CHAPTER

22

WRAPPER CLASSES

We see many applications on Internet which receive data from the user and send it to the server. For example, in a business application, we type our details like name, credit card number, address, etc. and send them to the server. The server expects this data in the form of objects and hence we are supposed to send objects. In our data, 'name' is a `String` type object, but credit card number is just an `int` type value, which is not an object. This primitive datatype should also be converted into an object and then sent to the server. To do this conversion, we need wrapper classes.

A wrapper class contains a field where it stores the primitive datatype. When we create an object to a wrapper class, it carries the primitive datatype within it and hence the object can be sent to the server. The server can retrieve the primitive datatype from the object and use it.

There is another reason why we need wrapper classes. In Java, collection classes are defined in `java.util` package which handle only objects, not the primitives. Hence, if we want to use collection classes on primitives, we should convert them into objects. Here, also we need the help of wrapper classes.

Wrapper Classes

A wrapper class is a class whose object wraps or contains a primitive data type. When we create an object to a wrapper class, it contains a field and in this field, we can store a primitive data type. In other words, we can wrap a primitive value into a wrapper class object. For example, if we create an object to `Character` wrapper class, it contains a single field `char` and it is possible to store a character like `A` there, as shown in the Figure 22.1. So, `Character` is a wrapper class of `char` data type.

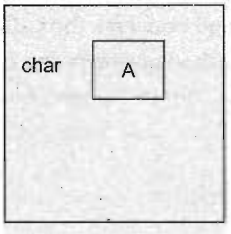

Figure 22.1 Character class object contains char type field in it

Important Interview Question

Why do we need wrapper classes?

1. They convert primitive data types into objects and this is needed on Internet to communicate between two applications.

2. The classes in java.util package handle only objects and hence wrapper classes help in this case also.

Table 22.1 contains the list of wrapper classes that are defined in `java.lang` package. They are useful to convert the primitive data types into object form.

Table 22.1

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

Number Class

`Number` is an abstract class whose subclasses are Byte, Short, Integer, Long, Float, and Double. All the methods of `Number` class are commonly available in all these subclasses.

Number Class Methods

`Number class` includes following methods:

- `byte byteValue()`: This method converts the calling object into byte value. The calling object can be an object of Byte, Short, Integer, Long, Float, or Double class.
- `short shortValue()`: This method converts the calling object into short value.
- `int intValue()`: This method converts the calling object into int value.
- `long longValue()`: This method converts the calling object into long value.
- `float floatValue()`: This method converts the calling object into float value.
- `double doubleValue()`: This method converts the calling object into double value.

All these methods are available to Byte, Short, Integer, Long, Float, and Double classes.

Character Class

The `Character` class wraps a value of the primitive type `char` in an object. If we create `Character` class object, it contains a `char` type field. In this field, we can store a primitive char value like `A`. `Character` class has only one constructor which accepts primitive data type.

```
Character(char ch)
```

So, we can create `Character` class object and store `A` there, as:

```
Character obj = new Character('A');
```

Important Methods of Character Class

`Character class` includes following important methods:

- `char charValue()`:This method is useful to convert `Character` class object again into primitive char value and returns that value. For example, we can use it as:

```
Character obj = new Character('A');
char ch = obj.charValue();
Now, ch contains 'A'.
```

- `int compareTo(Character obj)`: This method is useful to compare two Character objects. It is called as:

```
int x = obj1.compareTo(obj2);
```

- where, `obj1` and `obj2` are `Character` class objects.
- If `obj1 == obj2`, then this method returns 0.
- If `obj1 < obj2`, then it returns negative value.
- If `obj1 > obj2`, then it returns positive value.
- `String toString()`: This method converts Character object into String object and returns that String object.
- `static Character valueOf(char ch)`: This method converts a single character `ch` into Character object and returns that object.
- `static boolean isDigit(char ch)`: This method returns true if `ch` is a digit (0 to 9) otherwise returns false.
- `static boolean isLetter(char ch)`: This method returns true if `ch` is a letter (A to Z or a to z).
- `static boolean isUpperCase(char ch)`: This method returns true if `ch` is an uppercase letter (A to Z).
- `static boolean isLowerCase(char ch)`: This method returns true if `ch` is a lowercase letter (a to z).
- `static boolean isSpaceChar(char ch)`: This method returns true if `ch` represents a space which is coming from spacebar.
- `static boolean isWhitespace(char ch)`: This method returns true if `ch` represents white space. White space is a space that comes on pressing tab, enter, or backspace buttons.

- `static boolean isLetterOrDigit(char ch)`: This method returns true if ch is either letter or digit.
- `static char toUpperCase(char ch)`: This method converts ch into uppercase and returns that upper case letter.
- 13) `static char toLowerCase(char ch)`: This method converts ch into lowercase and returns that lower case letter.

Now, let us write a program to accept a character from keyboard and test what type of character is. Program 1 represents the logic for this, where the character is tested repeatedly using Character class methods to know the type of the character. This program executes till the user presses Enter button.

Program 1: Write a program that accepts the character from the keyboard and displays its type.

```
//Accept a character from keyboard and display what it is
import java.io.*;
class CharTest
{
      public static void main(String args[ ])
      throws IOException
      {
            //to accept a char from keyboard
            char ch;
            BufferedReader br = new BufferedReader(new
            InputStreamReader(System.in));

            while(true)  //execute repeatedly
            {
                  System.out.print("Enter a character: ");
                  ch = (char)br.read();

                  //test and display the type of character
                  System.out.print("You entered: ");
                  if(Character.isDigit(ch))
                  System.out.println("a digit");
                  else if (Character.isUpperCase(ch))
                  System.out.println("an uppercase letter");
                  else if (Character.isLowerCase(ch))
                  System.out.println("a lowercase letter");
                  else if(Character.isSpaceChar(ch))
                  System.out.println("a spacebar character");
                  else if(Character.isWhitespace(ch)){
                  System.out.println("a whitespace character");
                  return;
                  }
            else System.out.println("Sorry, I dont know that");
            br.skip(2); //to skip \n code from br
      }

      }
}
```

Output:

```
C:\> javac CharTest.java
C:\> java CharTest
Enter a character: y
You entered: a lowercase letter
Enter a character:
You entered: a spacebar character
Enter a character: 9
You entered: a digit
Enter a character:      [Press Enter]
You entered: a whitespace character
```

```
C:\>
```

In this program, we first accept a character from keyboard and display what type of character it is. This is done repeatedly in while loop till Enter button is pressed by the user.

Byte Class

The Byte class wraps a value of the primitive type 'byte' in an object. The `Byte` class object contains a byte type field. In this field, we can store a primitive byte number. Remember byte number ranges from -128 to +127.

Constructors

`Byte` class has two constructors. The first one takes byte number as its parameter and converts it into `Byte` class object and the next one takes a String type parameter and converts that string into `Byte` class object.

- `Byte(byte num)`

So, we can create Byte object as: `Byte obj = new Byte(120);`

- `Byte(String str)`

This constructor suggests that we can create a Byte object by converting a string that contains a byte number, as:

- `Byte obj = new Byte("120");`

Please try to understand that these two constructors can be seen in every wrapper class. This means every wrapper class contains two constructors—the first one takes the corresponding primitive data type and the other takes a string parameter. Apart from this, some wrapper classes have a third constructor also. But `Character` class has only the first type of constructor.

Important Interview Question

Which of the wrapper classes contains only one constructor?

(or) Which of the wrapper classes does not contain a constructor with String as parameter?

A) Character.

Important Methods of Byte Class

`Byte class` includes following important methods:

- `int compareTo(Byte b)`

This method is useful to compare the contents of two `Byte` class objects. It is called as:

```
int x = obj1.compareTo(obj2);
```

where, obj1 and obj2 are Byte class objects.

If obj1 == obj2, then this method returns 0.

If obj1 < obj2, then it returns negative value.

If obj1 > obj2, then it returns positive value.

boolean equals(Object obj)

This method compares the Byte object with any other object `obj`. If both have same content, then it returns true otherwise false.

- `static byte parseByte(String str)`: This method returns the primitive byte number contained in the string `str`.
- `String toString()`: This method converts Byte object into String object and returns that String object.
- `static Byte valueOf(String str)`: This method converts a string `str` that contains some byte number into `Byte` class object and returns that object.
- `static Byte valueOf(byte b)`: This method converts the primitive byte `b` into Byte object.

Let us write a program to see how one can create `Byte` class objects. Then we want to compare them and display which object has less or more contents when compared to another.

Program 2: Write a program which shows the use of `Byte` class objects.

```
//Creating Byte class objects and comparing them
import java.io.*;
class ByteDemo
{
      public static void main(String args[]) throws IOException

      {
            //to accept data from keyboard
            BufferedReader br = new BufferedReader(new
            InputStreamReader(System.in));

            //accept a byte number as string s1
            System.out.print("Enter a byte no: ");
            String s1 = br.readLine();

            //create Byte object b1 using s1
            Byte b1 = new Byte(s1);

            //accept another byte number as string s2
            System.out.print("Enter a byte no: ");
            String s2 = br.readLine();

            //create Byte object b2 using s2
            Byte b2 = new Byte(s2);

            //compare both the Byte objects contents
            int n= b1.compareTo(b2);

            if(n==0) System.out.println("Both bytes are same");
            else if(n<0) System.out.println(b1 + " is less");
            else System.out.println(b2+ " is less");

      }
}
```

Output:

```
C:\> javac ByteDemo.java
C:\> java ByteDemo
Enter a byte no: 120
Enter a byte no: 124
120 is less
```

Short Class

`Short` class wraps a value of primitive data type 'short' in its object. Short class object contains a short type field that stores a short number.

Constructors

Short class has two constructors. The first one takes short number as its parameter and converts it into Short class object and the next one takes a String type parameter and converts that string into Short class object.

- `Short(short num)`

This constructor is useful to construct the Short class object by supplying a short number to it, as:

```
short s = 14007;
Short  obj = new Short(s);
```

- `Short(String str)`

This constructor is useful to construct the Short class object by passing a string `str` to it as:

```
String str = "14007";
Short obj = new Short(str);
```

Important Methods of Short Class

`Short class` includes following important methods:

- `int compareTo(Short obj)` : This method compares the numerical value of two Short class objects and returns 0, -ve value, or +ve value.
- `boolean equals(Object obj)` : This method compares the Short object with any other object `obj`. If both have the same content then it returns true otherwise false.
- `static short parseShort(String str)` : This method returns `int` equivalent of the string `str`.
- `String toString()` : This method returns a string form of the Short object.
- `static Short valueOf(String str)` : This method converts a string `str` that contains some short number into Short class object and returns that object.

Integer Class

The `Integer` class wraps a value of the primitive type `int` in an object. The Integer class object contains an `int` type field. In this field, we can store a primitive `int` number.

Constructors

`Integer` class has two constructors. The first one takes `int` number as its parameter and converts it into Integer class object and the next one takes a String type parameter and converts that string into Integer class object.

- `Integer(int num)` : This means Integer object can be created, as:

`Integer obj = new Integer(123000);` Here, we are converting a primitive `int` value into Integer object. This is called 'boxing'.

Important Interview Question

What is boxing?

Converting a primitive datatype into an object is called 'boxing'.

- `Integer(String str)`: This constructor suggests that we can create an Integer object b converting a string that contains an `int` number, as:

```
Integer obj = new Integer("198663");
```

Important Methods of Integer Class

`Integer class` includes following important methods:

- `int compareTo(Integer obj)`: This method compares the numerical value of two Integ class objects and returns 0,-ve value, or +ve value.
- `boolean equals(Object obj)`: This method compares the Integer object with any oth object `obj`. If both have the same content, then it returns true otherwise false.
- `static int parseInt(String str)`: This method returns `int` equivalent of the string `str`.
- `String toString()`: This method returns a string form of the Integer object.
- `static Integer valueOf(String str)`: This method converts a string `str` that contai some `int` number into Integer class object and returns that object.
- `static String toBinaryString(int i)`: This method converts decimal integer number into binary number system and returns that binary number as a string.
- `static String toHexString(int i)`: This method converts decimal integer number `i` in hexadecimal number system and returns that hexadecimal number as a string.
- `static String toOctalString(int i)`: This method converts decimal integer number `i` in octal number system and returns that octal number as a string.
- `int intValue()`: This method converts Integer object into primitive `int` type value. This called 'unboxing'.

Important Interview Question

What is unboxing?

Converting an object into its corresponding primitive datatype is called unboxing.

What happens if a string like "Hello"is passed to parseInt() method?

Ideally, a string with an integer value should be passed to parseInt() method. So, on passing "Hello", an exception called 'NumberFormatException' occurs since the parseInt() method cannot convert the given string "Hello" into an integer value.

Program 3: Write a program to accept an integer number from keyboard and convert it into oth number systems.

```
//Convert int into binary, hexadecimal, and octal format
import java.io.*;
class Convert
{
      public static void main(String args[ ])
      throws IOException
      {
            BufferedReader br = new BufferedReader(new
            InputStreamReader(System.in));
            System.out.print("Enter an integer: ");
            String str = br.readLine();

            //convert string into int
            int i = Integer.parseInt(str);
            System.out.println("In decimal: "+ i);
```

```
                //convert int into other systems
                str = Integer.toBinaryString(i);
                System.out.println("In binary: "+str);

                str = Integer.toHexString(i);
                System.out.println("In hexadecimal: "+str);

                str = Integer.toOctalString(i);
                System.out.println("In octal: "+str);
        }
}
```

Output:

```
C:\> javac Convert.java
C:\> java Convert
Enter an integer: 456
In decimal: 456
In binary: 111001000
In hexadecimal: 1c8
In octal: 710
```

Long Class

The `Long` class contains a primitive 'long' type data. The object of `Long` class contains a field where we can store a long value.

Constructors

Long class has two constructors. The first one takes long number as its parameter and converts it into Long class object and the next one takes a String type parameter and converts that string into Long class object.

- `Long(long num)`: This means Long object can be created as: `Long obj = new Long(12300044);`
- `Long(String str)`: This constructor suggests that we can create Long object by converting a string that contains a long number, as:

```
String str = "12300044";
Long obj = new Long(str);
```

Important Methods of Long Class

`Long class` includes following important methods:

- `int compareTo(Long obj)`: This method compares the numerical value of two Long class objects and returns 0, -ve value, or +ve value.
- `boolean equals(Object obj)`: This method compares the Long object with any other object `obj`. If both have same content, then it returns true otherwise false.
- `static long parseLong(String str)`: This method returns long equivalent of the string `str`.
- `String toString()`: This method converts Long object into String object and returns the String object.

- `static Long valueOf(String str)`: This method converts a string `str` that contains some long number into Long object and returns that object.

Float Class

The `Float` class wraps a value of the primitive type `float` in an object. The Float class object contains a float type field that stores a primitive float number.

Constructors

`Float` class has three constructors. The first one takes float number as its parameter and converts it into Float class object and the next one takes a double type number and converts into Float class object and then the third one takes a String type parameter and converts that string into Float class object.

- `Float(float num)`: This means Float object can be created as:

```
float f = 12.987f;
Float obj = new Float(f);
```

- `Float(double num)`: This constructor is useful to create Float class object with a double type value converted into float.
- `Float(String str)`: This constructor suggests that we can create a Float object by converting a string that contains a float number, as:

```
Float obj = new Float("12.987");
```

Important Methods of Float Class

`Float class` includes following important methods:

- `int compareTo(Float obj)`: This method compares the numerical value of two Float class objects and returns 0, -ve value, or +ve value.
- `boolean equals(Object obj)`: This method compares the Float object with any other object `obj`. If both have same content then it returns true otherwise false.
- `static float parseFloat(String str)`: This method returns float equivalent of the string `str`.
- `String toString()`: This method returns a string form of the Float object.
- `static Float valueOf(String str)`: This method converts a string `str` that contains some float number into Float object and returns that object.

Double Class

The `Double` class wraps a value of the primitive type `double` in an object. The Double class object contains a double type field that stores a primitive double number.

Constructors

`Double` class has two constructors. The first one takes double number as its parameter and converts it into Double class object and the next one takes a String type parameter and converts that string into Double class object.

- `Double(double num)` : This means Integer object can be created as:

```
double d = 12.1223;
Double obj = new Double(d);
```

- `Double(String str)` : This constructor is useful to create a Double object by converting a string that contains a double number, as:

```
String str = "12.1223";
Double obj = new Double(str);
```

Important Methods of Double Class

`Double class` includes following important methods:

- `int compareTo(Double obj)` : This method compares the numerical value of two Double class objects and returns 0, -ve value, or +ve value.
- `boolean equals(Object obj)` : This method compares the Double object with any other object `obj`. If both have same content then it returns true otherwise false.
- `static double parseDouble(String str)` : This method returns double equivalent of the string `str`.
- `String toString()` : This method returns a string form of the Double object.
- `static Double valueOf(String str)` : This method converts a string `str` that contains a double number into Double class object and returns that object.

Boolean Class

The Boolean class object contains a primitive 'boolean' type data. The object of Boolean class contains a field where we can store a boolean value.

Constructors

`Boolean` class has two constructors. The first one takes boolean number as its parameter and converts it into Boolean class object and the next one takes a String type parameter and converts that string into Boolean class object.

- `Boolean(boolean value)` : This means Boolean object can be created, as:

```
Boolean obj = new Boolean(true);
```

- `Boolean(String str)` : This constructor suggests that we can create Boolean object by converting a string that contains a boolean value, as:

```
String str = "false";
Boolean obj = new Boolean(str);
```

Important Methods of Boolean Class

`Boolean class` includes following important methods:

- `int compareTo(Boolean obj)` : This method compares the numerical value of two Boolean class objects and returns 0, -ve value, or +ve value.

numbers between 0 and 1. Suppose we want to generate them between 0 and 10, then we oug multiply the method's output by 10. This is shown in Program 4.

Program 4: Write a program that generates random numbers repeatedly between '0' and '10'. need to also ensure that if the generated number is '0' then program gets terminated.

```
//Generating random nos. between 0 and 10
class Random
{
      public static void main(String args[ ])
      throws Exception
      {
            System.out.println("Random no.s between 0 and 10: ");

            while(true)
            {
                  /*random() returns double type between 0 and 1. But we
                  the no. as integer and between 0 and 10. So multiply it
                  by 10 and convert into int. */

                  double d= 10*Math.random();
                  int i = (int)d;
                  System.out.println(i);

                  //Let the execution wait till 2000 milli seconds = 2 sec
                  Thread.sleep(2000);

                  if(i==0) System.exit(0); //come out
            } //end of while
      }
}
```

Output:

```
C:\> javac Random.java
C:\> java Random
4
3
5
5
4
3
3
9
0
```

In the above program, we used a method:

```
Thread.sleep(2000);
```

which made processing to be suspended for 2000 milli seconds time. This means the JVM execution for 2 seconds (2000 milli seconds= 2 sec) and then starts again with the next state Thread class is found in `java.lang` package.

> **Note**
>
> *For more information about threads, refer Chapter 26.*

Conclusion

Since OOPS is all about objects, it is necessary to convert every thing into objects. Wrapper cl are useful to convert primitive data types into objects. Moreover, they contain methods to reco an object into primitive data. The methods of a wrapper class facilitate a programmer to work objects easily and effectively.