Exception Handling

Handling bad user input…

Produced by: Dr. Siobhán Drohan
Maireád Meagher

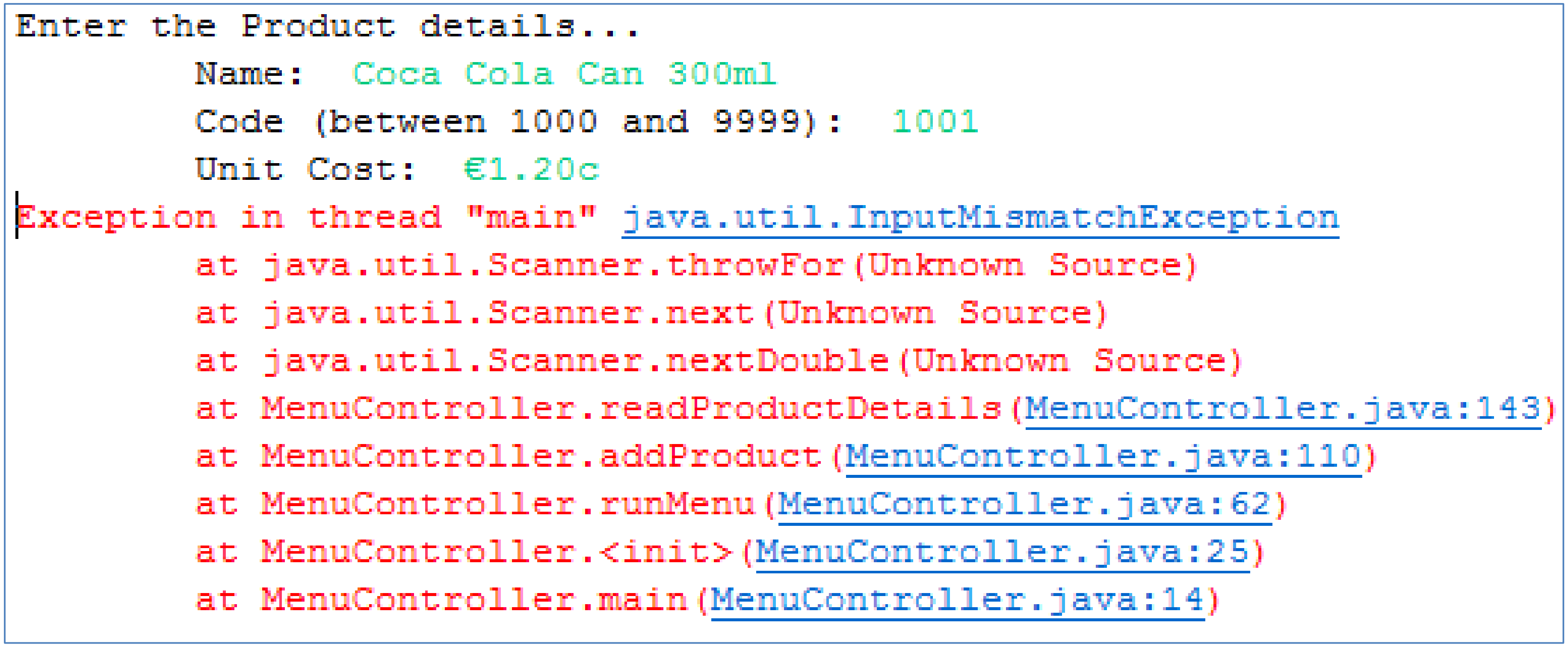
Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

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

ShopV4.0 (or any version)

- When testing it, did you try to enter a **String** instead of an **int**? e.g. for the Product code?

- What happened?



```
Enter the Product details...
        Name:  Coca Cola Can 300ml
        Code (between 1000 and 9999):  1001
        Unit Cost:  €1.20c
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Unknown Source)
        at java.util.Scanner.next(Unknown Source)
        at java.util.Scanner.nextDouble(Unknown Source)
        at MenuController.readProductDetails(MenuController.java:143)
        at MenuController.addProduct(MenuController.java:110)
        at MenuController.runMenu(MenuController.java:62)
        at MenuController.<init>(MenuController.java:25)
        at MenuController.main(MenuController.java:14)
```

ShopV4.0 is NOT robust

ShopV4.0 (or any version)

- The following code caused a runtime error...

```
double unitCost = input.nextDouble();
```

- This is called a **runtime exception**.
- How do we fix this? How do we stop the program from crashing?

What are Exceptions?

- An Exception is an object that signals that some unusual condition has occurred while the program is executing.

- Exceptions are intended to be *detected* and *handled*, so that the program can continue in a sensible way if at all possible.

- Java has many predefined Exception objects, and we can also create our own.

When an exception occurs…

...the normal flow of execution is disrupted and transferred to code, which can handle the exception condition.

The exception mechanism is a lot cleaner than having to check an error value after every method call that could potentially fail.

RuntimeException

- is a subclass of the Exception class
- encompasses all exceptions which can ordinarily happen at run-time.
- these exceptions can be thrown by any java statement or a method call.
- can be avoided through good programming practices!

RuntimeException	**Example Causes**
ArithmeticException	Can be caused by dividing by zero.
ArrayIndexOutOfBoundsException	Referencing an array index number of 7 when only 5 exist in the array.
NullPointerException	trying to access an object that has no memory allocated yet.

Catching Exceptions

Catching an exception means declaring that you can handle exceptions of a particular class from a particular block of code.

- You specify the block of code and then provide handlers for various classes of exception.

- If an exception occurs then execution transfers to the corresponding piece of handler code.

try and catch

To catch exceptions, you surround a block of code with a "try, catch" statement.

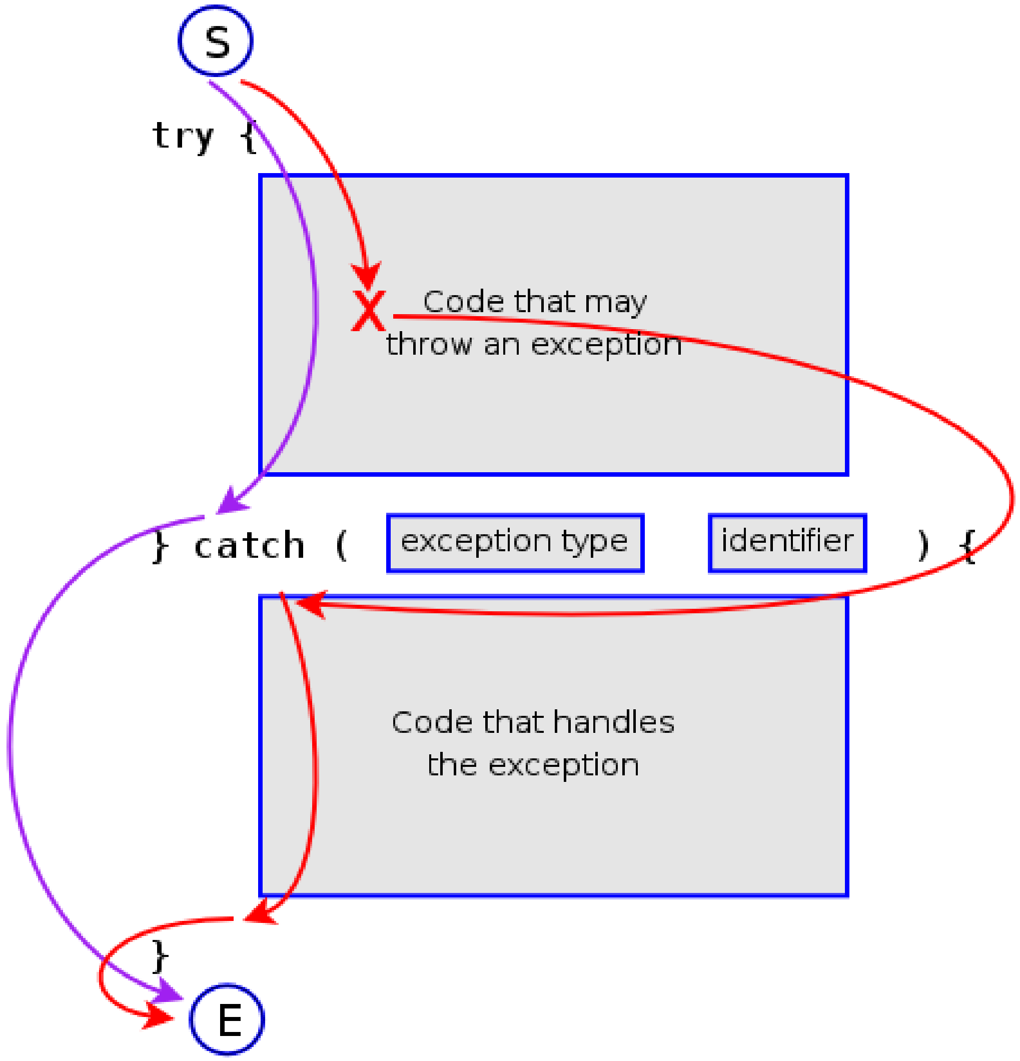
```
try{
    // The try clause is the piece of code which you want to try to execute.
   // it contains statements in which an exception could be raised
}
catch (Exception e){
   // The catch clauses are the handlers for the various exceptions.
   //it contains code to handle Exception and recover
}
```

Example of try and catch

```
try{
        myMethod();
}
catch (Exception e){
        System.err.println(“Caught Exception:  “ + e)
}
```

The parameter ***e*** is of type Exception and we can use it to print out what exception occurred.

Flow of control in Exception Handing


S
try {
Code that may throw an exception
} catch (
exception type
identifier
) {
Code that handles the exception
}
E

ShopV5.0 – making our app robust

```
try {
    System.out.print("Please enter the product code: ");
    code = input.nextInt();
  }
  catch (Exception e) {
     input.nextLine(); //swallows the buffer contents
     System.out.println("Number expected - you entered  text");
   }
```

Improve – loop until input valid

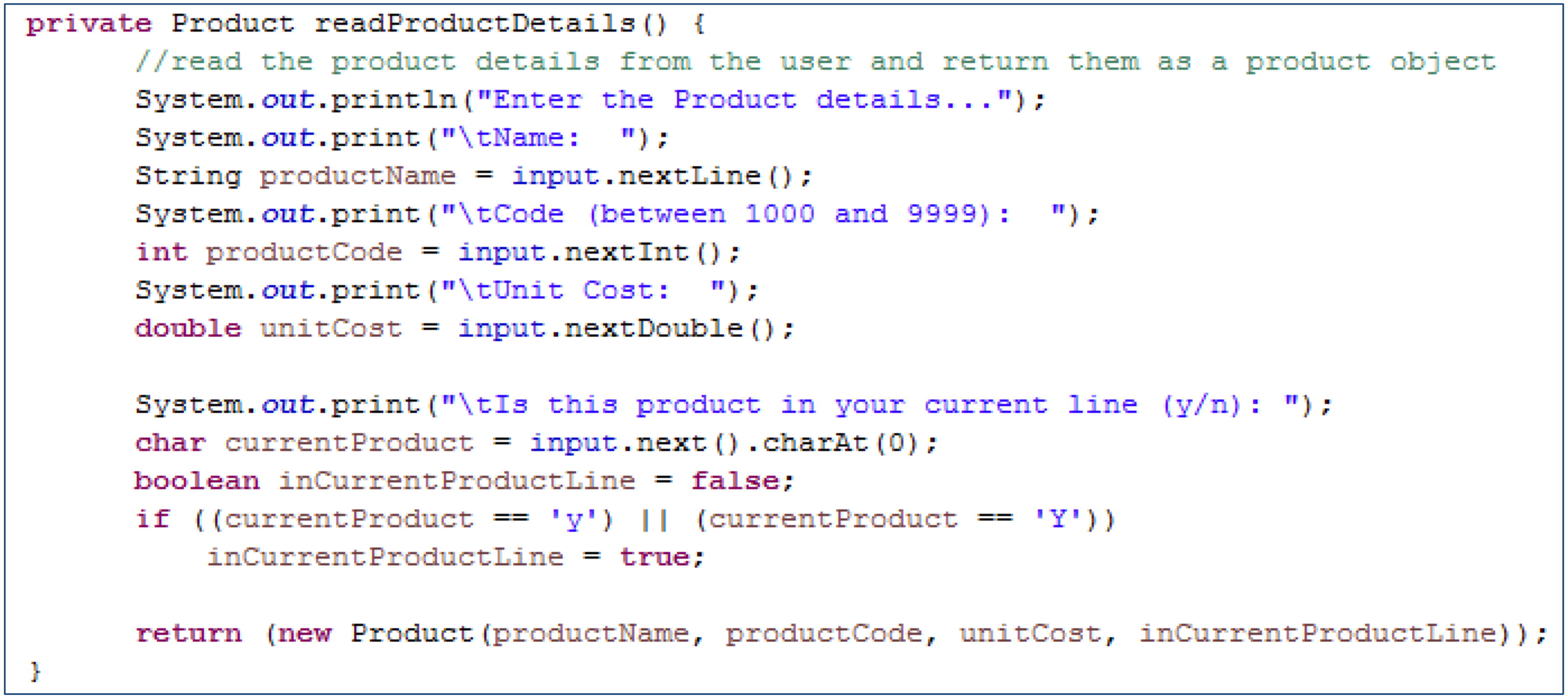
```
boolean goodInput = false;    //Loop  Control Variable

 while (! goodInput ) {
      try {
           System.out.print("Please enter the product code: ");
           code = input.nextInt();
           goodInput = true;
      }
      catch (Exception e) {
           input.nextLine(); //swallows the buffer contents
           System.out.println("Num expected - you entered  text");
      }
 }
```

Using do..while

```
  boolean goodInput = false;
  do {
      try {
          System.out.print("Please enter the product code: ");
          code = input.nextInt();
          goodInput = true;
      }
      catch (Exception e) {
          input.nextLine(); //swallows the buffer contents
          System.out.println("Num expected - you entered  text");
      }
}  while (!goodInput);
```

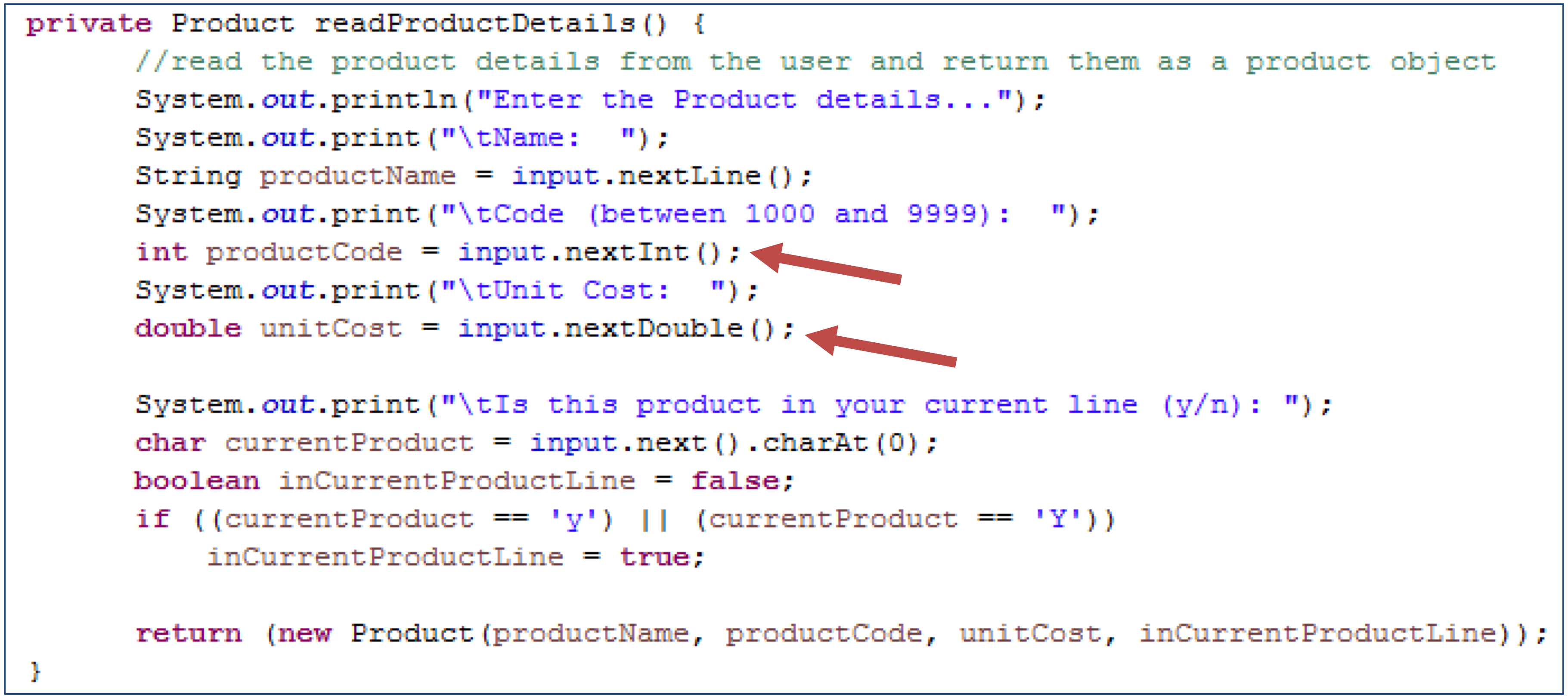
ShopV5.0 – making our app robust

What could cause a runtime exception here?



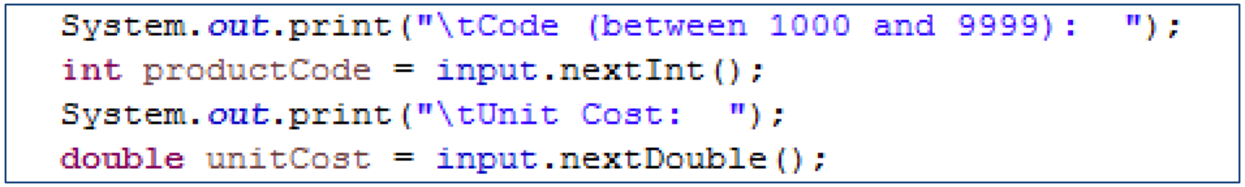
```java
private Product readProductDetails() {
      //read the product details from the user and return them as a product object
      System.out.println("Enter the Product details...");
      System.out.print("\tName:  ");
      String productName = input.nextLine();
      System.out.print("\tCode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

ShopV5.0 – making our app robust

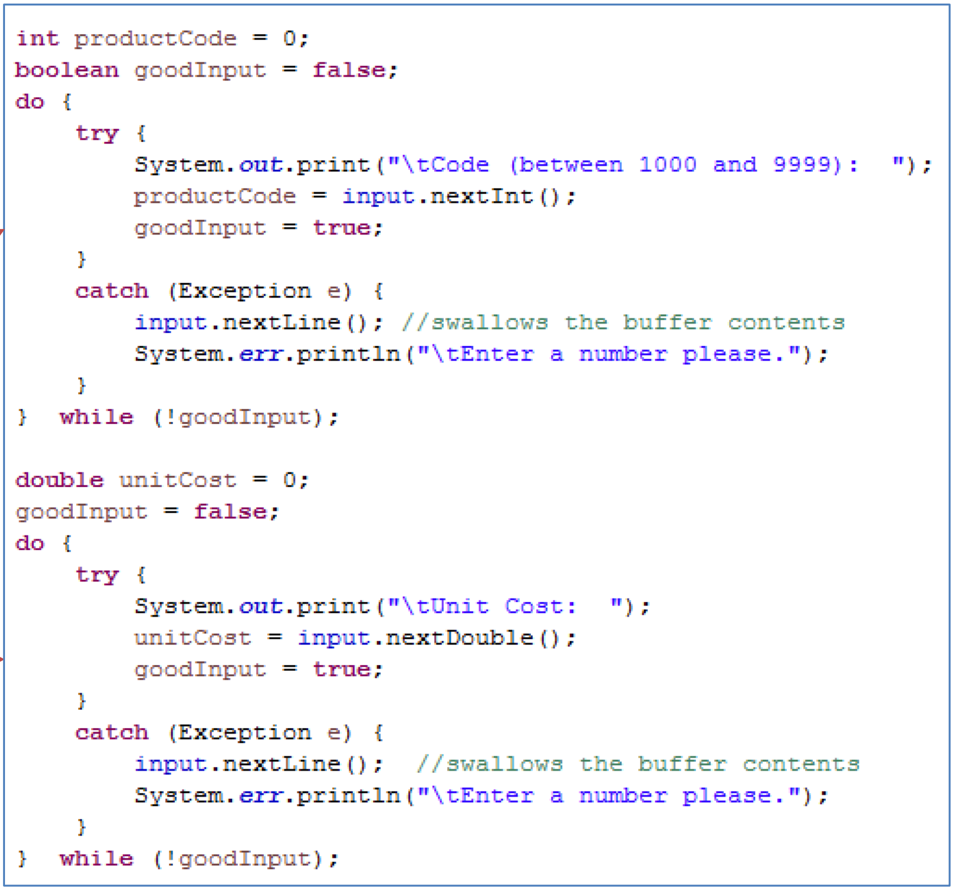


```java
private Product readProductDetails() {
      //read the product details from the user and return them as a product object
      System.out.println("Enter the Product details...");
      System.out.print("\tName:  ");
      String productName = input.nextLine();
      System.out.print("\tCode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

ShopV5.0 – making our app robust

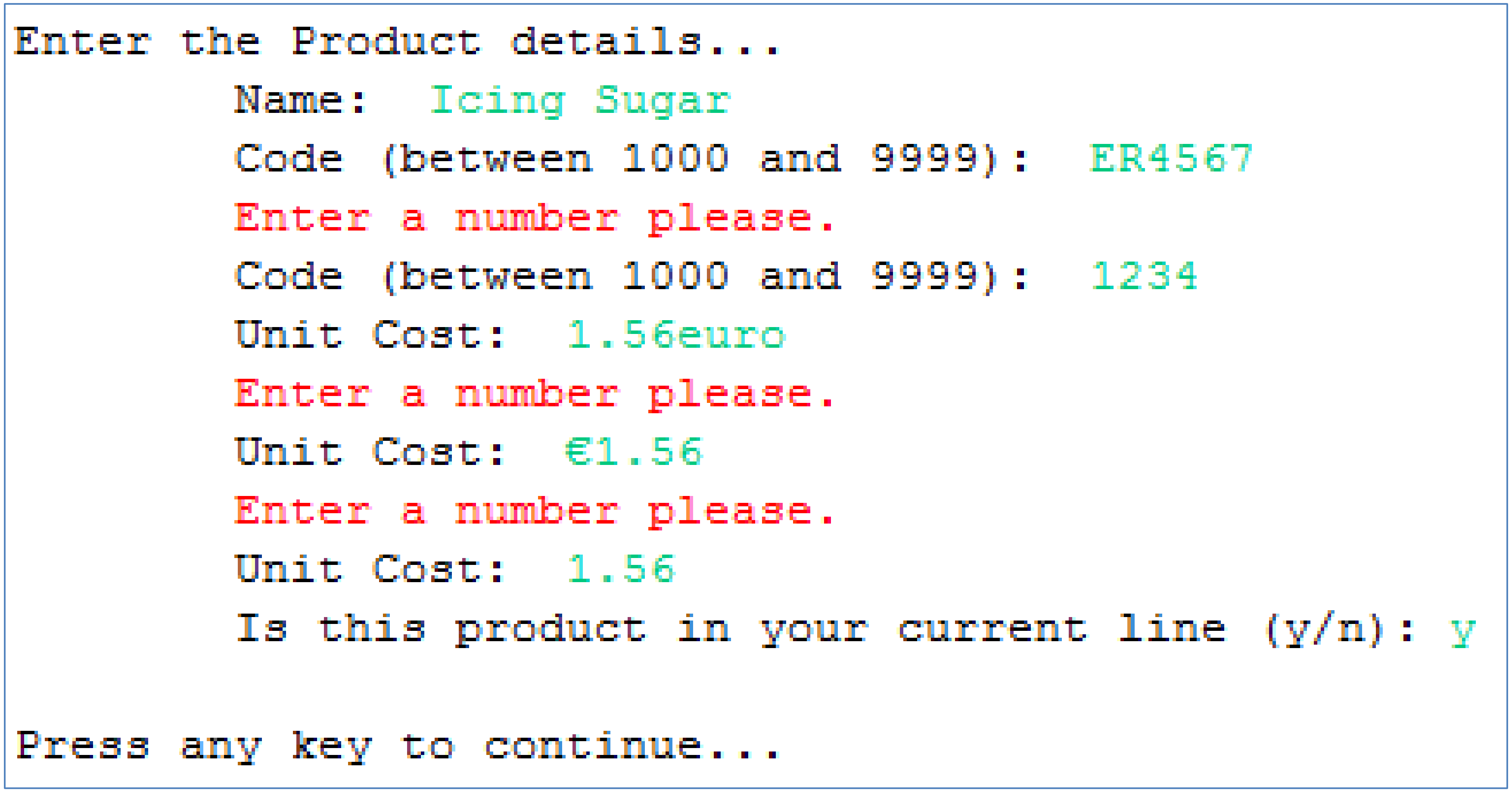


```
System.out.print("\tCode (between 1000 and 9999):  ");
int productCode = input.nextInt();
System.out.print("\tUnit Cost:  ");
double unitCost = input.nextDouble();
```



```
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!

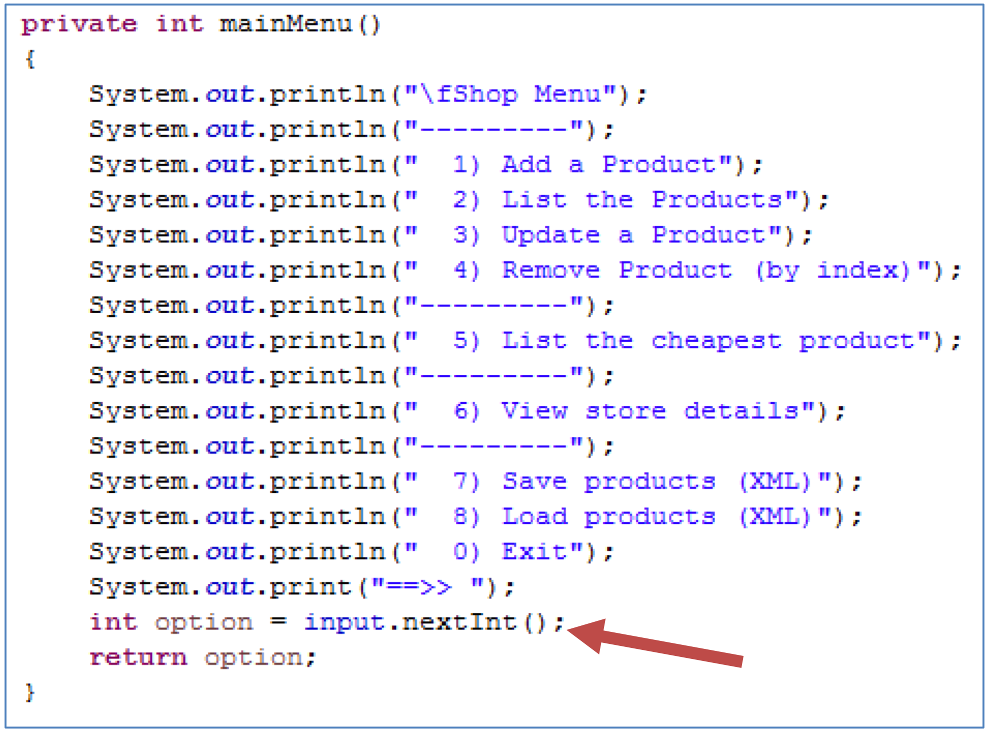


```
Enter the Product details...
        Name:  Icing Sugar
        Code (between 1000 and 9999):  ER4567
        Enter a number please.
        Code (between 1000 and 9999):  1234
        Unit Cost:  1.56euro
        Enter a number please.
        Unit Cost:  €1.56
        Enter a number please.
  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

Press any key to continue...
```

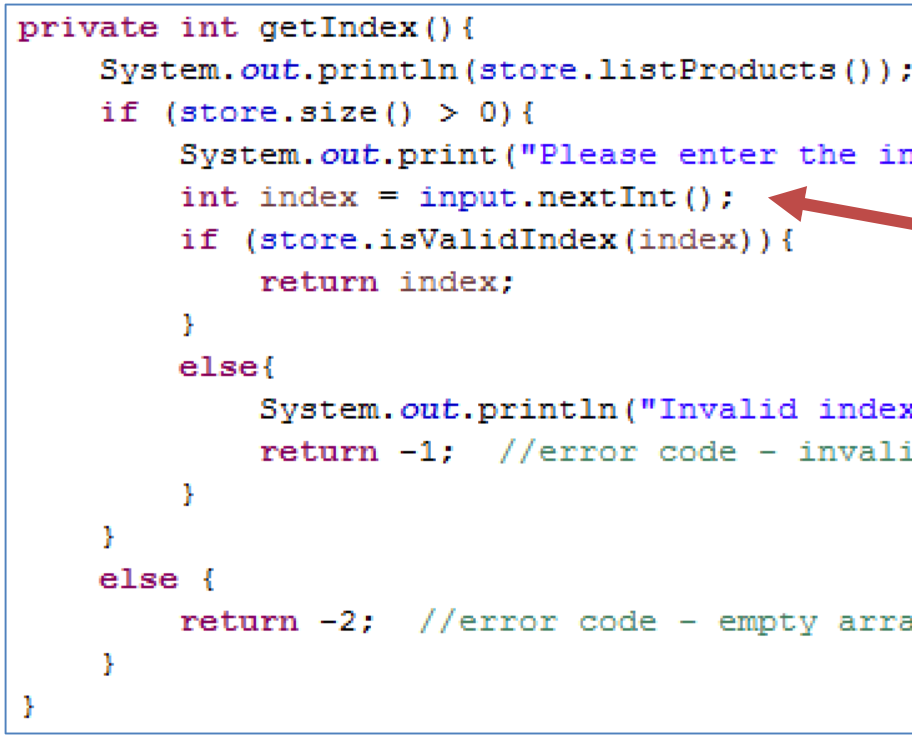
nextInt() and nextDouble() are now exception handled!

ShopV5.0 – 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;
}
```

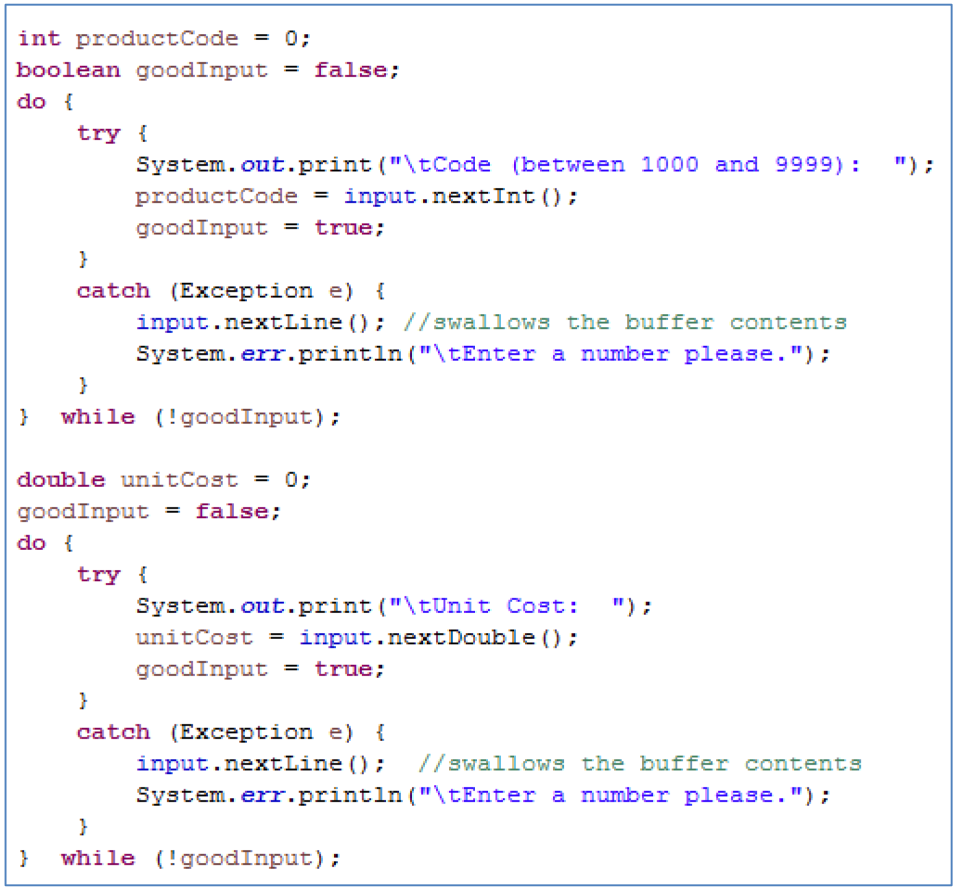


```
private int getIndex(){
    System.out.println(store.listProducts());
    if (store.size() > 0){
        System.out.print("Please enter the in
        int index = input.nextInt();
        if (store.isValidIndex(index)){
            return index;
        }
        else{
            System.out.println("Invalid 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?

ShopV5.0 – 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?

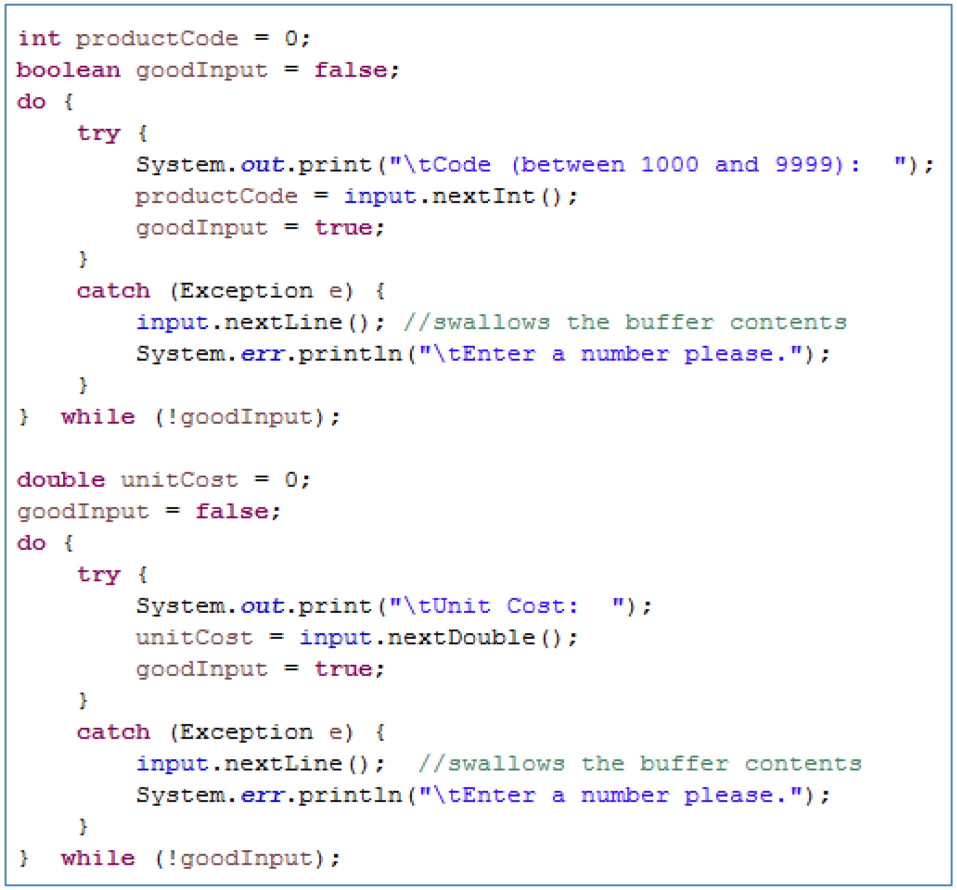


```
int productCode = 0;
boolean goodInput = false;
do {
    try {
        System.out.print("\tCode (between 1000 and 9999):  ");
        productCode = input.nextInt();
        goodInput = true;
    }
    catch (Exception e) {
        input.nextLine(); //swallows the buffer contents
        System.err.println("\tEnter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

ShopV5.0 – making our app robust

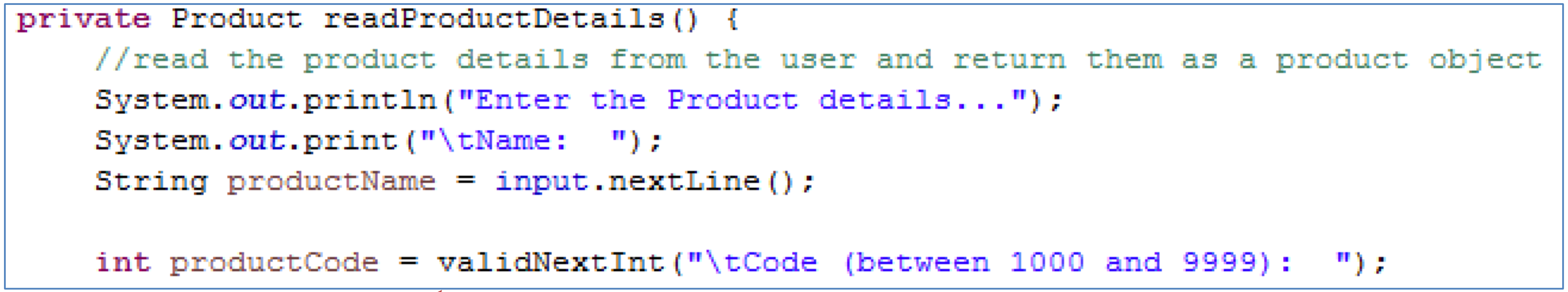
For this new method:

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

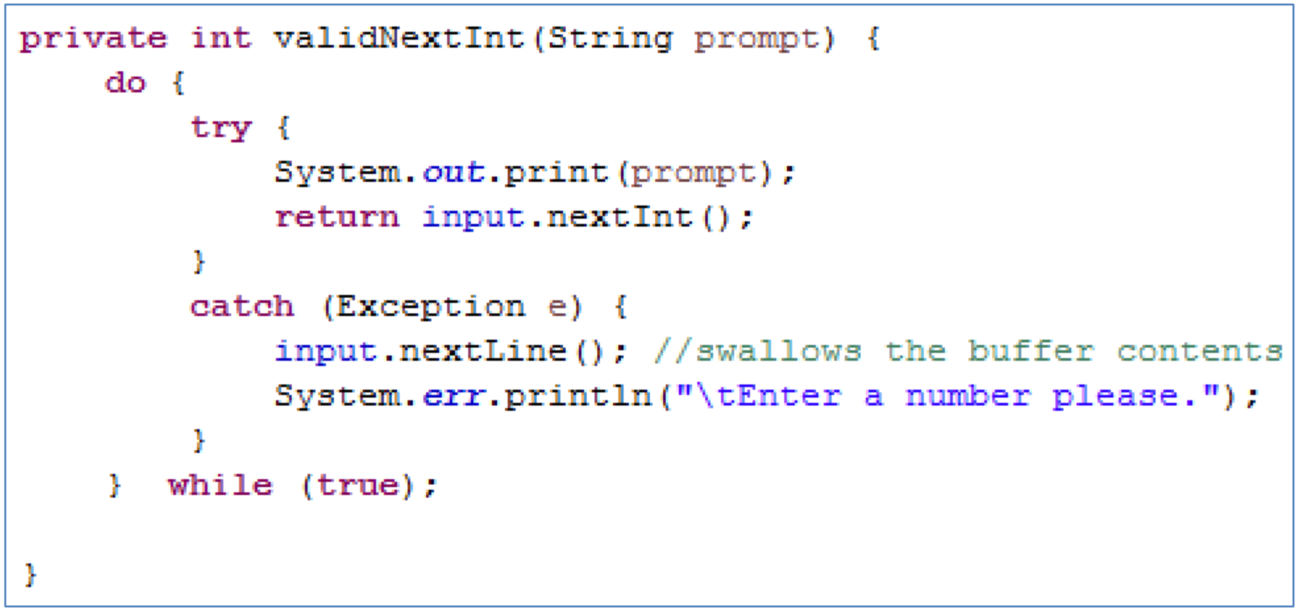


```
int productCode = 0;
boolean goodInput = false;
do {
    try {
        System.out.print("\tCode (between 1000 and 9999):  ");
        productCode = input.nextInt();
        goodInput = true;
    }
    catch (Exception e) {
        input.nextLine(); //swallows the buffer contents
        System.err.println("\tEnter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

ShopV5.0 – making our app robust

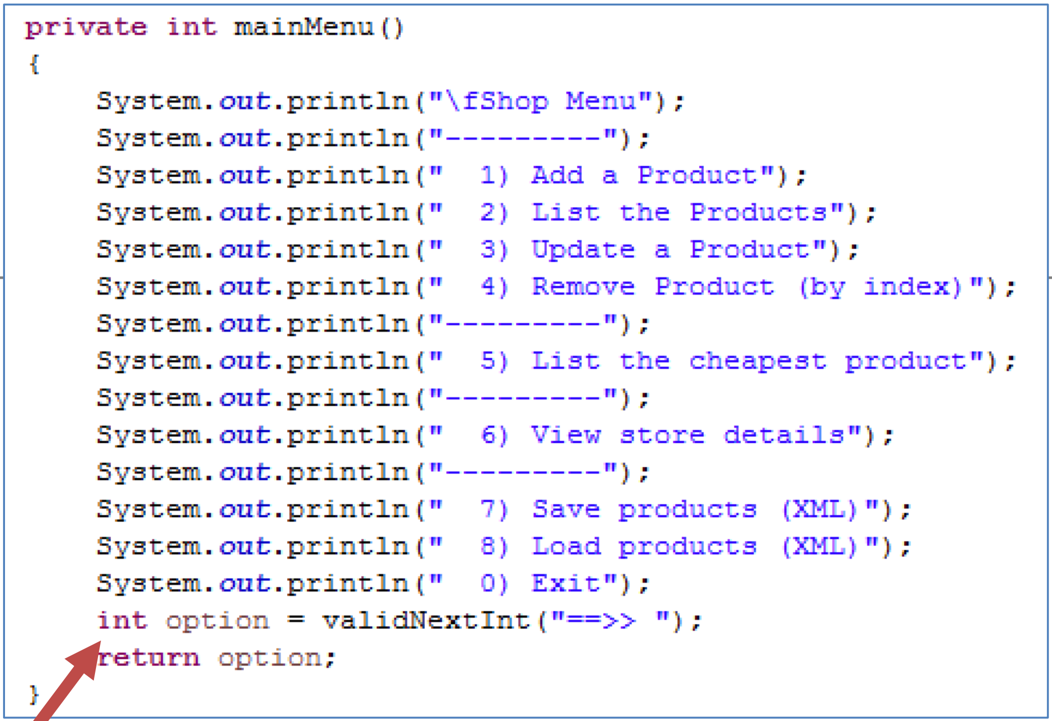


```java
private Product readProductDetails() {
    //read the product details from the user and return them as a product object
    System.out.println("Enter the Product details...");
    System.out.print("\tName:  ");
    String productName = input.nextLine();

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

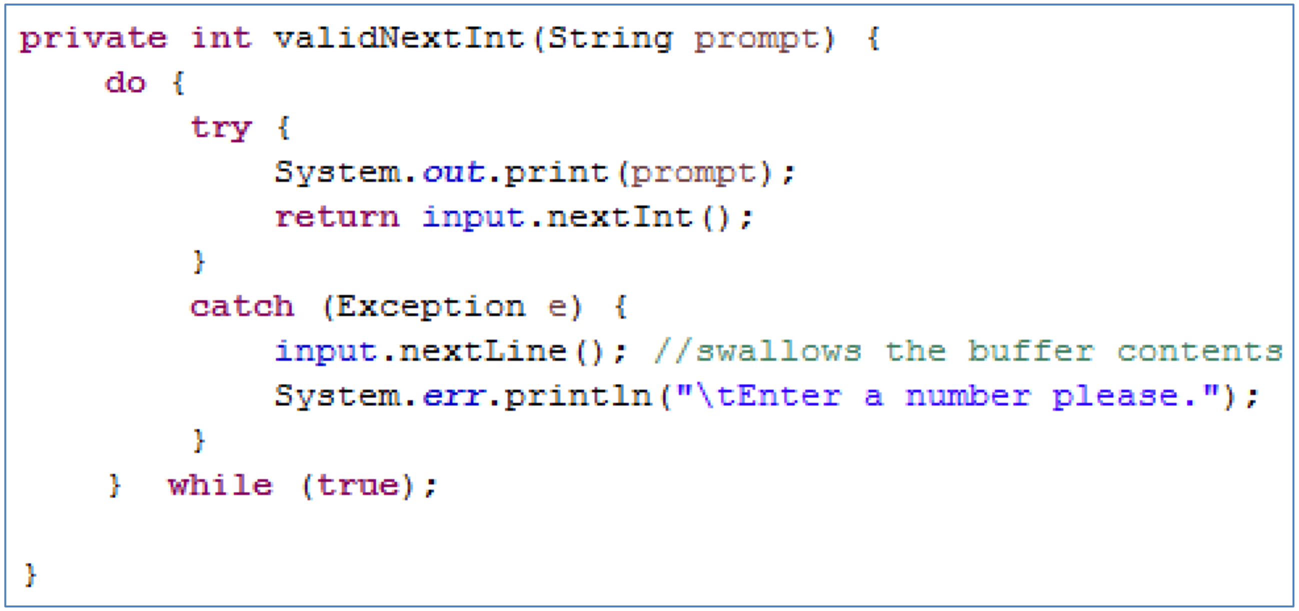


```java
private int validNextInt(String prompt) {
    do {
        try {
            System.out.print(prompt);
            return input.nextInt();
        }
        catch (Exception e) {
            input.nextLine(); //swallows the buffer contents
            System.err.println("\tEnter a number please.");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

}
```

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



```
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;
}
```



```
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**.

ShopV5.0 – 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();

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

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

```
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);

}
```

Any Questions?