Test Driven Development

More JUnit Tests for the Shop app

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/

Topic List

- Product and ProductTest.java
- JUnit Testing of Store.java
- Testing Driver.java

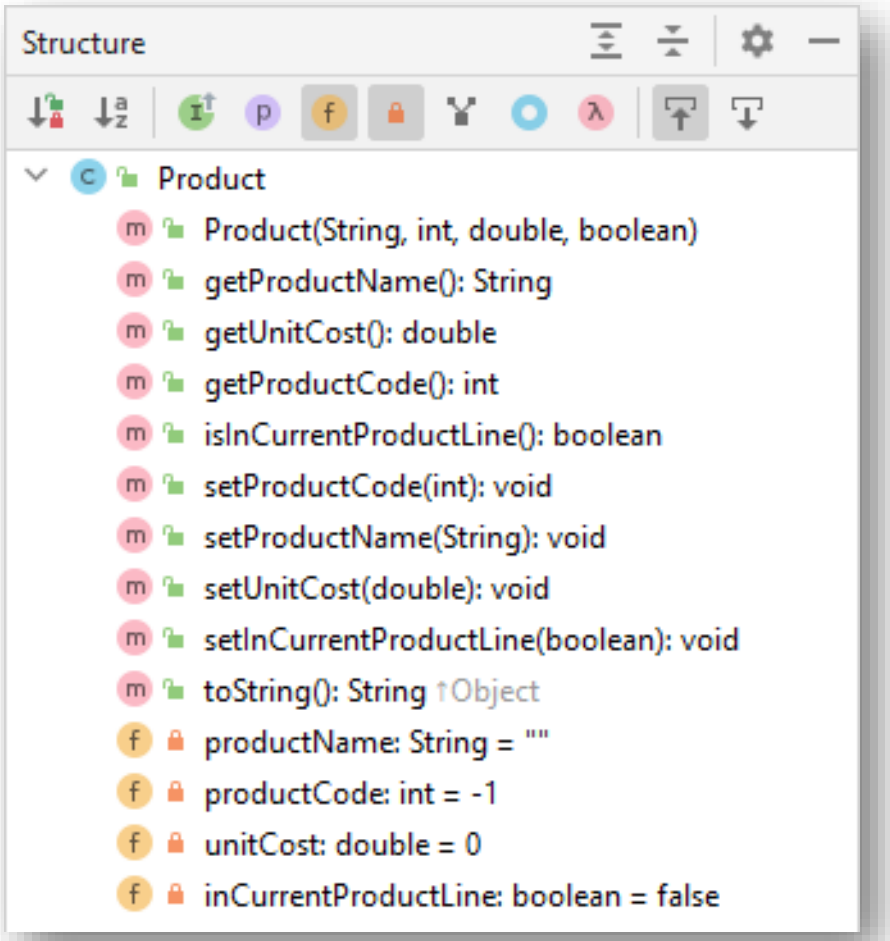
Product.java

```
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 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);
    }

}
```

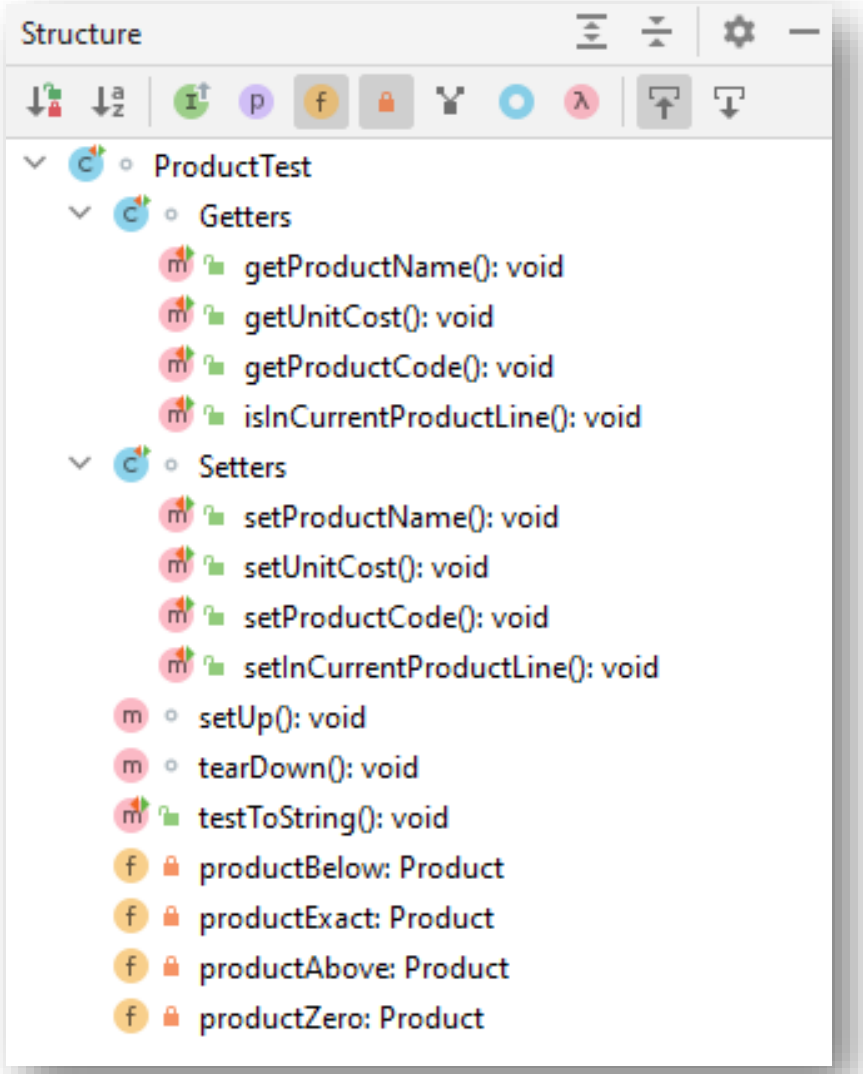
```
    public void setProductCode(int productCode) {
        if (Utilities.validRange(productCode, 1000, 9999)) {
            this.productCode = productCode;
        }
    }

    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;
    }
```

Product.java



ProductTest.java



ProductTest.java

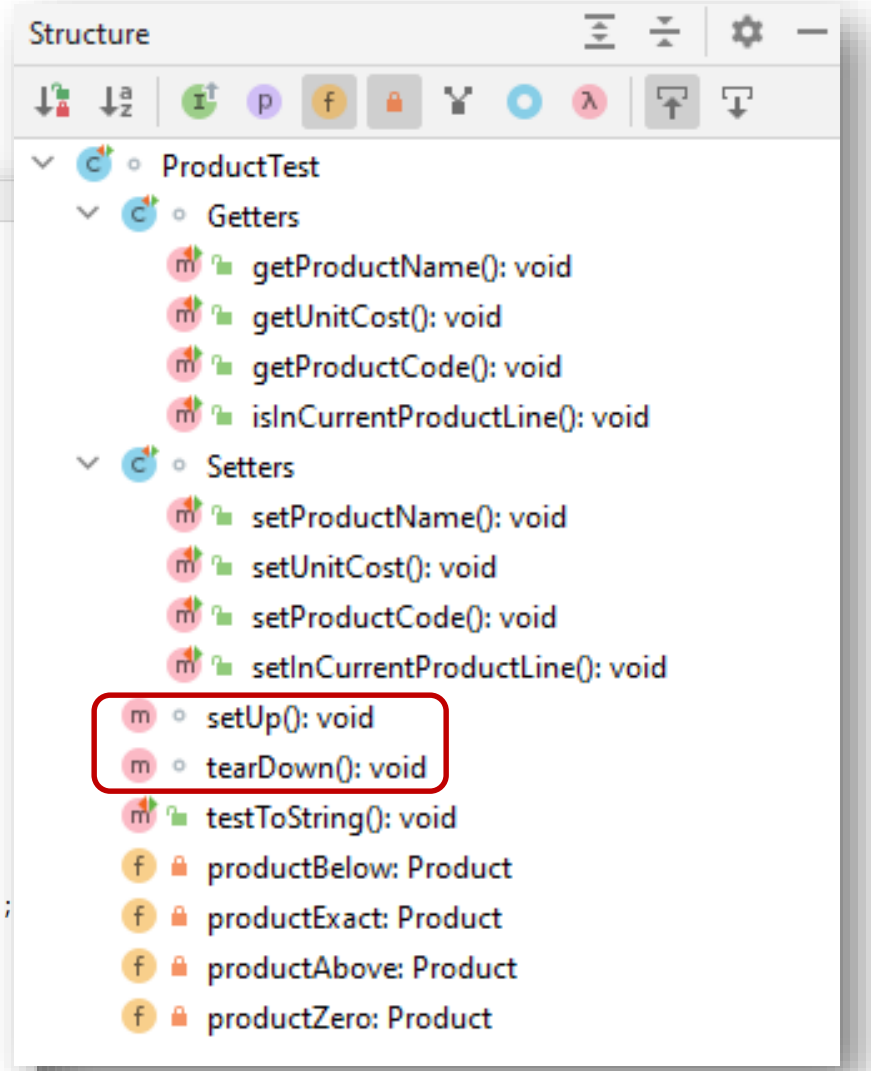
ProductTest.java | Product.java

```
1   package models;
2
3   import org.junit.jupiter.api.AfterEach;
4   import org.junit.jupiter.api.BeforeEach;
5   import org.junit.jupiter.api.Nested;
6   import org.junit.jupiter.api.Test;
7
8   import static org.junit.jupiter.api.Assertions.*;
9
10  class ProductTest {
11
12      private Product productBelow, productExact, productAbove, productZero;
13
14      @BeforeEach
15      void setUp() {...}
25
26      @AfterEach
27      void tearDown() { productBelow = productExact = productAbove = productZero = null; }
30
31      @Nested
32      class Getters {...}
66
67      @Nested
68      class Setters {...}
119
120     @Test
121     void testToString() {...}
128 }
```

Structure

- ProductTest
  - Getters
    - getProductName(): void
    - getUnitCost(): void
    - getProductCode(): void
    - isInCurrentProductLine(): void
  - Setters
    - setProductName(): void
    - setUnitCost(): void
    - setProductCode(): void
    - setInCurrentProductLine(): void
  - setUp(): void
  - tearDown(): void
  - testToString(): void
  - productBelow: Product
  - productExact: Product
  - productAbove: Product
  - productZero: Product


ProductTest.java
1 package models;
2
3 import org.junit.jupiter.api.AfterEach;
4 import org.junit.jupiter.api.BeforeEach;
5 import org.junit.jupiter.api.Nested;
6 import org.junit.jupiter.api.Test;
7
8 import static org.junit.jupiter.api.Assertions.*;
9
10 class ProductTest {
11
12 private Product productBelow, productExact, productAbove, productZero;
13
14 @BeforeEach
15 void setUp() {
16 //name, 19 chars, code 999, unitCost 1, inCurrentProductLine true.
17 productBelow = new Product(productName: "Television 42Inches", productCode: 999, unitCost: 1, inCurrentProductLine: true);
18 //name, 20 chars, code 1000, unitCost 999, inCurrentProductLine true.
19 productExact = new Product(productName: "Television 50 Inches", productCode: 1000, unitCost: 999, inCurrentProductLine: true);
20 //name, 21 chars, code 10000, unitCost 1000, inCurrentProductLine false.
21 productAbove = new Product(productName: "Television 60 Inches.", productCode: 10000, unitCost: 1000, inCurrentProductLine: false);
22 //name, 0 chars, code 9999, unitCost 0, inCurrentProductLine false.
23 productZero = new Product(productName: "", productCode: 9999, unitCost: 0, inCurrentProductLine: false);
24 }
25
26 @AfterEach
27 void tearDown() {
28 productBelow = productExact = productAbove = productZero = null;
29 }


Structure
ProductTest
Getters
getProductName(): void
getUnitCost(): void
getProductCode(): void
isInCurrentProductLine(): void
Setters
setProductName(): void
setUnitCost(): void
setProductCode(): void
setInCurrentProductLine(): void
setUp(): void
tearDown(): void
testToString(): void
productBelow: Product
productExact: Product
productAbove: Product
productZero: Product

ProductTest.java

```
30
31      @Nested
32      class Getters {
33
34          @Test
35          void getProductName() {
36              assertEquals( expected: "Television 42Inches", productBelow.getProductName());
37              assertEquals( expected: "Television 50 Inches", productExact.getProductName());
38              assertEquals( expected: "Television 60 Inches", productAbove.getProductName());
39              assertEquals( expected: "", productZero.getProductName());
40          }
41
42          @Test
43          void getUnitCost() {
44              assertEquals( expected: 1, productBelow.getUnitCost());
45              assertEquals( expected: 999, productExact.getUnitCost());
46              assertEquals( expected: 1000, productAbove.getUnitCost());
47              assertEquals( expected: 0, productZero.getUnitCost());
48          }
49
50          @Test
51          void getProductCode() {
52              assertEquals( expected: -1, productBelow.getProductCode());
53              assertEquals( expected: 1000, productExact.getProductCode());
54              assertEquals( expected: -1, productAbove.getProductCode());
55              assertEquals( expected: 9999, productZero.getProductCode());
56          }
57
58          @Test
59          void isInCurrentProductLine() {
60              assertTrue(productBelow.isInCurrentProductLine());
61              assertTrue(productExact.isInCurrentProductLine());
62              assertFalse(productAbove.isInCurrentProductLine());
63              assertFalse(productZero.isInCurrentProductLine());
64          }
65      }
```

Structure

- ProductTest
  - Getters
    - getProductName(): void
    - getUnitCost(): void
    - getProductCode(): void
    - isInCurrentProductLine(): void
  - Setters
    - setProductName(): void
    - setUnitCost(): void
    - setProductCode(): void
    - setInCurrentProductLine(): void
  - setUp(): void
  - tearDown(): void
  - testToString(): void
  - productBelow: Product
  - productExact: Product
  - productAbove: Product
  - productZero: Product

ProductTest.java

```
67      @Nested
68      class Setters {
69          @Test
70          void setProductName() {
71              assertEquals( expected: "Television 42Inches", productBelow.getProductName());
72
73              productBelow.setProductName("iPhone 13 Charcoal.");   //19 chars - update performed
74              assertEquals( expected: "iPhone 13 Charcoal.", productBelow.getProductName());
75
76              productBelow.setProductName("iPhone 12 - Charcoal");   //20 chars - update performed
77              assertEquals( expected: "iPhone 12 - Charcoal", productBelow.getProductName());
78
79              productBelow.setProductName("iPhone 11: - Charcoal");   //21 chars - update ignored
80              assertEquals( expected: "iPhone 12 - Charcoal", productBelow.getProductName());
81          }
82
83          @Test
84          void setUnitCost() {
85              assertEquals( expected: 999, productExact.getUnitCost());
86              productExact.setUnitCost(99.99);  //no validation performed
87              assertEquals( expected: 99.99, productExact.getUnitCost());
88          }
89
90          @Test
91          void setProductCode() {...}
106
107         @Test
108         void setInCurrentProductLine() {...}
117
118     }
```

Structure

- ProductTest
  - Getters
    - getProductName(): void
    - getUnitCost(): void
    - getProductCode(): void
    - isInCurrentProductLine(): void
  - Setters
    - setProductName(): void
    - setUnitCost(): void
    - setProductCode(): void
    - setInCurrentProductLine(): void
  - setUp(): void
  - tearDown(): void
  - testToString(): void
  - productBelow: Product
  - productExact: Product
  - productAbove: Product
  - productZero: Product

ProductTest.java

```java

class ProductTest {

    private Product productBelow, productExact, productAbove, productZero;

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

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

    @Nested
    class Getters {...}

    @Nested
    class Setters {...}

    @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"));
    }
}
```

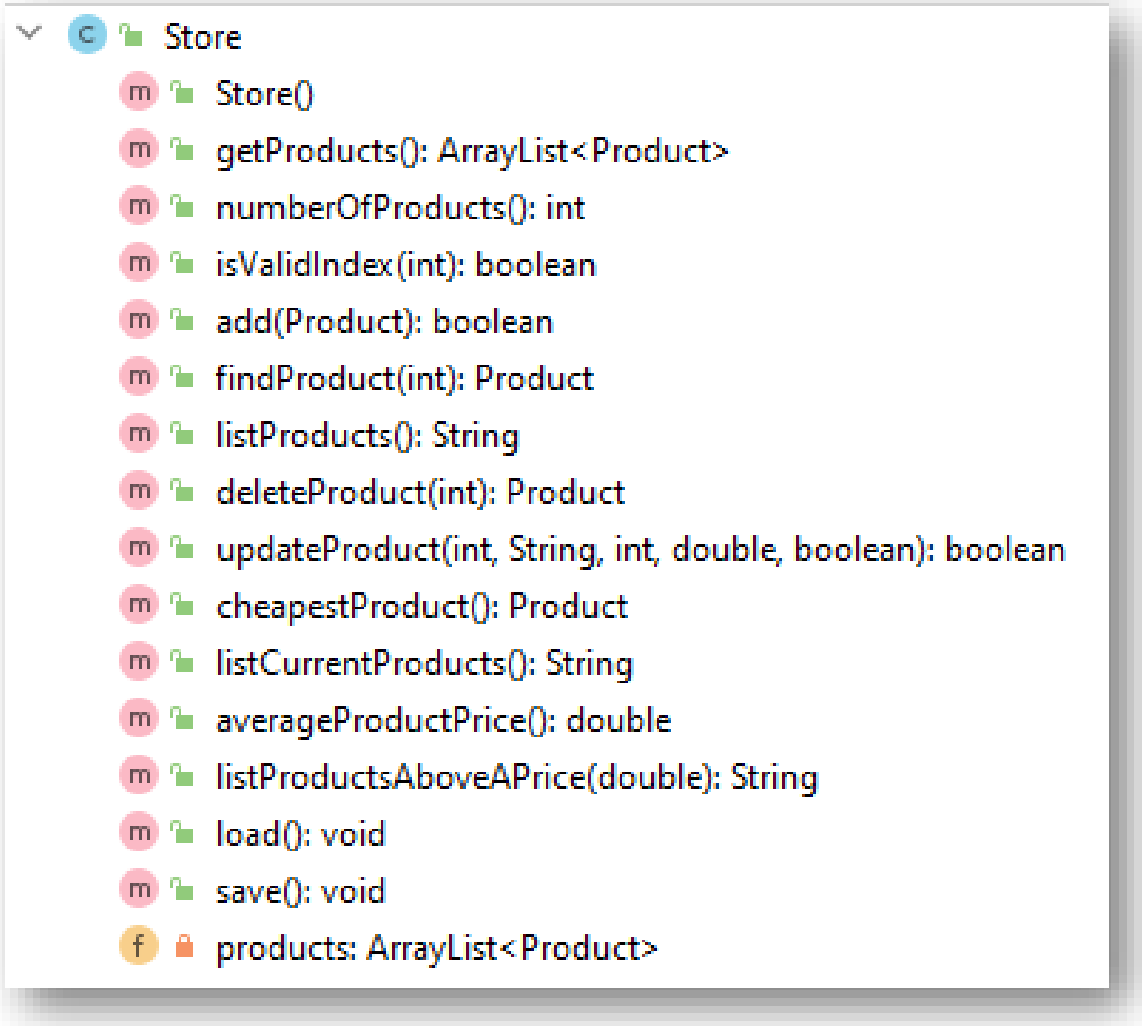
Structure

- ProductTest
  - Getters
    - getProductName(): void
    - getUnitCost(): void
    - getProductCode(): void
    - isInCurrentProductLine(): void
  - Setters
    - setProductName(): void
    - setUnitCost(): void
    - setProductCode(): void
    - setInCurrentProductLine(): void
  - setUp(): void
  - tearDown(): void
  - testToString(): void
  - productBelow: Product
  - productExact: Product
  - productAbove: Product
  - productZero: Product

Topic List

- Product and ProductTest.java
- JUnit Testing of Store.java
- Testing Driver.java

Store.java

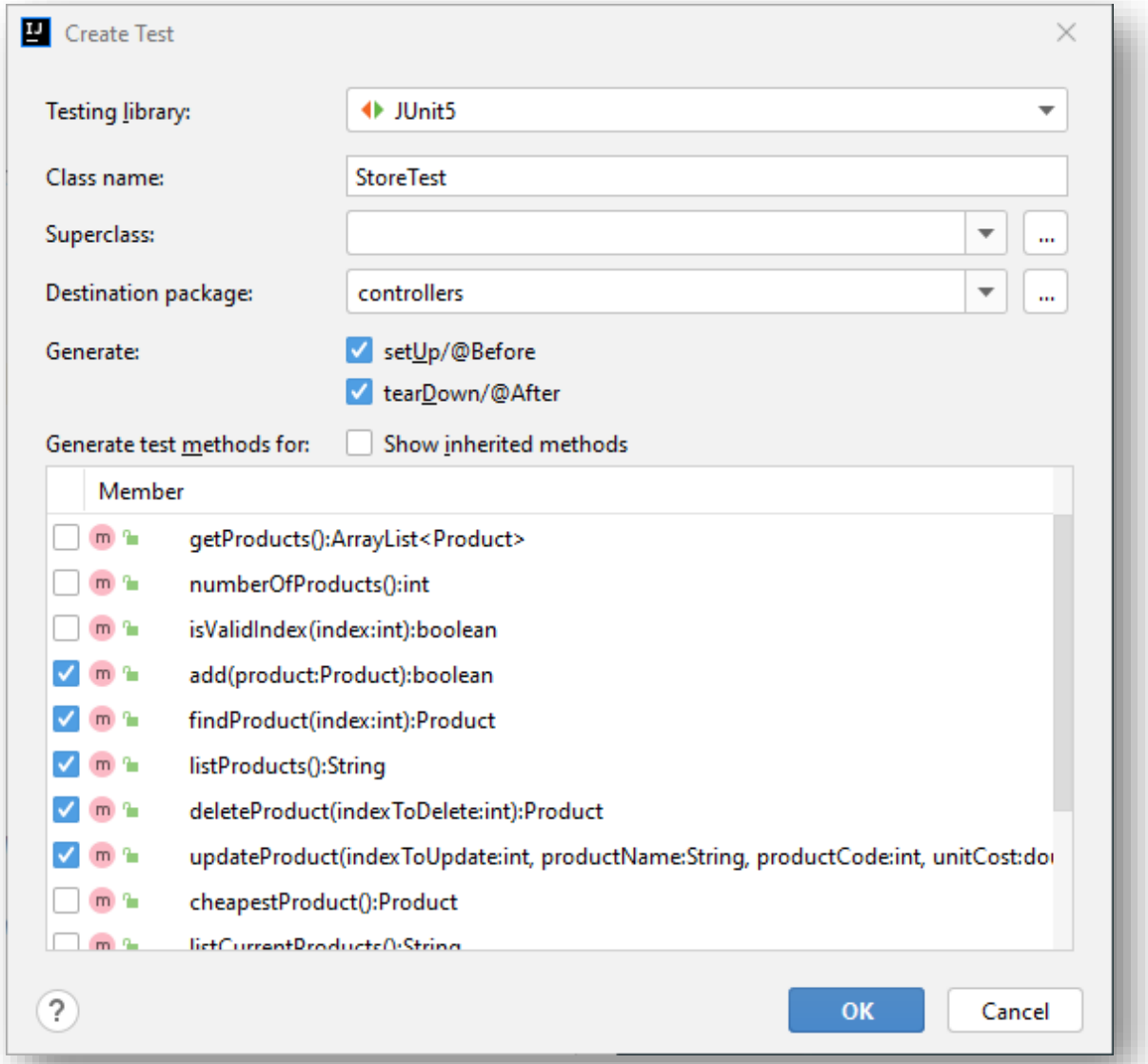


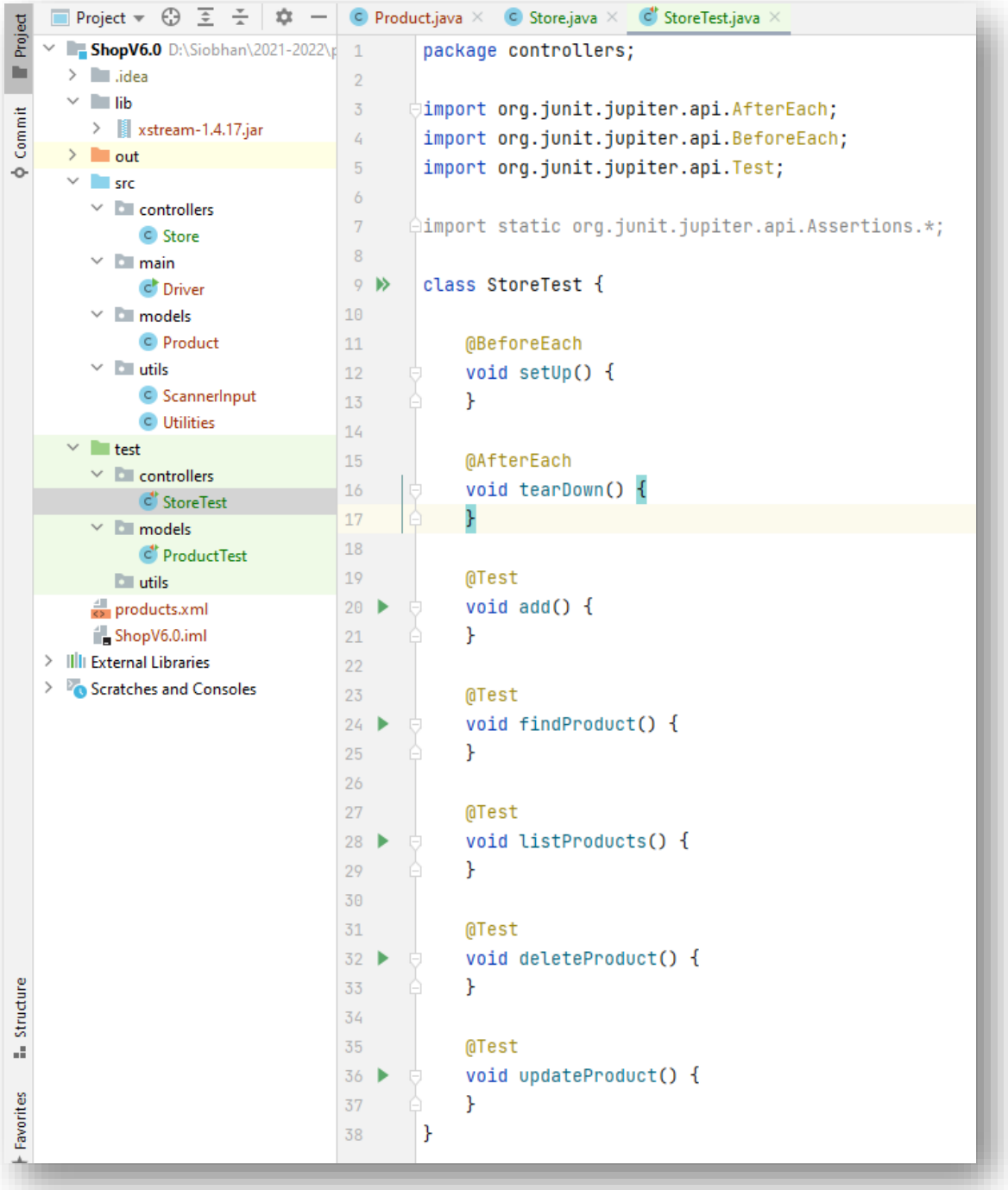
We will look at writing tests for a few of these methods.

Open Store.java and call “Create Test”

Call the test class, **StoreTest**

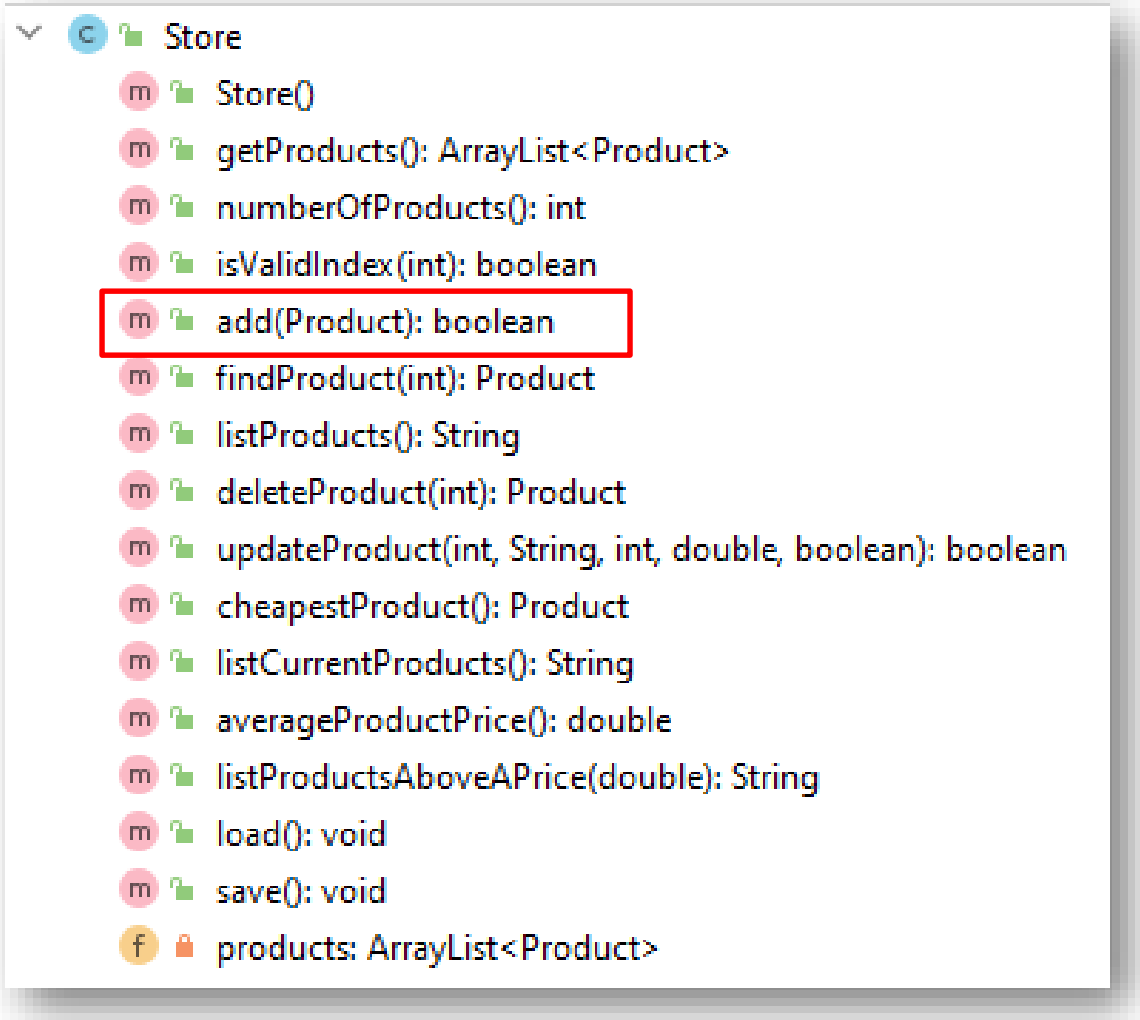
Generate the default **setUp()** and **tearDown()** methods and also generate test methods for selected member methods.





Generated StoreTest.java

Store.java – testing add(Product)



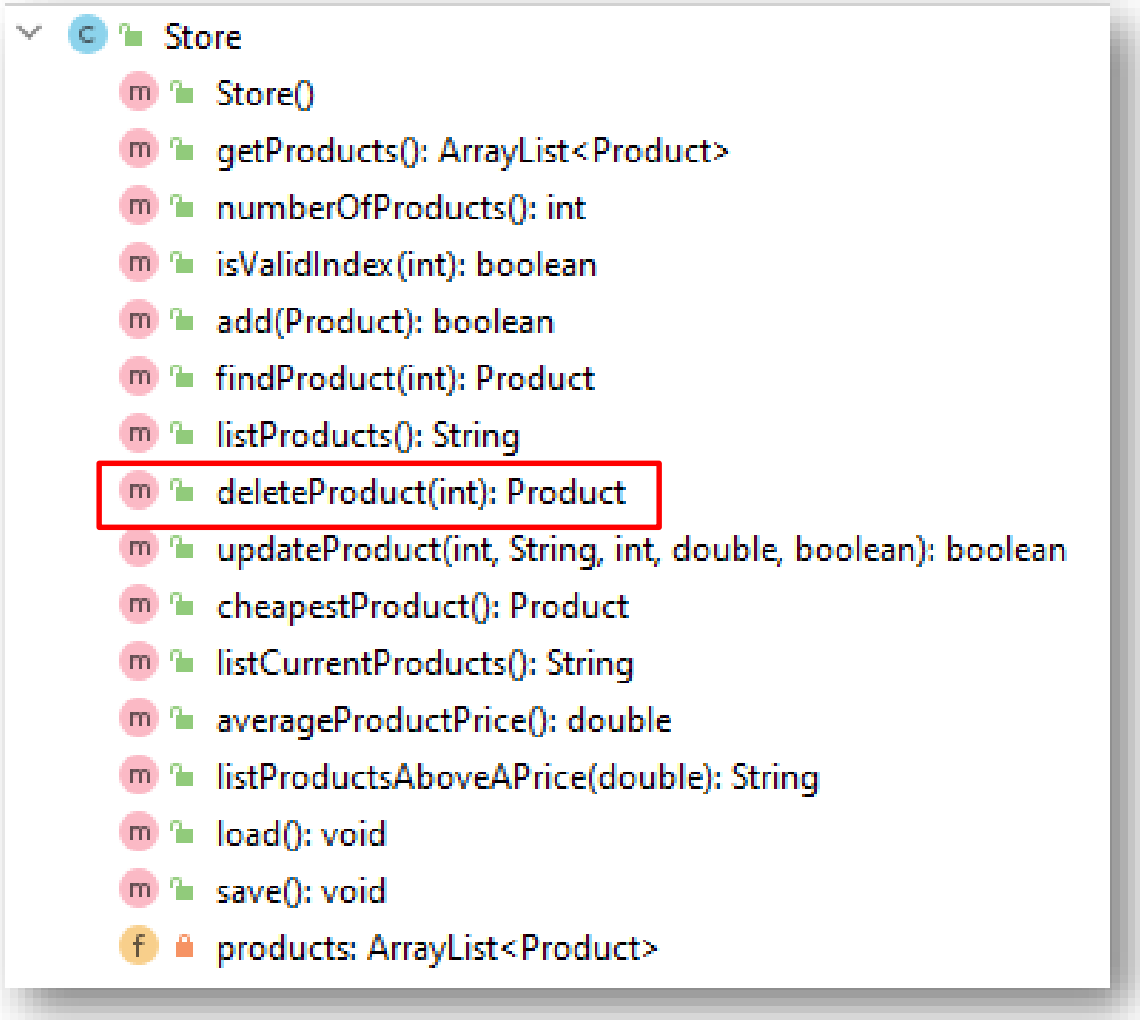

```
public boolean add (Product product){
    return products.add (product);
}
```

```
class StoreTest {

    private Product productBelow, productExact, productAbove, productZero;
    private Store storeWithProducts = new Store();
    private Store storeEmpty = new Store();

    @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);

        storeWithProducts.add(productBelow);
        storeWithProducts.add(productExact);
        storeWithProducts.add(productAbove);
    }

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

testing add(Product)

```
@Test
void addingToAnArrayListThatHasProductsIsSuccessful() {
    assertEquals(3, storeWithProducts.numberOfProducts());
    assertTrue(storeWithProducts.add(productZero));
    assertEquals(4, storeWithProducts.numberOfProducts());
    assertEquals(productZero, storeWithProducts.findProduct(3
}

@Test
void addingToAnArrayListThatHasNoProductsIsSuccessful() {
    assertEquals(0, storeEmpty.numberOfProducts());
    assertTrue(storeEmpty.add(productZero));
    assertEquals(1, storeEmpty.numberOfProducts());
    assertEquals(productZero, storeEmpty.findProduct(0));
}
```

Store.java – testing deleteProduct(Product)



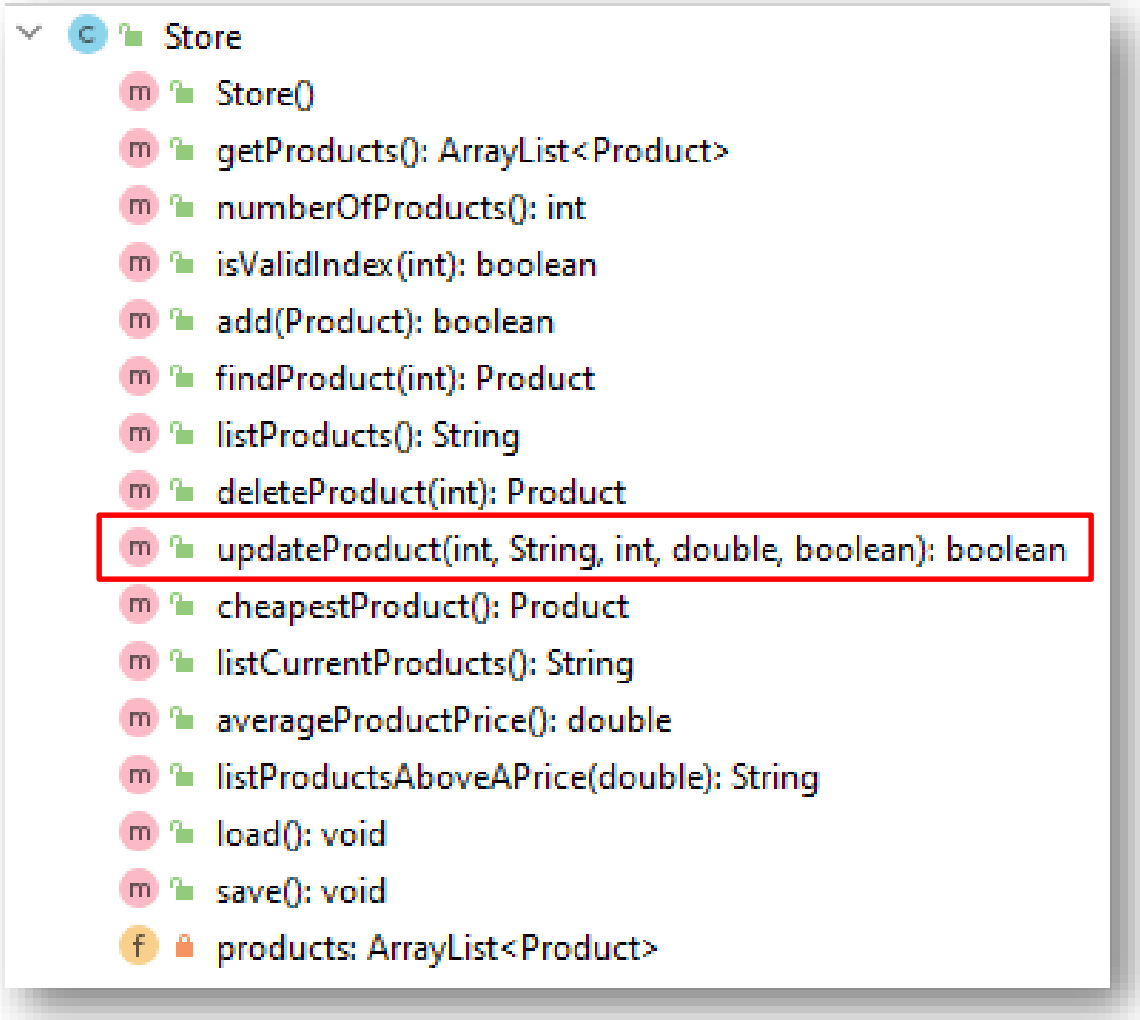

```
public Product deleteProduct(int indexToDelete) {
    if (isValidIndex(indexToDelete)) {
        return products.remove(indexToDelete);
    }
    return null;
}
```

```java
class StoreTest {

    private Product productBelow, productExact, productAbove, productZero;
    private Store storeWithProducts = new Store();
    private Store storeEmpty = new Store();

    @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);

        storeWithProducts.add(productBelow);
        storeWithProducts.add(productExact);
        storeWithProducts.add(productAbove);
    }

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

testing deleteProduct(Product)

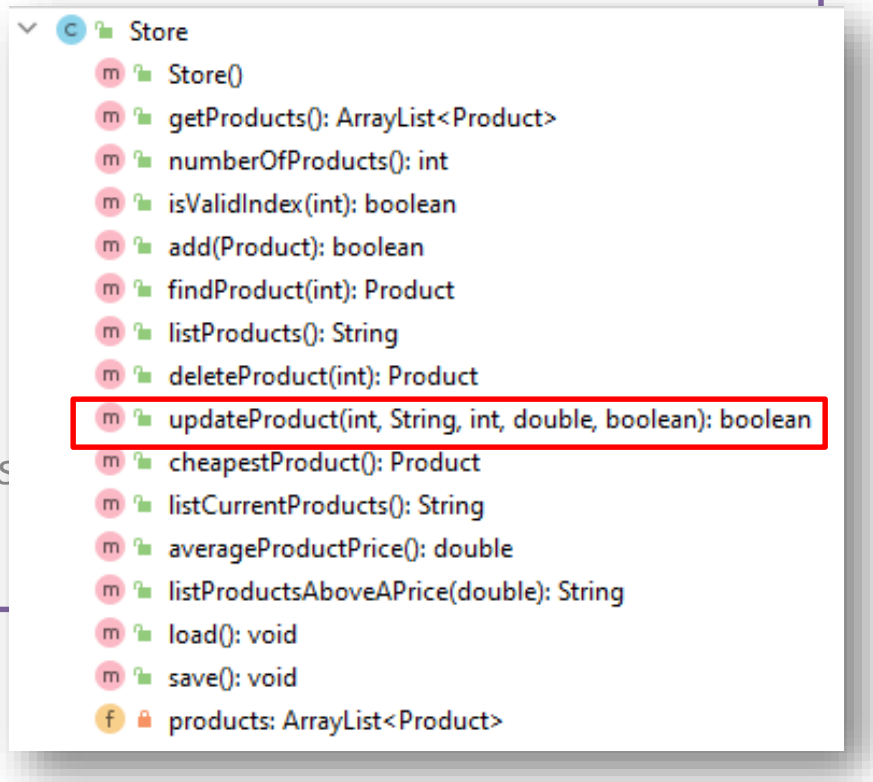
```java
@Test
void deletingAProductThatDoesNotExistReturnsNull(){
    assertNull(storeEmpty.deleteProduct(0));
    assertNull(storeWithProducts.deleteProduct(-1));
    assertNull(storeWithProducts.deleteProduct(3));
}

@Test
void deletingAProductThatExistsDeletesAndReturnsDeletedObject(
    assertEquals(3, storeWithProducts.numberOfProducts());
    assertEquals(productAbove, storeWithProducts.deleteProduct(2)
    assertEquals(2, storeWithProducts.numberOfProducts());
}
```

Store.java – testing updateProduct



Store.java – testing updateProduct

```
public boolean updateProduct(int indexToUpdate, String productName, int productCode, double unitCost, boolean inCurrentProductL
    //find the product object by the index number
    Product foundProduct = findProduct(indexToUpdate);

    //if the product exists, use the details passed in the updateDetails parameter to
    //update the found product in the ArrayList.
    if (foundProduct != null) {
        foundProduct.setProductName(productName);
        foundProduct.setProductCode(productCode);
        foundProduct.setUnitCost(unitCost);
        foundProduct.setInCurrentProductLine(inCurrentProductLine);
        return true;
    }

    //if the product was not found, return false, indicating that the update was not succes
    return false;
}
```

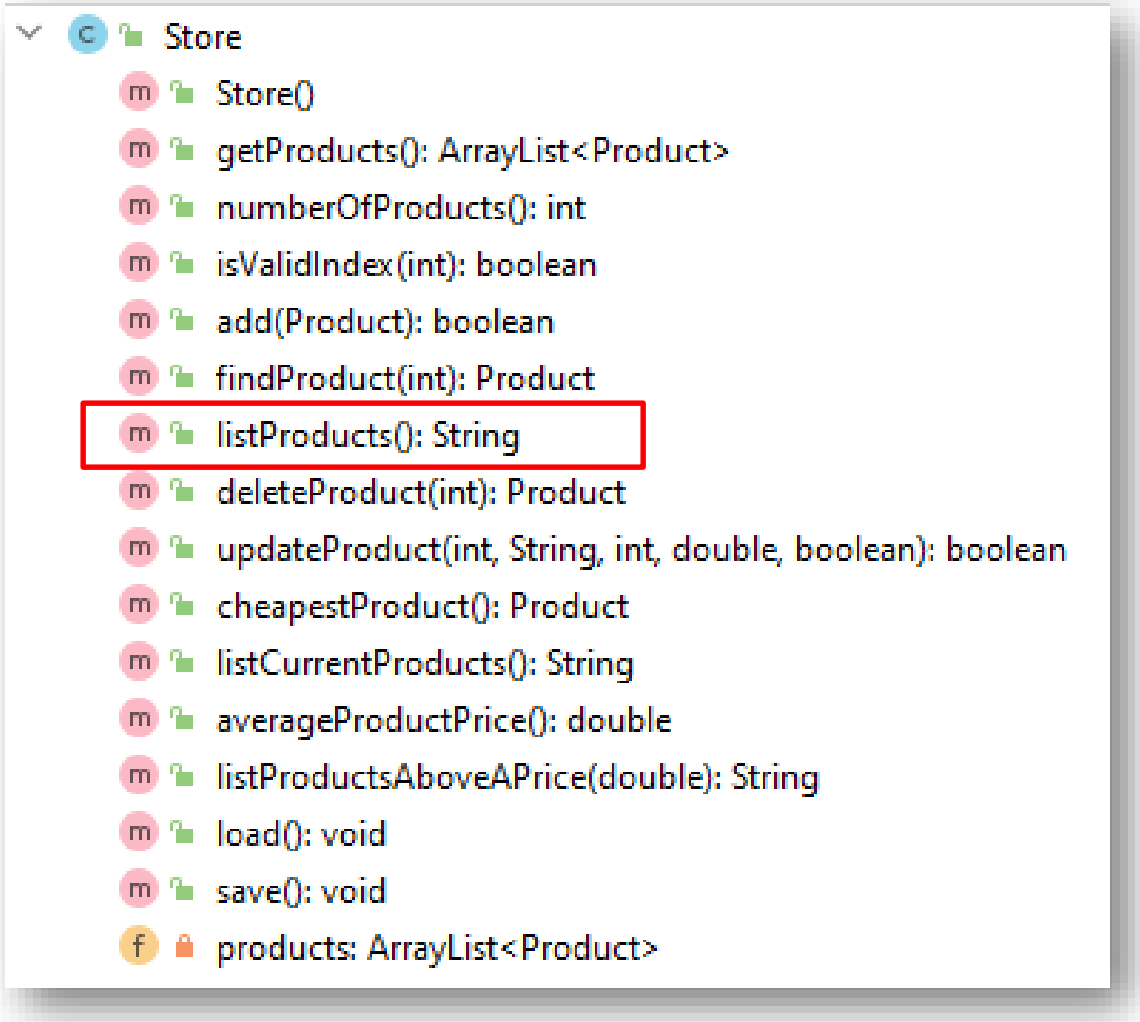


testing updateProduct

```
class StoreTest {

    private Product productBelow, productExact, productAbove, productZero;
    private Store storeWithProducts = new Store();
    private Store storeEmpty = new Store();

    @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 tru
        productExact = new Product("Television 50 Inches", 1000, 999, true
        //name, 21 chars, code 10000, unitCost 1000, inCurrentProductLine
        productAbove = new Product("Television 60 Inches.", 10000, 1000,
        //name, 0 chars, code 9999, unitCost 0, inCurrentProductLine false.
        productZero = new Product("", 9999, 0, false);

        storeWithProducts.add(productBelow);
        storeWithProducts.add(productExact);
        storeWithProducts.add(productAbove);
    }

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

```
    @Test
    void updatingANoteThatExistsReturnsTrueAndUpdates(){
        //check product index 2 exists and check the contents
        assertEquals(productAbove, storeWithProducts.findProduct(2));
        assertEquals("Television 60 Inches", storeWithProducts.findProduct(2).getProductNam
        assertEquals(-1, storeWithProducts.findProduct(2).getProductCode());
        assertEquals(1000, storeWithProducts.findProduct(2).getUnitCost());
        assertFalse(storeWithProducts.findProduct(2).isInCurrentProductLine());

        //update product 2 with new information and ensure contents updated successfully
        assertTrue(storeWithProducts.updateProduct(2, "Updating Product", 2000, 19.99, true
        assertEquals("Updating Product", storeWithProducts.findProduct(2).getProductName(
        assertEquals(2000, storeWithProducts.findProduct(2).getProductCode());
        assertEquals(19.99, storeWithProducts.findProduct(2).getUnitCost());
        assertTrue(storeWithProducts.findProduct(2).isInCurrentProductLine());
    }

    @Test
    void updatingAProductThatDoesNotExistReturnsFalse(){
        assertFalse(storeWithProducts.updateProduct(3, "Updating Product", 2, 19.99, false));
        assertFalse(storeWithProducts.updateProduct(-1, "Updating Product", 1002, 14.49, tru
        assertFalse(storeEmpty.updateProduct(0, "Updating Product", 1003, 199.99, false));
    }
```

Store.java – testing listProduct()



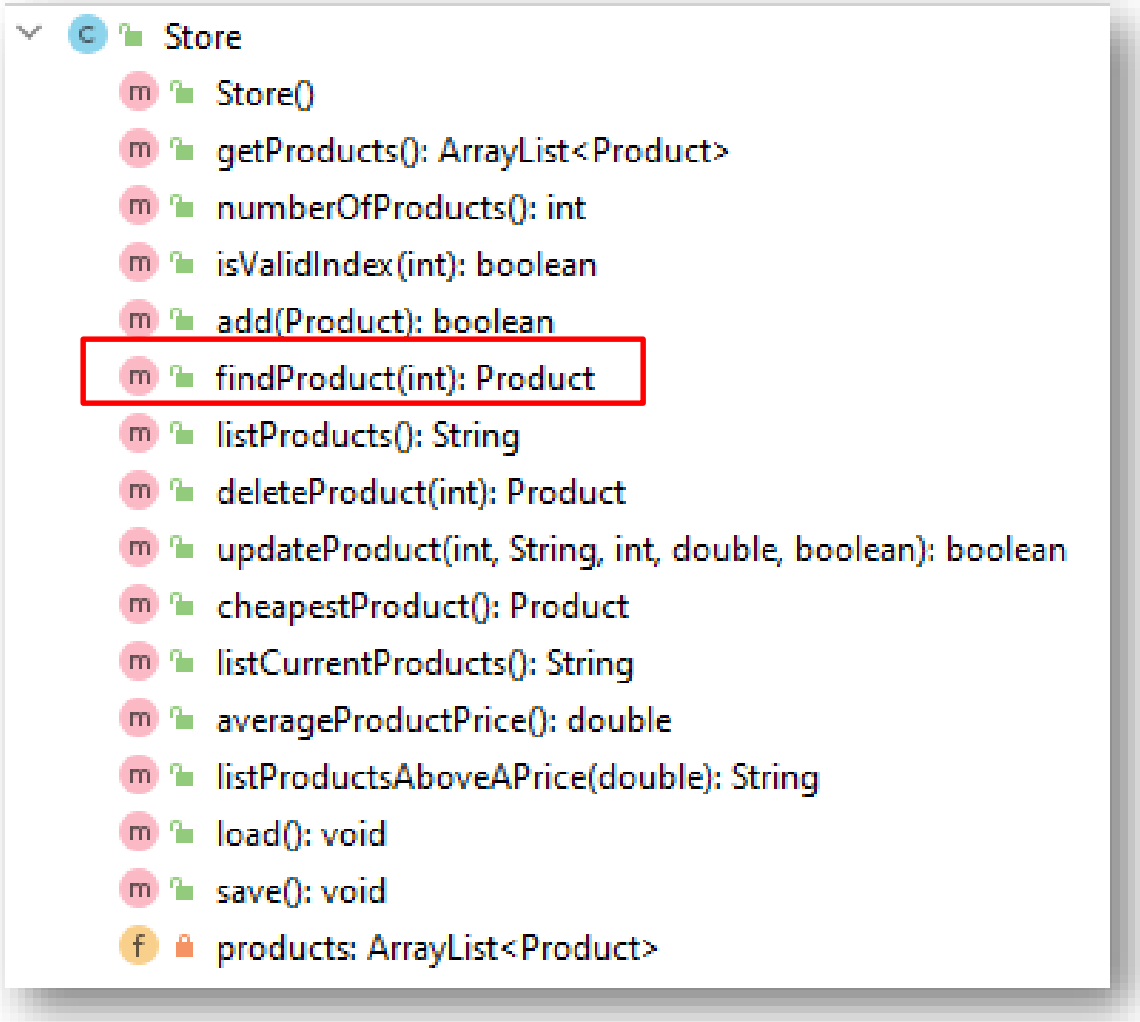

```
public String listProducts() {
    if (products.isEmpty()) {
        return "No products in the store";
    } else {
        String listOfProducts = "";
        for (int i = 0; i < products.size(); i++) {
            listOfProducts += i + ": " + products.get(i) + "\n";
        }
        return listOfProducts;
    }
}
```

testing listProduct()

```
class StoreTest {

    private Product productBelow, productExact, productAbove, productZero;
    private Store storeWithProducts = new Store();
    private Store storeEmpty = new Store();

    @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);

        storeWithProducts.add(productBelow);
        storeWithProducts.add(productExact);
        storeWithProducts.add(productAbove);
    }

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

```
@Test
void listProductsReturnsNoProductsStoredWhenArrayListIsEmpty() {
    assertEquals(0, storeEmpty.numberOfProducts());
    assertTrue(storeEmpty.listProducts().toLowerCase().contains("no products"
}

@Test
void listProductsReturnsProductsWhenArrayListHasProductsStored() {
    assertEquals(3, storeWithProducts.numberOfProducts());
    String productsString = storeWithProducts.listProducts();
    assertTrue(productsString.contains("Television 42Inches"));
    assertTrue(productsString.contains("Television 50 Inches"));
    assertTrue(productsString.contains("Television 60 Inches"));
}
```

Store.java – testing findProduct(int)

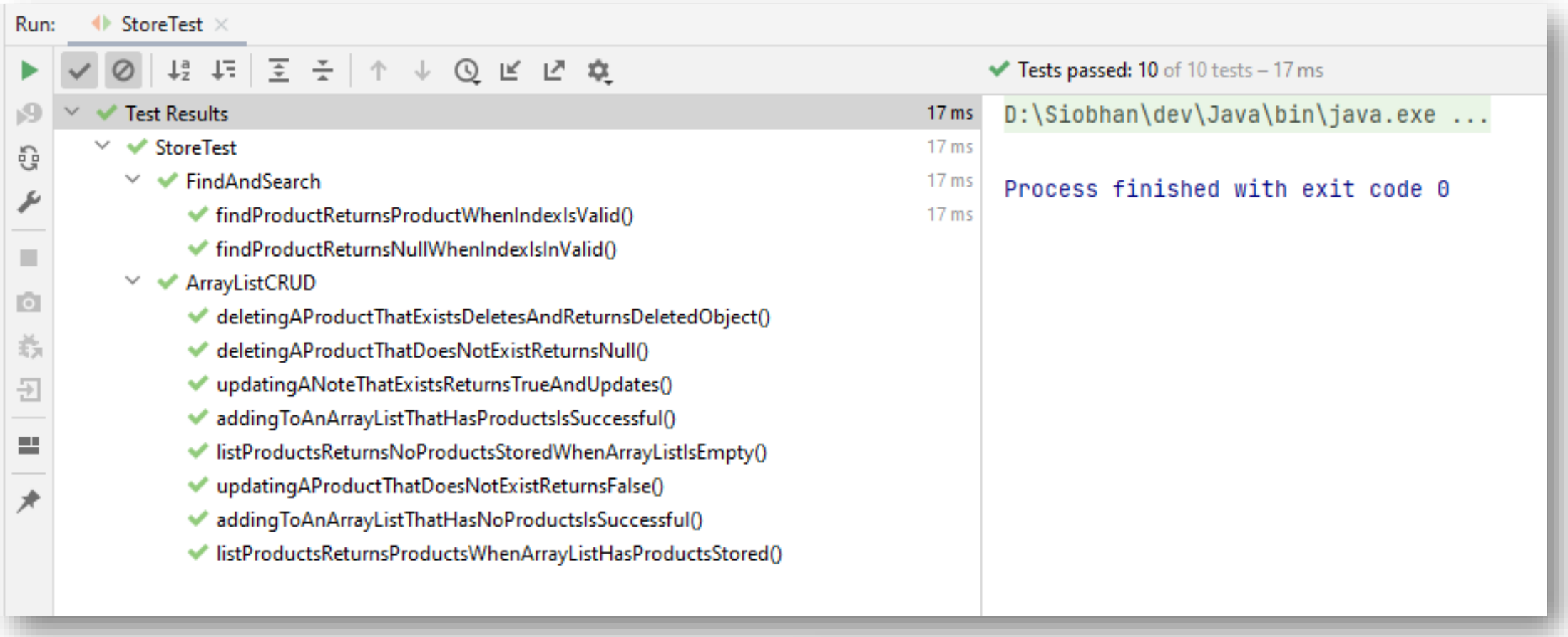


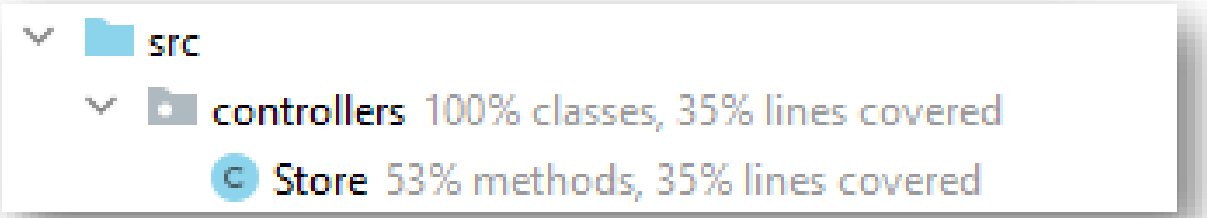

```
public Product findProduct(int index) {
    if (isValidIndex(index)) {
        return products.get(index);
    }
    return null;
}
```

```
class StoreTest {

    private Product productBelow, productExact, productAbove, productZero;
    private Store storeWithProducts = new Store();
    private Store storeEmpty = new Store();

    @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);

        storeWithProducts.add(productBelow);
        storeWithProducts.add(productExact);
        storeWithProducts.add(productAbove);
    }

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

testing findProduct(int)

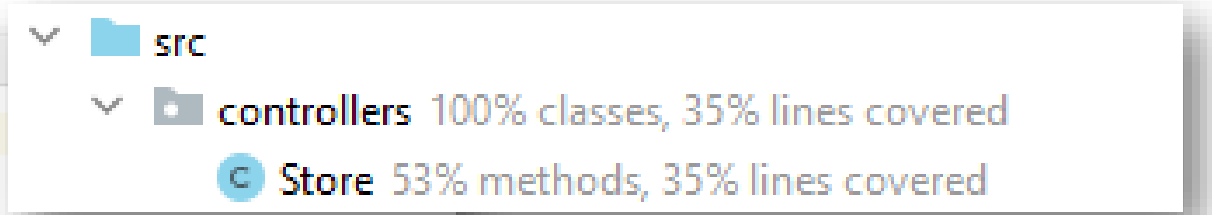
```
@Test
void findProductReturnsProductWhenIndexIsValid() {
    assertEquals(3, storeWithProducts.numberOfProducts());
    assertEquals(productBelow, storeWithProducts.findProduct(0));
    assertEquals(productAbove, storeWithProducts.findProduct(2));
}

@Test
void findProductReturnsNullWhenIndexIsInValid() {
    assertEquals(0, storeEmpty.numberOfProducts());
    assertNull(storeEmpty.findProduct(0));
    assertEquals(3, storeWithProducts.numberOfProducts());
    assertNull(storeWithProducts.findProduct(-1));
    assertNull(storeWithProducts.findProduct(3));
}
```


Run:
StoreTest
Tests passed: 10 of 10 tests – 17 ms
Test Results 17 ms
StoreTest 17 ms
FindAndSearch 17 ms
findProductReturnsProductWhenIndexIsValid() 17 ms
findProductReturnsNullWhenIndexIsInValid()
ArrayListCRUD
deletingAProductThatExistsDeletesAndReturnsDeletedObject()
deletingAProductThatDoesNotExistReturnsNull()
updatingANoteThatExistsReturnsTrueAndUpdates()
addingToAnArrayListThatHasProductsIsSuccessful()
listProductsReturnsNoProductsStoredWhenArrayListIsEmpty()
updatingAProductThatDoesNotExistReturnsFalse()
addingToAnArrayListThatHasNoProductsIsSuccessful()
listProductsReturnsProductsWhenArrayListHasProductsStored()
D:\Siobhan\dev\Java\bin\java.exe ...
Process finished with exit code 0


src
controllers 100% classes, 35% lines covered
Store 53% methods, 35% lines covered

Store.java

```java
public class Store {

    private ArrayList<Product> products;

    public Store(){
        products = new ArrayList<Product>();
    }

    public ArrayList<Product> getProducts() {
        return products;
    }

    /**
     * This method returns the number of product objects stored in the ArrayList.
     *
     * @return An int value representing the number of product objects in the ArrayList.
     */
    public int numberOfProducts() { return products.size(); }

    /**
     * This method takes in a number and checks if it is a valid index in the products ArrayList.
     *
     * @param index A number representing a potential index in the ArrayList.
     * @return True of the index number passed is a valid index in the ArrayList, false otherwise.
     */
    public boolean isValidIndex(int index) {
        return (index >= 0) && (index < products.size());
    }
```



Topic List

- Product and ProductTest.java
- JUnit Testing of Store.java
- Testing Driver.java

Driver.java (and ScannerInput.java)

- JUnit is not used to test the class that takes input from the console.

- Why do you think this is?

Any Questions?