**Unit – 1 => OOPS Concepts: Encapsulation, Inheritance and polymorphism, Classes and data abstraction, constructors and destructors**

**Previous Year Questions (2023)-**

1. **Part-A: What do you mean by Abstraction?**
2. **Part-A: What is the use of Constructor?**
3. **Part-B: Differentiate Constructor and Destructor.**
4. **Part-B: Explain Polymorphism with examples.**
5. **Part-C: Explain Inheritance with example.**

**Previous Year Questions (2022)-**

1. **Part-A: How constructor is created in java ?**
2. **Part-A: What is Polymorphism ?**
3. **Part-B: Explain the basics concept of OOP ?**
4. **Part-B: Explain Constructors and Destructors in java with examples ?**

**Previous Year Questions (2021)-**

1. **Part-A: What is Encapsulation ?**
2. **Part-A: Define Inheritance ?**
3. **Part-B: Explain overview of OOPS concepts ?**
4. **Part-B: Describe about structure of Java Program with an example ?**
5. **Part-B: Explain Constructors and Destructors in java ?**

# JAVA OOPS CONCEPTS

**Simula** is considered the first object-oriented programming language. **The programming paradigm where everything is represented as an object** is known as a truly object-oriented programming language. Object-Oriented Programming is **a methodology or paradigm to design a programs using classes and objects**.

**Smalltalk** is considered the first truly object-oriented programming language.

The popular object-oriented languages are Java, C#, PHP, Python, C++, etc.

The **main aim of object-oriented programming is to implement real-world entities**, for example, object, classes, abstraction, inheritance, polymorphism, etc.

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. It simplifies software development and maintenance by providing some concepts:

* Object
* Class
* Inheritance
* Polymorphism
* Abstraction
* Encapsulation

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

* Coupling
* Cohesion
* Association
* Aggregation
* Composition

## 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
* 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 unidirectional 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.

# What is an object in Java ?

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

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.

*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.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.

# What is a class in Java ?

**Collection of objects** is called class. A class is a **group of objects which have common properties**. It is a **logical entity**. It **can't be physical**.

A class can also be defined as a **template or blueprint from which objects are created**. Class **doesn't consume any space**. A class in Java can contain:

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

**Syntax to declare a class:**

class <class\_name>{

field;

method;

}

# 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.

# Object and Class Example: main within the class

In this example, we have created a Student class which has two data members id and name. We are creating the object of the Student class by new keyword and printing the object's value.

Here, we are creating a main() method inside the class.

//Java Program to illustrate how to define a class and fields

//Defining a Student class.

class Student {

//defining fields

int id; //field or data member or instance variable

String name;

//creating main method inside the Student class

public static void main(String args[]){

//Creating an object or instance

Student s1=new Student(); //creating an object of Student

//Printing values of the object

System.out.println(s1.id); //accessing member through reference variable

System.out.println(s1.name);

}

}

### Output:

0

Null

# Object and Class Example: main outside the class

In real time development, we create classes and use it from another class. It is a better approach than previous one. Let's see a simple example, where we are having main() method in another class.

We can have multiple classes in different Java files or single Java file. If you define multiple classes in a single Java source file, it is a good idea to save the file name with the class name which has main() method.

//Java Program to demonstrate having the main method in another class

//Creating Student class. class Student {

int id;

String name;

}

//Creating another class TestStudent1 which contains the main method

class TestStudent1 {

public static void main(String args[]){

Student s1=new Student(); System.out.println(s1.id); System.out.println(s1.name);

}

}

### Output:

0

Null

# 3 Ways to initialize object

There are 3 ways to initialize object in Java.

* By reference variable
* By method
* By constructor

## Object and Class Example: Initialization through reference

Initializing an object means storing data into the object. Let's see a simple example where we are going to initialize the object through a reference variable.

class Student {

int id;

String name;

}

class TestStudent2 {

public static void main(String args[]){

Student s1 = new Student();

s1.id=101;

s1.name="Sonoo";

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

}

}

Output:

101 Sonoo

## Object and Class Example: Initialization through method

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method. Here, we are displaying the state (data) of the objects by invoking the displayInformation() method.

class Student {

int rollno;

String name;

void insertRecord (int r, String n){

rollno=r;

name=n;

}

void displayInformation(){

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

}

}

class TestStudent4{

public static void main(String args[]){

Student s1=new Student();

Student s2=new Student();

s1.insertRecord(111,"Karan");

s2.insertRecord(222,"Aryan");

s1.displayInformation();

s2.displayInformation();

}

}

**Output**:

111 Karan

222 Aryan



As you can see in the above figure, object gets the memory in heap memory area. The reference variable refers to the object allocated in the heap memory area. Here, s1 and s2 both are reference variables that refer to the objects allocated in memory.

## Object and Class Example: Initialization through a constructor

## Object and Class Example: Employee

Let's see an example where we are maintaining records of employees.

class Employee {

int id;

String name;

float salary;

void insert(int i, String n, float s){

id=i;

name=n;

salary=s;

}

void display(){

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

}

}

public class TestEmployee {

public static void main(String[] args){

Employee e1=new Employee();

Employee e2=new Employee();

Employee e3=new Employee();

e1.insert(101,"ajeet",45000); e2.insert(102,"irfan",25000); e3.insert(103,"nakul",55000); e1.display();

e2.display();

e3.display();

}

}

### Output:

101 ajeet 45000.0

102 irfan 25000.0

103 nakul 55000.0

## Object and Class Example: Rectangle

## 

class Rectangle {

int length;

int width;

void insert(int l, int w){

length=l;

width=w;

}

void calculateArea(){

System.out.println(length\*width);

}

}

class TestRectangle1{

public static void main(String args[]){

Rectangle r1=new Rectangle();

Rectangle r2=new Rectangle();

r1.insert(11,5);

r2.insert(3,15); r1.calculateArea(); r2.calculateArea();

}

}

### Output:

55

45

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

There are many ways to create an object in java. They are:

* 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

## Creating multiple objects by one type only

We can create multiple objects by one type only as we do in case of primitives.

Initialization of primitive variables:

int a=10, b=20;

Initialization of reference variables:

Rectangle r1=new Rectangle(),r2=new Rectangle();

Let's see the example:

//Java Program to illustrate the use of Rectangle class which has length and width data members

class Rectangle {

int length;

int width;

void insert(int l, int w){

length=l;

width=w;

}

void calculateArea(){

System.out.println(length\*width);

}

}

class TestRectangle2 {

public static void main(String args[]){

Rectangle r1=new Rectangle(),r2=new Rectangle();

r1.insert(11,5);

r2.insert(3,15); r1.calculateArea(); r2.calculateArea();

}

}

### Output:

55

45

# Inheritance in Java

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

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

## Why use inheritance in java

* For Method Overriding (so runtime polymorphism can be achieved).
* For Code Reusability.

## Terms used in Inheritance

1. **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
2. **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
3. **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
4. **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

## The syntax of Java Inheritance

class Subclass-name extends Superclass-name

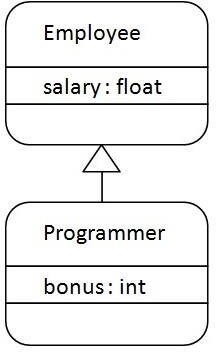
{

//methods and fields

}

The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality. In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

As displayed in the above figure, Programmer is the subclass and Employee is the superclass. The relationship between the two classes is Programmer IS-A Employee. It means that Programmer is a type of Employee.



class Employee {

float salary = 40000;

}

class Programmer extends Employee {

int bonus = 10000;

public static void main(String args[]){

Programmer p=new Programmer();

System.out.println("Programmer salary is:"+p.salary);

System.out.println("Bonus of Programmer:"+p.bonus);

}

}

### Output:

Programmer salary is: 40000.0

Bonus of programmer is: 10000

In the above example, Programmer object can access the field of own class as well as of Employee class i.e. code reusability.

## Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only.

# 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.

There 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

*Note:* We can use access modifiers while declaring a constructor. It controls the object creation. In other words, we can have private, protected, public or default constructor in Java.

## Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

## Java Default Constructor

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

Syntax of default constructor:

<class\_name>(){}

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 {

//creating a default constructor

Bike1() {

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

}

//main method

public static void main(String args[]){

//calling a default constructor

Bike1 b=new Bike1();

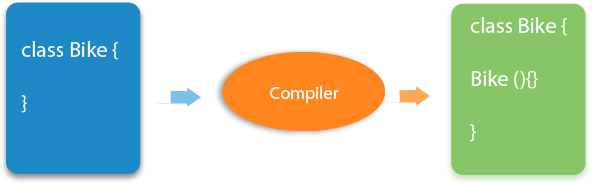
}

}

### Output:

Bike is created

***Rule:*** If there is no constructor in a class, compiler automatically creates a default constructor.



## Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

## Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

//Java Program to demonstrate the use of the parameterized constructor.

class Student4{

int id;

String name;

//creating a parameterized constructor

Student4(int i,String n){

id = i;

name = n;

}

//method to display the values

void display(){

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

}

public static void main(String args[]){

//creating objects and passing values

Student4 s1 = new Student4(111,"Karan");

Student4 s2 = new Student4(222,"Aryan");

//calling method to display the values of object s1.display();

s2.display();

}

}

### Output:

111 Karan

222 Aryan

## 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.

## Example of Constructor Overloading

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();

}

}

### Output:

111 Karan 0

222 Aryan 25

## DIFFERENCE BETWEEN CONSTRUCTOR AND METHOD IN JAVA

|  |  |
| --- | --- |
| **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

In this example, we are going to copy the values of one object into another using Java constructor.

//Java program to initialize the values from one object to another object.

class Student6{

int id;

String name;

//constructor to initialize integer and string  
 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();

}

}

### Output:

111 Karan

111 Karan

## 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();

}

}

### Output:

111 Karan

111 Karan

**Java Destructor**

In programming, a **destructor** is a method called when an object is destroyed, typically to release resources like memory, file handles, or network connections. In languages like C++, destructors are explicitly used to manage resource cleanup. However, **Java takes a different approach** due to its automatic memory management system powered by **garbage collection**.

**Destructor in Java**

Java does **not** have explicit destructors like C++ because the garbage collector automatically manages Java’s memory. **Garbage collection** tracks objects in memory, and when they are no longer referenced, it automatically reclaims the memory.

Although Java does not provide destructors, it used to have a method called **finalize()** that could be overridden to define cleanup operations before an object is garbage collected. The **finalize() method has been deprecated** since **Java 9** and removed in **Java 18** due to its unpredictability and performance overhead. Instead, modern Java programs rely on other mechanisms for resource management, such as **AutoCloseable** and **try-with-resources**.

**Advantages of Destructors**

* Automatic Resource Management: By automating the process of freeing up memory and other resources, Java’s garbage collector lowers the possibility of memory leaks and improper resource management.
* Simplifies Programming: By eliminating the need for developers to manually monitor and release memory, programming becomes simpler and less prone to error.
* Enhances Performance: By effectively handling memory allocation and deallocation, automated memory management can aid in performance optimization.

**Memory Management in Java**

* Java’s **garbage collector** automatically manages the lifecycle of objects.
* When no references point to an object, it becomes eligible for garbage collection.
* The **garbage collector** reclaims memory at its discretion, making memory management automated and simplifying development.

**Why Java Doesn’t Need Destructors?**

In languages like C++, destructors are essential because programmers manually manage memory allocation and deallocation. In contrast, Java developers don’t have to worry about this due to its automatic garbage collection system. Instead of destructors, developers in Java can handle resource management using the **try-with-resources** statement and **AutoCloseable** interface, ensuring resources like files, database connections, or sockets are properly closed.

class Resource implements AutoCloseable {

public Resource() {

System.out.println("Resource acquired.");

}

@Override

public void close() {

System.out.println("Resource released.");

}

}

public class Main {

public static void main(String[] args) {

try (Resource res = new Resource()) {

System.out.println("Using the resource.");

} // Resource is automatically closed here

}

}

**Output**

Resource acquired.

Using the resource.

Resource released.

**CONSTRUCTOR VS DESTRUCTOR**

| **CONSTRUCTOR (JAVA)** | **DESTRUCTOR (JAVA)** |
| --- | --- |
| A constructor is used to initialize the object of a class in Java. | There is no explicit destructor in Java; instead, memory management is handled by the **Garbage Collector**. |
| It is declared as className(parameters) {}. | Java does not have destructors. The finalize() method was used but is now deprecated in favor of automatic garbage collection. |
| A constructor can accept parameters to initialize the object. | There are no arguments for a destructor since it does not exist in Java. |
| A constructor is invoked when a new object is created using the new keyword. | The finalize() method (deprecated) was called by the garbage collector just before object destruction, but generally, object cleanup is automatic. |
| It is used to allocate memory and initialize object properties. | Memory deallocation is handled automatically by the garbage collector in Java. |
| Constructors can be overloaded by defining multiple constructors with different parameter lists. | There is no destructor in Java to overload. |
| The constructor’s name must be the same as the class name. | Since destructors do not exist in Java, there is no such naming convention. |
| There can be multiple constructors in a class with different parameters (constructor overloading). | Java uses garbage collection for memory management, so there is no need for a destructor. |
| Java supports the concept of a copy constructor, though it must be explicitly defined if needed. | Java handles object destruction automatically; no copy destructor concept exists. |
| Constructors are invoked during object creation in the order defined. | Java garbage collection works in the background and determines the destruction order automatically. |