Java

Oops Concepts:

**Object** means a real-world entity such as a pen, chair, table, computer, watch, etc.

**Object-Oriented Programming** is a methodology to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

* [Object](https://www.javatpoint.com/object-and-class-in-java)
* Class
* [Inheritance](https://www.javatpoint.com/inheritance-in-java)
* [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java)
* [Abstraction](https://www.javatpoint.com/abstract-class-in-java)
* [Encapsulation](https://www.javatpoint.com/encapsulation)

Apart from these concepts, there are some other terms which are used in Object-Oriented design:

* Coupling
* Cohesion
* Association
* Aggregation
* Composition

Object

Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

**Example:** A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Class

*Collection of objects* is called class. It is a logical entity.

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

### Inheritance

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

### Polymorphism

If one task is performed in different ways, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

In Java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

#### Abstraction

Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.

In Java, we use abstract class and interface to achieve abstraction.

### Encapsulation

Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

### Coupling

Coupling refers to the knowledge or information or dependency of another class. It arises when classes are aware of each other. If a class has the details information of another class, there is strong coupling. In Java, we use private, protected, and public modifiers to display the visibility level of a class, method, and field. You can use interfaces for the weaker coupling because there is no concrete implementation.

### Cohesion

Cohesion refers to the level of a component which performs a single well-defined task. A single well-defined task is done by a highly cohesive method. The weakly cohesive method will split the task into separate parts. The java.io package is a highly cohesive package because it has I/O related classes and interface. However, the java.util package is a weakly cohesive package because it has unrelated classes and interfaces.

### Association

Association represents the relationship between the objects. Here, one object can be associated with one object or many objects. There can be four types of association between the objects:

* One to One
* One to Many
* Many to One, and
* Many to Many

Let's understand the relationship with real-time examples. For example, One country can have one prime minister (one to one), and a prime minister can have many ministers (one to many). Also, many MP's can have one prime minister (many to one), and many ministers can have many departments (many to many).

Association can be undirectional or bidirectional.

### Aggregation

Aggregation is a way to achieve Association. Aggregation represents the relationship where one object contains other objects as a part of its state. It represents the weak relationship between objects. It is also termed as a has-a relationship in Java. Like, inheritance represents the is-a relationship. It is another way to reuse objects.

### Composition

The composition is also a way to achieve Association. The composition represents the relationship where one object contains other objects as a part of its state. There is a strong relationship between the containing object and the dependent object. It is the state where containing objects do not have an independent existence. If you delete the parent object, all the child objects will be deleted automatically.

# **Objects and Classes in Java**

In object-oriented programming technique, we design a program using objects and classes.

An object in Java is the physical as well as a logical entity, whereas, a class in Java is a logical entity only.

### **What is an object in Java**

An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

An object has three characteristics:

* **State:** represents the data (value) of an object.
* **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
* **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

For Example, Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.

**An object is an instance of a class.** A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

**Object Definitions:**

* An object is *a real-world entity*.
* An object is *a runtime entity*.
* The object is *an entity which has state and behavior*.
* The object is *an instance of a class*.

## What is a class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

* **Fields**
* **Methods**
* **Constructors**
* **Blocks**
* **Nested class and interface**

### Instance variable in Java

A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

### Method in Java

In Java, a method is like a function which is used to expose the behavior of an object.

#### **Advantage of Method**

* Code Reusability
* Code Optimization

### new keyword in Java

The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

## 3 Ways to initialize object

There are 3 ways to initialize object in Java.

1. By reference variable
2. By method
3. By constructor

## What are the different ways to create an object in Java?

* By new keyword
* By newInstance() method
* By clone() method
* By deserialization
* By factory method etc.

## Anonymous object

Anonymous simply means nameless. An object which has no reference is known as an anonymous object. It can be used at the time of object creation only.

If you have to use an object only once, an anonymous object is a good approach. For example:

**new** Calculation();//anonymous object

Calling method through a reference:

Calculation c=**new** Calculation();

c.fact(5);

Calling method through an anonymous object

**new** Calculation().fact(5);

Let's see the full example of an anonymous object in Java

**class** Calculation{

**void** fact(**int**  n){

**int** fact=1;

**for**(**int** i=1;i<=n;i++){

   fact=fact\*i;

   }

  System.out.println("factorial is "+fact);

}

**public** **static** **void** main(String args[]){

**new** Calculation().fact(5);//calling method with anonymous object

}

}

Output:

Factorial is 120

//Java Program to demonstrate the working of a banking-system

1. //where we deposit and withdraw amount from our account.
2. //Creating an Account class which has deposit() and withdraw() methods
3. **class** Account{

**int** acc\_no;

String name;

**float** amount;

**void** insert(**int** a,String n,**float** amt){

acc\_no=a;

name=n;

amount=amt;

}

//deposit method

**void** deposit(**float** amt){

amount=amount+amt;

System.out.println(amt+" deposited");

}

//withdraw method

**void** withdraw(**float** amt){

**if**(amount<amt){

System.out.println("Insufficient Balance");

}**else**{

amount=amount-amt;

System.out.println(amt+" withdrawn");

}

}

//method to check the balance of the account

**void** checkBalance(){

System.out.println("Balance is: "+amount);

}

//method to display the values of an object

**void** display(){

System.out.println(acc\_no+" "+name+" "+amount);

}

}

//Creating a test class to deposit and withdraw amount

**class** TestAccount{

**public** **static** **void** main(String[] args){

Account a1=**new** Account();

a1.insert(832345,"Ankit",1000);

a1.display();

a1.checkBalance();

a1.deposit(40000);

a1.checkBalance();

a1.withdraw(15000);

a1.checkBalance();

}}

# **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.

here are two types of constructors in Java: no-arg constructor, and parameterized constructor.

**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**

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

## Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

## Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

//Java Program to create and call a default constructor

**class** Bike1{

Bike1()

{

System.out.println("Bike is created");

}

//main method

**public** **static** **void** main(String args[]){

//calling a default constructor

Bike1 b=**new** Bike1();

}

}

### **Q) What is the purpose of a default constructor?**

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type

Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

Constructor Overloading in java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

**class** Student5{

**int** id;

    String name;

**int** age;

    //creating two arg constructor

    Student5(**int** i,String n){

     id = i;

     name = n;

    }

    //creating three arg constructor

    Student5(**int** i,String n,**int** a){

    id = i;

     name = n;

     age=a;

    }

**void** display(){

System.out.println(id+" "+name+" "+age);

}

**public** **static** **void** main(String args[]){

     Student5 s1 = **new** Student5(111,"Karan");

     Student5 s2 = **new** Student5(222,"Aryan",25);

     s1.display();

     s2.display();

    }

}

|  |  |
| --- | --- |
| **Java Constructor** | **Java Method** |
| A constructor is used to initialize the state of an object. | A method is used to expose the behavior of an object. |
| A constructor must not have a return type. | A method must have a return type. |
| The constructor is invoked implicitly. | The method is invoked explicitly. |
| The Java compiler provides a default constructor if you don't have any constructor in a class. | The method is not provided by the compiler in any case. |
| The constructor name must be same as the class name. | The method name may or may not be same as the class name. |

## Java Copy Constructor

There is no copy constructor in Java. However, we can copy the values from one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in Java. They are:

* By constructor
* By assigning the values of one object into another
* By clone() method of Object class

By Constructor:

**class** Student6{

**int** id;

    String name;

 Student6(**int** i,String n){

    id = i;

    name = n;

    }

    //constructor to initialize another object

    Student6(Student6 s){

    id = s.id;

    name =s.name;

    }

**void** display(){System.out.println(id+" "+name);}

**public** **static** **void** main(String args[]){

    Student6 s1 = **new** Student6(111,"Karan");

    Student6 s2 = **new** Student6(s1);

    s1.display();

    s2.display();

   }

}

## Copying values without constructor

We can copy the values of one object into another by assigning the objects values to another object. In this case, there is no need to create the constructor.

**class** Student7{

**int** id;

    String name;

    Student7(**int** i,String n){

    id = i;

    name = n;

    }

    Student7(){}

**void** display(){System.out.println(id+" "+name);}

**public** **static** **void** main(String args[]){

    Student7 s1 = **new** Student7(111,"Karan");

    Student7 s2 = **new** Student7();

    s2.id=s1.id;

    s2.name=s1.name;

    s1.display();

    s2.display();

   }

}

# **Object Cloning in Java**

The **object cloning** is a way to create exact copy of an object. The clone() method of Object class is used to clone an object.

The **java.lang.Cloneable interface** must be implemented by the class whose object clone we want to create. If we don't implement Cloneable interface, clone() method generates **CloneNotSupportedException**.

The **clone() method** is defined in the Object class. Syntax of the clone() method is as follows:

**protected** Object clone() **throws** CloneNotSupportedException

### **Why use clone() method ?**

The **clone() method** saves the extra processing task for creating the exact copy of an object. If we perform it by using the new keyword, it will take a lot of processing time to be performed that is why we use object cloning.

### **Advantage of Object cloning**

Although Object.clone() has some design issues but it is still a popular and easy way of copying objects. Following is a list of advantages of using clone() method:

* You don't need to write lengthy and repetitive codes. Just use an abstract class with a 4- or 5-line long clone() method.
* It is the easiest and most efficient way for copying objects, especially if we are applying it to an already developed or an old project. Just define a parent class, implement Cloneable in it, provide the definition of the clone() method and the task will be done.
* Clone() is the fastest way to copy array.

### **Disadvantage of Object cloning**

Following is a list of some disadvantages of clone() method:

* To use the Object.clone() method, we have to change a lot of syntaxes to our code, like implementing a Cloneable interface, defining the clone() method and handling CloneNotSupportedException, and finally, calling Object.clone() etc.
* We have to implement cloneable interface while it doesn't have any methods in it. We just have to use it to tell the JVM that we can perform clone() on our object.
* Object.clone() is protected, so we have to provide our own clone() and indirectly call Object.clone() from it.
* Object.clone() doesn't invoke any constructor so we don't have any control over object construction.
* If you want to write a clone method in a child class then all of its superclasses should define the clone() method in them or inherit it from another parent class. Otherwise, the super.clone() chain will fail.
* Object.clone() supports only shallow copying but we will need to override it if we need deep cloning.

### **Example of clone() method (Object cloning)**

**class** Student18 **implements** Cloneable{

**int** rollno;

String name;

Student18(**int** rollno,String name){

**this**.rollno=rollno;

**this**.name=name;

}

**public** Object clone()**throws** CloneNotSupportedException{

**return** **super**.clone();

}

**public** **static** **void** main(String args[]){

**try**{

Student18 s1=**new** Student18(101,"amit");

Student18 s2=(Student18)s1.clone();

System.out.println(s1.rollno+" "+s1.name);

System.out.println(s2.rollno+" "+s2.name);

}**catch**(CloneNotSupportedException c){}

}

}

Output:101 amit

101 amit

As you can see in the above example, both reference variables have the same value. Thus, the clone() copies the values of an object to another. So we don't need to write explicit code to copy the value of an object to another.

If we create another object by new keyword and assign the values of another object to this one, it will require a lot of processing on this object. So to save the extra processing task we use clone() method.

### **Q) Does constructor return any value?**

Yes, it is the current class instance (You cannot use return type yet it returns a value).

### **Can constructor perform other tasks instead of initialization?**

Yes, like object creation, starting a thread, calling a method, etc. You can perform any operation in the constructor as you perform in the method.

**Is there Constructor class in Java?**

Yes.

**What is the purpose of Constructor class?**

Java provides a Constructor class which can be used to get the internal information of a constructor in the class. It is found in the java.lang.reflect package.