Object Oriented Programming with Java

Testing & Debugging

Testing

Goals of Testing

- "The goal of testing is just to find errors" – WRONG!
- The goal of testing is to gain assurance about the quality of your programs
- The testing you do for your class assignments is more likely to be debugging oriented, but this is not always the case

Testing

- Testing can mean many different things:
  → running a program on various inputs
  → any human or computer assessment of quality
  → evaluations before writing code

- The earlier we find an problem, the easier and cheaper it is to fix

- When are we done testing?

Software Testing

- The earlier you find a defect, the cheaper it is to fix
- It is up to you to figure out how much testing is enough and when

		Time detected				
		Requirements	Architecture	Construction	System test	Post-release
Time introduced	Requirements	1×	3×	5–10×	10×	10–100×
	Architecture	-	1×	10×	15×	25–100×
	Construction	-	-	1×	10×	10–25×

Facts about testing

- Maintenance activities consume ~70% of the total cost of software
  - In real world we spend most of our time testing and debugging code.

- Some software engineering approaches advocate writing test cases BEFORE you even write any code
  - What does this accomplish?

Software Testing

- Modern day testing is preventive

- The testing you do for your class assignments is more likely to be debugging oriented

  → How do you catch bugs?

  → How do you design a good test case?

Reviews

- A ***review*** is a meeting in which several people examine a design document or section of code
- It is a common and effective form of human-based testing
- Presenting a design or code to others:
  - makes us think more carefully about it
  - provides an outside perspective
- Reviews are sometimes called *inspections* or *walkthroughs*

© 2004 Pearson Addison-Wesley. All rights reserved

Test Cases

- A ***test case*** is a set of input and user actions, coupled with the expected results
- Often test cases are organized formally into *test suites* which are stored and reused as needed (like JUnit)
- For medium and large systems, testing must be a carefully managed process
- Many organizations have a separate Quality Assurance (QA) department to lead testing efforts

© 2004 Pearson Addison-Wesley. All rights reserved

Defect and Regression Testing

- **Defect testing** is the execution of test cases to find errors. The act of fixing an error may introduce new errors
- After fixing a set of errors we should perform **regression testing** – running previous test suites to ensure new errors haven't been introduced
- It is not possible (in general) to create test cases for all possible input and user actions
- Therefore we should design tests to maximize their ability to find problems

© 2004 Pearson Addison-Wesley. All rights reserved

Creating Test Cases

- Common behavior (expected inputs / usage)
- Uncommon behavior (border cases, very complex inputs)

- How do you identify such properties?
  → Black box testing – not looking at the code, just intent
  → White box testing – looking at the code/implementation
  → Practice, looking at other projects

Black-Box Testing

- In **black-box testing**, test cases are developed without considering the internal logic
- They are based on the input and expected output
- Input can be organized into **equivalence categories**. Two input values in the same equivalence category would produce similar results. Therefore a good test suite will cover all equivalence categories and focus on the boundaries between categories

© 2004 Pearson Addison-Wesley. All rights reserved

White-Box Testing

- **White-box testing** focuses on the internal structure of the code
- The goal is to ensure that every path through the code is tested
- Paths through the code are governed by any conditional or looping statements in a program
- A good testing effort will include both black-box and white-box tests

© 2004 Pearson Addison-Wesley. All rights reserved

Limits of Testing

Let's imagine you could, in theory, test your code on every possible combination of input and output

- Would your code be guaranteed to be correct?

- No! Perhaps you wrote correct code but it doesn't do what the customer asked. Probably because the customer wasn't clear, or they didn't know what they wanted. This happens all the time

→ defining what correctness means is a tough problem!

JUnit Testing Framework

JUnit: Unit Testing for Java

Unit Testing: Providing individual test cases that target specific **units** of code, usually a specific method or a specific branch of code.

JUnit: package that helps you write unit tests and execute them.

Offers a form of regression testing: make a few changes, re-run all JUnit tests to check if we broke previously-tested code

JUnit: Unit Testing for Java

We write test methods that call methods such as:

```
assertTrue(boolExpr)
assertEquals(Object expected, Object actual)
assertNotNull(Object o)
fail()
```

Integrates very well with IDEs (also works on command-line)

Java assertions

`assertEquals(expected, actual)`	Checks if the two are equal
`assertTrue(condition)`	Checks if the condition is true
`assertFalse(condition)`	Checks if the condition is false
`assertNotNull(object)`	Checks that an object isn't null
`assertNull(object)`	Checks that an object is null
`assertSame(object1, object2)`	Checks if two object references point to the same object
`assertNotSame(object1, object2)`	Checks if two object references do not point to the same object
`assertArrayEquals(expectedArray, resultArray)`	Checks whether two arrays are equal to each other

If an assert isn't convenient, you can also use `if` statements combined with the `fail()` method

Junit: Example of a class to be tested

```
public class Triangle {
   public int side1, side2, side3;
   public Triangle(int side1, int side2, int side3) {
      this.side1 = side1;
      this.side2 = side2;
      this.side3 = side3;
   }
   //is this area formula correct? We should test it...
   public double area () {
      //Heron's Formula for area of a triangle
      double s = (side1 + side2 + side3)/2;
      return Math.sqrt(s*(side1-s)*(side2-s)*(side3-s));
   }
}
```

Junit: Example of a tester class

```java
import org.junit.*;
import static org.junit.Assert.*;


public class TriangleTester {
  public static void main(String args[]) {
    org.junit.runner.JUnitCore.main("TriangleTester");
  }
// start adding tests here...
}
```

This is the typical boilerplate you can use to start writing a unit tester.

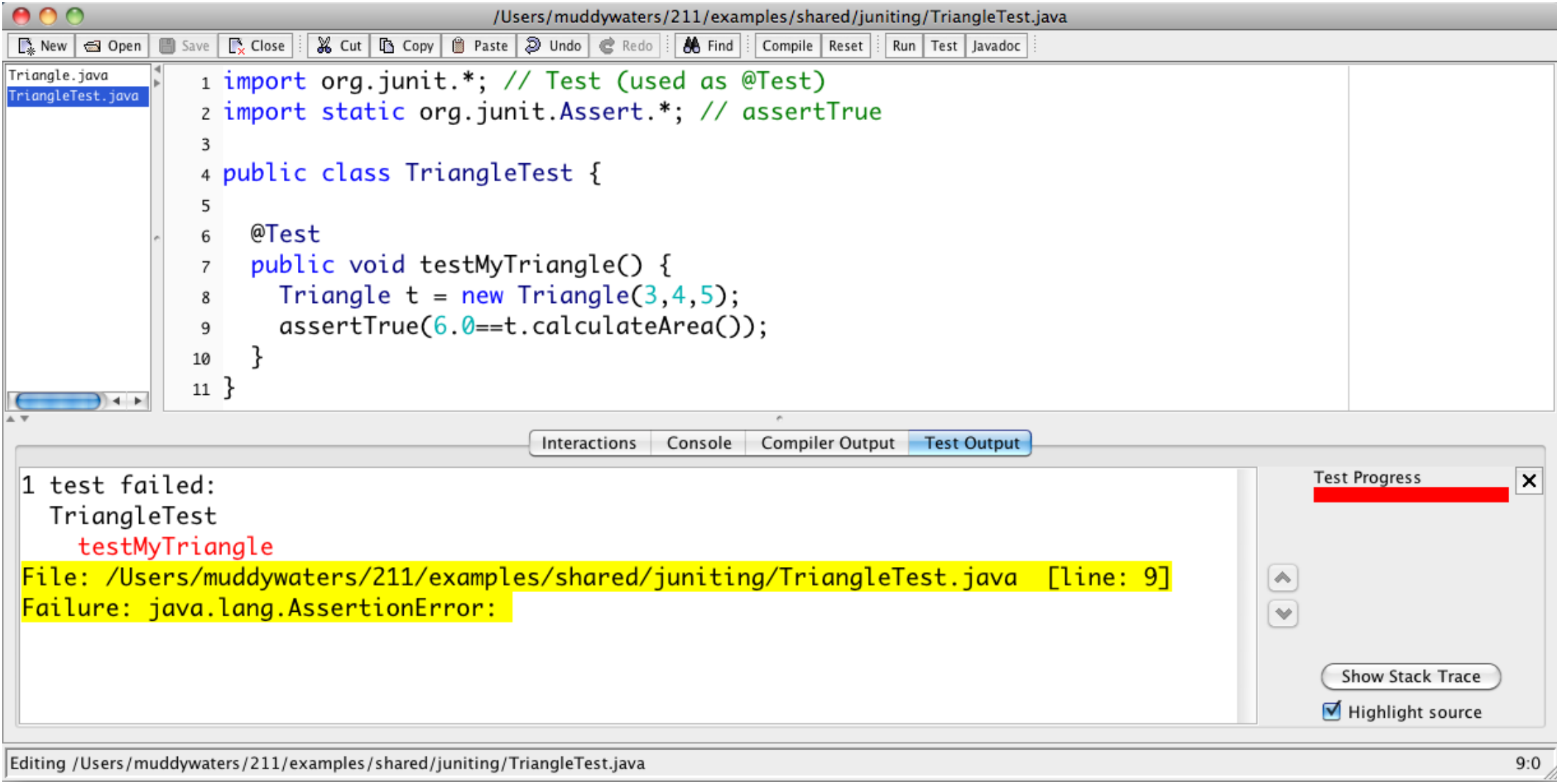
This code does **not** contain any tests yet, just the foundation.

Compile and run with the following commands in the terminal

```
> javac -cp .:junit.jar Triangle.java TriangleTester.java
> java -cp .:junit.jar TriangleTester
```

JUnit Example – DrJava

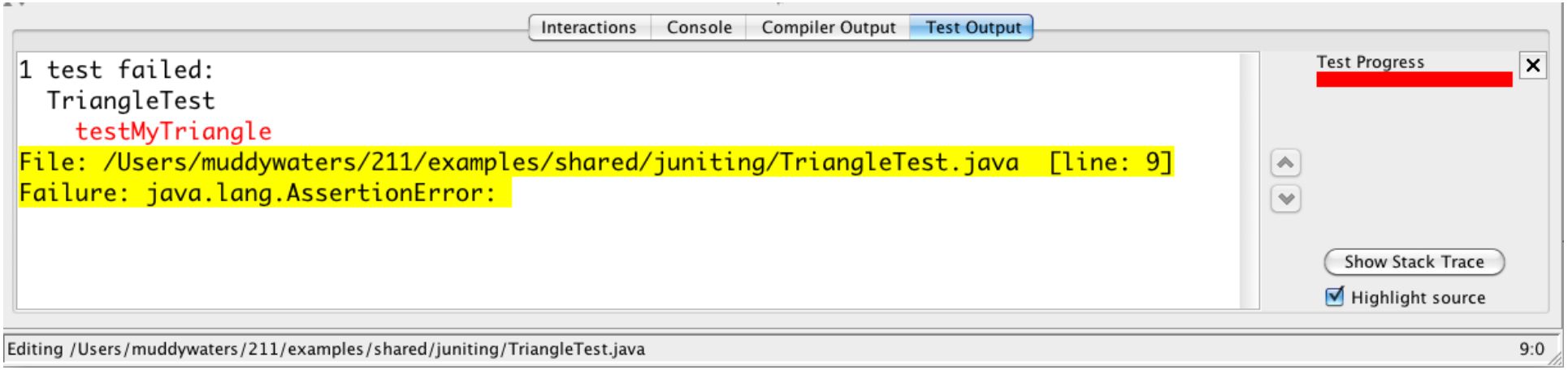
- Create test case → assert something → run the test



JUnit: Running Tests – DrJava

We can run our JUnit tests – just click the "Test" button with your testing class.
→ get results (green=good, red=bad), and see a trace of where the failing test failed.

Here, our assertion of equality being true failed. Time to go double check Heron's Formula as we've implemented it!

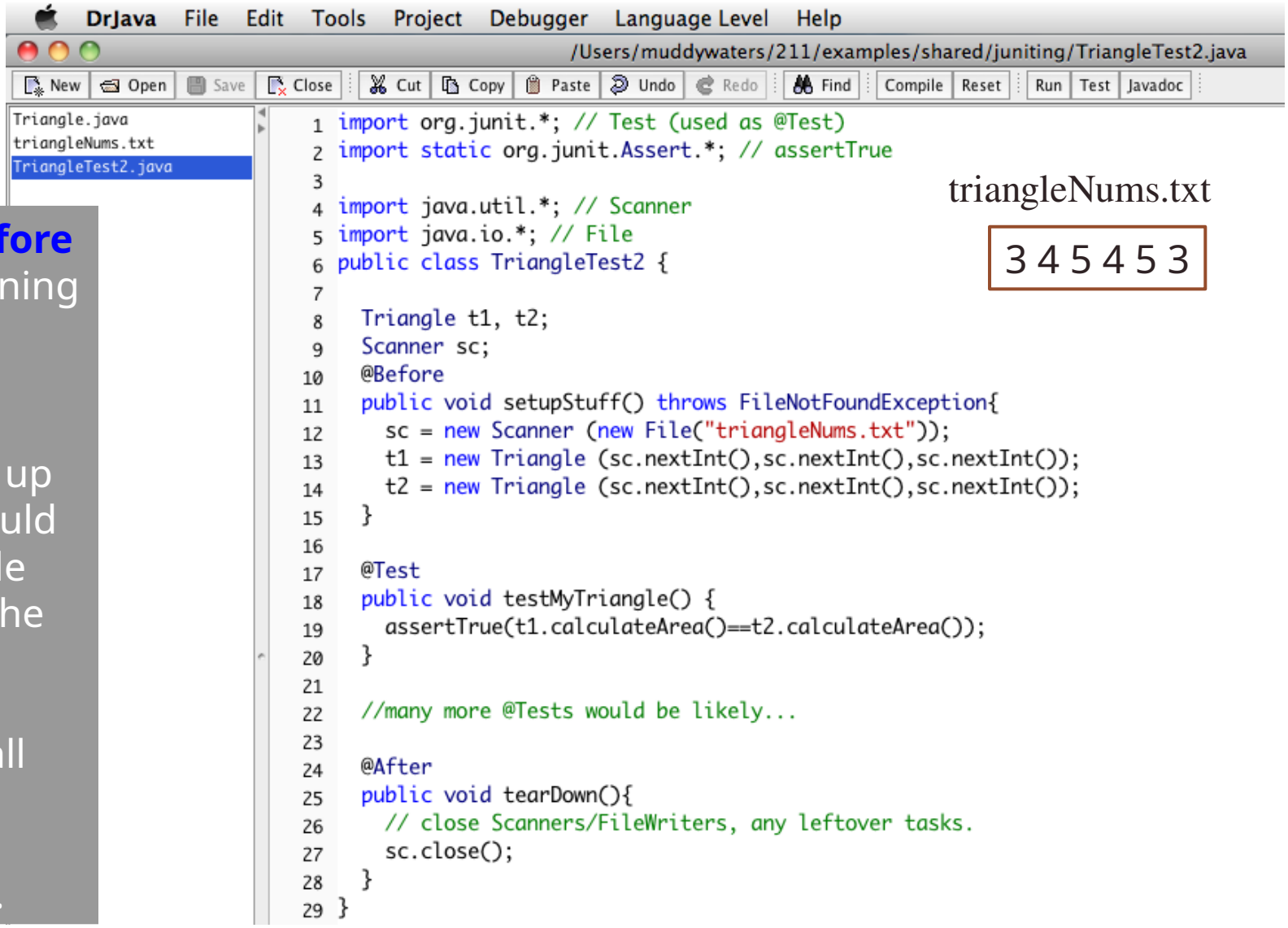


JUnit: @Before and @After

JUnit runs the **@Before** method before running all of our **@Test** methods

→ great for setting up our data values: Could use value in multiple @Test methods in the class

@After runs after all tests. Good for clearing memory, closing buffers, etc.

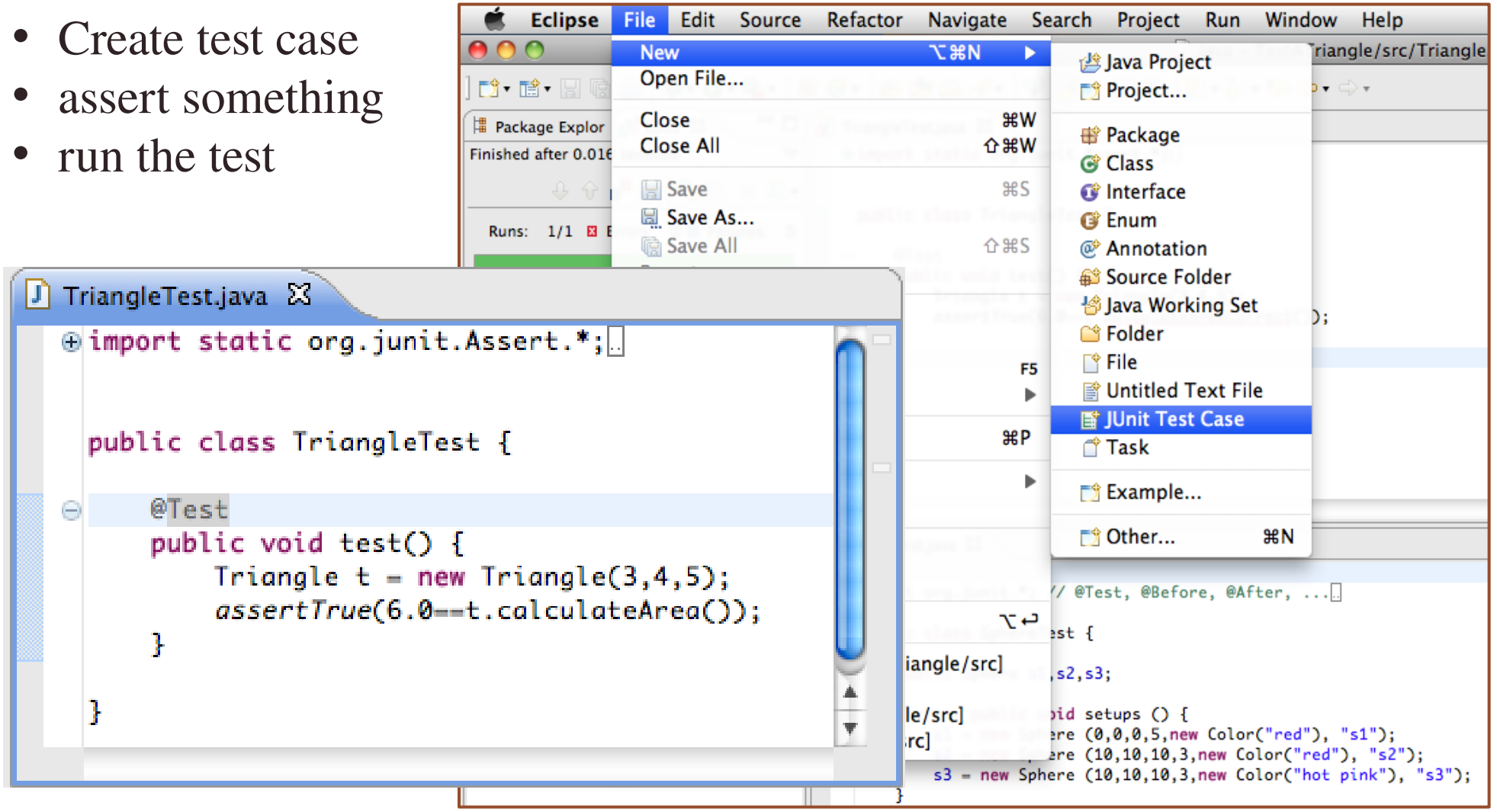


triangleNums.txt

3 4 5 4 5 3

JUnit Example – Eclipse

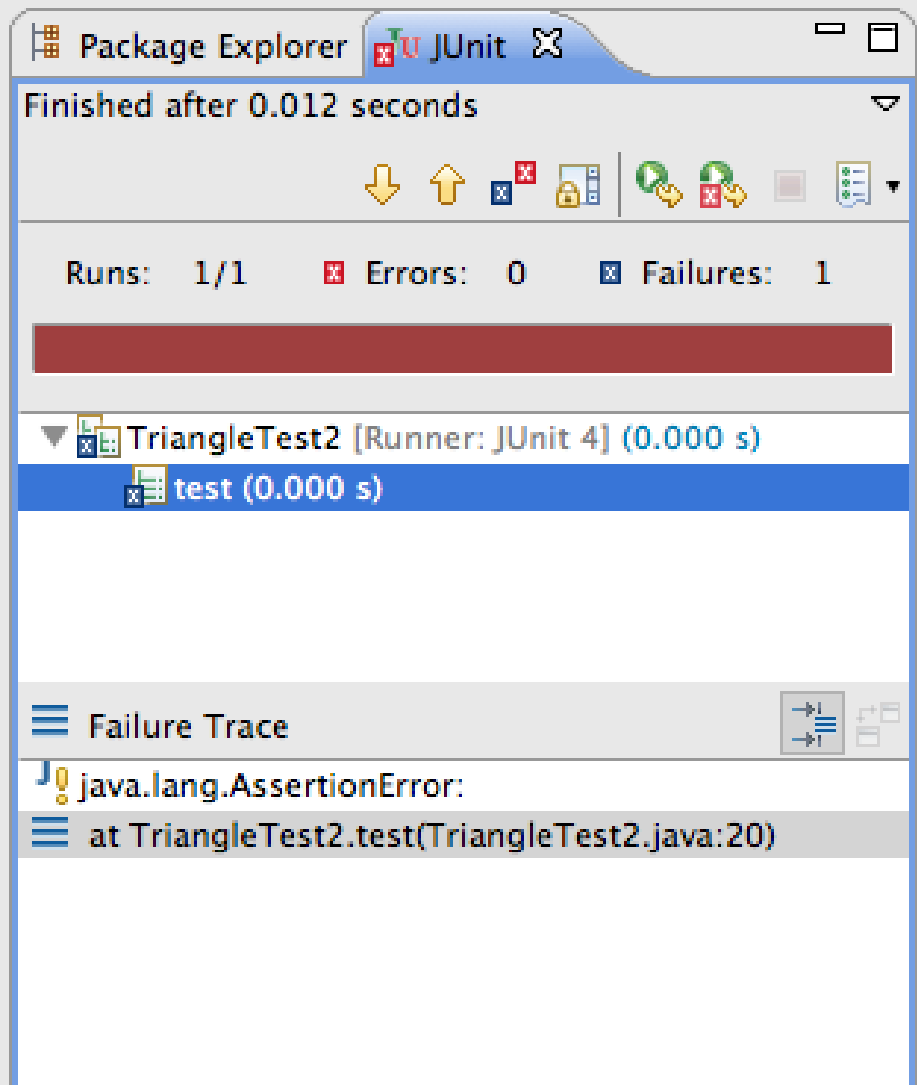
- Create test case
- assert something
- run the test



JUnit: Running Tests – Eclipse

We can run our JUnit tests, get results (green=good, red=bad), and even look at a trace of where the failing test failed.

Here, our assertion of equality being true failed. Time to go double check Heron's Formula as we've implemented it!



JUnit annotations

Annotation	Description
@Test	This annotation is a replacement of org.junit.TestCase which indicates that public void method to which it is attached can be executed as a test Case.
@Before *renamed @BeforeEach in Junit 5*	This annotation is used if you want to execute some statement such as preconditions before each test case.
@BeforeClass *renamed @BeforeAll in Junit 5*	This annotation is used if you want to execute some statements before all the test cases for e.g. test connection must be executed before all the test cases.
@After *renamed @AfterEach in Junit 5*	This annotation can be used if you want to execute some statements after each Test Case for e.g resetting variables, deleting temporary files ,variables, etc.
@AfterClass *renamed @AfterAll in Junit 5*	This annotation can be used if you want to execute some statements after all test cases for e.g. Releasing resources after executing all test cases.
@Ignores	This annotation can be used if you want to ignore some statements during test execution for e.g. disabling some test cases during test execution.
@Test(timeout=500)	This annotation can be used if you want to set some timeout during test execution for e.g. if you are working under some SLA (Service level agreement), and tests need to be completed within some specified time.
@Test(expected=RuntimeException.class)	This annotation can be used if you want to handle some exception during test execution. For, e.g., if you want to check whether a particular method is throwing specified exception or not.
@Test(timeout=500, expected=Exception.class)	You can combine two directives

Debugging

What is a Debugger?

Debugger is an interactive tool that allows us to slowly trace through a running program, inspecting values, modifying values, and generally observing the execution of the program.

→ doesn't modify the source code

→ directly shows what memory is in variables

It is a ***valuable alternative to inserting print statements***, re-running the code, and seeing what printed out.

→ we can't insert extra print statements once it begins running

→ the print statements change the meaning of the code (bad)

Usage: Stepping

- **Breakpoint:** a location in source code where the debugger must pause when reached. Then stepping is allowed.
- **Step:** evaluate the next statement; pause.
- **Step into:** take a step. if a method is called, continue stepping through its code
- **Step over:** take a step. if a method is called, completely call/return from it as a single action.
- **Step out:** take a step. if a method is called, run code until a return statement, and pause at that returned-to site (if other breakpoints reached, pause no matter what)

Debugging with DrJava

Debugging is built in

To set a breakpoint:
→ put cursor on the line and press Ctrl-B (⌘-B)
→ debugger pauses **before** running the statement

To run:
→ enable **Debugger → Debug Mode** (⌘-Shift-D)

→ run as normal; breakpoints will trigger

→ choose to step in various ways, or resume

Watches / Watchlist

A variable of interest (one we want to watch) may be added to a watch list.

- → each time the debugger pauses, each watched value is re-inspected and displayed
- → used for local variables, fields, etc.

An interactive interpreter may allow typing in other expressions to see their values/effect.

DrJava and Watchlists

With Debug Mode enabled, the Debugging Panel is visible.

→ Watches area (very useful!)

→ Stacks/Threads (we won't be looking here)

Type-in variables that you'll want to track

→ click in upper-leftmost cell under "Name" to type new entry

→ might only be in scope during part of execution; that's ok.

Modifying Values Mid-execution

Debuggers often allow you to directly modify values.

→ explore "what-if" scenarios

→ quickly test different parts of code

Can't modify source code while it runs (at least, won't see those changes until the next run/debugging).

Effort of modifying values isn't remembered in later runs/debugging.

While debugging, and paused:

→ type assignment statements into the Interactions Panel.

→ watch-list might not show change until next step is taken.

Other IDEs

Most heavy-weight IDEs (e.g. Eclipse, jGrasp, IntelliJ IDEA, JSwat, etc.) will have debuggers of various design

Many languages have debuggers available.

→ you'll experience the gdb (GNU debugger) for C at some point

Not all debuggers are built into an IDE

→ we have command-line debuggers, too!

No matter what, the same principles apply: set breakpoints, watch values, inspect/change values as it runs, step into/over/out, etc.