

Handling User Input

Packages, Utilities, Parsing & Wrappers

Produced Dr. Siobhán Drohan
by:



Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

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

Recap of making
ShopV5.0 robust

ShopV5.0 – 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));
}
```

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();
    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 index: ");
        int index = input.nextInt();
        if (store.isValidIndex(index)) {
            return index;
        }
        else {
            System.out.println("Invalid index");
            return -1; //error code - invalid index
        }
    }
    else {
        return -2; //error code - empty array
    }
}
```

- 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

```
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.

```
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);
}
```

```
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;
}
```

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

```
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);
}
```

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);
}
```

Using packages and utilities

Developing ShopV6.0

ShopV5.0

- MenuController 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("\nEnter a number please.");
        }
    } while (true);
}
```

```
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("\nEnter a decimal number please.");
        }
    } while (true);
}
```

ShopV5.0

- MenuController 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);
}
```

```
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);
}
```

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

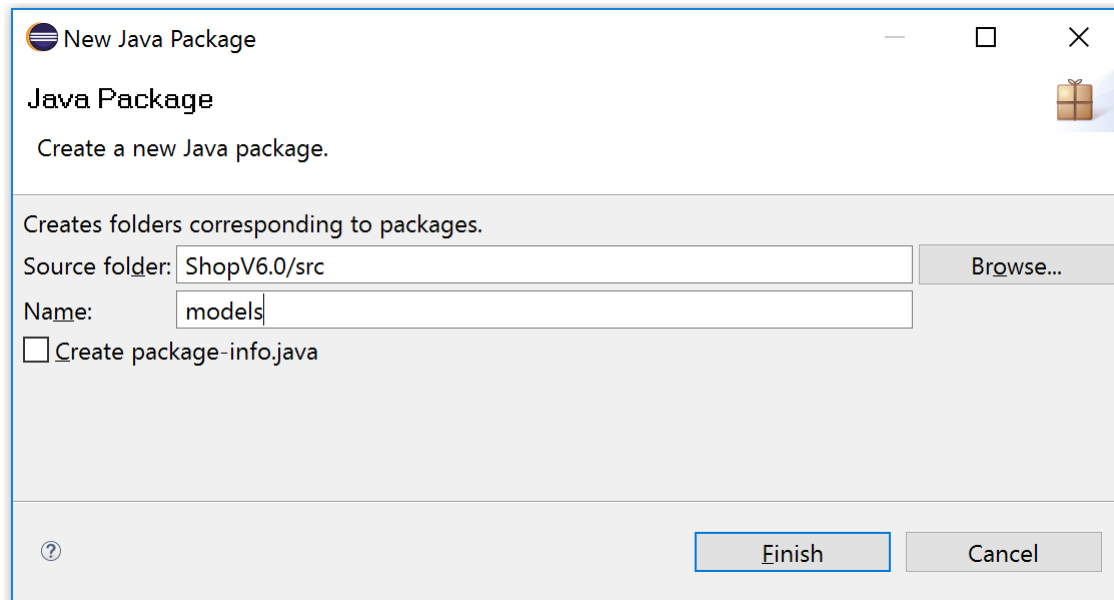
ShopV6.0 – utilities and packages



- In the next few slides, we will remove these methods from the MenuController class and put them into a separate “utility” class.
- As our app is getting larger, we will start using “packages” to structure our app.

ShopV6.0 – utilities and packages

- Create a new app called ShopV6.0.
- Right-click on the **src** folder and select New → Package. Enter “models” as the package name.



- Create two more packages: “controllers” and “utils”.

ShopV6.0 – utilities and packages

- ShopV5.0 [ict-programming-2017 master]
 - src
 - (default package)
 - MenuController.java
 - Product.java
 - Store.java
 - JRE System Library [JavaSE-1.8]
 - Referenced Libraries
 - xstream-1.4.8.jar
 - lib
 - xstream-1.4.8.jar

Copy the ShopV5.0 files into the ShopV6.0 project to the locations specified in the screen shot below.

- > ShopV6.0 [ict-programming-2017 master]
 - src
 - controllers
 - MenuController.java
 - Store.java
 - models
 - Product.java
 - utils
 - JRE System Library [JavaSE-1.8]
 - Referenced Libraries
 - xstream-1.4.8.jar
 - lib
 - xstream-1.4.8.jar

When we have copied all the existing code to this new format, you can see we have errors!

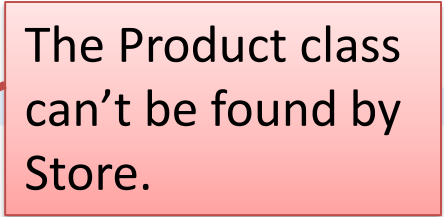
ShopV6.0 – utilities and packages

```
*MenuController.java  *Store.java
119
120 //gather details to update the selected product with
121 Product productDetails = readProductDetails();
122
123 //retrieve the product and update it using the details entered i
124 Product productToUpdate = store.get(index);
125 productToUpdate.setInCurrentProductLine (productDetails.isInCurre
productToUpdate.setProductCode (productDetails.getProductCode ());
productToUpdate.setProductName (productDetails.getProductName ());
productToUpdate.setUnitCost (productDetails.getUnitCost ());
    }
}
131
132 //=====
133 // HELPER / UTILITY METHODS
134 //=====
135
136 private Product readProductDetails() {
137     //read the product details from the user and return them as a produc
138     System.out.println("Enter the Product details...");
139     System.out.print("\tName:  ");
140     String productName = input.nextLine();
141
142     int productCode = validNextInt("\tCode (between 1000 and 9999):  ");
143     double unitCost = validNextDouble("\tUnit Cost:  ");
144
```

The Product class can't be found by MenuController.

ShopV6.0 – utilities and packages

```
*MenuController.java *Store.java
1 package controllers;
2
3 import java.io.FileReader;
11
12 /**
13  * A Store class which was specifically developed to demonstrate
14  * the use of an ArrayList of Product.
15  *
16  * @author Siobhan Drohan
17  * @version 1.0
18  */
19 public class Store
20 {
21     private String storeLocation;
22     private ArrayList<Product> products;
23
24     /**
25      * Constructor for objects of class Store
26      * @param storeLocation The city where the specific store is located
27      */
28     public Store(String istoreLocation)
29     {
30         storeLocation = istoreLocation;
31         products = new ArrayList<Product>();
32     }
33
```



The Product class can't be found by Store.

ShopV6.0 – utilities and packages

```
*MenuController.java *Store.java
119
120 //gather details to update the selected product with
121 Product productDetails = readProductDetails();
122
123 //retrieve the product and update it using the details entered i
124 Product productToUpdate = store.get(index);
125 productToUpdate.setInCurrentProductLine(productDetails.isInCurre
126 productToUpdate.setProductCode(productDetails.getProductCode());
127 productToUpdate.setProductName(productDetails.getProductName());
128 productToUpdate.setUnitCost(productDetails.getUnitCost());
129 }
130 }
131
132 //=====
133 // HELPER / UTILITY METHODS
134 //=====
135
136 private Product readProductDetails() {
137 //re
138 Syst
139 Syst
140 Stri
141
142 int
143 doub
144
```

Product cannot be resolved to a type

7 quick fixes available:

- Import 'Product' (models)
- Create class 'Product'
- Create interface 'Product'
- Create enum 'Product'
- Add type parameter 'Product' to 'MenuController'
- Add type parameter 'Product' to 'readProductDetails()'
- Fix project setup...

Problems Javadoc
31 errors, 0 warnings, 0 o

import 'Product' (models)
in both classes.

ShopV6.0 – utilities and packages

```

v ShopV6.0 [ict-programming-2017 master]
v src
  v controllers
    > MenuController.java
    > Store.java
  v models
    > Product.java
  # utils
  > JRE System Library [JavaSE-1.8]
v Referenced Libraries
  > xstream-1.4.8.jar
v lib
  xstream-1.4.8.jar

```

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

ShopV6.0 – utilities and packages

Creating our first utility class...

- In the **utils** package, create a new class called **ScannerInput**.
- Cut the `validNextDouble` and `validNextInt` methods from `MenuController` and paste them into `ScannerInput`.
- Change the accessor modifier for these methods from `private` to `public`. Make each method `static`.
- Add a local `Scanner` object for each method and import the `Scanner` class.

Creating our
first utility
class...

```
MenuController.java  Store.java  *ScannerInput.java ✖
1 package utils;
2
3 import java.util.Scanner;
4
5 public class ScannerInput {
6
7     @SuppressWarnings("resource")
8     public static int validNextInt(String prompt) {
9         Scanner input = new Scanner(System.in);
10        do {
11            try {
12                System.out.print(prompt);
13                return input.nextInt();
14            }
15            catch (Exception e) {
16                input.nextLine(); //swallows the buffer contents
17                System.err.println("\tEnter a number please.");
18            }
19        } while (true);
20    }
21
22    @SuppressWarnings("resource")
23    public static double validNextDouble(String prompt) {
24        Scanner input = new Scanner(System.in);
25        do {
26            try {
27                System.out.print(prompt);
28                return input.nextDouble();
29            }
30            catch (Exception e) {
31                input.nextLine(); //swallows the buffer contents
32                System.err.println("\tEnter a decimal number please.");
33            }
34        } while (true);
35    }
36 }
```


ShopV6.0 – utilities and packages

Calling the methods in our first utility class...

MenuController can't find our new methods...

```
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("Code (between 1000 and 9999): ");
    double unitCost = validNextDouble("Unit Cost: ");

    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));
}
```

ShopV6.0 – utilities and packages

Calling the methods in our first utility class...

```
import static utils.ScannerInput.*;
```

```
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: ");  
  
    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));  
}
```

ShopV6.0 – utilities and packages

- 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!

Wrappers and Parsing

Another approach for validating
input in ShopV6.0

Another approach to validating input

- Currently, our validation of **int** input is as follows:

```
@SuppressWarnings("resource")
public static int validNextInt(String prompt) {
    Scanner input = new Scanner(System.in);
    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);
}
```

Another approach to validating input

- We can use wrapper classes and parsing for validating input:

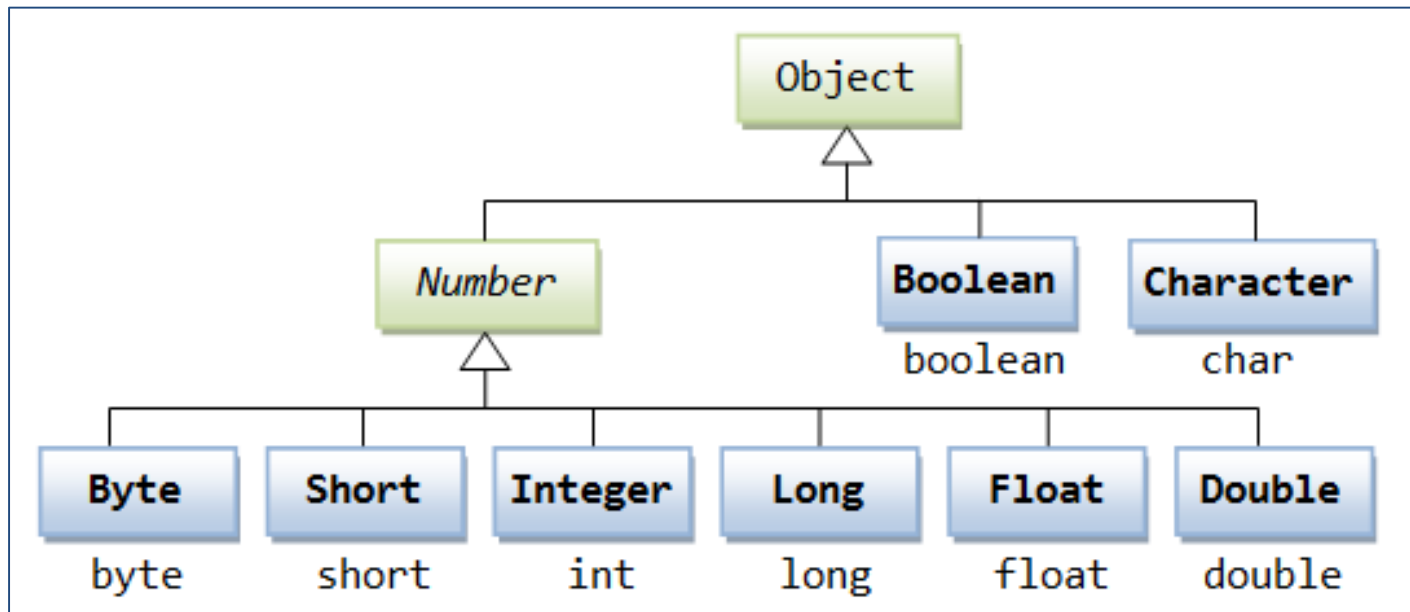
```
public class ScannerInput {  
  
    @SuppressWarnings("resource")  
    public static int validNextInt(String prompt) {  
        Scanner input = new Scanner(System.in);  
        do {  
            try {  
                System.out.print(prompt);  
                return Integer.parseInt(input.next());  
            }  
            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**.

Wrapper classes

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



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.

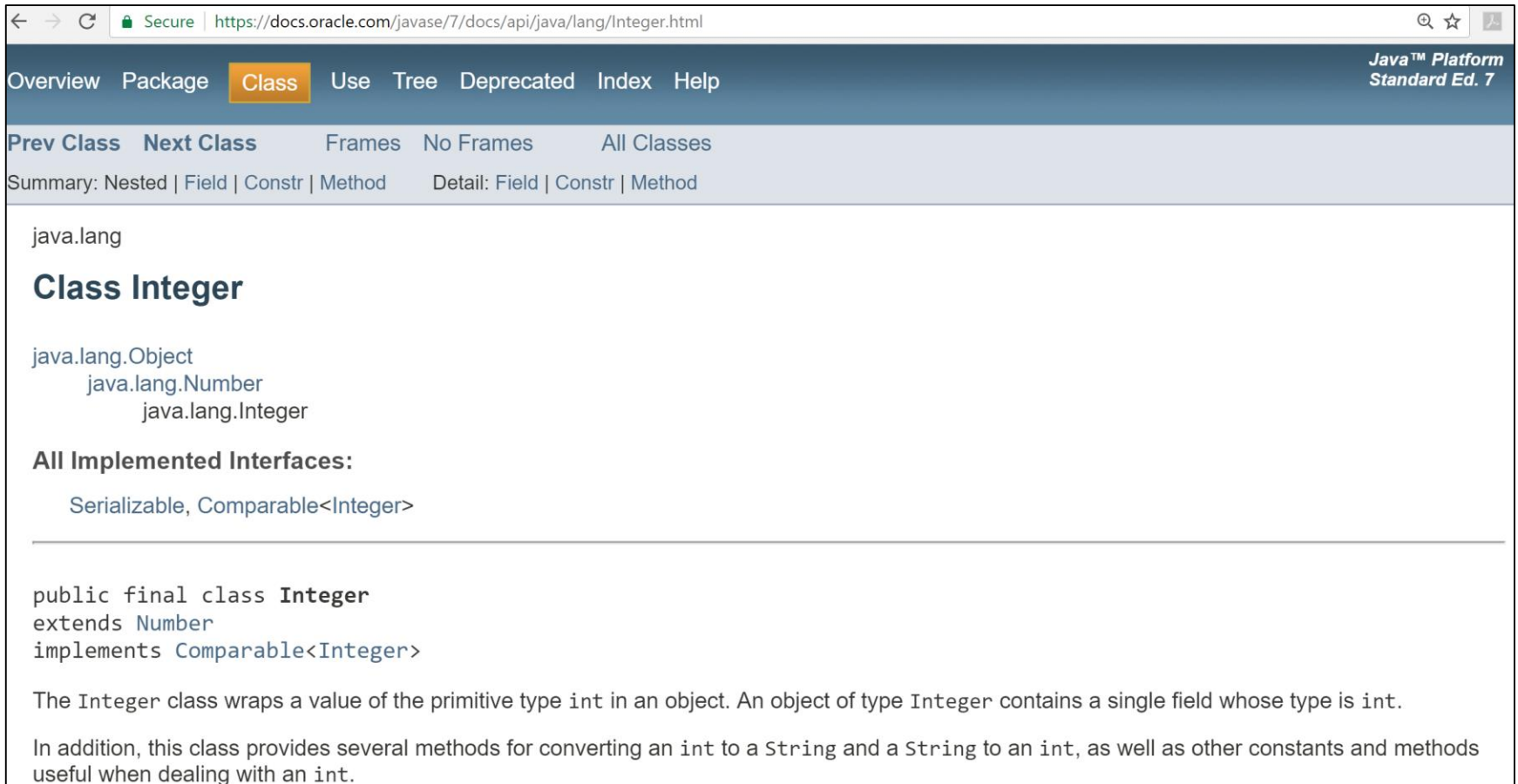
Wrapper classes

- 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.

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  
    }  
}
```

Parsing



The screenshot shows the Java Platform Standard Ed. 7 API documentation for the `Integer` class. The page is titled "Class Integer" and is part of the `java.lang` package. The navigation bar includes "Overview", "Package", "Class", "Use", "Tree", "Deprecated", "Index", and "Help". The "Class" tab is selected. The page content includes the package name `java.lang`, the class name `Class Integer`, the inheritance hierarchy (`java.lang.Object`, `java.lang.Number`, `java.lang.Integer`), and the implemented interfaces (`Serializable`, `Comparable<Integer>`). The source code for the class is shown as `public final class Integer extends Number implements Comparable<Integer>`. The description states that the `Integer` class wraps a value of the primitive type `int` in an object and provides methods for converting between `int` and `String`.

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.

**Any
Questions?**





Except where otherwise noted, this content is licensed under a Creative Commons Attribution-NonCommercial 3.0 License.

For more information, please see <http://creativecommons.org/licenses/by-nc/3.0/>



Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

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