Chapter 9

Wrapper Classes

Note: Primitive data types like int, float, double, char and long cannot be directly store and retrieve in collections. Primitive data types may be converted into object types by using the wrapper classes stored in the *java.lang* package.

Wrapper Classes

As we know that, Collection cannot handle primitive data types. It may be converted into object types by using the Wrapper classes.

As the name says, a wrapper class wraps (encloses) around a data type and gives it an object appearance. Wherever, the data type is required as an object, this object can be used. Wrapper classes include methods to unwrap the object and give back the data type.

Observe the following conversion.

```
int k = 100;
Integer it1 = new Integer(k);
```

The **int** data type **k** is converted into an object, **it1** using **Integer** class. The **it1** object can be used in Java programming wherever **k** is required an object.

The following code can be used to unwrap (getting back **int** from **Integer** object) the object **it1**.

```
int m = it1.intValue();
System.out.println(m*m); // prints 10000
```

intValue() is a method of **Integer** class that returns an **int** data type.

In the above code, **Integer** class is known as a wrapper class (because it wraps around **int** data type to give it an impression of object). To wrap (or to convert) each primitive data type, there comes a wrapper class. Eight wrapper classes exist in **java.lang** package that represent 8 data types. Following list gives.

Table 7.1 Wrapper classes for converting primitive types.

Primitive data type	Wrapper class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Following is the hierarchy of the above classes.

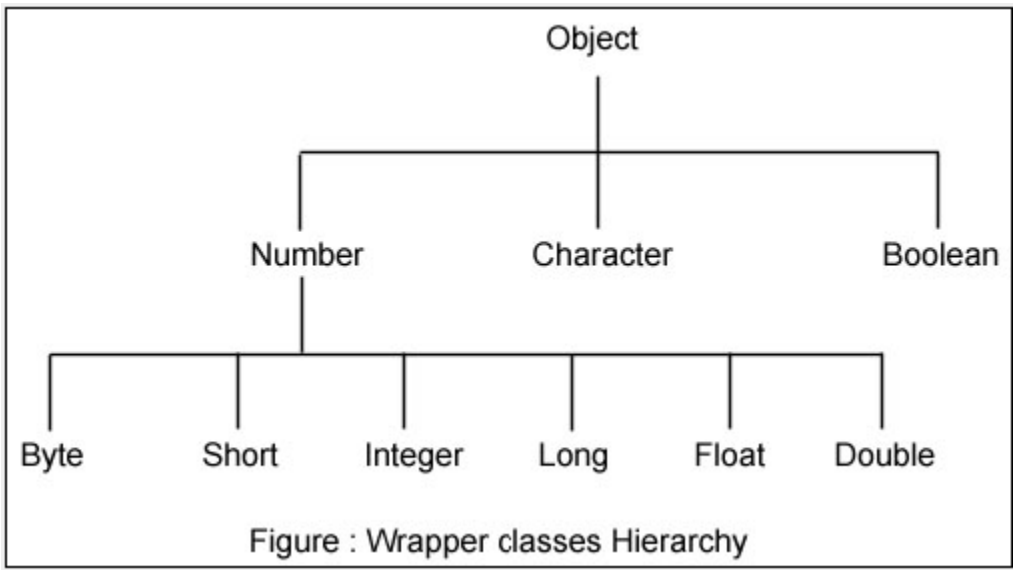


Figure : Wrapper classes Hierarchy

All the 8 wrapper classes are placed in **java.lang** package so that they are implicitly imported and made available to the programmer. As you can observe in the above hierarchy, the super class of all numeric wrapper classes is **Number** and the super class for **Character** and **Boolean** is Object. All the wrapper classes are defined as **final** and thus designers prevented them from inheritance.

Importance of Wrapper classes

There are mainly two uses with wrapper classes.

1. **To convert simple data types into objects, that is, to give object form to a data type; here constructors are used.**

Constructor Calling	Conversion Action
Integer intVal = new Integer(i)	Primitive integer to Integer object
Float floatVal = new Float(f)	Primitive float to Float object
Double doubleVal = new Double(d)	Primitive double to Double object
Long longVal = new Long(l)	Primitive long to Long Object.

Table 7.2 Converting primitive numbers to object number using constructor method

Note: I,f,d and l are primitive data values representing int, float, double and long types.

Method Calling	Conversion Action
int I = IntVal.intValue( );	Integer object to primitive int
float f = FloatVal.floatValue( );	Float object to primitive float
long l = LongVal.longValue( );	Long object to primitive long
double d = DoubleVal.doubleValue( )	Double object to primitive double

Table 7.3 Converting object numbers to primitive number using typeValue() method

```
public class WrappingUnwrapping
{
 public static void main(String args[])
 {                                                    //  data types
   byte grade = 2;
   int marks = 50;
   float price = 8.6f;                                  // observe a suffix of
   double rate = 50.5;
                                                // data types to objects
   Byte g1 = new Byte(grade);                   // wrapping
   Integer m1 = new Integer(marks);
   Float f1 = new Float(price);
   Double r1 = new Double(rate);
                          // let us print the values from objects
   System.out.println("Values of Wrapper objects (printing as objects)");
   System.out.println("Byte object g1:  " + g1);
   System.out.println("Integer object m1:  " + m1);
   System.out.println("Float object f1:  " + f1);
```

```
    System.out.println("Double object r1:  " + r1);
    // objects to data types (retrieving data types from objects)
    byte bv = g1.byteValue();              // unwrapping
    int iv = m1.intValue();
    float fv = f1.floatValue();
    double dv = r1.doubleValue();
            // let us print the values from data types
    System.out.println("Unwrapped values (printing as data types)");
    System.out.println("byte value, bv: " + bv);
    System.out.println("int value, iv: " + iv);
    System.out.println("float value, fv: " + fv);
    System.out.println("double value, dv: " + dv);
  }
}
```

```
Values of Wrapper objects (printing as objects)
Byte object g1:  2
Integer object m1:  50
Float object f1:  8.6
Double object r1:  50.5
Unwrapped values (printing as data types)
byte value, bv: 2
int value, iv: 50
float value, fv: 8.6
double value, dv: 50.5
```

As you can observe from the screenshot, constructors of wrapper classes are used to convert data types into objects and the methods of the form **typeValue()** are used to retrieve back the data type from the object.

Wrapping and Unwrapping

int marks = 50;
Integer m1 = new Integer(marks); // **Autoboxing**

In the above statement, an **int** data type **marks** is given an object form **m1** just by passing the variable to the constructor of **Integer** class. Wherever, **marks** is required as an object, **m1** can be used.

After using the Integer object in programming, now the programmer may require back the data type, as objects cannot be used in arithmetic operations. Now the object **m1** should be unwrapped.

int iv = m1.intValue(); // **UnBoxing**

For unwrapping, the method **intValue()** of Integer class used. The **int** value **iv** can be used in arithmetic operations.

2. **To convert strings into data types (known as parsing operations), here methods of type parseValue() are used.**

Parsing Operations

The other usage of wrapper classes is converting strings into data types. Sometimes, in programming, the programmer retrieves data in the form of strings. Data types are retrieved as strings. For example, the **readLine()** method of **BufferedReader** returns a string always of any data type it reads, command-line arguments are retrieved as strings and a text field in GUI gets a string. All these strings should be converted back to data types to use in arithmetic operations.

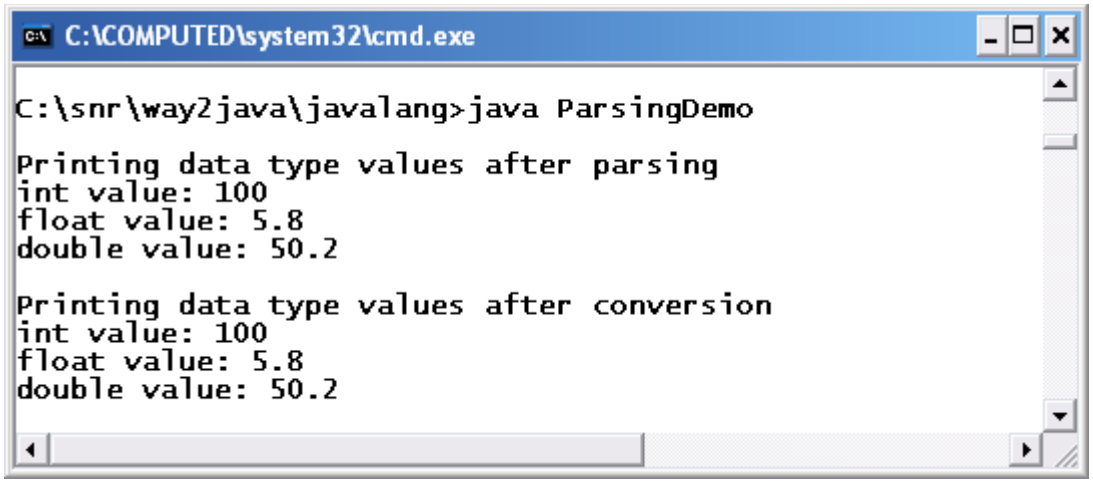
The following program illustrates.

```
public class ParsingDemo
{
 public static void main(String args[])
 {
  String price ="100";              // declare some strings
  String rate = "5.8f";
  String tax ="50.2";
                    // performing parsing operations on strings
  int x = Integer.parseInt(price);
  float f1 = Float.parseFloat(rate);
  double y = Double.parseDouble(tax);

  System.out.println("\nPrinting data type values after parsing");
  System.out.println("int value: " + x);
  System.out.println("float value: " + f1);
  System.out.println("double value: " + y);
   // another style of converting strings into data types, very less used
```

```
    Integer i1 = new Integer(price);
    Float f2 = new Float(rate);
    Double d1 = new Double(tax);
   // extracting data types from wrapper objects
    int x1 = i1.intValue();
    float f3 = f2.floatValue();
    double d2= d1.doubleValue();

    System.out.println("\nPrinting data type values after conversion");
    System.out.println("int value: " + x1);
    System.out.println("float value: " + f3);
    System.out.println("double value: " + d2);
  }
}
```



String price = "100";
int x = Integer.parseInt(price);

parseInt() is a static method of **java.lang.Integer** class that takes a string as parameter and returns an **int** value corresponding to the string. Similarly, the methods **parseFloat()**, **parseDouble()** of **Float** and **Double** classes return float and double values. This style of conversion is mostly used in coding.

There is another way of converting strings to data types, but is less used. Following is the other way.

```
String price ="100";
Integer i1 = new Integer(price);
int x1 = i1.intValue();
```

Pass the string, **price**, to the constructor of Integer class. Now the **price** is represented as **i1**, an object of Integer class. From object **i1**, extract the **int** value with **intValue()** method of Integer class. Similarly, there exists methods like **floatValue()** and **doubleValue()** of classes Float and Double that return a float value and double value.

Now finally, you can observe that, a string value can be converted to a data type in two ways – using **parseInt()** method and using **intValue()** method; both belonging to Integer class.

The next program is another example on string conversions into all data types. Two styles of conversions are given.

1. Using parsing operation (1st way)
2. Using typeValue() method wrapper classes (2nd way)

```
public class Conversions
{
 public static void main(String args[])
 {                          // converting string to byte
  String str2 = "10";               // 1st way
  byte b1 = Byte.parseByte(str2);
  System.out.println(b1*b1);        // prints 100
                                        // 2nd way
  Byte by1 = new Byte(str2);
  byte b2 = by1.byteValue();
  System.out.println(b2*b2);        // prints 100

                           // converting string to short
  short s1 = Short.parseShort(str2);  // 1st way
  System.out.println(s1*s1);        // prints 100

  Short sh1 = new Short(str2);   // 2nd way
  short s2 = sh1.shortValue();
  System.out.println(s2*s2);        // prints 100
```

```
                        // converting string to int

  int i1 = Integer.parseInt(str2);// 1st way
  System.out.println(i1*i1);        // prints 100

  Integer in1 = new Integer(str2);          // 2nd way
  int i2 = in1.intValue();
  System.out.println(i2*i2);        // prints 100

                        // converting string to long

  long l1 = Long.parseLong(str2);         // 1st way
  System.out.println(l1*l1);        // prints 100

  Long lo1 = new Long(str2);                // 2nd way
  long l2 = lo1.longValue();
  System.out.println(l2*l2);        // prints 100

                        // converting string to float

  String str3 = "10.5f";              // 1st way
      // or it can be String str3 = "10.5";
  float f1 = Float.parseFloat(str3);
  System.out.println(f1*f1);        // prints 110.25

  Float fl1 = new Float(str3);       // 2nd way
  float f2 = fl1.floatValue();
  System.out.println(f2*f2);        // prints 110.25

                        // converting string to double
Int n =5;
  String str4 = "10.5";                     // 1st way
  double d1 = Double.parseDouble(str4);
  System.out.println(d1*d1);        // prints 110.25

  Double do1 = new Double(str4);        // 2nd way
  double d2 = do1.doubleValue();
  System.out.println(d2*d2);        // prints 110.25
```

```
                // converting string to character

    String str1 = "A";
    char ch1 = str1.charAt(0);
    System.out.println(ch1);          // prints A

                // converting string to boolean

    String str5 = "true";             // 1st way
    boolean b3 = Boolean.parseBoolean(str5);
    System.out.println(b3);
                                // 2nd way
    Boolean bo1 = new Boolean(str5);
    boolean b4 = bo1.booleanValue();
    System.out.println(b4);
  }
}
```