Test Driven Development

Using JUnit to Test Product.java

Produced by: Dr. Siobhán Drohan
Mairead Meagher
Siobhan Roche

Ollscoil Teicneolaíochta an Oirdheiscirt
South East Technological University

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

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 in
        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;
    }
```

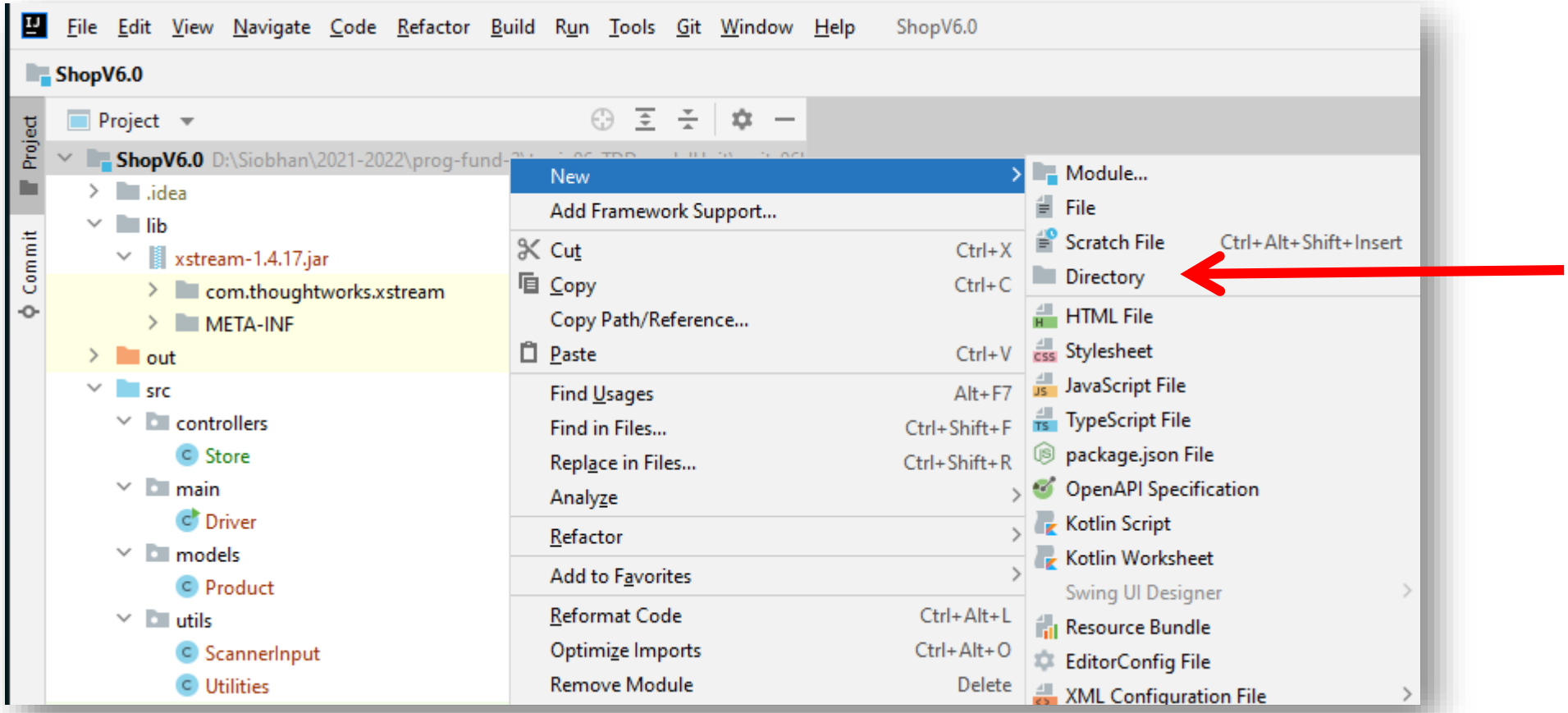
Utilities.java

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

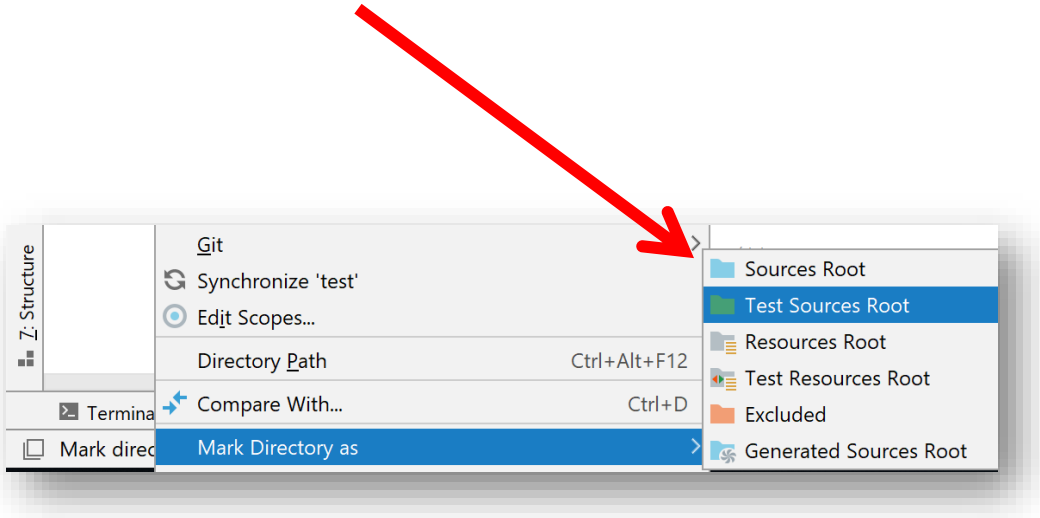
```
public class Utilities {

    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…

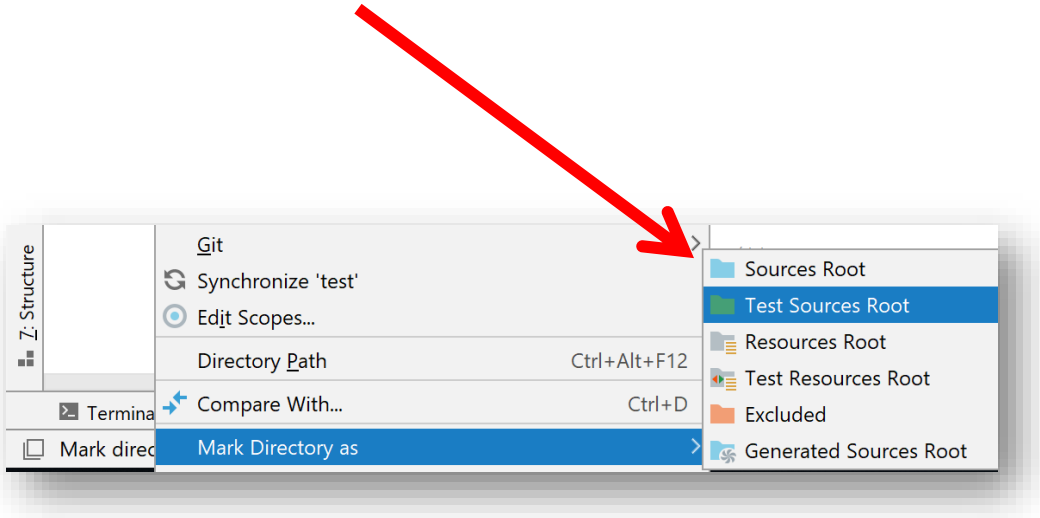
In IntelliJ, create a new directory and call it test



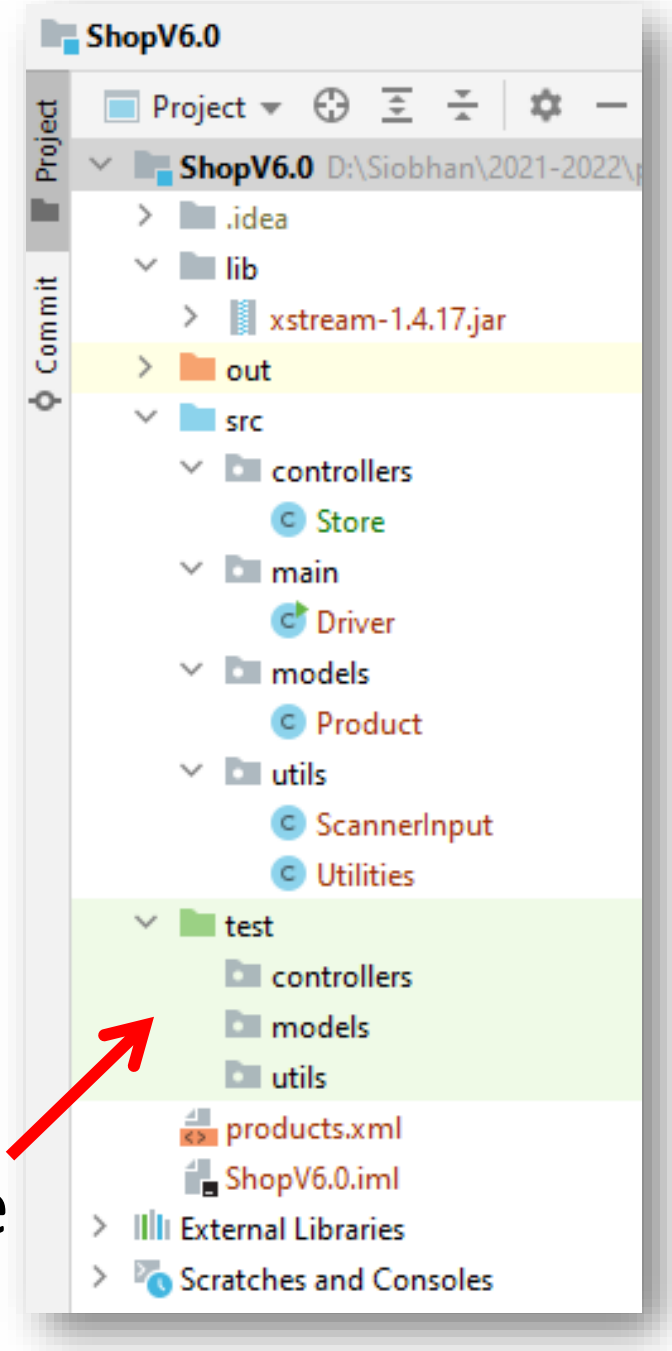
Then “mark” this new directory as a **Test Sources Root** directory



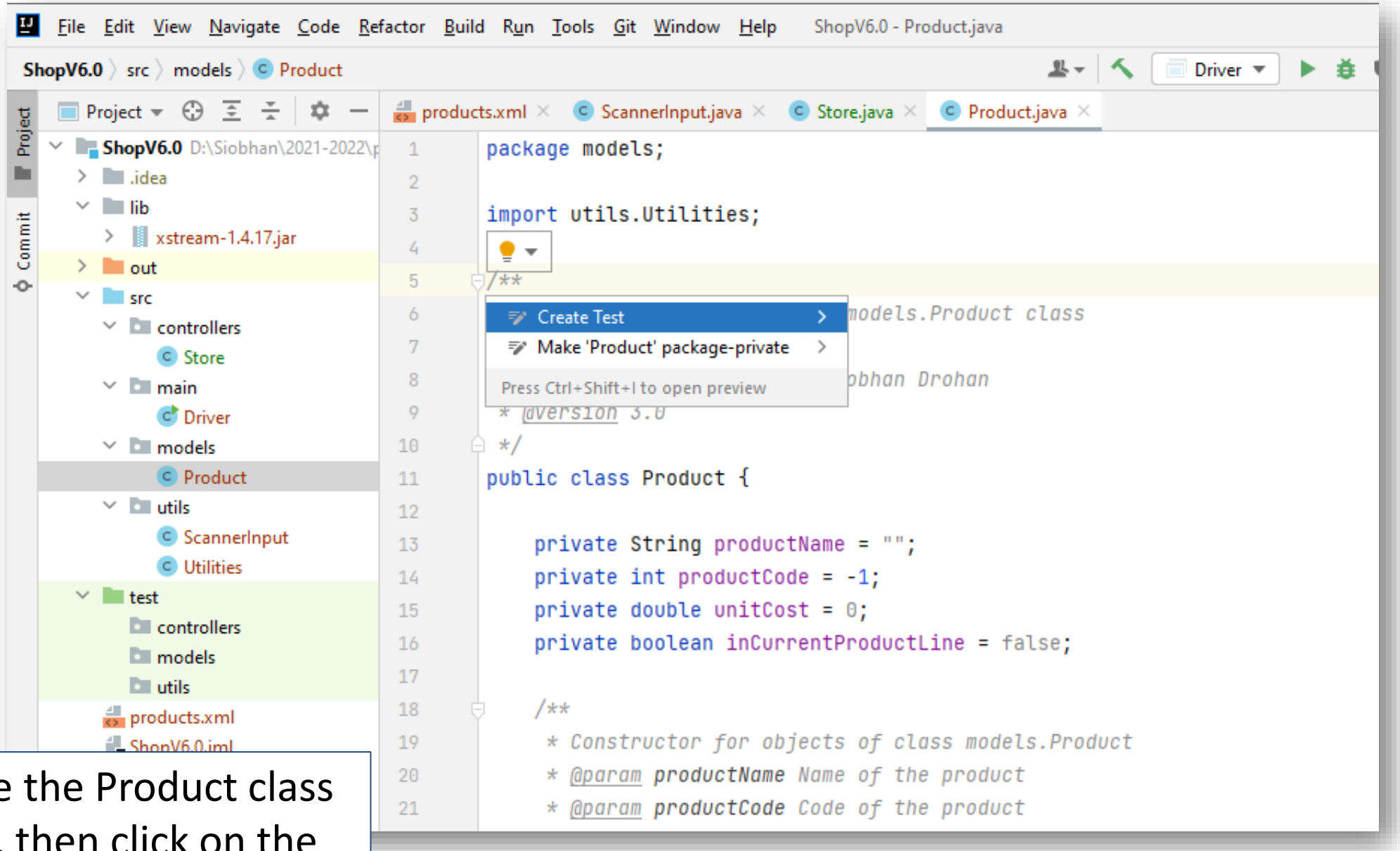
Then “mark” this new directory as a **Test Sources Root** directory



Create three packages in the test folder.



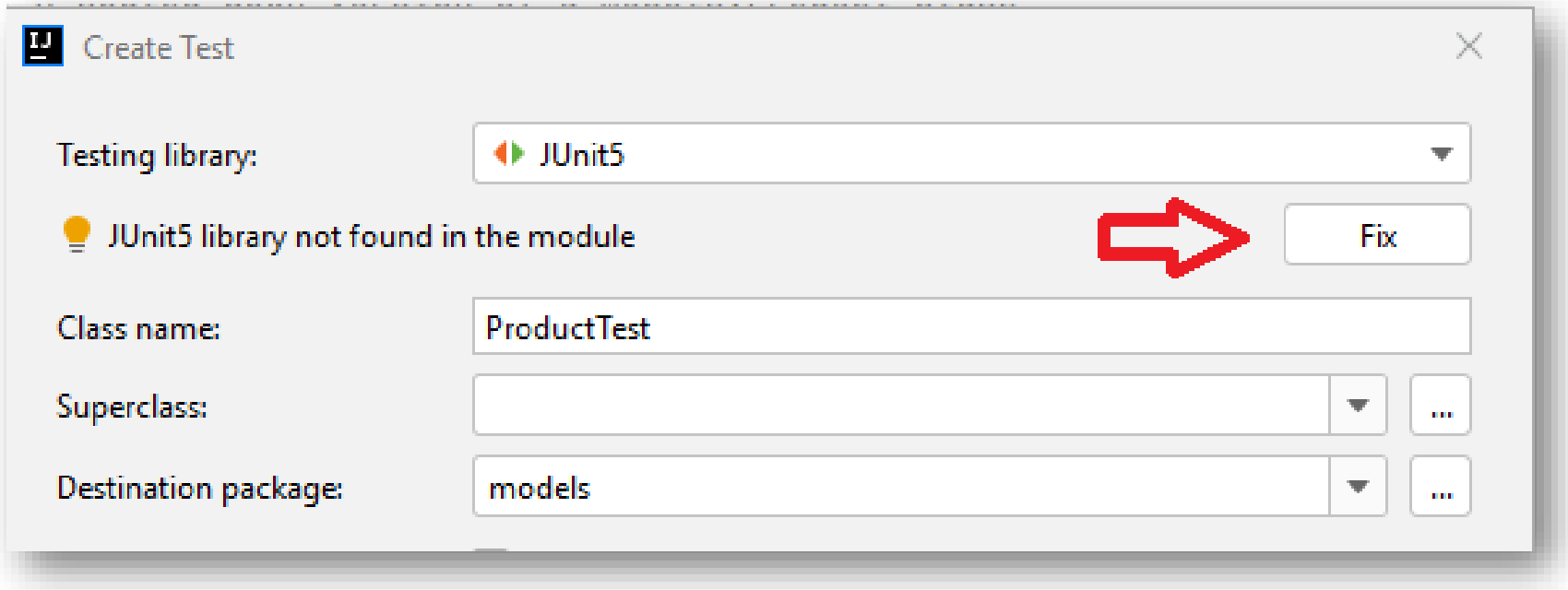
Generating a test class for Product.Java …



1

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

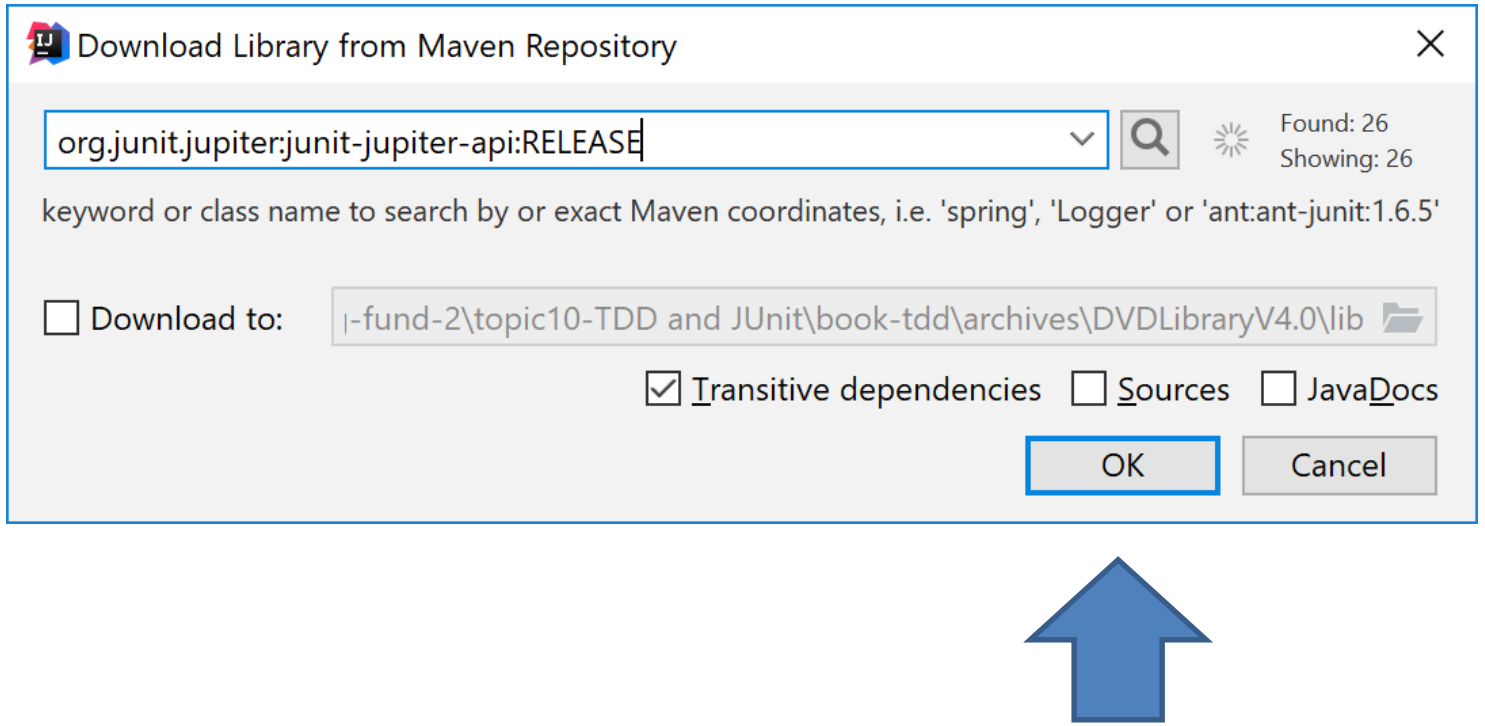
2 Choose JUnit5

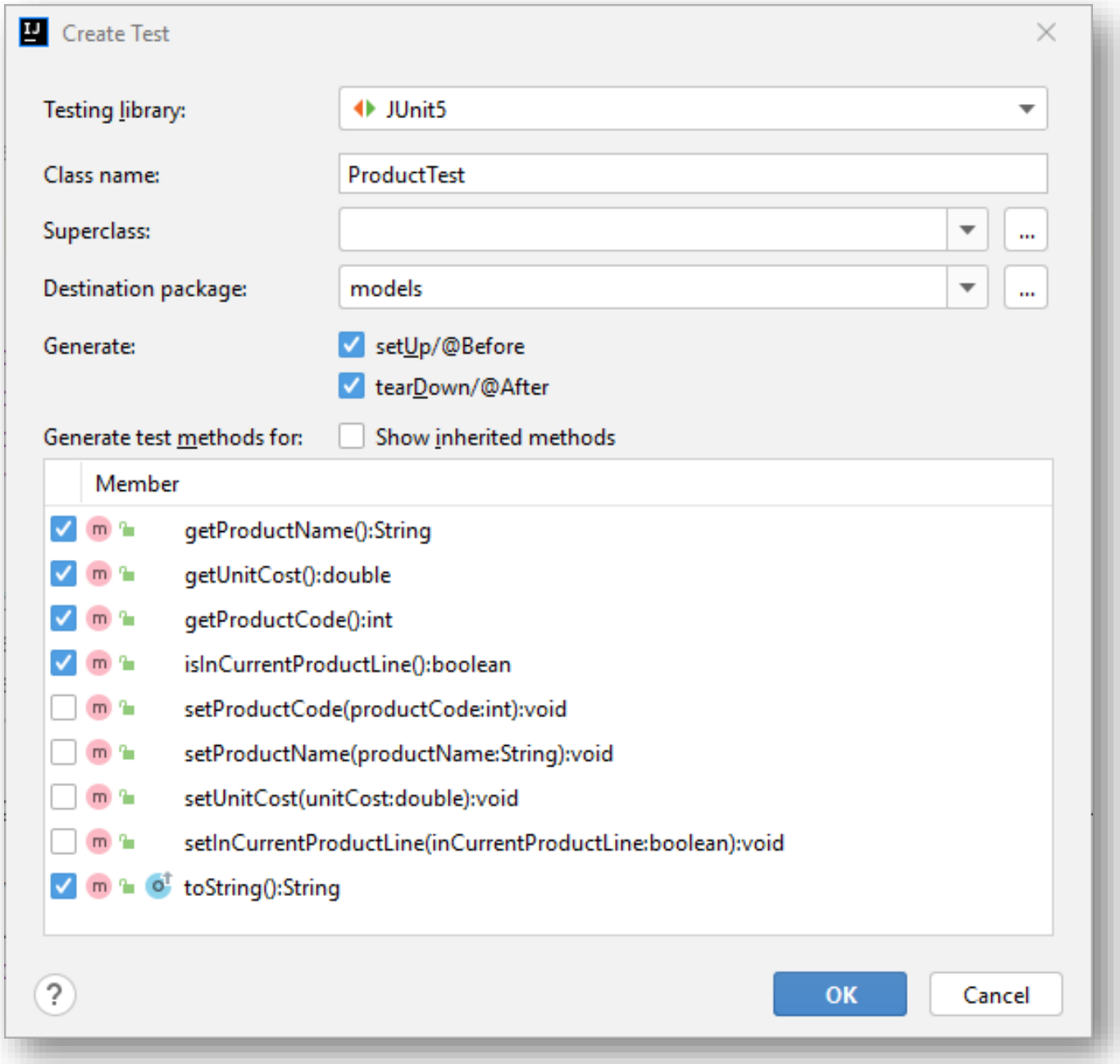


3 Click the Fix button…

4

…and click the OK button do download JUnit5 from Maven

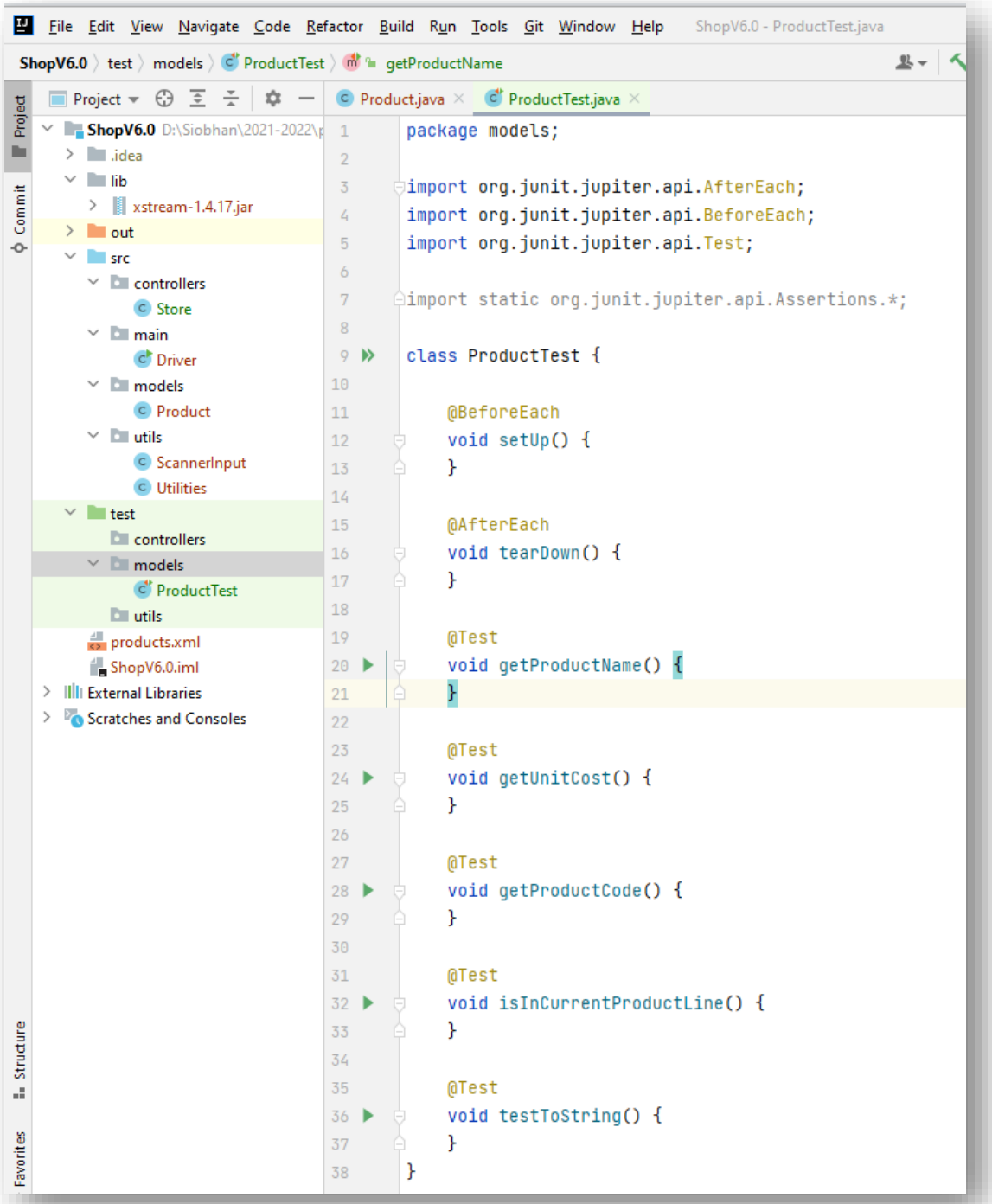




5

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 inCurrentProductLi
        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

}
```

```java
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

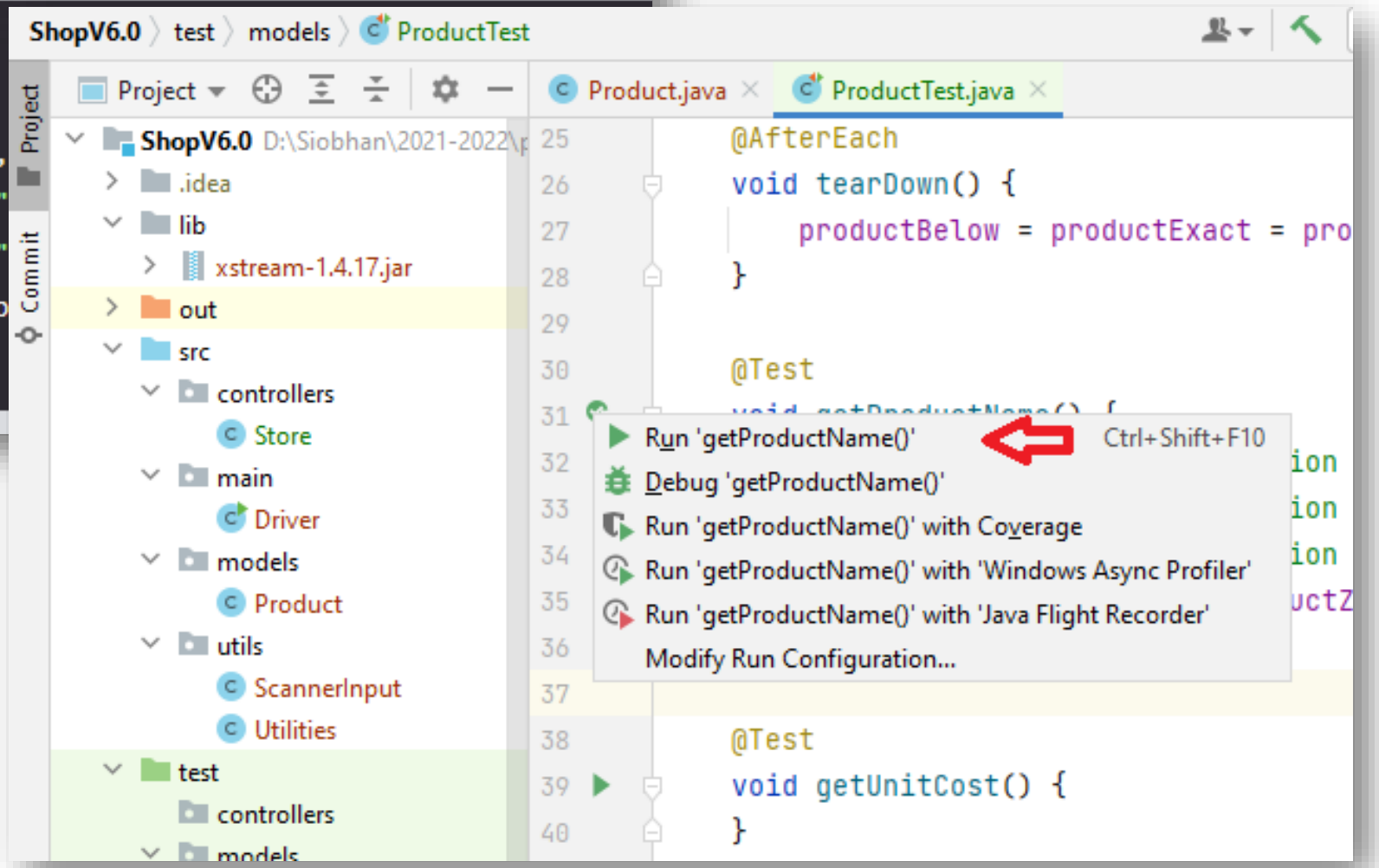
Include four assertions to test the getProductName() method

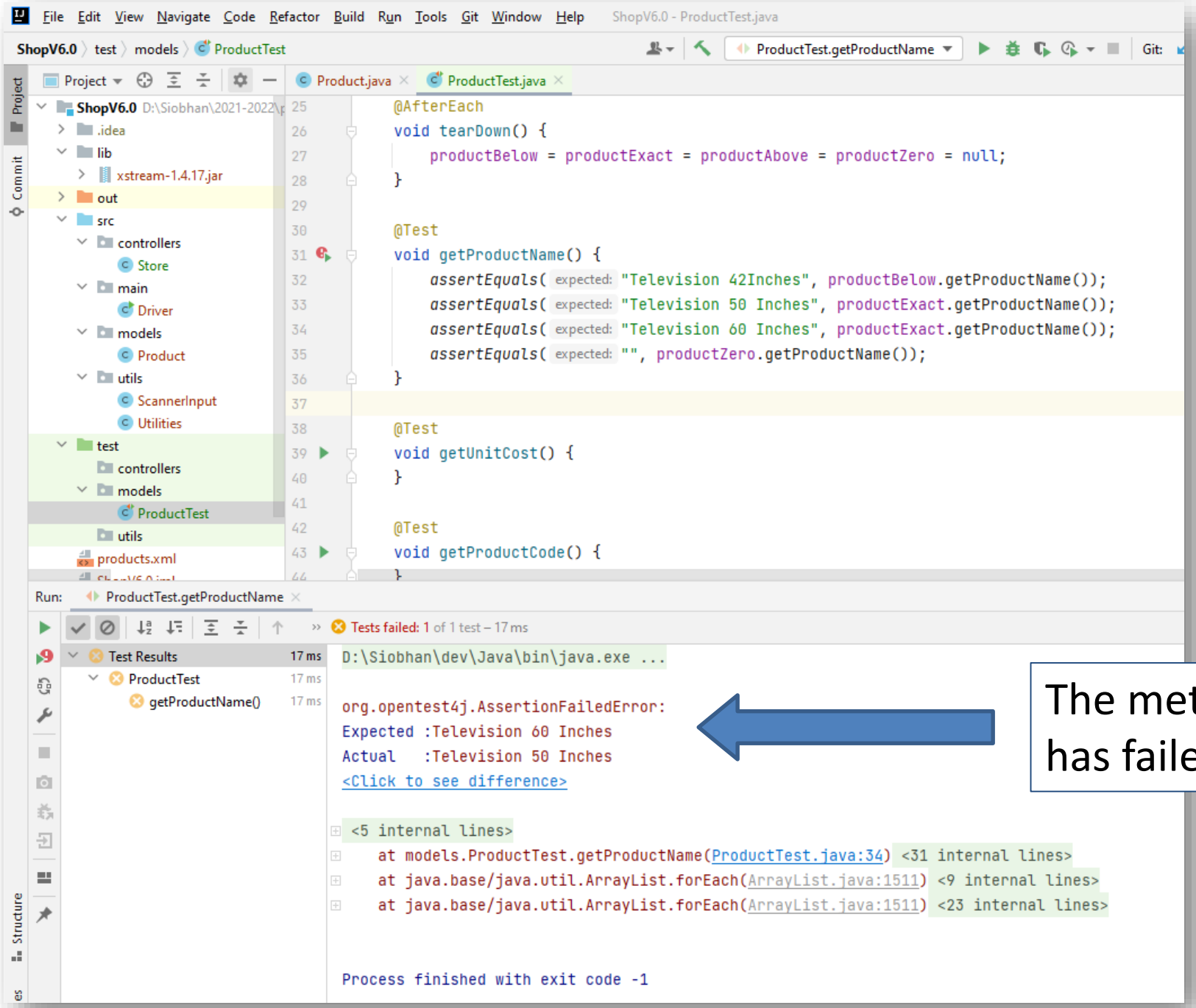
getProductName()

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

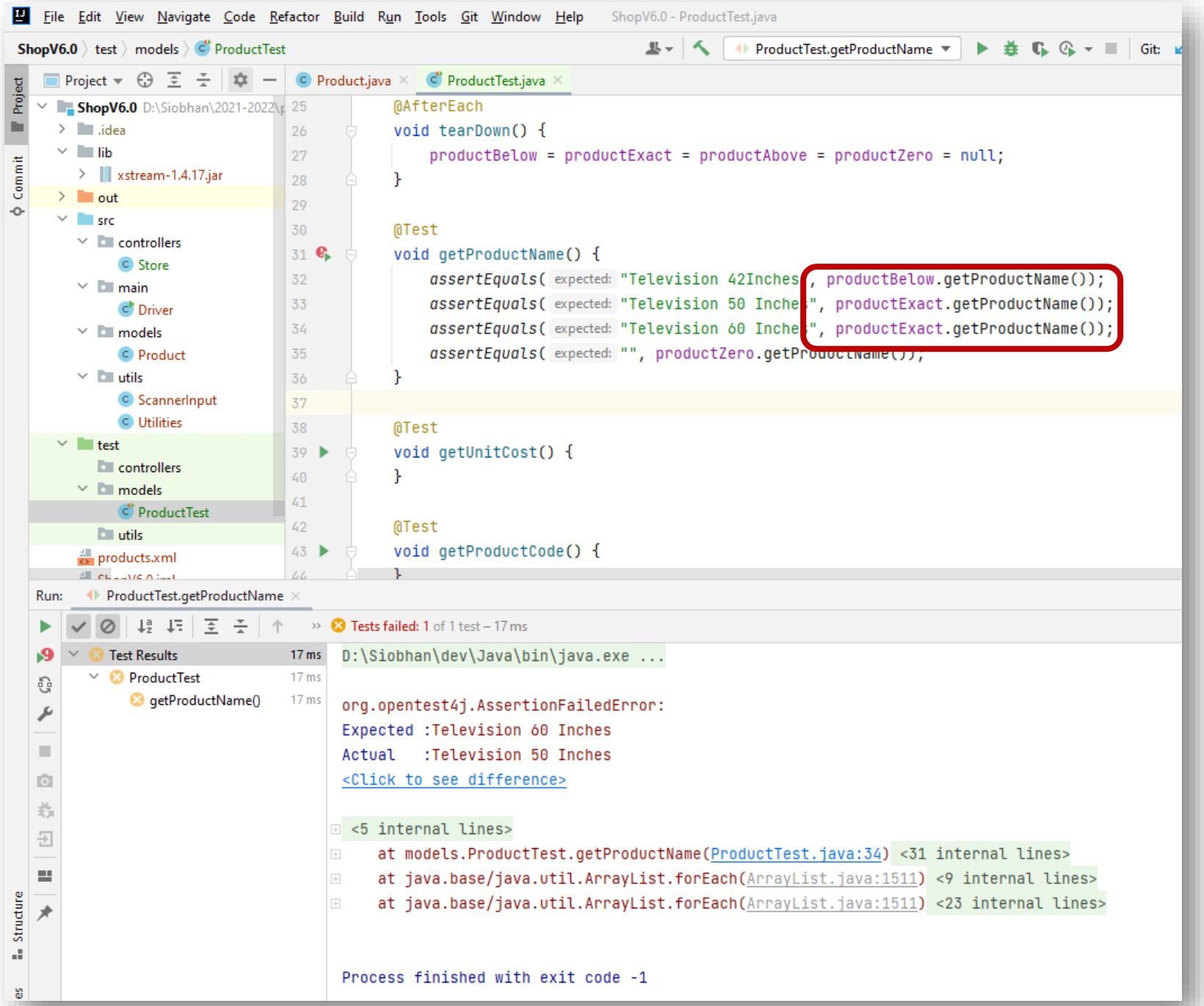
```
@Test
void getProductName() {
    assertEquals("Television 42Inches",
    assertEquals("Television 50 Inches"
    assertEquals("Television 60 Inches"
    assertEquals("", productZero.getPro
}
```

Run the test method:

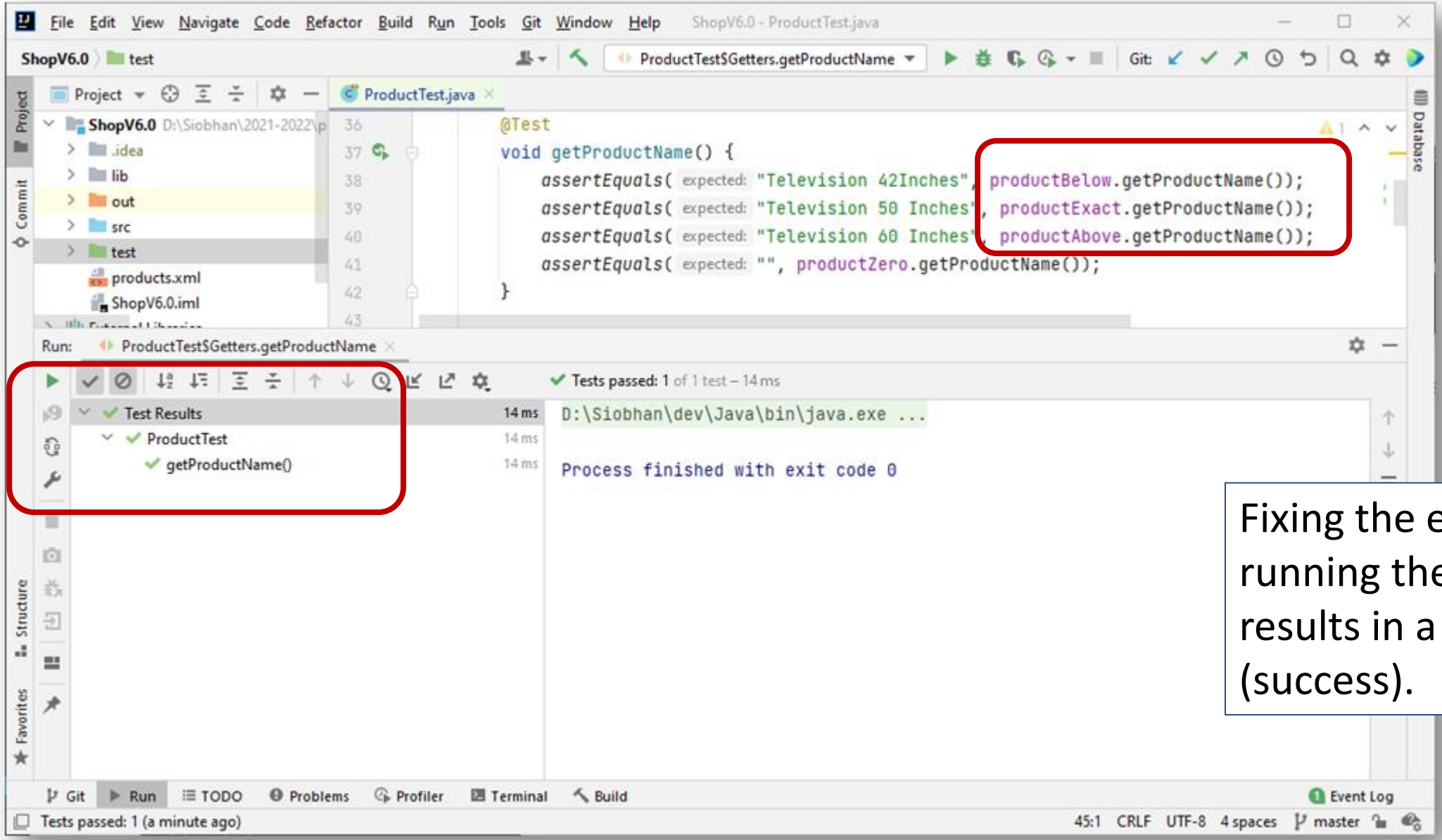




The method has failed

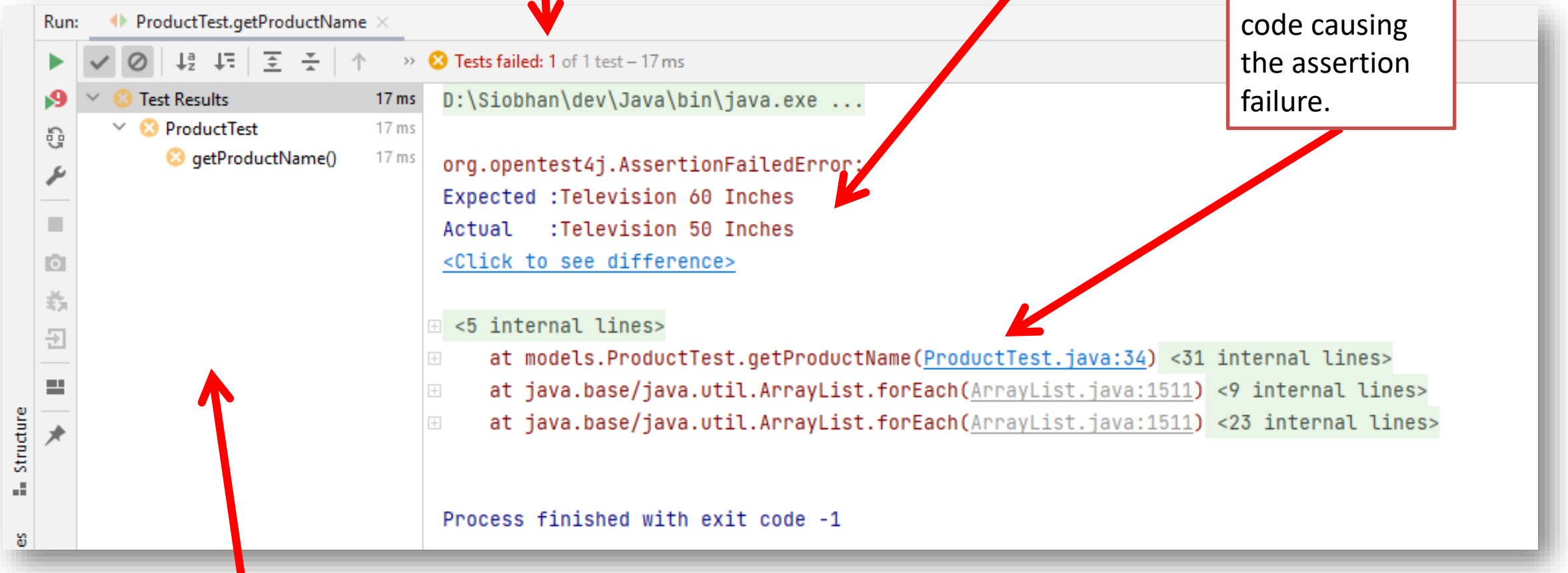


The error is in the third assertion…it should have been the productAbove object and not the productExact one.



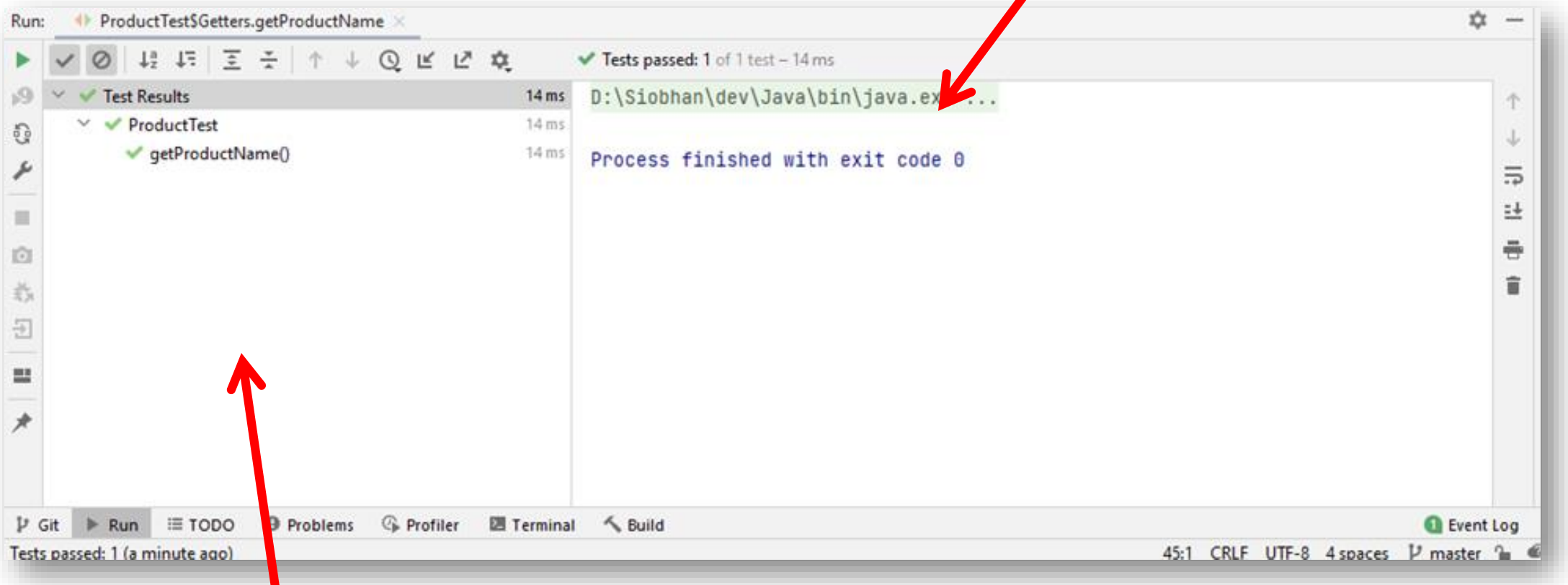
Fixing the error and running the test again, results in a green tick (success).

JUnit Interface


Number of tests that ran and status
Assertion Error
Class name and line of code causing the assertion failure.
Run: ProductTest.getProductName
Tests failed: 1 of 1 test – 17 ms
Test Results 17 ms
ProductTest 17 ms
getProductName() 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.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
Structure
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 inCurrentProductLi
        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

}
```

```java
@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

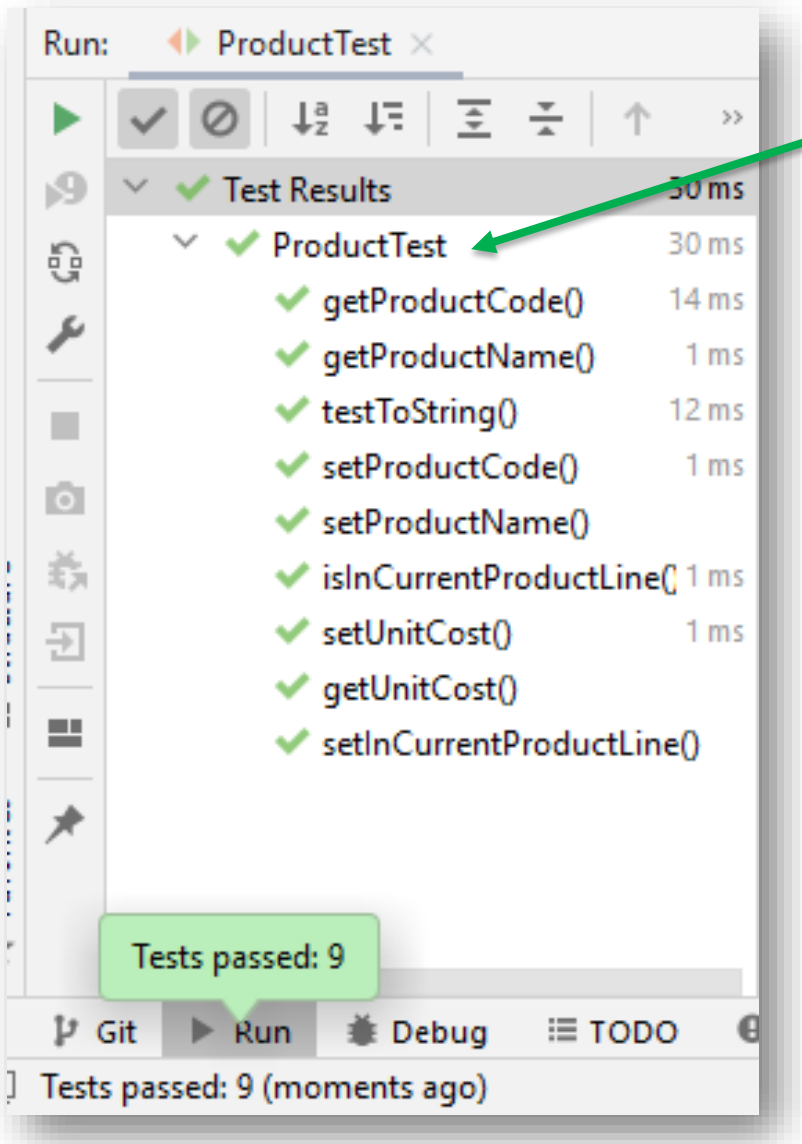
Testing the toString Method

Extract of Product.java

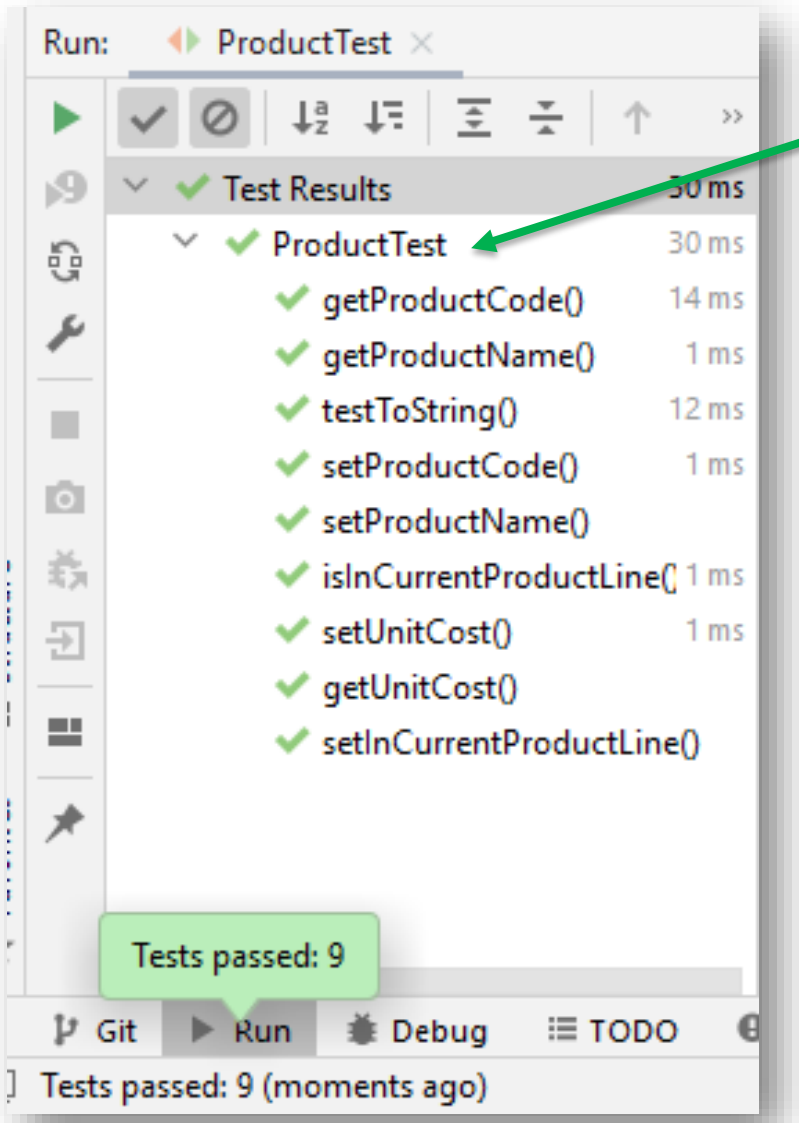
```
public class Product {

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

    public Product(String productName, int productCode, double unitCost, boolean inCurrentProductLi
        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

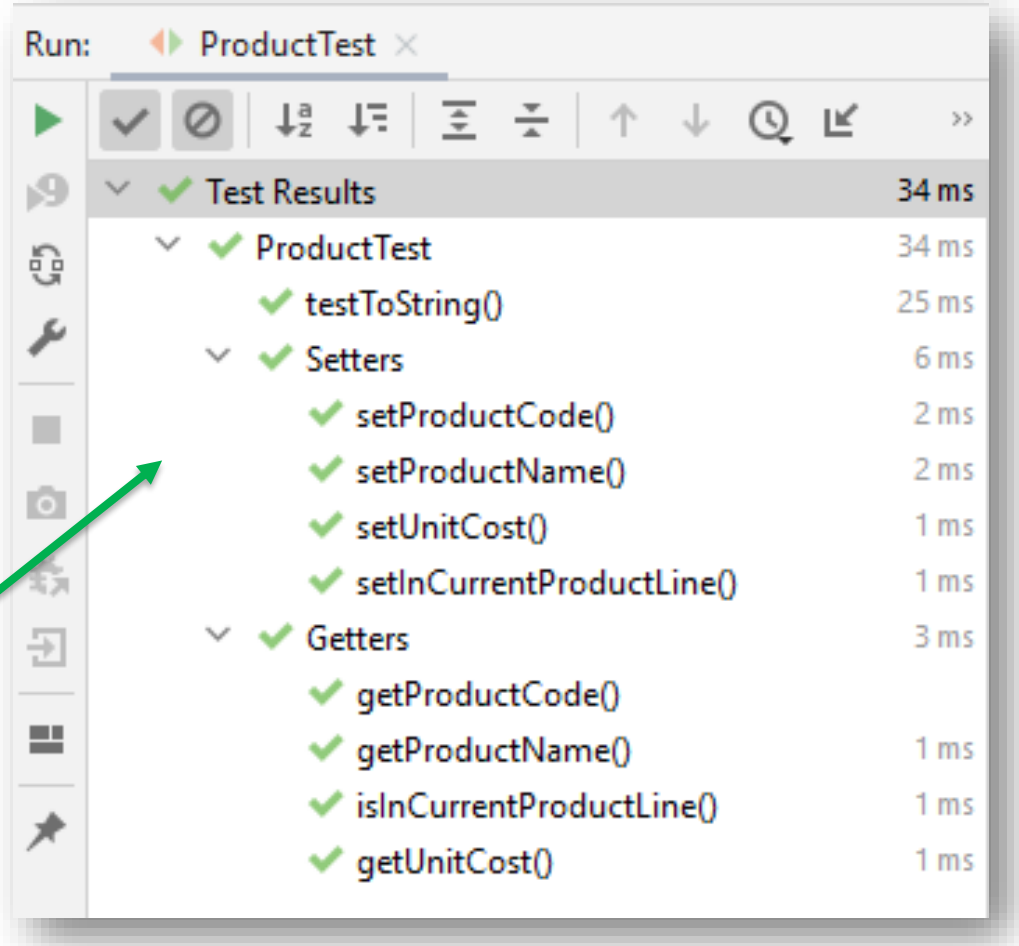


As you add and run more tests to ProductTest.java, they are displayed under the ProductTest heading.



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

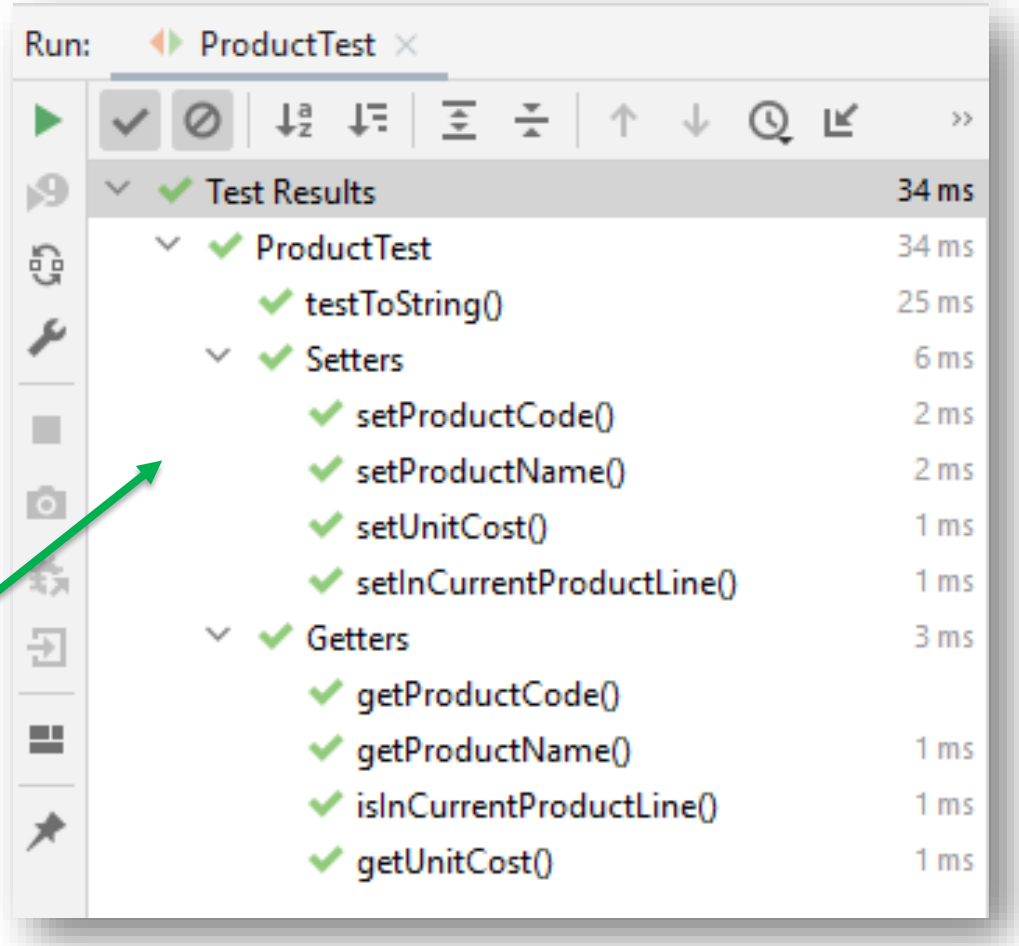
But this layout would be much easier to read…especially as more and more test methods are added…



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...



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

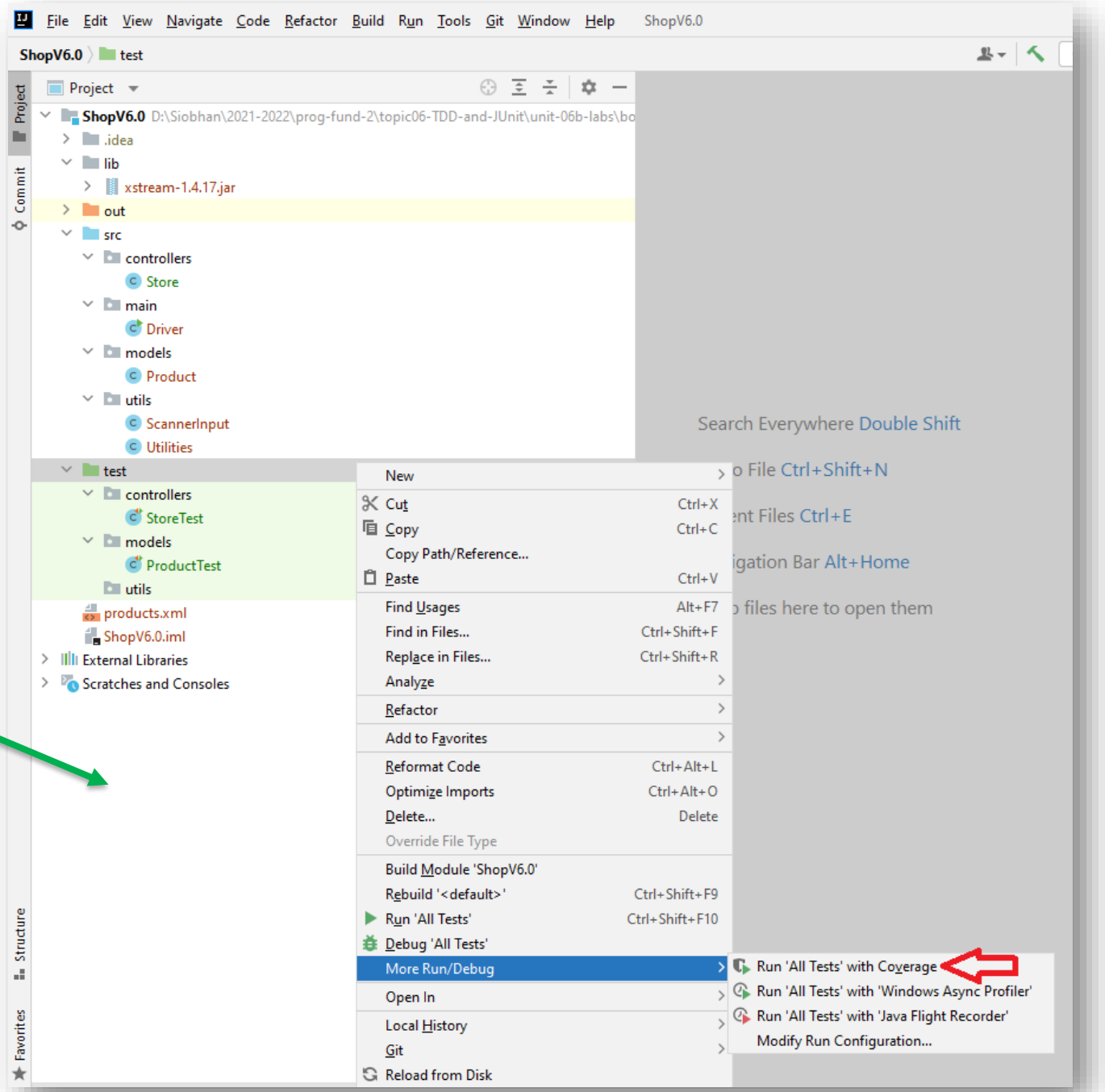
```
@Nested
class Getters {

    //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());
        }
    }
}
```

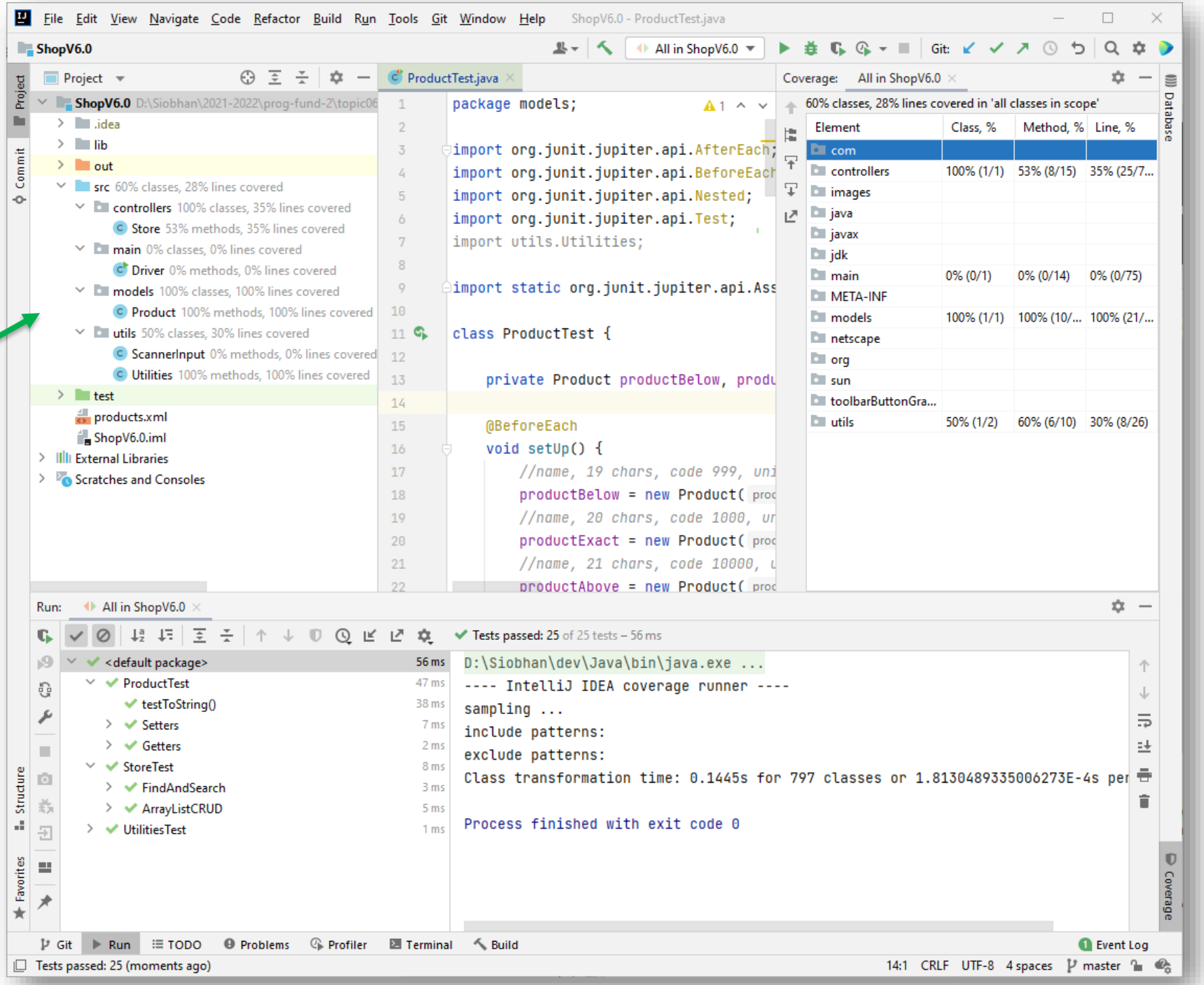
ProductTest.Java

Have we tested everything in Product.java?

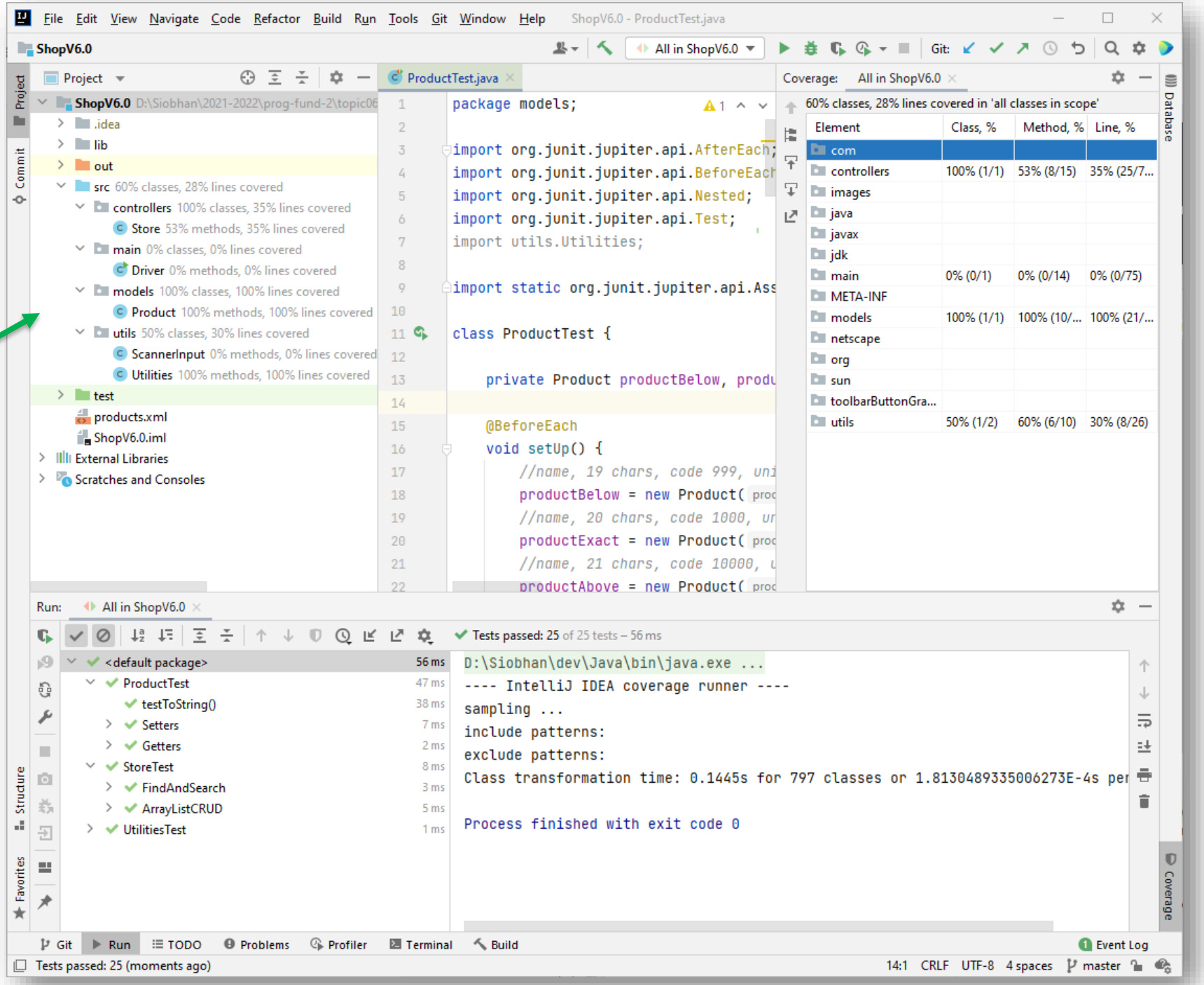
We can run a Coverage Report with our tests to see what code is tested and what isn't.

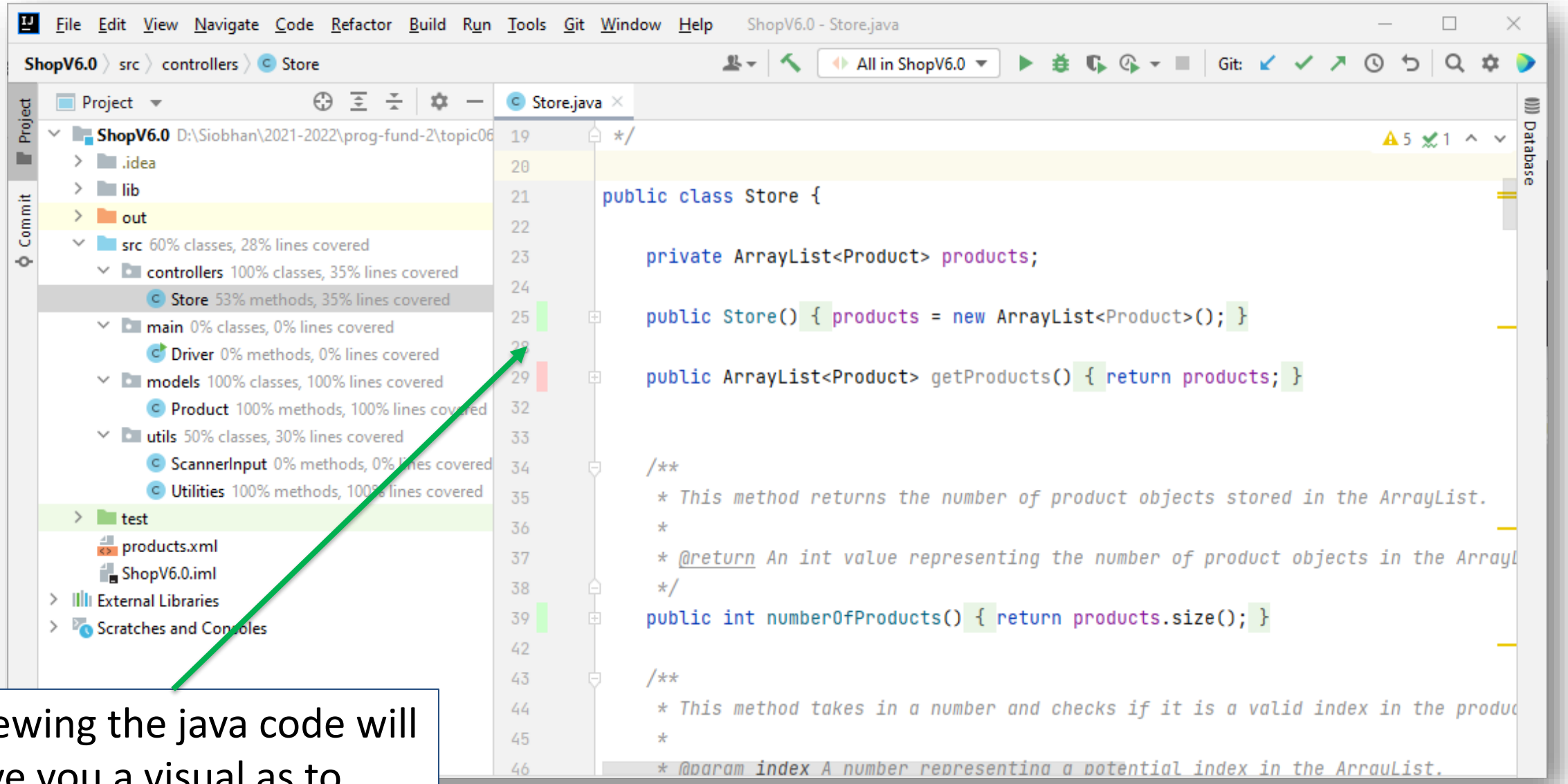


Percentages are applied to each class.



Percentages are applied to each class.





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

Any Questions?