Handling User Input

Utilities, Parsing & Wrappers, and Packages

Produced by: Dr. Siobhán Drohan
Ms. Mairead Meagher

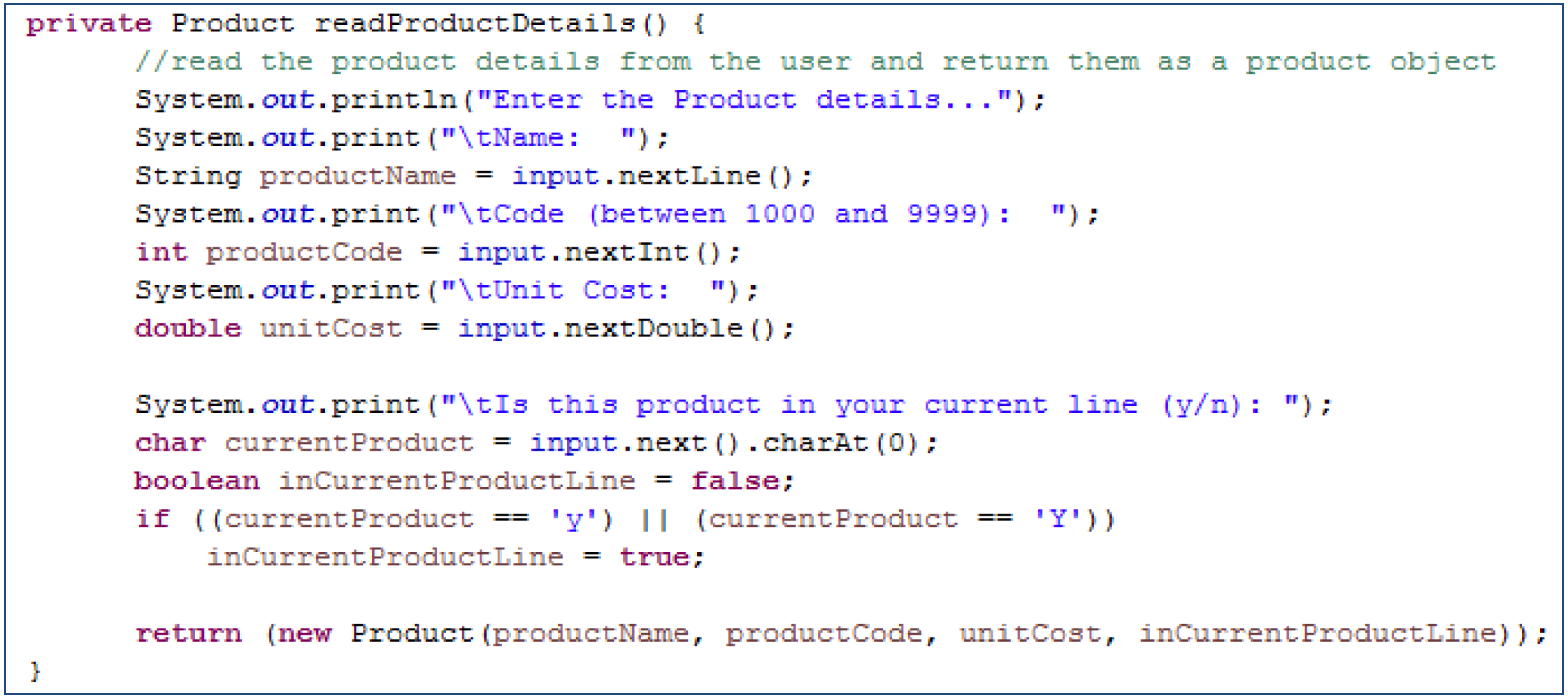
Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

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

Evolving Shop to be robust and have a package structure

Shop – making our app robust

What could cause a runtime exception here?

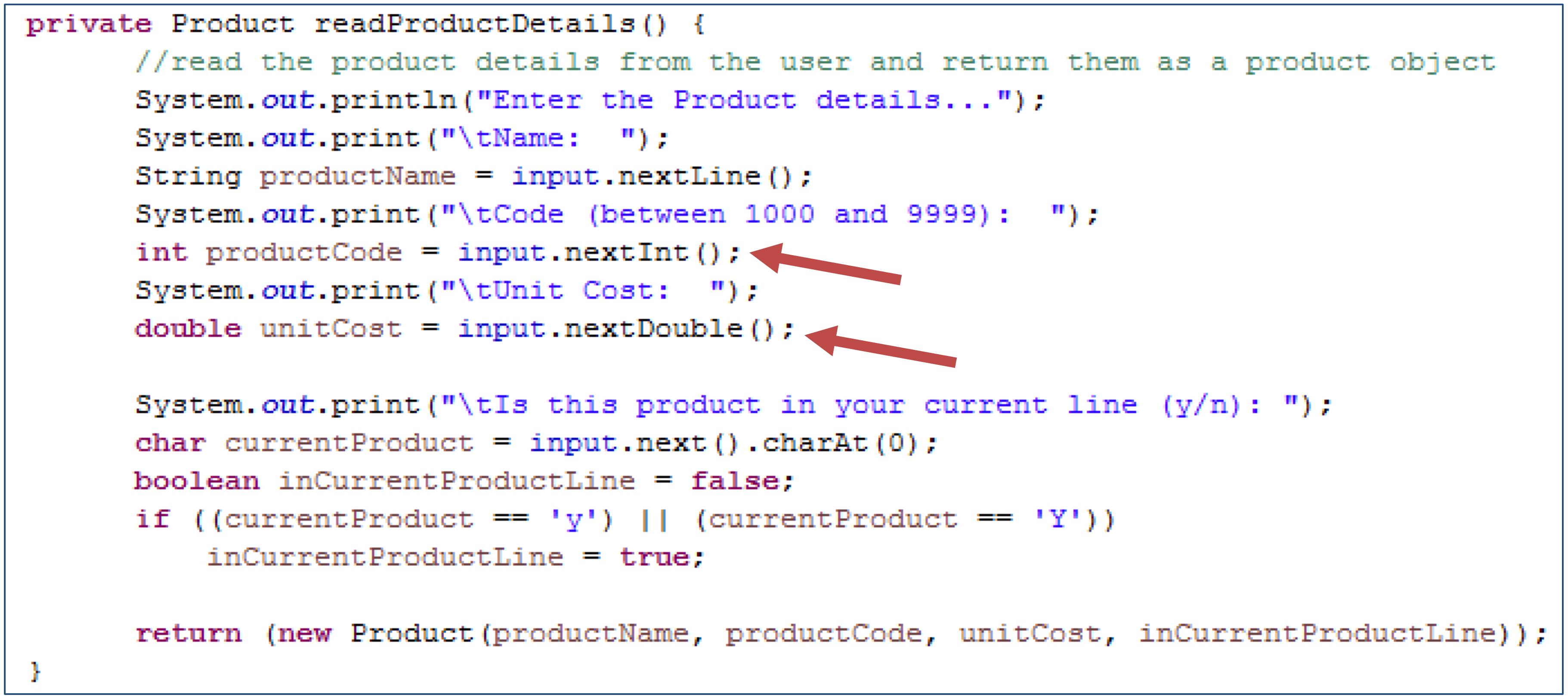


```
private Product readProductDetails() {
      //read the product details from the user and return them as a product object
      System.out.println("Enter the Product details...");
      System.out.print("\tName:  ");
      String productName = input.nextLine();
      System.out.print("\tCode (between 1000 and 9999):  ");
      int productCode = input.nextInt();
      System.out.print("\tUnit Cost:  ");
      double unitCost = input.nextDouble();

      System.out.print("\tIs this product in your current line (y/n): ");
      char currentProduct = input.next().charAt(0);
      boolean inCurrentProductLine = false;
      if ((currentProduct == 'y') || (currentProduct == 'Y'))
          inCurrentProductLine = true;

      return (new Product(productName, productCode, unitCost, inCurrentProductLine));
}
```

Shop – making our app robust



```
private Product readProductDetails() {
      //read the product details from the user and return them as a product object
      System.out.println("Enter the Product details...");
      System.out.print("\tName:  ");
      String productName = input.nextLine();
      System.out.print("\tCode (between 1000 and 9999):  ");
      int productCode = input.nextInt();
      System.out.print("\tUnit Cost:  ");
      double unitCost = input.nextDouble();

      System.out.print("\tIs this product in your current line (y/n): ");
      char currentProduct = input.next().charAt(0);
      boolean inCurrentProductLine = false;
      if ((currentProduct == 'y') || (currentProduct == 'Y'))
          inCurrentProductLine = true;

      return (new Product(productName, productCode, unitCost, inCurrentProductLine));
}
```

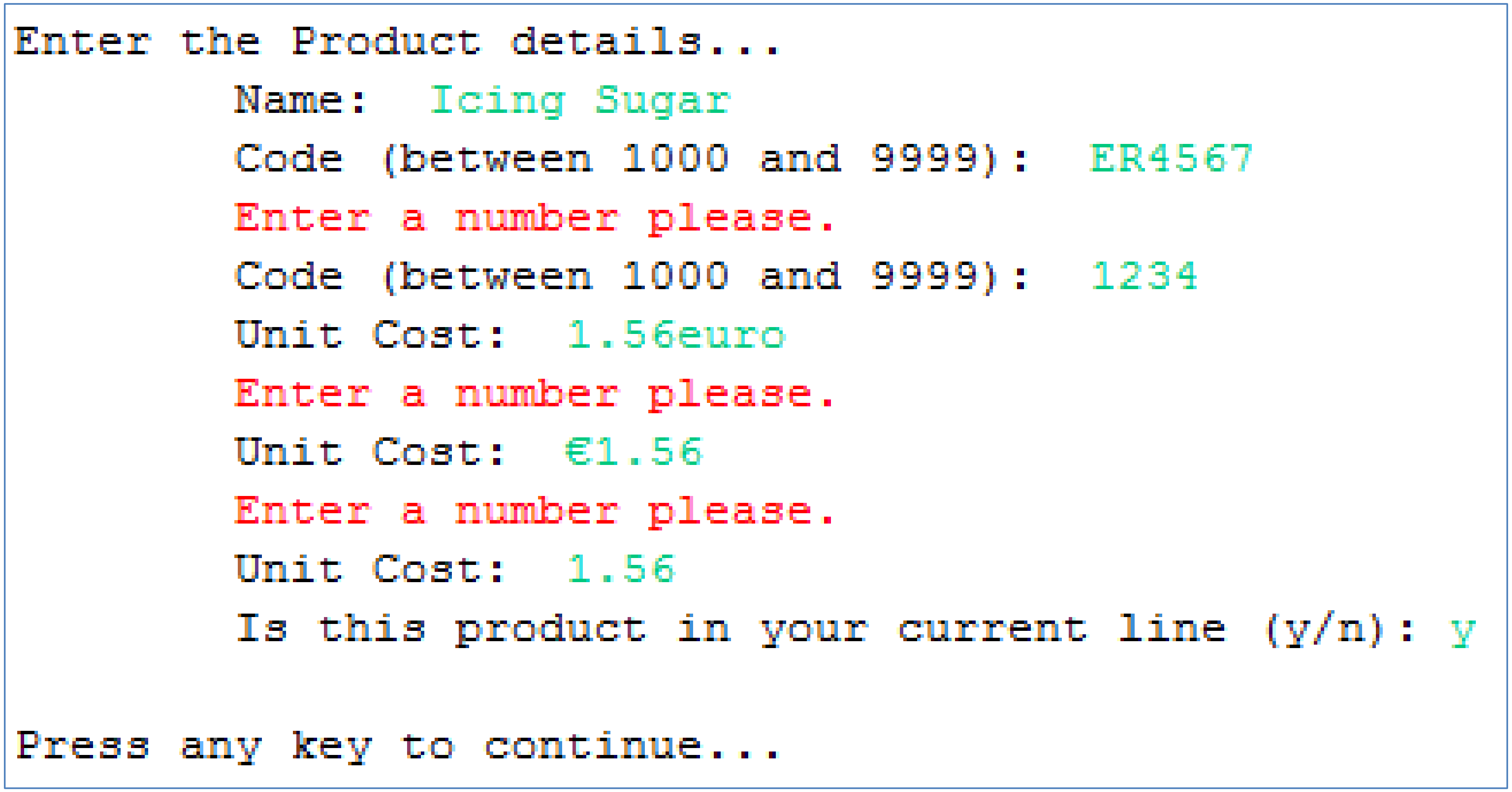
Shop – making our app robust

```java
System.out.print("\tCode (between 1000 and 9999):  ");
int productCode = input.nextInt();
System.out.print("\tUnit Cost:  ");
double unitCost = input.nextDouble();
```

```java
int productCode = 0;
boolean goodInput = false;
do {
    try {
        System.out.print("\tCode (between 1000 and 9999):  ");
        productCode = input.nextInt();
        goodInput = true;
    }
    catch (Exception e) {
        input.nextLine(); //swallows the buffer contents
        System.err.println("\tEnter a number please.");
    }
}  while (!goodInput);

double unitCost = 0;
goodInput = false;
do {
    try {
        System.out.print("\tUnit Cost:  ");
        unitCost = input.nextDouble();
        goodInput = true;
    }
    catch (Exception e) {
        input.nextLine();  //swallows the buffer contents
        System.err.println("\tEnter a number please.");
    }
}  while (!goodInput);
```

nextInt() and nextDouble() are now exception handled!



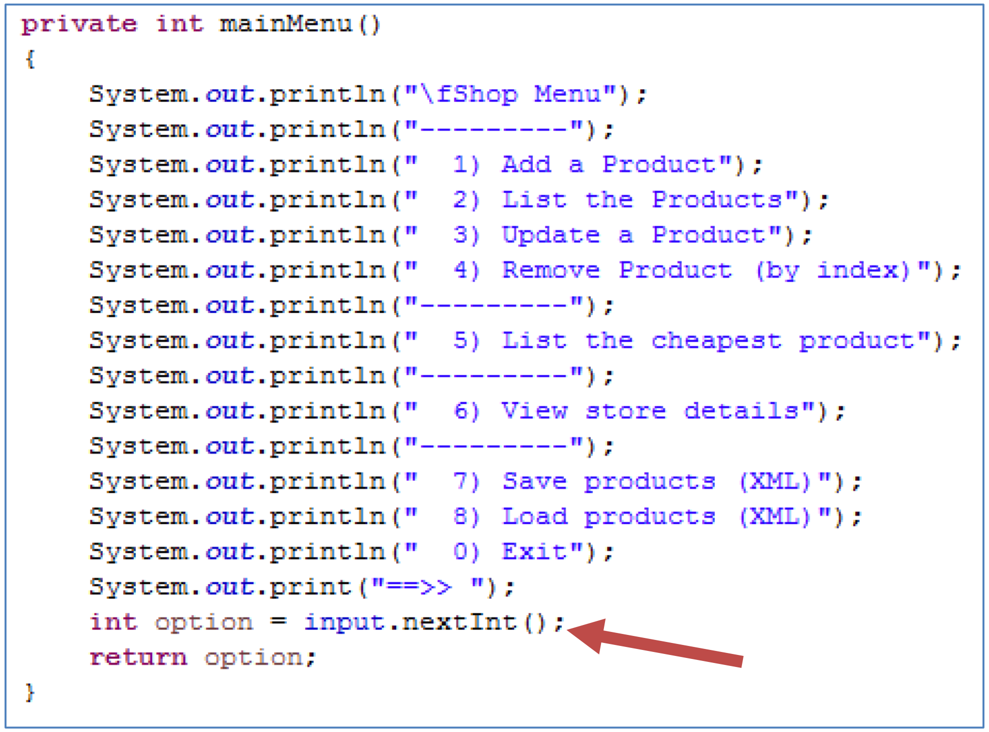
```
Enter the Product details...
        Name:  Icing Sugar
        Code (between 1000 and 9999):  ER4567
        Enter a number please.
        Code (between 1000 and 9999):  1234
        Unit Cost:  1.56euro
        Enter a number please.
        Unit Cost:  €1.56
        Enter a number please.
        Unit Cost:  1.56
        Is this product in your current line (y/n): y

Press any key to continue...
```

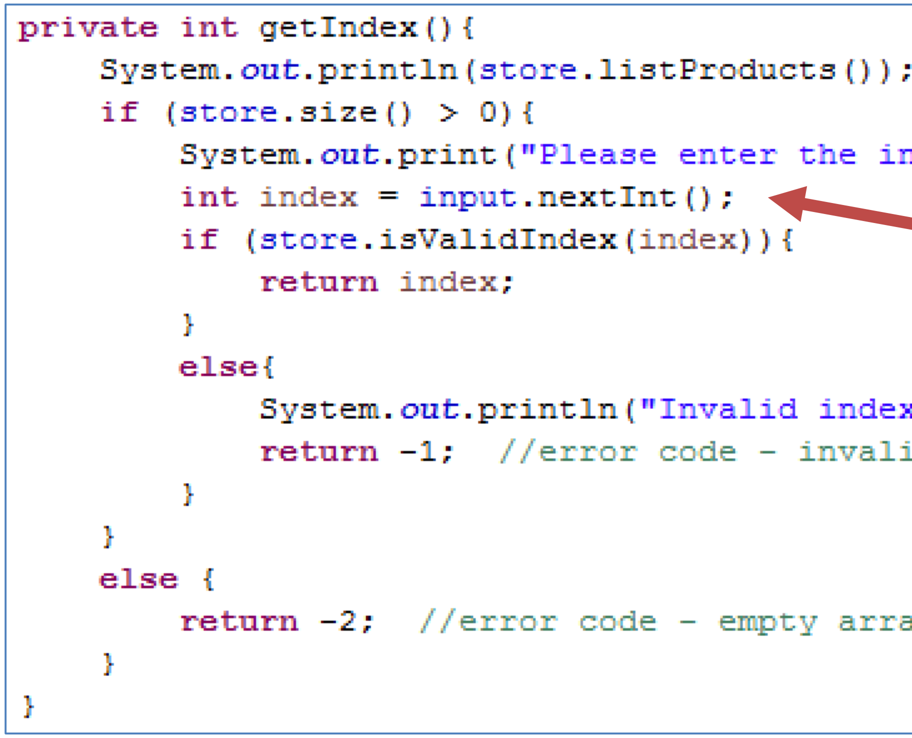
nextInt() and nextDouble() are now exception handled!

Shop – making our app robust

- But what about these int reads?



```
private int mainMenu()
{
    System.out.println("\fShop Menu");
    System.out.println("---------");
    System.out.println("  1) Add a Product");
    System.out.println("  2) List the Products");
    System.out.println("  3) Update a Product");
    System.out.println("  4) Remove Product (by index)");
    System.out.println("---------");
    System.out.println("  5) List the cheapest product");
    System.out.println("---------");
    System.out.println("  6) View store details");
    System.out.println("---------");
    System.out.println("  7) Save products (XML)");
    System.out.println("  8) Load products (XML)");
    System.out.println("  0) Exit");
    System.out.print("==>> ");
    int option = input.nextInt();
    return option;
}
```

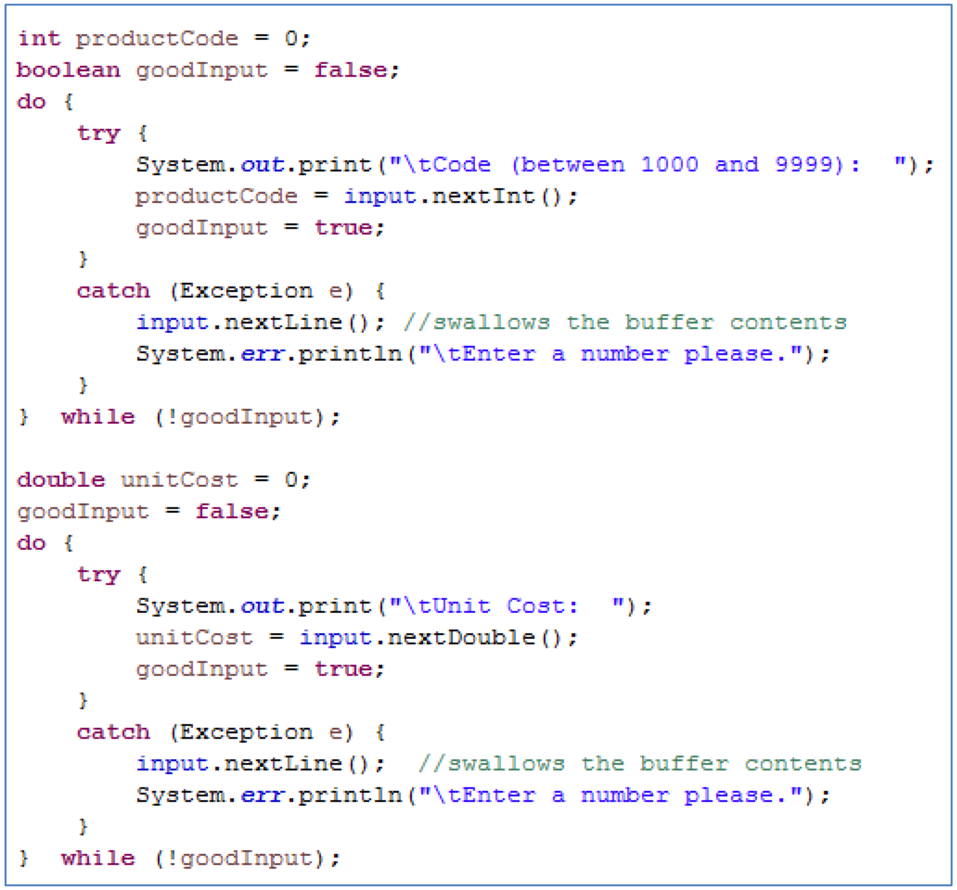


```
private int getIndex(){
    System.out.println(store.listProducts());
    if (store.size() > 0){
        System.out.print("Please enter the in
        int index = input.nextInt();
        if (store.isValidIndex(index)){
            return index;
        }
        else{
            System.out.println("Invalid index
            return -1;  //error code - invali
        }
    }
    else {
        return -2;  //error code - empty arra
    }
}
```

- Do I have to repeat the same code here?
- What happens if I add more int reads?

Shop – making our app robust

- In order to have **DRY** code, we should really write a private helper/utility method that can validate our int input.
- How would we write it?



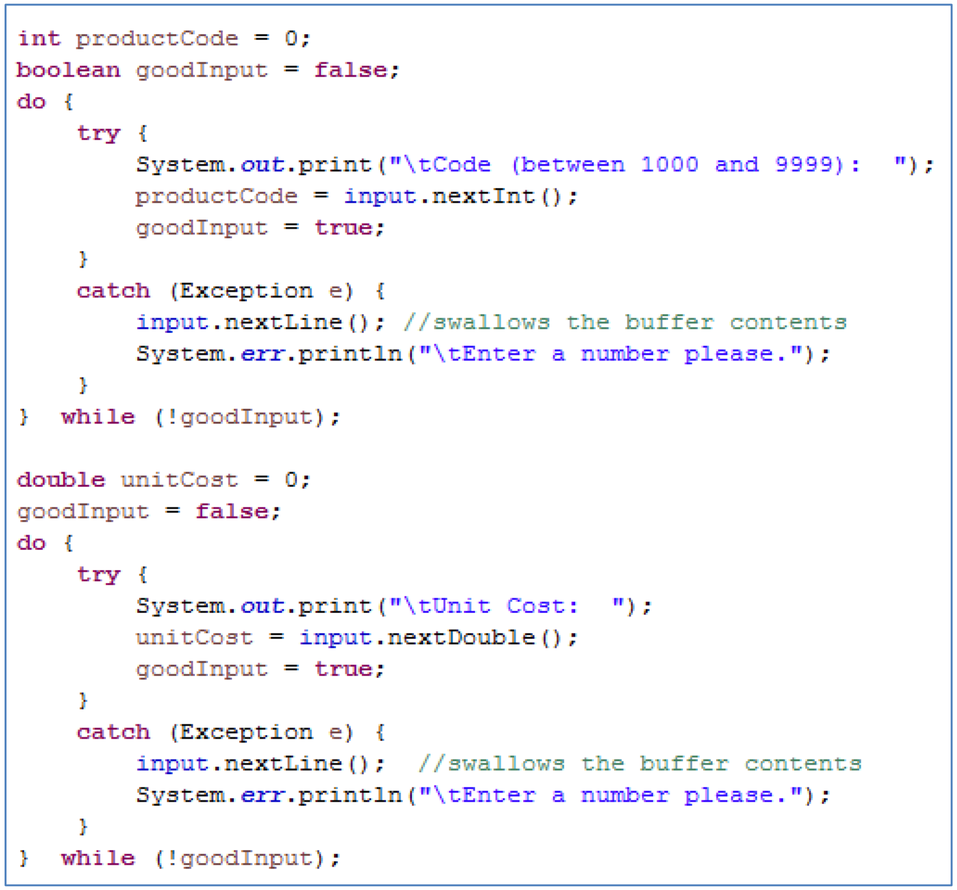
```
int productCode = 0;
boolean goodInput = false;
do {
    try {
        System.out.print("\tCode (between 1000 and 9999):  ");
        productCode = input.nextInt();
        goodInput = true;
    }
    catch (Exception e) {
        input.nextLine(); //swallows the buffer contents
        System.err.println("\tEnter a number please.");
    }
}  while (!goodInput);

double unitCost = 0;
goodInput = false;
do {
    try {
        System.out.print("\tUnit Cost:  ");
        unitCost = input.nextDouble();
        goodInput = true;
    }
    catch (Exception e) {
        input.nextLine();  //swallows the buffer contents
        System.err.println("\tEnter a number please.");
    }
}  while (!goodInput);
```

Shop – making our app robust

For this new method:

- We need to pass in a “prompt” string to be printed to the console.
- And return a valid int.



```
int productCode = 0;
boolean goodInput = false;
do {
    try {
        System.out.print("\tCode (between 1000 and 9999):  ");
        productCode = input.nextInt();
        goodInput = true;
    }
    catch (Exception e) {
        input.nextLine(); //swallows the buffer contents
        System.err.println("\tEnter a number please.");
    }
}  while (!goodInput);

double unitCost = 0;
goodInput = false;
do {
    try {
        System.out.print("\tUnit Cost:  ");
        unitCost = input.nextDouble();
        goodInput = true;
    }
    catch (Exception e) {
        input.nextLine();  //swallows the buffer contents
        System.err.println("\tEnter a number please.");
    }
}  while (!goodInput);
```

Shop – making our app robust

```java
private Product readProductDetails() {
    //read the product details from the user and return them as a product object
    System.out.println("Enter the Product details...");
    System.out.print("\tName:  ");
    String productName = input.nextLine();

    int productCode = validNextInt("\tCode (between 1000 and 9999):  ");
```

Here we are calling the new helper method to read a valid **int**.

```java
private int validNextInt(String prompt) {
    do {
        try {
            System.out.print(prompt);
            return input.nextInt();
        }
        catch (Exception e) {
            input.nextLine(); //swallows the buffer contents
            System.err.println("\tEnter a number please.");
        }
    }  while (true);

}
```

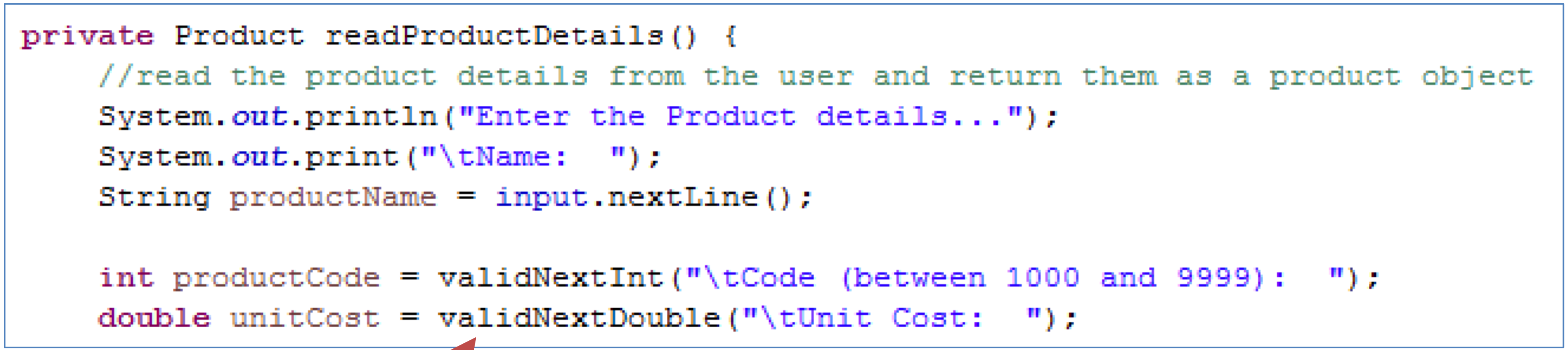
```java
private int mainMenu()
{
    System.out.println("\fShop Menu");
    System.out.println("---------");
    System.out.println("  1) Add a Product");
    System.out.println("  2) List the Products");
    System.out.println("  3) Update a Product");
    System.out.println("  4) Remove Product (by index)");
    System.out.println("---------");
    System.out.println("  5) List the cheapest product");
    System.out.println("---------");
    System.out.println("  6) View store details");
    System.out.println("---------");
    System.out.println("  7) Save products (XML)");
    System.out.println("  8) Load products (XML)");
    System.out.println("  0) Exit");
    int option = validNextInt("==>> ");
    return option;
}
```

```java
private int validNextInt(String prompt) {
    do {
        try {
            System.out.print(prompt);
            return input.nextInt();
        }
        catch (Exception e) {
            input.nextLine(); //swallows the buffer contents
            System.err.println("\tEnter a number please.");
        }
    }  while (true);

}
```

And again, we are calling the new helper method to read a valid **int**.

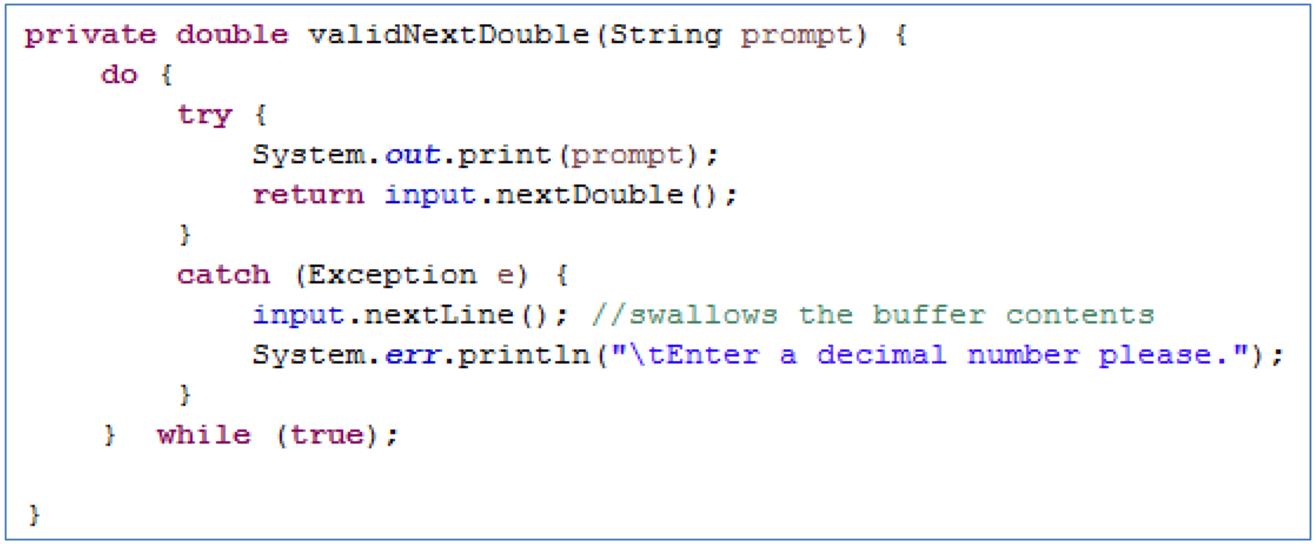
Shop – making our app robust



```java
private Product readProductDetails() {
    //read the product details from the user and return them as a product object
    System.out.println("Enter the Product details...");
    System.out.print("\tName:  ");
    String productName = input.nextLine();

    int productCode = validNextInt("\tCode (between 1000 and 9999):  ");
    double unitCost = validNextDouble("\tUnit Cost:  ");
```

Lets write a helper method now to read a valid **double**…

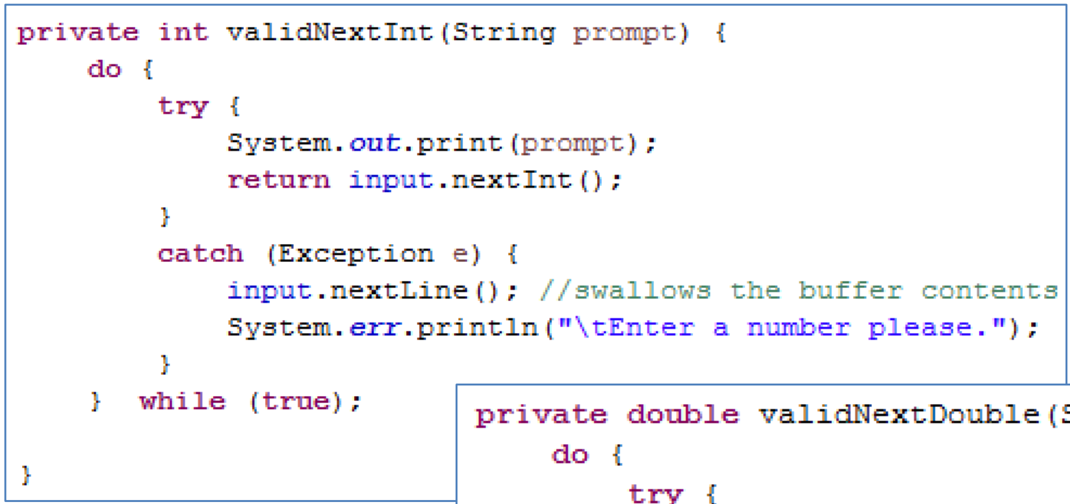


```java
private double validNextDouble(String prompt) {
    do {
        try {
            System.out.print(prompt);
            return input.nextDouble();
        }
        catch (Exception e) {
            input.nextLine(); //swallows the buffer contents
            System.err.println("\tEnter a decimal number please.");
        }
    } while (true);

}
```

Shop

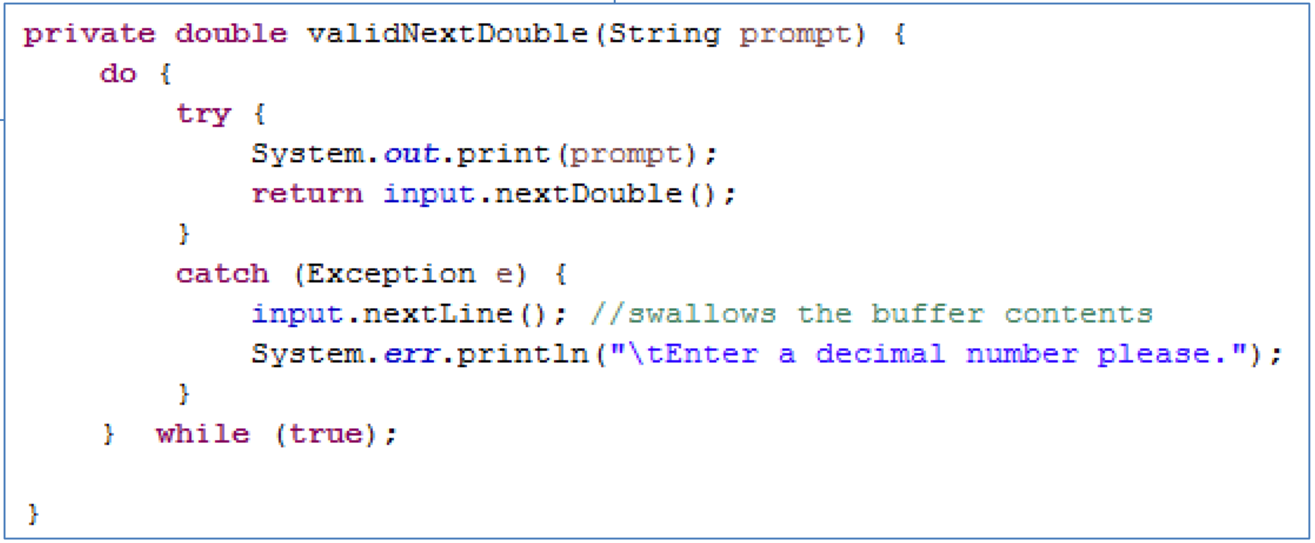
- Driver now has these two utility methods:



```
private int validNextInt(String prompt) {
    do {
        try {
            System.out.print(prompt);
            return input.nextInt();
        }
        catch (Exception e) {
            input.nextLine(); //swallows the buffer contents
            System.err.println("\tEnter a number please.");
        }
    } while (true);

}
```

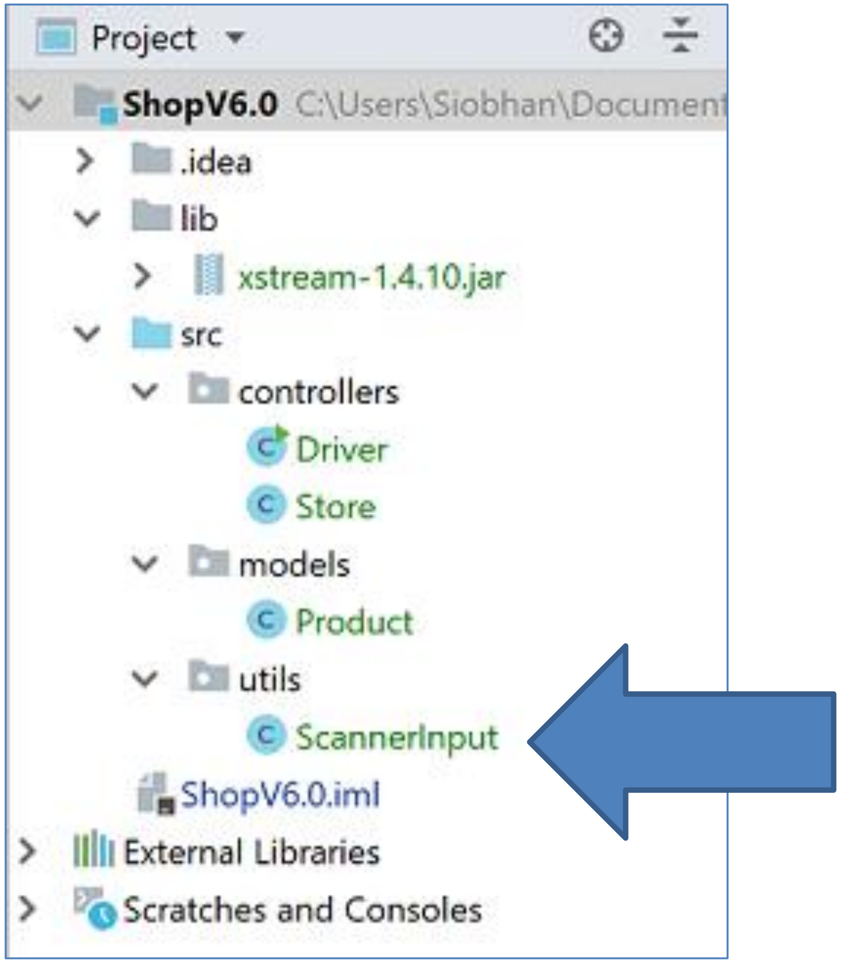
Do you think these methods could be used in another app?



```
private double validNextDouble(String prompt) {
    do {
        try {
            System.out.print(prompt);
            return input.nextDouble();
        }
        catch (Exception e) {
            input.nextLine(); //swallows the buffer contents
            System.err.println("\tEnter a decimal number please.");
        }
    } while (true);

}
```

Shop – utilities



- In the next few slides, we will remove these methods from the Driver class and re-write them into a separate “utility” class, ScannerInput.

Using utilities

Evolving Shop to use a ScannerInput Utility class

Shop – utilities

Creating our first utility class…

- In the **utils** package, create a new class called **ScannerInput**.
- Delete the validNextInt and validNextDouble methods from Driver; we are going to use a second approach for validating our input.

Creating our first utility class…

```
import java.util.Scanner;

public class ScannerInput {

    public static int readNextInt(String prompt) {
        do {
            var scanner = new Scanner(System.in);
            try {
                System.out.print(prompt);
                return Integer.parseInt(scanner.next());
            }
            catch (NumberFormatException e) {
                System.err.println("\tEnter a number please.");
            }
        }  while (true);
    }

    public static double readNextDouble(String prompt) {
        do {
            var scanner = new Scanner(System.in);
            try{
                System.out.print(prompt);
                return Double.parseDouble(scanner.next());
            }
            catch (NumberFormatException e) {
                System.err.println("\tEnter a number please.");
            }
        }  while (true);
    }
}
```

Another approach to validating input

- In ScannerInput, we are now using wrapper classes and parsing for validating input:

```
public static int readNextInt(String prompt) {
       do {
           var scanner = new Scanner(System.in);
           try {
               System.out.print(prompt);
               return Integer.parseInt(scanner.next());
           }
           catch (NumberFormatException e) {
               System.err.println("\tEnter a number please.");
           }
       }  while (true);
   }
```

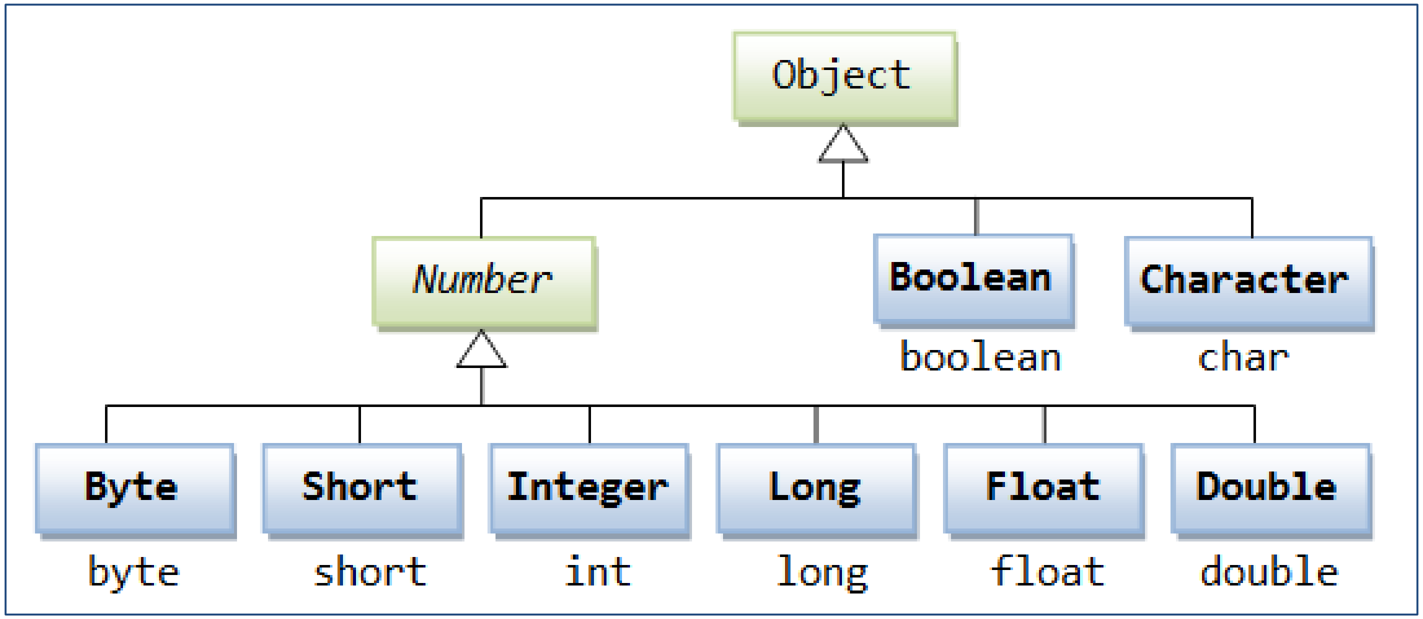
Wrapper classes

- Normally, when we work with Numbers, we use primitive data types such as byte, int, long, double, etc.
- However, in development, we come across situations where we need to use objects instead of primitive data types.
- In order to achieve this, Java provides **wrapper classes**.

https://www.tutorialspoint.com/java/java_numbers.htm

Wrapper classes

- All the wrapper classes (Integer, Long, Byte, Double, Float, Short) are subclasses of the abstract class Number.



https://www.tutorialspoint.com/java/java_numbers.htm

Wrapper classes

- The object of the wrapper class contains or wraps its respective primitive data type.
- Converting primitive data types into object is called **autoboxing**, and this is taken care by the compiler.
- Therefore, while using a wrapper class you just need to pass the value of the primitive data type to the constructor of the Wrapper class.

https://www.tutorialspoint.com/java/java_numbers.htm

Wrapper classes

- The Wrapper object will be converted back to a primitive data type, and this process is called **unboxing**.
- The **Number** class is part of the java.lang package.

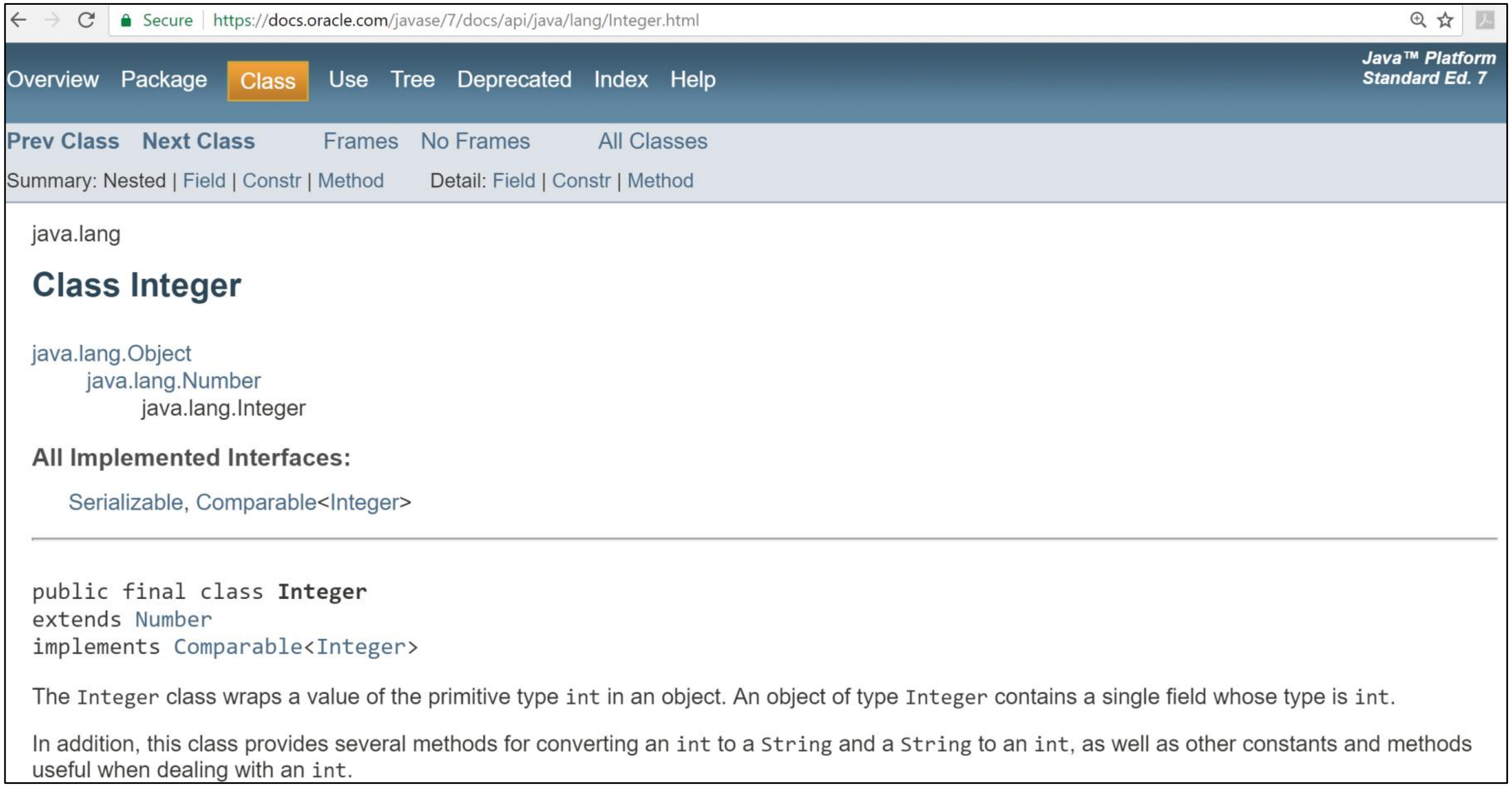
https://www.tutorialspoint.com/java/java_numbers.htm

Wrapper classes – boxing/unboxing

```
public class Test {
        public static void main(String args[]) {
                Integer x = 5; // boxes int to an Integer object
                x =  x + 10;   // unboxes the Integer to an int
                System.out.println(x);    //prints 15 to console
        }
}
```

https://www.tutorialspoint.com/java/java_numbers.htm

Parsing


Secure
https://docs.oracle.com/javase/7/docs/api/java/lang/Integer.html
Overview Package Class Use Tree Deprecated Index Help
Java™ Platform Standard Ed. 7
Prev Class Next Class Frames No Frames All Classes
Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.lang

Class Integer

java.lang.Object
    java.lang.Number
        java.lang.Integer

All Implemented Interfaces:

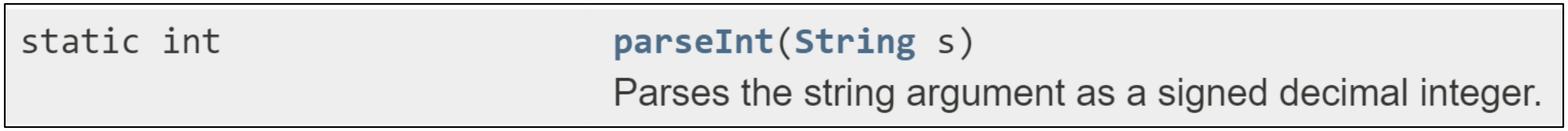
Serializable, Comparable<Integer>

```
public final class Integer
extends Number
implements Comparable<Integer>
```

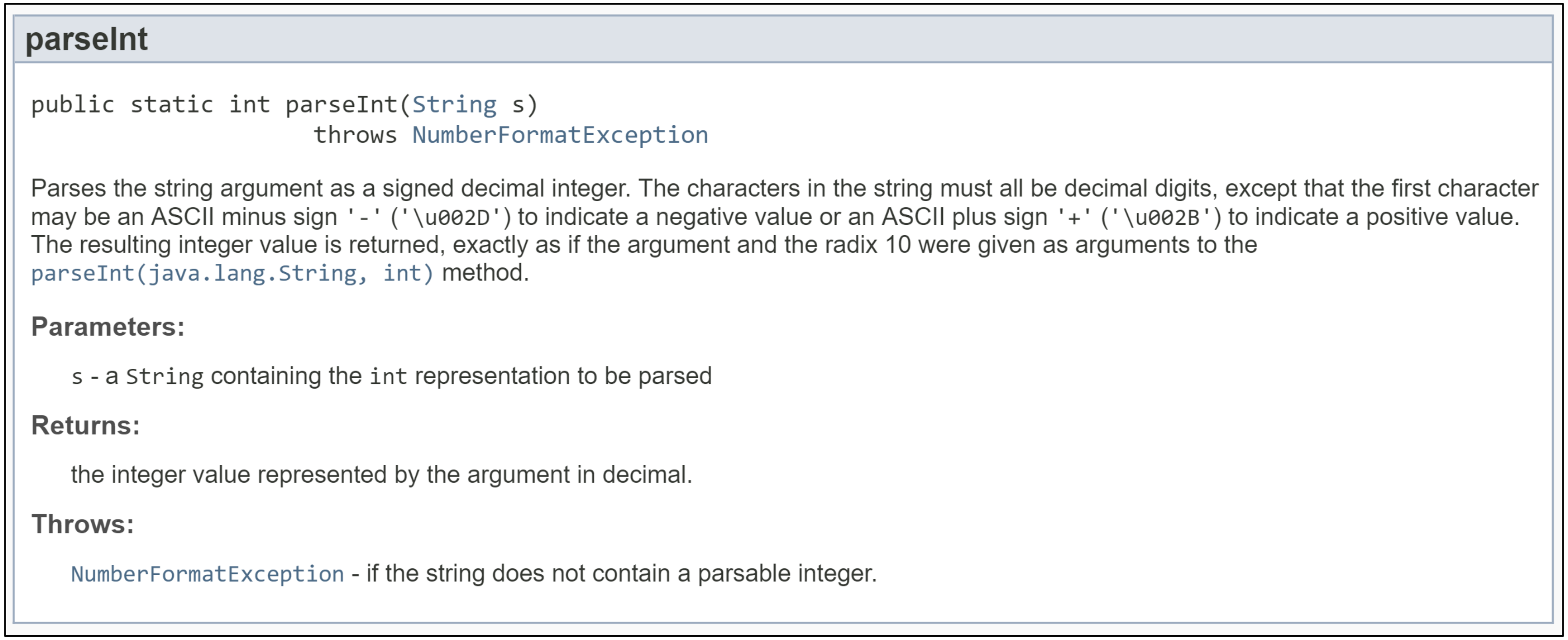
The `Integer` class wraps a value of the primitive type `int` in an object. An object of type `Integer` contains a single field whose type is `int`.

In addition, this class provides several methods for converting an `int` to a `String` and a `String` to an `int`, as well as other constants and methods useful when dealing with an `int`.

Parsing



`static int`	`parseInt(String s)` Parses the string argument as a signed decimal integer.



parseInt

```
public static int parseInt(String s)
                    throws NumberFormatException
```

Parses the string argument as a signed decimal integer. The characters in the string must all be decimal digits, except that the first character may be an ASCII minus sign `'-'` (`'\u002D'`) to indicate a negative value or an ASCII plus sign `'+'` (`'\u002B'`) to indicate a positive value. The resulting integer value is returned, exactly as if the argument and the radix 10 were given as arguments to the `parseInt(java.lang.String, int)` method.

Parameters:

`s` - a `String` containing the `int` representation to be parsed

Returns:

the integer value represented by the argument in decimal.

Throws:

`NumberFormatException` - if the string does not contain a parsable integer.

Shop – utilities

Calling the methods in our first utility class

Driver now can't find our new methods…

```
//gather the product data from the user and create a new product.
private void addProduct() {
    System.out.print("Enter the Product Name:  ");
    String productName = input.nextLine();

    int productCode = readNextInt("Enter the product code: ");
    double unitCost = readNextDouble("Enter the Unit Cost:  ");

    System.out.print("Is this product in your current line (y/n): ");
    char currentProduct = input.next().charAt(0);
    boolean inCurrentProductLine = false;
    if ((currentProduct == 'y') || (currentProduct == 'Y'))
        inCurrentProductLine = true;

    store.add(new Product(productName, productCode, unitCost, inCurrentProductLine))
}
```

Shop – utilities

Calling the methods in our first utility class

import utils.ScannerInput.*;

Then change method calls to: ***ScannerInput.readNextInt();***
ScannerInput.readNextDouble();

```
//gather the product data from the user and create a new product.
private void addProduct() {
    System.out.print("Enter the Product Name:  ");
    String productName = input.nextLine();

    int productCode = ScannerInput.readNextInt("Enter the product code: "
    double unitCost = ScannerInput.readNextDouble("Enter the Unit Cost:

    System.out.print("Is this product in your current line (y/n): ");
    char currentProduct = input.next().charAt(0);
    boolean inCurrentProductLine = false;
    if ((currentProduct == 'y') || (currentProduct == 'Y'))
        inCurrentProductLine = true;

    store.add(new Product(productName, productCode, unitCost, inCurrentPr
}
```

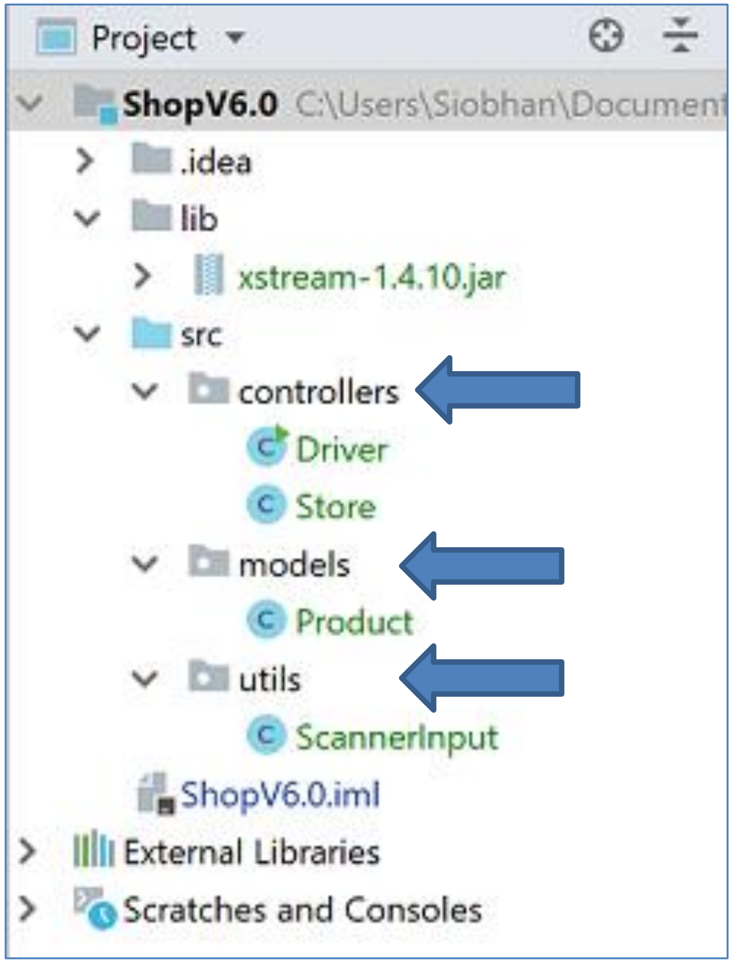
Shop – utilities

- When testing the app, you might notice that our dummy reads for emptying the buffer are now causing a problem!
- We can get rid of these now and, as we are creating a new Scanner object for each **int** and **double** read, we don’t have to worry about emptying our buffers anymore!

Using Packages

Evolving Shop to use a Package Structure

Shop – packages

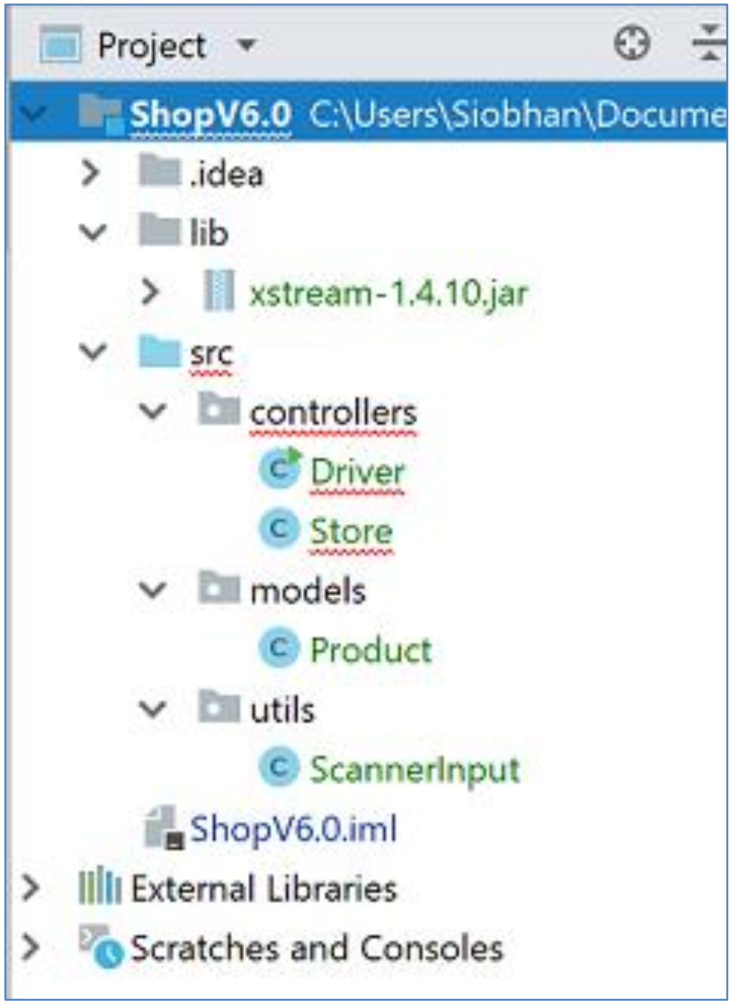


- As our app is getting larger, we will start using “packages” to structure our app layout.

Shop – utilities and packages

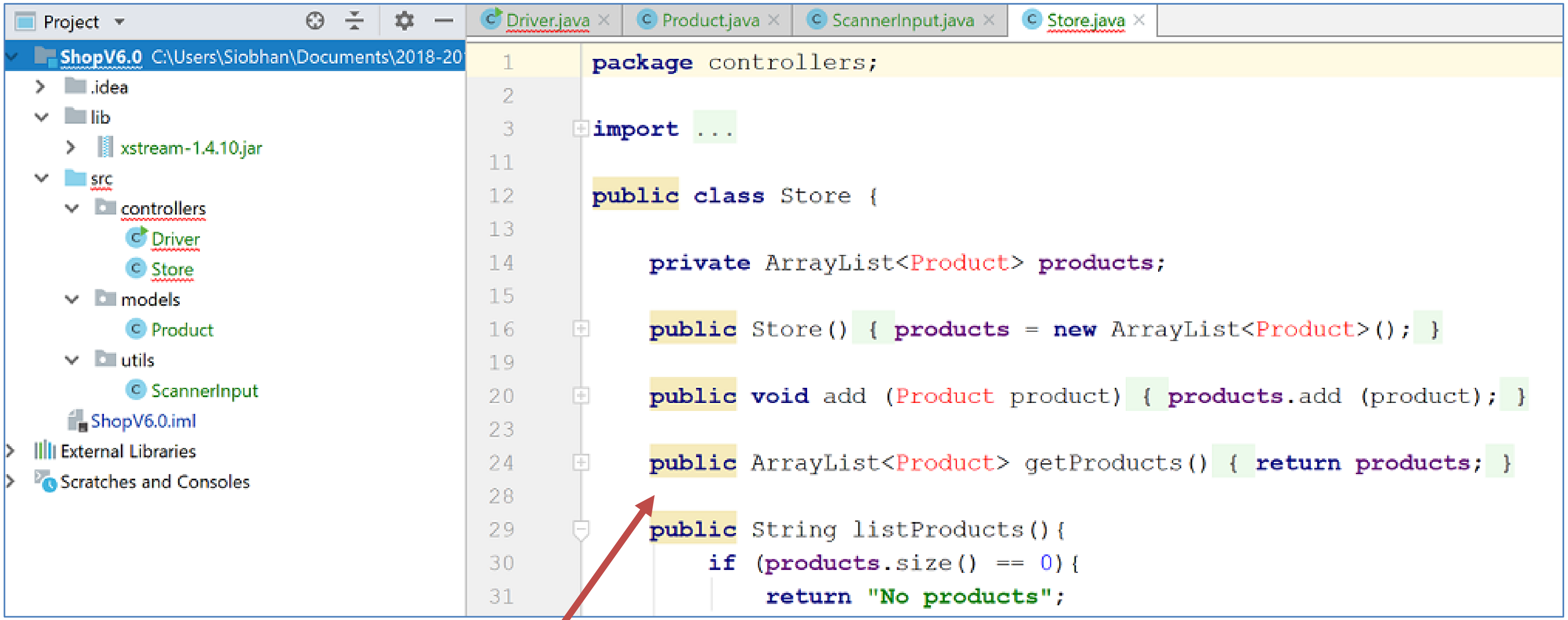
- Refactor your Shop project to create this package structure.
  - Right-click on the **src** folder and select New → Package. Enter “models” as the package name.
  - Repeat this process and create two other packages called “controllers” and “utils”.

Shop – utilities and packages



Copy the Shop classes into package locations specified in the screen shot.

Shop – utilities and packages



Project

ShopV6.0 C:\Users\Siobhan\Documents\2018-20
- .idea
- lib
  - xstream-1.4.10.jar
- src
  - controllers
    - Driver
    - Store
  - models
    - Product
  - utils
    - ScannerInput
- ShopV6.0.iml
- External Libraries
- Scratches and Consoles

Driver.java | Product.java | ScannerInput.java | Store.java

```java
package controllers;

import ...

public class Store {

    private ArrayList<Product> products;

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

    public void add (Product product) { products.add (product); }

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

    public String listProducts(){
        if (products.size() == 0){
            return "No products";
```

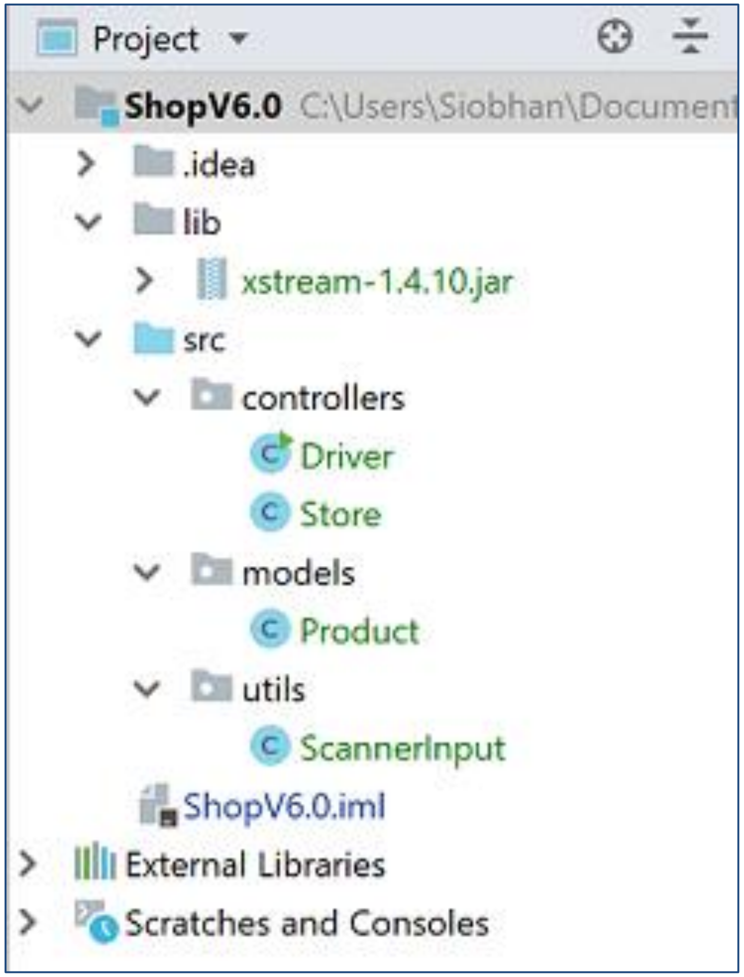
This move caused the following error: The Product class can't be found in the Driver and Store class.

Shop – utilities and packages

To fix this error, use IntelliJ’s Alt + Enter to:

- import **models.Product;** into Driver and Store classes.
- import **utils.ScannerInput;** into Driver.

Shop – utilities and packages



- The errors are now gone.
- Test the app to make sure it is running as expected.

Any Questions?