

# 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.trimString(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;
}
```

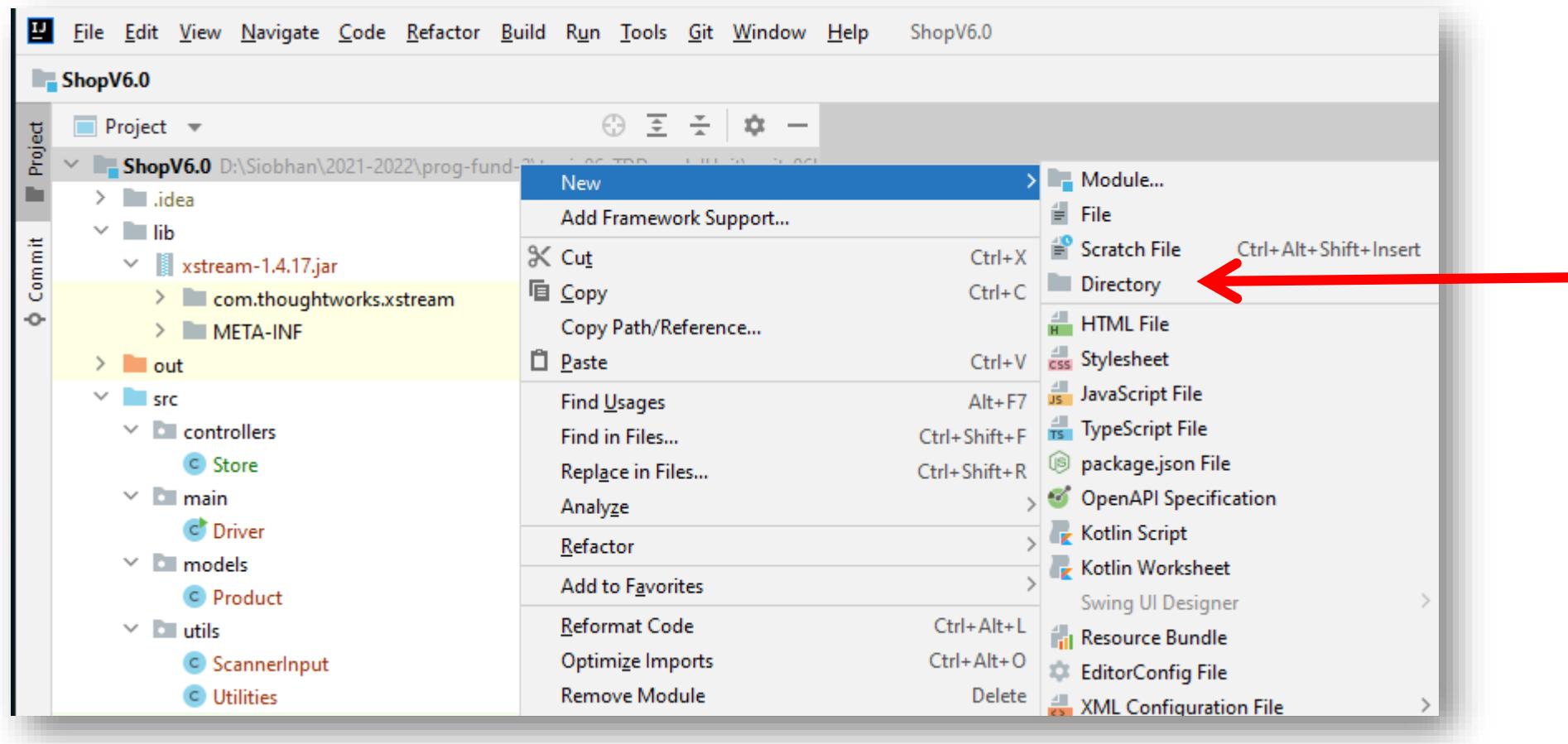
# Utilities.java

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

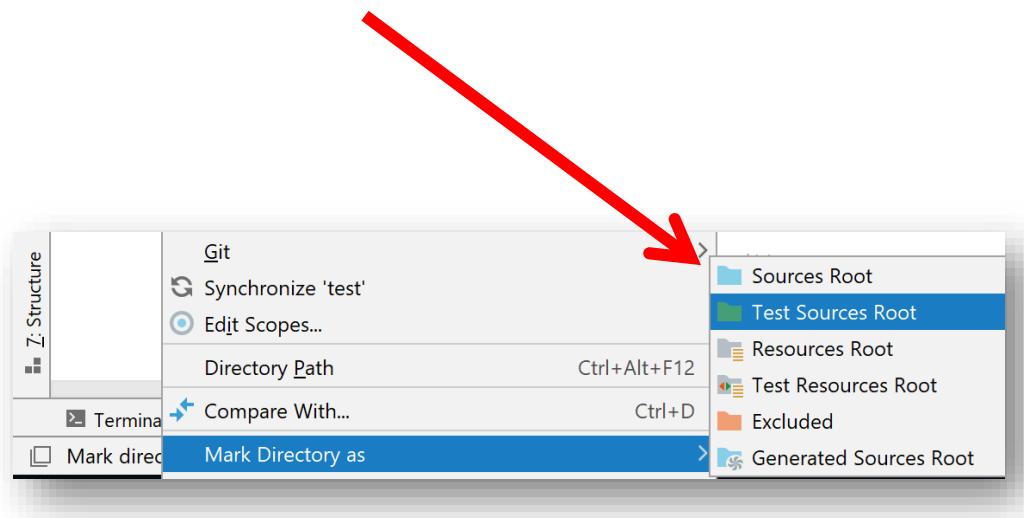
```
public class Utilities {  
  
    public static double toTwoDecimalPlaces(double number){  
        return (int) (number * 100 ) / 100.0;  
    }  
  
    public static boolean YNtoBoolean(char charToConvert){  
        return ((charToConvert == 'y') || (charToConvert == 'Y'));  
    }  
  
    public 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...

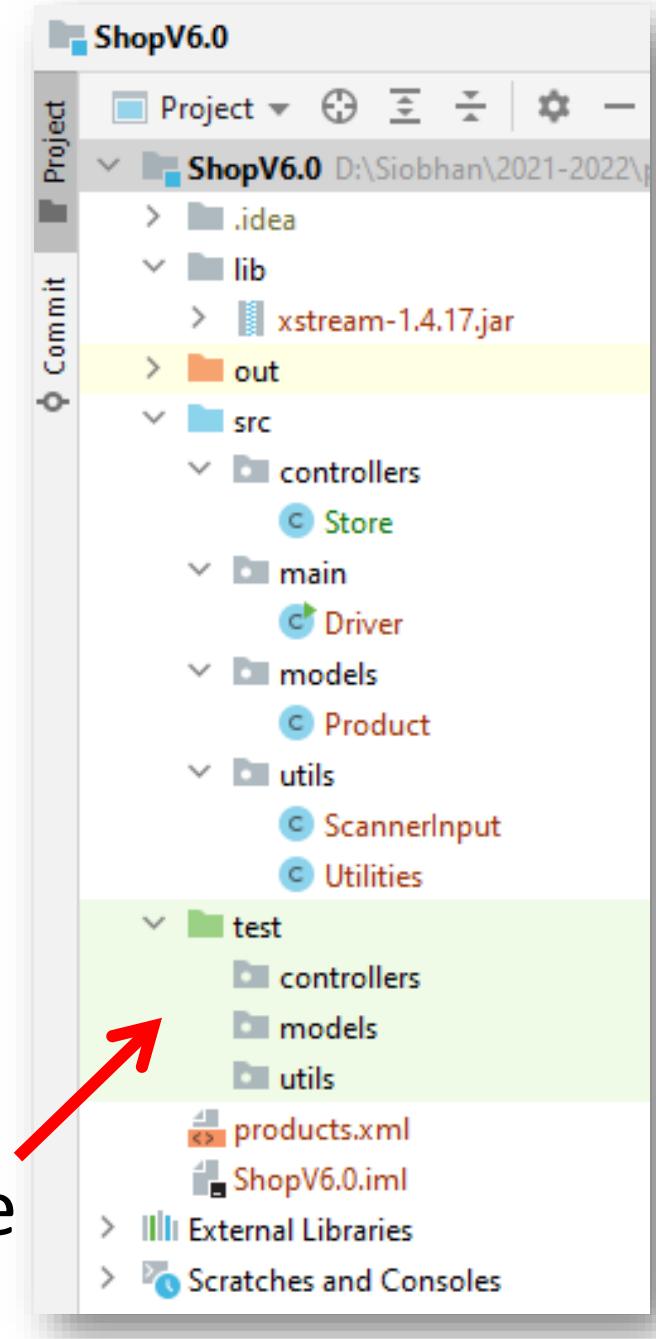
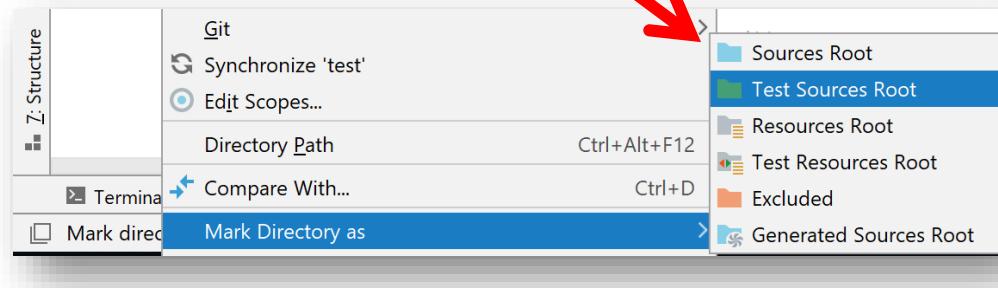
# In IntelliJ, create a new directory and call it **test**



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

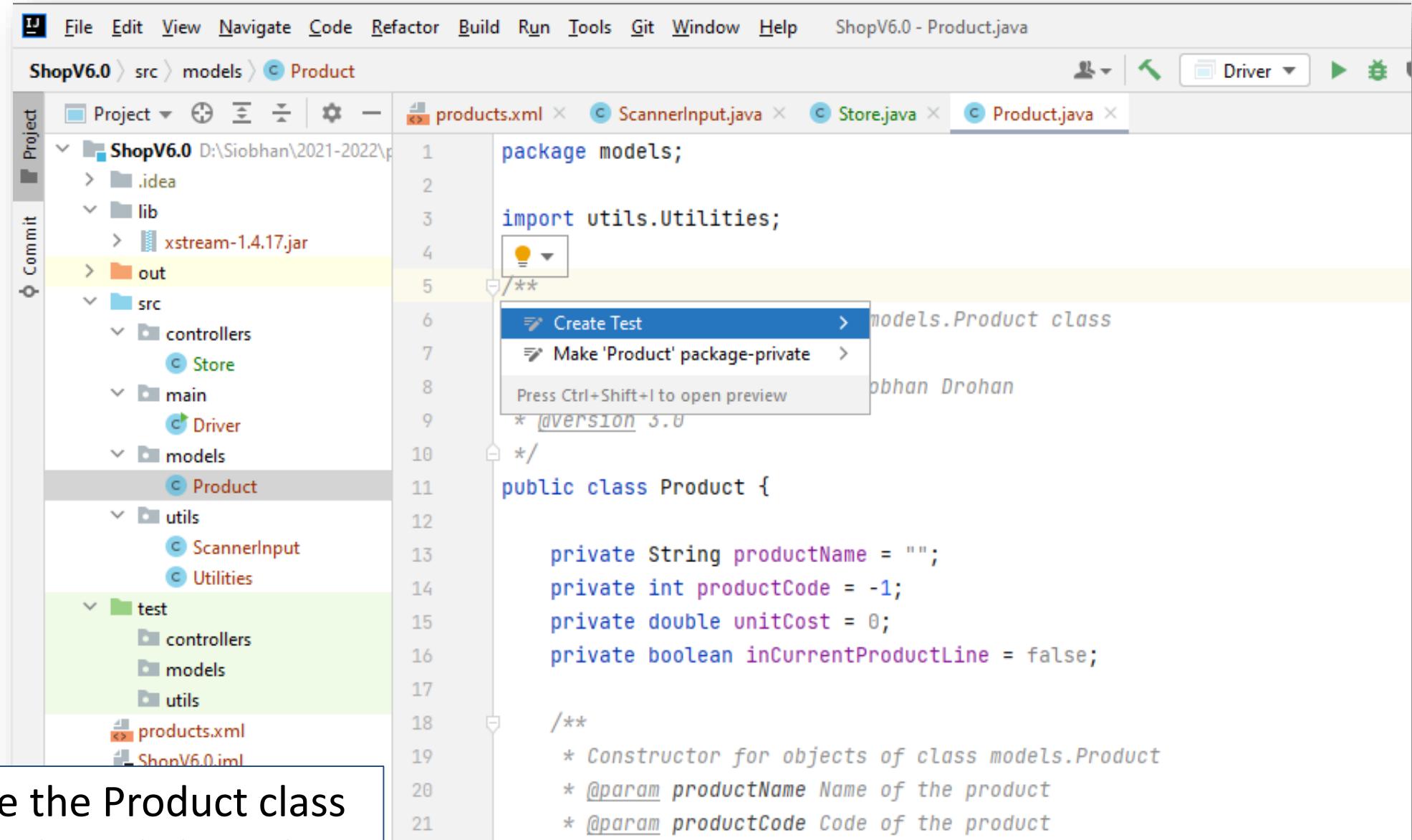


Then “mark” this new directory as a **Test Sources Root** directory



Create three packages in the test folder.

Generating a test class for Product.Java ...



The screenshot shows the IntelliJ IDEA interface with the following details:

- File Menu:** File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, Git, Window, Help.
- Title Bar:** ShopV6.0 - Product.java
- Toolbar:** Project, Commit, Create Test, Make 'Product' package-private, Press Ctrl+Shift+I to open preview, obhan Drohan.
- Project Structure:** Shows the project structure under ShopV6.0:
  - .idea
  - lib (xstream-1.4.17.jar)
  - out
  - src
    - controllers (Store)
    - main (Driver)
    - models (Product)
    - utils (ScannerInput, Utilities)
  - test
    - controllers
    - models
    - utils
- Code Editor:** The code for Product.java is displayed:

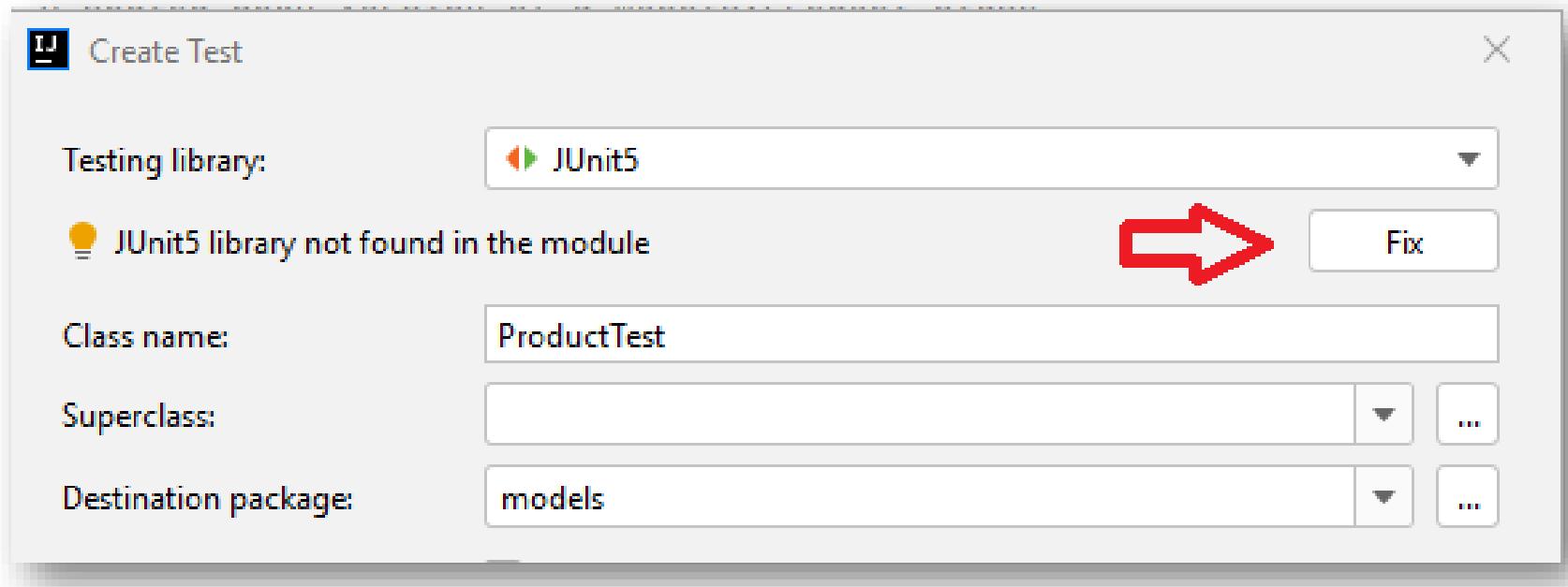
```
1 package models;
2
3 import utils.Utilities;
4
5 /**
6  * Create Test > models.Product class
7  * Make 'Product' package-private >
8  * Press Ctrl+Shift+I to open preview
9  * @version 5.0
10 */
11 public class Product {
12
13     private String productName = "";
14     private int productCode = -1;
15     private double unitCost = 0;
16     private boolean inCurrentProductLine = false;
17
18     /**
19      * Constructor for objects of class models.Product
20      * @param productName Name of the product
21      * @param productCode Code of the product
22 }
```

1

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

2

Choose JUnit5

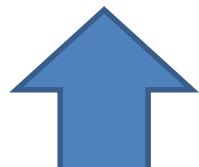
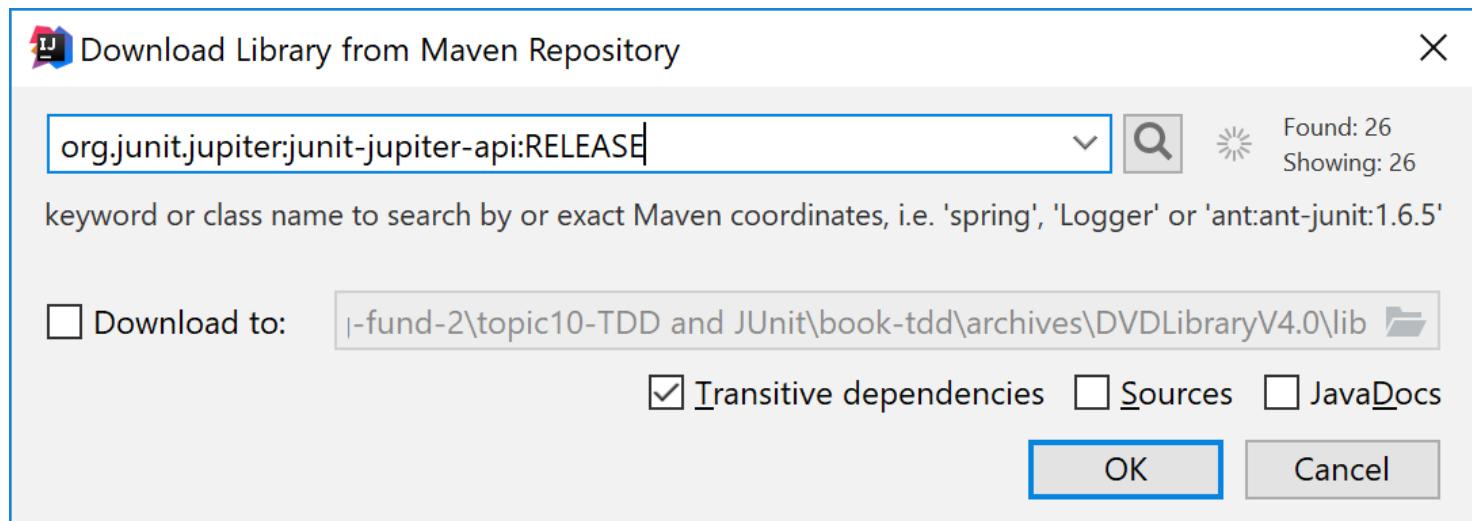


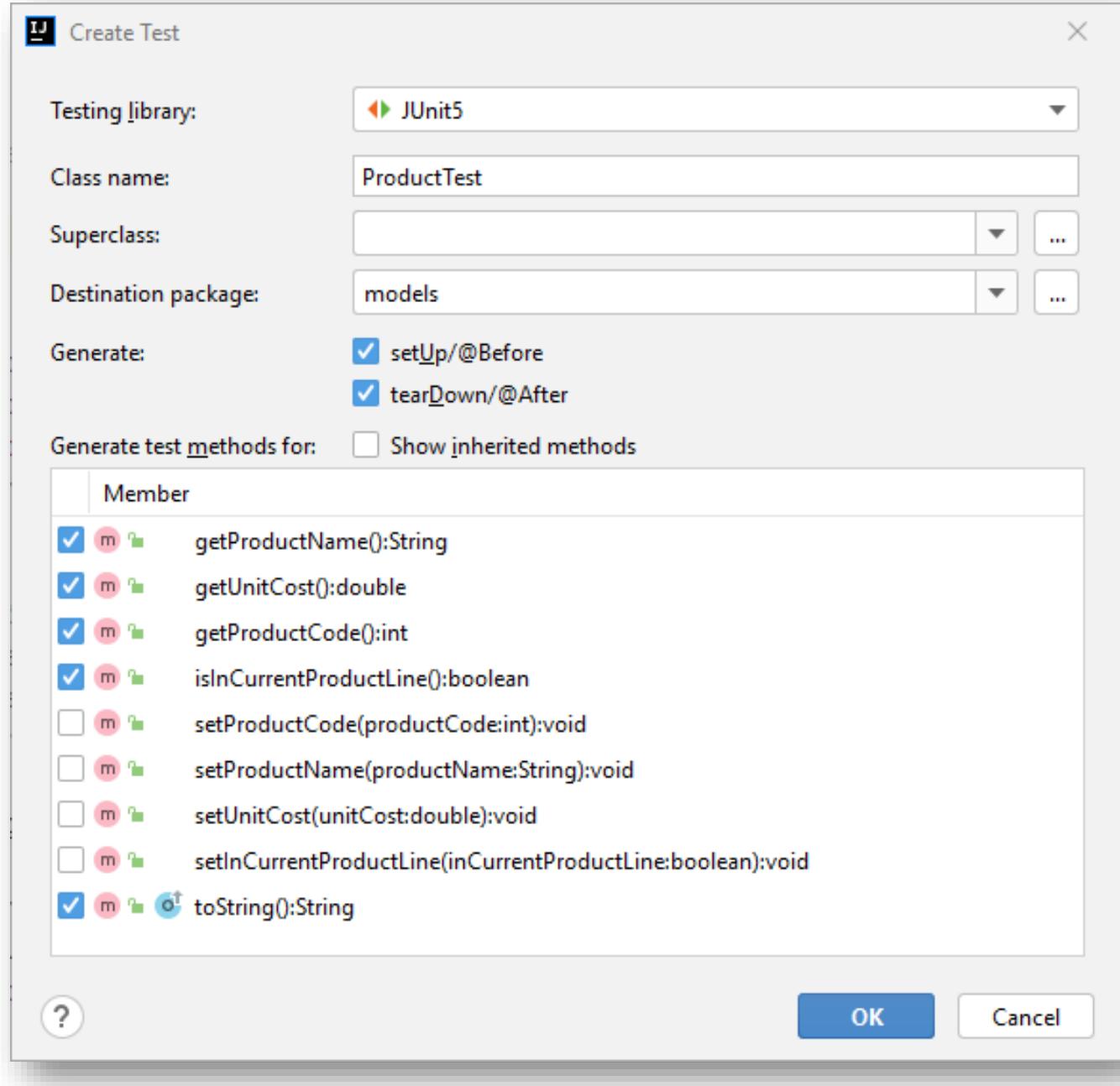
3

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.

The screenshot shows the IntelliJ IDEA interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, Git, Window, and Help. The title bar indicates the current file is ShopV6.0 - ProductTest.java. The left sidebar has tabs for Project, Commit, and Favorites, with Project selected. The Project tree shows the project structure: ShopV6.0 (containing .idea, lib, out, src, controllers, main, models, utils), test (containing controllers, models, ProductTest, utils, products.xml, ShopV6.0.iml), External Libraries, and Scratches and Consoles. The ProductTest.java file is open in the main editor area, displaying the following code:

```
package models;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

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

class ProductTest {

    @BeforeEach
    void setUp() {
    }

    @AfterEach
    void tearDown() {
    }

    @Test
    void getProductName() {
    }

    @Test
    void getUnitCost() {
    }

    @Test
    void getProductCode() {
    }

    @Test
    void isInCurrentProductLine() {
    }

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

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:

The screenshot shows an IDE interface with the following details:

- Code Editor:** On the left, a code editor displays a test method:

```
@Test  
void getProductName() {  
    assertEquals("Television 42Inches",  
    assertEquals("Television 50 Inches"  
    assertEquals("Television 60 Inches"  
    assertEquals("", productZero.getPro  
}  
}
```
- Project Structure:** In the center, the project structure for "ShopV6.0" is shown:
  - ShopV6.0
  - .idea
  - lib
  - xstream-1.4.17.jar
  - out
  - src
    - controllers
    - main
    - Driver
    - models
      - Product
    - utils
      - ScannerInput
      - Utilities
  - test
  - controllers
  - models
- Code View:** On the right, the file `ProductTest.java` is open, showing:

```
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40
```

Annotations: `@AfterEach`, `@Test`, `void tearDown()`, and `void getProductName()`.

A context menu is open over the line `void getProductName()`, listing the following options:
  - Run 'getProductName()' (highlighted with a red arrow)
  - Debug 'getProductName()'
  - Run 'getProductName()' with Coverage
  - Run 'getProductName()' with 'Windows Async Profiler'
  - Run 'getProductName()' with 'Java Flight Recorder'
  - Modify Run Configuration...

Run the test method:

The screenshot shows an IntelliJ IDEA interface with the following details:

- Project:** ShopV6.0
- File:** ProductTest.java
- Code:** A Java test class with three test methods: `getProductName()`, `getUnitCost()`, and `getProductCode()`. The `getProductName()` method fails with the following assertion error:

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

- Run:** ProductTest.getProductName
- Test Results:** 1 test failed (17 ms total). The failed test is `getProductName()`.
- Output:** The console output shows the failure message and a stack trace:

```
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
```

A large blue arrow points from the error message in the Test Results pane to a callout box containing the text "The method has failed".

The method  
has failed

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "ShopV6.0". The "src" and "test" directories are expanded, showing "controllers", "main", "models", and "utils" packages. Inside "models", "ProductTest.java" is selected.
- Code Editor:** The code for `ProductTest.java` is displayed. It includes an `@AfterEach` block for cleanup and three `@Test` blocks for testing `getProductName()`. The third assertion, which compares `productExact` to "Television 60 Inches", is highlighted with a red rectangle.
- Run Results:** The "Run" tool window shows the output of the test run. It indicates "Tests failed: 1 of 1 test - 17 ms". The "Test Results" section shows the failure of the `getProductName()` test. The error message is:

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

Below this, the stack trace is shown:

```
<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>
```
- Status Bar:** The status bar at the bottom right shows "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.

```
ProductTest.java
36     @Test
37     void getProductName() {
38         assertEquals( expected: "Television 42Inches", productBelow.getProductName());
39         assertEquals( expected: "Television 50 Inches", productExact.getProductName());
40         assertEquals( expected: "Television 60 Inches", productAbove.getProductName());
41         assertEquals( expected: "", productZero.getProductName());
42     }

```

Run: ProductTest\$Getters.getProductName

Test Results

- ProductTest
- getProductName()

Tests passed: 1 of 1 test - 14 ms

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

Process finished with exit code 0

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

# JUnit Interface

Run: ProductTest.getName <--> Tests failed: 1 of 1 test – 17 ms

9 Test Results 17 ms

ProductTest 17 ms

getProductName() 17 ms

Tests failed: 1 of 1 test – 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.getName([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

- A red arrow points from the text "Number of tests that ran and status" to the status bar at the top of the run window.
- A red arrow points from the text "Assertion Error" to the error message in the center of the run window.
- A red arrow points from the text "Class name and line of code causing the assertion failure." to the stack trace in the run window.
- A red arrow points from the text "Itemised tests and their individual result." to the tree view on the left side of the run window.

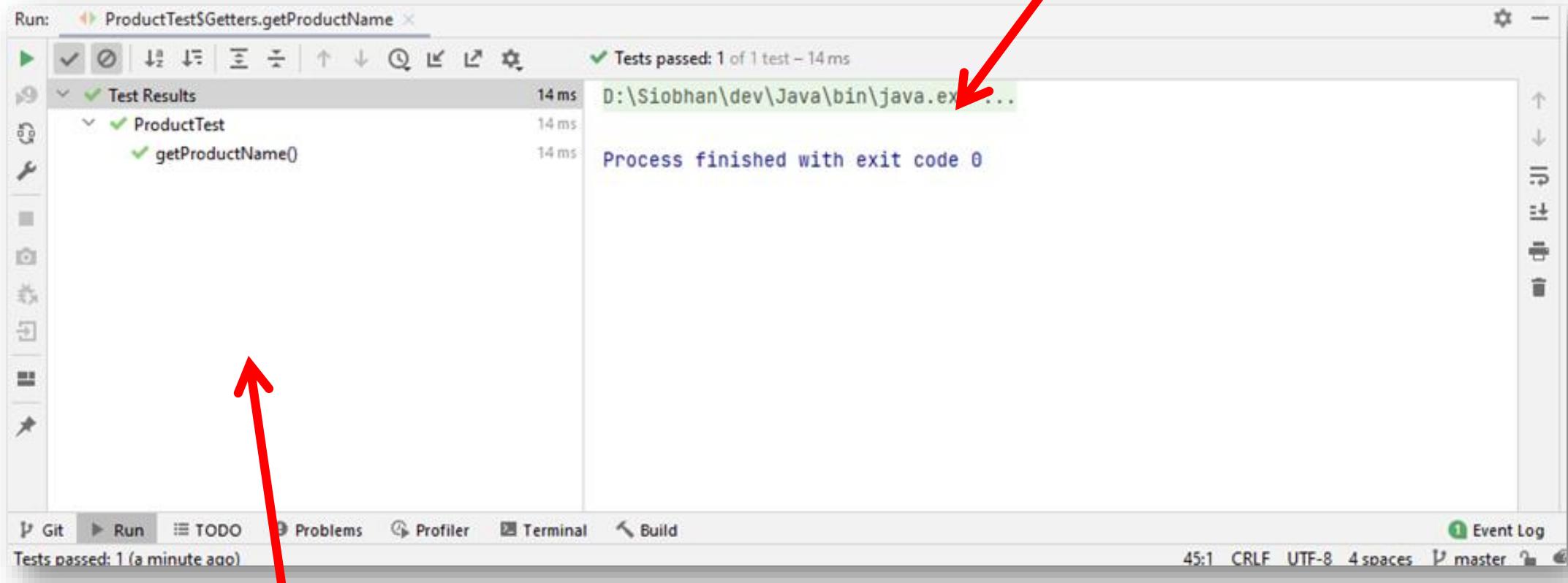
Itemised tests and their individual result.

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;

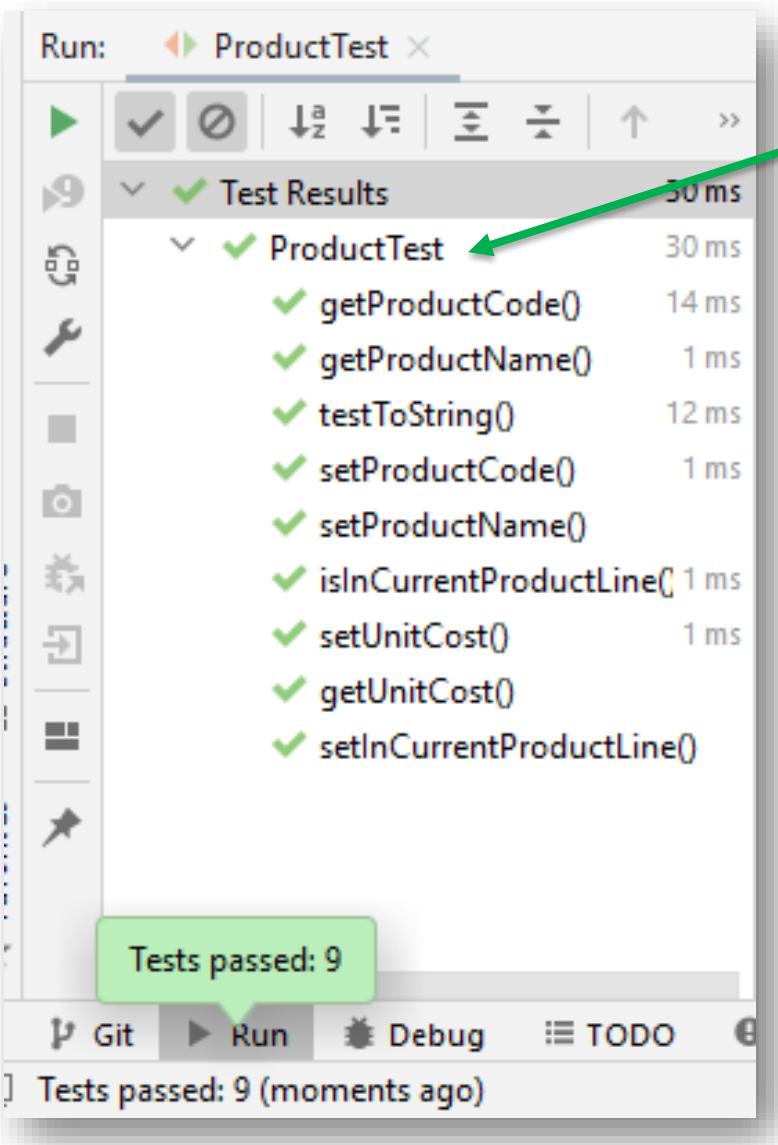
    public Product(String productName, int productCode, double unitCost, boolean inCurrentProductLine) {
        this.productName = Utilities.truncateString(productName, 20);
        setProductCode(productCode);
        this.unitCost = unitCost;
        this.inCurrentProductLine = inCurrentProductLine;
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

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

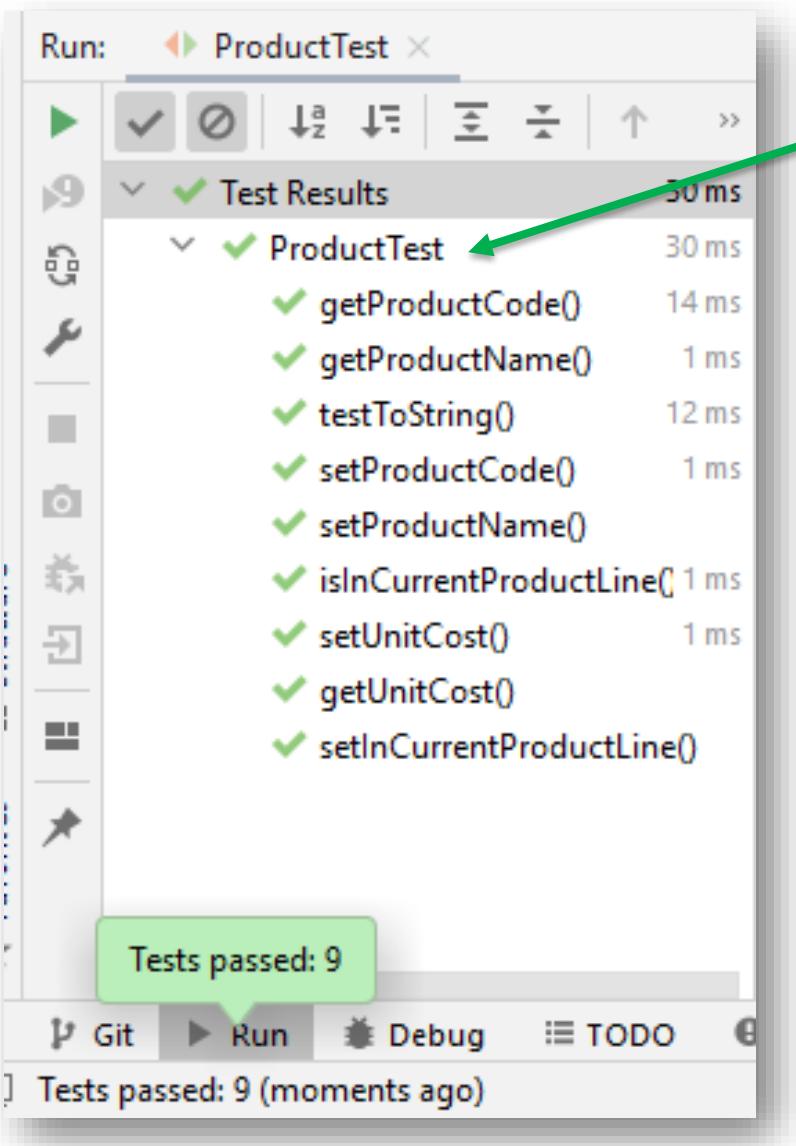
    //Code Omitted
}
```

```
@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

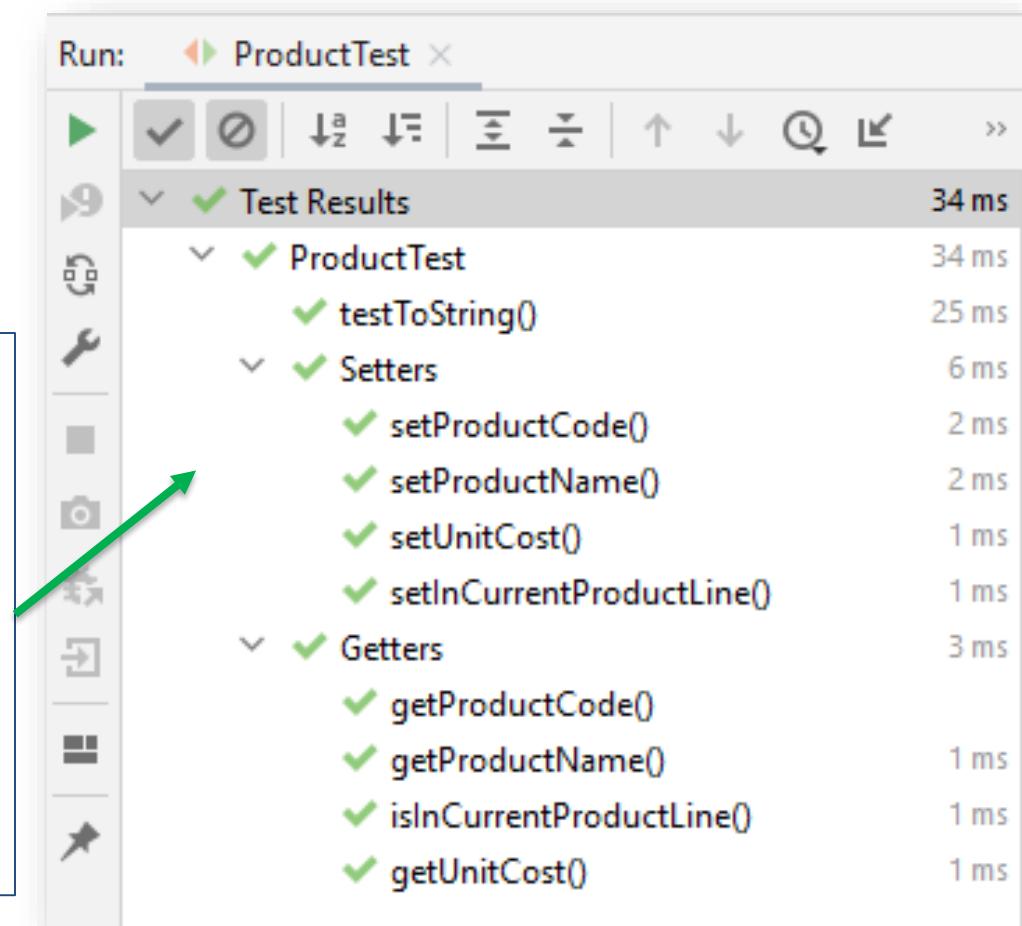


As you add and run more tests to `ProductTest.java`, they are displayed under the `ProductTest` heading.



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

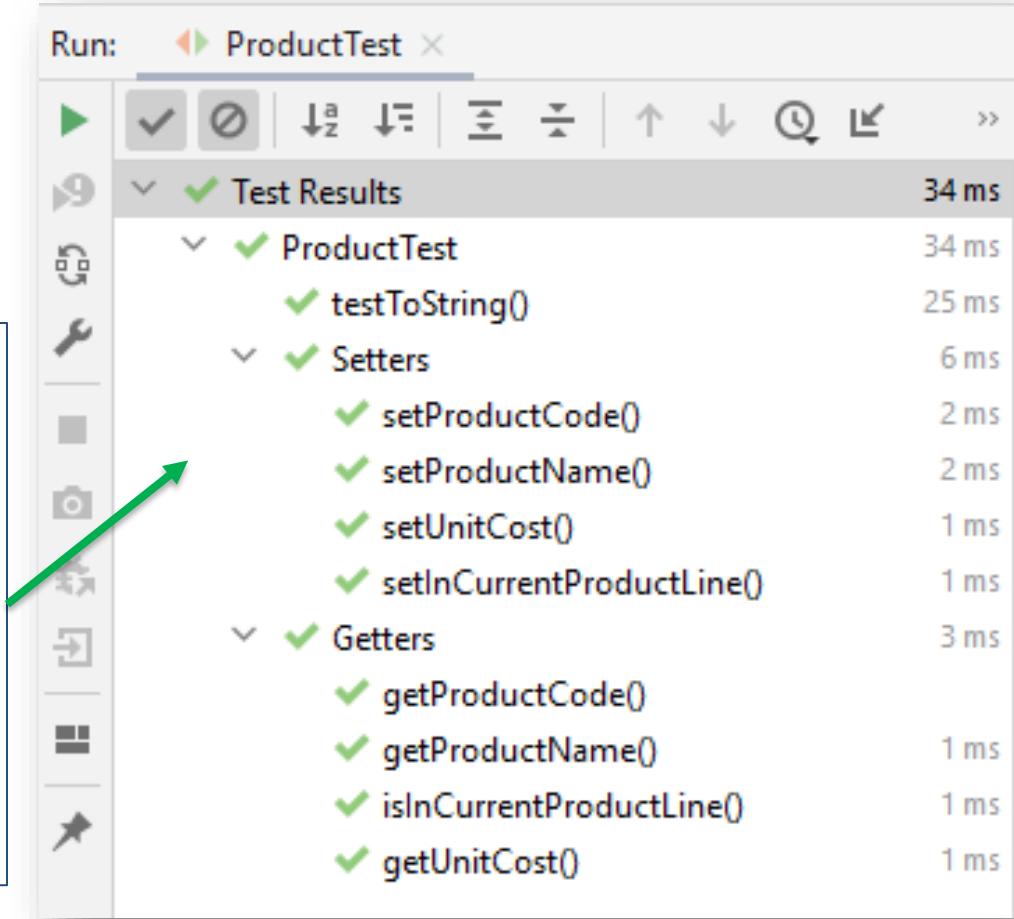
But this layout would be much easier to read...especially as more and more test methods are added...



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.

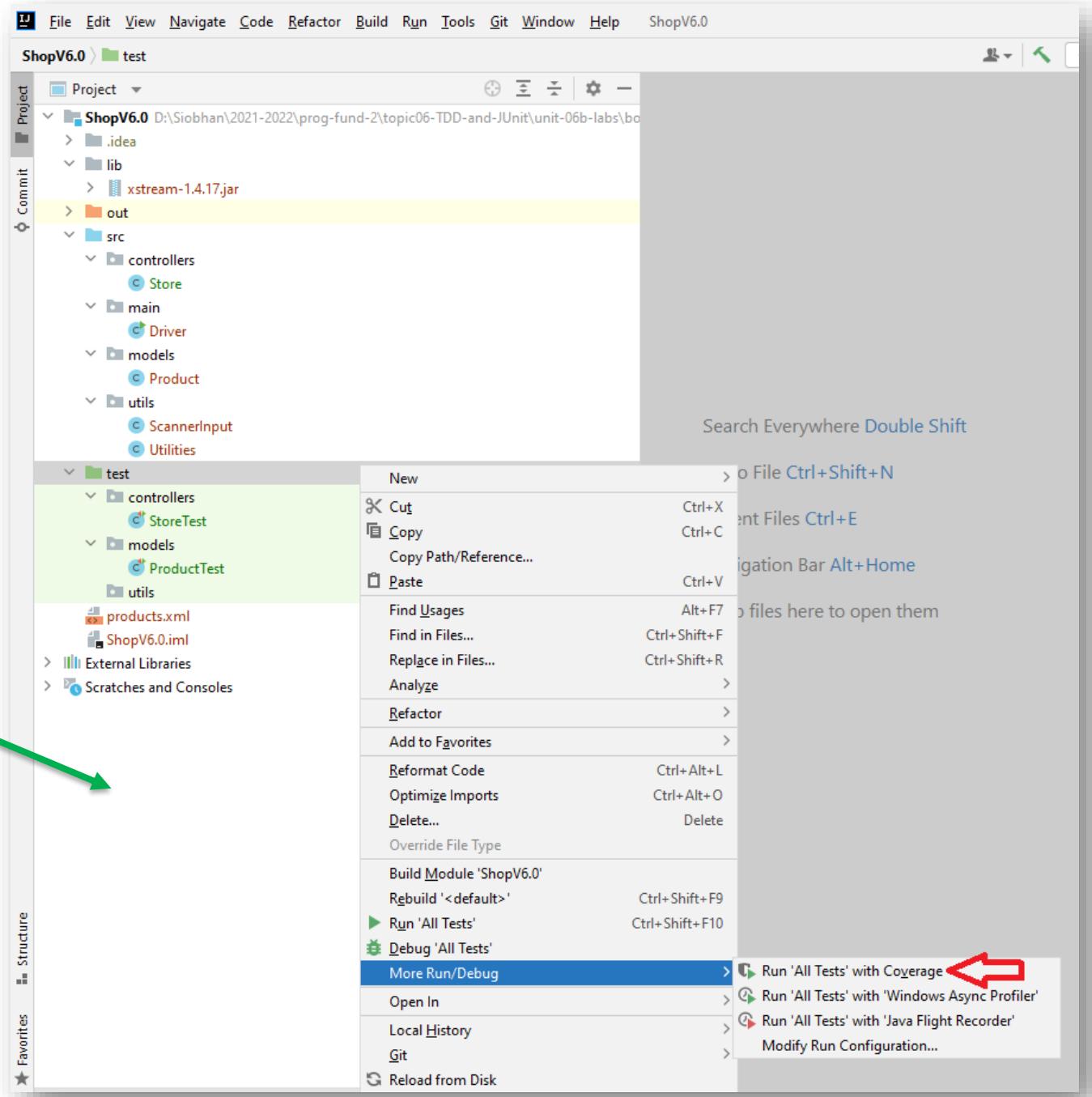
```
@Nested  
class Getters {  
  
    //getter methods go in here  
  
}
```

```
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.



File Edit View Navigate Code Refactor Build Run Tools Git Window Help ShopV6.0 - ProductTest.java

ShopV6.0

Project Commit

ProductTest.java

Coverage: All in ShopV6.0

ShopV6.0 D:\Siobhan\2021-2022\prog-fund-2\topic06

src 60% classes, 28% lines covered

- controllers 100% classes, 35% lines covered
  - Store 53% methods, 35% lines covered
  - Driver 0% methods, 0% lines covered
- main 0% classes, 0% lines covered
  - Driver 0% methods, 0% lines covered
- models 100% classes, 100% lines covered
  - Product 100% methods, 100% lines covered
- utils 50% classes, 30% lines covered
  - ScannerInput 0% methods, 0% lines covered
  - Utilities 100% methods, 100% lines covered

test

- products.xml
- ShopV6.0.iml

External Libraries

Scratches and Consoles

Element Class, % Method, % Line, %

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

Run: All in ShopV6.0

Structure Favorites

Tests passed: 25 of 25 tests – 56 ms

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 class  
Process finished with exit code 0

Git Run TODO Problems Profiler Terminal Build

Tests passed: 25 (moments ago)

14:1 CRLF UTF-8 4 spaces master Event Log

Percentages are applied to each class.

File Edit View Navigate Code Refactor Build Run Tools Git Window Help ShopV6.0 - ProductTest.java

ShopV6.0

Project Commit

ProductTest.java

Coverage: All in ShopV6.0

ShopV6.0 D:\Siobhan\2021-2022\prog-fund-2\topic06

src 60% classes, 28% lines covered

- controllers 100% classes, 35% lines covered
  - Store 53% methods, 35% lines covered
  - Driver 0% methods, 0% lines covered
- main 0% classes, 0% lines covered
  - Driver 0% methods, 0% lines covered
- models 100% classes, 100% lines covered
  - Product 100% methods, 100% lines covered
- utils 50% classes, 30% lines covered
  - ScannerInput 0% methods, 0% lines covered
  - Utilities 100% methods, 100% lines covered

test

- products.xml
- ShopV6.0.iml

External Libraries

Scratches and Consoles

Element Class, % Method, % Line, %

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

Run: All in ShopV6.0

Structure Favorites

Tests passed: 25 of 25 tests – 56 ms

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 class  
Process finished with exit code 0

Git Run TODO Problems Profiler Terminal Build

Tests passed: 25 (moments ago)

14:1 CRLF UTF-8 4 spaces master Event Log

Percentages are applied to each class.

The screenshot shows the IntelliJ IDEA interface with the following details:

- File Bar:** File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, Git, Window, Help, ShopV6.0 - Store.java
- Title Bar:** ShopV6.0 > src > controllers > Store
- Toolbars:** Standard toolbar with icons for file operations, Git status, and search.
- Project Tool Window:** Shows the project structure:
  - ShopV6.0 (D:\Siobhan\2021-2022\prog-fund-2\topic06)
  - .idea
  - lib
  - out
  - src (60% classes, 28% lines covered)
    - controllers (100% classes, 35% lines covered)
      - Store (53% methods, 35% lines covered)
    - main (0% classes, 0% lines covered)
      - Driver (0% methods, 0% lines covered)
    - models (100% classes, 100% lines covered)
      - Product (100% methods, 100% lines covered)
    - utils (50% classes, 30% lines covered)
      - ScannerInput (0% methods, 0% lines covered)
      - Utilities (100% methods, 100% lines covered)
  - test
  - products.xml
  - ShopV6.0.iml
  - External Libraries
  - Scratches and Consoles
- Code Editor:** The Store.java file is open, showing Java code for a Store class. A green arrow points from the explanatory text below to the coverage information in the Project tool window.
- Status Bar:** Shows code coverage statistics: 53% methods, 35% lines covered.

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

Any  
Questions?

