Constructor

Constructor

1. A constructor in Java is a block of code, syntactically similar to a method that is used to initialize the state of an object in a class.

2. In other words, a constructor is a special member function of a class used to initialize instance variables of a class.

3. The sole purpose of a constructor is to perform the initialization of data fields of an object in the class.

4. Java constructor can perform any action such as initializing the instance variables with legal initial values.

Constructor

5. It is automatically invoked when an instance of a class is created using the new operator.
6. Constructors can also accept arguments like methods and can be overloaded.
7. If we try to create an object of the class without specifying any constructor in the class, compiler automatically create a default constructor for us.
8. **Syntax to Declare Constructor:**

```
Access modifiers_name class_name(formal_parameter_list)
    {
        // Constructor body which is a block of statements.
        // Here, we can initialize the values of instance variables.
    }
```

9. A Java constructor cannot be – Static, Abstract, Final, Synchronized

Constructor Example

The following example code defines a constructor in the class.

```
// Employee Class
public class Employee
     {
          // Constructor
          public Employee()
          {
          // constructor code goes here.
          }
     }
```

Characteristics of Java Constructor

1. Constructor’s name must be exactly the same as the class name in which it is defined.
2. The constructor should not have any return type even void also because if there is a return type then JVM would consider as a method, not a constructor.
3. Java constructor may or may not contain parameters. Parameters are local variables to receive value (data) from outside into a constructor.

4. A constructor is automatically called and executed by JVM at the time of object creation. JVM (Java Virtual Machine) first allocates the memory for variables (objects) and then executes the constructor to initialize instance variables.

5. It is always called and executed only once per object. This means that when an object of a class is created, constructor is called. When we create second object, the constructor is again called during the second time.

Characteristics of Java Constructor

5. Whenever we create an object or instance of a class, the constructor automatically calls by the JVM .
6. If we don't define any constructor inside the class, Java compiler automatically creates a default constructor at compile-time and assigns default values for all variables declared in the class.
7. The default values for variables are as follows:
   a. Numeric variables are set to 0.
   b. Strings are set to null.
   c. Boolean variables are set to false.
8. Constructors provide thread safety, meaning no thread can access the object until the execution of constructor is completed.
9. We can do anything in the constructor similar to a method. Using constructors, we can perform all initialization activities of an object.

How to Call Constructor in Java

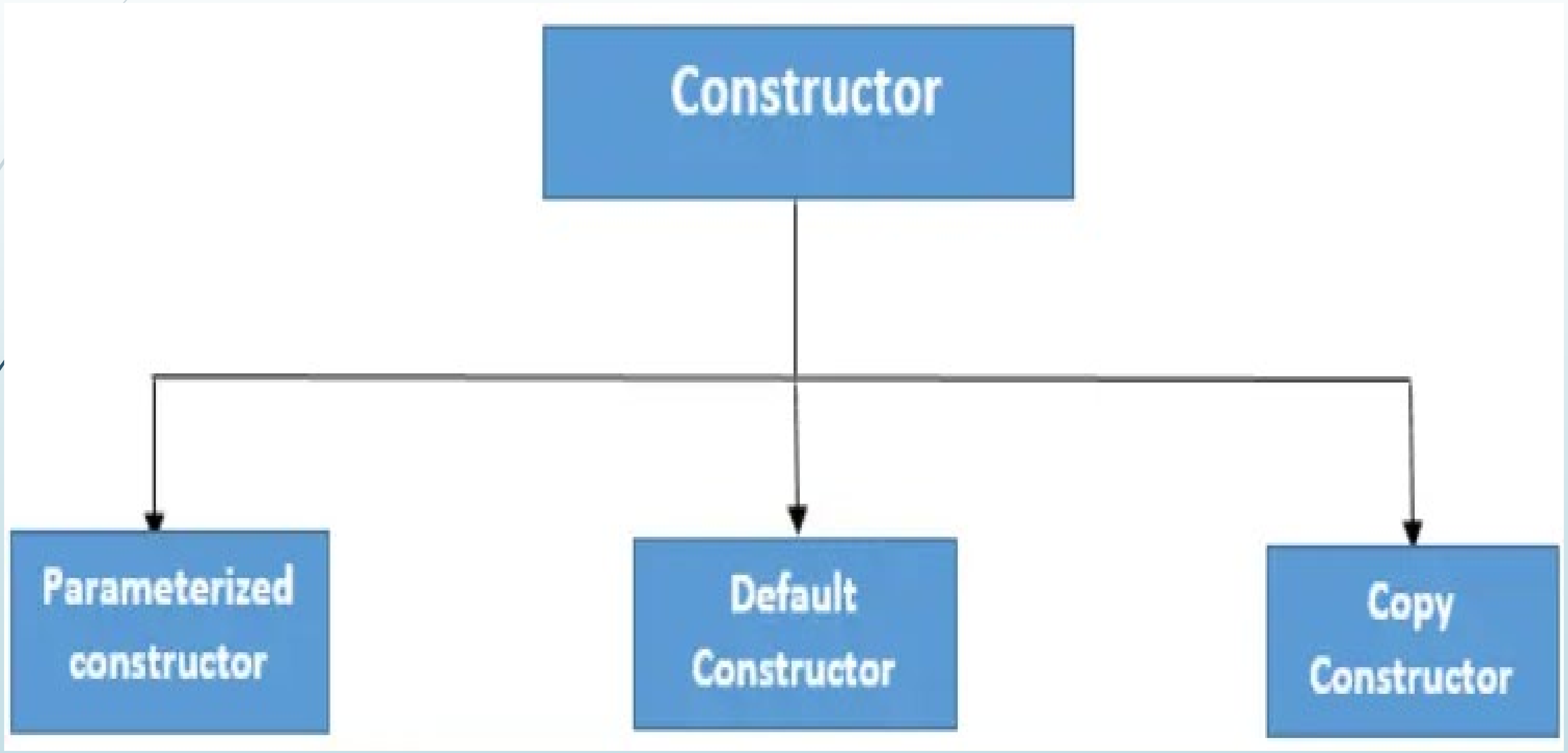
There are the following ways to call a constructor in java.

1. Employee emp = new Employee(); // Here, Employee is name of class.
2. new Employee(); // It is calling Employee() constructor.
3. super();
4. this();
5. Class.forName(“com.mysql.jdbc.Driver”).newInstance();

new keyword

- ❖ When we create an object of class by using new keyword, a constructor is automatically called by JVM.
- ❖ **new** is a special keyword that allocates the memory to store objects whose type is specified by a constructor.
- ❖ After allocation of memory, it calls constructor to initialize objects, which are stored in the heap (a region of memory for storing objects).
- ❖ When the constructor ends, a new keyword returns memory addresses to the object so that it can be accessed from anywhere in the application.

Types of Constructors in Java



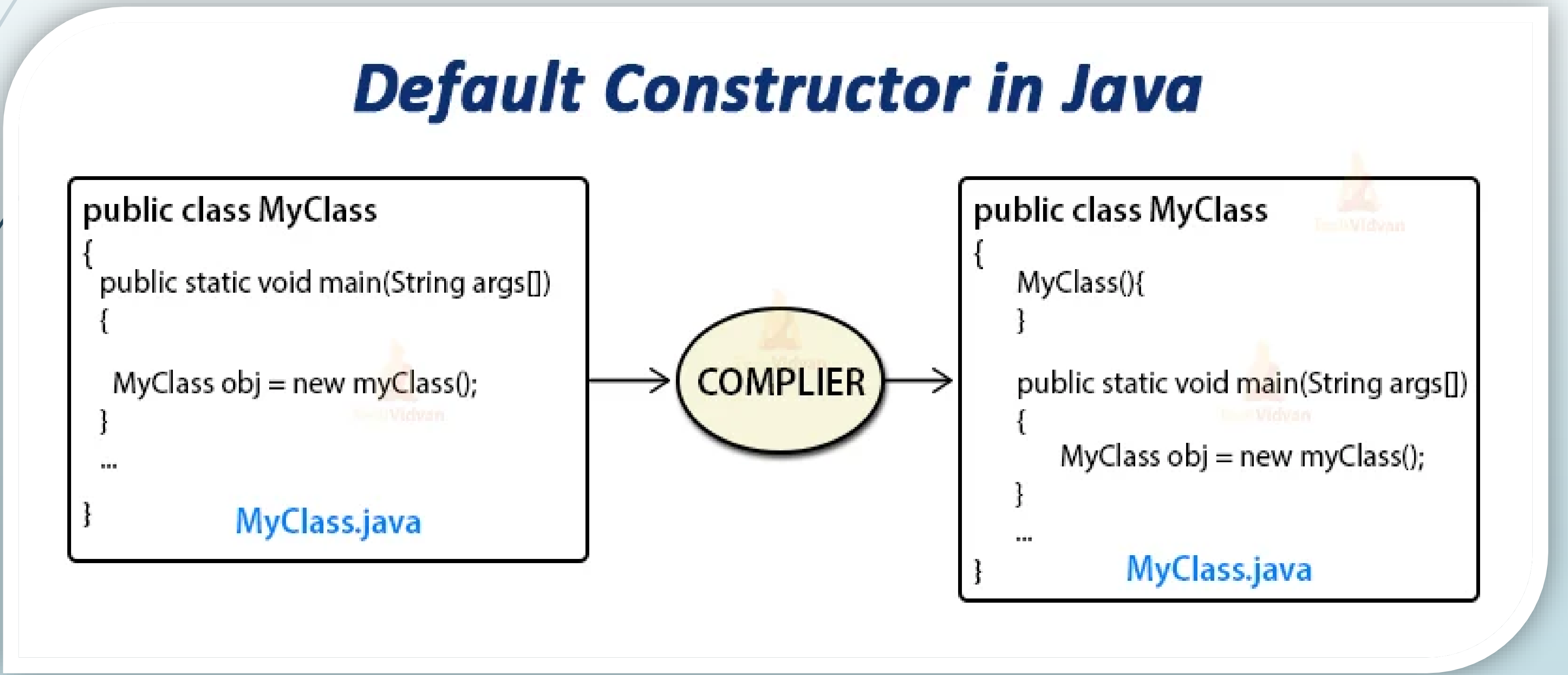
Types of Constructors in Java

Basically, there are four types of constructors in Java programming.

1. Default Constructor
2. Non-parameterized constructor
3. Parameterized Constructor
4. Copy Constructor

Default Constructor

A constructor that has no parameter is known as default constructor in Java. When a class does not declare a constructor, Java compiler automatically adds a constructor for that class.



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

    MyClass obj = new myClass();
  }
  ...

}
```

```
public class MyClass
{
    MyClass(){
    }

    public static void main(String args[])
    {
        MyClass obj = new myClass();
    }
    ...
}
```

Features of Default Constructor

1. When Java compiler adds a default constructor, it also adds a statement called super() to call the superclass constructor.
2. Default constructor does not accept any argument value and has no statements in its body except super() statement.
3. A default constructor automatically initializes instance variables with default values. For example, string is initialized by null value.
4. It has no **throws** clause.
5. The access modifiers of the default constructor should be the same as the access modifier of the class.
6. For example, if the top-level class is declared public, the default constructor should implicitly be declared as public.

Non-parameterized Constructor in Java

1. A constructor which has no parameters in the parentheses but contains statements inside its body is called a non-parametrized constructor.
2. We also call it as a non-argument constructor.
3. A non-parameterized constructor has the same signature as that of default constructor, except for only one difference between them.
4. Using non-parameterized constructor, we can initialize any values for the instance variables.

Remember: A reference variable never contains an object directly. It contains an address that points to data stored in the memory location.

Parameterized Constructor in Java

1. A constructor that takes one or more parameters and contains statements inside its body is called parameterized constructor in Java.
2. In the parameterized constructor, instance variables automatically initialize at runtime when we pass values to parameters during object creation.
3. The parameterized constructor is used to provide different values to distinct objects. It allows us to initialize instance variables with unlike values.
4. In the case of the default constructor, values remain the identical for all objects.
5. Example:

```
Employee (String name, int age) {
 // Constructor code.
}
```

Difference between Constructor and Methods in Java

S. No	Constructor	Method
1.	Constructor is a special type of method that is used to initialize the state of an object.	Method is used to expose the behaviour of an object.
2.	It has no return type even void also.	It has both void and return type.
3.	If we don't provide any constructor in the class, Java Compiler provides a default constructor for that class.	Method is not provided by the compiler in any case.
4.	Constructor name must be the same as name of the class.	Method name may or may not be the same name as the class name.
5.	The purpose of a constructor is to create an object of a class.	The purpose of a method is to execute the functionality of the application.
6.	They are not inherited by subclasses.	They are inherited by subclasses.

Some Points to remember about the Constructor

1. The task of a constructor in Java is to initialize instance variables of an object.
2. A constructor of class is automatically executed by JVM when an object is instantiated.
3. When an object of class is instantiated, constructor is called before any methods.
4. If you declare your own constructor, you have to decide what values should be given to instance variables.
5. **this** and **super** keyword must be the first line in the constructor.

Some Points to remember about the Constructor (Cont..)

6. Constructor overloading is possible but overriding is not possible.

7. It cannot be inherited in the subclass.

8. A constructor can also call another constructor of the same class using ‘**this**’ and for argument constructor use ‘**this(para_list**).

Constructor chaining in Java

1. Constructor chaining in Java is a technique of calling one constructor from within another constructor by using **this** and **super** keywords.

2. The keyword "**this**" is used to call a constructor from within another constructor in the same class.

3. The keyword "**super**" is used to call the parent (super) class constructor from within child (subclass) class constructor. It occurs through inheritance.

4. We cannot call the constructor directly by name because it is illegal in Java.

Order of Execution of Constructor in Inheritance Relationship

1. When we create a subclass object, the subclass constructor first calls its superclass constructor before performing its own tasks.

2. If the superclass is derived from another class, the superclass constructor calls its superclass constructor before performing its own tasks.

3. This process continues until it reaches the last chained constructor and the end of the chain will always object class constructor.

4. This is because every class is inherited from Object class by default. This is called constructor chaining in java.

Calling one constructor from another constructor in Java

To call one constructor from another constructor is called constructor chaining in java.

This process can be implemented in two ways:

1. Using **this()** keyword to call the current class constructor within the "same class".
2. Using **super()** keyword to call the superclass constructor from the "base class".

For example:

1. this(); ➡ To call default current class constructor.
2. this("name"); ➡ To call one parameter current class constructor with String argument.
3. this("John", 30) ➡ To call two parameters current class constructor with String and integer argument.

Super Class Constructor Call in Java

1. The statement super() calls the no-argument constructor of its superclass and the super(argument) invokes the superclass constructor where the argument must match.

   For example:

   1. **super();** ➡ To call the default superclass constructor.
   2. **super(10);** ➡ To call the superclass constructor with one int parameter.

2. We must use the **super()** keyword to call the superclass constructor but if you do not put a super keyword, Java compiler will put automatically the super() keyword.
3. The **super()** keyword must be in the first line of the constructor.
4. A constructor is used to create an object of a class.
5. Unlike properties and methods, the constructor of superclass cannot be inherited in the subclass.
6. It can only be called from the subclass constructor by using super keyword.

Object Class

1. **Object** class is the parent of all classes in Java.
2. **Object** class has a constructor with no parameters.
3. Therefore, the statement **super()** inside the child class constructor calls the constructor of Object.
4. When we create an object of subclass, all parent class constructor gets called, up to topmost class **Object**.
5. Thus, JVM executes constructors in the reverse order, from **top to bottom**.
6. We call this process as **constructor chaining** in Java.

Copy Constructor in Java

1. A constructor which is used to copy the data of one object to another object of the same class type is called copy constructor in Java.
2. A constructor that creates a copy of an existing object is called copy constructor.
3. It provides an easy and attractive mechanism to create an exact copy of an object from an object sent as an argument to the constructor.
4. A copy constructor is called when it takes a reference to an object of the same class as an argument and returns a new copy with the same values as the argument.
5. By default, Java compiler does not create any copy constructor in a class.
6. Like a constructor, a copy constructor also does not have a return type.

Private Constructor in Java

1. When a constructor is declared with a private access modifier in a class, it is called private constructor in Java.

2. It is visible within the same class where it has been declared.

3. Java private constructor is useful when we want to prevent users from creating an object of class from outside.

4. Only within the class, we can make a new object of class with private constructor.

5. For example, classes such as (**Math** class or **System** class) have private constructor. We can never instantiate them from outside of the class.

Use of Private Constructor in Java

1. We use private constructor when we want to prevent other classes from creating an object of a class.

2. We can use it in single tone classes where the object of the class cannot be created outside the class.

Advantages of Private Constructor

1. Private constructor is essential for implementing the singleton design pattern, ensuring that a class has only one instance across the entire application program.
2. It allows us to construct an immutable class where we cannot change the state of an object after creation.
3. Private constructors provide the class with complete control over its initialization process.
4. It prevents the external classes from creating objects.