ShreeSoft INFORMATICS

Address: Tirumala Plaza, Upendra Nagar, Cidco, Nashik.

Join us on Telegram:
https://t.me/ShreeSoftinformatics

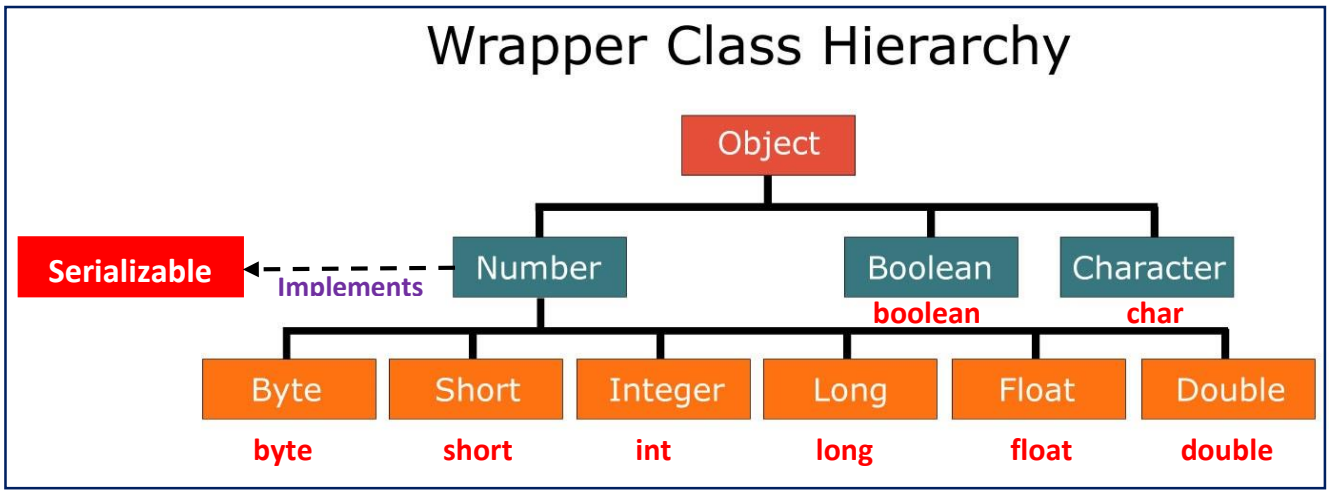
Join us on WhatsApp:
https://chat.whatsapp.com/BrLM0ZoWMbpA8nXi9Mrocl

Contact: +91 8308341531

Java Wrapper Classes

What is Wrapper Classes?

- A **Wrapper class** in Java is the type of class that provides a mechanism to convert the primitive data types into the objects and vice-versa.
- When a wrapper class is created, there is a creation of a new field in which we store the primitive data types. The object of the wrapper class wraps or holds its respective primitive data type.
- Java 5.0 version introduced two important new features autoboxing and unboxing that convert primitive data type values into objects and objects into primitive data type values automatically.
- The process of automatic conversion of primitive data type into an object is known as **Autoboxing** and vice versa **unboxing**.
- All the wrapper classes Byte, Short, Integer, Long, Double and, Float, are subclasses of the abstract class **Number.** While Character and Boolean wrapper classes are the subclasses of class **Object.**
- A wrapper class is bundled default with the Java library and it is located in (jre/lib/rt.jar file).



Example: Code for Boxing and Unboxing an int value

Boxing an int value:

```
int x = 13;
Integer i1 = new Integer(x); // pass the primitive type to the wrapper class constructor
```

Unboxing an int value:

```
int x = i1.intValue(); // get the value from wrapper object using intValue() method.
```

Need for Wrapper Classes:

1. The wrapper objects hold much more memory compared to primitive types.

2. Wrapper Class will **c**onvert primitive data types into objects. The object is needed to support synchronization in multithreading.

3. Wrapper class objects allow null values while primitive data type doesn't allow it.

4. Wrapper classes are also used to provide a variety of utility functions for primitive data types like converting primitive types to string objects and vice-versa, converting to various bases like binary, octal or hexadecimal, or comparing various objects.

Uses of Wrapper Classes in Java?

1. **Serialization:** In Serialization, We need to convert the objects into streams. If we have a primitive value and we want to serialize them then we can do this by converting them with the help of wrapper classes.

2. **Synchronization:** In Multithreading, Java synchronization works with objects.

3. **java.util package:** The package java.util provides many utility classes to deal with objects rather than values.

4. **Collection Framework:** The Collection Framework in Java works only with objects. All classes of the collection framework like ArrayList, LinkedList, Vector, HashSet, LinkedHashSet, TreeSet, PriorityQueue, ArrayDeque, etc, work only with objects.

5. **Changing the value inside a Method:** So, if we pass a primitive value using call by value, it will not change the original value. But, it will change the original value if we convert the primitive value into an object.

6. **Polymorphism:** Wrapper classes also help in achieving Polymorphism in Java.

Number Class:

- **Number class in Java** is an abstract class that is the superclass of Byte, Short, Integer, Long, Float, and Double classes.

- The Byte, Short, Integer, Long, Float, and Double classes are the most commonly used wrapper classes that represent numeric values.

- All the numeric type wrappers inherit from abstract class Number.

Number class methods:

1. **byte byteValue():** This method returns the value of an object as a byte which may involve rounding or truncation. In other words, it converts the calling object into byte value. The calling object may be an object of Byte, Short, Integer, Long, Float, or Double class.

2. **short shortValue():** This method returns the value of an object as a short which may involve rounding or truncation. In simple words, it converts the calling object into short value.

3. **abstract int intValue():** It returns the value of an object as a short which may involve rounding or truncation. That is, it converts the calling object into int value.

4. **abstract long longValue():** It returns the value of an object as long which may involve rounding or truncation. It is used to convert the calling object into long value.

5. **abstract float floatValue():** This method returns the value of an object as float which may involve rounding.

6. **abstract double doubleValue():** This method returns the value of an object as long which may involve rounding.

All these methods provided by abstract Number class in java are available to Byte, Short, Integer, Long, Float, and Double classes.

Byte Class:

- **Byte class in Java** is a wrapper class that wraps (converts) a value of primitive data type into an object.

- An object of Byte class contains a single byte type field. In this field, we can store a primitive type byte value. Note that the range of byte is from -128 to +127.

- Byte class was introduced in JDK 1.1. It is present in java.lang.Byte package.

Byte Field Constants:

Java Byte class provides several field constants that are as follows:

1. **static int BYTES:**
   It is the number of bytes that is used to represent a byte value in two's complement binary form. It was introduced in JDK 1.8.

2. **static byte MAX_VALUE:**
   It represents a constant that can hold the maximum value of byte(2^7-1).

3. **static byte MIN_VALUE:**
   It represents a constant that can hold the minimum value of byte (2^{-7}).

4. **static int SIZE:**
   It is the number of bits that is used to represent a byte value in two's complement binary form. It was introduced in JDK 1.5.

5. **static Class<Byte> TYPE:**
   It is a class instance that represents the primitive type byte.

Byte Class Constructor

1. **Byte(byte num):** This form of constructor accepts byte number as its parameter and converts it into Byte class object.
   Syntax:
   Byte obj = new ~~Byte~~(121);

2. **Byte(String str):** This constructor accepts a parameter of String type and converts that string into Byte class object.
   Syntax:
   Byte obj = new ~~Byte~~("121");

Note:

We cannot create object of wrapper class using constructor because in java 9 constructor deprecated. This deprecation will done because saves memory and CPU cycles. Instead of constructor we can create object using valueOf( ) method of wrapper class.

Byte Class Methods:

In addition to methods inherited from Number class and Object class, Java Byte class also provides some useful important methods.

1. **int compareTo(Byte b):** This method is used to compare the contents of two Byte class objects numerically.

2. **boolean equals(Object obj):** This method is used to compare Byte object with another Byte object obj. If both have the same contents, this method returns true otherwise returns false.

3. **static byte parseByte(String str):** This method returns the primitive byte value contained in the string argument str.

4. **String toString():** The toString() method converts Byte object into String object and returns that String object.

5. **static Byte valueOf(String str):** It converts string str containing some byte value into Byte class object and returns that Byte object.
   In other words, it returns a Byte object holding a value given by the specified String.

6. **static Byte valueOf(byte b):** This method converts the primitive byte value into Byte class object. In other words, it returns a Byte object representing the specified byte value.

7. **static int compare(byte x, byte y):** It is used to compare two byte values numerically.

8. **static Byte decode(String str):** This method decodes a String into a Byte.

9. **static int toUnsignedInt(byte x):** This method converts byte value to an int by an unsigned conversion.

10. **static long toUnsignedLong(byte x):** This method converts byte value to a long by an unsigned conversion.

Program 1: Write a program to read 2 number from console using BufferedReader class and perform arithmetic operations.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class Sample1 {
	public static void main(String[] args) throws IOException{
		InputStreamReader isr = new InputStreamReader(System.in);
		BufferedReader br = new BufferedReader(isr);

		System.out.println("Enter First byte value");
		byte b1 = Byte.parseByte(br.readLine());

		System.out.println("Enter First byte value");
		byte b2 = Byte.parseByte(br.readLine());

		System.out.println("Addition:"+(b2+b1));
	}
}
```

Program 2: Write a program to read 2 number from console using BufferedReader class and compare the content of 2 byte object.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class Sample2 {
	public static void main(String[] args) throws IOException{
		InputStreamReader isr = new InputStreamReader(System.in);
		BufferedReader br = new BufferedReader(isr);

		System.out.println("Enter First byte value");
		String num1 = br.readLine();

		System.out.println("Enter First byte value");
		String num2 = br.readLine();

		Byte b1 = Byte.valueOf(num1);
		Byte b2 = Byte.valueOf(num2);

		int flag = b1.compareTo(b2);

		if(flag>0)
			System.out.println("First object content is greater than second object");
		else if(flag<0)
			System.out.println("First object content is less than second object");
		else
			System.out.println("Content of both Bytes are equal");
	}
}
```

Program 3: Write a program to read 2 number from console using BufferedReader class and to check the content of 2 byte object is equal or not.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class Sample3 {

	public static void main(String[] args) throws IOException{
		InputStreamReader isr = new InputStreamReader(System.in);
		BufferedReader br = new BufferedReader(isr);
		System.out.println("Enter First byte value");
		String num1 = br.readLine();
		System.out.println("Enter First byte value");
		String num2 = br.readLine();
		Byte b1 = Byte.valueOf(num1);
		Byte b2 = Byte.valueOf(num2);
		boolean flag = b1.equals(b2);
		if(flag==true)
		System.out.println("Content of both Bytes are equal");
		else
		System.out.println("Content of both Bytes are not equal");
	}
}
```

Program 4: Write a program to show the usage of decode().

```
public class Sample4 {

	public static void main(String[] args) {
		String decimal = "45";
		String octal = "005";
		String hex = "0x0f";

		Byte dec=Byte.decode(decimal);
		System.out.println("decode(45) = " + dec);

		dec=Byte.decode(octal);
		System.out.println("decode(005) = " + dec);

		dec=Byte.decode(hex);
		System.out.println("decode(0x0f) = " + dec);
	}
}
```

Program 5: Write a program to show the usage of compare().

```
import java.util.Scanner;

public class Sample {
	public static void main(String[] args) {
		Scanner scanner = new Scanner(System.in);
		System.out.println("Enter First value:");
		byte b1=scanner.nextByte();

		System.out.println("Enter Second value:");
		byte b2=scanner.nextByte();

		int val=Byte.compare(b1, b2);

		if(val>0)
			System.out.println("First value greater than second value");
		else if(val<0)
			System.out.println("First value less than second value");
		else
			System.out.println("Both values are equal");
	}
}
```

Short Class:

1. The Short class wraps a value of primitive type short in an object. An object of type Short contains a single field whose type is short.
2. In addition, this class provides several methods for converting a short to a String and a String to a short, as well as other constants and methods useful when dealing with a short.

Short Class Constants:

1. **static int BYTES:**
   It is the number of bytes that is used to represent a short value in two's complement binary form. It was introduced in JDK 1.8.

2. **static short MAX_VALUE:**
   It represents a constant that can hold the maximum value of short ($2^{15} - 1$).

3. **static short MIN_VALUE:**
   It represents a constant that can hold the minimum value of short (-2^{15}).

4. **static int SIZE:**
   It is the number of bits that is used to represent a short value in two's complement binary form. It was introduced in JDK 1.5.

5. **static Class<Short> TYPE:**
   It is a Class instance that represents the primitive type short.

Short Class Constructor

1. **Short(short num):** This form of constructor accepts short number as its parameter and converts it into Short class object.
   Syntax:
   Short s1 = new Short(100);
2. **Short(String str):** This constructor accepts a parameter of String type and converts that string into Short class object.
   Syntax:
   Short s1 = new Short("100");

Short Class Methods:

1. **int compareTo(Short obj):** This method is used to compare contents of two Short class objects numerically.

2. **boolean equals(Object obj):** This method is used to compare Short object with another Short object obj. If both have the same contents, this method returns true otherwise returns false.

3. **static short parseShort(String str):** This method returns the primitive byte value contained in the string argument str.

4. **String toString():** The toString() method converts Short object into String object and returns that String object. It returns a string form of Short object.

5. **static Short valueOf(String str):** It converts string str containing some short value into Short class object and returns that Short object.
   In other words, it returns a Short object holding a value given by the specified String.

6. **static Short valueOf(short s):** This method converts the primitive short value into Short class object. In other words, it returns a Short object representing the specified short value.

7. **static int compare(short x, short y):** It is used to compare two short values numerically. It returns 0, -ve value, or +value.

8. **static Short decode(String str):** This method decodes a String into a Short.

9. **static int toUnsignedInt(short x):** This method converts short value to an int by an unsigned conversion.

10. **static long toUnsignedLong(short x):** This method converts short value to a long by an unsigned conversion.

Integer Class:

1. The Java Integer class comes under the Java.lang.Number package.
2. This class wraps a value of the primitive type int in an object. An object of Integer class contains a single field of type int value.
3. 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.

Integer Class Constant:

1. **static int MAX_VALUE:** It represents a constant that can hold the maximum value of int ($2^{31} - 1$).
2. **static int MIN_VALUE:** It represents a constant that can hold the minimum value of short (-2^{31}).
3. **static int SIZE:** It is the number of bits that are used to represent an int value in two's complement binary form. It was introduced in JDK 1.5.
4. **static Class<Short> TYPE:** It is a Class instance that represents the primitive type short. It was introduced in JDK 1.1 version.

Integer Class Constructor:

1. **Integer(int num):** This form of constructor accepts int number as its parameter and converts it into an Integer class object.

   Syntax:

   Integer i1 = new Integer(2147483644);

2. **Integer(String str):** This constructor accepts a parameter of String type and converts that string into Integer class object.

   Syntax:

   Integer i1 = new Integer("2147483644");

Integer Class Methods:

1. **static String toBinaryString(int i):** This method is used to convert a decimal integer value i into binary number system. After converting, it returns that binary number in the form of string.
2. **static String toHexString(int i):** This method is used to convert a decimal integer value i into hexadecimal number system. After converting, it returns that hexadecimal number in the form of string.
3. **static String toOctalString(int i):** This method is used to convert a decimal integer value i into octal number system. After converting, it returns that octal number in the form of a string.
4. **static int signum(int i):** This method returns the signum function of the specified int value.
5. **static int bitCount(int i):** This method is used to retrieve the number of set bits in two's complement of the specified integer.
6. **static int reverse(int i):** This method returns a primitive int value reversing the order of bits in two's complement form of the specified int value.
7. **static int reverseBytes(int i):** This method returns a primitive int value reversing the order of bytes in two's complement form of the specified int value.
8. **public static int rotateLeft(int i, int distance)**
   The **rotateLeft()** method of Java **Integer class** returns the value obtained by rotating the two's complement binary representation of the specified int value **left** by the specified number of bits. (Bits shifted out of the left hand, or high-order).

9. **public static int rotateRight(int i, int distance)**
   The **rotateRight()** method of Java **Integer class** returns the value obtained by rotating the two's complement binary representation of the specified int value **right** by the specified number of bits. (Bits shifted out of the right hand, or low-order).

10. **public static int signum(int i)**
    The signum() method of Java Integer class returns the signum function of the specified int value in the method argument. The signum function can be computed as:
    - -1, if the specified number is negative.
    - 0, if the specified number is zero.
    - 1, if the specified number is positive.

11. **public static int sum(int a, int b)**
    public static long sum(long a, long b)
    public static float sum(float a, float b)
    public static double sum(double a, double b)
    The sum() method returns the sum of its method arguments which will be specified by a user

12. **public static int max(int a, int b)**:
    Returns the greater of two int values as if by calling Math.max.

13. **public static int min(int a, int b)**:
    Returns the smaller of two int values as if by calling Math.min.

14. **public static int remainderUnsigned(int dividend, int divisor)**
    Returns the unsigned remainder from dividing the first argument by the second where each argument and the result is interpreted as an unsigned value.

Program 6: Write a program to read decimal value and convert it into Binary, Octal and Hexadecimal value.

```
import java.util.Scanner;
public class Sample {
	public static void main(String[] args) {
		Scanner scanner = new Scanner(System.in);

		System.out.println("Enter Decimal Number:");
		int x = scanner.nextInt();

		System.out.println("Binary Number:"+Integer.toBinaryString(x));
		System.out.println("Octal Number:"+Integer.toOctalString(x));
		System.out.println("Hexadecimal Number:"+Integer.toHexString(x));
	}
}
```

Program 7: Write a program to show the usage of reverse().

```
import java.util.Scanner;
public class Sample {
	public static void main(String[] args) {
		Scanner scanner = new Scanner(System.in);

		System.out.println("Enter Decimal Number:");
		int x = scanner.nextInt();

		System.out.println("Reverse Number:"+Integer.reverse(x));
	}
}
```

Program 8: Write a program to show the usage of rotateLeft().

```
import java.util.Scanner;
public class IntegerRotateLeft {
	public static void main(String[] args) {
		Scanner scanner = new Scanner(System.in);

		System.out.println("Enter Number:");
		int value = scanner.nextInt();
		System.out.println("Binary equivalent: "+Integer.toBinaryString(value));

		int Result = Integer.rotateLeft(value, 2);

		System.out.println("Value after left rotation: "+Result);
		System.out.println("New Binary value after Rotated Left:
		"+Integer.toBinaryString(Result));
	}
}
```

Program 9: Write a program to show the usage of rotateRight().

```
import java.util.Scanner;
public class IntegerRotateLeft {
	public static void main(String[] args) {
		Scanner scanner = new Scanner(System.in);

		System.out.println("Enter Number:");
		int value = scanner.nextInt();

		System.out.println("Binary equivalent: "+Integer.toBinaryString(value));

		int Result = Integer.rotateRight(value, 2);

		System.out.println("Value after left rotation: "+Result);
		System.out.println("New Binary value after Rotated Left: "+Integer.toBinaryString(Result));
	}
}
```

Program 10: Write a program to show the usage of signum().

```
public class IntegerSignum {
	public static void main(String[] args) {
		System.out.println("Result: "+Integer.signum(7));
		System.out.println("Result: "+Integer.signum(-13));
		System.out.println("Result: "+Integer.signum(0));
	}
}
```

Program 11: Write a program to show the usage of remainderUnsigned().

```
import java.util.Scanner;
public class IntegerRotateLeft {
	public static void main(String[] args) {
		Scanner scanner = new Scanner(System.in);

		System.out.println("Enter Dividend and Divisor:");
		int dividend = scanner.nextInt();
		int divisor = scanner.nextInt();

		System.out.println("Result by function:"+Integer.remainderUnsigned(dividend, divisor));
		System.out.println("Result by Operator:"+(dividend%divisor));
	}
}
```


Long Class:

Long class Constant:

1. **static int BYTES:**
   It is the number of bytes that are used to represent a long value in two's complement binary form. It was introduced in JDK 1.8 version.
2. **static long MAX_VALUE:**
   It represents a constant that can hold the maximum value of long ($2^{63} - 1$).
3. **static long MIN_VALUE:**
   It represents a constant that can hold the minimum value of long (-2^{63}).
4. **static int SIZE:**
   It is the number of bits that are used to represent a long value in two's complement binary form. It was introduced in JDK 1.5.
5. **static Class<Long> TYPE:**
   It is a Class instance that represents the primitive type long. It was introduced in JDK 1.1 version.

Long Class Constructor:

1. **Long(long num):** This form of constructor accepts long number as its parameter and converts it into a Long class object.
   Syntax:
   ```
   Long l1 = new Long(9232652545856);
   ```

2. **Long(String str):** This constructor accepts a parameter of String type and converts that string into Long class object. The string contains a long value.
   Syntax:
   ```
   Long l1 = new Long("9232652545856");
   ```

Note: **Methods available in Long class same as Integer class.**

Float Class:

- **Float class in Java** is a wrapper class that wraps (converts) a value of primitive data type "float" in an object.
- An object of Float class contains a single float type field that store a primitive float number.
- Float class was introduced in JDK 1.0. It is present in java.lang.Float package.

Float class Constructor:

There are 3 constructor available for Float class as

1. **Float(float num):** This form of constructor accepts a float number as its parameter and converts it into a Float class object.
   Syntax:
   ```
   float f = 13.44f;
   Float obj = new Float(f);
   ```

2. **Float(double num):** This form of constructor takes a double type number and then converts it into Float class object.
   Syntax:
   ```
   double d = 13.44;
   Float obj = new Float(d);
   ```

3. **Float(String str):** This constructor accepts a parameter of String type and converts that string into a Float class object. The string contains a float value.
   Syntax:
   ```
   String str = "77.789f";
   Float obj = new Float(str);
   ```

Float class methods:

The methods available in Byte, Short, Long and Integer class will also available in Float class. Here some methods which is only present in Float class as

1. **public boolean isNaN()**
   public static boolean isNaN(float v): It returns true if the specified number is a Not-a-Number(NaN) value, otherwise returns false.

2. **static boolean isFinite(float f):** This method returns true if the argument is a finite floating-point value, otherwise returns false (for NaN and infinity arguments).

3. **boolean isInfinite():** This method returns true if this Float number is infinitely large in magnitude, otherwise returns false.

4. **static boolean isInfinite(float v):** It returns true if the specified value is infinitely large in magnitude, otherwise returns false.

Program 12: Write a program to show the usage of isNaN().

```
public class Sample5 {
	public static void main(String[] args) {
		Float f1 = Float.valueOf((float) (1.0 / 0.0));
		boolean res = f1.isNaN();
		if (res)
			System.out.println(f1 + " is NaN");
		else
			System.out.println(f1 + " is not NaN");

		f1 = Float.valueOf((float) (0.0 / 0.0));
		res = f1.isNaN();
		if (res)
			System.out.println(f1 + " is NaN");
		else
			System.out.println(f1 + " is not NaN");
	}
}
```

Program 12: Write a program to Check the number is finite or not.

```
public class MyClass {
	public static void main(String[] args) {
		//creating double values
		float x = 5.2f;
		float y = 1.0f/0.0f;
		float z = Double.NaN;

		//checking double values for finite
		System.out.println("Is x finite: " + Double.isFinite(x));
		System.out.println("Is y finite: " + Double.isFinite(y));
		System.out.println("Is z finite: " + Double.isFinite(z));
	}
}
```

Program 12: Write a program to Check the number is finite or not.

```
public class MyClass {
	public static void main(String[] args) {
		//creating double values
		float x = 5.2f;
		float y = 1.0f/0.0f;
		float z = Double.NaN;
```

```
        //checking double values for finite
        System.out.println("Is x finite: " + Double.isInfinite(x));
        System.out.println("Is y finite: " + Double.isInfinite(y));
        System.out.println("Is z finite: " + Double.isInfinite(z));
    }
}
```

Double Class:

The Double class is almost same as Float class. Only difference between Double and Float class is there range.

Boolean Class:

- **Boolean class in Java** is a wrapper class that wraps (converts) a primitive "boolean" data type value in an object.
- An object of Boolean class contains a single boolean type field where we can store a primitive boolean value.
- Boolean class was introduced in JDK 1.0. It is present in java.lang.Boolean package.
- Boolean class implements Serializable, Comparable<Boolean> and constable interfaces.

Boolean class Constant:

1. **public static final** Boolean ***TRUE:*** The Boolean object corresponding to the primitive value false.
2. **public static final** Boolean ***FALSE:*** The Boolean object corresponding to the primitive value true.
3. **public static final** Class<Boolean> ***TYPE:*** It is a Class instance that represents the primitive type boolean. It was introduced in JDK 1.1 version.

Boolean class Constructor:

1. **Boolean(boolean value):** This form of constructor accepts a boolean value as its parameter and converts it into a Boolean class object.
   Syntax:
   Boolean bool = new Boolean(true);

2. **Boolean(String str):** This constructor accepts a parameter of String type and converts that string into a Boolean object. The string contains a boolean value.
   Syntax:
   Boolean bool = new Boolean("false");

Boolean class Methods:

The methods available in Byte, Short, Long and Integer class will also available in Boolean class. Here some methods which is only present in Boolean class as

1. **boolean logicalAnd(boolean a, boolean b):** This method returns the result of applying the logical AND operator to the specified boolean operands a and b.
2. **boolean logicalOr(boolean a, boolean b):** This method returns the result of applying the logical OR operator to the specified boolean operands a and b.
3. **boolean logicalXor(boolean a, boolean b):** This method returns the result of applying the logical XOR operator to the specified boolean operands a and b.

Program 13: Write a program to find the maximum of 3 numbers.

```
public class Maximum {
	public static void main(String[] args) {
		Scanner scanner = new Scanner(System.in);

		System.out.println("Enter 3 Number:");
		int x = scanner.nextInt();
		int y = scanner.nextInt();
		int z = scanner.nextInt();

		if(Boolean.logicalAnd(x>y, x>z)==true)
			System.out.println(x+" is maximum");
		else if(Boolean.logicalAnd(y>x, y>z)==true)
			System.out.println(y+" is maximum");
		else
			System.out.println(z+" is maximum");
		scanner.close();
	}
}
```

Character Class:

1. The Java Character class comes under the Java.lang package.
2. This class wraps a value of the primitive type char in an object. An object of Character class contains a single field of type char value.

Character class Constructor:

```
Character character = new Character(char value);
```

Character class Methods:

1. **Char charValue**(): Returns the value of this Character object.

2. **static int codePointAt**(char[] a, int index): Returns the code point at the given index of the char array.

3. **static int codePointAt**(char[] a, int index, int limit): Returns the code point at the given index of the char array, where only array elements with index less than limit can be used.

4. **static int codePointAt**(**CharSequence** seq, int index): Returns the code point at the given index of the CharSequence.

5. **static int codePointBefore**(char[] a, int index): Returns the code point preceding the given index of the char array.

6. **static int codePointBefore**(char[] a, int index, int start): Returns the code point preceding the given index of the char array, where only array elements with index greater than or equal to start can be used.

7. **static int codePointBefore(CharSequence** seq, int index): Returns the code point preceding the given index of the CharSequence.

8. **static int codePointCount**(char[] a, int offset, int count): Returns the number of Unicode code points in a subarray of the char array argument.

9. **static int codePointCount(CharSequence** seq, int beginIndex, int endIndex): Returns the number of Unicode code points in the text range of the specified char sequence.

10. **static int compare**(char x, char y): Compares two char values numerically.

11. **int compareTo(Character** anotherCharacter): Compares two Character objects numerically.

12. **boolean equals(Object** obj): Compares this object against the specified object.

13. **static String getName**(int codePoint): Returns the Unicode name of the specified character codePoint, or null if the code point is **unassigned**.

14. **static int getNumericValue**(char ch): Returns the int value that the specified Unicode character represents.

15. **static int getNumericValue**(int codePoint): Returns the int value that the specified character (Unicode code point) represents.

16. **static boolean isAlphabetic**(int codePoint): Determines if the specified character (Unicode code point) is an alphabet.

17. **static boolean isDigit**(char ch): Determines if the specified character is a digit.

18. **static boolean isDigit**(int codePoint):Determines if the specified character (Unicode code point) is a digit.

19. **static boolean isLetter**(char ch): Determines if the specified character is a letter.

20. **static boolean isLetter**(int codePoint): Determines if the specified character (Unicode code point) is a letter.

21. **static boolean isLetterOrDigit**(char ch): Determines if the specified character is a letter or digit.

22. **static boolean isLetterOrDigit**(int codePoint): Determines if the specified character (Unicode code point) is a letter or digit.

23. **static boolean isLowerCase**(char ch): Determines if the specified character is a lowercase character.

24. **static boolean isLowerCase**(int codePoint): Determines if the specified character (Unicode code point) is a lowercase character.

25. **static boolean isSpace**(char ch): **Deprecated.** Replaced by isWhitespace(char).

26. **static boolean isTitleCase**(char ch): Determines if the specified character is a titlecase character.

27. **static boolean isTitleCase**(int codePoint): Determines if the specified character (Unicode code point) is a titlecase character.

28. **static boolean isUpperCase**(char ch): Determines if the specified character is an uppercase character.

29. **static boolean isUpperCase**(int codePoint): Determines if the specified character (Unicode code point) is an uppercase character.

30. **static boolean isValidCodePoint**(int codePoint): Determines whether the specified code point is a valid **Unicode code point value**.

31. **static boolean isWhitespace**(char ch): Determines if the specified character is white space according to Java.

32. **static boolean isWhitespace**(int codePoint): Determines if the specified character (Unicode code point) is white space according to Java.

33. **static char toLowerCase**(char ch): Converts the character argument to lowercase using case mapping information from the UnicodeData file.

34. **static int toLowerCase**(int codePoint): Converts the character (Unicode code point) argument to lowercase using case mapping information from the UnicodeData file.

35. **String toString**(): Returns a String object representing this Character's value.

36. **static String toString**(char c): Returns a String object representing the specified char.

37. **static char toTitleCase**(char ch): Converts the character argument to titlecase using case mapping information from the UnicodeData file.

38. **static int toTitleCase**(int codePoint): Converts the character (Unicode code point) argument to titlecase using case mapping information from the UnicodeData file.

39. **static char toUpperCase**(char ch): Converts the character argument to uppercase using case mapping information from the UnicodeData file.

40. **static int toUpperCase**(int codePoint): Converts the character (Unicode code point) argument to uppercase using case mapping information from the UnicodeData file.

41. **static Character valueOf**(char c): Returns a Character instance representing the specified char value.