JUnit Framework

Four Phase Test and Test Planning

Produced by:

Mairead Meagher
Dr. Siobhán Drohan
Eamonn de Leastar

Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
http://www.wit.ie/

Topic List

– Four Phase Test.

– Planning a more complicated Test Case.

– Excuses for not Testing.

Four Phase Test

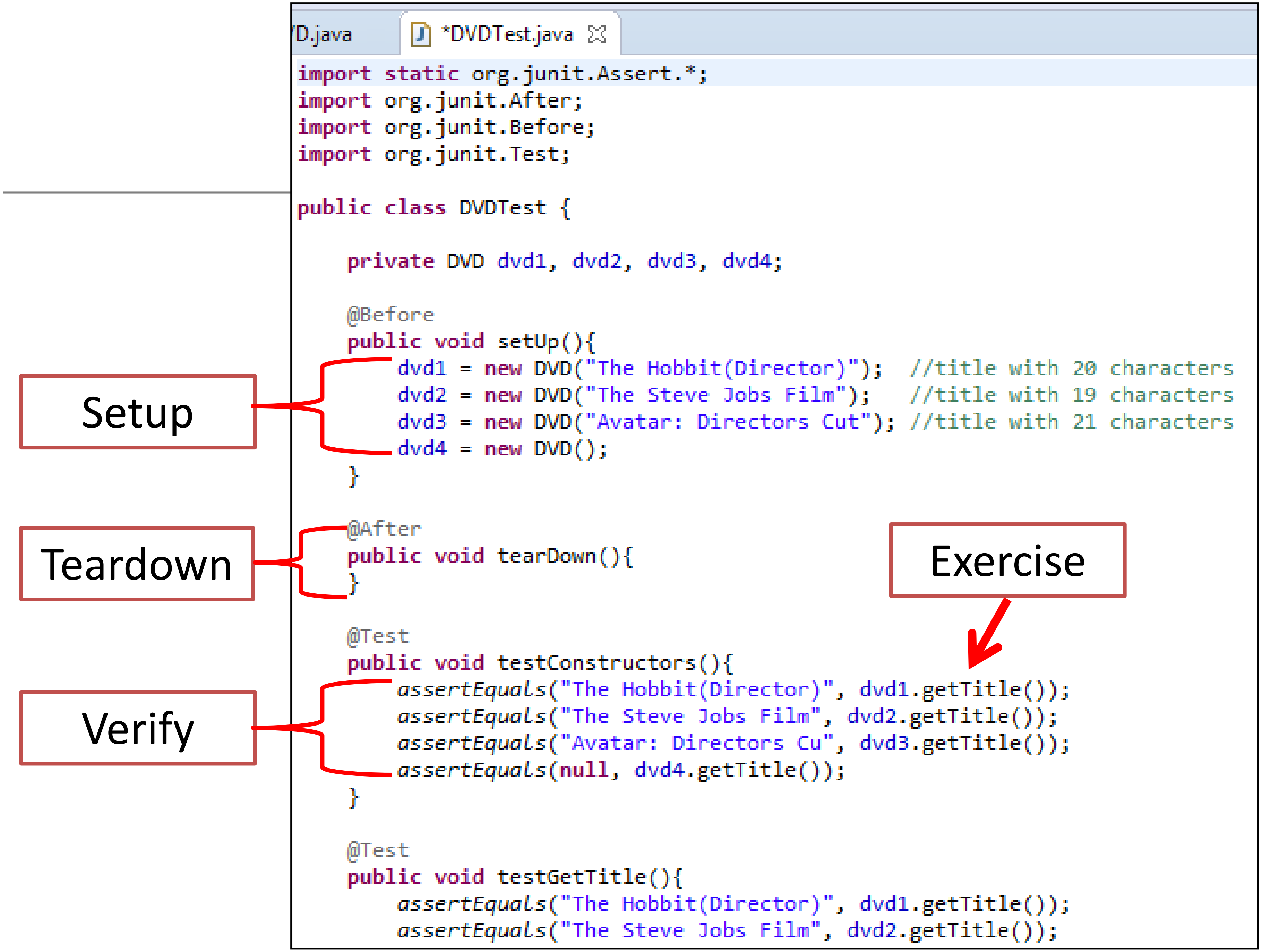
- How do we structure our test logic to make what we are testing obvious?

- We structure each test with four distinct phases executed in sequence.

Setup
Exercise
Verify
Teardown

How it works

Setup	We set up the test fixture (the “before” picture) so that we are in a position to exercise the tests. This could be objects that we need to create, values we need to set, other methods we need to call, etc.
Exercise	We interact with the system we are testing.
Verify	We do whatever is necessary to determine whether the expected outcome has been obtained.
Teardown	We tear down the test fixture to put the world back into the state in which we found it.



```java
import static org.junit.Assert.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class DVDTest {

    private DVD dvd1, dvd2, dvd3, dvd4;

    @Before
    public void setUp(){
        dvd1 = new DVD("The Hobbit(Director)");  //title with 20 characters
        dvd2 = new DVD("The Steve Jobs Film");   //title with 19 characters
        dvd3 = new DVD("Avatar: Directors Cut"); //title with 21 characters
        dvd4 = new DVD();
    }

    @After
    public void tearDown(){
    }

    @Test
    public void testConstructors(){
        assertEquals("The Hobbit(Director)", dvd1.getTitle());
        assertEquals("The Steve Jobs Film", dvd2.getTitle());
        assertEquals("Avatar: Directors Cu", dvd3.getTitle());
        assertEquals(null, dvd4.getTitle());
    }

    @Test
    public void testGetTitle(){
        assertEquals("The Hobbit(Director)", dvd1.getTitle());
        assertEquals("The Steve Jobs Film", dvd2.getTitle());
```

Topic List

– Four Phase Test.

– Planning a more complicated Test Case.

– Excuses for not Testing.

Planning JUnit Tests

- Method to test: A static method designed to find the largest number in a list of numbers.

- The following tests would seem to make sense:

  - [7, 8, 9] → 9
  - [8, 9, 7] → 9
  - [9, 7, 8] → 9

```
public static int largest (int[] list)
{
...
}
```

  - [supplied test data] → expected result

More Test Data + First Implementation

- Already have this data:
    - [7, 8, 9] -> 9
    - [8, 9, 7] -> 9
    - [9, 7, 8] -> 9
- What about this set of values:
    - [7, 9, 8, 9] -> 9
    - [1] -> 1
    - [-9, -8, -7] -> -7

```
public static int largest (int[] list)
{
   int index;
   int max = Integer.MAX_VALUE;

   for (index = 0; index < list.length - 1; index++)
   {
      if (list[index] > max)
      {
         max = list[index];
      }
    }

   return max;
}
```

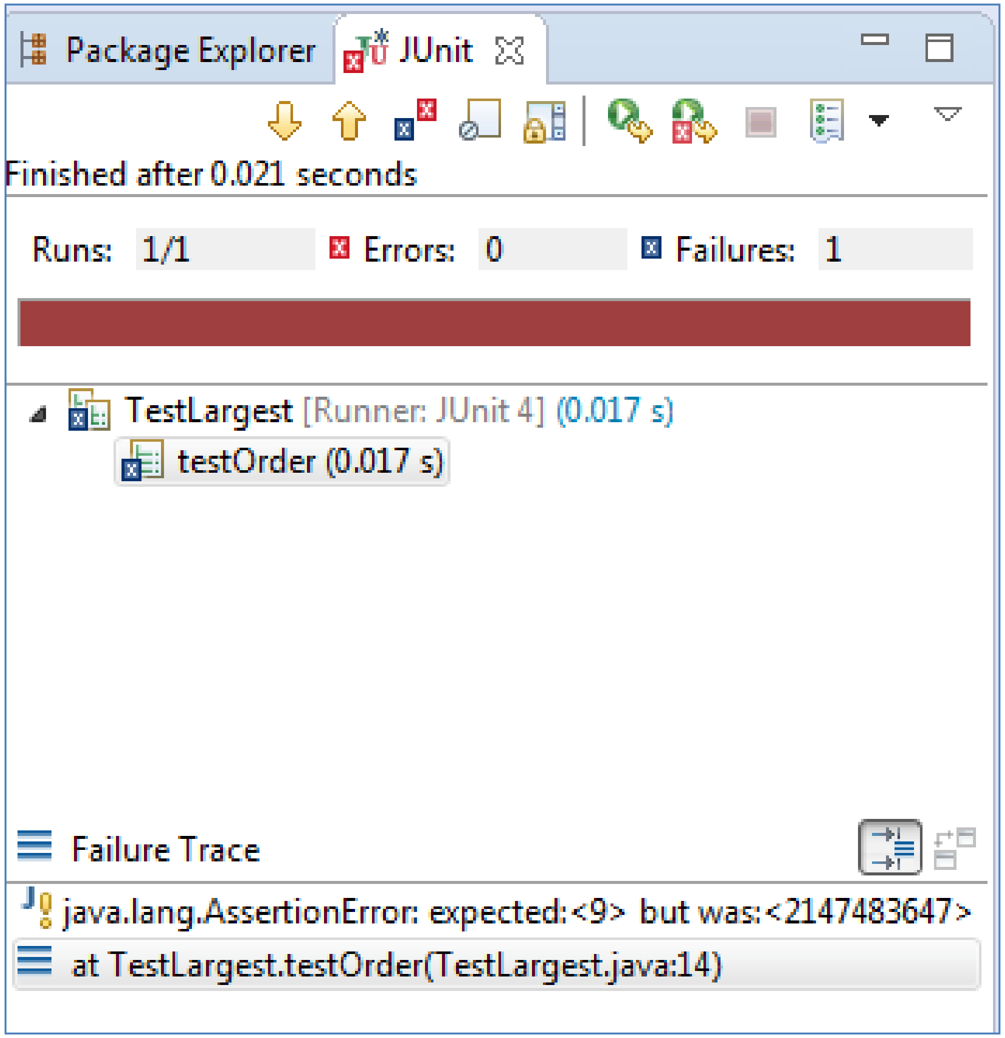
Writing the Test

- This is a TestCase called TestLargest.
- It has one Unit Test - to verify the behaviour of the largest method.

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

public class TestLargest
{

  @Test
  public void testOrder ()
  {
    int[] arr = new int[3];
    arr[0] = 8;
    arr[1] = 9;
    arr[2] = 7;
    assertEquals(9, Largest.largest(arr));
  }
}
```

Running the Test

- Why did it return such a huge number instead of our 9?
- Where could that very large number have come from?



Finished after 0.021 seconds

Runs: 1/1 Errors: 0 Failures: 1

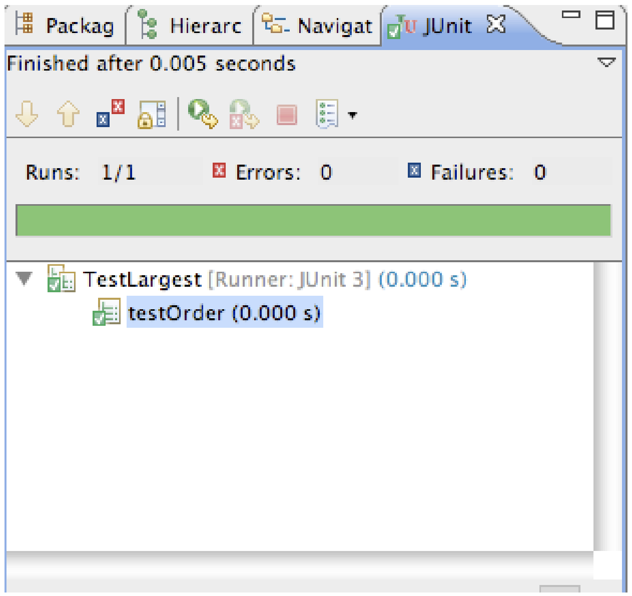
TestLargest [Runner: JUnit 4] (0.017 s)
testOrder (0.017 s)

Failure Trace

```
java.lang.AssertionError: expected:<9> but was:<2147483647>
at TestLargest.testOrder(TestLargest.java:14)
```

Bug

- First line should initialize max to zero, not MAX_VALUE.



Finished after 0.005 seconds

Runs: 1/1 Errors: 0 Failures: 0

TestLargest [Runner: JUnit 3] (0.000 s)
testOrder (0.000 s)

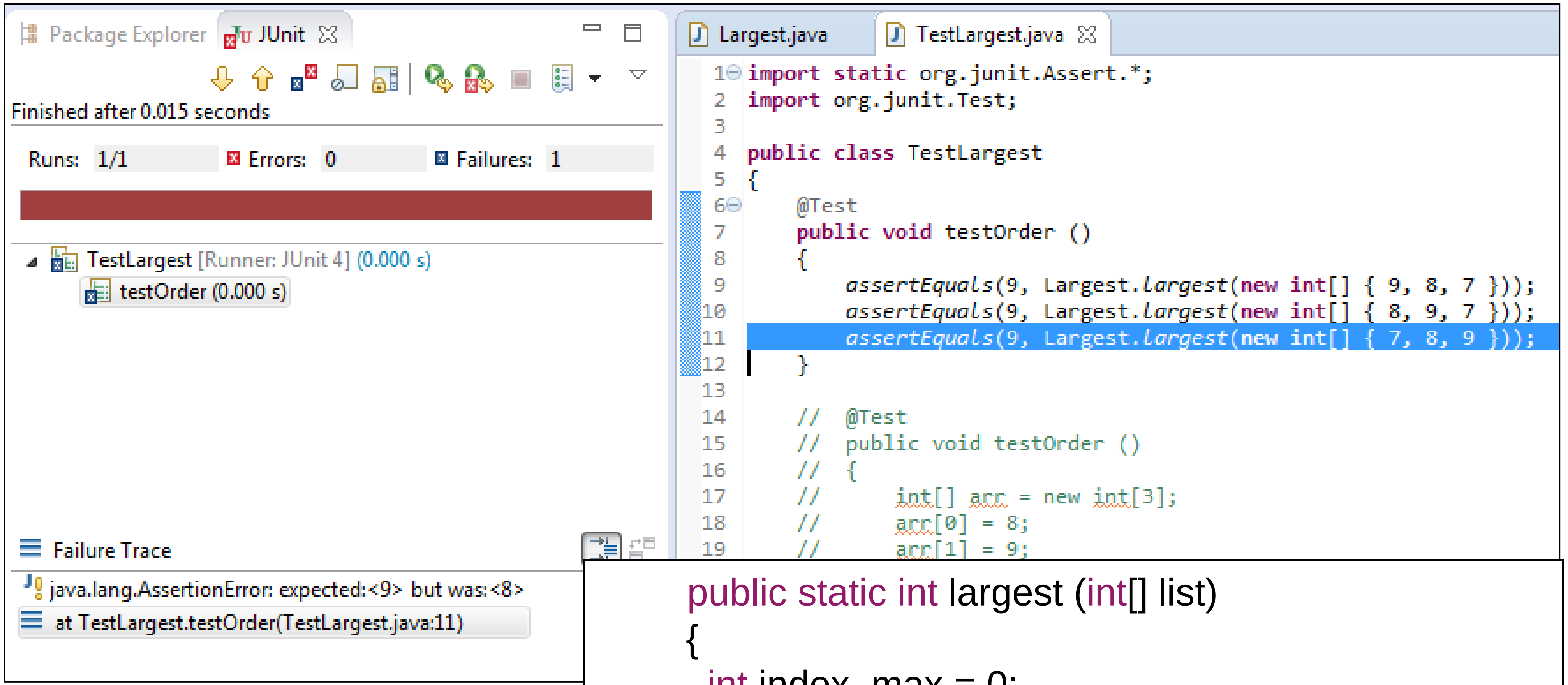
```
public static int largest (int[] list)
{
  //int index, max = Integer.MAX_VALUE;
  int index, max = 0;

  for (index = 0; index < list.length - 1; index++)
  {
    if (list[index] > max)
    {
      max = list[index];
    }
  }
  return max;
}
```

Further Tests

- What happens when the largest number appears in different places in the list - first or last, and somewhere in the middle?
  - Bugs most often show up at the “edges”.
  - In this case, edges occur when the largest number is at the start or end of the array that we pass in.

- Aggregate into a single unit test:

```
@Test
public void testOrder ()
{
  assertEquals(9, Largest.largest(new int[] { 9, 8, 7 }));
  assertEquals(9, Largest.largest(new int[] { 8, 9, 7 }));
  assertEquals(9, Largest.largest(new int[] { 7, 8, 9 }));
}
```



Package Explorer | JUnit

Finished after 0.015 seconds

Runs: 1/1 Errors: 0 Failures: 1

TestLargest [Runner: JUnit 4] (0.000 s)
testOrder (0.000 s)

Failure Trace

```
java.lang.AssertionError: expected:<9> but was:<8>
at TestLargest.testOrder(TestLargest.java:11)
```

Largest.java | TestLargest.java

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

public class TestLargest
{
    @Test
    public void testOrder ()
    {
        assertEquals(9, Largest.largest(new int[] { 9, 8, 7 }));
        assertEquals(9, Largest.largest(new int[] { 8, 9, 7 }));
        assertEquals(9, Largest.largest(new int[] { 7, 8, 9 }));
    }

    //  @Test
    //  public void testOrder ()
    //  {
    //      int[] arr = new int[3];
    //      arr[0] = 8;
    //      arr[1] = 9;
```

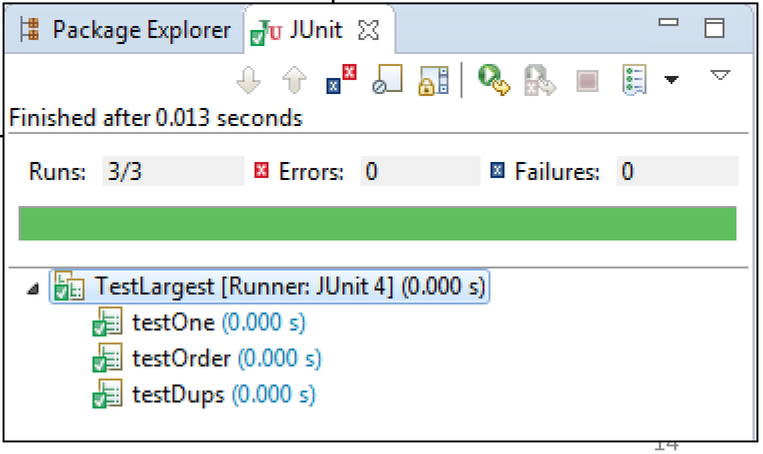
Failure + Fix

```
public static int largest (int[] list)
{
  int index, max = 0;
  //for (index = 0; index < list.length - 1; index++)
  for (index = 0; index < list.length; index++)
  {
    if (list[index] > max)
    {
      max = list[index];
    }
  }
  return max;
}
```

Further Boundary Conditions

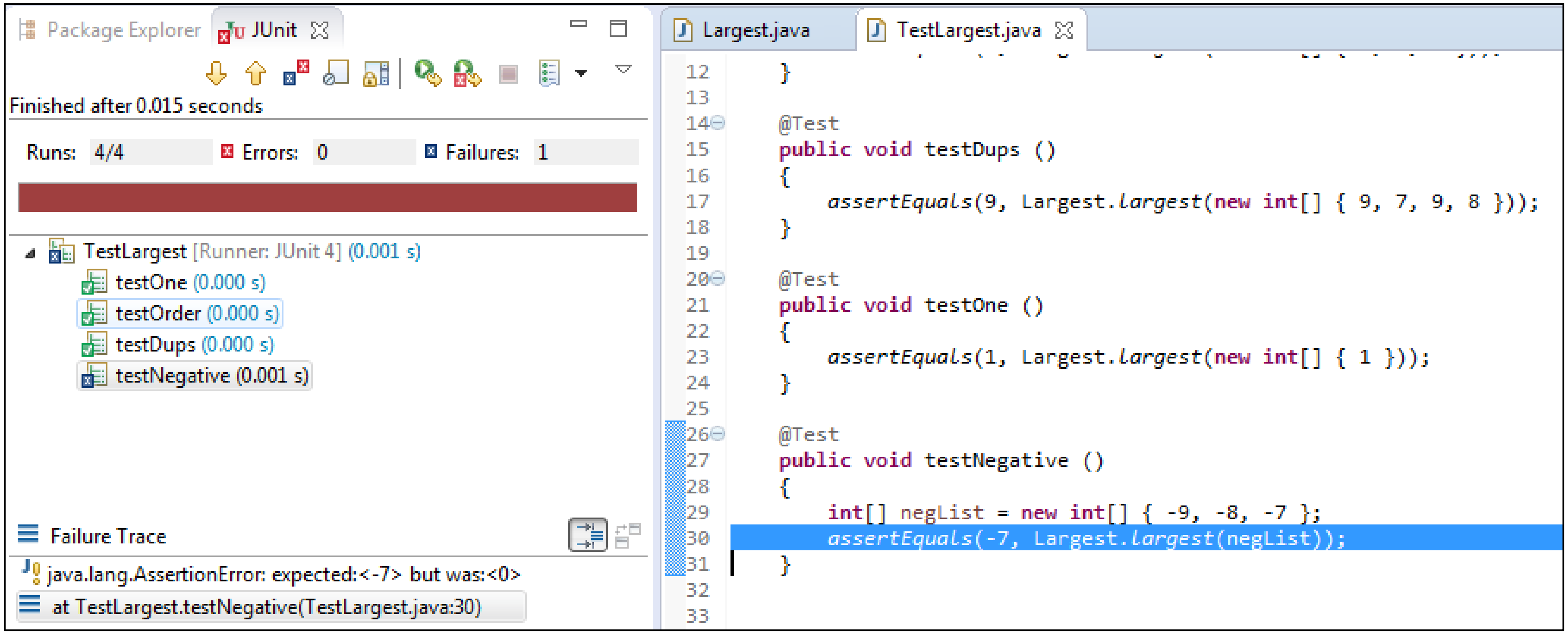
- Now exercising multiple tests

```
@Test
public void testDups ()
{
  assertEquals(9, Largest.largest(new int[] { 9, 7, 9, 8 }));
}

@Test
public void testOne ()
{
  assertEquals(1, Largest.largest(new int[] { 1 }));
}
```



```
Finished after 0.013 seconds

Runs: 3/3    Errors: 0    Failures: 0

TestLargest [Runner: JUnit 4] (0.000 s)
    testOne (0.000 s)
    testOrder (0.000 s)
    testDups (0.000 s)
```

Failure on testNegative



Finished after 0.015 seconds

Runs: 4/4   Errors: 0   Failures: 1

- TestLargest [Runner: JUnit 4] (0.001 s)
  - testOne (0.000 s)
  - testOrder (0.000 s)
  - testDups (0.000 s)
  - testNegative (0.001 s)

Failure Trace

```
java.lang.AssertionError: expected:<-7> but was:<0>
at TestLargest.testNegative(TestLargest.java:30)
```

```
    }

    @Test
    public void testDups ()
    {
        assertEquals(9, Largest.largest(new int[] { 9, 7, 9, 8 }));
    }

    @Test
    public void testOne ()
    {
        assertEquals(1, Largest.largest(new int[] { 1 }));
    }

    @Test
    public void testNegative ()
    {
        int[] negList = new int[] { -9, -8, -7 };
        assertEquals(-7, Largest.largest(negList));
    }
```

fix testNegative

- Choosing 0 to initialize max was a bad idea;
- Should have been MIN VALUE, so as to be less than all negative numbers as well.

```
public static int largest (int[] list)
{
    //int index, max = 0;
    int index = 0;
    int max = Integer.MIN_VALUE;

    for (index = 0; index < list.length; index++)
    {
        if (list[index] > max)
        {
            max = list[index];
        }
     }
     return max;
}
```

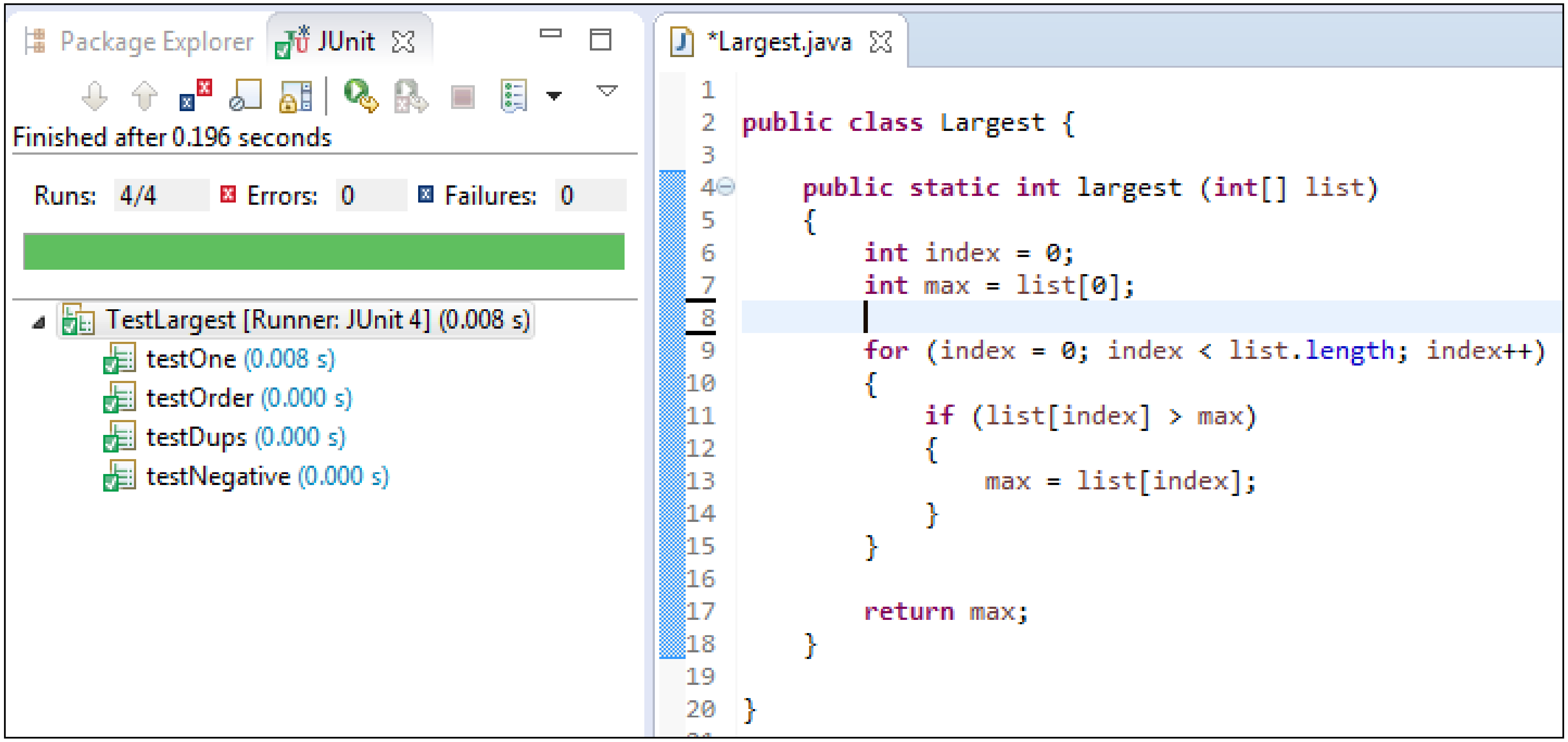
Is there a better approach for setting the max value?

- Maybe instead of the MIN VALUE, we set max to be the first element in the list array.

- Would that work?

```
public static int largest (int[] list)
{
    //int index, max = 0;
    int index = 0;
    int max = list[0];

    for (index = 0; index < list.length; index++)
    {
        if (list[index] > max)
        {
            max = list[index];
        }
     }
     return max;
}
```

Yes and this is the preferred approach!



Finished after 0.196 seconds

Runs: 4/4 Errors: 0 Failures: 0

TestLargest [Runner: JUnit 4] (0.008 s)
- testOne (0.008 s)
- testOrder (0.000 s)
- testDups (0.000 s)
- testNegative (0.000 s)

```java

public class Largest {

    public static int largest (int[] list)
    {
        int index = 0;
        int max = list[0];

        for (index = 0; index < list.length; index++)
        {
            if (list[index] > max)
            {
                max = list[index];
            }
        }

        return max;
    }

}
```

Topic List

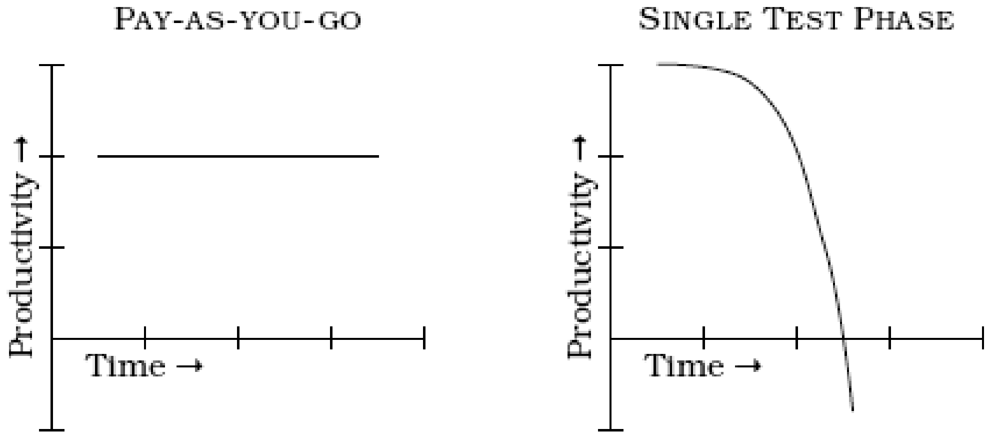
– Four Phase Test.

– Planning a more complicated Test Case.

– Excuses for not Testing.

Excuses for not Testing (1)

- *It takes too much time to write the tests:*
  - The trade-off is not “test now” versus “test later”
  - It's linear work now versus exponential work and complexity trying to fix and rework at the end.



Excuses for not Testing (2)

- *“It takes too long to run the tests”*
  - Separate out the longer-running tests from the short ones.
  - Only run the long tests once a day, or once every few days as appropriate, and run the shorter tests constantly.
- *“It's not developers job to test his/her code”*
  - Integral part of developer job is to create working code.
- *“But it compiles!”*
  - Compiler's blessing is a pretty shallow compliment.

Any Questions?

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
http://www.wit.ie/