

## Chapter 2: Testing

### Data Structures in Java: From Abstract Data Types to the Java Collections Framework

by Simon Gray



Copyright © 2007 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

---

---

---

---

---

---

---

---

### Data

- Commercial Software contains between 6 and 30 faults per 1000 lines of code (kloc).
- This has been consistently reported from the industry for 20 years.

Hatton (1998) *Does OO Sync with how we think?*  
**IEEE Software**

Copyright © 2007 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

---

---

---

---

---

---

---

---

Product	Size
Windows NT	50M LOC
Linux Kernel	1.5M LOC
Mozilla	2M LOC

Copyright © 2007 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

---

---

---

---

---

---

---

---

### DoD Data

- The DoD reports defect rates of 5-15 faults per kloc.

Copyright © 2007 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

---

---

---

---

---

---

---

### Defects

1. The later in the life cycle that an error is detected the more expensive it is to repair.
2. Typically errors remain latent and are not detected until well after the stage at which they are made.
  - 54% of errors detected after coding and unit testing.
  - 45% of these errors were requirements and design errors.

Copyright © 2007 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

---

---

---

---

---

---

---

### Economics

1.25 hr (average) to find source of failure  
2-9 hrs to repair

A good program (5 defects/kloc) requires between 16.25 and 51.25 hours to clean every kloc (assuming no new errors are introduced).

Copyright © 2007 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

---

---

---

---

---

---

---

- “About 40-50% of the effort on current software projects is spent on avoidable rework.”
- “About 80% of the avoidable rework comes from 20% of the defects.”

Copyright © 2007 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

---

---

---

---

---

---

---

---

- “Peer reviews catch 60% of the defects.”
- “Disciplined personal practices can reduce defect introduction rates by up to 75%.”

Copyright © 2007 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

---

---

---

---

---

---

---

---

- Finding and fixing a *severe* software problem after delivery is often 100 times more expensive than finding and fixing it during the requirements and design phase.
- Finding and fixing *non-severe* software defects after delivery is about twice as expensive as finding these defects pre delivery.

Copyright © 2007 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

---

---

---

---

---

---

---

---

### Check the Specification

- Verify that the ADT's API contains all the operations clients will need and that the names are consistent and meaningful
- Verify that the specification is internally consistent. Ensure operation pre- and post-conditions are not in conflict with ADT invariants
- Design a test plan based on the behavior described in the ADT

Copyright © 2007 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

---

---

---

---

---

---

---

### Testing in the Life-Cycle

Copyright © 2007 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

---

---

---

---

---

---

---

### Objectives of Testing

Testing cannot show the absence of defects, it can only show that defects are present.

Copyright © 2007 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

---

---

---

---

---

---

---

1. Testing is a process of executing a program with the intent of finding an error.
2. A good test case is one that has a high probability of finding an as yet undiscovered error.
3. A successful test is one that uncovers an as yet undiscovered error.

Copyright © 2007 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

---

---

---

---

---

---

---

---

### Unit and Black-box Testing

- Treat a class and its methods as black boxes
  - For a given input you know *what* should be output by the “box”, but you don’t *how* it was generated
- An **Oracle** (test case) “predicts” what *should* be produced. It consists of four columns

Format for an oracle

Operation	Purpose	Object State	Expected Result
A call to an operation of the ADT to be tested.	The role the operation plays in the test case.	The expected state of the test object once the operation is complete.	The expected output (if any).

Copyright © 2007 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

---

---

---

---

---

---

---

---

### Oracle Examples for Rectangle

Test Case 2.1 Instantiation of a Rectangle object using default values for the attributes

Operation	Purpose	Object State	Expected Result
Rectangle r1 = new Rectangle()	To create a rectangle using the default values.	<i>height = 1.0</i> <i>length = 1.0</i>	A new Rectangle object with default values for the attributes
r1.getLength()	To verify instantiation and accessor method.	Note how the accessor methods are used to verify state set by a constructor or changed by a mutator.	1.0
r1.getHeight()	To verify instantiation and accessor method.		

Test Case 2.2 Instantiation of a Rectangle object using legal, client-supplied values

Operation	Purpose	Object State	Expected Result
Rectangle r1 = new Rectangle(2.0, 3.0)	To create a rectangle with client-supplied values.	<i>height = 2.0</i> <i>length = 3.0</i>	A new Rectangle object with client-supplied values for the attributes
r1.getHeight()	To verify instantiation and accessor method.		2.0
r1.getLength()	To verify instantiation and accessor method.		3.0

Copyright © 2007 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

---

---

---

---

---

---

---

---

### Oracle Examples for Rectangle

Test Case 2.5 Mutator to change a Rectangle's length: legal input			
Operation	Purpose	Object State	Expected Result
Rectangle r1 = new Rectangle(1.0, 2.0)	Create a rectangle with client-supplied values	<i>length</i> = 1.0 <i>height</i> = 2.0	A new Rectangle object with client-supplied values for the attributes
r1.setLength(5)	Test mutator with legal input	<i>length</i> = 5.0 <i>height</i> = 2.0	
r1.getLength()			5.0

  

Test Case 2.6 Mutator to change a Rectangle's length: illegal input			
Operation	Purpose	Object State	Expected Result
Rectangle r1 = new Rectangle(1.0, 2.0)	Create a rectangle with client-supplied values	<i>length</i> = 1.0 <i>height</i> = 2.0	A new Rectangle object with client-supplied values for the attributes
r1.setLength(0)	Test mutator with illegal input		IllegalArgumentException

Copyright © 2007 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

---

---

---

---

---

---

---

---

---

---

### Clear-box Testing

- Once the internals of a method are known, additional tests might be suggested. That is, provide additional oracles based on what is known about the implementation
- Path coverage** – test all possible paths through the code (!!)

```

stmt0;
if ( a < 0 ) {
  stmt1;
  stmt2;
}
else {
  stmt3;
  stmt4;
}
stmt5;
```

Copyright © 2007 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

---

---

---

---

---

---

---

---

---

---

### Testing with JUnit

The green bar of happiness; all tests passed!

Names of the testing methods corresponding to the oracles you prepared. A green check mark means the test passed.

Copyright © 2007 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

---

---

---

---

---

---

---

---

---

---

## JUnit Basics

- **TestCase:** The class your test class should extend to get the JUnit support methods
- **Test fixture:** instance(s) of the class to be tested

```
import junit.framework.*;           // JUnit stuff
import junit.extensions.*;         // JUnit stuff
import gray.adts.shapes.Rectangle; // the class to test

public class RectangleTester extends TestCase {
    // To inherit the testing and test run methods we will need

    private Rectangle r1, r2; // test fixture; stores the Rectangle
                             // objects under test
}
```

Copyright © 2007 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

---

---

---

---

---

---

---

---

## JUnit Basics: setUp() and tearDown()

```
/* Called AUTOMATICALLY before EACH testXXX() method is run. */
protected void setUp() {
    r1 = new Rectangle();
    r2 = new Rectangle( 2.0, 3.0 );
}

/* Called AUTOMATICALLY after EACH testXXX() method is run. */
protected void tearDown() {
    r1 = null;
    r2 = null;
}
```

Make sure each test method starts with a clean copy of the test fixture.

This is the sequence of calls the JUnit framework does for you automatically for each test method that is invoked.

```
setUp();
testXXX(); // run the test
tearDown(); // do house cleaning
```

Copyright © 2007 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

---

---

---

---

---

---

---

---

## Coding an Oracle as a JUnit Test Method

```
/** methods provided by JUnit
 * Test Case 2.1: Verify that instantiation was done properly using
 * default values for the Rectangle's dimensions.
 */
public void testInstantiateDefault() {
    assertEquals( 1.0, r1.getLength() );
    assertEquals( 1.0, r1.getHeight() );
}

/**
 * Test Case 2.6: Verify mutator for length catches illegal input.
 */
public void testInstantiateDefault() {
    try {
        r1.setLength( 0 ); // 0 is illegal; exception should be thrown
        fail( "Should raise IllegalArgumentException" );
    } catch ( IllegalArgumentException e ) {
        assertTrue( true ); // expected an exception, so the test passes
    }
}
```

Copyright © 2007 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Remember: setUp() will have been called prior to each test method getting called, creating test object r1 anew for each test

---

---

---

---

---

---

---

---

## When a Test Fails

The screenshot shows the JUnit GUI with the following details:

- Test class name:** gray.shapes.RectangleTester
- Results:** Name: S5, Errors: 0, Failures: 1
- Failed Test:** testColorFromMethod(gray.shapes.RectangleTester) expected <23.0> but was <17.1>
- Stack Trace:** junit.framework.AssertionFailedError: expected <23.0> but was <17.1> at gray.shapes.RectangleTester.testColorFromMethod(RectangleTester.java:52) at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method) at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source) at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source) at java.lang.reflect.Method.invoke(Unknown Source)

Annotations in the image:

- A red bar at the top of the results section is labeled: "The red bar of sadness; some tests failed"
- The failed test name is labeled: "Names of the testing methods corresponding to the oracles you prepared. A red X means the test failed."
- The stack trace is labeled: "Stack trace telling what was expected, what was generated and where the test failed; very handy!"

Copyright © 2007 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

---

---

---

---

---

---

---

---