CONTRUCTORS

Definition:

Constructor is a **special type of method** that is used to initialize the object. Constructor is **invoked at the time of object creation**. Once defined, the constructor is automatically called immediately after the object is created, before the **new** operator completes.

- **Rules for creating constructor:**

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type
3. Constructors can be declared public or private (for a Singleton)
4. Constructors can have no-arguments, some arguments and var-args;
5. A constructor is always called with the **new** operator
6. The default constructor is a no-arguments one;
7. If you don't write ANY constructor, the compiler will generate the default one;
8. Constructors CAN'T be **static**, **final** or **abstract**;
9. When overloading constructors (defining methods with the same name but with different arguments lists) you must define them with different arguments lists (as number or as type)

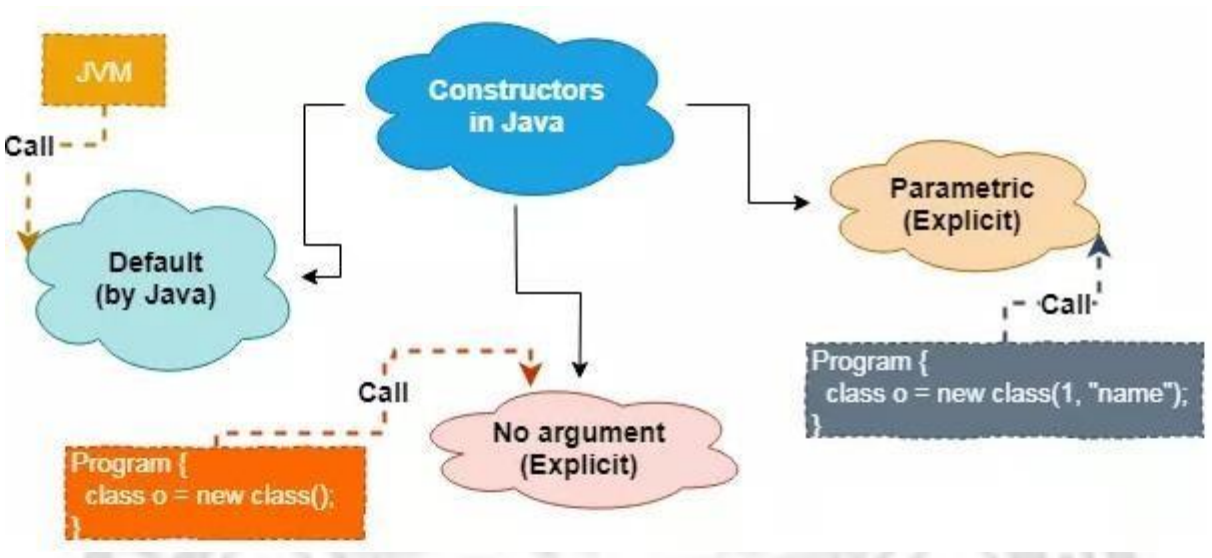
- **What happens when a constructor is called?**

1. All data fields are initialized to their default value (0, false or null).
2. All field initializers and initialization blocks are executed, in the order in which theyoccur in the class declaration.
3. If the first line of the constructor calls a second constructor, then the body of the second constructor is executed.
4. The body of the constructor is executed.

- **Types of constructors**

There are two types of constructors:

1. Default constructor
2. no-arg constructor
3. Parameterized constructor



```
Program {
  class o = new class();
}
```

```
Program {
  class o = new class(1, "name");
}
```

1. Default Constructor

- **Default constructor refers to a constructor that is automatically created by compiler in the absence of explicit constructors.**

Rule: If there is no constructor in a class, compiler automatically creates a default constructor.
Purpose of Default Constructor: It is used to provide the default values to the object members like 0, null etc. depending on the data type.

Example:

```
class student
{
        int  id;
        String name;
        void display()
        {
                System.out.println(id+&quot; &quot;+name);
        }
        public static void main(String args[])
        {
                student s1=new student();
                student s2=new student();
                s1.display();
                s2.display();
        }
}
```

Output:

```
0 null
1    null
```

<u>2) No-Argument Constructor</u>

- **Constructor without parameters is called no-argument constructor**.

<u>Purpose of No-Arg Constructor:</u> It is used to provide values to be common for all objects of the class.

<u>Syntax of default constructor:</u>

```
Classname()
{
        // Constructor body
}
```

<u>Example:</u>

```
class Box
{
   double  width;
   double height;
   double depth;

   // This is the constructor for Box
   Box()
   {
           System.out.println("Constructing Box…");
           width=10;
           height=10;
           depth=10;
   }
// Compute and return volume
 double volume()
  {
             return width*height*depth;
  }
}
class BoxDemo
{
           public static void main(String arg[])
           {
                       // declare, allocate and initialize Box objects
```

```
                Box mybox1=new Box();
                Box mybox2=new Box();
                double vol;

                // Get volume of first box
                vol=mybox1.volume();
                System.out.println("Volume is " +vol);

                // Get volume of second box
                vol=mybox2.volume();
                System.out.println("Volume is "+vol);
        }
}
```

Output:

```
        Constructing Box
        Constructing Box
        Volume is 1000.0
        Volume is 1000.0
```

As you can see, both **mybox1** and **mybox2** were initialized by the **Box( )** constructor when they were created. Since the constructor gives all boxes the same dimensions, 10 by 10 by 10, both **mybox1** and **mybox2** will have the same volume.

3 Parameterized Constructor

A constructor that takes parameters is known as parameterized constructor.

Purpose of parameterized constructor

Parameterized constructor is used to provide different values to the distinct objects.

Example:

```
class Box
{
  double  width;
  double height;
  double depth;

  // This is the constructor for Box
```

```
   Box(double w, double h, double d)
   {
          width=w;height=h;depth=d;
   }

 // Compute and return volume
   double volume()
   {
 return width*height*depth;
   }
 }
 class BoxDemo
 {
 public static void main(String arg[])
 {
          // declare, allocate and initialize Box objects
          Box mybox1=new  Box(10,20,15);
          Box mybox2=new Box(3,6,9);
          double vol;
          // Get volume of first box

          vol=mybox1.volume();
          System.out.println("Volume is " +vol);

          // Get volume of second box
          vol=mybox2.volume();
          System.out.println("Volume is " +vol);
 }
 }
```

Output:

```
Volume is 3000.0
Volume is 162.0
```

As you can see, each object is initialized as specified in the parameters to its constructor. Forexample, in the following line,

Box mybox1 = new Box(10, 20, 15);

the values 10, 20, and 15 are passed to the **Box( )** constructor when **new** creates the object. Thus,

mybox1's copy of **width**, **height**, and **depth** will contain the values 10, 20, and 15, respectively.

Difference between constructor and method:

There are many differences between constructors and methods. They are given below

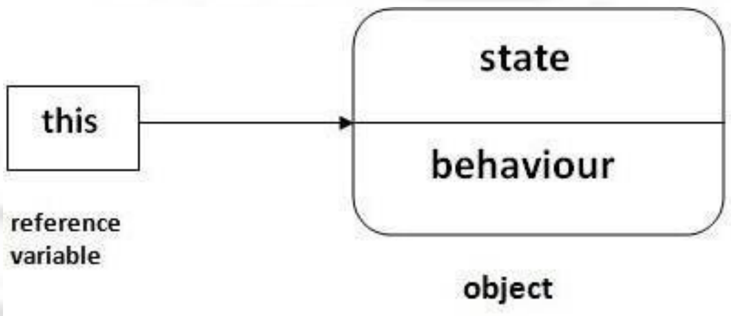
Constructor	Method
Constructor is used to initialize the state of anobject.	Method is used to expose behaviour of anobject.
Constructor must not have return type.	Method must have return type.
Constructor is invoked implicitly.	Method is invoked explicitly.
The java compiler provides a default constructor if you don't have any constructor.	Method is not provided by compiler in anycase.
Constructor name must be same as the classname.	Method name may or may not be same asclass name.

"this" KEYWORD:

Definition:

In java, **this** is a reference variable that refers to the current object.

- **Usage of this keyword**
  1. this keyword can be used to refer current class instance variable.
  2. this() can be used to invoke current class constructor.
  3. this keyword can be used to invoke current class method (implicitly)
  4. this can be passed as an argument in the method call.
  5. this can be passed as argument in the constructor call.
  6. this keyword can also be used to return the current class instance.



Instance Variable Hiding:

It is illegal in Java to declare two local variables with the same name inside the same or enclosing scopes.
We can also have local variables, which overlap with the names of the class' instance

variables.

However, **when a local variable has the same name as an instance variable, the local variable *hides* the instance variable.**

We can use **"this"** keyword to to resolve any namespace collisions that might occur between instance variables and local variables.

Example:

```
class Student
{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee)
{
        this.rollno=rollno;
        this.name=name;
        this.fee=fee;
}
void display()
{
        System.out.println(rollno+" "+name+" "+fee);
}
}

class TestThis2
{
public static void main(String args[])
{
        Student s1=new Student(111,"ankit",5000f);
        Student s2=new Student(112,"sumit",6000f);
        s1.display();
        s2.display();
}
}
```

Output:

```
111 ankit 5000
112 sumit 6000
```

CONSTRUCTOR OVERLOADING:

Definition:

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Example of Constructor Overloading:

```
class Box
{
        double  width;
        double height;
        double depth;

        // constructor used when all the dimensions are specified
        Box(double w, double h, double d)
        {
                width=w;
                height=h;
                depth=d;
        }

        // constructor used when no dimensions are specified
        Box()
        {
                width=-1;
                height=-1;
                depth=-1;
        }

       // constructor used when cube is created
       Box(double len)
       {
                width = height = depth = len;
        }

        // Compute and return volume
     double volume()
        {
                return width*height*depth;
        }
  }
```

```
class ConsOverloadDemo
{
public static void main(String arg[])
{
    // declare, allocate and initialize Box objects
    Box mybox1=new Box(10,20,15);
    Box mybox2=new Box();
    Box mybox3=new Box(7);
    double vol;

    // Get volume of first box
    vol=mybox1.volume();
    System.out.println("Volume of Box1 is  "+vol);

    // Get volume of second box
    vol=mybox2.volume();
    System.out.println("Volume of Box2 is  "+vol);

    // Get volume of cube
    vol=mybox2.volume();
    System.out.println("Volume of Cube is  "+vol);
}
}
```

Output:

```
Volume of Box1 is 3000.0
Volume of Box2 is -1.0
Volume of the cube is 343.0
```

As we can see, the proper overloaded constructor is called based upon the parameters specifiedwhen **new** is executed.

CONSTRUCTOR CHAINING:

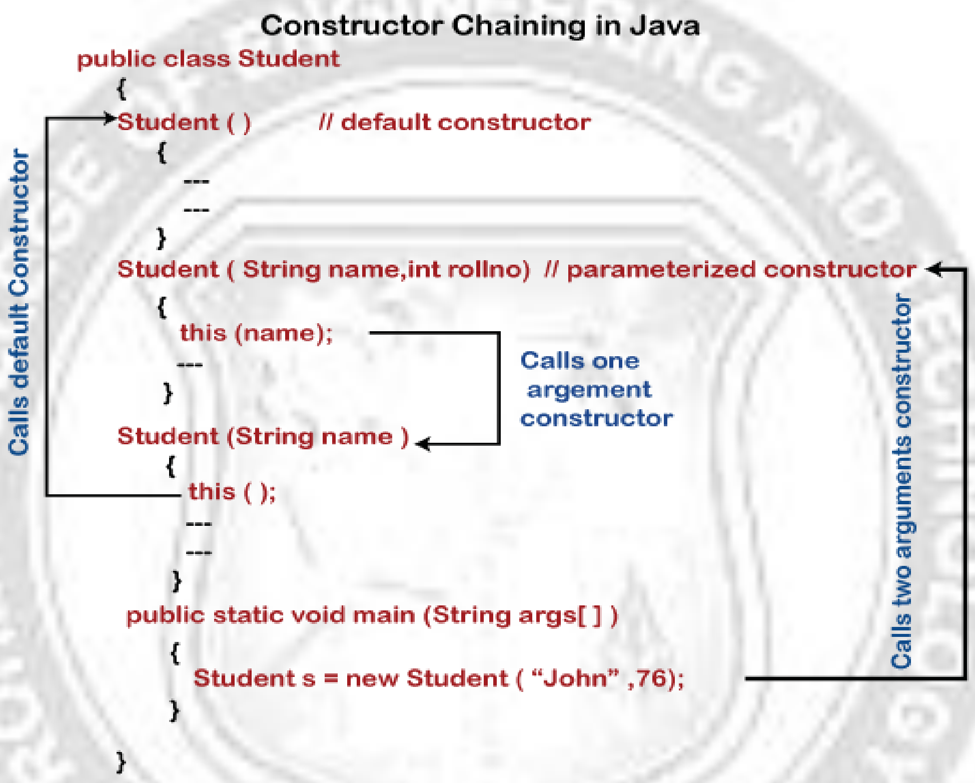
Constructor chaining is the process of calling one constructor of a class from another constructor of the same class or another class using the current object of the class.

- It occurs through inheritance.

Ways to achieve Constructor Chaining:

We can achieve constructor chaining in two ways:

- **Within the same class:** If we want to call the constructor from the same class, then we use **this** keyword.
- **From the base class:** If we want to call the constructor that belongs to different classes (parent and child classes), we use the **super** keyword to call the constructor from the base class.


Constructor Chaining in Java

```
public class Student
 {
  Student ( )        // default constructor
   {
     ---
     ---
   }
  Student ( String name,int rollno)  // parameterized constructor
   {
    this (name);
     ---
   }
  Student (String name )
   {
    this ( );
     ---
     ---
   }
  public static void main (String args[ ] )
   {
    Student s = new Student ( "John" ,76);
   }
 }
```
Calls default Constructor
Calls one argement constructor
Calls two arguments constructor

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

Advantage:

- ✓ Avoids duplicate code while having multiple constructors.
- ✓ Makes code more readable

Example

```
class Shape
{
  int radius,length,breadth;

  Shape(int radius)
  {
    this.radius=radius;
  }
```

```
   Shape(int r,int l,int b)
   {
      this(r);
      length=l;
      breadth=b;
   }

   void areaCircle()
   {
      System.out.println("Area of Circle is "+(3.14*radius*radius));
   }
   void areaRectangle()
   {
      System.out.println("Area of Rectangle is "+(length*breadth));
   }
}
public class ConstructorChaining
{
   public static void main(String arg[])
   {
   Shape s1=new Shape(5,10,50);
   s1.areaCircle();
   s1.areaRectangle();
   }
}
```

Output:

```
Area of Circle is 78.5
Area of Rectangle is 500
```