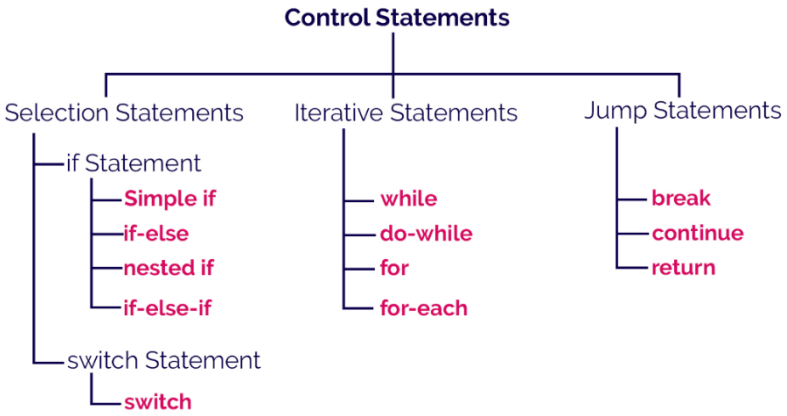
UNIT – 2

Control statements



(NOTE : For theory and flowcharts Read text book pg.no 79 - 96)

➢ Conditional / Selection / Decision making statements

Simple if - java program to check the number is +ve.

```
class SimpleIf
{
        public static void main (String args[])
        {
                int x=10;
                if(x>0)
                        System.out.println("x is a positive number ");
        }
}
```

```
E:\JAVA>javac SimpleIf.java

E:\JAVA>java SimpleIf
x is a positive number
```

If-else - java program to check the number is +ve or –ve.

```
class IfElse
{
        public static void main (String args[])
        {
                int x=-10;
                if(x>0)
                        System.out.println("x is a +ve number ");
                else
                        System.out.println("x is a -ve number ");
        }
}
```

```
E:\JAVA>javac IfElse.java

E:\JAVA>java IfElse
x is a -ve number
```

Nested if - java program to check the greatest among 3 numbers.

```
class NestedIf
{
       public static void main (String args[])
       {
              int a=30,b=50,c=40;
              if(a>b)
              {
                     if(a>c)
                            System.out.println("A is big");
                     else
                            System.out.println("C is big");
              }
              else
              {
                     if(b>c)
                            System.out.println("B is big");
                     else
                            System.out.println("C is big");
              }

       }
}
```

```
E:\JAVA>javac NestedIf.java

E:\JAVA>java NestedIf
B is big
```

If else if - java program to check the number is +ve or –ve or equals to 0

```
class IfElseIf
{
        public static void main (String args[])
        {
                int x=0;
                if(x>0)
                        System.out.println("x is a +ve number ");
                else if(x==0)
                        System.out.println("x is equalto 0 ");
                else
                        System.out.println("x is a -ve number ");
        }
}
```

```
E:\JAVA>javac IfElseIf.java

E:\JAVA>java IfElseIf
x is equalto 0
```

Switch case - java program to check the weekday name.

```
import java.util.*;
class SwitchCase
{
        public static void main (String args[])
        {
                Scanner sc=new Scanner(System.in);
                int a=sc.nextInt();
                switch (a)
                {
                        case 1:
                        System.out.println("Monday");break;
                        case 2:
                        System.out.println("Tuesday"); break;
                        case 3:
                        System.out.println("Wednesday"); break;
                        case 4:
                        System.out.println("Thursday"); break;
                        case 5:
                        System.out.println("Friday"); break;
                        case 6:
                        System.out.println("Saturday"); break;
                        case 7:
                        System.out.println("Sunday"); break;
                        default:System.out.println("Invalid day");
                }
        }
}
```

```
E:\JAVA>javac SwitchCase.java

E:\JAVA>java SwitchCase
3
Wednesday

E:\JAVA>javac SwitchCase.java

E:\JAVA>java SwitchCase
9
Invalid day
```

➢ Looping statements

While loop

```
class WhileLoop
{
        public static void main (String args[])
        {
                int i=1;
                while(i<=5)
                {
                        System.out.println(i);
                        i++;
                }
        }
}
```

```
E:\JAVA>javac WhileLoop.java

E:\JAVA>java WhileLoop
1
2
3
4
5
```

do while loop

```
class DoWhileLoop
{
        public static void main (String args[])
        {
                int i=1;
                do
                {
                        System.out.println(i);
                        i++;
                }while(i<=5);
        }
}
```

```
E:\JAVA>javac DoWhileLoop.java

E:\JAVA>java DoWhileLoop
1
2
3
4
5
```

for loop

```
class ForLoop
{
        public static void main (String args[])
        {
                int i;
                for(i=1;i<=5;i++)
                {
                        System.out.println(i);
                }
        }
}
```

```
E:\JAVA>javac ForLoop.java

E:\JAVA>java ForLoop
1
2
3
4
5
```

for each loop

```
class ForEach
{
        public static void main (String args[])
        {
                int a[]={10,20,30,40,50};
                for(int i : a)
                {
                        System.out.println(i);
                }
        }
}
```

```
E:\JAVA>javac ForEach.java

E:\JAVA>java ForEach
10
20
30
40
50
```

Nested loops:

A Nested loop means a loop statement inside another loop statement. That is why nested loops are also called “loop inside loop“.

Syntax:

```
for ( initialization; condition; increment )
{
        for ( initialization; condition; increment )
                // statements of inside loop
        // statements of outer loop
}
```

Program to printing pattern – right angle triangle

```
class NestedLoop
{
    public static void main(String args[])
    {
        for(int i=0;i<5;i++)
        {
                for(int j=0;j<=i;j++)
                {
                        System.out.print("*"+" ");
                }
                System.out.println();
        }
    }
}
```

```
E:\JAVA\ANU>javac NestedLoop.java

E:\JAVA\ANU>java NestedLoop
*
* *
* * *
* * * *
* * * * *
```

Program to printing pattern – right angled triangle

```
class NestedLoop1
{
    public static void main(String args[])
    {
        for(int i=0;i<=5;i++)
        {
                for(int j=0;j<5-i;j++)
                {
                        System.out.print("*"+" ");
                }
                System.out.println();
        }
    }
}
```

```
E:\JAVA\ANU>javac NestedLoop1.java

E:\JAVA\ANU>java NestedLoop1
* * * * *
* * * *
* * *
* *
*
```

➢ Jump Statements

break

```
class Break
{
	public static void main (String args[])
	{
		for(int i=1;i<=5;i++)
		{
			System.out.println(i);
			if(i==3)
				break;
		}
	}
}
```

```
E:\JAVA>javac Break.java

E:\JAVA>java Break
1
2
3
```

continue

```
class Continue
{
	public static void main (String args[])
	{
		for(int i=1;i<=5;i++)
		{
			if(i==3)
				continue;
			System.out.println(i);
		}
	}
}
```

```
E:\JAVA>javac Continue.java

E:\JAVA>java Continue
1
2
4
5
```

return - java program to check the maximum of 2 values using method with return value

```
class Return
{
	static int max(int a,int b)
	{
		if(a>b)
			return a;
		else
			return b;
	}
	public static void main (String args[])
	{
		int a=20,b=30;
		int max_val=max(a,b);
		System.out.println(max_val);
	}
}
```

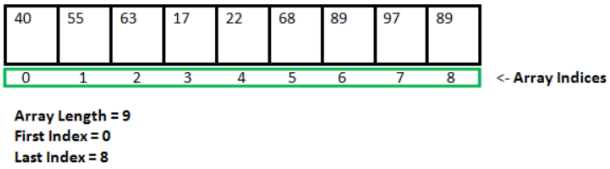
```
E:\JAVA>javac Return.java

E:\JAVA>java Return
30
```

Arrays

Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on. Following are some important points about Java arrays.

- ✓ In Java, all arrays are dynamically allocated.
- ✓ Arrays may be stored in contiguous memory [consecutive memory locations].
- ✓ Since arrays are objects in Java, we can find their length using the object property length. This is different from C/C++, where we find length using size of.
- ✓ A Java array variable can also be declared like other variables with [] after the data type.
- ✓ The values in the array are ordered, and each has an index beginning with 0.
- ✓ Java array can also be used as a static field, a local variable, or a method parameter.
- ✓ This storage of arrays helps us randomly access the elements of an array [Support Random Access]



Advantages

- ✓ Code Optimization: It makes the code optimized, we can retrieve or sort the data efficiently.
- ✓ Random access: We can get any data located at an index position.

Disadvantages

- ✓ Size Limit: We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.
- ✓ Java cannot store heterogeneous data. It can only store a single type of primitives

Types of Array in java

Single Dimensional Array

- An array ehich has only one dimension. Also known as a linear array, the elements are stored in a single row.

Creating, Initializing, and Accessing an Arrays

Syntax to Declare an Array in Java

dataType[] arr; (or)

dataType []arr; (or)

dataType arr[];

Ex : int arr[];

Instantiation of an Array in Java

datatype array_name[]=new datatype[size];

Here, type specifies the type of data being allocated, size determines the number of elements in the array, and var-name is the name of the array variable that is linked to the array. To use new to allocate an array, you must specify the type and number of elements to allocate.

Ex : int arr[]=new int[10];

Assigning values to array or Array literal in java

int arr[]={10,20,30,40,50}

int arr[] = new int[]{ 1,2,3,4,5,6,7,8,9,10 };

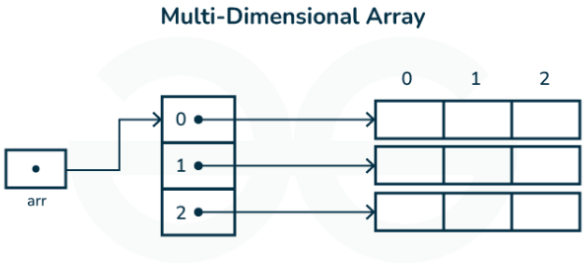
Note: The elements in the array allocated by new will automatically be initialized to zero.

```
class SingleDimension
{
        public static void main(String args[])
        {
                int a[]={10,20,30,40,50};
                int b[]=new int[]{60,70,80};
                for (int i = 0; i < a.length; i++)
                        System.out.print(a[i]+" ");
                System.out.println();
                for (int j = 0; j < b.length; j++)
                        System.out.print(b[j]+" ");
        }
}
```

```
E:\JAVA>javac SingleDimension.java

E:\JAVA>java SingleDimension
10 20 30 40 50
60 70 80
E:\JAVA>
```

Multidimensional Array

- In this case, data is stored in row and column based index (also known as matrix form).



Syntax to Declare Multidimensional Array in Java

dataType[][] arrayRefVar; (or)

dataType [][]arrayRefVar; (or)

dataType arrayRefVar[][];

Ex : int arr[][];

Instantiation of an Array in Java

datatype array_name[]=new datatype[row][col];

Ex : int arr[][]=new int[3][3];

Assigning values to array or Array literal in java

int arr[][] = { { 2, 7, 9 }, { 3, 6, 1 }, { 7, 4, 2 } };

int arr[][] = new int[][]{{1,2,3},{4,5,6}};

```
class MultiDimension
{
	public static void main(String args[])
	{
		int a[][] = {{2,7,9}, {3,6,1}, {7,4,2}};
		int b[][] = new int[][]{{1,2},{4,5}};

		for (int i = 0; i < a.length; i++)
		{
			for (int j = 0; j < a.length; j++)
				System.out.print(a[i][j]+" ");
			System.out.println();
		}
		System.out.println();
		for (int i = 0; i < b.length; i++)
		{
			for (int j = 0; j < b.length; j++)
				System.out.print(b[i][j]+" ");
			System.out.println();
		}

	}
}
```

```
E:\JAVA>javac MultiDimension.java

E:\JAVA>java MultiDimension
2 7 9
3 6 1
7 4 2

1 2
4 5
```

Jagged Array

- A jagged array is an array of arrays such that member arrays can be of different sizes, i.e., we can create a 2-D array but with a variable number of columns in each row. These types of arrays are also known as Jagged arrays.

Syntax: data_type array_name[][] = new data_type[n][]; //n: no. of rows

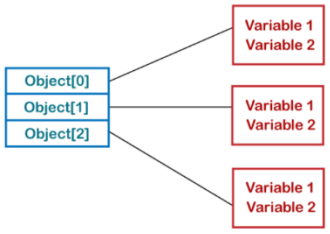
Ex : int a[][]=new int[][]={{2,7,9},{2,5,1,3,2},{4,9,2}}

```
class Jagged
{
        public static void main(String args[])
        {
                int a[][]=new int[][]{{2,7,9}, {3,6,1}, {7,4,2}, {2,5,1,3,2}, {4,9,2}};
                for (int i = 0; i < 5; i++)
                {
                        for (int j = 0; j < a[i].length; j++)
                                System.out.print(a[i][j]+" ");
                        System.out.println();
                }
                System.out.println();
        }
}
```

```
E:\JAVA>javac Jagged.java

E:\JAVA>java Jagged
2 7 9
3 6 1
7 4 2
2 5 1 3 2
4 9 2
```

Array of Objects:

Java is an object-oriented programming language. Most of the work done with the help of objects. We know that an array is a collection of the same data type that dynamically creates objects and can have elements of primitive types. Java allows us to store objects in an array. In Java, the class is also a user-defined data type. An array that conations class type elements are known as an array of objects.



Creating an Array of Objects

Before creating an array of objects, we must create an instance of the class by using the new keyword. We can use any of the following statements to create an array of objects.

Syntax :

ClassName[] objarr;
ClassName objarr[];

Ex:

Student[] arrd;
Student arrd[];

declare and instantiate an array of objects:

ClassName objarr[]=new ClassName[size];

- For example, if you have a class Student, and we want to declare and instantiate an array of Demo objects with five objects/object references then it will be written as:

Ex: Student arrd[]=new Student[5];

- And once an array of objects is instantiated like this, then the individual elements of the array of objects needs to be created using the new keyword.
- Once the array of objects is instantiated, we need to initialize it with values. We cannot initialize the array in the way we initialize with primitive types as it is different from an array of primitive types. In an array of objects, we have to initialize each element of array i.e. each object/object reference needs to be initialized.
  Example program:

```
class Student
{
        public int id;
        public String name;
        Student(int pin, String names)
        {
                id = pin;
                name = names;
        }
        public static void main(String args[])
        {
                Student[] a = new Student[3];
                a[0] = new Student(1, "aaa");
                a[1] = new Student(2, "bbb");
                a[2] = new Student(3, "ccc");
                for(int i=0;i<a.length;i++)
                {
                        System.out.printf("student at %d is %d %s\n",i,a[i].id,a[i].name);
                }
        }
}
```

```
E:\JAVA>javac Student.java

E:\JAVA>java Student
student at 0 is 1 aaa
student at 1 is 2 bbb
student at 2 is 3 ccc
```

Class and Objects

Classes

In Java, classes and objects are basic concepts of Object Oriented Programming (OOPs) that are used to represent real-world concepts and entities. The class represents a group of objects having similar properties and behavior. For example, Student is a class while a particular student named Ravi is an object.

Properties of Java Classes

- Class is not a real-world entity. It is just a template or blueprint or prototype from which objects are created.
- Class does not occupy memory.
- Class is a group of variables of different data types and a group of methods.
- A Class in Java can contain:
  - ✓ Data member
  - ✓ Method
  - ✓ Constructor
  - ✓ Nested Class
  - ✓ Interface

Components of Java Classes

In general, class declarations can include these components, in order:

- ✓ Modifiers: A class can be public or has default access (Refer this for details).
- ✓ Class keyword: class keyword is used to create a class.
- ✓ Class name: The name should begin with an initial letter (capitalized by convention).
- ✓ Superclass(if any): The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
- ✓ Interfaces(if any): A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
- ✓ Body: The class body is surrounded by braces, { }.

Syntax for class declaration

```
access_modifier class <class_name>
{
    data member;
    method;
    constructor;
    nested class;
    interface;
}
```

program

```
class Stu
{
    int id;
    String name;
    public static void main(String args[])
    {
        Stu s = new Stu();
        System.out.println(s.id);
        System.out.println(s.name);
    }
}
```

```
E:\JAVA>javac Stu.java

E:\JAVA>java Stu
0
null
```

Constructors are used for initializing new objects. Fields are variables that provide the state of the class and its objects, and methods are used to implement the behavior of the class and its objects.

There are various types of classes that are used in real-time applications such as nested classes, anonymous classes, and lambda expressions.

Objects

An object in Java is a basic unit of Object-Oriented Programming and represents real-life entities. Objects are the instances of a class that are created to use the attributes and methods of a class. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :

- ✓ State: It is represented by attributes of an object. It also reflects the properties of an object.
- ✓ Behavior: It is represented by the methods of an object. It also reflects the response of an object with other objects.
- ✓ Identity: It gives a unique name to an object and enables one object to interact with other objects.

Syntax: ClassName obj_name = new ClassName()

Ex: Student s=new Student()

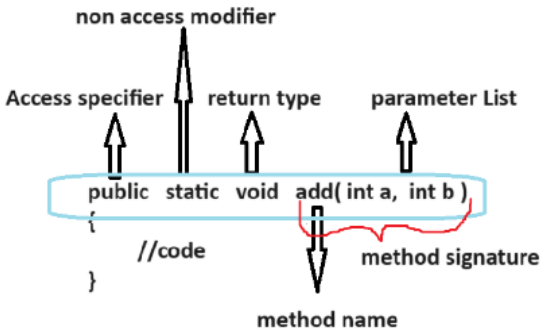
```
class Stu
{
    void msg()
    {
        System.out.println("this is method");
    }
    public static void main(String args[])
    {
        Stu s = new Stu();
        s.msg();
    }
}
```

```
E:\JAVA>javac Stu.java

E:\JAVA>java Stu
this is method
```

Methods in java

- A method is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation. It is used to achieve the reusability of code. We write a method once and use it many times. We do not require to write code again and again. It also provides the easy modification and readability of code, just by adding or removing a chunk of code.

Method declaration



- Method Signature: Every method has a method signature. It is a part of the method declaration. It includes the method name and parameter list.
- Access Specifier: Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides four types of access specifier:
    - Public: The method is accessible by all classes when we use public specifier in our application.
    - Private: When we use a private access specifier, the method is accessible only in the classes in which it is defined.
    - Protected: When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.
    - Default: When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.
- Return Type: Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use void keyword.
- Method Name: It is a unique name that is used to define the name of a method. It must be corresponding to the functionality of the method. Suppose, if we are creating a method for subtraction of two numbers, the method name must be subtraction(). A method is invoked by its name.

- Parameter List: It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name. If the method has no parameter, left the parentheses blank.
- Method Body: It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly braces.

Naming a Method

While defining a method, remember that the method name must be a verb and start with a lowercase letter. If the method name has more than two words, the first name must be a verb followed by adjective or noun. In the multi-word method name, the first letter of each word must be in uppercase except the first word.

Ex : sum(), area(), areaOfCircle(), stringSlicing()….

Types of Method

- Predefined Method
- User-defined Method

Predefined Method

In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods. It is also known as the standard library method or built-in method. We can directly use these methods just by calling them in the program at any point. Some pre-defined methods are length(), equals(), nextInt(), sqrt(), etc.

```
class PreMethods
{
        public static void main(String[] args)
        {
                System.out.print("maximum number : " + Math.max(10,7));
        }
}
```

```
E:\JAVA\ANU>javac PreMethods.java

E:\JAVA\ANU>java PreMethods
maximum number : 10
```

User-defined Method

The method written by the user or programmer is known as a user-defined method. These methods are modified according to the requirement.

There are two ways to create a method in Java:

➢ **Instance method**

These methods are created inside the class outside the main method. The method of the class is known as an instance method. It is a non-static method defined in the class. Before calling or invoking the instance method, it is necessary to create an object of its class. Access the instance data using the object name.

Syntax:

```
// Instance Method
void method_name(){
  body // instance area
}
```

➢ **Static Method**

These methods are created inside the class outside the main method by using static keyword. A method with static is known as static method. The main advantage of a static method is that we can

call it without creating an object. It can access static data members and also change the value of it. It is invoked by using the class name or by direct method call. The best example of a static method is the main() method.

Program on methods

```
class UserMethods
{
        static void sum(int a,int b)
        {
                System.out.println("Static method Addition "+(a+b));
        }
        void add(int a,int b)
        {
                System.out.println("Instance method Addition "+(a+b));
        }
        public static void main(String[] args)
        {
                System.out.println("**** USER DEFINED METHODS ****");
                UserMethods.sum(10,5);
                UserMethods um=new UserMethods();
                um.add(20,30);
        }
}
```

```
E:\JAVA\ANU>javac UserMethods.java

E:\JAVA\ANU>java UserMethods
**** USER DEFINED METHODS ****
Static method Addition 15
Instance method Addition 50
```

Java program on static fields and Methods

- **Static** variables and methods belong to a class and are called with the Class Name rather than using object variables, like ClassName.methodName();
- There is only one copy of a static variable or method for the whole class. For example, the main method is static because there should only be 1 main method.
- Static methods only have access to other static variables and static methods. Static methods cannot access or change the values of instance variables or the this reference (since there is no calling object for them), and static methods cannot call non-static methods. However, non-static methods have access to all variables (instance or static) and methods (static or non-static) in the class.

```
class StaticFields
{
        static int a=100;
        static void sum(int a,int b)
        {
                System.out.println("Static method Addition "+(a+b));
        }
        public static void main(String[] args)
        {
                StaticFields.sum(10,5);
                System.out.println("Static variable "+StaticFields.a);
        }
}
```

```
E:\JAVA\ANU>javac StaticFields.java

E:\JAVA\ANU>java StaticFields
Static method Addition 15
Static variable 100
```

Constructors in Java

- In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.
- It is a special type of method which is used to initialize the object.
- Every time an object is created using the new() keyword, at least one constructor is called.
- It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating Java constructor

- ✓ Constructor name must be the same as its class name
- ✓ A Constructor must have no return type
- ✓ A Java constructor cannot be abstract, static and final.

Types of Java constructors

There are Three types of constructors in Java:

- ✓ Default constructor
- ✓ No-arg constructor
- ✓ Parameterized constructor

Default constructor

If you do not implement any constructor in your class, Java compiler inserts a default constructor into your code. This constructor is known as default constructor. We would not find it in your source code (the java file) as it would be inserted into the code during compilation and exists in .class file.

Lets understand this using an example program.

```
class DefaultCon
{
        public static void main(String[] args)
        {
                DefaultCon dc = new DefaultCon();
        }
}
```

After compilation

```
class DefaultCon
{
        DefaultCon()
        {
        }
        public static void main(String[] args)
        {
                DefaultCon dc = new DefaultCon();
        }
}
```

no-arg constructor

Constructor with no arguments is known as no-arg constructor. The signature is same as default constructor; however, body can have any code unlike default constructor where the body of the constructor is empty.

```
class NoArgCon
{
        NoArgCon()
        {
                System.out.println("No-Arg constructor");
        }
        public static void main(String[] args)
        {
                NoArgCon nac = new NoArgCon();
        }
}
```

```
E:\JAVA\ANU>javac NoArgCon.java

E:\JAVA\ANU>java NoArgCon
No-Arg constructor
```

Parameterized constructor

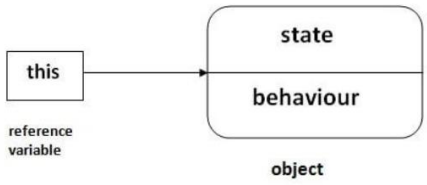
A constructor with arguments (or you can say parameters) is known as a Parameterized constructor. During initialization of an object, which constructor should get invoked depends upon the parameters one passes.

```
class ParaCon
{
        ParaCon(float a,float b)
        {
                System.out.println("Multiplication "+(a+b));
        }
        public static void main(String[] args)
        {
                System.out.println("Parameterized constructor");
                ParaCon pc = new ParaCon(4.2f,4.2f);
        }
}
```

```
E:\JAVA\ANU>javac ParaCon.java

E:\JAVA\ANU>java ParaCon
Parameterized constructor
Multiplication 8.4
```

this keyword

In Java, this is a reference variable that refers to the current class object.



Usage of this keyword

Here is given the 6 usage of java this keyword.

- ✓ this can be used to refer current class instance variable.
- ✓ this can be used to invoke current class method (implicitly)
- ✓ this() can be used to invoke current class constructor.
- ✓ this can be passed as an argument in the method call.
- ✓ this can be passed as argument in the constructor call.
- ✓ this can be used to return the current class instance from the method.

this: to refer current class instance variable

The this keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

```
class VarThis
{
        int a;
        VarThis(int a)
        {
                this.a=a+1;
                System.out.println("instance a ="+this.a);
                System.out.println("local a ="+a);
        }
        public static void main(String[] args)
        {
                VarThis t=new VarThis(10);


        }
}
```

```
E:\JAVA\ANU>javac VarThis.java

E:\JAVA\ANU>java VarThis
instance a =11
local a =10
```

this: to invoke current class method

You may invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method. Let's see the example.

```
class MethodThis
{
        void msg()
        {
                System.out.println("msg method");
        }
        void call()
        {
                this.msg();
                System.out.println("call method");
        }
        public static void main(String[] args)
        {
                MethodThis t=new MethodThis();
                t.call();
        }
}
```

```
E:\JAVA\ANU>javac MethodThis.java

E:\JAVA\ANU>java MethodThis
msg method
call method
```

this() : to invoke current class constructor

The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

```
class ConThis
{
        ConThis()
        {
                this(10,20);
                System.out.println("no-arg Constructor");
        }
        ConThis(int a,int b)
        {
                System.out.println("parameterized Constructor");
        }
        public static void main(String[] args)
        {
                ConThis t=new ConThis();
        }
}
```

```
E:\JAVA\ANU>javac ConThis.java

E:\JAVA\ANU>java ConThis
parameterized Constructor
no-arg Constructor
```

Adding a Constructor

- Also called as constructor chaining.
- Constructor chaining is the process of calling one constructor from another constructor. To perform this we use this keyword inside the constructor, which can be used to invoke the other constructor of the same class.
- One of the main use of constructor chaining is to avoid duplicate codes while having multiple constructor (by means of constructor overloading) and make code more readable.

- Why do we need constructor chaining?

  This process is used when we want to perform multiple tasks in a single constructor rather than creating a code for each task in a single constructor we create a separate constructor for each task and make their chain which makes the program more readable.

  - **Rules of Constructor Chaining**
    - ✓ An expression that uses this keyword must be the first line of the constructor.
    - ✓ Order does not matter in constructor chaining.
    - ✓ There must exist at least one constructor that does not use this

```
class ChainCon
{
        ChainCon()
        {
                this(10);
                System.out.println("no parameters");
        }
        ChainCon(int x)
        {
                this(20, 30);
                System.out.println("one parameter "+x);
        }
        ChainCon(int x, int y)
        {
                System.out.println("two parameters "+x+" "+y);
        }
        public static void main(String args[])
        {
                ChainCon c=new ChainCon();
        }
}
```

```
E:\JAVA\ANU>javac ChainCon.java

E:\JAVA\ANU>java ChainCon
two parameters 20 30
one parameter 10
no parameters
```

Constructor overloading in Java

In Java, we can overload constructors like methods. The constructor overloading can be defined as the concept of having more than one constructor with different parameters so that every constructor can perform a different task.

```
class ConOverload
{
        ConOverload()
        {
                int a=1,b=2;
                System.out.println("no parameter Addition "+(a+b));
        }
        ConOverload(int x)
        {
                System.out.println("one parameter Integer Addition "+(x+x));
        }
        ConOverload(float x)
        {
                System.out.println("one parameter Float Addition "+(x+x));
        }

        ConOverload(int x, int y)
        {
                System.out.println("two parameters Addition "+(x+y));
        }
        public static void main(String args[])
        {
                ConOverload c1=new ConOverload();
                ConOverload c2=new ConOverload(5);
                ConOverload c3=new ConOverload(3.4f);
                ConOverload c4=new ConOverload(10,20);
        }
}
```

output:

```
E:\JAVA\ANU>javac ConOverload.java

E:\JAVA\ANU>java ConOverload
no parameter Addition 3
one parameter Integer Addition 10
one parameter Float Addition 6.8
two parameters Addition 30
```