JUnit Framework

Four Phase Test and Test Planning

Produced by:

Mairead Meagher
Dr. Siobhán Drohan
Eamonn de Leastar
Siobhan Roche

Ollscoil Teicneolaíochta an Oirdheiscirt
South East Technological University

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

Topic List

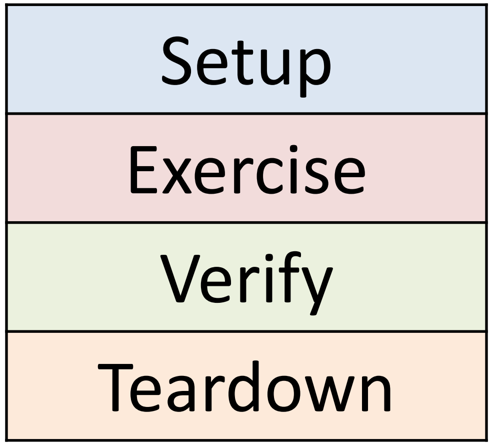
– Four Phase Test.

– Planning a more complicated Test Case.

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

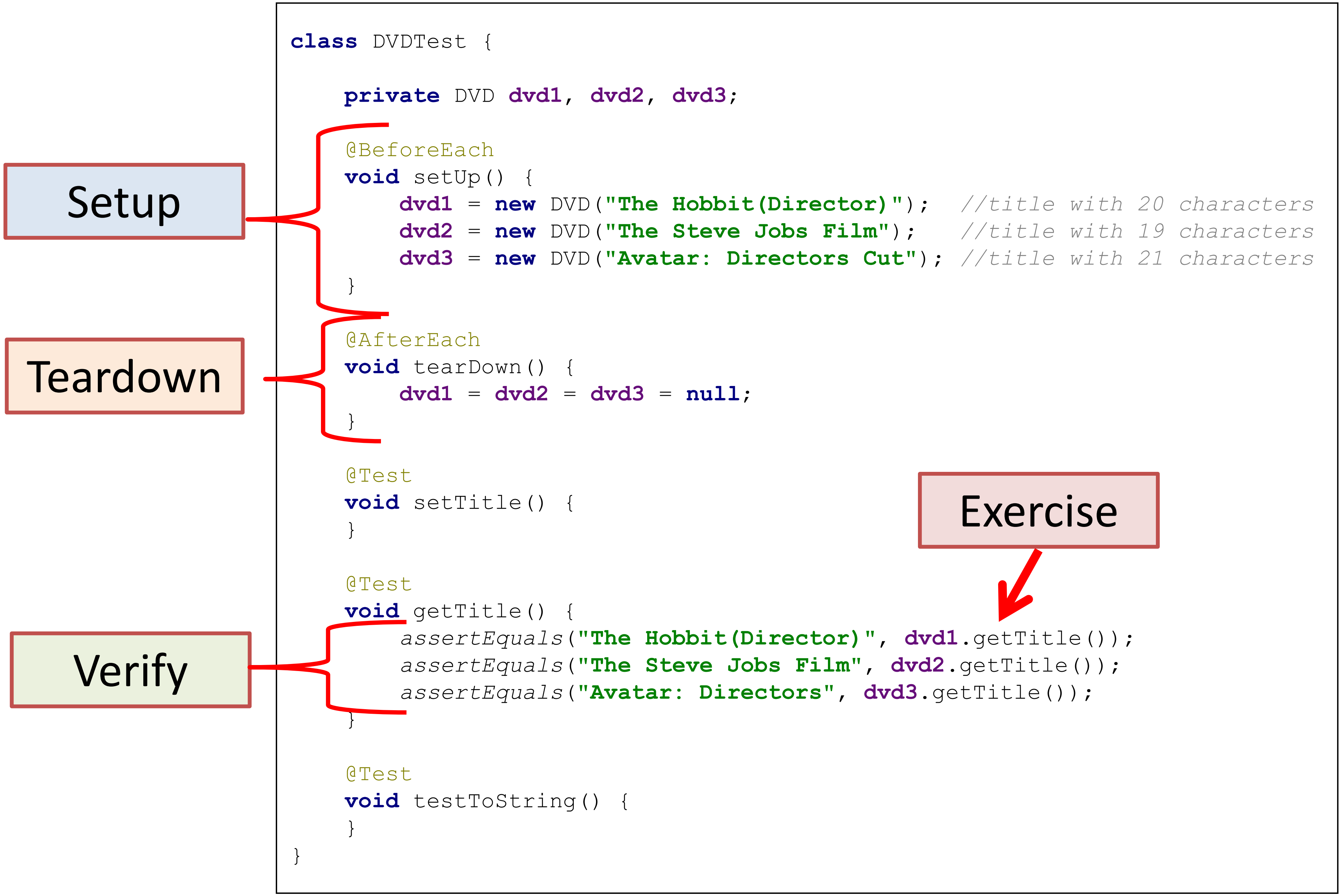
Exercise

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.



```
class DVDTest {

    private DVD dvd1, dvd2, dvd3;

    @BeforeEach
    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
    }

    @AfterEach
    void tearDown() {
        dvd1 = dvd2 = dvd3 = null;
    }

    @Test
    void setTitle() {
    }

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

    @Test
    void testToString() {
    }
}
```

Topic List

– Four Phase Test.

– Planning a more complicated Test Case.

Planning JUnit Tests

- **Method to test:** A static method designed to return the largest number in a primitive array of int.

- **Suggested tests:** 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

- Already planned tests with 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

[supplied test data] → expected result

More Test Data

- Already planned tests with 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;
}
```

[supplied test data] → expected result

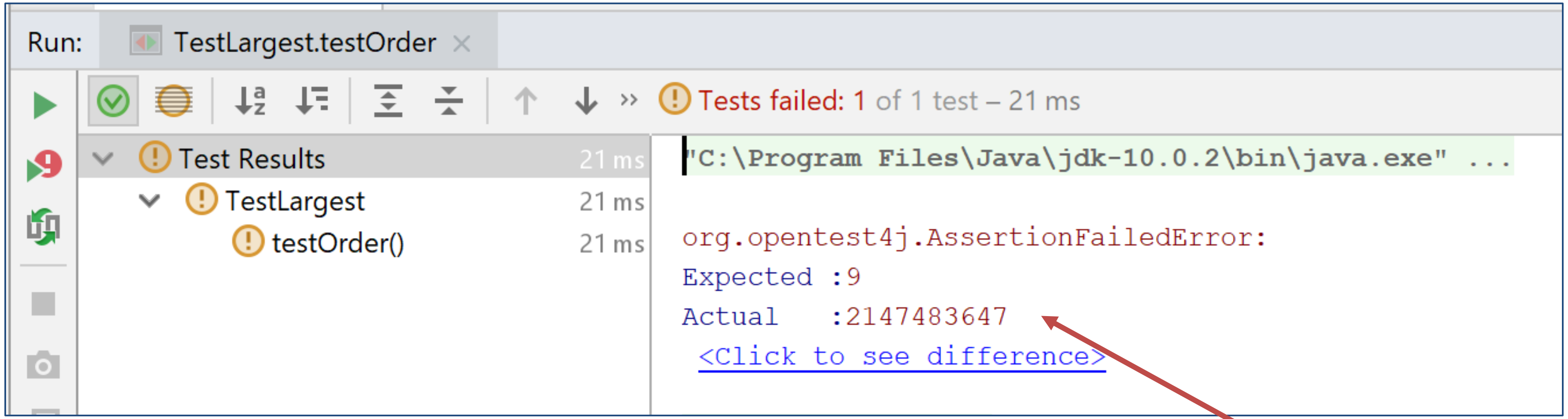
Writing the Test

This is a **TestCase** called **TestLargest**

It has one Unit Test - to verify the behaviour of the largest method.

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

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



Tests failed: 1 of 1 test – 21 ms

Test Results	21 ms
TestLargest	21 ms
testOrder()	21 ms

```
"C:\Program Files\Java\jdk-10.0.2\bin\java.exe" ...

org.opentest4j.AssertionFailedError:
Expected :9
Actual   :2147483647
 <Click to see difference>
```

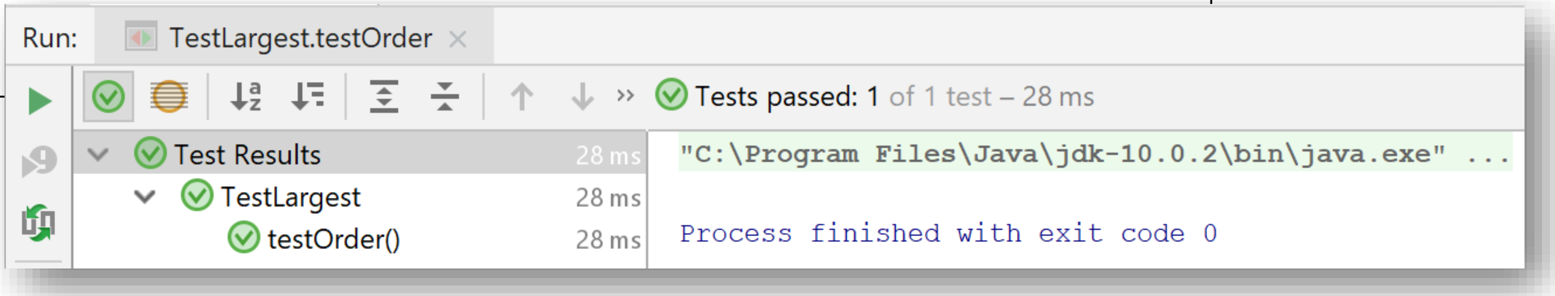
Why did it return such a huge number instead of our 9?

Where could that very large number have come from?

Bug

We should initialize **max** to zero, not MAX_VALUE.

```
public class Largest {

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

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

        return max;
    }
}
```



Tests passed: 1 of 1 test – 28 ms

```
Test Results        28 ms
  TestLargest       28 ms
    testOrder()     28 ms
```

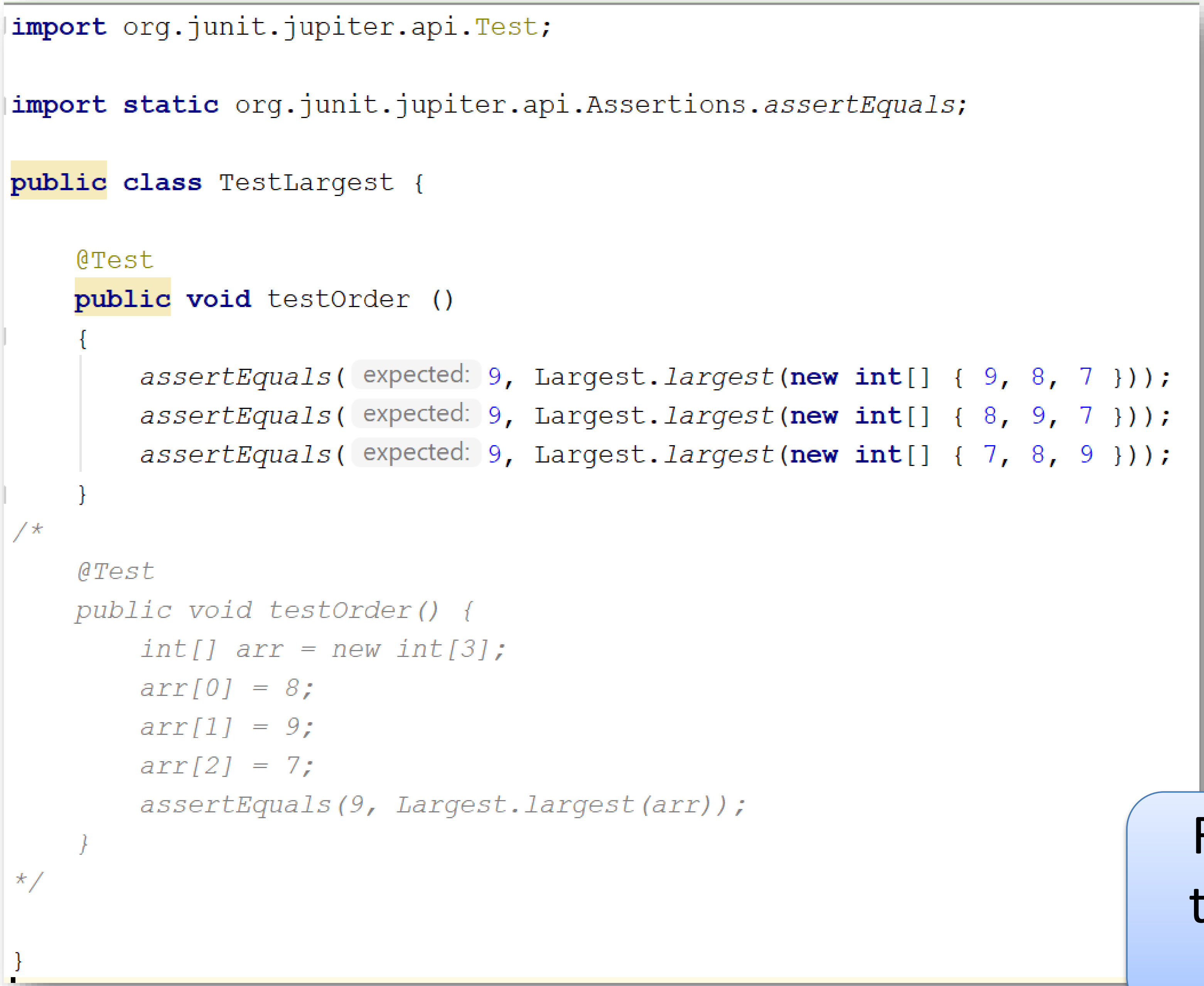
```
"C:\Program Files\Java\jdk-10.0.2\bin\java.exe" ...

Process finished with exit code 0
```

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.

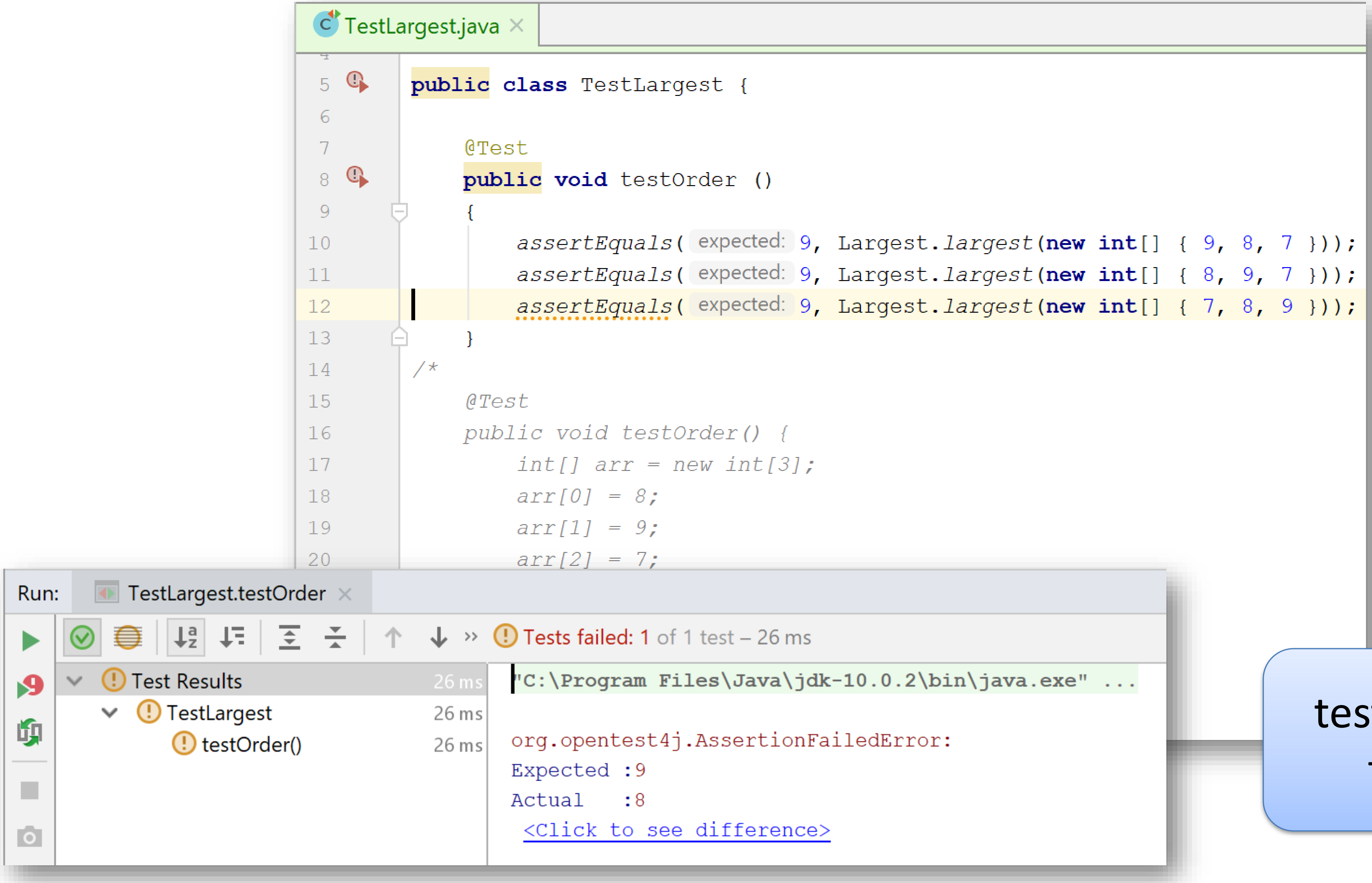
- Aggregate into a single unit test:

```
@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 }));
}
```



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

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

public class TestLargest {

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

}
```

Refactored testOrder() method



```
public class TestLargest {

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

Tests failed: 1 of 1 test – 26 ms

Test Results	26 ms
TestLargest	26 ms
testOrder()	26 ms

```
"C:\Program Files\Java\jdk-10.0.2\bin\java.exe" ...

org.opentest4j.AssertionFailedError:
Expected :9
Actual   :8
 <Click to see difference>
```

testOrder() failed

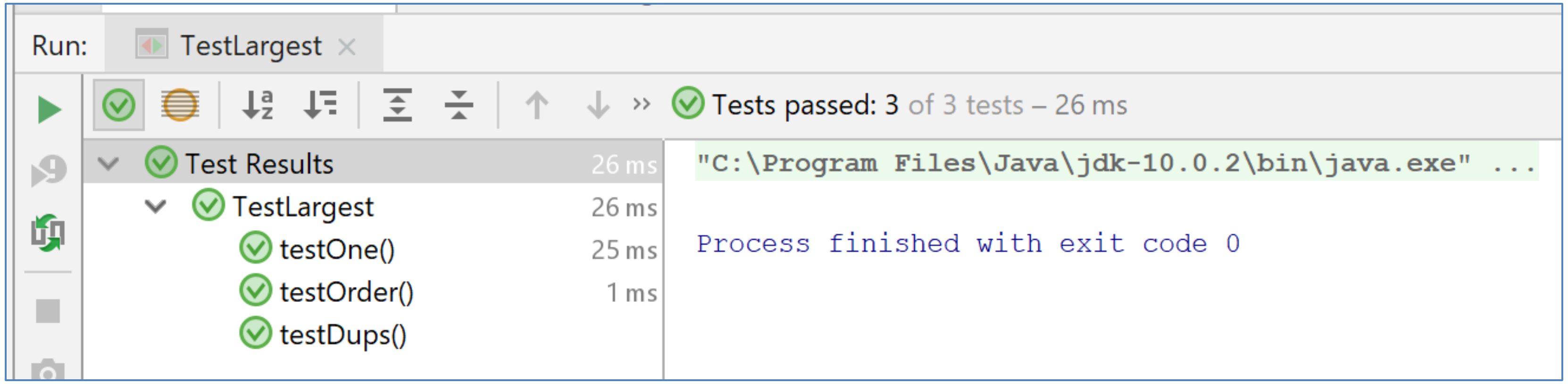
```java
public class Largest {

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

Code Fix

Further Boundary Conditions

Exercising More Tests

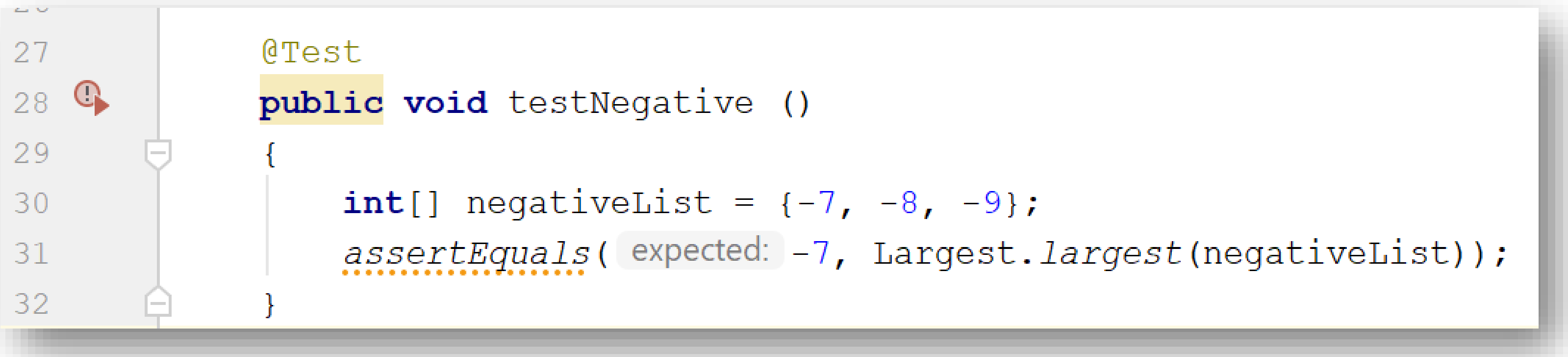
```
@Test
public void testDups ()
{
    assertEquals(9, Largest.largest(new int[] { 9, 7, 9, 8 }));
}

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



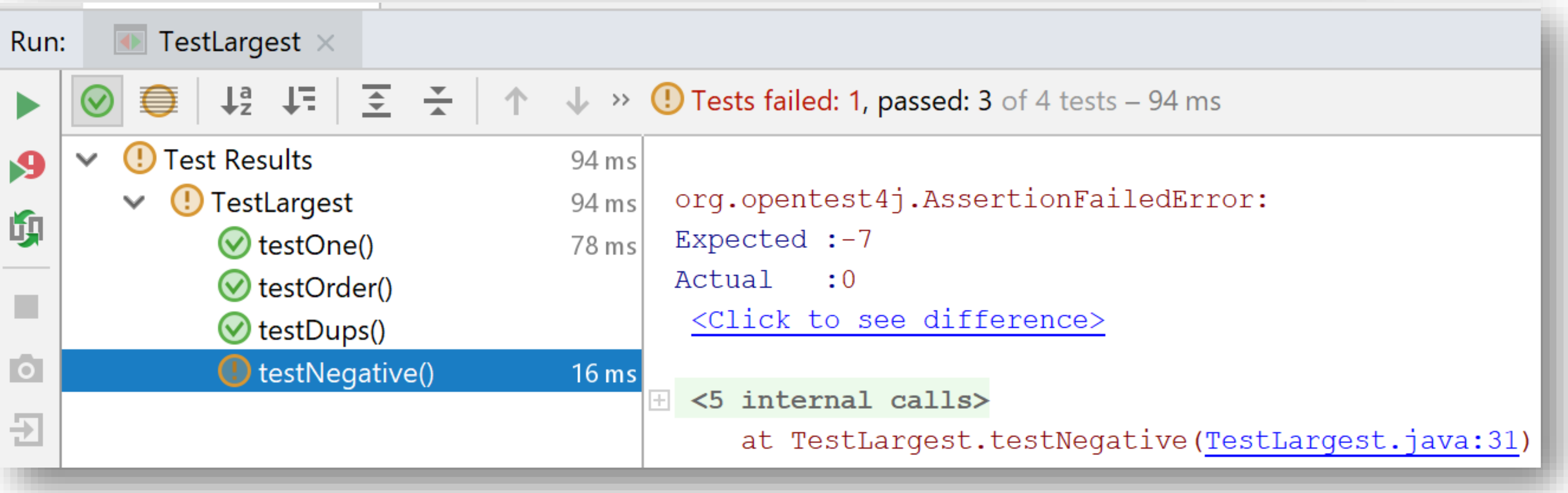
```
✓ Tests passed: 3 of 3 tests – 26 ms

✓ Test Results        26 ms
  ✓ TestLargest       26 ms
    ✓ testOne()       25 ms
    ✓ testOrder()      1 ms
    ✓ testDups()
```

```
"C:\Program Files\Java\jdk-10.0.2\bin\java.exe" ...

Process finished with exit code 0
```

Failure on testNegative



```
27        @Test
28        public void testNegative ()
29        {
30            int[] negativeList = {-7, -8, -9};
31            assertEquals( expected: -7, Largest.largest(negativeList));
32        }
```



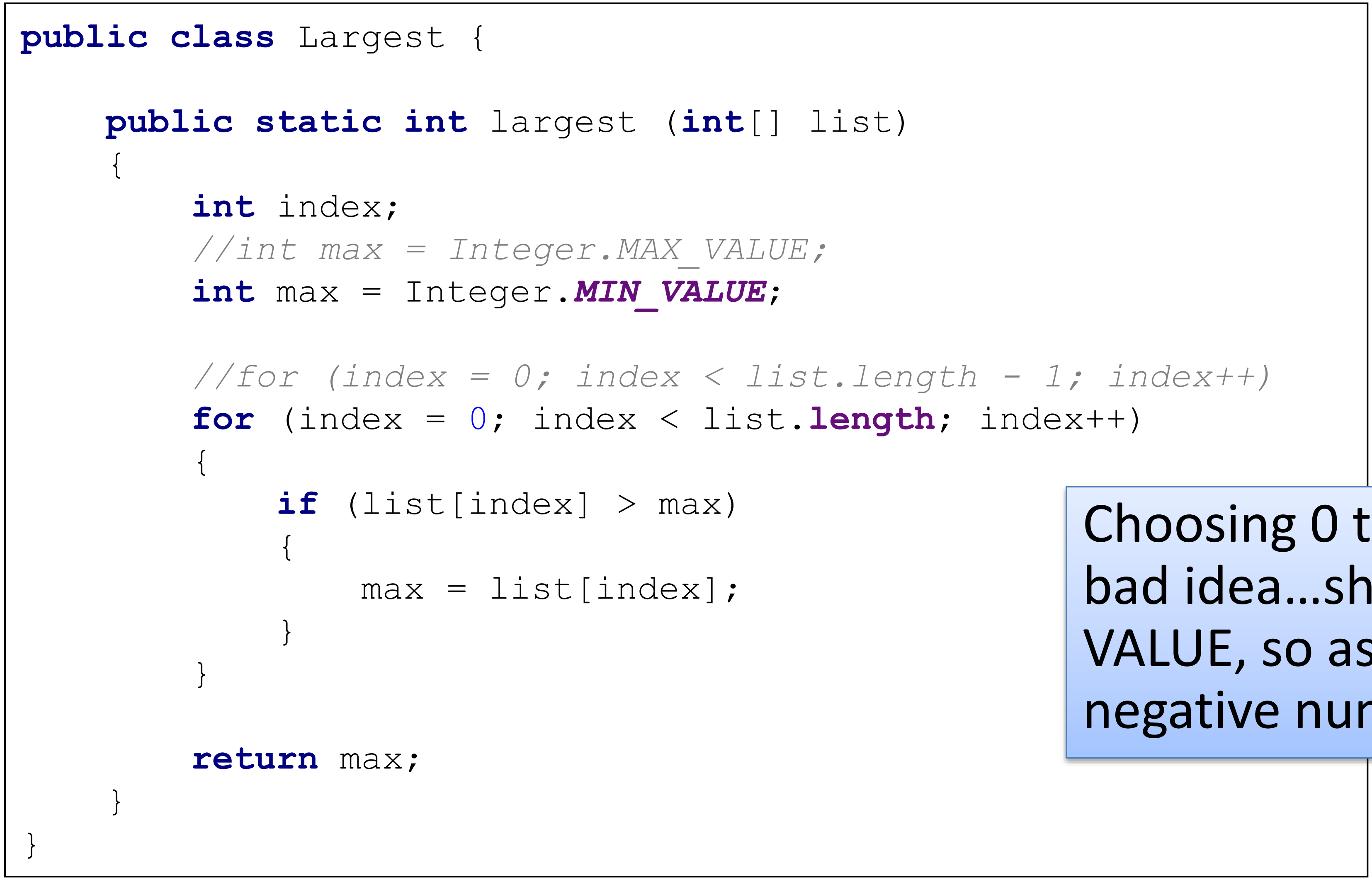
Run: TestLargest ×

Tests failed: 1, passed: 3 of 4 tests – 94 ms

```
Test Results          94 ms
  TestLargest         94 ms
    testOne()         78 ms
    testOrder()
    testDups()
    testNegative()    16 ms
```

```
org.opentest4j.AssertionFailedError:
Expected :-7
Actual   :0
 <Click to see difference>

<5 internal calls>
    at TestLargest.testNegative(TestLargest.java:31)
```

fix testNegative



```java
public class Largest {

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

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

        return max;
    }
}
```

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.

Is there a better approach for setting the max value?

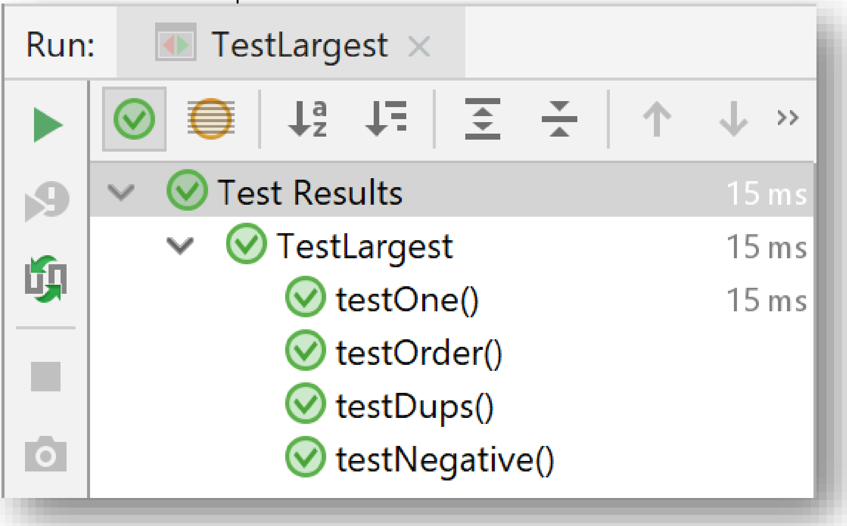
- Maybe instead of the MIN VALUE, we set max to be the first element in the list array.

- Would that work?

```
public static int largest (int[] list)
{
    int index = 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!

```java
public class Largest {

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

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

        return max;
    }
}
```



```
Run:   TestLargest

Test Results              15 ms
  TestLargest             15 ms
    testOne()             15 ms
    testOrder()
    testDups()
    testNegative()
```

Any Questions?