

Test Driven Development

Using JUnit to Test Product.java

Produced Dr. Siobhán Drohan
by: Mairead Meagher
 Siobhan Roche

Let's first write a complete test class and then we will discuss the theory behind it.

Testing Product.java

Product.java

```
package models;

import utils.Utilities;

public class Product {

    private String productName = "";
    private int productCode = -1;
    private double unitCost = 0;
    private boolean inCurrentProductLine = false;

    public Product(String productName, int productCode, double unitCost, boolean inCurrentProductLine) {
        this.productName = Utilities.truncateString(productName, 20);
        setProductCode(productCode);
        this.unitCost = unitCost;
        this.inCurrentProductLine = inCurrentProductLine;
    }

    public String toString()
    {
        return "Product description: " + productName
            + ", product code: " + productCode
            + ", unit cost: " + unitCost
            + ", currently in product line: " + Utilities.booleanToYN(inCurrentProductLine);
    }
}
```

Product.java

```
package models;
```

```
import utils.Utilities;
```

```
public class Product {
```

```
    private String productName = "";
    private int productCode = -1;
    private double unitCost = 0;
    private boolean inCurrentProductLine = false;
```

```
    public Product(String productName, int productCode, double unitCost, boolean inCurrentProductLine) {
        this.productName = Utilities.truncateString(productName, 20);
        setProductCode(productCode);
        this.unitCost = unitCost;
        this.inCurrentProductLine = inCurrentProductLine;
    }
```

```
    public String toString()
    {
        return "Product description: " + productName
            + ", product code: " + productCode
            + ", unit cost: " + unitCost
            + ", currently in product line: " + Utilities.booleanToYN(inCurrentProductLine);
    }
}
```

```
    public void setProductCode(int productCode) {
        if (Utilities.validRange(productCode, 1000, 9999)) {
            this.productCode = productCode;
        }
    }
```

```
    public void setProductName(String productName) {
        if (Utilities.validateStringLength(productName, 20)) {
            this.productName = productName;
        }
    }
```

```
    public void setUnitCost(double unitCost) {
        this.unitCost = unitCost;
    }
```

```
    public void setInCurrentProductLine(boolean inCurrentProductLine) {
        this.inCurrentProductLine = inCurrentProductLine;
    }
```

```
    public String getProductName(){
        return productName;
    }
```

```
    public double getUnitCost(){
        return unitCost;
    }
```

```
    public int getProductCode() {
        return productCode;
    }
```

```
    public boolean isInCurrentProductLine() {
        return inCurrentProductLine;
    }
}
```

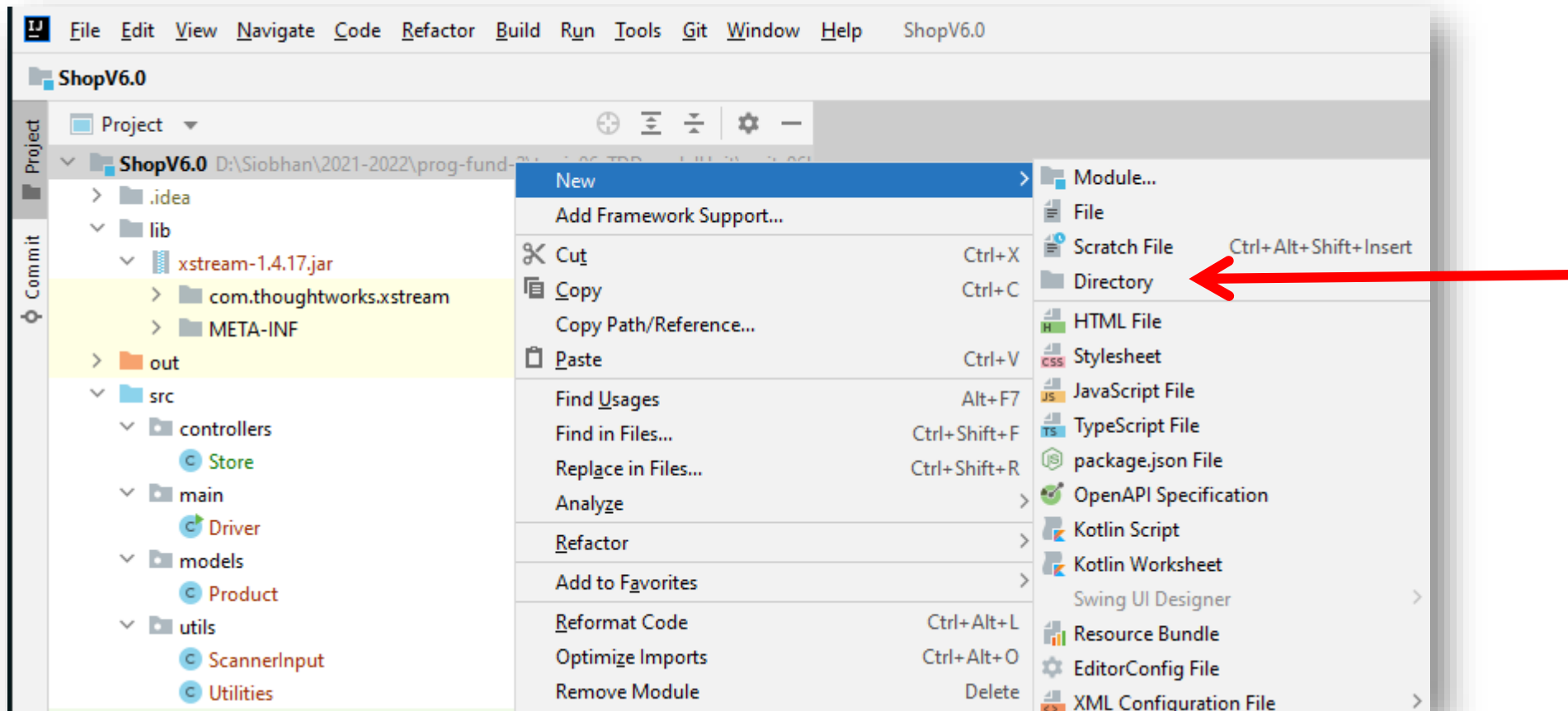
Utilities.java

Some of these utility methods are used in the Product class:

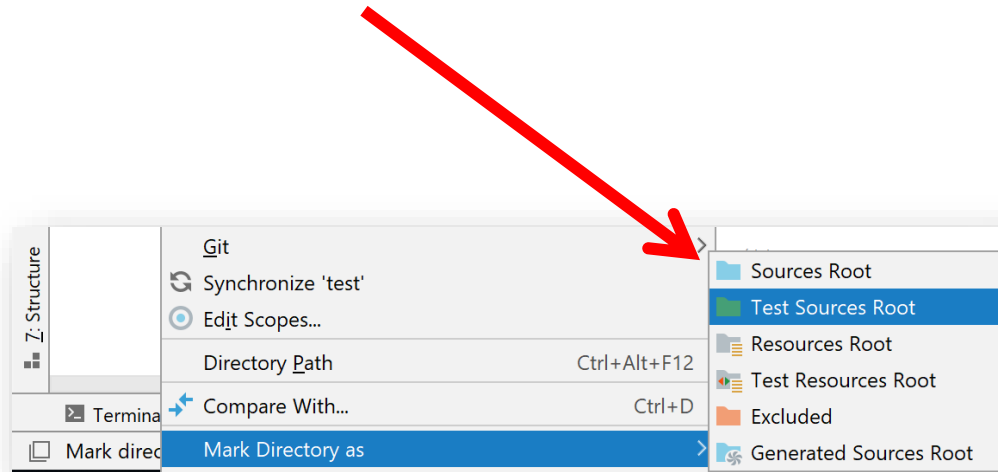
```
public class Utilities {  
  
    public static double toTwoDecimalPlaces(double number){  
        return (int) (number * 100 ) / 100.0;  
    }  
  
    public static boolean YNtoBoolean(char charToConvert){  
        return ((charToConvert == 'y') || (charToConvert == 'Y'));  
    }  
  
    public static char booleanToYN(boolean booleanToConvert){  
        return booleanToConvert ? 'Y' : 'N';  
    }  
  
    public static String truncateString(String stringToTruncate, int length){  
        if (stringToTruncate.length() <= length) {  
            return stringToTruncate;  
        }  
        else{  
            return stringToTruncate.substring(0, length);  
        }  
    }  
  
    public static boolean validateStringLength(String strToCheck, int maxLength){  
        return strToCheck.length() <= maxLength;  
    }  
  
    public static boolean validRange(int numberToCheck, int min, int max) {  
        return ((numberToCheck >= min) && (numberToCheck <= max));  
    }  
}
```

Creating a Test Source Directory...

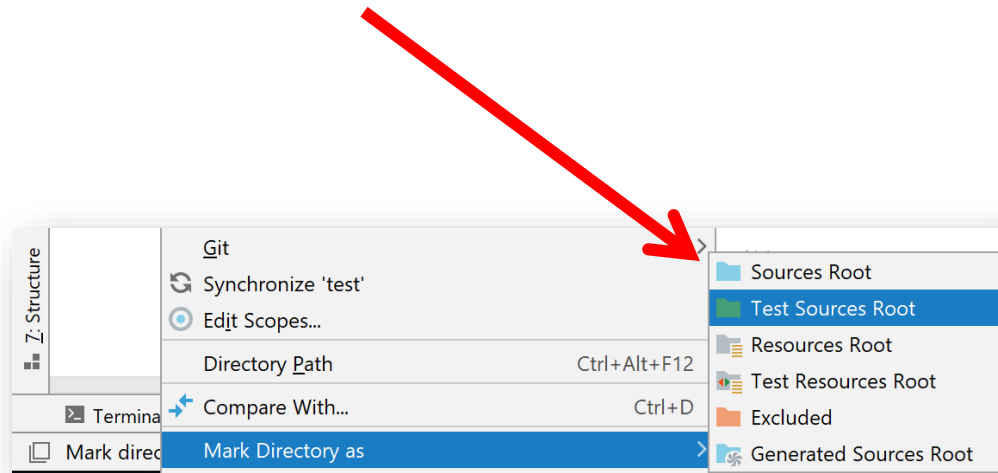
In IntelliJ, create a new directory and call it **test**



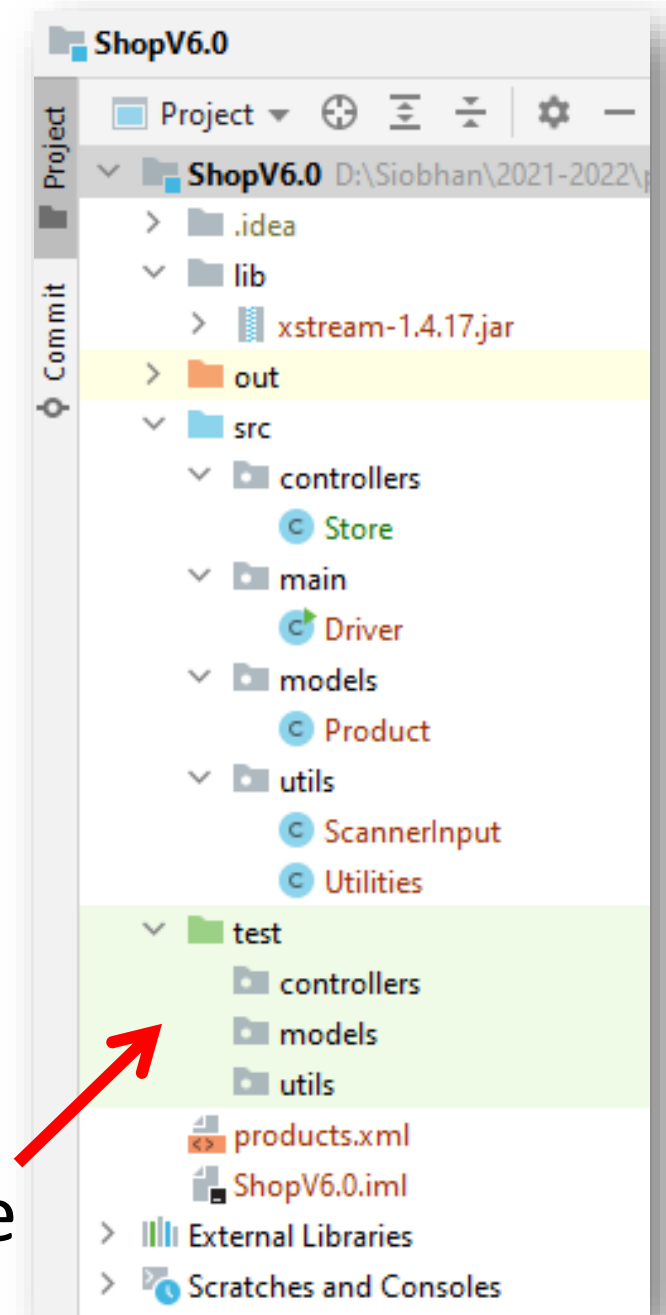
Then “mark” this new
directory as a
Test Sources Root directory



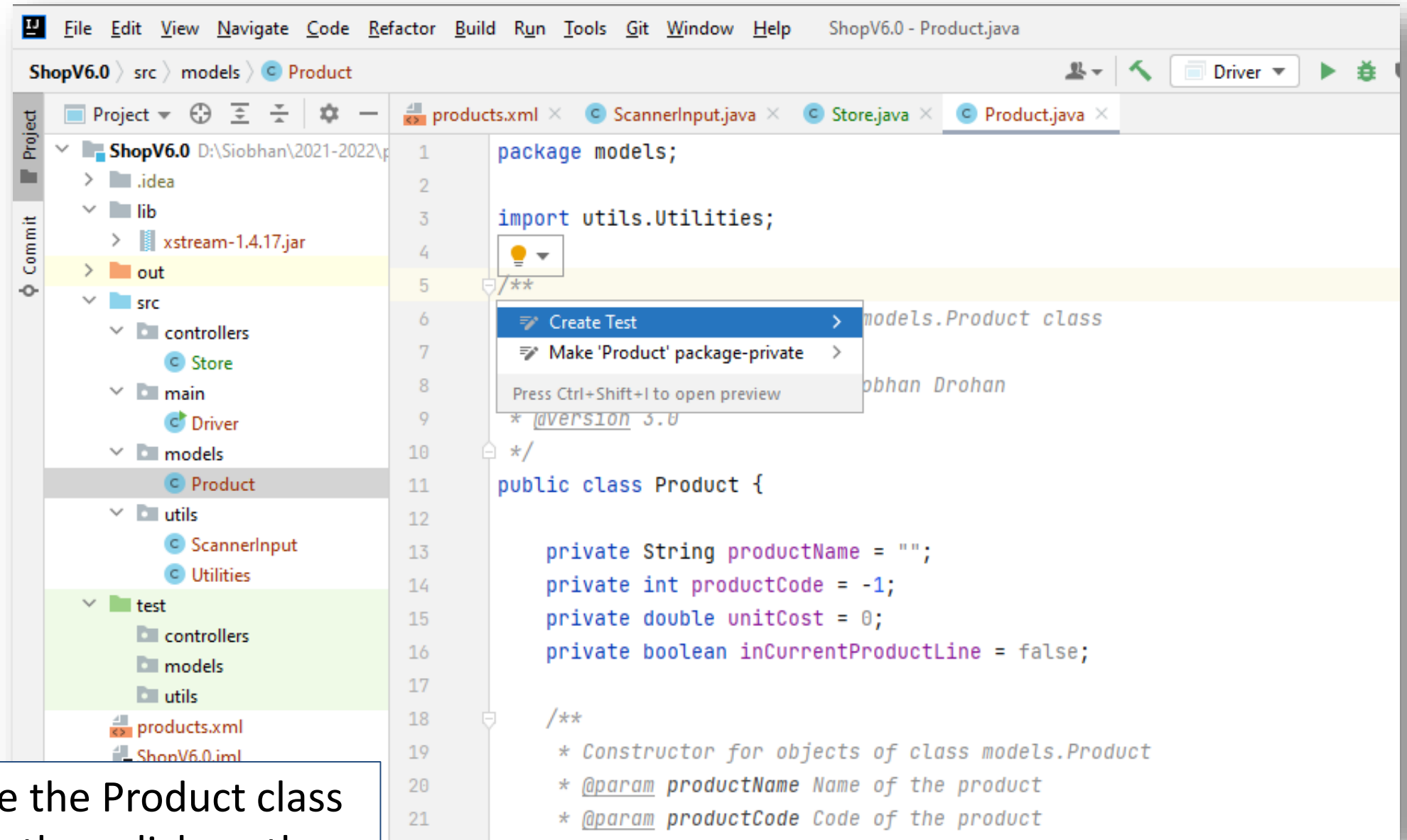
Then “mark” this new
directory as a
Test Sources Root directory



Create three
packages in the
test folder.



Generating a test class for Product.Java ...

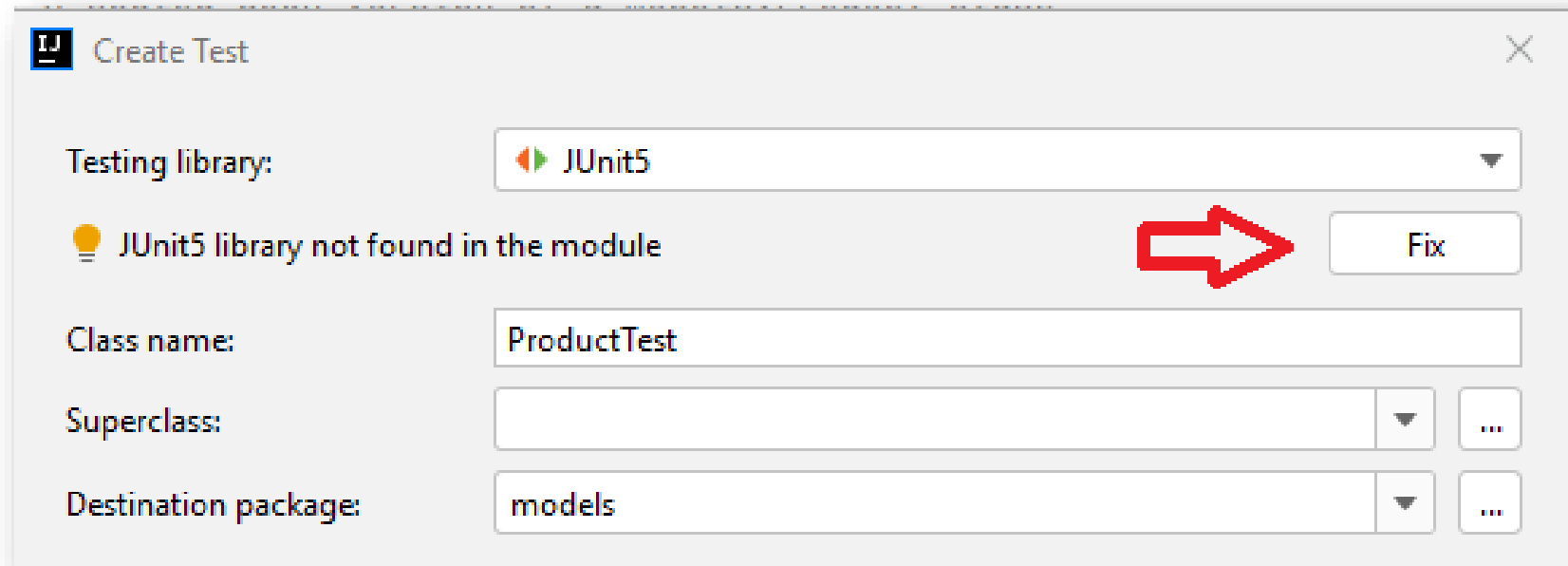


1

Click above the Product class declaration, then click on the light bulb...select **Create Test**

2

Choose JUnit5



Create Test

Testing library: JUnit5

JUnit5 library not found in the module

Fix

Class name: ProductTest

Superclass:

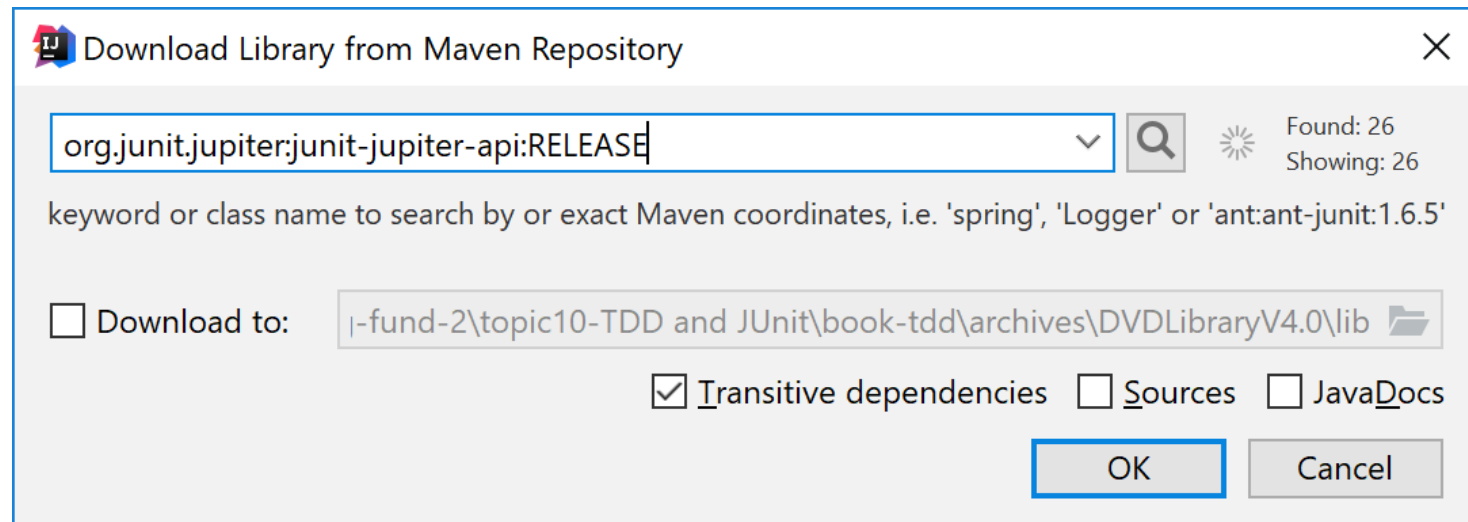
Destination package: models

3

Click the Fix button...

4

...and click the
OK button to
download JUnit5
from Maven



Download Library from Maven Repository

org.junit.jupiter:junit-jupiter-api:RELEASE

Found: 26
Showing: 26


keyword or class name to search by or exact Maven coordinates, i.e. 'spring', 'Logger' or 'ant:ant-junit:1.6.5'

☐ Download to: | -fund-2\topic10-TDD and JUnit\book-tdd\archives\DVDLibraryV4.0\lib

☒ Transitive dependencies ☐ Sources ☐ JavaDocs

OK Cancel



 Create Test ✕

Testing library: JUnit5

Class name: ProductTest

Superclass: ▼ ...

Destination package: models ▼ ...

Generate: ☒ setUp/@Before
☒ tearDown/@After

Generate test methods for: ☐ Show inherited methods

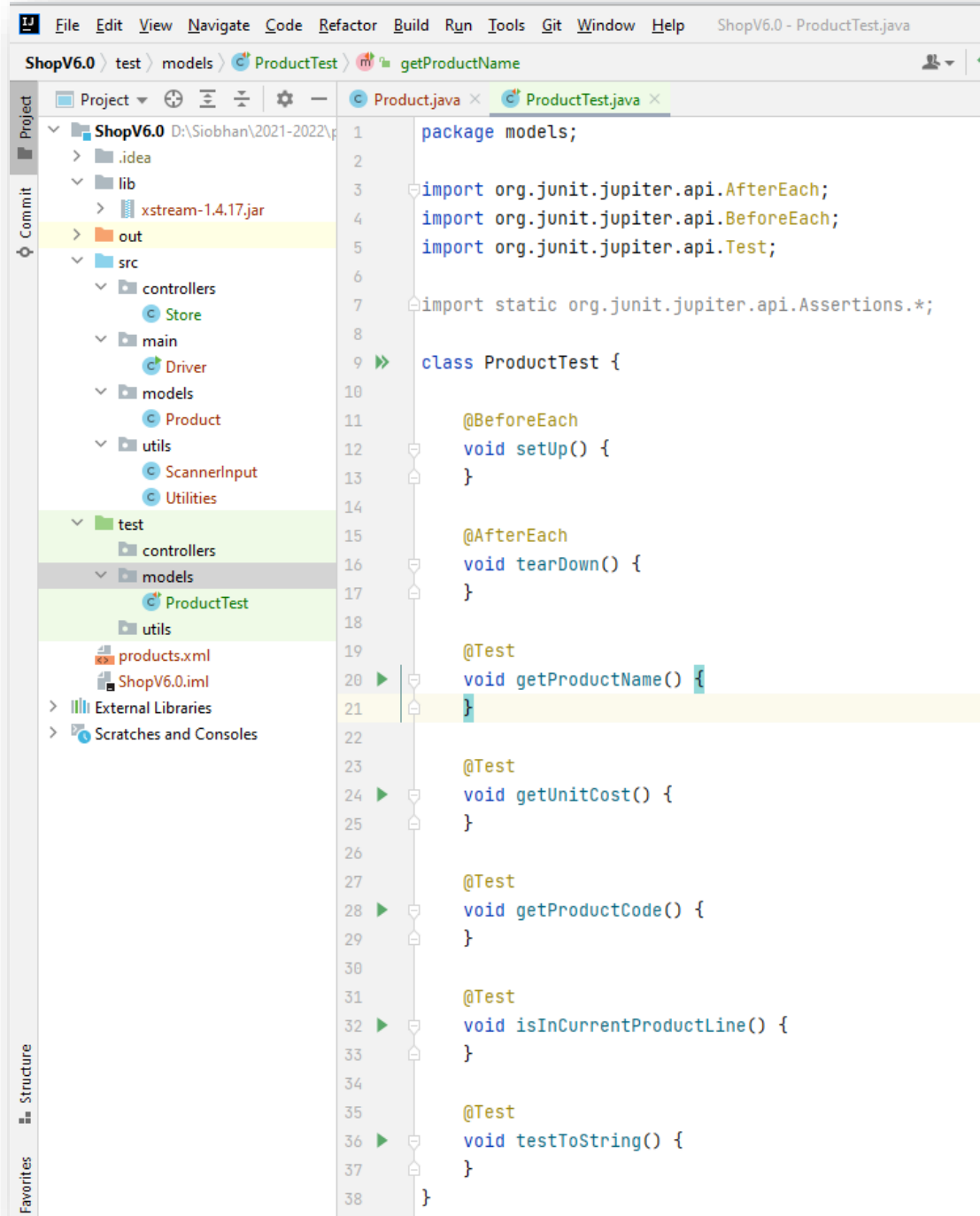
	Member
<input checked="" type="checkbox"/> m 🔗	getProductName():String
<input checked="" type="checkbox"/> m 🔗	getUnitCost():double
<input checked="" type="checkbox"/> m 🔗	getProductCode():int
<input checked="" type="checkbox"/> m 🔗	isInCurrentProductLine():boolean
<input type="checkbox"/> m 🔗	setProductCode(productCode:int):void
<input type="checkbox"/> m 🔗	setProductName(productName:String):void
<input type="checkbox"/> m 🔗	setUnitCost(unitCost:double):void
<input type="checkbox"/> m 🔗	setInCurrentProductLine(inCurrentProductLine:boolean):void
<input checked="" type="checkbox"/> m 🔗 🔗	toString():String

? OK Cancel

5

Now that JUnit5 is downloaded...select the checkboxes as shown to generate test methods.

Note the class name is set to ProductTest.



6

ProductTest.java is generated with starting code.

ProductTest.Java

Testing a Getter Method:
getProductName()

Extract of Product.java

```
public class Product {

    private String productName = "";
    private int productCode = -1;
    private double unitCost = 0;
    private boolean inCurrentProductLine = false;

    public Product(String productName, int productCode, double unitCost, boolean inCurrentProductLine) {
        this.productName = Utilities.truncateString(productName, 20);
        setProductCode(productCode);
        this.unitCost = unitCost;
        this.inCurrentProductLine = inCurrentProductLine;
    }

    public String getProductName(){
        return productName;
    }

    public void setProductName(String productName) {
        if (Utilities.validateStringLength(productName, 20)) {
            this.productName = productName;
        }
    }

    //Code Omitted

}
```

```
private Product productBelow, productExact, productAbove, productZero;

@BeforeEach
void setUp() {
    //name, 19 chars, code 999, unitCost 1, inCurrentProductLine true.
    productBelow = new Product("Television 42Inches", 999, 1, true);
    //name, 20 chars, code 1000, unitCost 999, inCurrentProductLine true.
    productExact = new Product("Television 50 Inches", 1000, 999, true);
    //name, 21 chars, code 10000, unitCost 1000, inCurrentProductLine false.
    productAbove = new Product("Television 60 Inches.", 10000, 1000, false);
    //name, 0 chars, code 9999, unitCost 0, inCurrentProductLine false.
    productZero = new Product("", 9999, 0, false);
}

@AfterEach
void tearDown() {
    productBelow = productExact = productAbove = productZero = null;
}
```

ProductTest.java

Add four Product objects and instantiate them with the details shown.

getProductName()

Refactor the *getProductName* method so that it looks like this:

```
@Test
void getProductName() {
    assertEquals("Television 42Inches", productBelow.getProductName());
    assertEquals("Television 50 Inches", productExact.getProductName());
    assertEquals("Television 60 Inches", productExact.getProductName());
    assertEquals("", productZero.getProductName());
}
```

ProductTest.java

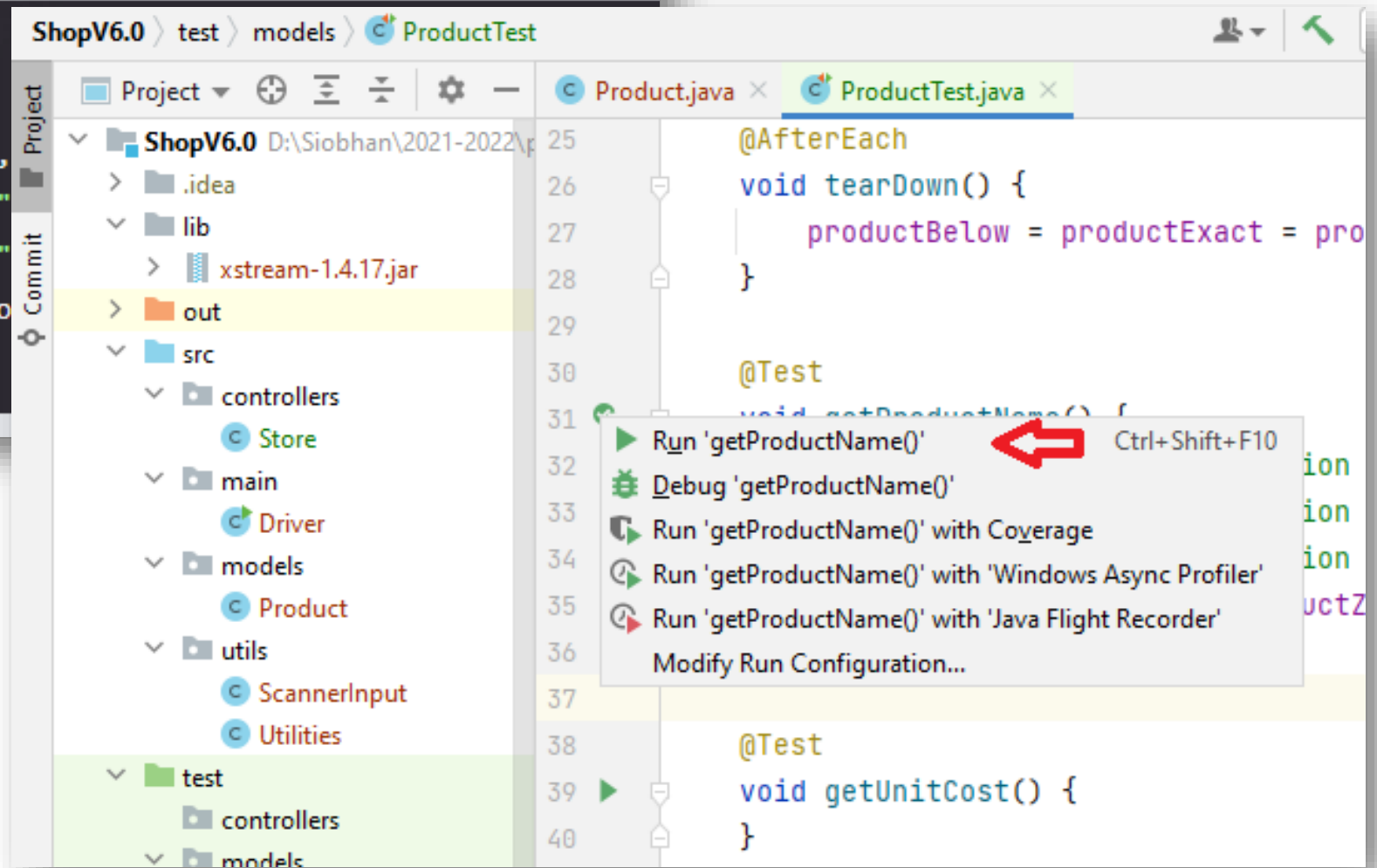
Include four assertions to test the *getProductName()* method

getProductName()

Refactor the `getProductName` method so that it looks like this:

```
@Test
void getProductName() {
    assertEquals("Television 42Inches",
    assertEquals("Television 50 Inches",
    assertEquals("Television 60 Inches",
    assertEquals("", productZero.getPro
}
```

Run the test method:



ShopV6.0 - ProductTest.java

ShopV6.0 > test > models > ProductTest

ProductTest.java

```
25 @AfterEach
26 void tearDown() {
27     productBelow = productExact = productAbove = productZero = null;
28 }
29
30 @Test
31 void getProductName() {
32     assertEquals("expected: 'Television 42Inches'", productBelow.getProductName());
33     assertEquals("expected: 'Television 50 Inches'", productExact.getProductName());
34     assertEquals("expected: 'Television 60 Inches'", productExact.getProductName());
35     assertEquals("expected: ''", productZero.getProductName());
36 }
37
38 @Test
39 void getUnitCost() {
40 }
41
42 @Test
43 void getProductCode() {
44 }
```

Run: ProductTest.getProductName

Tests failed: 1 of 1 test - 17 ms

Test Results

- ProductTest
 - getProductName() 17 ms
 - org.opentest4j.AssertionFailedError:
Expected :Television 60 Inches
Actual :Television 50 Inches
<Click to see difference>
 - <5 internal lines>
 - at models.ProductTest.getProductName(ProductTest.java:34) <31 internal lines>
 - at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 internal lines>
 - at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <23 internal lines>

Process finished with exit code -1

The method
has failed

ShopV6.0 - ProductTest.java

ShopV6.0 > test > models > ProductTest

ProductTest.java

```
25 @AfterEach
26 void tearDown() {
27     productBelow = productExact = productAbove = productZero = null;
28 }
29
30 @Test
31 void getProductName() {
32     assertEquals("expected: Television 42Inches", productBelow.getProductName());
33     assertEquals("expected: Television 50 Inches", productExact.getProductName());
34     assertEquals("expected: Television 60 Inches", productExact.getProductName());
35     assertEquals("expected: ", productZero.getProductName());
36 }
37
38 @Test
39 void getUnitCost() {
40 }
41
42 @Test
43 void getProductCode() {
44 }
```

Run: ProductTest.getProductName

Tests failed: 1 of 1 test - 17 ms

Test Results

- ProductTest
 - getProductName() 17 ms

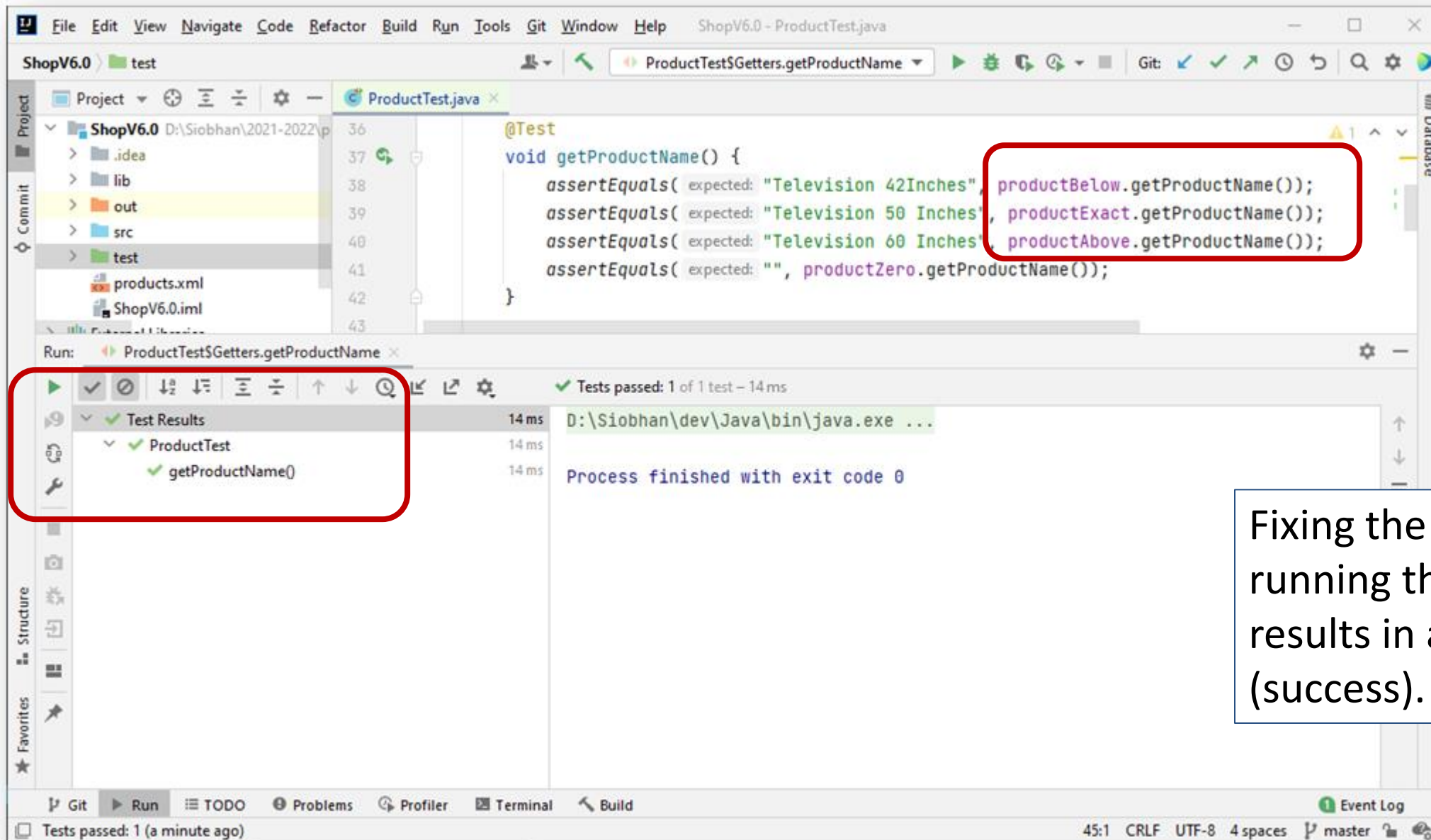
org.opentest4j.AssertionFailedError:
Expected :Television 60 Inches
Actual :Television 50 Inches
<Click to see difference>

<5 internal lines>

- at models.ProductTest.getProductName(ProductTest.java:34) <31 internal lines>
- at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 internal lines>
- at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <23 internal lines>

Process finished with exit code -1

The error is in the third assertion...it should have been the productAbove object and not the productExact one.



Fixing the error and running the test again, results in a green tick (success).

JUnit Interface

Number of tests
that ran and status

Assertion Error

Class name
and line of
code causing
the assertion
failure.

Run: ProductTest.getProductNames x

Tests failed: 1 of 1 test – 17 ms

Test Results 17 ms

- ProductTest 17 ms
 - getProductNames() 17 ms

D:\Siobhan\dev\Java\bin\java.exe ...

org.opentest4j.AssertionFailedError:
Expected :Television 60 Inches
Actual :Television 50 Inches
[Click to see difference](#)

<5 internal lines>

- at models.ProductTest.getProductNames([ProductTest.java:34](#)) <31 internal lines>
- at java.base/java.util.ArrayList.forEach([ArrayList.java:1511](#)) <9 internal lines>
- at java.base/java.util.ArrayList.forEach([ArrayList.java:1511](#)) <23 internal lines>

Process finished with exit code -1

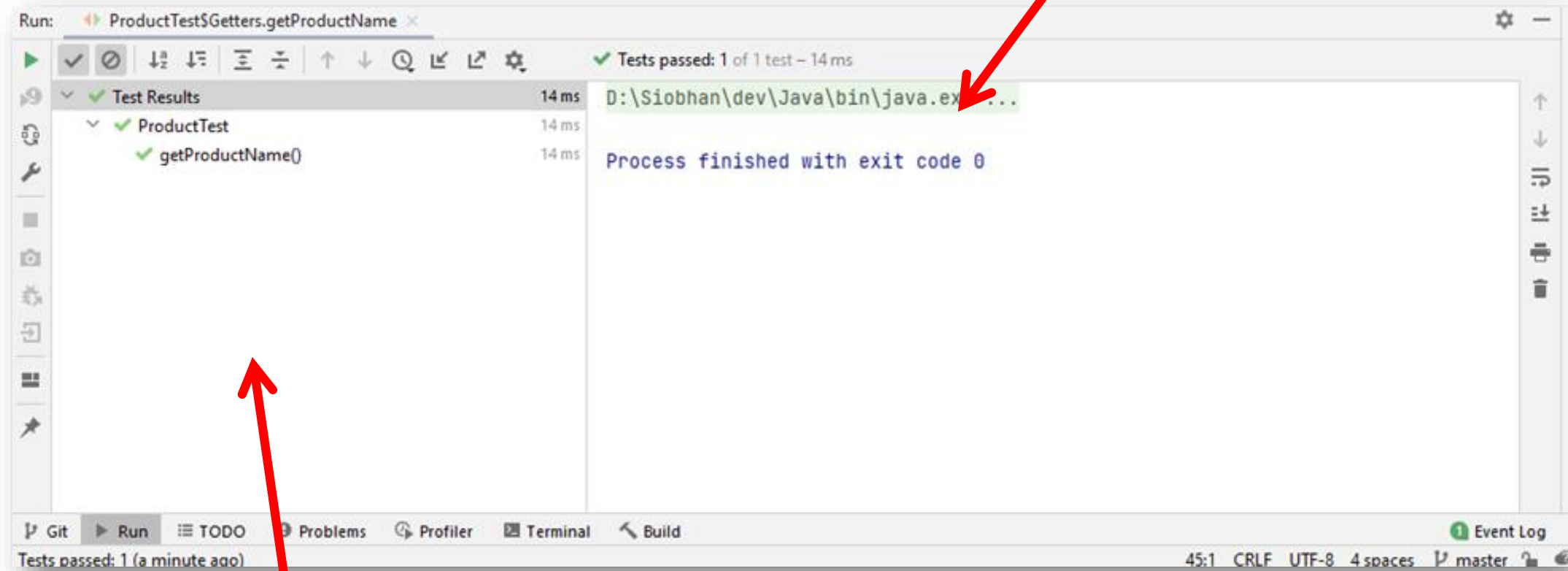
Itemised tests and their
individual result.

Green = Pass

Orange - Failure

Number of tests
that ran and status

Exit Code 0 – Success



Itemised tests and their
individual result.

Green = Pass

Orange - Failure

ProductTest.Java

Testing a Setter Method:
setProductName()

Extract of Product.java

```
public class Product {

    private String productName = "";
    private int productCode = -1;
    private double unitCost = 0;
    private boolean inCurrentProductLine = false;

    public Product(String productName, int productCode, double unitCost, boolean inCurrentProductLine) {
        this.productName = Utilities.truncateString(productName, 20);
        setProductCode(productCode);
        this.unitCost = unitCost;
        this.inCurrentProductLine = inCurrentProductLine;
    }

    public String getProductName(){
        return productName;
    }

    public void setProductName(String productName) {
        if (Utilities.validateStringLength(productName, 20)) {
            this.productName = productName;
        }
    }

    //Code Omitted

}
```

```
@Test
void setProductName() {
    assertEquals("Television 42Inches", productBelow.getProductName());

    productBelow.setProductName("iPhone 13 Charcoal."); //19 chars - update performed
    assertEquals("iPhone 13 Charcoal.", productBelow.getProductName());

    productBelow.setProductName("iPhone 12 - Charcoal"); //20 chars - update performed
    assertEquals("iPhone 12 - Charcoal", productBelow.getProductName());

    productBelow.setProductName("iPhone 11: - Charcoal"); //21 chars - update ignored
    assertEquals("iPhone 12 - Charcoal", productBelow.getProductName());
}
```

ProductTest.Java

Testing the toString Method

Extract of Product.java

```
public class Product {

    private String productName = "";
    private int productCode = -1;
    private double unitCost = 0;
    private boolean inCurrentProductLine = false;

    public Product(String productName, int productCode, double unitCost, boolean inCurrentProductLine) {
        this.productName = Utilities.truncateString(productName, 20);
        setProductCode(productCode);
        this.unitCost = unitCost;
        this.inCurrentProductLine = inCurrentProductLine;
    }

    public String toString()
    {
        return "Product description: " + productName
            + ", product code: " + productCode
            + ", unit cost: " + unitCost
            + ", currently in product line: " + Utilities.booleanToYN(inCurrentProductLine);
    }

    //Code Omitted

}
```

```
@Test
```

```
void testToString() {
```

```
    String toStringContents = productExact.toString();
```

```
    assertTrue(toStringContents.contains("Product description: " + productExact.getProductName()));
```

```
    assertTrue(toStringContents.contains("product code: " + productExact.getProductCode()));
```

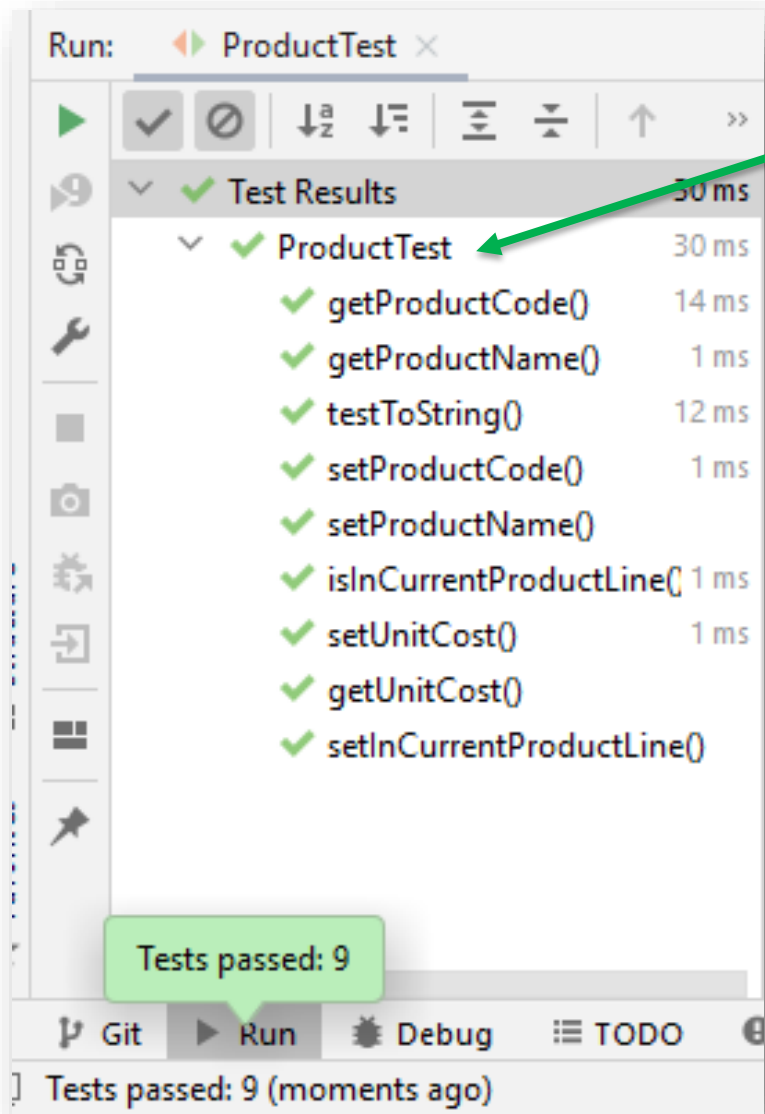
```
    assertTrue(toStringContents.contains("unit cost: " + productExact.getUnitCost()));
```

```
    assertTrue(toStringContents.contains(" currently in product line: Y"));
```

```
}
```


Tidying up the JUnit Interface

As you add and run more tests to ProductTest.java, they are displayed under the **ProductTest** heading.



As you add and run more tests to ProductTest.java, they are displayed under the **ProductTest** heading.

Run: ProductTest x

Test Results 30 ms

- ProductTest 30 ms
 - getProductCode() 14 ms
 - getProductName() 1 ms
 - testToString() 12 ms
 - setProductCode() 1 ms
 - setProductName() 1 ms
 - isInCurrentProductLine() 1 ms
 - setUnitCost() 1 ms
 - getUnitCost()
 - setInCurrentProductLine()

Tests passed: 9

Git Run Debug TODO

Tests passed: 9 (moments ago)

But this layout would be much easier to read...especially as more and more test methods are added...

Run: ProductTest x

Test Results 34 ms

- ProductTest 34 ms
 - testToString() 25 ms
 - Setters 6 ms
 - setProductCode() 2 ms
 - setProductName() 2 ms
 - setUnitCost() 1 ms
 - setInCurrentProductLine() 1 ms
 - Getters 3 ms
 - getProductCode() 1 ms
 - getProductName() 1 ms
 - isInCurrentProductLine() 1 ms
 - getUnitCost() 1 ms

To get this layout, we will need to use the `@Nested` class approach WITHIN `ProductTest` e.g.

```
@Nested
class Getters {

    //getter methods go in here

}
```

But this layout would be much easier to read...especially as more and more test methods are added...

Run: ProductTest

Test Results		34 ms
ProductTest		34 ms
testToString()		25 ms
Setters		6 ms
setProductCode()		2 ms
setProductName()		2 ms
setUnitCost()		1 ms
setInCurrentProductLine()		1 ms
Getters		3 ms
getProductCode()		
getProductName()		1 ms
isInCurrentProductLine()		1 ms
getUnitCost()		1 ms

To get this layout, we will need to use the **@Nested** class approach WITHIN ProductTest e.g.

```
@Nested
class Getters {

    //getter methods go in here

}
```

```
class ProductTest {

    @Nested
    class Getters {

        @Test
        void getProductName() {
            assertEquals("Television 42Inches", productBelow.getProductName());
            assertEquals("Television 50 Inches", productExact.getProductName());
            assertEquals("Television 60 Inches", productAbove.getProductName());
            assertEquals("", productZero.getProductName());
        }

        @Test
        void getUnitCost() {
            assertEquals(1, productBelow.getUnitCost());
            assertEquals(999, productExact.getUnitCost());
            assertEquals(1000, productAbove.getUnitCost());
            assertEquals(0, productZero.getUnitCost());
        }

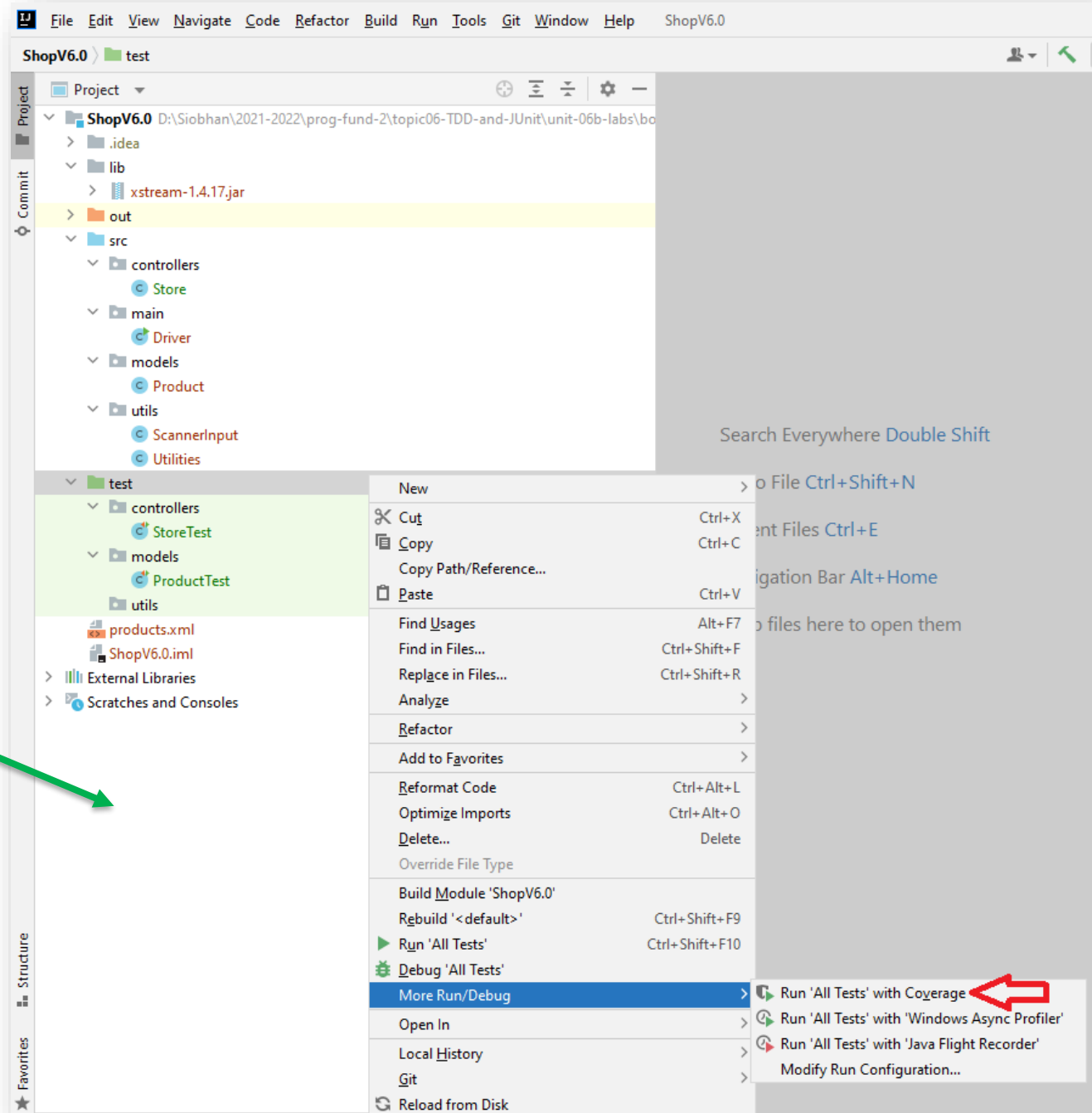
        @Test
        void getProductCode() {
            assertEquals(-1, productBelow.getProductCode());
            assertEquals(1000, productExact.getProductCode());
            assertEquals(-1, productAbove.getProductCode());
            assertEquals(9999, productZero.getProductCode());
        }

        @Test
        void isInCurrentProductLine() {
            assertTrue(productBelow.isInCurrentProductLine());
            assertTrue(productExact.isInCurrentProductLine());
            assertFalse(productAbove.isInCurrentProductLine());
            assertFalse(productZero.isInCurrentProductLine());
        }
    }
}
```

ProductTest.Java

Have we tested everything in
Product.java?

We can run a Coverage Report with our tests to see what code is tested and what isn't.



Percentages are applied to each class.

The screenshot displays the IntelliJ IDEA IDE with the following components:

- Project View (Left):** Shows the project structure for ShopV6.0. The 'src' directory is expanded, showing subdirectories like controllers, main, models, and utils, each with associated class and line coverage percentages.
- Code Editor (Center):** Displays the source code for ProductTest.java. The code includes package declarations, imports for JUnit and utilities, and a class definition with test methods.
- Coverage Table (Right):** A table titled '60% classes, 28% lines covered in 'all classes in scope''. It lists various packages and their coverage statistics.
- Run Console (Bottom):** Shows the execution results of the tests, indicating that all 25 tests passed within 56 ms.

Element	Class, %	Method, %	Line, %
com			
controllers	100% (1/1)	53% (8/15)	35% (25/7...
images			
java			
javax			
jdk			
main	0% (0/1)	0% (0/14)	0% (0/75)
META-INF			
models	100% (1/1)	100% (10/...	100% (21/...
netscape			
org			
sun			
toolbarButtonGra...			
utils	50% (1/2)	60% (6/10)	30% (8/26)

Run: All in ShopV6.0

Tests passed: 25 of 25 tests – 56 ms

56 ms <default package>
47 ms ProductTest
38 ms testToString()
7 ms Setters
2 ms Getters
8 ms StoreTest
3 ms FindAndSearch
5 ms ArrayListCRUD
1 ms UtilitiesTest

Process finished with exit code 0

Percentages are applied to each class.

The screenshot displays the IntelliJ IDEA IDE with the following components:

- Project View (Left):** Shows the project structure for ShopV6.0. The 'src' directory is expanded, showing sub-packages like controllers, main, models, and utils. Each package has associated coverage percentages (e.g., 'src' is 60% classes, 28% lines covered).
- Code Editor (Center):** Displays the code for ProductTest.java. The code includes package declarations, imports for JUnit, and a class definition with test methods.
- Coverage Table (Right):** A table titled 'Coverage: All in ShopV6.0' showing coverage for 60 classes. The table has columns for Element, Class, %, Method, %, and Line, %. The 'com' package is highlighted.
- Run Console (Bottom):** Shows the execution results of the tests. It indicates that 25 of 25 tests passed in 56 ms. The console output includes the command used to run the tests and the final exit code.

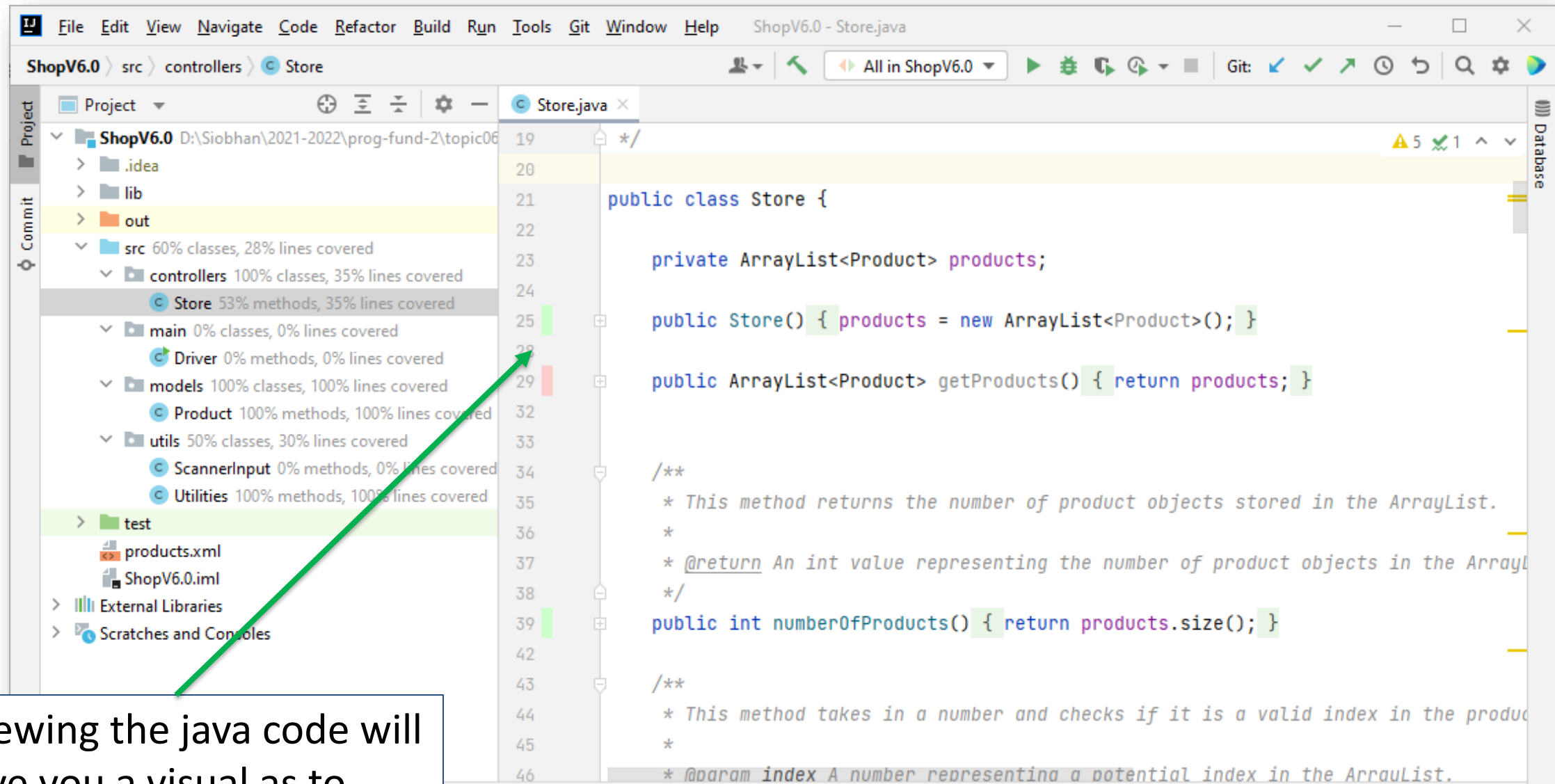
Element	Class, %	Method, %	Line, %
com			
controllers	100% (1/1)	53% (8/15)	35% (25/7...)
images			
java			
javax			
jdk			
main	0% (0/1)	0% (0/14)	0% (0/75)
META-INF			
models	100% (1/1)	100% (10/...	100% (21/...
netscape			
org			
sun			
toolbarButtonGra...			
utils	50% (1/2)	60% (6/10)	30% (8/26)

Run: All in ShopV6.0

Tests passed: 25 of 25 tests - 56 ms

56 ms <default package>
47 ms ProductTest
38 ms testToString()
7 ms Setters
2 ms Getters
8 ms StoreTest
3 ms FindAndSearch
5 ms ArrayListCRUD
1 ms UtilitiesTest

D:\Siobhan\dev\Java\bin\java.exe ...
---- IntelliJ IDEA coverage runner ----
sampling ...
include patterns:
exclude patterns:
Class transformation time: 0.1445s for 797 classes or 1.8130489335006273E-4s per
Process finished with exit code 0



Viewing the java code will give you a visual as to what code has been **tested** and what **hasn't**.

**Any
Questions?**

