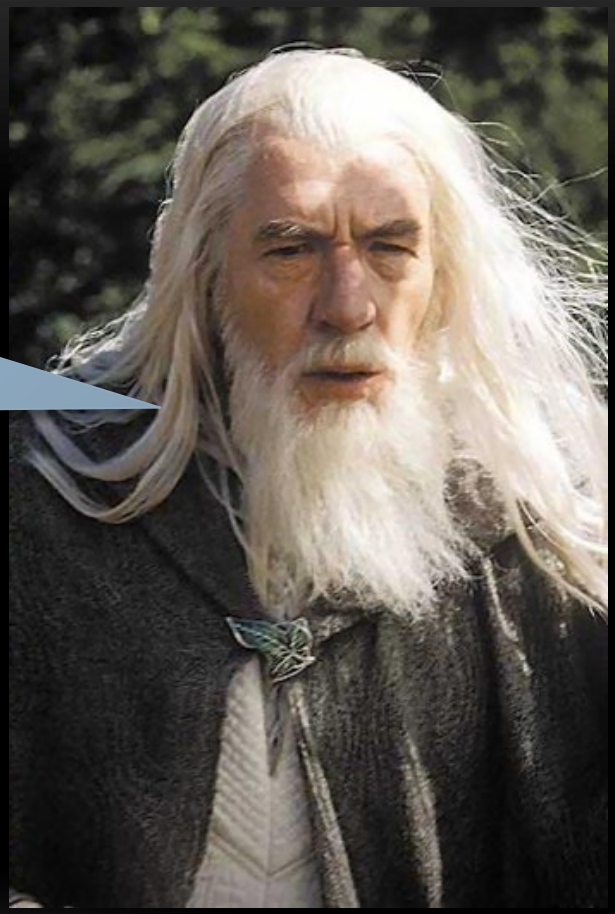
```
 9  public class RingTest {
10
11      private OneRing ring;
12
13      @Before
14      public void forge_ring() {
15          this.ring = new OneRing();
16      }
17
18      @Test
19      public void ring_should_have_correct_inscription() {
20          String inscription = ring.getInscription();
21
22          String testInscription = "Three Rings for the Elven-kings under the sky, " + "" +
23                  "Seven for the Dwarf-lords in their halls of stone, " +
24                  "Nine for Mortal Men doomed to die, " +
25                  "One for the Dark Lord on his dark throne " +
26                  "In the Land of Mordor where the Shadows lie. " +
27                  "One Ring to rule them all, One Ring to find them, " +
28                  "One Ring to bring them all and in the darkness bind them " +
29                  "In the Land of Mordor where the Shadows lie.";
30
31          assertTrue(inscription.equals(testInscription));
32      }
33
34      @Test
35      public void ring_should_control_three_elven_rings() {
36          List<ElvenRing> elvenRings = ring.getElvenRings();
37          assertTrue(elvenRings.size() == 3);
38      }
39
40  }
41
```

This is all a lot of fun, but so far all our refactoring has been of the test code itself. Is this normal?

The refactoring of the production code is perhaps the most important part, but so far we haven't needed to.



Let’s skip ahead to some hours later, after we have dutifully test driven the design of the other rings.

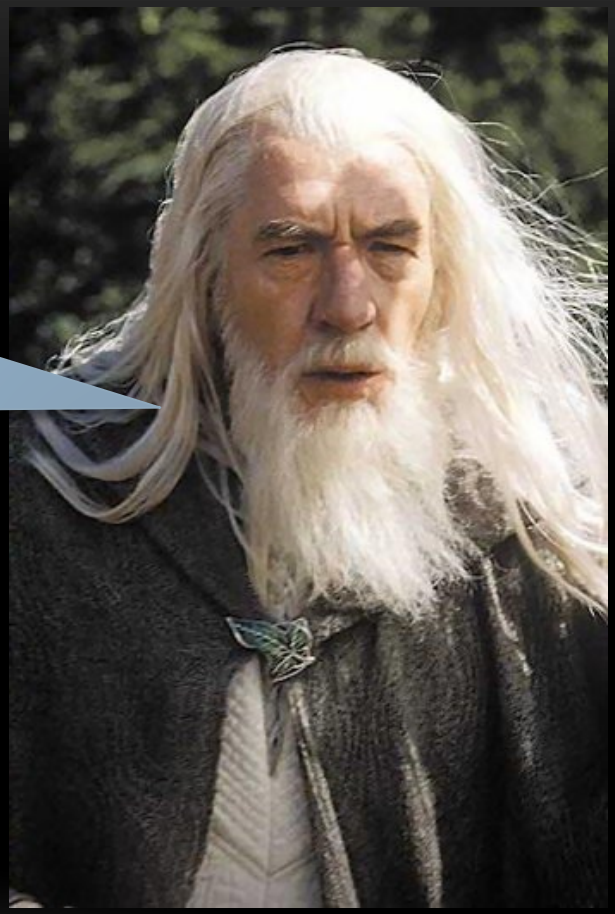


ALL THE TESTS PASS, BUT WHAT A MESS!

```java
package edu.colorado.mcnultym.onering;

import java.util.ArrayList;
import java.util.List;

public class OneRing {

    String inscription;
    List<ElvenRing> elvenRings;
    List<DwarvenRing> dwarvenRings;
    List<HumanRing> humanRings;

    public OneRing() {

        this. inscription = "Three Rings for the Elven-kings under the sky, " + "" +
            "Seven for the Dwarf-lords in their halls of stone, " +
            "Nine for Mortal Men doomed to die, " +
            "One for the Dark Lord on his dark throne " +
            "In the Land of Mordor where the Shadows lie. " +
            "One Ring to rule them all, One Ring to find them, " +
            "One Ring to bring them all and in the darkness bind them " +
            "In the Land of Mordor where the Shadows lie.";

        elvenRings = new ArrayList<ElvenRing>();
        for (int i = 0; i < 3; i++) {
            elvenRings.add(new ElvenRing());
        }

        dwarvenRings = new ArrayList<DwarvenRing>();
        for (int i = 0; i < 7; i++) {
            dwarvenRings.add(new DwarvenRing());
        }

        humanRings = new ArrayList<HumanRing>();
        for (int i = 0; i < 9; i++) {
            humanRings.add(new HumanRing());
        }
    }

    public String getInscription() {
        return inscription;
    }

    public List<ElvenRing> getElvenRings() {
        return elvenRings;
    }

    public List<DwarvenRing> getDwarvenRings() {
        return dwarvenRings;
    }
```

This is terrible object oriented code! I thought TDD was supposed to give us good, clean code!

That's why the refactor step is so important! Let's get rid of that duplicate code!



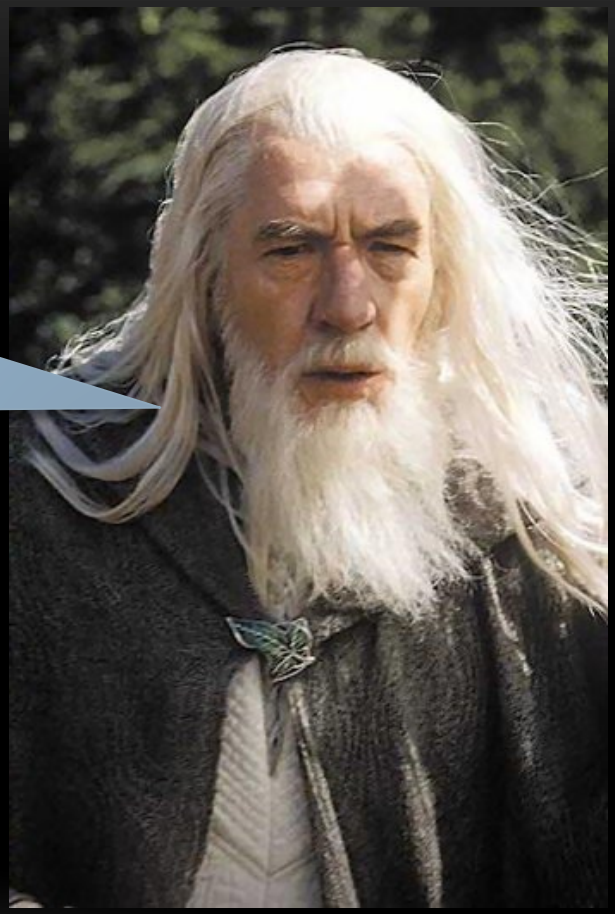
CLEANER CONSTRUCTOR

- We really don't need all the ring creation steps to take place in the constructor, so let's put that stuff in its own separate function.

```
public OneRing() {

    this. inscription = "Three Rings for the Elven-kings under the sky, " + "" +
        "Seven for the Dwarf-lords in their halls of stone, " +
        "Nine for Mortal Men doomed to die, " +
        "One for the Dark Lord on his dark throne " +
        "In the Land of Mordor where the Shadows lie. " +
        "One Ring to rule them all, One Ring to find them, " +
        "One Ring to bring them all and in the darkness bind them " +
        "In the Land of Mordor where the Shadows lie.";

    gainControlOfRings();
}

private void gainControlOfRings() {
    elvenRings = new ArrayList<ElvenRing>();
    for (int i = 0; i < 3; i++) {
        elvenRings.add(new ElvenRing());
    }

    dwarvenRings = new ArrayList<DwarvenRing>();
    for (int i = 0; i < 7; i++) {
        dwarvenRings.add(new DwarvenRing());
    }

    humanRings = new ArrayList<HumanRing>();
    for (int i = 0; i < 9; i++) {
        humanRings.add(new HumanRing());
    }

}
```

Well, this is better, but all those for loops look awfully similar. And also, the three different ring classes are really similar too!

You learn quickly! Yes, there's a lot of duplicate code here, so let's think of a way to clean it up.



THE RING OF POWER CLASS

- By creating an abstract RingOfPower class, we are able to remove a lot of duplicate code.

```java
package edu.colorado.mcnultym.onering;

public abstract class RingOfPower {

    private String message = "Sauron rules me!";

    protected String getMessage() {
        return this.message;
    }

    public abstract void printMessage();

}
```

```java
public class ElvenRing extends RingOfPower {

    @Override
    public void printMessage() {
        System.out.println("I'm an elven ring! I'm pretty and fair!");
        System.out.println(this.getMessage());
    }

}
```

```java
public class DwarvenRing extends RingOfPower{

    @Override
    public void printMessage() {
        System.out.println("I'm a dwarven ring! You should go mine some shiny objects now!");
        System.out.println(this.getMessage());
    }

}
```

```java
public class HumanRing extends RingOfPower {

    @Override
    public void printMessage() {
        System.out.println("I'm a human ring! I corrupt the souls of men!");
        System.out.println(this.getMessage());
    }

}
```

NEW AND IMPROVED RING COLLECTION

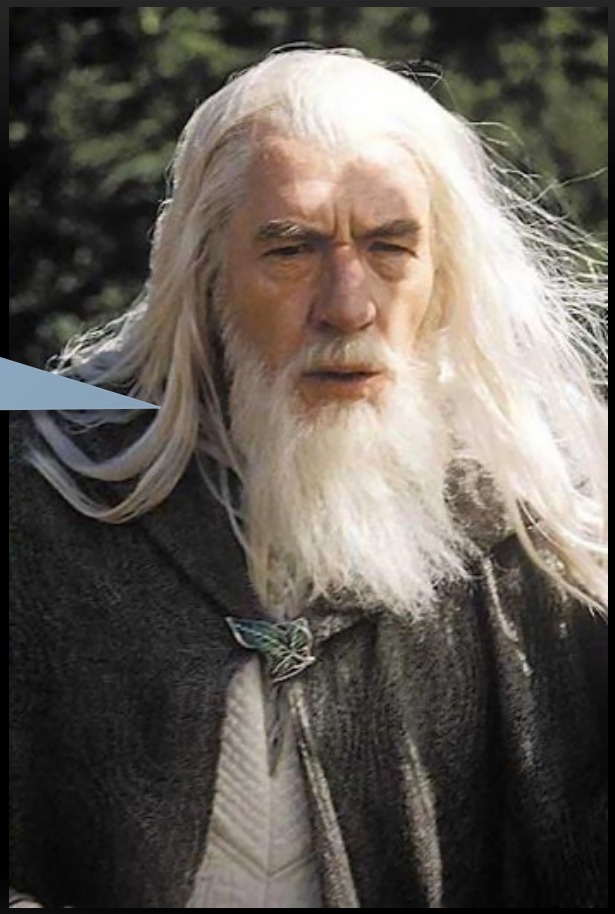
- Now we can remove the duplication in the OneRing class

```
        gainControlOfRings();
    }

    private void gainControlOfRings() {
        elvenRings = new ArrayList<RingOfPower>();
        dwarvenRings = new ArrayList<RingOfPower>();
        humanRings = new ArrayList<RingOfPower>();

        ElvenRing elvenRingTemplate = new ElvenRing();
        DwarvenRing dwarvenRingTemplate = new DwarvenRing();
        HumanRing humanRingTemplate = new HumanRing();

        controlRings(elvenRings, 3, elvenRingTemplate);
        controlRings(dwarvenRings, 7, dwarvenRingTemplate);
        controlRings(humanRings, 9, humanRingTemplate);

    }

    private void controlRings(List<RingOfPower> rings, int numberToControl, RingOfPower ringToAdd) {
        for (int i = 0; i < numberToControl; i++) {
            rings.add(ringToAdd);
        }
    }
```

Wow, that is a lot better! Thank Morgoth for refactoring!

Now for the final phase of my plan … Ruling them All!

All right, let's do this one more time!



RED

```
51
52      @Test
53      public void ring_should_rule_all_rings_of_power() {
54          List<RingOfPower> all = ring.getAllRings();
55          ring.rule(all);
56          for (RingOfPower r : all) {
57              assertTrue(r.ruled());
58          }
59      }
60
61  }
62
```

Problems | Console | History | Search | JUnit

Finished after 0.08 seconds

Runs: 5/5 Errors: 1 Failures: 0

edu.colorado.mcnultym.onering.RingTest [Runner: JUnit 4] (0.001 s)

ring_should_have_correct_inscription (0.001 s)

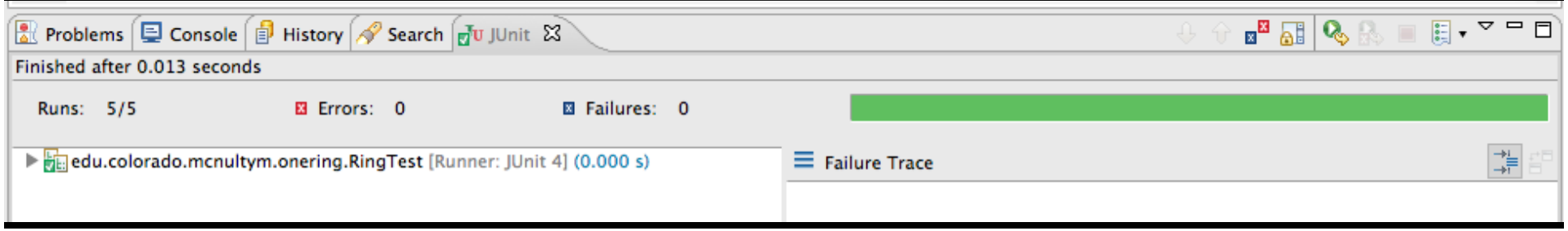
Failure Trace

java.lang.NullPointerException

GREEN

```
63
64      public List<RingOfPower> getAllRings() {
65          List<RingOfPower> allRings = new ArrayList<RingOfPower>();
66          allRings.addAll(elvenRings);
67          allRings.addAll(dwarvenRings);
68          allRings.addAll(humanRings);
69
70          return allRings;
71      }
72
73      public void rule(List<RingOfPower> all) {
74          for (RingOfPower ring : all) {
75              ring.setRuled(true);
76          }
77      }
78
```

```
1   package edu.colorado.mcnultym.onering;
2
3   public abstract class RingOfPower {
4
5       private boolean ruled = false;
6
7       protected String getMessage() {
8           if (!this.ruled) {
9               return "I am a free ring!";
10          } else {
11              return "I am ruled by Sauron!";
12          }
13      }
14
15      public abstract void printMessage();
16
17      public void setRuled(boolean ruled) {
18          this.ruled = ruled;
19          this.printMessage();
20      }
21
22      public boolean ruled() {
23          return this.ruled;
24      }
25
26  }
27
```



REFACTOR?

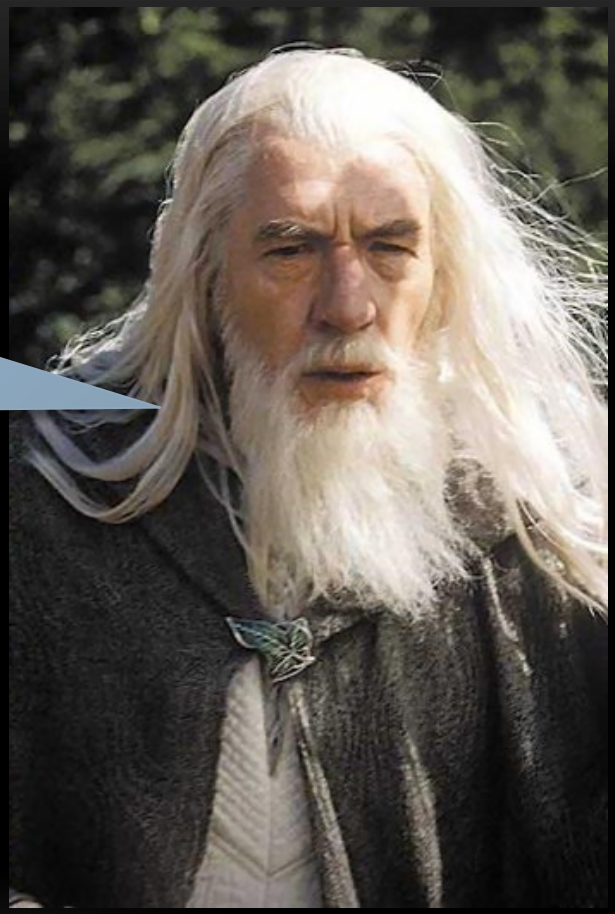
Woo hoo! Look at my ring! It rules them all! … I guess I could still use to do some more refactoring, though.

Problems | Console | History | Search | JUnit

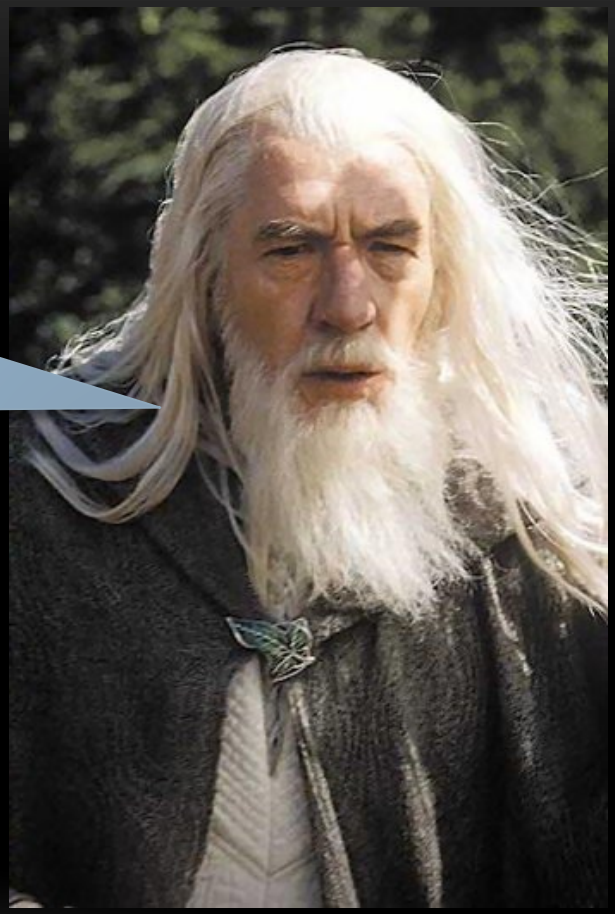
<terminated> RingTest [JUnit] /System/Library/Frameworks/JavaVM.frame

```
I'm an elven ring! I'm pretty and fair!
I am ruled by Sauron!
I'm an elven ring! I'm pretty and fair!
I am ruled by Sauron!
I'm an elven ring! I'm pretty and fair!
                                        ine some shiny objects now!
                                        hiny objects now!
                                        objects now!
                                        jects now!
                                        jects now!
                                        objects now!
                                        shiny objects now!
                                        ouls of men!
I'm a human ring! I corrupt the souls of men!
I am ruled by Sauron!
I'm a human ring! I corrupt the souls of men!
I am ruled by Sauron!
I'm a human ring! I corrupt the souls of men!
I am ruled by Sauron!
I'm a human ring! I corrupt the souls of men!
I am ruled by Sauron!
I'm a human ring! I corrupt the souls of men!
I am ruled by Sauron!
I'm a human ring! I corrupt the souls of men!
I am ruled by Sauron!
I'm a human ring! I corrupt the souls of men!
I am ruled by Sauron!
I'm a human ring! I corrupt the souls of men!
I am ruled by Sauron!
I'm a human ring! I corrupt the souls of men!
I am ruled by Sauron!
```

There is probably always more refactoring to do, but it looks like you're on a good path!



Oh wait, did you say rule ALL the rings? Well, that was a mistake. Great, now I'd better go find some hobbits…



CONCLUSION

- Test driven development leads to cleaner code
- By writing tests before writing production code, we have a clearer idea of requirements and only create code that fits within the requirements – no extra bloat
- The refactoring step ensures duplicate code is removed, and that it is done often enough that the amount of duplicate code never gets out of hand
- In other words, TDD rules (them all)!

FURTHER READING

- If you liked this presentation and would like to learn more, please check out the following very excellent books: