NUMBER

- Numbers are basic to just about any computation.

- Furthermore, the numbers are used for array indices, temperatures, salaries, ratings, and an infinite variety of things.

- Java has represent numbers as follows:
    - built-in or primitive types
    - wrapper (object) types

Built-in type	**Object wrapper**	**Size of built-in (bits)**	**Contents**
byte	Byte	8	Signed integer
short	Short	16	Signed integer
int	Integer	32	Signed integer
long	Long	64	Signed integer
float	Float	32	IEEE-754 floating point
double	Double	64	IEEE-754 floating point
n/a	BigInteger	UnlimitedArbitrary-size	immutable integer value
n/a	BigDecimal	Unlimited Arbitrary-size and-precision	immutable floating-point value

Wrapper Classes in Java

- A Wrapper class is a class whose object wraps or contains a primitive data types.

- When we create an object to a wrapper class, it contains a field and in this field, we can store a primitive data types.

- In other words, we can wrap a primitive value into a wrapper class object.

- The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive*.

    - **Autoboxing:** Automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing. For example – conversion of int to Integer, long to Long, double to Double etc.

    - **Unboxing:** It is just the reverse process of autoboxing. Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing. For example – conversion of Integer to int, Long to long, Double to double etc.

Need of Wrapper Classes

- **Change the value in Method:** Java supports only call by value. So, if we pass a primitive value, it will not change the original value. Hence, convert primitive data types into objects through the wrapper classes. Objects are needed if we wish to modify the arguments (i.e., original value) passed into a method.

- **Serialization:** We need to convert the objects into streams to perform the serialization.

    - A *stream* is a sequence of objects that supports various methods which can be pipelined to produce the desired result.

- **Synchronization:** Java synchronization works with objects in Multithreading.

- **java.util package:** The java.util package provides the utility classes to deal with objects.

- **Collection Framework:** Java collection framework works with objects only. All classes of the collection framework (ArrayList, LinkedList, Vector, HashSet, Linked HashSet, TreeSet, Priority Queue, Array Deque, etc.) deal (i.e., store) only withobjects (reference types) and not primitive types.

Constructors Of Wrapper Classes In Java:

Every wrapper class in java has two constructors,

1. First constructor takes corresponding primitive data as an argument
2. Second constructor takes string as an argument.

Notes :

- The string passed to second constructor should be parse-able to number, otherwise you will get run time NumberFormatException.

- Wrapper Class Character has only one constructor which takes char type as an argument. It doesn’t have a constructor which takes String as an argument. Because, String can not be converted into Character.

- Wrapper class Float has three constructors. The third constructor takes double type as an argument.

```
public class Number {

        public static void main(String[] args) {

                Byte B1 = new Byte((byte) 10);   //Constructor which takes byte
                                                   value as an argument
                Byte B2 = new Byte("10");        //Constructor which takes
                                                  String as an argument

                //Byte B3 = new Byte("abc");     //Run Time Error :
                                                  NumberFormatException
                                                 //Because, String abc cann't be
                                                  parse-able to byte

                Short S1 = new Short((short) 20);   //Constructor which takes short
                                                    value as an argument
                Short S2 = new Short("20");         //Constructor which takes
                                                    String as an argument

                Integer I1 = new Integer(30);          //Constructor which takes
                                                       int value as an argument
                Integer I2 = new Integer("30");        //Constructor which takes
                                                       String as an argument
        Long L1 = new Long(40);              //Constructor which takes
                                              long value as an argument
        Long L2 = new Long("40");            //Constructor which takes
                                             String as an argument

        Float F1 = new Float(12.2f);         //Constructor which takes
                                              float value as an argument
        Float F2 = new Float("15.6");         //Constructor which takes
                                             String as an argument
        Float F3 = new Float(15.6d);         //Constructor which takes
                                             double value as an argument

        Double D1 = new Double(17.8d);       //Constructor which takes
                                             double value as an argument
        Double D2 = new Double("17.8");     //Constructor which takes
                                             String as an argument

        Boolean BLN1 = new Boolean(false);   //Constructor which takes
                                           boolean value as an argument
        Boolean BLN2 = new Boolean("true");   //Constructor which takes
                                               String as an argument

        Character C1 = new Character('D');  //Constructor which takes
                                             char value as an argument
        Character C2 = new Character("abc");  //Compile time error :
                                               String abc can not be
                                               converted to character
    }

}
```

Parsing Methods Of Wrapper Classes In Java

- All wrapper classes in java have methods to parse the given string to corresponding primitive data provided the string should be parse-able.

- If the string is not parse-able, you will get NumberFormatException.

- All parsing methods of wrapper classes are static i.e., you can refer them directly using class name.

Example:

```
public class Number {
	public static void main(String[] args) {

		byte b = Byte.parseByte("10");
		System.out.println(b);   //Output : 10

		short s = Short.parseShort("25");
		System.out.println(s);   //Output : 25

		int i = Integer.parseInt("123");
		System.out.println(i);   //Output : 123

		long l = Long.parseLong("100");
		System.out.println(l);   //Output : 100

		float f = Float.parseFloat("12.35");
		System.out.println(f);   //Output : 12.35

		double d = Double.parseDouble("12.87");
		System.out.println(d);   //Output : 12.87

		boolean bln = Boolean.parseBoolean("true");
		System.out.println(bln);   //Output : true

		boolean bln1 = Boolean.parseBoolean("abc");
		System.out.println(bln1);   //Output : false

		//char c = Character.parseChar("abc");   //compile time error
		//parseChar() is not defined for Character wrapper class
		}
}
```

Output:

```
10
25
123
100
12.35
12.87
true
false
```

Methods of wrapper class

- The most common methods of the Integer wrapper class are summarized in below table. Similar methods for the other wrapper classes are found in the Java API documentation.

Method	Purpose
parseInt(s)	returns a signed decimal integer value equivalent to string s
toString(i)	returns a new String object representing the integer i
byteValue()	returns the value of this Integer as a byte
doubleValue()	returns the value of this Integer as a double
floatValue()	returns the value of this Integer as a float
intValue()	returns the value of this Integer as an int
shortValue()	returns the value of this Integer as a short
longValue()	returns the value of this Integer as a long
int compareTo(int i)	Compares the numerical value of the invoking object with that of i. Returns 0 if the values are equal. Returns a negative value if the invoking object has a lower value. Returns a positive value if the invoking object has a greater value.
static int compare(int num1, int num2)	Compares the values of num1 and num2. Returns 0 if the values are equal. Returns a negative value if num1 is less than num2. Returns a positive value if num1 is greater than num2.
boolean equals(Object, intObj)	Returns true if the invoking Integer object is equivalent to intObj. Otherwise, it returns false.

Example:

```
public class Number {

        public static void main(String[] args) {

                Integer intObj1 = new Integer (25);
                Integer intObj2 = new Integer ("25");
                Integer intObj3= new Integer (35);
                //compareTo demo
                System.out.println("Comparing by using compareTo Obj1
                    and Obj2: " +  intObj1.compareTo(intObj2));
                System.out.println("Comparing by using compareTo Obj1
                    and Obj3: " + intObj1.compareTo(intObj3));

                //Equals demo
                System.out.println("Comparing by using equals Obj1 and
                            Obj2: " + intObj1.equals(intObj2));
                System.out.println("Comparing by using equals Obj1 and
                            Obj3: " + intObj1.equals(intObj3));

                Float f1 = new Float("2.25f");
                Float f2 = new Float("20.43f");
                Float f3 = new Float(2.25f);
                System.out.println("Comparing by using compare f1 and f2: "
                                    + Float.compare(f1,f2));
                System.out.println("Comparing by using compare f1 and f3: "
                                    + Float.compare(f1,f3));

                //Addition of Integer with Float
                Float f = intObj1.floatValue() + f1;
                System.out.println("Addition of intObj1 and f1: "+
                                        intObj1 +"+" +f1+"=" +f );

        }
}
```

Output:

```
Comparing using compareTo Obj1 and Obj2: 0
Comparing using compareTo Obj1 and Obj3: -1
Comparing using equals Obj1 and Obj2: true
Comparing using equals Obj1 and Obj3: false
Comparing using compare f1 and f2: -1
Comparing using compare f1 and f3: 0
Addition of intObj1 and f1: 25+2.25=27.25
```

Converting Numbers to Objects and Vice Versa

- The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive*.

    - **Autoboxing:** Automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing. For example – conversion of int to Integer, long to Long, double to Double etc.

    - **Unboxing:** It is just the reverse process of autoboxing. Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing. For example – conversion of Integer to int, Long to long, Double to double etc.

Example: Auto-boxing and Auto-unboxing

```
  public class Number {

    public static void main(String[] args) {

        System.out.println("Converting Primitive type to Object type");
        Integer IObj = 10;              //Auto-boxing
        System.out.println(IObj);
        Float FObj = 10.6f;             //Auto-boxing
        System.out.println(FObj);
        Double DObj = 12.5;             //Auto-boxing
        System.out.println(DObj);

        System.out.println();

        System.out.println("Converting Object type to Primitive type");
        int iObj = IObj;                //Auto-unboxing
        System.out.println(iObj);
        float fObj = FObj;              //Auto-unboxing
        System.out.println(fObj);
        double dObj = DObj;             //Auto-unboxing
        System.out.println(dObj);

    }
}
```

Output:

```
Converting Primitive type to Object type
10
10.6
12.5

Converting Object type to Primitive type
10
10.6
12.5
```

Example: Explicitly convert between an int and an Integer object, or vice versa, by using the wrapper class methods.

```
    public class Number {

        public static void main(String[] args) {

                        // int to Integer
                        // Integer i1 = new Integer(42);
                        Integer i1 = Integer.valueOf(42);
                        System.out.println(i1.toString()); // or just i1

                        // Integer to int
                        int i2 = i1.intValue();
                        System.out.println(i2);
        }
}
```

Output:

```
42
42
```

Example: Shows autoboxing (in the call to foo(i), i is wrapped automatically) and auto-unboxing (the return value is automatically unwrapped).

```
public class Number {

        public static void main(String[] args) {

                int i = 42;
                int result = foo(i);                //Auto-unboxing
                System.out.println(result);
        }

        public static Integer foo(Integer i) {  //Auto-boxing

                System.out.println("Object = " + i);
                Integer sb = new Integer(123);
                // Integer sb = Integer.valueOf(123)
                return sb;
        }

}
```

Output:

```
Object = 42
123
```

Checking Whether a String Is a Valid Number

Example: Write a program which read a sting and convert it to integer number.

```
import java.util.Scanner;

public class Number {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.println("Enter a number");
        String aNumber = sc.next();
        int result;

        try {
            result = Integer.parseInt(aNumber);
            System.out.println("The " + result + " is a valid integer.");
        }
        catch(NumberFormatException exc) {

            System.out.println("The " + aNumber + " is an invalid
                                                  integer.");
            return;
        }
        }
}
```

Output:

```
Enter a number
12345
The 12345 is a valid integer.

Enter a number
1234.67
The 1234.67 is an invalid integer.
```

Example: Write a program which read a string and convert it to double number.

```
import java.util.Scanner;

public class Number {

        public static void main(String[] args) {

                Scanner sc=new Scanner(System.in);
                System.out.println("Enter a number");
                String aNumber = sc.next();
                double result;

                try {
                        result = Double.parseDouble(aNumber);
                        System.out.println("The " + result + " is a valid double.");
                }
                catch(NumberFormatException exc) {

                        System.out.println("The " + aNumber + " is an invalid
                                                              double.");
                        return;
                }
        }
}
```

Output:

```
Enter a number
1234.67
The 1234.67 is a valid double.
```

Storing a Larger Number in a Smaller Number

Create a float variable and assign it 3.0 to it and observe what happen.

```
        float f = 3.0;
```

if you had written:

```
        double tmp = 3.0;
        float f = tmp;                  //won't even compile!
```

It can be fixedby any one of several ways:

```
        1) By making the 3.0 a float (probably the best solution)
                float f = 3.0f; // or just 3f

        2) By making f a double
                double f = 3.0;
                float f = (float)3.0;// By putting in a cast

        3) By assigning an integer value of 3, which will get "promoted"
                float f = 3;
```

Example: Write a program which convert double number to integer and byte type.

```
public class Number {

        public static void main(String[] args) {

                int i;
                double j = 2.75;
                //i = j; // EXPECT COMPILE ERROR
                i = (int) j;            // with cast; i gets 2
                System.out.println("i =" + i);

                byte b;
                //b = i; // EXPECT COMPILE ERROR
                b = (byte) i;           // with cast, i gets 2
                System.out.println("b =" + b);
        }
}
```

Output:

```
i = 2
b = 2
```

Taking a Fraction of an Integer Without Using Floating Point

suppose you want to multiply, **0.666 * 5** then

- multiply an integer by a fraction without converting the fraction to a floating-point number.
- Multiply the integer by the numerator and divide by the denominator.
- integers and floating-point numbers are stored differently.
- for efficiency purposes, to multiply an integer by a fractional value without converting the values to floating point and back.
- it doesn't require a “cast”.

Example:

```
Public class Number {

    Public static void main(String[] args) {

        double d1 = 0.666 * 5;          //fast but obscure and inaccurate
        System.out.println(d1);

        double d2 = 2/3 * 5;            // convert 0.666 to 2/3
        System.out.println(d2);         //wrong answer,  2/3 = 0, 0*5 = 0

        double d3 = 2d/3d * 5;          // "normal"
        System.out.println(d3);

        double d4 = (2*5)/3d;           // one step done as integers,
        System.out.println(d4);         //almost same answer

        int i5 = 2*5/3;                 // fast, approximate integer answer
        System.out.println(i5);
        }
}
```

Output:

```
3.3
0.0
3.333333333333333
3.3333333333333335
3
```

Ensuring the Accuracy of Floating-Point Numbers

- In java, integer division by 0 consider as logical error so it throws an ArithmeticException.

- floating-point computation generated a sensible result

- Floating-point operations, however, do not throw an exception because they are defined over an (almost) infinite range of values.

    - if you divide a positive floating-point number by zero, then Java signal errors by producing the constant POSITIVE_INFINITY.

    - if you divide a negative floating-point value by zero, then Java signal errors by producing the constant NEGATIVE_INFINITY.

    - Otherwise generate an invalid result NaN (Not a Number)

- Values for these three public constants are defined in both the Float and the Double wrapper classes.

- The value NaN has the unusual property that it is not equal to itself (i.e., NaN != NaN).

- x==NaN never be true,

- To check particular value is NaN or not, the methods Float.isNaN(float) and Double.isNaN(double) must be used.

Example:

```
public class Number {

        public static void main(String[] args) {

                double d = 123;
                double e = 0;
                if (d/e == Double.POSITIVE_INFINITY)
                        System.out.println("Check for POSITIVE_INFINIT");

                double s = Math.sqrt(-1);
               System.out.println("The s = " + s);

                if (s == Double.NaN)
                        System.out.println("Comparison with NaN incorrectly
                                                                        returns true");
                if (Double.isNaN(s))
                        System.out.println("Double.isNaN() correctly returns true");
        }
}
```

Output:

```
Check for POSITIVE_INFINITY works
The s = NaN
Double.isNaN() correctly returns true
```

Example: Calculate the area of a triangle by using Heron’s formula for both in float and in double.

```
public class Number {

        public static void main(String[] args) {

                float af, bf, cf;
                float sf, areaf;

                double ad, bd, cd;
                double sd, aread;

                // Area of triangle in float
                af = 12345679.0f;
                bf = 12345678.0f;
                cf = 1.01233995f;

                sf = (af+bf+cf)/2.0f;
                System.out.println("sf = " + sf);
                areaf = (float)Math.sqrt(sf * (sf - af) * (sf - bf) * (sf - cf));
                System.out.println("Single precision: " + areaf);


                // Area of triangle in double
                ad = 12345679.0;
                bd = 12345678.0;
                cd = 1.01233995;

                sd = (ad+bd+cd)/2.0d;
                System.out.println("sd = " + sf);
                aread = Math.sqrt(sd * (sd - ad) * (sd - bd) * (sd - cd));
                System.out.println("Double precision: " + aread);
        }
}
```

Output:

```
sf = 1.2345679E7
Single precision: 0.0                          //result is incorrect
sd = 1.2345679E7
Double precision: 972730.0557076167            //result is correct
```

The double values are correct, but the floating-point value comes out as zero due to rounding errors. This happens because, in Java, operations involving only float values are performed as 32-bit calculations. Related languages such as C automatically promote these to double during the computation, which can eliminate some loss of accuracy.

Comparing Floating-Point Numbers

- Compare two floating-point numbers for equality.
    - The equals() method of Float and Double class returns true if the two values are the same bit for bit (i.e., if and only if the numbers are the same or are both NaN).

- To actually compare floating-point numbers for equality, it is generally desirable to compare them within some tiny range of allowable differences; this range is often regarded as a tolerance or as epsilon.

Example:

```
public class Number {

    final static double EPSILON = 0.0000001;

    /** Compare two doubles within a given epsilon */
    Public static boolean equals(double a, double b, double eps) {
        if (a==b)
            return true;
        // If the difference is less than epsilon, treat as equal.
        return Math.abs(a - b) < eps;
    }

    /** Compare two doubles, using default epsilon */
    public static boolean equals(double a, double b) {
        return equals(a, b, EPSILON);
    }

    public static void main(String[] args) {

        double da = 3 * .3333333333;
        double db = 0.99999992857;

        // Compare two numbers that are expected to be close.
        if (da == db) {
            System.out.println("Java considers " + da + "==" + db);
        }

        // else compare with our own equals overload
        elseif (equals(da, db, EPSILON)) {
            System.out.println("Equal within epsilon " + EPSILON);
        }
        else {
            System.out.println(da + " != " + db);
        }
    }
}
```

Output:

```
Equal within epsilon 1.0E-7
```

Rounding Floating-Point Numbers

- To round floating-point numbers properly, use Math.round().
- It has two overloads:
  - if you give it a double, you get a long type result.
  - if you give it a float, you get an int type result.

Example:

```
public class Number {

        public static void main(String[] args) {

            double d=5.67;
            System.out.println("The rounding of " + d + " is " + Math.round(d));

            float f=9.4255f;
            System.out.println("The rounding of " + f + " is " + Math.round(f));
        }
}
```

Output:

```
The rounding of 5.67 is 6
The rounding of 9.4255 is 9
```

Example: Rounding the number from 0.1, 0.15, 0.2, 0.25, ..., 0.95.

```
public class Number {

      public static void main(String[] args) {

          for (double d = 0.1; d<1.0; d+=0.05) {

             System.out.println("The rounding of " + d + " is " + Math.round(d));
          }
```

```
        }
}
```

Output:

```
The rounding of 0.1 is 0
The rounding of 0.15000000000000002 is 0
The rounding of 0.2 is 0
The rounding of 0.25 is 0
The rounding of 0.3 is 0
The rounding of 0.35 is 0
The rounding of 0.39999999999999997 is 0
The rounding of 0.44999999999999996 is 0
The rounding of 0.49999999999999994 is 0
The rounding of 0.5499999999999999 is 1
The rounding of 0.6 is 1
The rounding of 0.65 is 1
The rounding of 0.7000000000000001 is 1
The rounding of 0.7500000000000001 is 1
The rounding of 0.8000000000000002 is 1
The rounding of 0.8500000000000002 is 1
The rounding of 0.9000000000000002 is 1
The rounding of 0.9500000000000003 is 1
```

Formatting Numbers

- For formatting numbers, JAVA use a NumberFormat subclass.

- NumberFormat class is present in java.text package (i.e., import java.text.numberFormat) and it is an abstract class.

- A DecimalFormat object appropriate to the user's locale can be obtained from the factory method NumberFormat.getInstance() and manipulated using set methods.

- *Creating a NumberFormat object*

    - NumberFormat nf = NumberFormat.getInstance()

- *Some important Set Methods of NumberFormat class*

    - void setMaximumFractionDigits(int newValue)

        - Sets the maximum number of digits allowed in the fraction portion of a number.

    - void setMaximumIntegerDigits(int newValue)
        - Sets the maximum number of digits allowed in the integer portion of a number.

    - void setMinimumFractionDigits(int newValue)
        - Sets the minimum number of digits allowed in the fraction portion of a number.

    - void setMinimumIntegerDigits(int newValue)
        - Sets the minimum number of digits allowed in the integer portion of a number.

Example: Write a program to print a double number having maximum fraction digit 2.

```
import java.text.NumberFormat;

public class NumberFormatTest {

	public static void main(String[] args) {

		NumberFormat nf=NumberFormat.getInstance();
		nf.setMaximumFractionDigits(2);

		double d=123.456;
		//The double number having maximum fraction digit 2.
		System.out.println("After formatting the number " + d + " is "
							+  nf.format(d));
	}
}
```

Output:

```
After formatting the number 123.456 is 123.46
```

Example: Write a program to print a double number having maximum fraction digit 4 and minimum fraction digit 2.

```
import java.text.NumberFormat;

public class NumberFormatTest {
        public static void main(String[] args) {

                NumberFormat nf = NumberFormat.getInstance();
                nf.setMaximumFractionDigits(4);
                nf.setMinimumFractionDigits(2);

                double d=123.4;
                System.out.println("After formatting the number " + d + " is "
                                                        +  nf.format(d));
                double e=12.14567;
                System.out.println("After formatting the number " + e + " is "
                                                        +  nf.format(e));
        }
}
```

Output:

```
After formatting the number 123.4 is 123.40
After formatting the number 12.14567 is 12.1457
```

Example: Write a program to set minimum integer digit to 3, maximum fraction digit to 4 and minimum fraction digit to 2 of a decimal number.

```
import java.text.NumberFormat;

public class NumberFormatTest {

        public static void main(String[] args) {

                NumberFormat nf=NumberFormat.getInstance();
                nf.setMinimumIntegerDigits(3);
                nf.setMaximumFractionDigits(4);
                nf.setMinimumFractionDigits(2);

                double a=12.4;
                double b=121.14567;

                System.out.println("After formatting the number " + a + " is "
                                                        +  nf.format(a));
                System.out.println("After formatting the number " + b + " is "
                                                        +  nf.format(b));
        }
}
```

Output:

```
After formatting the number 12.4 is 012.40
After formatting the number 121.14567 is 121.1457
```

Example:

```
import java.text.NumberFormat;

public class NumberFormatTest {

	public static void main(String[] args) {

		final double data[] = {0, 1, 22d/7, 100.2345678};

		NumberFormat nf=NumberFormat.getInstance();

		nf.setMinimumIntegerDigits(3);
		nf.setMinimumFractionDigits(2);
		nf.setMaximumFractionDigits(4);

		// Now print using it.
		for (int i=0; i<data.length; i++) {
		   System.out.println(data[i] + "\tformats as " + nf.format(data[i]));
		}
	}
}
```

Output:

```
0.0	formats as 000.00
1.0	formats as 001.00
3.142857142857143	formats as 003.1429
100.2345678	formats as 100.2346
```

Changing the pattern dynamically

You can also construct a DecimalFormat with a particular pattern or change the pattern dynamically using applyPattern() .

Table 5-2. DecimalFormat pattern characters	
Character	**Meaning**
#	A digit, leading zeroes are omitted.
0	A digit - always displayed, even if number has less digits (then 0 is

	displayed)
.	Locale-specific decimal separator (decimal point)
,	Locale-specific grouping separator (comma in English)
-	Locale-specific negative indicator (minus sign)
%	Shows the value as a percentage
;	Separates two formats: the first for positive and the second for negative values
’	Escapes one of the above characters so it appears
Anything else	Appears as itself

Example: Write a program to print a double number in the format, 3 digit in hole part and decimal number then two digit in fraction part.

```
import  java.text.NumberFormat;
import  java.text.DecimalFormat;

public class NumberFormatTest {

        public static void main(String[] args) {

                NumberFormat our = new DecimalFormat("###.##");
                double d=123.345;
                System.out.println(d + "\tformats as " + our.format(d));
        }
}
```

Output:

```
123.34
```

check where,

Input: 12.456   Output: 12.46
Input: 12345.47789   Output: 12345.48
Input: .345   Output: 0.34

Example: Write a program to print a double number in the format, 4 digit before decimal(if number of digit is less print zero) and 3 digit after decimal.

```
import java.text.NumberFormat;
import java.text.DecimalFormat;
```

```
public class NumberFormatTest {

        public static void main(String[] args) {

                NumberFormat our = new DecimalFormat("0000.###");
                //NumberFormat our = new DecimalFormat("0000.000");
                double d=1234.5678;
                double e=12.5678;
                double f=12.5;
                //double g=7.5;
                System.out.println(d + "\tformats as " + our.format(d));
                System.out.println(e + "\tformats as " + our.format(e));
                System.out.println(f + "\tformats as " + our.format(f));
                //System.out.println(g + "\tformats as " + our.format(g));
                                                        //Output: 0007.500
        }
}
```

Output:

```
1234.5678    formats as 1234.568
12.5678      formats as 0012.568
12.5    formats as 0012.5
```

Example: Write a program to print a decimal number grouping it to 2 digit at a time.

```
import java.text.NumberFormat;
import java.text.DecimalFormat;

public class NumberFormatTest {

        public static void main(String[] args) {
                NumberFormat our = new DecimalFormat("##,##.##");
                double d=1245677.5566;
                System.out.println(d + "\tformats as " + our.format(d));
        }
}
```

Output:

```
1245677.5566       formats as 1,24,56,77.56
```

Example: Write a program to print percentage of a double number.

```
import  java.text.NumberFormat;
import  java.text.DecimalFormat;

public class NumberFormatTest {

        public static void main(String[] args) {

                NumberFormat our = new DecimalFormat("%##.##");
                //Write a program to print a double number -ve sign
                                                        assigned to it.
                //NumberFormat our = new DecimalFormat("-##.##");
                double d=657.5566;
                System.out.println(our.format(d));
        }
}
```

Output:

```
%65755.66
```

Converting Between Binary, Octal, Decimal, and Hexadecimal

- **Integer.parseInt(***String input, int radix***)** to convert from any type of number to an Integer object.
- **Integer.valueOf(***String input, int radix***)** is an inbuilt method which convert any type of number to returns an Integer object.
- **Integer.toString(***int input, int radix***)** to convert from integer to any type. The return type is string.
  - There are also specialized versions of toString(int) that don't require you to specify the radix; for example, **toBinaryString()** to convert an integer to binary, **toHex-String()** for hexadecimal, and so on.

Example: Convert the string "1010" to decimal number. here the redix are binary, octal, hexadecimal. Hint: by using Integer.parseInt(String input, int radix)

```
import java.text.NumberFormat;
import java.text.DecimalFormat;

public class NumberFormatTest {

        public static voidmain(String[] args) {

                String str="1010";
                Integer iObj=Integer.parseInt(str,2);
                System.out.println("1010 in base 2 is "+iObj);
                Integer iObj1=Integer.parseInt(str,8);
                System.out.println("1010 in base 8 is "+iObj1);
                Integer iObj2=Integer.parseInt(str,16);
                System.out.println("1010 in base 16 is "+iObj2);
        }
}
```

Output:

```
1010 in base 2 is 10
1010 in base 8 is 520
1010 in base 16 is 4112
```

Example: Convert the number 42 to the binary, octal, hexadecimal and decimal number using Integer.toString(int input, int radix).

```
import java.text.NumberFormat;
import java.text.DecimalFormat;

public class NumberFormatTest {

        public static voidmain(String[] args) {
                int i=42;
                String res1=Integer.toString(i,2);
                String res2=Integer.toString(i,8);
                String res3=Integer.toString(i,16);
                String res4=Integer.toString(i,10);
                System.out.println("42 in base 2 is "+res1);
                System.out.println("42 in base 8 is "+res2);
                System.out.println("42 in base 16 is "+res3);
                System.out.println("42 in base 10 is "+res4);

                String res5=Integer.toBinaryString(i);
                System.out.println("42 in base 2 is "+res5);
        }
}
```

Output:

```
42 in base 2 is 101010
42 in base 8 is 52
42 in base 16 is 2a
42 in base 10 is 42
42 in base 2 is 101010
```

Example:

```
import java.text.NumberFormat;
import java.text.DecimalFormat;

publicclass NumberFormatTest {

        public static voidmain(String[] args) {
                String input = "101010";
                for (int radix : new int[] { 2, 8, 10, 16, 36 }) {
                System.out.print(input + " in base " + radix + " is " +
                                Integer.valueOf(input, radix) + "; \t");

                Int j = 42;
                System.out.println(j + " formatted in base " + radix + " is "
                                        + Integer.toString(j, radix));
                }
        }
}
```

Output:

```
101010 in base 2 is 42;            42 formatted in base 2 is 101010
101010 in base 8 is 33288;                 42 formatted in base 8 is 52
101010 in base 10 is 101010;    42 formatted in base 10 is 42
101010 in base 16 is 1052688;  42 formatted in base 16 is 2a
101010 in base 36 is 60512868;         42 formatted in base 36 is 16
```

Operating on a Series of Integers

- Need to work on a range of integers.
    - If the set of numbers are contiguous, use a for loop.

- If the ranges of numbers are discontinuous, use a java.util.BitSet class.

Example: For a contiguous set, use for loop.

```
public class Number {

     protected static String months[] = {"January", "February",  "March",
     "April", "May", "June", "July", "August", "September","October",
     "November", "December"};

     publicstaticvoidmain(String[] args) {

           // When you want a set of array indices, use a for loop starting at 0.
          for (int i = 0; i<months.length; i++)
                         System.out.println("Month " + months[i]);

          System.out.println(" ");

          // When you want an ordinal list of numbers, use a for loop starting at 1.
          for (int i = 1; i<= months.length; i++)
                    System.out.println("Month # " + i);

            System.out.println(" ");

          // For e.g., counting by 3 from 11 to 27, use a for loop
           for (int i = 11; i<= 27; i += 3)
                         System.out.println("i = " + i);
      }
}
```

Output:

```
Month January
Month February
Month March
Month April
Month May
Month June
Month July
Month August
Month September
Month October
Month November
Month December
Month # 1
Month # 2
Month # 3
Month # 4
Month # 5
Month # 6
Month # 7
Month # 8
Month # 9
Month # 10
Month # 11
Month # 12
```

```
i = 11
i = 14
i = 17
i = 20
i = 23
i = 26
```

Example: For discontinuous ranges of numbers, use a java.util.BitSet class.

- **BitSet** is a class defined in the java.util package. It creates an array of bits represented by boolean values.

- The size of the array is flexible and can grow to accommodate additional bit as needed.

- As it is an array, the index is zero-based and the bit values can be accessed only by non-negative integers as an index .

- The default value of the BitSet is boolean false with a representation as 0 (off).

- **BitSet() :** A no-argument constructor to create an empty BitSet object.

- BitSet uses about 1 bit per boolean value.

- **set(int Index) :** This method sets the bit at the specified index to true i.e adds a value.

- To access a specific value in the BitSet, the **get(int index)** method is used with an integer argument as an index.

```
import java.util.BitSet;

publicclass Number {

    protected static String months[] = {"January", "February",  "March",
    "April", "May", "June", "July", "August", "September","October",
    "November", "December"};

    public static void main(String[] args) {

        // Create a BitSet and turn on a couple of bits.
        BitSet b = new BitSet();
        b.set(0); // January
        b.set(3); // April
        b.set(8); // September

        // Presumably this would be somewhere else in the code.
        for (int i = 0; i<months.length; i++) {
            if (b.get(i))
                System.out.println("Month " + months[i]);
        }

        System.out.println();

        // Same example using an array
        int[] numbers = {0, 3, 8};

        // Presumably this would be somewhere else in the code.
        for (int n : numbers) {
            System.out.println("Month: " + months[n]);
        }
    }
}
```

Output:

```
Month January
Month April
Month September

Month: January
```

```
Month: April
Month: September
```

Formatting with Correct Plurals

- The examples of use of correct plurals are
  - 0 Books or Zero Books
  - 1 Book or One Book
  - 2 Books or Two Books
- We can solve the problem by using one of the following method
  - Using **conditional statement**, or
  - Using **ChoiceFormat**

Using conditional statement

Example: Using conditional statement

```
public class Number {

        publicstaticvoidmain(String[] args) {

                //int n=0;
                int n=1;
                //int n=2;
                System.out.println("I read " + n + " Book" + (n==1? "." : "s."));
        }
}
```

Output:

```
I read 1 Book.
```

Using ChoiceFormat

- Using *ChoiceFormat with pluralizedFormat*
  - ChoiceFormat accepts two arrays

  - It provides pluralized word

- Using *ChoiceFormat with quantizedFormat*
  - ChoiceFormat accepts string-based pattern
  - English text version of quantity

Example: Using ChoiceFormat with pluralizedFormat

```
import java.text.ChoiceFormat;

public class Number {

        // ChoiceFormat to just give pluralized word
        static double[] limits = { 0, 1, 2 };

        static String[]  formats = { "reviews", "review", "reviews"};

        static ChoiceFormat pluralizedFormat = new
                                                ChoiceFormat(limits, formats);

        Static int[] data = { -2,-1, 0, 1, 2, 3};

        public static void main(String[] args) {

                System.out.println("Pluralized Format");

                for (int i : data) {
                        System.out.println("Found " + i + " " +
                                                pluralizedFormat.format(i));
                }
        }
}
```

Output:

```
Pluralized Format
Found -2 reviews
Found -1 reviews
Found 0 reviews
Found 1 review
Found 2 reviews
Found 3 reviews
```

Example: Using ChoiceFormat with quantizedFormat

```
import java.text.ChoiceFormat;

public class Number {

        // ChoiceFormat to give English text version, quantified
        static ChoiceFormat quantizedFormat = new ChoiceFormat(
                        "0#no reviews | 1#one review | 1<many reviews");

        staticint[] data = { -2,-1, 0, 1, 2, 3 };

        public static void main(String[] args) {

                System.out.println("Quantized Format");

                for (int i : data) {
                        System.out.println("Found " +
                                                quantizedFormat.format(i));
                }
        }
}
```

Output:

```
Quantized Format
Found no reviews
Found no reviews
Found no reviews
Found one review
Found many reviews
Found many reviews
```

Generating Random Numbers

Java provides 2 techniques to generate random numbers using some built-in methods and classes as listed below:

- *Math.random method*
- *java.util.Random class*

Math.random method :

- This technique can generate random numbers of double type.
- It is a positive sign, greater than or equal to 0.0 and less than 1.0.
- Use java.lang.Math.random()

Example:

```
public class Number {

        public static void main(String[] args) {

                System.out.println("A random from java.lang.Math is " +
                                                            Math.random( ));
                System.out.println("A random from java.lang.Math is " +
                                                            Math.random( ));
        }
}
```

Output:

```
A random from java.lang.Math is 0.5329005395702993
A random from java.lang.Math is 0.7750836085959537
```

java.util.Random

- We can generate random numbers of types integers, float, double, long, booleans using this class.
- To generate random numbers
  - first create an instance of this class
  - then invoke methods such as nextInt(), nextDouble(), nextLong() etc using that instance.
- We can pass arguments to the methods for placing an upper bound on the range of the numbers to be generated.
  - For example, nextInt(6) will generate numbers in the range 0 to 5 both inclusive.

Example:

```java
import java.util.Random;

public class Number {

        public static void main(String[] args) {

                Random rand = new Random();

                // Generate random integers in range 0 to 999
                int rand_int1 = rand.nextInt(1000);
                int rand_int2 = rand.nextInt(1000);

                // Print random integers
                System.out.println("Random Integers: "+rand_int1);
                System.out.println("Random Integers: "+rand_int2);

                // Generate Random doubles
                double rand_dub1 = rand.nextDouble();
                double rand_dub2 = rand.nextDouble();

                // Print random doubles
                System.out.println("Random Doubles: "+rand_dub1);
                System.out.println("Random Doubles: "+rand_dub2);

                boolean rand_bool1 = rand.nextBoolean();
                System.out.println("Random Booleans: "+ rand_bool1);

        }
}
```

Output:

```
Random Integers: 730
Random Integers: 751
Random Doubles: 0.7653875428555855
Random Doubles: 0.013700301907223489
Random Booleans: false
```

Example:write a program to generate 3 double type random numbers and 3 integer type random numbers.

```
Import java.util.Random;

public class Number {

        public static void main(String[] args) {

                Random r = new Random();

                for (inti=0; i<3; i++)
                        System.out.println("A double from
                                        java.util.Random is " + r.nextDouble());

                for (inti=0; i<3; i++)
                        System.out.println("An integer from
                                                java.util.Random is " + r.nextInt());
        }
}
```

Output:

```
A double from java.util.Random is 0.3318176839519682
A double from java.util.Random is 0.6198671474168848
A double from java.util.Random is 0.10231027097147605
An integer from java.util.Random is 926712278
An integer from java.util.Random is -1110763465
An integer from java.util.Random is 2107995229
```

Example: write a program to generate 5 Gaussian random numbers.

```
import java.util.Random;

public class Number {

        public static void main(String[] args) {

                Random r  = new Random();

                for (inti = 0; i< 5; i++)
                        System.out.println("A gaussian random double is " +
                                                                r.nextGaussian());
        }
}
```

Output:

```
A gaussian random double is -0.12654554740642937
A gaussian random double is -1.3442928221637291
A gaussian random double is -0.2011491511058577
A gaussian random double is 0.023075071226920853
A gaussian random double is 1.0032986463793938
```

Note: A Gaussian or normal distribution is a bell- curve of values from negative infinity topositive infinity, with the majority of the values around zero (0.0).

Calculating Trigonometric Functions

- Use java.lang.Math class.
- For compute sine, cosine, and other trigonometric functions.
- Note that the arguments for trigonometric functions are in radians, not in degrees.
- Degree = radian * (180/PI)

Example:

```
import java.util.Random;

public class Number {

    public static void main(String[] args) {

        System.out.println("The cosine of 1.1418 is " + Math.cos(1.1418));

        System.out.println("The sine of PI/4 is " + Math.sin(Math.PI/4));

        System.out.println("Java’s PI is " + Math.PI);

        System.out.println("Java’s e is " + Math.E);

    }
}
```

Output:

```
The cosine of 1.1418 is 0.4159582880456274б
The sine of 1.1418 is 0.7071067811865475
Java’s PI is 3.141592653589793
Java’s e is 2.718281828459045
```

Taking Logarithms

- Find the logarithm of a number. It can be 2 types:
    - *logarithms to base e*
    - *logarithms to other bases*

logarithms to base e

- The java.lang.Math.log(double a) returns the natural logarithm (base e) of a double value.
- Use java.lang.Math.log() method and the syntax is
    - *public static double log(double a)*
        - *here a is a value*
- *Special cases:*
    - If the argument is NaN or less than zero, then the result is NaN.
    - If the argument is positive infinity, then the result is positive infinity.
    - If the argument is positive zero or negative zero, then the result is negative infinity.

Example:

```
public class Number {

        public static void main(String[] args) {

                // get two double numbers
                double x = 60984.1;
                double y = -497.99;
                // get the natural logarithm for x
                System.out.println("Math.log(" + x + ")=" + Math.log(x));

                // get the natural logarithm for y
                System.out.println("Math.log(" + y + ")=" + Math.log(y));
        }
}
```

Output:

```
Math.log(60984.1)=11.018368453441132
Math.log(-497.99)=NaN
```

logarithms to other bases

- The logarithms of other base can be determined by the following equation

$$\log_n\ (x) = \frac{\log_e\ (x)}{\log_e\ (n)}$$

- where x is the number whose logarithm you want, n is any desired base, and e is the natural logarithm base.

Example: Write the program to find log$_{10}$(10000)

```
public classNumber {

    public static double log_base(double base, double value) {

        return Math.log(value) / Math.log(base);
    }

    Public static void main(String[] args) {

        double d = Number.log_base(10, 10000);

        System.out.println("log10(10000) = " + d);

    }
}
```

Output:

```
log10(10000) = 4.0
```

Multiplying Matrices

Example: Write a program to multiply two matrices.

```
public class Number {

    public static int[][] multiply(int[][] m1, int[][] m2) {

        int m1rows = m1.length;

        int m1cols = m1[0].length;

        int m2rows = m2.length;

        int m2cols = m2[0].length;

        if (m1cols != m2rows)
            thrownewIllegalArgumentException("matrices
                don't match: " + m1cols + " != " + m2rows);

        int[][] result = newint[m1rows][m2cols];

        // multiply
        for (int i=0; i<m1rows; i++) {
            for (int j=0; j<m2cols; j++) {
                for (int k=0; k<m1cols; k++) {

                    result[i][j] += m1[i][k] * m2[k][j];
                }
            }
        }
        returnresult;
    }

    /** Matrix print.  */
    public static void mprint(int[][] a) {

        introws = a.length;

        intcols = a[0].length;

        System.out.println("array["+rows+"]["+cols+"] = {");
        for (int i=0; i<rows; i++) {

            System.out.print("{");

            for (int j=0; j<cols; j++)

                System.out.print(" " + a[i][j] + ",");

            System.out.println("},");
        }
        System.out.println("};");
    }

    public static void main(String[] args) {
        int x[][] = {
                { 3, 2, 3 },
                { 5, 9, 8 },
              };

        int y[][] = {
                { 4, 7 },
                { 9, 3 },
                { 8, 1 },
             };

        int z[][] = Number.multiply(x, y);

        Number.mprint(x);

        Number.mprint(y);

        Number.mprint(z);

    }
}
```

Output:

```
{ 3, 2, 3,},
{ 5, 9, 8,},
};
array[3][2] = {
{ 4, 7,},
{ 9, 3,},
{ 8, 1,},
};
array[2][2] = {
{ 54, 30,},
{ 165, 70,},
};
```

Handling Very Large Numbers

In java two classes to handle the large number

- BigInteger
    - It creates a large integer number and handles integer numbers larger than Long.MAX_VALUE.
- BigDecimal
    - It create a large decimal number (Real number) and handle floating-point values larger than Double.MAX_VALUE.
- Use the BigInteger or BigDecimal values in java.math package.

BigInteger

- It creates a large integer number and handles integer numbers larger than Long.MAX_VALUE.
- Need to import java.math.BigInteger
- Create BigInteger instance as follows:
    - BigInteger bi=new BigInteger(String val);
- It consists of many methods, but some of the methods are

Methods	Descriptions
abs()	It returns a BigInteger whose value is the absolute value of this BigInteger.
add()	This method returns a BigInteger by simply computing 'this + val' value.

substract()	This method returns a BigInteger by simply computing 'this - val' value.
multiply()	This method returns a BigInteger by computing 'this *val ' value.
divide()	This method returns a BigInteger by computing 'this /~val ' value.
pow()	This method returns a BigInteger whose value is 'this^{exponent}'.
negate()	This method returns a BigInteger whose value is '-this'.
compareTo()	Compares this BigInteger with the specified BigInteger.
equals()	Compares this BigInteger with the specified Object for equality.
floatValue()	Converts this BigInteger to a float.
intValue()	Converts this BigInteger to a int.

Example:

```
import java.math.BigInteger;

public class VeryLargeNumber {

        public static void main(String[] args) {

                // Create two new BigInteger
                BigInteger BI1 = newBigInteger("1234455666456765 32");
                BigInteger BI2 = newBigInteger("987654321875436782 89");
                BigInteger BI3 = newBigInteger("-532987654321123456789");
                BigInteger BI20 = newBigInteger("123445");

                // Absolute value of BigInteger
                BigInteger BI4 = BI3.abs();
                System.out.println("BigInteger Absolute value result = " + BI4);

                // Addition of two BigIntegers
                BigInteger BI5 = BI1.add(BI2);
                System.out.println("BigInteger Addition result = " + BI5);

                // Multiplication of two BigIntegers
                BigInteger BI6 = BI1.multiply(BI2);
                System.out.println("BigInteger Multiplication result= " + BI6);
                // Subtraction of two BigIntegers
                BigInteger BI7 = BI1.subtract(BI2);
                System.out.println("BigInteger Subtract result = " + BI7);

                // Division of two BigIntegers
                BigInteger BI8 = BI2.divide(bd2);
                System.out.println("BigInteger Division result = " + BI8);

                // BigInteger raised to the power of 2
                BigInteger BI9= BI1.pow(2);
                System.out.println("BigInteger Exponent result = " + BI9);

                // Negate value of BigInteger1
                BigInteger BI10 = BI1.negate();
                System.out.println("BigInteger  Negation result = " + BI10);

                // Compare the value of BigIntegers
                int BI11 = BI1.compareTo(BI2);
                System.out.println("BigInteger Compare result = " + BI 11);
                                                             // 0 or 1 or -1

                // equals value of BigInteger
                Boolean bd12 = bd1.equals(bd2);
                System.out.println("BigInteger Equal result = " + bd12);
                                                             // false or true


                int BI13 = BI20.intValue();
                System.out.println("Integer result = " + BI13);

                float BI14 = BI1.floatValue();
                System.out.println("Float result = " + BI14);
        }

}
```

Output:

```
BigInteger Absolute value result = 5329876543211234567 89
BigInteger Addition result = 98888877754189354821
BigInteger Multiplication result=
1219215474139646925244189237016521 3748
BigInteger Subtract result = -98641986620898001757
BigInteger Division result = 1
BigInteger Exponent result = 15238807924472166308212407975547024
BigInteger  Negation result = -123445566645676532
BigInteger Compare result = -1
BigInteger Equal result = false
Integer result = 123445
Float result = 1.23445563E17
```

BigDecimal

- It create a large decimal number (Real number) and handle floating-point values larger than Double.MAX_VALUE.

- Need to importjava.math.BigDecimal

- Create BigInteger instance as follows:

    - BigDecimal bi=new BigDecimal(String val)

- It consists of many methods, but some of the methods are

Methods	Descriptions
abs()	It returns a BigDecimal whose value is the absolute value of this BigDecimal.
add()	This method returns a BigDecimal by simply computing 'this + val' value.
substract()	This method returns a BigDecimal by simply computing 'this - val' value.
multiply()	This method returns a BigDecimal by computing 'this *val ' value.
divide()	This method returns a BigDecimal by computing 'this /~val ' value.
pow()	This method returns a BigDecimal whose value is 'this^{exponent}'.
negate()	This method returns a BigDecimal whose value is '-this'.
compareTo()	Compares this BigDecimal with the specified BigDecimal.
equals()	Compares this BigDecimal with the specified Object for equality.

Example:

```
import java.math.BigDecimal;

publicclass VeryLargeNumber {

        publicstaticvoidmain(String[] args) {

                // Create two new BigDecimals
                BigDecimal BD1 = newBigDecimal("124567890.0987654321");
                BigDecimal BD2 = newBigDecimal("987654321.123456789");
                BigDecimal BD3 = newBigDecimal("-532987654321.123456");

                // Absolute value of BigDecimal
                BigDecimal BD4 = BD3.abs();
                System.out.println("BigDecimal Absolute value result = " + BD4);

                // Addition of two BigDecimals
                BigDecimal BD5 = BD1.add(BD2);
                System.out.println("BigDecimal Addition result = " + BD5);

                // Multiplication of two BigDecimals
                BigDecimal BD6 = BD1.multiply(BD 2);
                System.out.println("BigDecimal Multiplication result= " + BD6);

                // Subtraction of two BigDecimals
                BigDecimal BD7 = BD1.subtract(BD2);
                System.out.println("BigDecimal Subtract result = " + BD7);

                // Division of two BigDecimals
                BigDecimal BD8 = BD2.divide(BD2);
                System.out.println("BigDecimal Division result = " + BD8);

                // BigDecima1 raised to the power of 2
                BigDecimal BD9= BD1.pow(2);
                System.out.println("BigDecimal Exponent result = " + BD9);

                // Negate value of BigDecimal1
                BigDecimal BD10 = BD1.negate();
                System.out.println("BigDecimal Negation result = " + BD10);

                // Negate value of BigDecimal1
                int BD11 = BD1.compareTo(BD2);
                System.out.println("BigDecimal Compare result = " + BD11);
                                                                // 0 or 1 or -1

                // Negate value of BigDecimal1
                Boolean BD12 = BD1.equals(BD2);
                System.out.println("BigDecimal Equal result = " + BD12);
                                                                // false or true
        }

}
```

Output:

```
BigDecimal Absolute value result = 532987654321.123456789
BigDecimal Addition result = 1112222211.2222222211
BigDecimal Multiplication result=
123030014929277547.5030955772112635269
BigDecimal Subtract result = -863086431.0246913569
BigDecimal Division result = 1
BigDecimal Exponent result = 15517159243658102.97302514857789971041
BigDecimal Negation result = -124567890.0987654321
BigDecimal Compare result = -1
BigDecimal Equal result = false
```

Program: TempConverter

The following program prints a table of Fahrenheit temperatures and the corresponding Celsius temperatures by using *printf*, which control the formatting of the converted temperatures.

Example:

```
public class TempConverter {

        public static void main(String[] args) {
                TempConverter t = newTempConverter();
                t.start();
                t.data();
                t.end();

        }
        protected void start() {
                System.out.println("  Fahr   Centigrade");
        }

        protected void data() {
                for (inti=-40; i<=120; i+=10) {
                        floatc = (i-32)*(5f/9);
                        print(i, c);
                }
        }


        protected void print(floatf, floatc) {
                System.out.printf("%6.2f    %6.2f%n", f, c);
        }

        protected void end() {
                System.out.println("------------------");
        }
}
```

Output:

```
Fahr   Centigrade
-40.00    -40.00
-30.00    -34.44
-20.00    -28.89
-10.00    -23.33
  0.00    -17.78
 10.00    -12.22
 20.00     -6.67
 30.00     -1.11
 40.00      4.44
 50.00     10.00
 60.00     15.56
 70.00     21.11
 80.00     26.67
 90.00     32.22
100.00     37.78
110.00     43.33
120.00     48.89
------------------
```