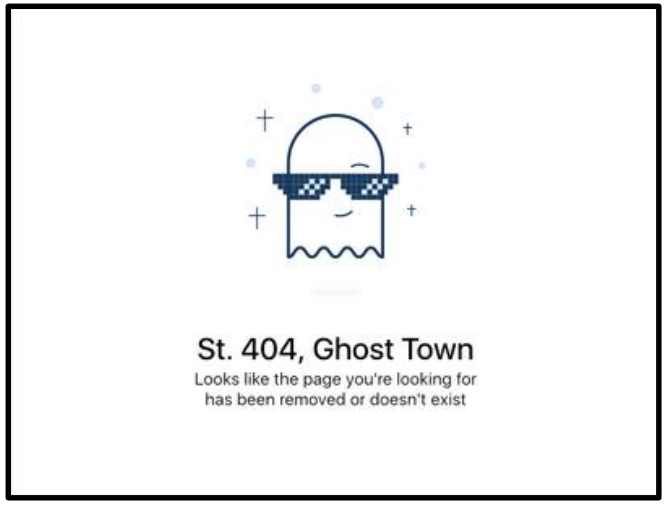
Exception Handling

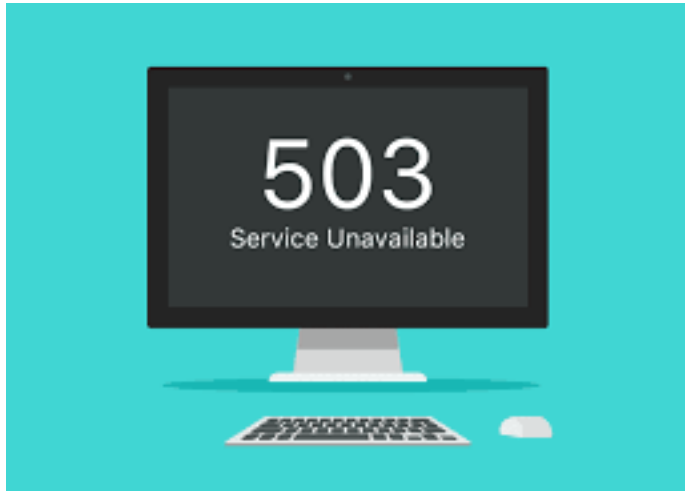
Handling invalid user input

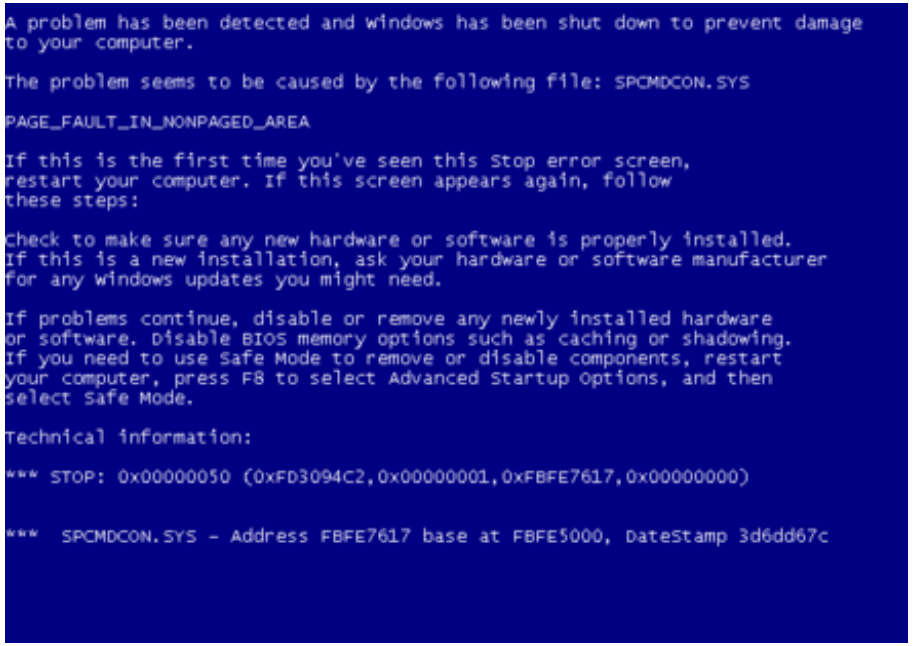
Produced by:
Dr. Siobhán Drohan
Ms. Mairéad Meagher
Ms Siobhan Roche

Ollscoil Teicneolaíochta an Oirdheiscirt
South East Technological University

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

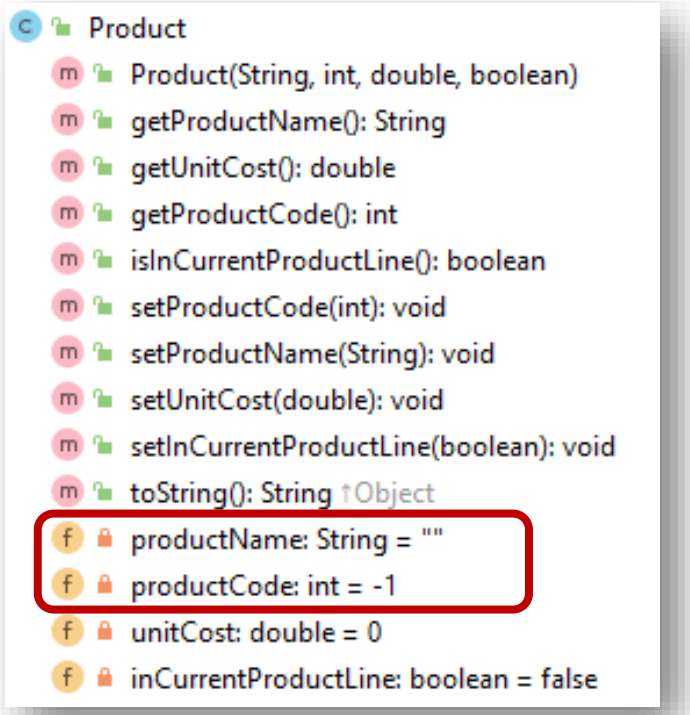

St. 404, Ghost Town
Looks like the page you're looking for
has been removed or doesn't exist

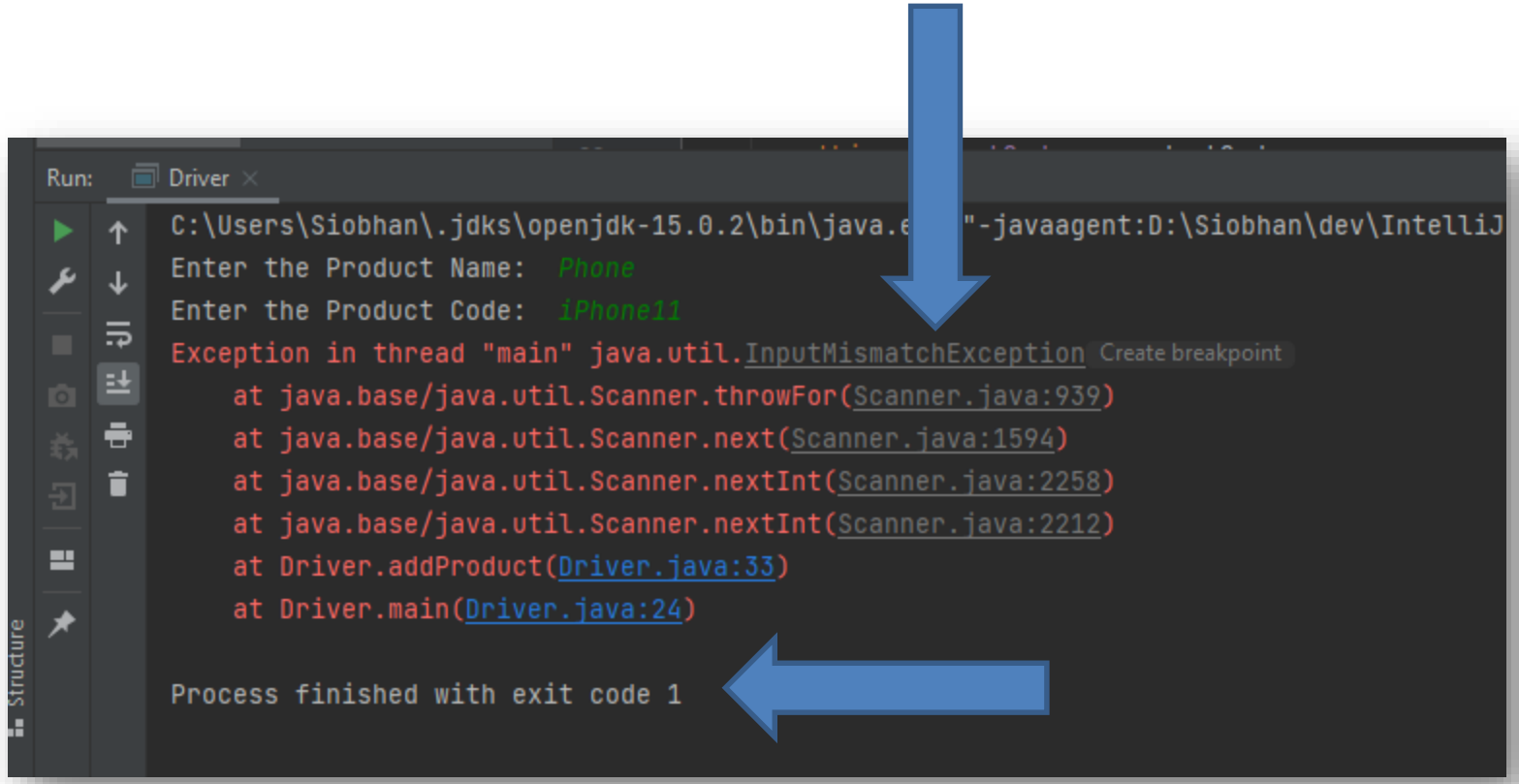

503
Service Unavailable


A problem has been detected and windows has been shut down to prevent damage
to your computer.
The problem seems to be caused by the following file: SPCMDCON.SYS
PAGE_FAULT_IN_NONPAGED_AREA
If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:
Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any windows updates you might need.
If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.
Technical information:
*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)
*** SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c

Lab01b, Step 3 – InputMismatch Exceptions

- Do you remember, in our first week, we crashed our ShopV1.0 project with an InputMismatchException:

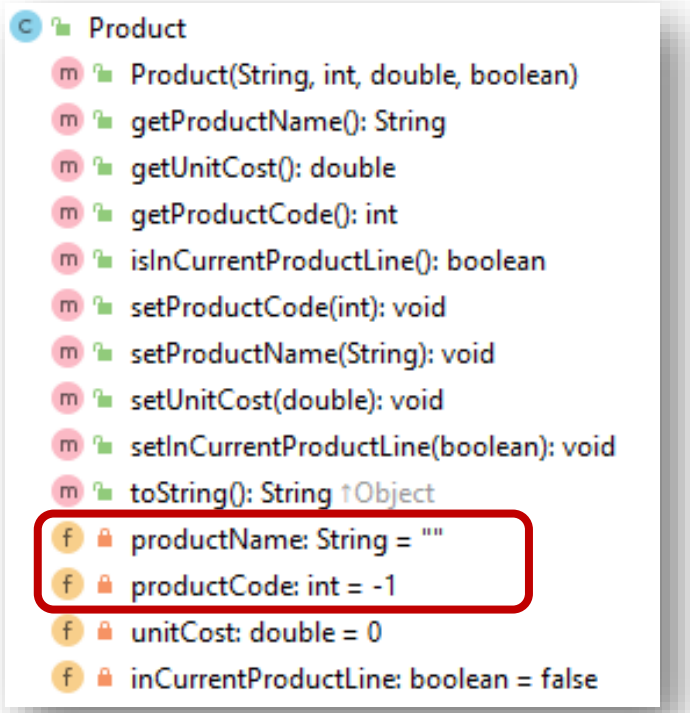




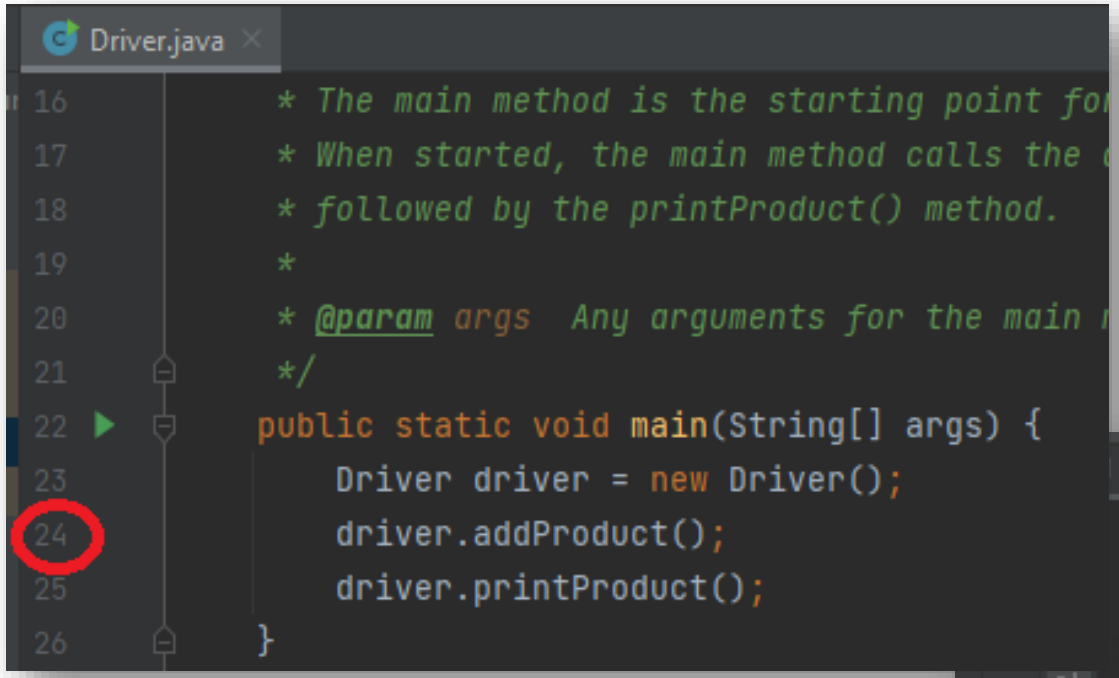
Lab01b, Step 3 – InputMismatch Exceptions

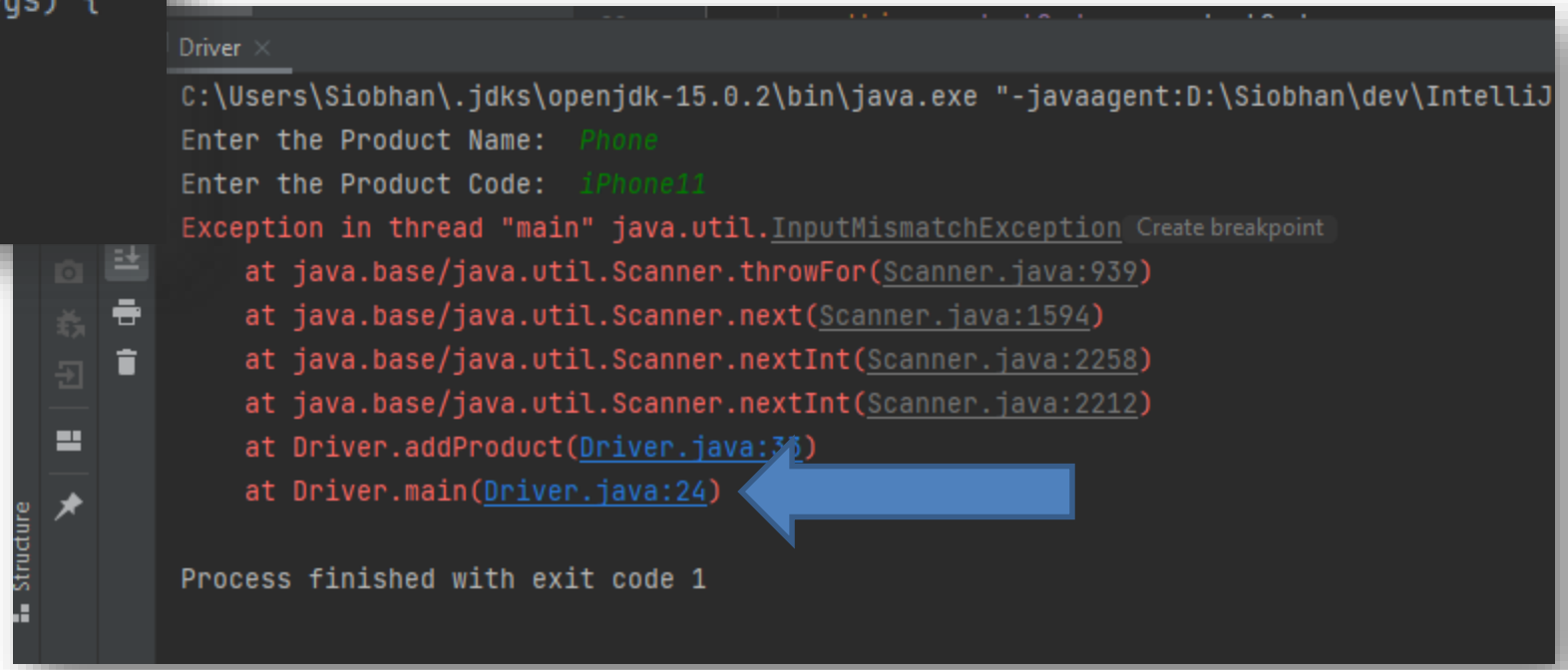
The program crashed because we were expecting the user to type in an **int** for *Product Code*, however they typed in a *String, iPhone11*.

We used a Scanner method **nextInt()** to read this input but as the input was a String, the program couldn't handle it, so it crashed.



Lab01b, Step 3 – InputMismatch Exceptions





Driver.java

```
25          driver.printProduct();
26      }
27
28      //gather the product data from the user and create a new product object.
29      private void addProduct(){
30          System.out.print("Enter the Product Name:  ");
31          String productName = input.nextLine();
32          System.out.print("Enter the Product Code:  ");
33          int productCode = input.nextInt();
34          System.out.print("Enter the Unit Cost:  ");
35          double unitCost = input.nextDouble();
36
37          //Ask the user to type in either a Y or an N.  This is then
38          //converted to either a True or a False (i.e. a boolean value).
39          System.out.print("Is this product in your current line (y/n): ");
40          char currentProduct = input.next().charAt(0);
41          boolean inCurrentProductLine = false;
42          if ((currentProduct == 'y') || (currentProduct == 'Y'))
43              inCurrentProductLine = true;
44
45          product = new Product(productName, productCode, unitCost, inCurrentProductLine);
46      }
```

```
jdk-15.0.2\bin\java.exe "-javaagent:D:\Siobhan\dev\IntelliJ
one
hone11
java.util.InputMismatchException Create breakpoint
Scanner.throwFor(Scanner.java:939)
    at java.base/java.util.Scanner.next(Scanner.java:1594)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
    at Driver.addProduct(Driver.java:33)
    at Driver.main(Driver.java:24)

Process finished with exit code 1
```

Structure

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**.

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.

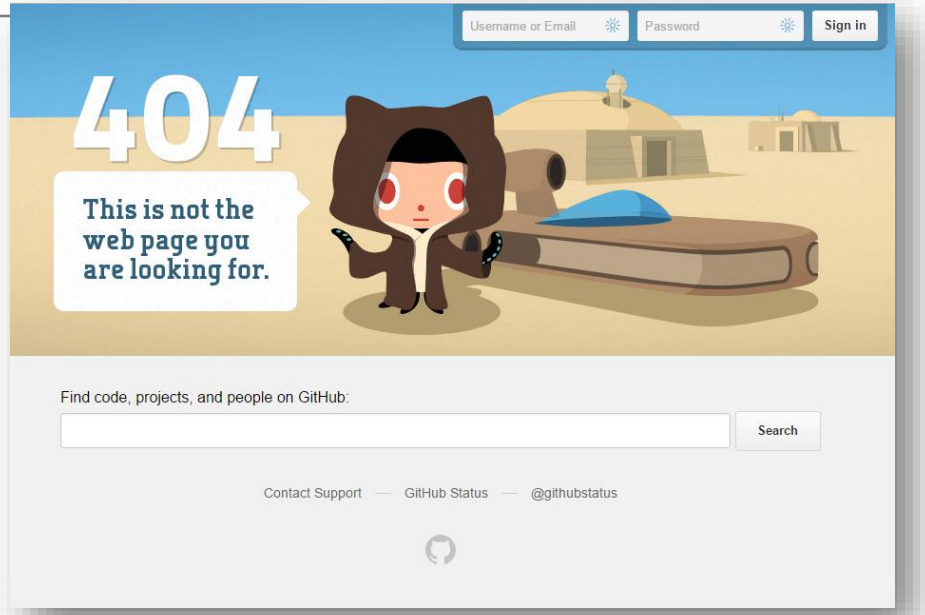
A RuntimeException…

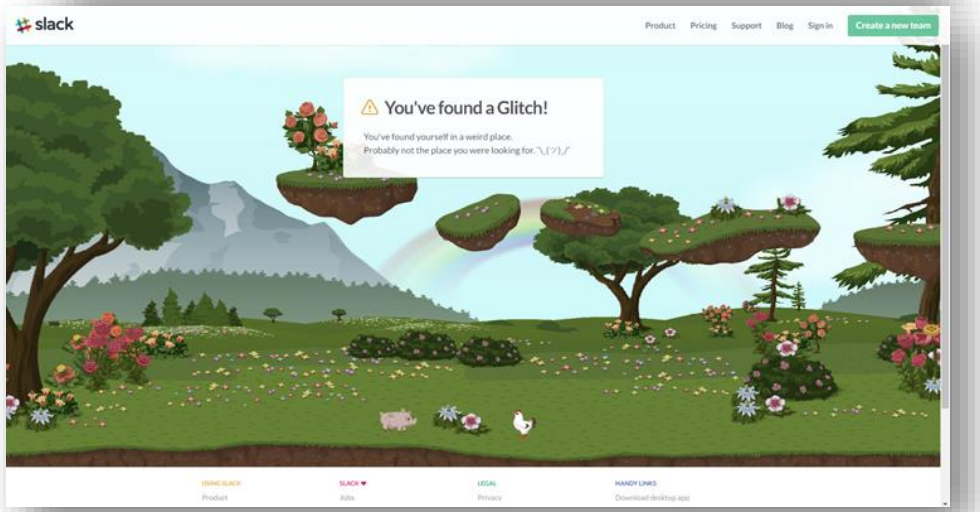
- is a subclass of the Exception **class**.
- encompasses all exceptions which can ordinarily happen at run-time.
- these exceptions can be thrown by any java statement or a method call.
- can be avoided through good programming practices!

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.
InputMismatchException	Caused by passing invalid input (wrong type).

Catching Exceptions - **handlers**

- **Catch**ing 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
}
```

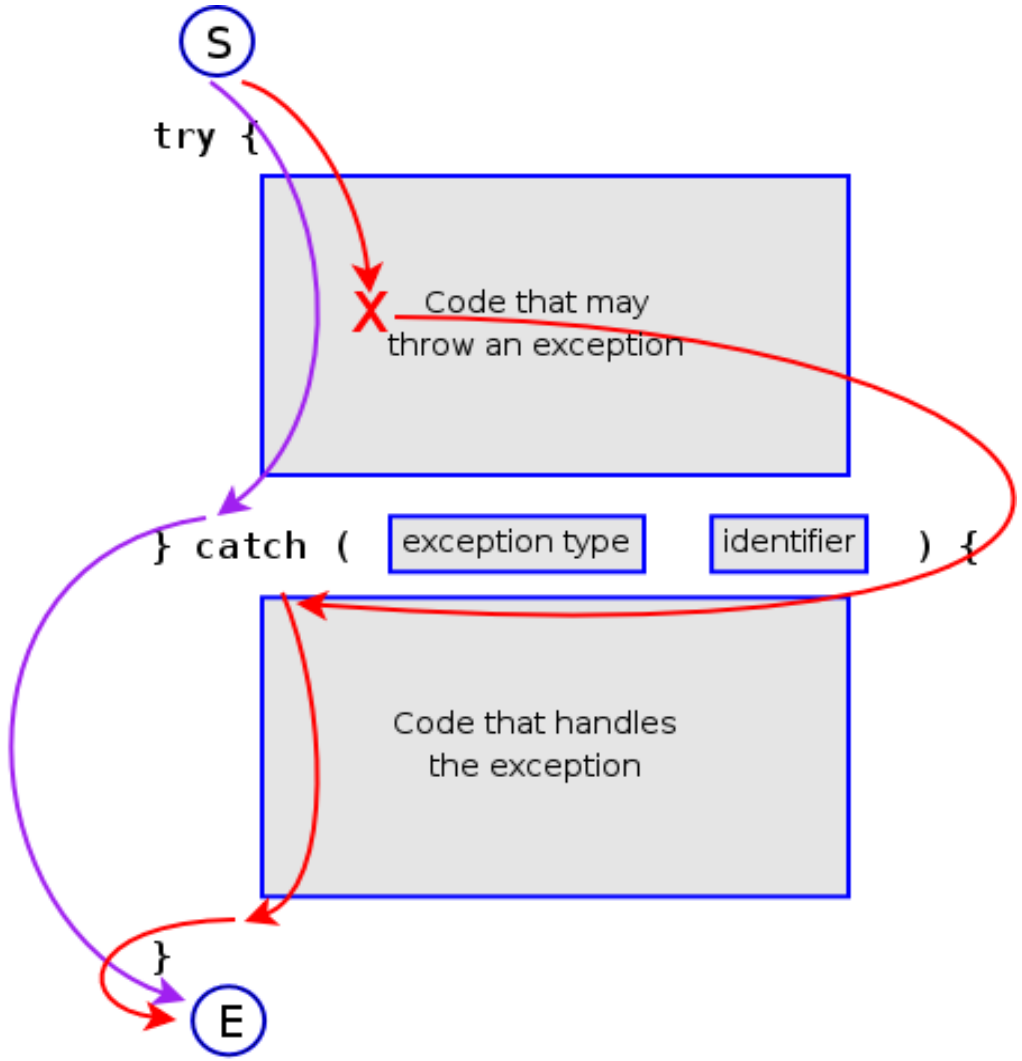
try and catch - example

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

The parameter ***e*** is of type **Exception.**

We can use ***e*** to print out
what exception occurred.

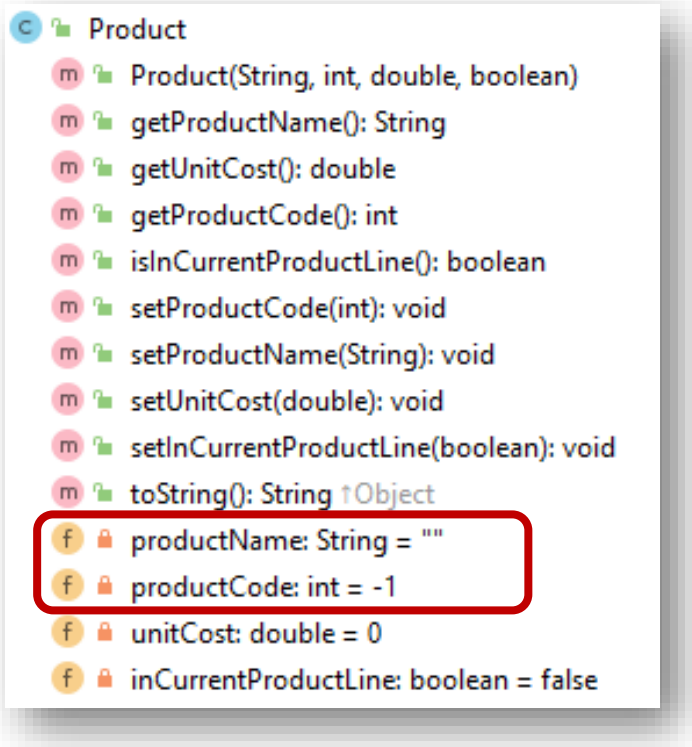
Flow of control in Exception Handing

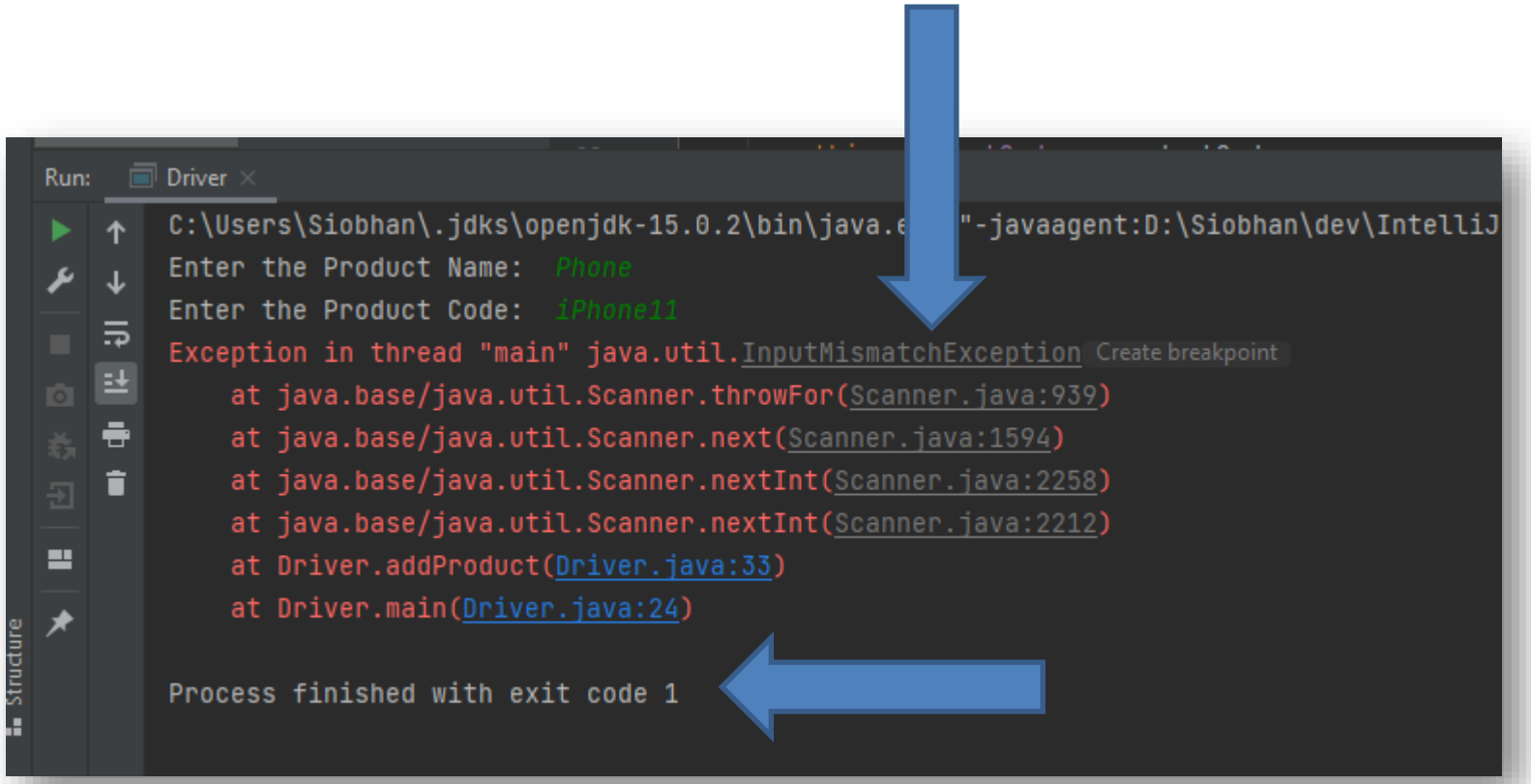


S = Start
E = End

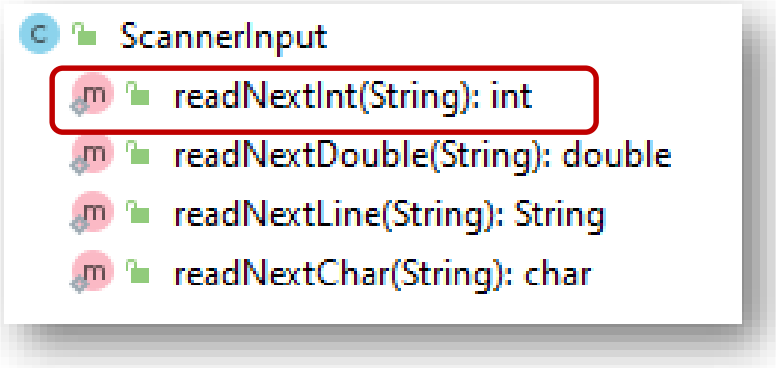
Purple path = everything ok
Red path = exception is caught and handled

Returning to our – InputMismatch Exception





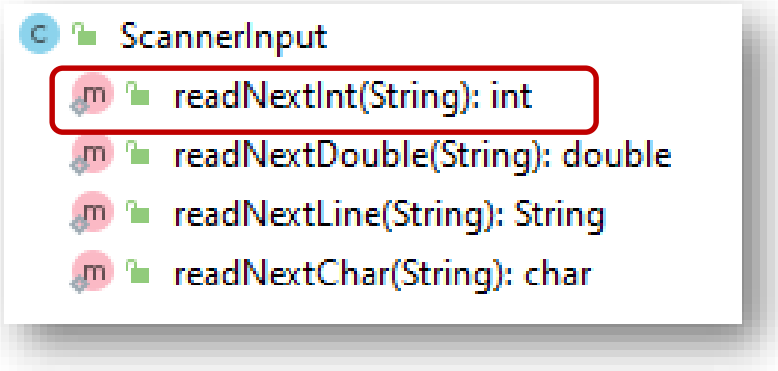
Lab03: ScannerInput Class is Introduced



The ScannerInput class:

- has four methods for reading four different data types.
- creates a new Scanner object in each method, thereby getting rid of the Scanner bug (where the buffers are not flushed properly after each int/double/float read).
- Uses the try/catch construct to handle any InputMismatchExceptions and to stop them crashing our program (i.e. recover from them).
- is a “utility” class that we can reuse in all our projects.

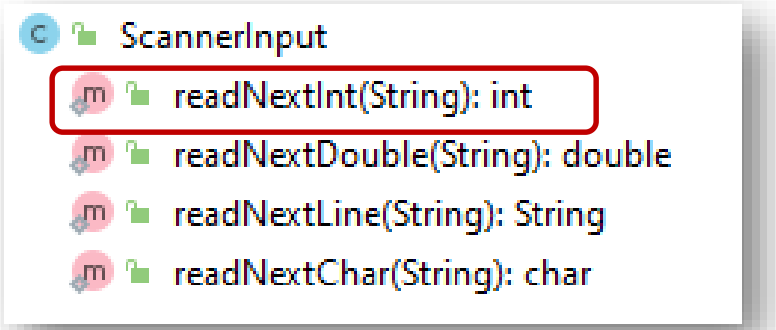
Lab03: ScannerInput Class is Introduced



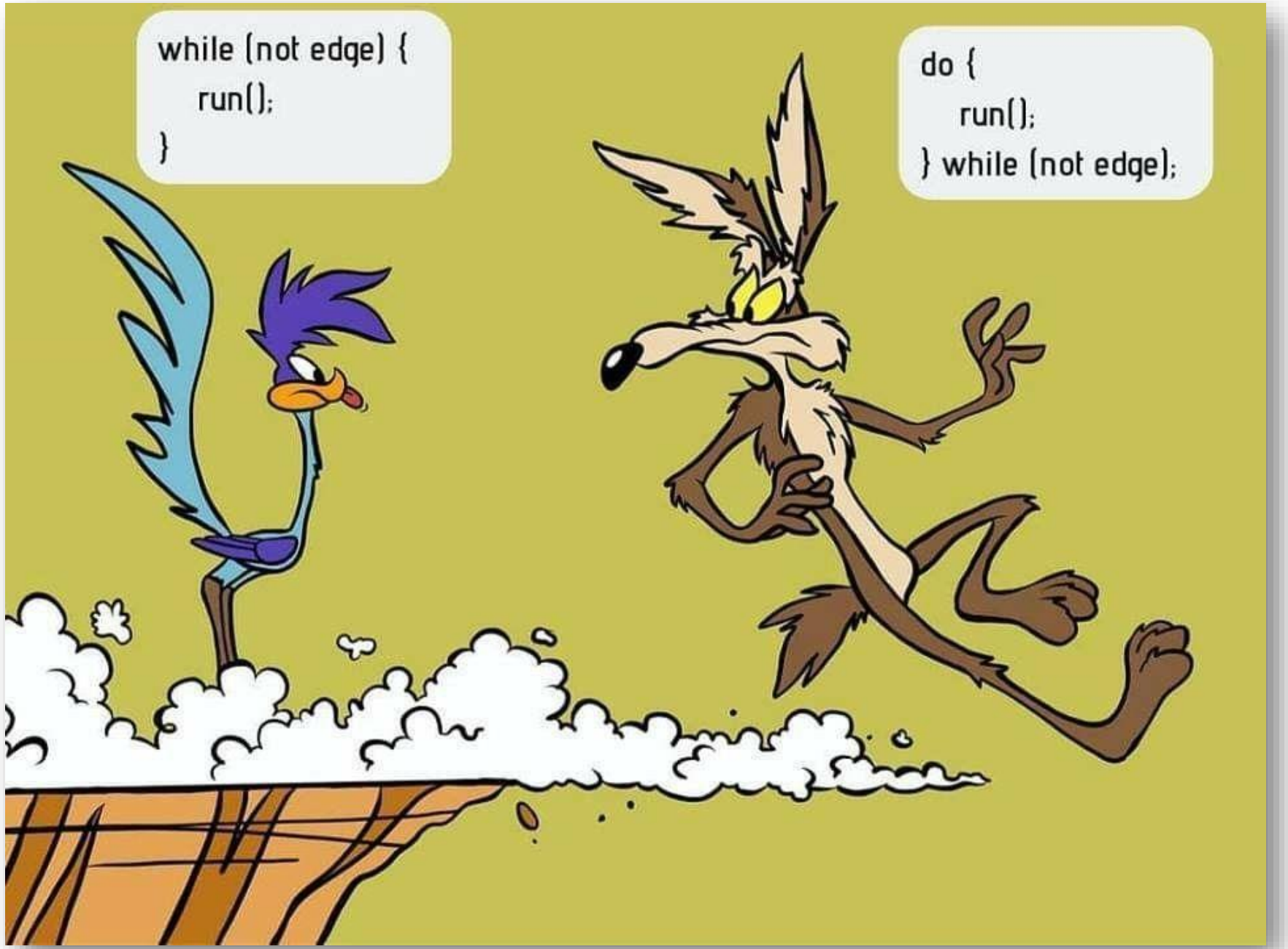
readNextInt:

- This method takes in a String prompt and prints it to the console (e.g. of a prompt would be *"Please enter the product code"*).
- The user input is then read and it is checked to see if what the user entered is ACTUALLY an **int**.
- If the value:
  - is an **int**, the value is returned.
  - isn't an **int**, then the prompt is printed again to the user and the validation of the input continues until the user enters a valid **int**.

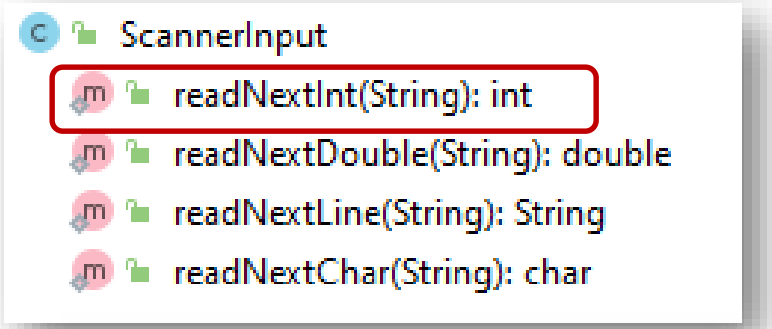
Lab03: ScannerInput Class is Introduced

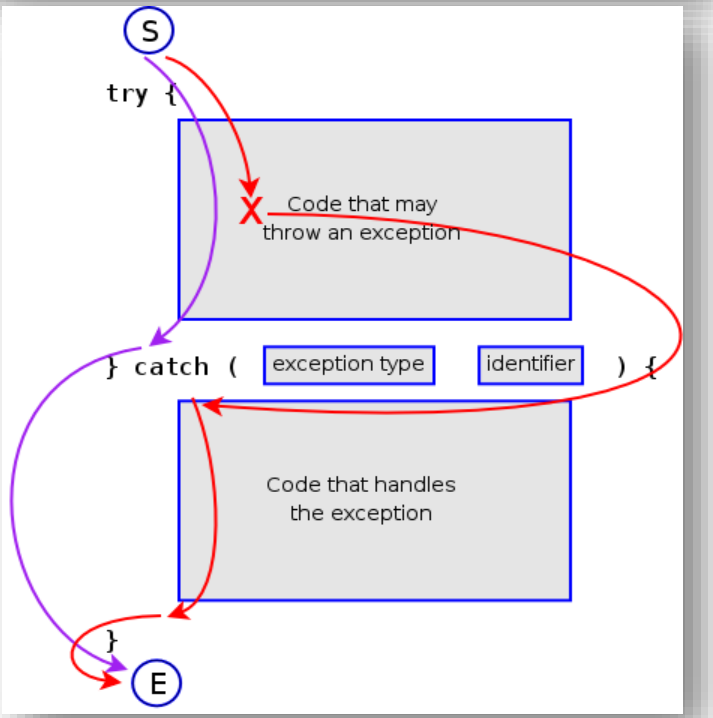



```java
public static int readNextInt(String prompt) {
    do {
        var scanner = new Scanner(System.in);
        try {
            System.out.print(prompt);
            return Integer.parseInt(scanner.next());
        }
        catch (NumberFormatException e) {
            System.err.println("\tEnter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


while (not edge) {
run();
}
do {
run();
} while (not edge);

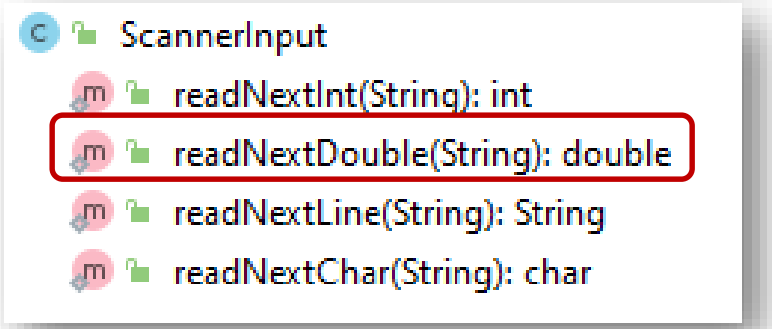
Lab03: ScannerInput Class is Introduced






```java
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);
}
```

Lab03: ScannerInput Class is Introduced

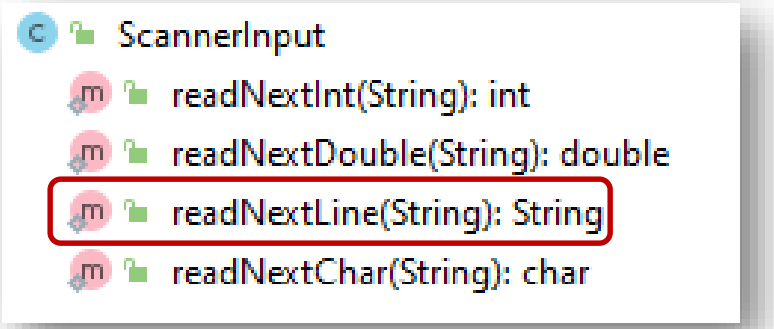


readNextDouble:

This method behaves the same as the **int** one above, except that it is validating for a **double** value.

```
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);
}
```

Lab03: ScannerInput Class is Introduced

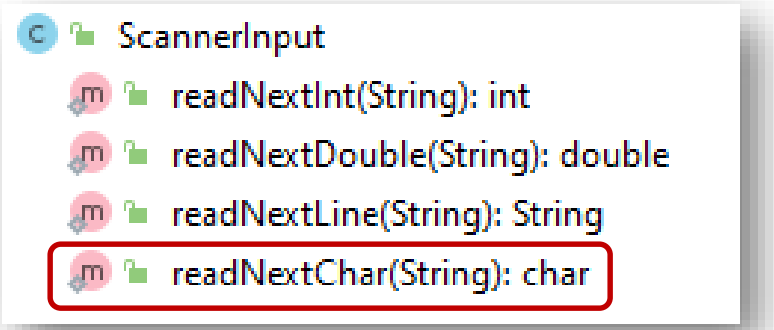


readNextLine:

- This method takes in a String prompt and prints it to the console.
- The full user input is then read using the Scanner **nextLine()** method.
- The **String** value is returned with no validation.

```
public static String readNextLine(String prompt) {
    Scanner input = new Scanner(System.in);
    System.out.print(prompt);
    return input.nextLine();
}
```

Lab03: ScannerInput Class is Introduced



readNextChar:

- This method takes in a String prompt and prints it to the console.
- The first **char** the user entered is then read using the Scanner **next().charAt(0)** method.
- The **char** is returned with no validation.

```
public static char readNextChar(String prompt) {
    Scanner input = new Scanner(System.in);
    System.out.print(prompt);
    return input.next().charAt(0);
}
```

Using ScannerInput Utility in our Driver

Now that we know what the ScannerInput can recover from InputMismatchExceptions and also gets rid of the Scanner bug, we will now look at using it in our Driver class.

Remove the old Scanner object

The first thing you can do is delete the following line of code from Driver:

```
private Scanner input = new Scanner(System.in);
```

We no longer want to use the Scanner class like this,
we want to use our ScannerInput class instead.

Deleting this line of code will cause a LOT of syntax errors!

Using ScannerInput in a method

Let's start with a straight forward fix.

The current code in the `printProductsAboveAPrice` method is:

```
private void printProductsAboveAPrice(){
    System.out.print("View the products costing more than this price:  ");
    double price = input.nextDouble();
    System.out.println(store.listProductsAboveAPrice(price));
}
```

We will now replace the first two lines of code in the method with one line. Our method will now look like this:

```
private void printProductsAboveAPrice(){
    double price = ScannerInput.readNextDouble("View the products costing more than this price:  ");
    System.out.println(store.listProductsAboveAPrice(price));
}
```

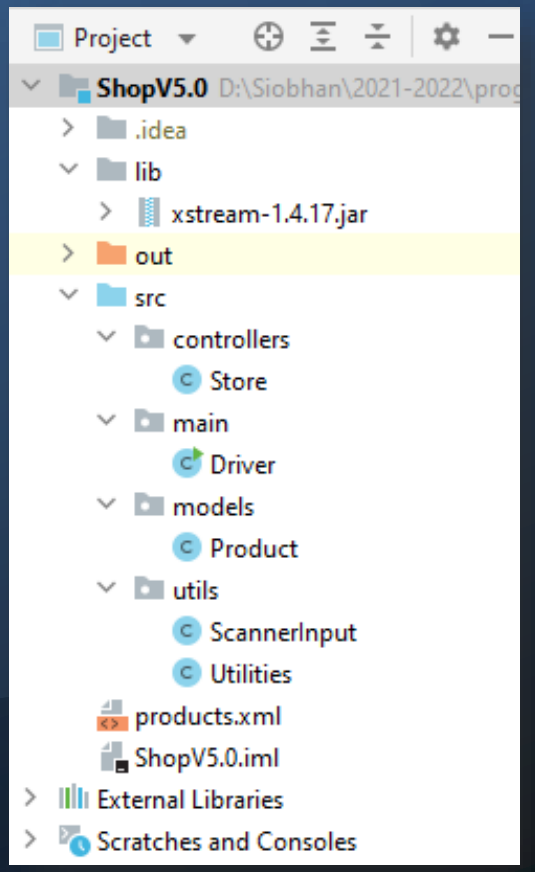
Using ScannerInput in add Product()

```
private void addProduct(){

    String productName = ScannerInput.readNextLine("Enter the Product Name:  ");
    int productCode = ScannerInput.readNextInt("Enter the Product Code:  ");
    double unitCost = ScannerInput.readNextDouble("Enter the Unit Cost:  ");

    //Ask the user to type in either a Y or an N.  This is then
    //converted to either a True or a False (i.e. a boolean value).
    char currentProduct = ScannerInput.readNextChar("Is this product in your current line (y/n): ");
    boolean inCurrentProductLine = false;
    if ((currentProduct == 'y') || (currentProduct == 'Y'))
        inCurrentProductLine = true;

    boolean isAdded = store.add(new Product(productName, productCode, unitCost, inCurrentProductLine));
    if (isAdded){
        System.out.println("Product Added Successfully");
    }
    else{
        System.out.println("No Product Added");
    }
}
```

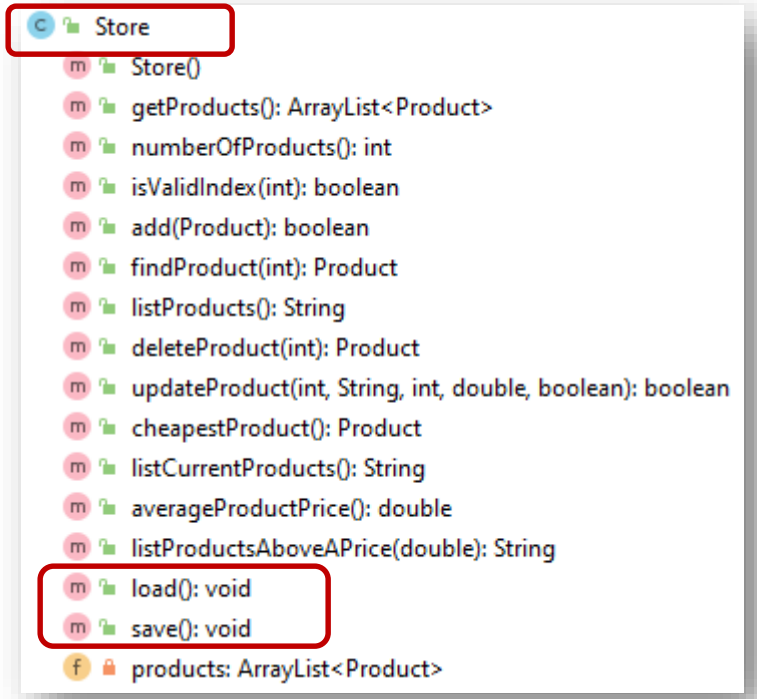
Using ScannerInput in mainMenu()

```
private int mainMenu(){
    int option = ScannerInput.readNextInt("""
            Shop Menu
            ---------
               1) Add a product
               2) List the Products
               ----------------------------
               3) List the current products
               4) Display average product unit cost
               5) Display cheapest product
               6) List products that are more expensive than a given price
               ----------------------------
               0) Exit
            ==>>  """);
    return option;
}
```

XStream and Exceptions

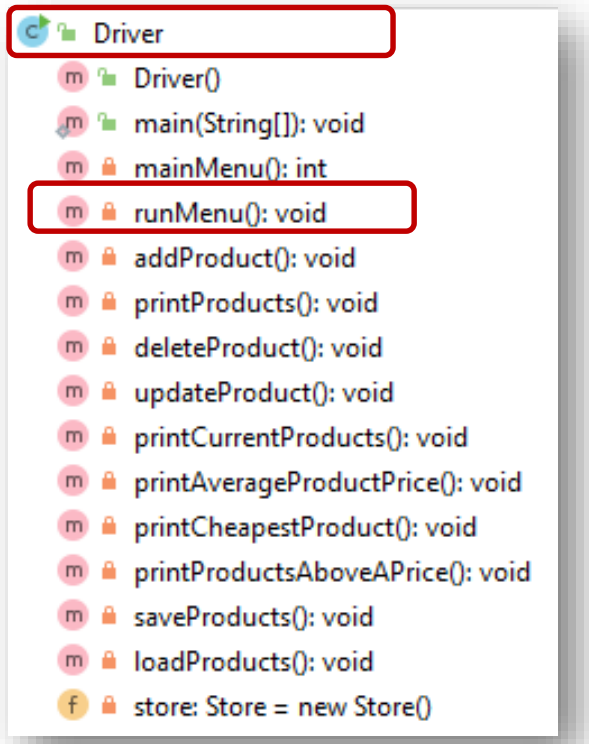


In the last slide deck, we covered using the XStream component to save and load XML data to/from the harddisk. Now we will look in more detail at the Exception Handling.



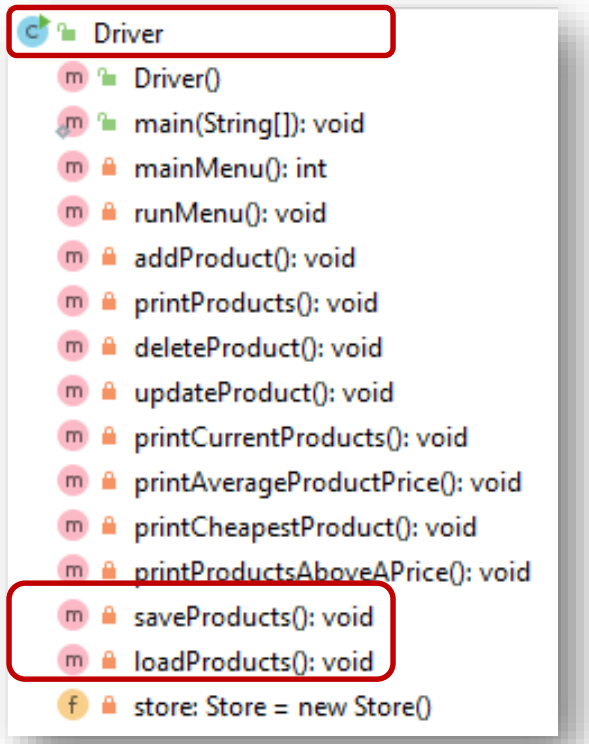

```
@SuppressWarnings("unchecked")
public void load() throws Exception {
    //list of classes that you wish to include in the serialisation, separated by a
comma
    Class<?>[] classes = new Class[] { Product.class };

    //setting up the xstream object with default security and the above classes
    XStream xstream = new XStream(new DomDriver());
    XStream.setupDefaultSecurity(xstream);
    xstream.allowTypes(classes);

    //doing the actual serialisation to an XML file
    ObjectInputStream is = xstream.createObjectInputStream(new
FileReader("products.xml"));
    products = (ArrayList<Product>) is.readObject();
    is.close();
}
```

```
public void save() throws Exception {
    XStream xstream = new XStream(new DomDriver());
    ObjectOutputStream out =
                    xstream.createObjectOutputStream(new FileWriter("products.xml"));
    out.writeObject(products);
    out.close();
}
```



Adding **Save** and **Load** functionality to the **menu**.

```
switch (option) {
    case 1 -> addProduct();
    case 2 -> printProducts();
    case 3 -> updateProduct();
    case 4 -> deleteProduct();
    case 5 -> printCurrentProducts();
    case 6 -> printAverageProductPrice();
    case 7 -> printCheapestProduct();
    case 8 -> printProductsAboveAPrice();
    case 9 -> saveProducts();
    case 10 -> loadProducts();
    default -> System.out.println("Invalid option entered: " + option);
}
```



Adding **Save** and **Load** functionality to the **menu**.

```
//save all the products in the store to a file on the hard disk
private void saveProducts() {
    try {
        store.save();
    } catch (Exception e) {
        System.err.println("Error writing to file: " + e);
    }
}
```

```
//load all the products into the store from a file on the hard disk
private void loadProducts() {
    try {
        store.load();
    } catch (Exception e) {
        System.err.println("Error reading from file: " + e);
    }
}
```

Summary

- Crash v Exceptions
- Detect and Handle
  - Enables program to continue
- Java's predefined Exception objects
- try / catch block
- Introduction to
  - do while loop
    - Always runs once
    - Condition is test at the end

Any Questions?