

Lesson 05 Demo 01

Implementing Wrapper Classes

Objective: To implement Java wrapper classes to represent primitive data types as objects, enabling utility methods and use in collections

Tools Required: Eclipse IDE

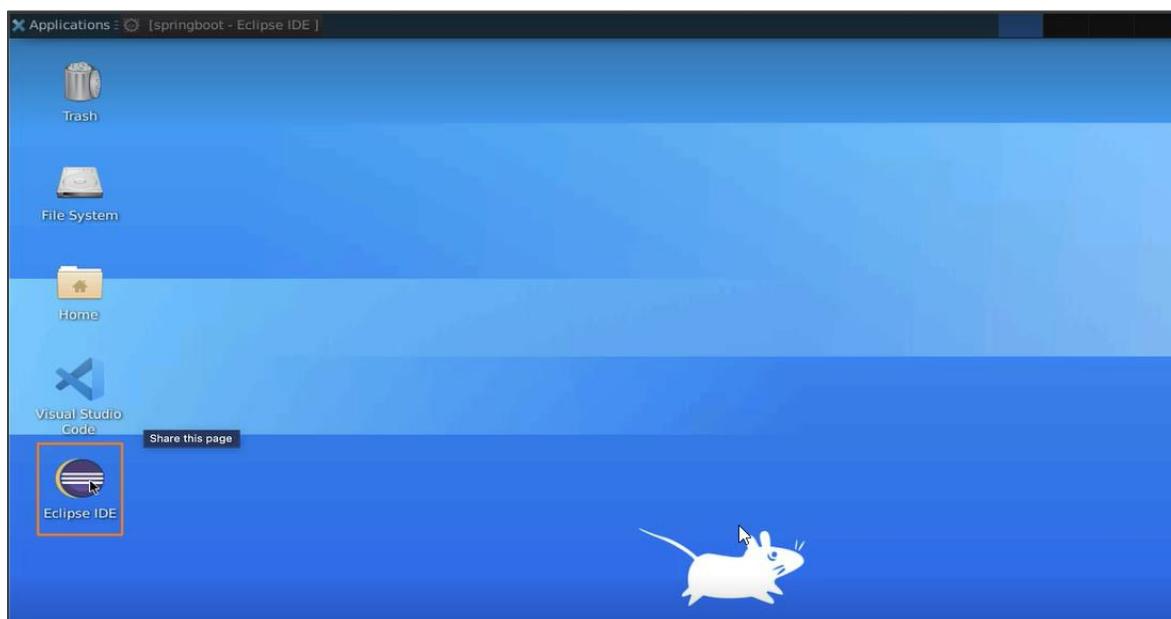
Prerequisites: None

Steps to be followed:

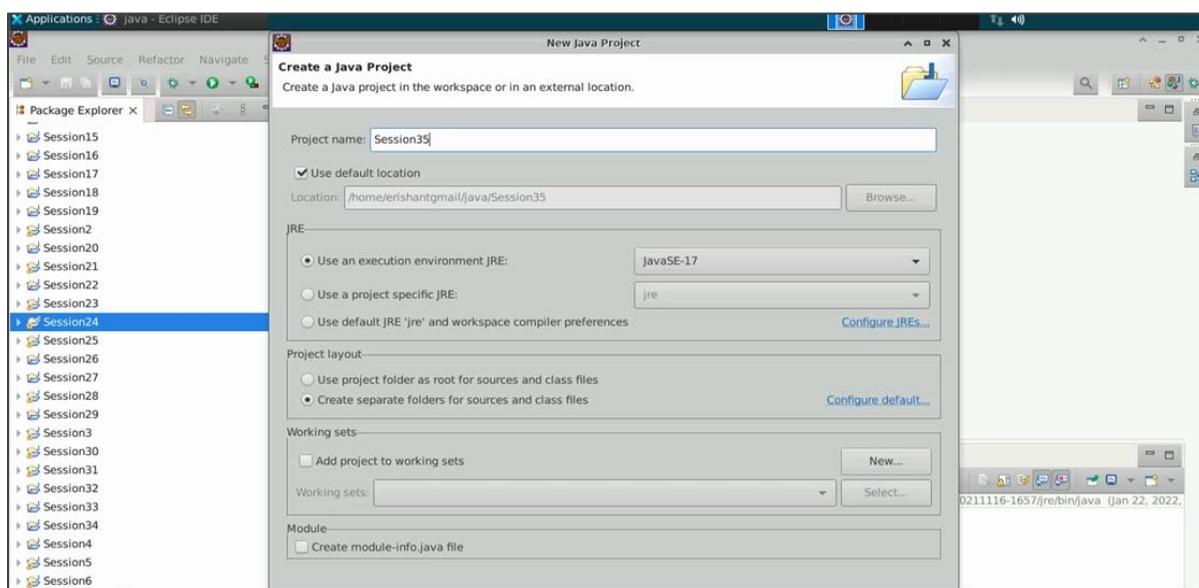
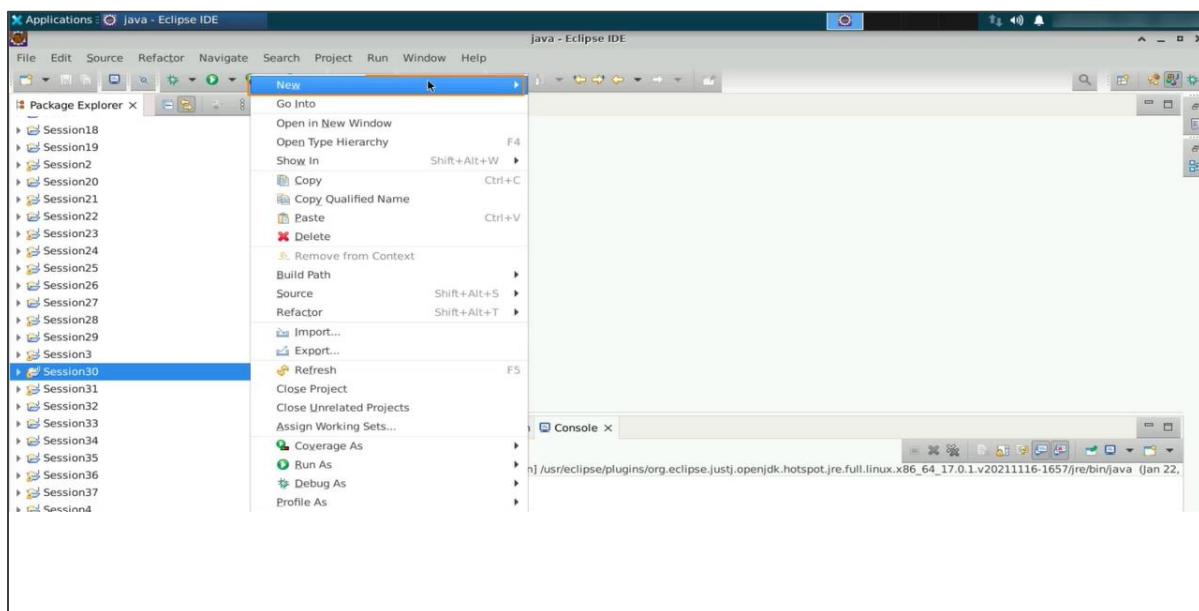
1. Create a Java project
2. Create primitive variables
3. Add methods on the wrapper classes

Step 1: Create a Java project

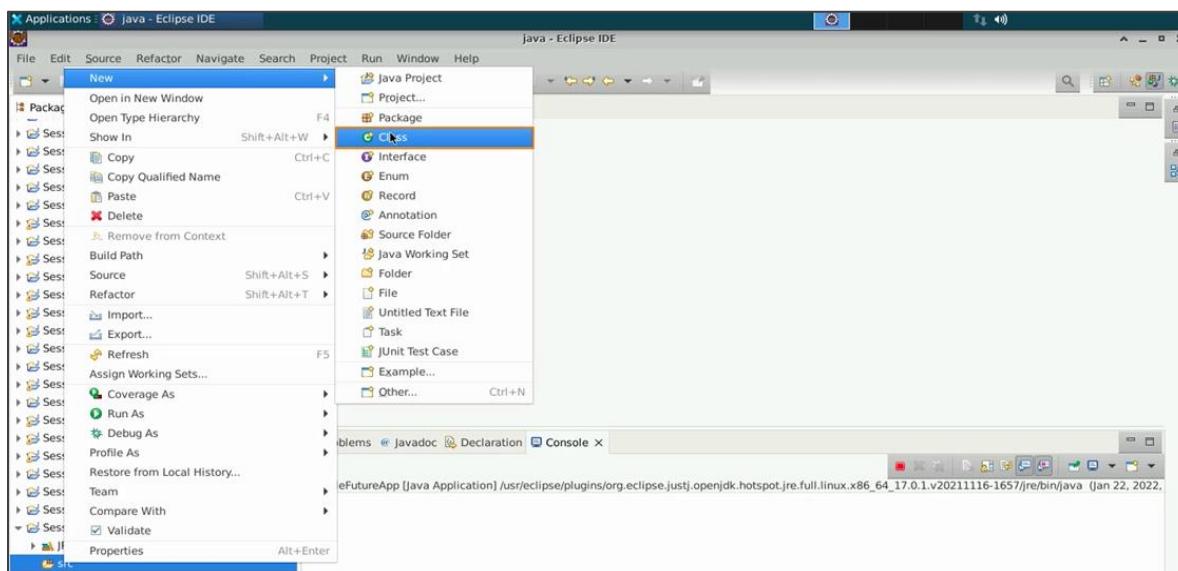
1.1 Open the Eclipse IDE



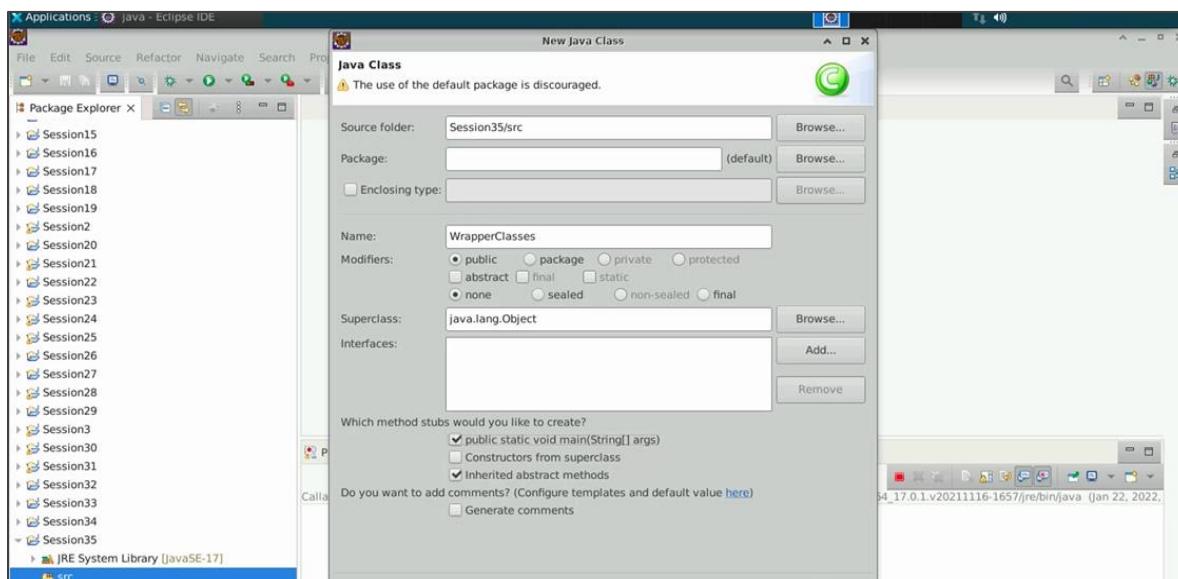
1.2 Create a new Java project called Session35



1.3 Right-click on the **src** folder, click on **New**, and then select **Class**



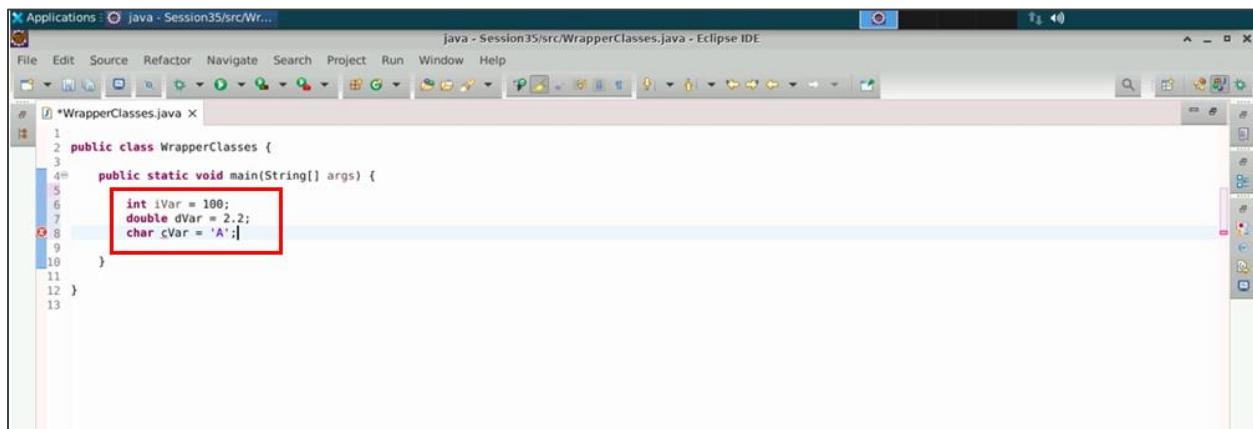
1.4 Name the class **WrapperClasses** and make sure it contains the main method



Step 2: Create primitive variables

2.1 Inside the **WrapperClasses** class, declare the following primitive variables:

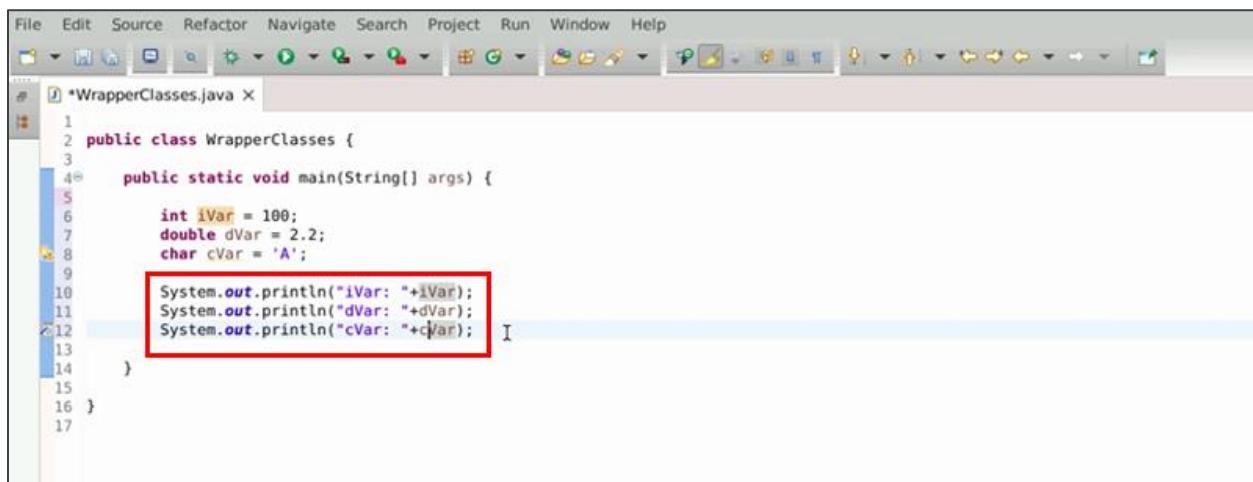
```
int iVar = 100;  
double dVar = 2.2;  
char cVar = 'A';
```



The screenshot shows the Eclipse IDE interface with the file "WrapperClasses.java" open. The code defines a public class "WrapperClasses" with a main method. Inside the main method, three primitive variable declarations are shown: int iVar = 100; double dVar = 2.2; and char cVar = 'A';. These three lines of code are highlighted with a red rectangular box.

```
File Edit Source Refactor Navigate Search Project Run Window Help  
*WrapperClasses.java X  
1  
2 public class WrapperClasses {  
3  
4 public static void main(String[] args) {  
5  
6     int iVar = 100;  
7     double dVar = 2.2;  
8     char cVar = 'A';  
9  
10 }  
11  
12 }  
13
```

2.2 Use the **System.out.println()** method to print the values of the primitive variables



The screenshot shows the Eclipse IDE interface with the file "WrapperClasses.java" open. The code is identical to the previous step, but it now includes three System.out.println statements at the bottom of the main method to print the values of iVar, dVar, and cVar. These three lines are highlighted with a red rectangular box.

```
File Edit Source Refactor Navigate Search Project Run Window Help  
*WrapperClasses.java X  
1  
2 public class WrapperClasses {  
3  
4 public static void main(String[] args) {  
5  
6     int iVar = 100;  
7     double dVar = 2.2;  
8     char cVar = 'A';  
9  
10    System.out.println("iVar: "+iVar);  
11    System.out.println("dVar: "+dVar);  
12    System.out.println("cVar: "+cVar);  
13  
14 }  
15  
16 }  
17
```

2.3 Run the project

The screenshot shows the Eclipse IDE interface. The left pane displays the Java code for `WrapperClasses.java`. The right pane shows the `Console` view with the following output:

```
<terminated> WrapperClasses [Java Application] /usr/eclipse/plugins/org.eclipse.justj.open
ivar: 100
dvar: 2.2
cvar: A
```

You will see the values in the output as **100**, **2.2**, and **A**.

Step 3: Add methods on the wrapper classes

3.1 Create an Integer object to wrap the int primitive using the **valueOf()** method:

`Integer iRef = Integer.valueOf(iVar);`

The screenshot shows the `WrapperClasses.java` code in the Eclipse IDE. A red box highlights the line of code where the `valueOf()` method is called:

```
1
2 public class WrapperClasses {
3
4     public static void main(String[] args) {
5
6         int iVar = 100;
7         double dVar = 2.2;
8         char cVar = 'A';
9
10        System.out.println("iVar: "+iVar);
11        System.out.println("dVar: "+dVar);
12        System.out.println("cVar: "+cVar);
13
14        // Wrapper classes
15        // For every primitive we have a class called Wrapper Class
16        // int -> Integer
17        // double -> Double
18        // char -> Character
19
20        Integer iRef = Integer.valueOf(iVar);
21
22    }
23
24
25 }
```

This process of wrapping a primitive value into an object is called **boxing**.

3.2 Repeat this step for double and char variables, creating Double and Character objects, respectively

```

1  public class WrapperClasses {
2
3     public static void main(String[] args) {
4
5         int iVar = 100;
6         double dVar = 2.2;
7         char cVar = 'A';
8
9         System.out.println("iVar: "+iVar);
10        System.out.println("dVar: "+dVar);
11        System.out.println("cVar: "+cVar);
12
13        // Wrapper classes
14        // For every primitive we have a class called Wrapper Class
15        // int -> Integer
16        // double -> Double
17        // char -> Character
18
19        // BOXING
20        // We wrapped the primitive into an object
21        Integer iRef = Integer.valueOf(iVar);
22        Double dRef = Double.valueOf(dVar);
23        Character cRef = new Character(cVar);
24
25    }
26
27
28 }
29

```

3.3 Extract and print the primitive values from the wrapper objects using methods like **intValue()**, **doubleValue()**, and **charValue()**. Run the project by clicking on green play button.

```

1  public class WrapperClasses {
2
3     public static void main(String[] args) {
4
5         int iVar = 100;
6         double dVar = 2.2;
7         char cVar = 'A';
8
9         System.out.println("iVar: "+iVar);
10        System.out.println("dVar: "+dVar);
11        System.out.println("cVar: "+cVar);
12
13        // Wrapper classes
14        // For every primitive we have a class called Wrapper Class
15        // int -> Integer
16        // double -> Double
17        // char -> Character
18
19        // BOXING
20        // We wrapped the primitive into an object
21        Integer iRef = Integer.valueOf(iVar);
22        Double dRef = Double.valueOf(dVar);
23
24        // Character cRef = new Character(cVar);
25        Character cRef = Character.valueOf(cVar);
26
27        // UnBoxing
28        System.out.println("iVar is: "+iRef.intValue());
29        System.out.println("dVar is: "+dRef.doubleValue());
30        System.out.println("cVar is: "+cRef.charValue());
31
32    }
33
34 }
35

```

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Applications : java - Session35/src/WrapperClasses.java - Eclipse IDE
- Left Panel:** Shows the project structure with a file named "WrapperClasses.java".
- Code Editor:** Displays the Java code for "WrapperClasses.java". The code demonstrates primitive variable assignment and conversion to their corresponding wrapper classes (Integer, Double, Character) using the `valueOf()` method. It also shows the printing of these values using `System.out.println()`.
- Console View:** Shows the output of the application. The output is:


```
<terminated> WrapperClasses [Java Application] /usr/eclipse/plugins/org.eclipse.jdt.core
iVar: 100
dVar: 2.2
cVar: A
iVar is: 100
dVar is: 2.2
cVar is: A
```

You can see the same values are printed twice.

3.4 Create primitive variables and assign the values from the wrapper class objects directly without using value methods

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Applications : java - Session35/src/WrapperClasses.java - Eclipse IDE
- Left Panel:** Shows the project structure with a file named "WrapperClasses.java".
- Code Editor:** Displays the Java code for "WrapperClasses.java". This version of the code uses direct assignment from the wrapper class objects (`iRef`, `dRef`, `cRef`) to primitive variables (`i`, `d`, `c`) instead of using the `intValue()`, `doubleValue()`, and `charValue()` methods.
- Console View:** Shows the output of the application. The output is identical to the previous screenshot:


```
<terminated> WrapperClasses [Java Application] /usr/eclipse/plugins/org.eclipse.jdt.core
iVar: 100
dVar: 2.2
cVar: A
iVar is: 100
dVar is: 2.2
cVar is: A
```

3.5 Run the project and observe the output, which will display the values of the primitive variables as well as the extracted values from the wrapper objects

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Applications - Java - Session35/src/Wr...
- Menu Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Toolbar:** Standard Eclipse toolbar.
- Left Panel:** Shows the file `WrapperClasses.java`.
- Code Editor:** Contains the following Java code:// Wrapper classes
// For every primitive we have a class called Wrapper Class
// int -> Integer
// double -> Double
// char -> Character

// BOXING
// We wrapped the primitive into an object
Integer iRef = Integer.valueOf(iVar); // Explicit Way
Double dRef = Double.valueOf(dVar);

// Character cRef = new Character(cVar);
Character cRef = Character.valueOf(cVar);

// UnBoxing
System.out.println("iVar is: "+iRef.intValue());
System.out.println("dVar is: "+dRef.doubleValue());
System.out.println("cVar is: "+cRef.charValue());

//AUTO-BOXING
Integer iRef1 = iVar; //Implicit Way
Double dRef1 = dVar;
Character cRef1 = cVar;

// AUTO-UnBoxing
int i = iRef1; //iRef1.intValue();
double d = dRef1;
char c = cRef1;

System.out.println("i is: "+i);
System.out.println("d is: "+d);
System.out.println("c is: "+c);
- Right Panel:** Shows the `Console` view with the following output:

```
<terminated> WrapperClasses [Java Application] /usr/eclipse/plugins/org.eclipse.justj.open  
iVar: 100  
dVar: 2.2  
cVar: A  
iVar is: 100  
dVar is: 2.2  
cVar is: A  
i is: 100  
d is: 2.2  
c is: A
```

In conclusion, the demo illustrates the usage of Java wrapper classes to represent primitive data types as references, providing additional functionality and flexibility when working with objects in Java programs.