CS 635 Advanced Object-Oriented Design & Programming
Fall Semester, 2020
Doc 3 Dollar Words, Testing
Sep 1, 2020

Review

Object-Oriented Programming is good as it promotes

- Code reuse
- More readable code
- More maintainable code
- Better designs

Basic OO Heuristics

Keep related data and behavior in one place

A class should capture one and only one key abstraction

Beware of classes that have many accessor methods defined in their public interface

Dollar Words

Character value

a, A -> 1
b, B -> 2
...
z, Z -> 26

Word value

Sum of the character values in the word

cab -> 3 + 1 + 2 = 6

Dollar Word

Word value = 100

buzzy
nutty
arrowy
crusts
cutout
cutups
drossy
dryrot
envoys
flurry
grouts
growly
grumpy
dominates

draftsmen
driveling
dualities
ducklings
dumbfound
ebullient
ecstasies
ejections
electives
elephants
elsewhere
encumbers
energiser
Englewood
enticings
equalized
equipages

Problem

Given a string containing words find all the dollar words in the string

"This boulevard is in a status of tailspin!"   "boulevard", "status", "tailspin"

Approach One

What do we have to do

- Separate the words
- Compute the value of each word
- Select the words with value 100

Using Java 8

```
public class DollarWorld {
    Set<Character> wordSeparators =
            Collections.unmodifiableSet(Stream.of(',', '.', '?', ':', ';','!', ' ').collect(toSet()));

    public ArrayList<String> separateWords(String text) throws IOException {
        ArrayList<String> words = new ArrayList<String>();
        StringBuffer currentWord = new StringBuffer();
        StringReader textReader = new StringReader(text);
        int next;
        while (( next = textReader.read()) != -1) {
            char nextChar = (char) next;
            if (wordSeparators(nextChar)) {                    // Collect chars until find word separator
                words.add(currentWord.toString());
                currentWord = new StringBuffer();
            } else {
                currentWord.append(nextChar);
            }
        }
        if (currentWord.length() >0 ) {                        // when at end of text likely to have
            words.add(currentWord.toString());                 // word collected in current Word
        }
        return words;
    }
```

```
public ArrayList<String> dollarWordsIn(ArrayList<String> words) throws IOException {
    ArrayList<String> dollarWords = new ArrayList<String>();
   int currentWordValue = 0;

    for (String word:words) {
        currentWordValue = 0;
        StringReader wordChars = new StringReader(word.toLowerCase());
        int next;

       // Compute word value
        while ((next = wordChars.read() ) != -1) {
           char nextChar = (char)next;
            int charValue = nextChar - 'a' + 1;
            if ((charValue >0) && (charValue < 27))
                currentWordValue += charValue;
        }
        if (currentWordValue == 100)
              dollarWords.add(word);
    }
    return dollarWords;
}
```

```
public ArrayList<String> dollarWordsIn(ArrayList<String> words) throws IOException {
    ArrayList<String> dollarWords = new ArrayList<String>();
    ArrayList<Integer> wordValues = new ArrayList<Integer>();
    int currentWordValue = 0;

    for (String word:words) {
        currentWordValue = 0;
        StringReader wordChars = new StringReader(word.toLowerCase());
        int next;
        while ((next = wordChars.read() ) != -1) {              // Compute word value
            char nextChar = (char)next;
            int charValue = nextChar - 'a' + 1;
            if ((charValue >0) && (charValue < 27))
                currentWordValue += charValue;
        }
        wordValues.add(currentWordValue);
    }
    for(int k = 0; k < wordValues.size(); k++) {                // Select dollar words
        if (wordValues.get(k) == 100)
            dollarWords.add(words.get(k));
    }
    return dollarWords;
}
```

The Main Function

```
public ArrayList<String> dollarWordsIn(String text) throws IOException {
    return dollarWordsIn(separateWords(text));
}
```

```
@Test
void testDollarWords() throws IOException {
    String text = "This crusts is in a status of  truism!";
    List<String> dollarWords = new ArrayList<String>(Arrays.asList("crusts", "status", "truism"));
    DollarWorld testee = new DollarWorld();
    assertEquals(dollarWords, testee.dollarWordsIn(text));
}
```

Some More Tests

```
@Test
void separateWordsTest() throws IOException{
   ArrayList<String> correctAnswer = new ArrayList<String>(Arrays.asList("a", "b"));
   DollarWorld testee = new DollarWorld();
   assertEquals(correctAnswer, testee.separateWords("a b"));
}

@Test
void dollarWordsInTest() throws IOException {
   ArrayList<String> correctAnswer = new ArrayList<String>(Arrays.asList("status", "TRuISm"));
   ArrayList<String> input = new ArrayList<String>(Arrays.asList("foo", "status", "bar", "TRuISm"));
   DollarWorld testee = new DollarWorld();
   assertEquals(correctAnswer, testee.dollarWordsIn(input));
}
```

A Program in a Class

I wrote a program

Embedded it in a class

Issues

```
public class DollarWorld {
    Set<Character> wordSeparators =
            Collections.unmodifiableSet(Stream.of(',', '.', '?', ':', ';','!', ' ').collect(toSet()));
```

wordSeparators

Not part of the state of the object

Constant used in one method

Make static or put in the method

```
public class DollarWorld {
    static Set<Character> wordSeparators =
            Collections.unmodifiableSet(Stream.of(',', '.', '?', ':', ';','!', ' ').collect(toSet()));
```

Block Comments often Indicate a Method

```
// Compute word value
 while ((next = wordChars.read() ) != -1) {
    char nextChar = (char)next;
     int charValue = nextChar - 'a' + 1;
     if ((charValue >0) && (charValue < 27))
         currentWordValue += charValue;
 }
```

Here is the Method

```
public int wordValue(String word) throws IOException {
    int wordValue = 0;
    StringReader wordChars = new StringReader(word.toLowerCase());
    int next;
    while ((next = wordChars.read() ) != -1) {
        char nextChar = (char)next;
        int charValue = nextChar - 'a' + 1;
        if ((charValue >0) && (charValue < 27))
            wordValue += charValue;
    }
    return wordValue;
}
```

Here is the Resulting dollarWordsIn method

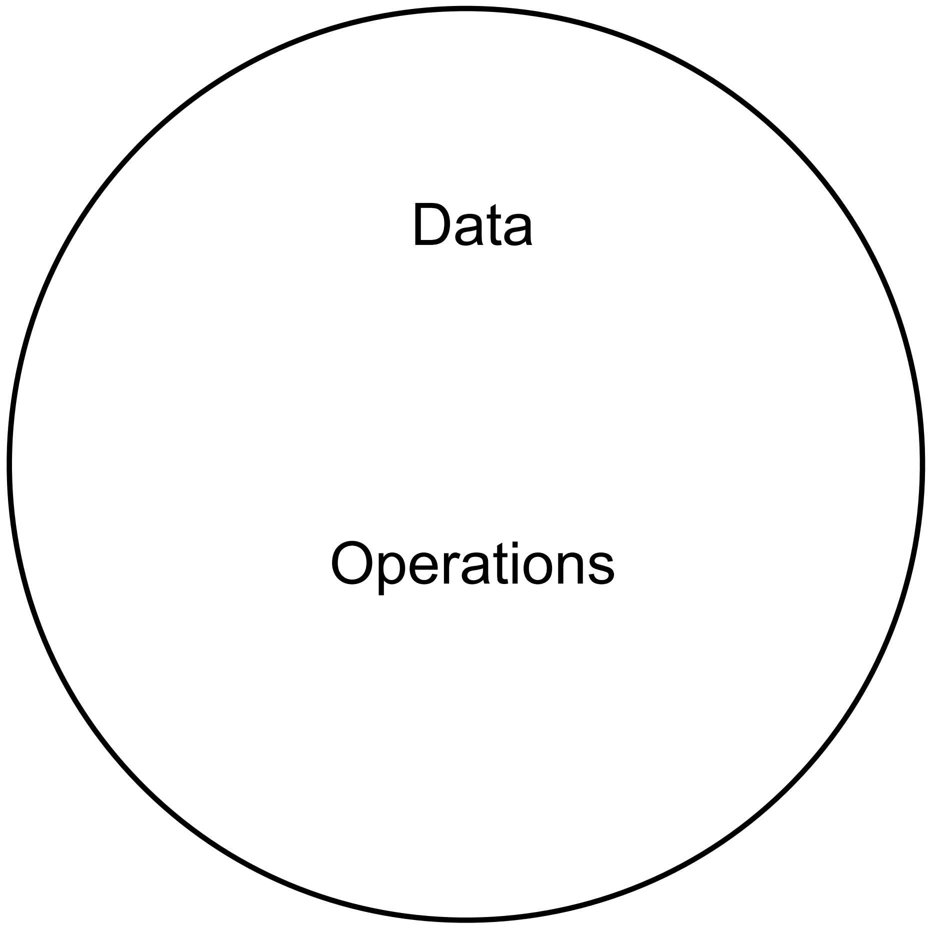
```
public ArrayList<String> dollarWordsIn(ArrayList<String> words) throws IOException {
   ArrayList<String> dollarWords = new ArrayList<String>();

   for (String word:words) {
      if (wordValue(word) == 100)
          dollarWords.add(word);
   }
   return dollarWords;
}
```

Issue: Code Reuse

Any Code reuse possible?

No Data

Just functions



A Different Approach

What building blocks can I make

What things need get done

- Who should do them

Then write program

Kotlin

Regular function

```
fun decrement(value: Int) : Int {
    return value - 1;
}
```

```
@Test
fun decrementTest() {
   assertTrue( 2 == decrement(3));
}
```

Extension Methods

Can add methods to existing classes

Class the method is in

↓

```
fun Int.inc(): Int {
   return this + 1;
}
```

```
@Test
fun incTest() {
   assertTrue( 2 == 1.inc());
}
```

Task: Convert Char to value

Char is the data, so add method to Char class

```
fun **Char**.wordValue(): Int {
    val wordValue = this.toLowerCase() - 'a' + 1
    return when {
        wordValue < 0 -> 0
        wordValue > 26 -> 0
        else -> wordValue
    }
}
```

```
@Test
fun testCharWordValue() {
    assertEquals(1, 'a'.wordValue())
    assertEquals(1, 'A'.wordValue())
    assertEquals(26, 'Z'.wordValue())
    assertEquals(0, '+'.wordValue())
}
```

Task: Compute value of a String

String (CharSequence) is the data, so add method to CharSequence interface

```
fun CharSequence.wordValue(): Int {
    return this.sumBy { it.wordValue() }
}
```

```
@Test
fun testCharSequenceWordValue() {
    assertEquals(2, "aa".wordValue())
    assertEquals(6, "abc".wordValue())
    assertEquals(0, "".wordValue())
    assertEquals(100,"buzzy".wordValue())
}
```

Task: Determine if String is Dollar Word

String (CharSequence) is the data, so add method to CharSequence interface

```
fun CharSequence.isDollarWord(): Boolean {
    return this.wordValue() == 100
}

@Test
fun testIsDollarWord() {
    val dollarWords = listOf("crusts", "status", "truism","TRuISm", "comport", "grouper")
    dollarWords.forEach {
        assertTrue { it.isDollarWord() }
    }

    val nonDollarWords = listOf("cat", "", "Mouse")
    nonDollarWords.forEach {
        assertFalse { it.isDollarWord() }
    }
}
```

Task: Determine if Char separates Words

Char is the data, so add method to Char class

```
fun **Char**.isWordSeparator(): Boolean {
    val separators = setOf(',', '.', '?', ':', ';','!', ' ')
    return separators.contains(this)
}
```

```
@Test
fun testIsSeparator() {
    assertTrue(','.isWordSeparator())
    assertFalse('q'.isWordSeparator())
    assertTrue(' '.isWordSeparator())
}
```

Task: Break String into Parts

String is the data, so add method to CharSequence interface

```
fun CharSequence.separatedBy(separator: (Char) -> Boolean): List<CharSequence> {
    val words = mutableListOf<CharSequence>()
    val currentWord = StringBuilder()

    this.forEach {
        if (separator(it)) {
            words.add(currentWord.toString())
            currentWord.clear()
        } else {
            currentWord.append(it)
        }
    }
    if (currentWord.isNotEmpty())
        words.add(currentWord.toString())
    return words
}
```

Task: Find all Dollar Words in a String

String is the data, so add method to CharSequence interface

```
fun CharSequence.dollarWords(): List<CharSequence> {
    val words = this.separatedBy { it.isWordSeparator()  }
    return words.filter { it.isDollarWord() }
}
```

```
@Test
fun testDollarWords() {
    val text = "This crusts is in a status of  truism!"
    val dollarWords = listOf("crusts", "status", "truism")
    assertEquals(dollarWords, text.dollarWords())
}
```

Code Reuse Possible?

Most method are specific to Dollar words

Any can easily be used elsewhere

isWordSeparator

- Might be useful elsewhere

separatedBy

- More likely to be used elsewhere

Part of OO Design Process

Who is on the team?

- What are the goals of the system?
- What must the system accomplish?
- What objects are required to model the system and accomplish the goals?

What are their tasks, responsibilities?

- What does each object have to know in order to accomplish each goal it is involved with?
- What steps toward accomplishing each goal is it responsible for?

Unit Testing

Testing

Johnson's Law

If it is not tested it does not work

The more time between coding and testing

- More effort is needed to write tests
- More effort is needed to find bugs
- Fewer bugs are found
- Time is wasted working with buggy code
- Development time increases
- Quality decreases

Unit Testing

Tests individual code segments

Automated tests

What wrong with:

Using print statements

Writing driver program in main

Writing small sample programs to run code

Running program and testing it be using it

We have a QA Team, so why should I write tests?

When to Write Tests

First write the tests

Then write the code to be tested

Writing tests first saves time

- Makes you clear of the interface & functionality of the code
- Removes temptation to skip tests

What to Test

Everything that could possibly break

Test values

- Inside valid range
- Outside valid range
- On the boundary between valid/invalid

GUIs are very hard to test

- Keep GUI layer very thin
- Unit test program behind the GUI, not the GUI

Common Things Programs Handle Incorrectly

Adapted with permission from “A Short Catalog of Test Ideas” by Brian Marick,
http://www.testing.com/writings.html

Strings
Empty String

Collections
Empty Collection
Collection with one element
Collection with duplicate elements
Collections with maximum possible size

Numbers
Zero
The smallest number
Just below the smallest number
The largest number
Just above the largest number

XUnit

Free frameworks for Unit testing

SUnit originally written by Kent Beck 1994

JUnit written by Kent Beck & Erich Gamma

Available at: http://www.junit.org/

Ports to many languages at:
 http://www.xprogramming.com/software.htm

JUnit Versions

3.x
Old version
Works with a versions of Java

4.x
Uses Annotations
Requires Java 5 or later

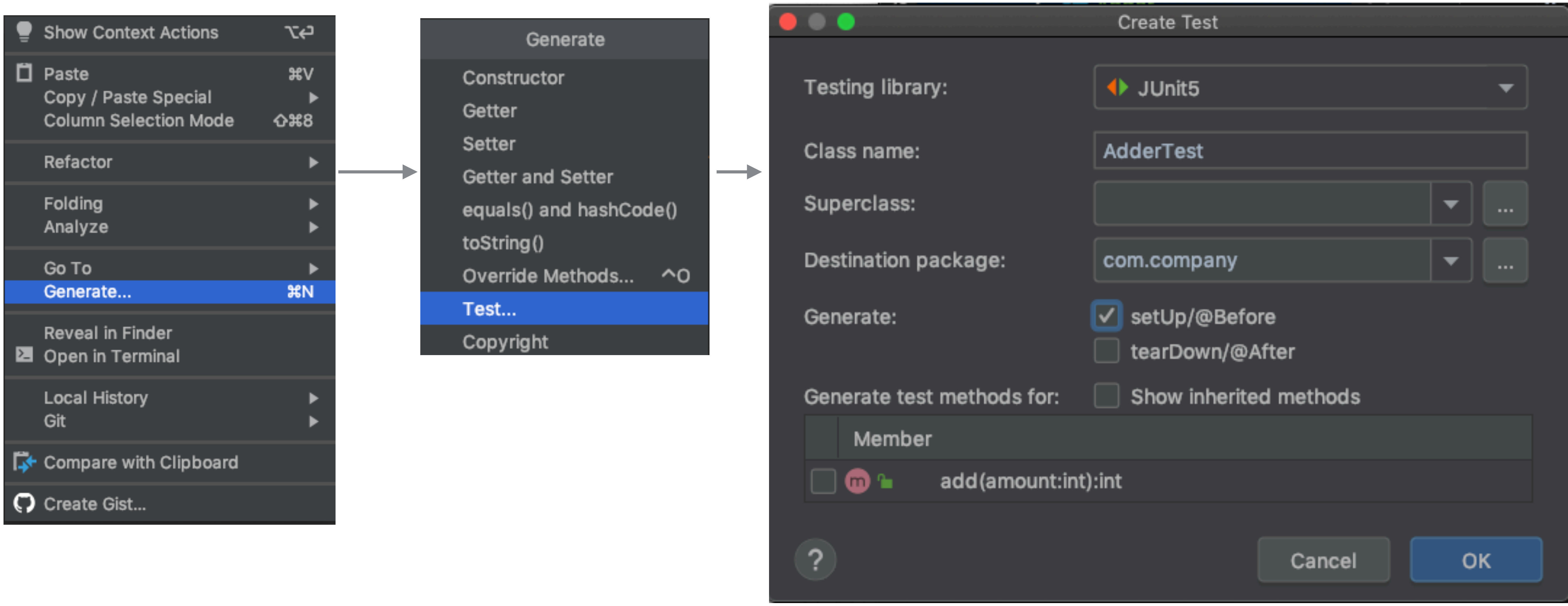
5.x
Supports Java 8 and later

Simple Class to Test

```
public class Adder {
    private int base;
    public Adder(int value) {
        base = value;
    }

    public int add(int amount) {
        return base + amount;
    }
}
```

Creating Test Case in Intellij

Put cursor in class you want test, right click



Testing library: JUnit5

Class name: AdderTest

Superclass:

Destination package: com.company

Generate: [✓ setUp/@Before] tearDown/@After

Generate test methods for: Show inherited methods

	Member
	add(amount:int):int

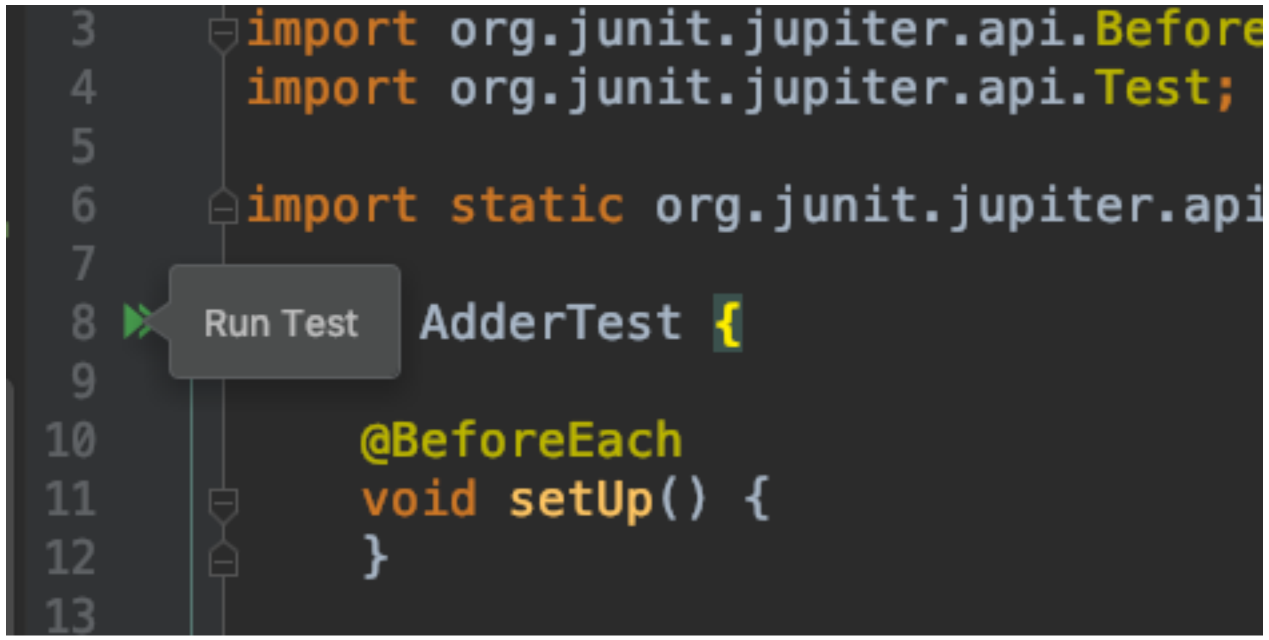
Test Class Created

```
import org.junit.jupiter.api.BeforeEach;

import static org.junit.jupiter.api.Assertions.*;

class AdderTest {

    @BeforeEach
    void setUp() {
    }
}
```

Test Class

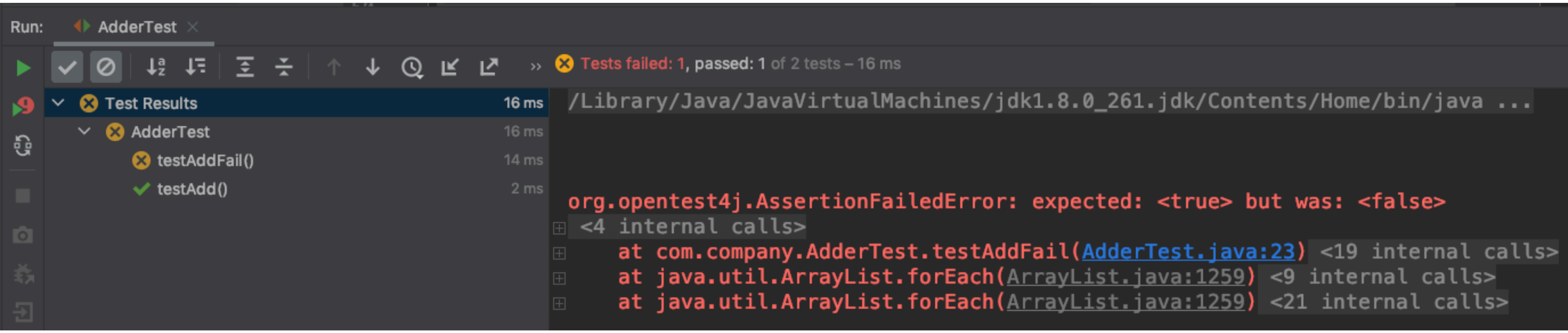
```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class AdderTest {

    @BeforeEach
    void setUp() {
    }

    @Test
    public void testAdd() {
        Adder example = new Adder(3);
        assertEquals(4, example.add(1));
    }

    @Test
    public void testAddFail() {
        Adder example = new Adder(3);
        assertTrue(3 == example.add(1));
    }
}
```

Running the Tests



```
import org.junit.jupiter.api.Before
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api

             AdderTest {

    @BeforeEach
    void setUp() {
    }
```

The result


Run: AdderTest

Tests failed: 1, passed: 1 of 2 tests – 16 ms

```
Test Results        16 ms
  AdderTest         16 ms
    testAddFail()   14 ms
    testAdd()        2 ms
```

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_261.jdk/Contents/Home/bin/java ...


org.opentest4j.AssertionFailedError: expected: <true> but was: <false>
 <4 internal calls>
    at com.company.AdderTest.testAddFail(AdderTest.java:23) <19 internal calls>
    at java.util.ArrayList.forEach(ArrayList.java:1259) <9 internal calls>
    at java.util.ArrayList.forEach(ArrayList.java:1259) <21 internal calls>
```

Assert Methods

assertArrayEquals()
assertTrue()
assertFalse()
assertEquals()
assertNotEquals()
assertSame()
assertNotSame()
assertNull()
assertNotNull()
fail()

Annotations - JUnit 5

AfterAll
AfterEach
BeforeAll
BeforeEach
Disabled
DisplayName
DisplayNameGeneration
Nested
Order
RepeatedTest
Tag
Tags
Test
TestFactory
TestInstance
TestMethodOrder
TestTemplate
Timeout

Using Before

```
class AdderTest {
    Adder example;

    @BeforeEach
    void setUp() {
        example = new Adder(3);
    }

    @Test
    public void testAdd() {
        assertEquals(4, example.add(1));
    }

    @Test
    public void testAddFail() {
        assertFalse(3 == example.add(1));
    }
}
```

Fowlers' Comments

Make sure all tests are fully automatic and that they check their own results.

A suite of tests is a powerful bug detector that decapitates the time it takes to find bugs.

Run tests frequently. Run those exercising the code you’re working on at least every few minutes; run all tests at least daily.

Fowlers' Comments

It is better to write and run incomplete tests than not to run complete tests.

The style I follow is to look at all the things the class should do and test each one of them for any conditions that might cause the class to fail. This is not the same as testing every public method, which is what some programmers advocate. Testing should be risk-driven; remember, I’m trying to find bugs, now or in the future. Therefore I don’t test accessors that just read and write a field: They are so simple that I’m not likely to find a bug there.

Fowlers' Comments

Think of the boundary conditions under which things might go wrong and concentrate your tests there.

When you get a bug report, start by writing a unit test that exposes the bug.