**Question Bank**

**Module 2**

**Object Oriented Programming using Java (21CIC34)**

**Semester:3rd**

1. What is a class? What are its characteristics? Give its general structure. ( CO2)

Class is a set of objects which shares common characteristics/ behavior and common properties/ attributes.  
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 group of methods.

Class is a template, blue print or contract that defines what an object field and method will be



Defining a Class in Java

Java provides a reserved keyword **class** to define a class. The keyword must be followed by the class name. Inside the class, we declare methods and variables.

In general, class declaration includes the following in the order as it appears:

1. **Modifiers:** A class can be public or has default access.
2. **class keyword:** The class keyword is used to create a class.
3. **Class name:** The name must begin with an initial letter (capitalized by convention).
4. **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.
5. **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.
6. **Body:** The class body surrounded by braces, { }.

### **Syntax:**

|  |
| --- |
| * Class defines a new data type.   class classname  { type instance-variable1;  type instance-variable2;  ……….  ……….  type methodname1(parameter list)  {  ……// body of method  }  type methodnameN(parameter list)  {  ……// body of method  } } |

1. What is Constructor.Differentiate constructors and methods.What is constructor overloading? Explain Each type with an example. ( CO2)

# **Constructors in Java**

In [Java](https://www.javatpoint.com/java-tutorial)

, a constructor is a block of codes similar to the method. It is called when an instance of the [class(object)](https://www.javatpoint.com/object-and-class-in-java)

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.

### **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

Difference between constructor and method in Java

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

|  |  |
| --- | --- |
| **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. |

1. Distinguish between the abstract class and interface. ( CO2)

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |
| 6) An **abstract class** can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| 7) An **abstract class** can be extended using keyword "extends". | An **interface** can be implemented using keyword "implements". |
| 8) A Java **abstract class** can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| 9)**Example:** public abstract class Shape{ public abstract void draw(); } | **Example:** public interface Drawable{ void draw(); } |

1. What is inheritance? Mention the different types. Demonstrate Multi level inheritance with an example. ( CO2)

It is **the mechanism in java by which one class is allowed to inherit the features(fields and methods) of another class**.

Important terminology:

• **Super Class or a base class or a parent class**:

The class whose features are inherited is known as super class

• **Sub Class or a derived class, extended class, or child class**:

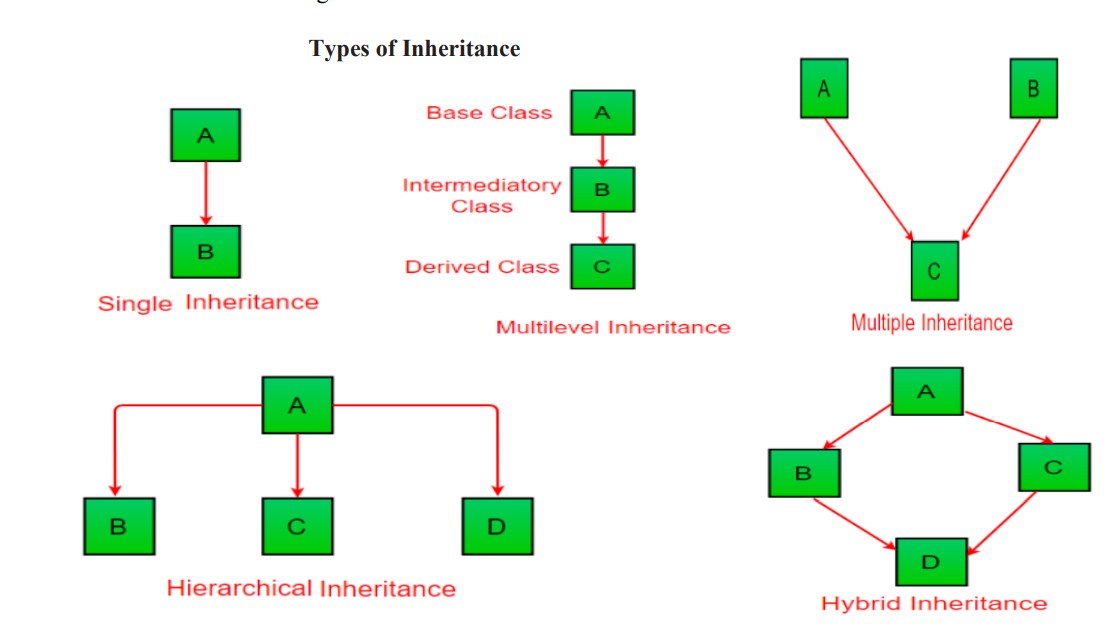
\*The class that acquires or inherits the other class features is known as sub class().

\*The subclass can add its own fields and methods in addition to the superclass fields and methods.

• Reusability:

Inheritance supports the concept of ―reusability‖, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class.

Types of inheritance:



Ex of multilevel inheritance

class Car{

public Car()

{

System.out.println("Class Car");

}

public void vehicleType()

{

System.out.println("Vehicle Type: Car");

}

}

class Maruti extends Car{

public Maruti()

{

System.out.println("Class Maruti");

}

public void brand()

{

System.out.println("Brand: Maruti");

}

public void speed()

{

System.out.println("Max: 90Kmph");

}

}

public class Maruti800 extends Maruti{

public Maruti800()

{

System.out.println("Maruti Model: 800");

}

public void speed()

{

System.out.println("Max: 80Kmph");

}

public static void main(String args[])

{

Maruti800 obj=new Maruti800();

obj.vehicleType();

obj.brand();

obj.speed();

}

}

Output:

|  |
| --- |
| Class Car //first 3 lines are output public class without void  Class Maruti  Maruti Model: 800  Vehicle Type: Car  Brand: Maruti  Max: 80Kmph |

1. Demonstrate the use of super and this keyword with an example. ( CO2)

Super : [link](https://www.javatpoint.com/super-keyword#:~:text=1)%20super%20is%20used%20to,child%20class%20have%20same%20fields.&text=In%20the%20above%20example%2C%20Animal,have%20a%20common%20property%20color.)

This : [link](https://www.javatpoint.com/this-keyword)

1. Discuss static and runtime polymorphism with example. ( CO2)

### 1. Static polymorphism (or compile-time polymorphism)

Like most of the other OOP programming languages, [Java polymorphism](https://www.upgrad.com/blog/polymorphism-in-oops/) allows the incorporation of multiple methods within a class. The methods use the same name but the parameter varies. This represents the static polymorphism. This polymorphism is resolved during the compiler time and is achieved through the method overloading.  (Compile-time Polymorphism) Static Polymorphism in Java **decides which method to execute during compile time**. In static polymorphism, the object would behave differently for the same trigger. This is why multiple methods are incorporated into the same class.

* The parameters number should vary.
* The parameter types should be different.
* Different order of parameters. For example, if a method accepts a string and a long, while the other method accepts a long and a string. However, this type of order makes it difficult for the API to understand.

Due to the difference in the parameters, every method has a different signature. The Java compiler has an idea of which method is called.

class MethodOverloading {

// this method accepts int

private static void display(int a){

System.out.println("Got Integer data.");

}

// this method accepts String object

private static void display(String a){

System.out.println("Got String object.");

}

public static void main(String[] args) {

display(1);

display("Hello");

}

}

**Output**:

Got Integer data.

Got String object.

Here, both overloaded methods accept one argument. However, one accepts the argument of type int whereas other accepts String object.

Run time polymorphism:

class Animal {

public void displayInfo() {

System.out.println("I am an animal.");

}

}

class Dog extends Animal {

@Override

public void displayInfo() {

System.out.println("I am a dog.");

}

}

class Main {

public static void main(String[] args) {

Dog d1 = new Dog();

d1.displayInfo();

}

}

1. Write short notes on a) final class b) abstract class. ( CO2)

Final class:

final class:

1)if u declare a class as final u cant use it for extending i.e., for inheritance purpose bcz see if u have a class x and if u want another class y (class y extend x) to extend its properties then see u can say like class(refer 17-26)) y inheriting  class x properties we are not changing values then why cant we extend y with class final x.

the reason is :(

     #see if u inherit class x properties into y u can even change its values in class y as class x fields are int class y;

Input:

final class hansi {

    int x = 10;

}

class hansi1 extends hansi {

    void kkr() {

        x = 20;

    }

}

Output:

The type hansi1 cannot subclass the final class hansi

Another example:

## Java final class

If you make any class as final, you cannot extend it.

### **Example of final class**

1. **final** **class** Bike{}
3. **class** Honda1 **extends** Bike{
4. **void** run(){System.out.println("running safely with 100kmph");}
6. **public** **static** **void** main(String args[]){
7. Honda1 honda= **new** Honda1();
8. honda.run();
9. }
10. }

Output:Compile Time Error

ABSTRACT CLASS:

### **Abstract class in Java**

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

#### **Points to Remember**

* An abstract class must be declared with an abstract keyword.
* It can have abstract and non-abstract methods.
* It cannot be instantiated.
* It can have [constructors](https://www.javatpoint.com/java-constructor) and static methods also.
* It can have final methods which will force the subclass not to change the body of the method.

1. A final method can not be overridden. Illustrate with an example. ( CO2)

|  |
| --- |
| **Example of final method**  1. **class** Bike{ 2. **final** **void** run(){System.out.println("running");} 3. } 5. **class** Honda **extends** Bike{ 6. **void** run(){System.out.println("running safely with 100kmph");} 8. **public** **static** **void** main(String args[]){ 9. Honda honda= **new** Honda(); 10. honda.run(); 11. } 12. }   Output:Compile Time Error |

1. Create a Java Class “Shape” with constructor to initialize the one parameter “dimension”.Now create three sub classes of Shape with following methods (i) “Circle” with methods to calculate the area and circumference of the circle with dimension as radius. (ii) “Square” with methods to calculate the area and length of diagonal of the square with dimension as length of one side. (assuming length of each side of the square is same). (iii) “Sphere” with methods to calculate the volume and surface area of the sphere with dimension as radius of the sphere. Write appropriate main method to create object of each class and test every method. ( CO2)
2. Explain garbage collection with its advantages and disadvantages. ( CO2)

in java, garbage means unreferenced objects.

Garbage Collection is a way to destroy the unused objects.

To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

### **Advantage of Garbage Collection**

* It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
* It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

## How can an object be unreferenced?

There are many ways:

* By nulling the reference
* By assigning a reference to another
* By anonymous object etc.

### **) By nulling a reference:**

1. Employee e=**new** Employee();
2. e=**null**;

### **2) By assigning a reference to another:**

1. Employee e1=**new** Employee();
2. Employee e2=**new** Employee();
3. e1=e2;//now the first object referred by e1 is available for garbage collection

### **3) By anonymous object:**

**new** Employee();

## finalize() method

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as

**protected** **void** finalize(){}

#### **Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).**

## gc() method

The gc() method is used to invoke the garbage collector explicitly to perform cleanup processing. The gc() is found in System and Runtime classes as

**public** **static** **void** gc(){}

#### **Note: Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.**

### **Simple Example of garbage collection in java**

1. **public** **class** TestGarbage1{
2. **protected** **void** finalize()
3. {
4. System.out.println("object is garbage collected");
5. }
6. **public** **static** **void** main(String args[]){
7. TestGarbage1 s1=**new** TestGarbage1();
8. TestGarbage1 s2=**new** TestGarbage1();
9. s1=**null**;
10. s2=**null**;
11. System.gc();
12. }
13. }

**[Test it Now](https://www.javatpoint.com/opr/test.jsp?filename=TestGarbage1" \t "_blank)**

object is garbage collected

object is garbage collected

#### **Note: Neither finalization nor garbage collection is guaranteed.**

1. Write a short note on finalize() method. ( CO2)

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

**protected** **void** finalize(){}

#### **Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).**

1. Explain the characteristics of a static variable and static method with example( CO2)

Java static variable

If you declare any variable as static, it is known as a static variable.

* The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
* The static variable gets memory only once in the class area at the time of class loading.
* Static variable maintains only one copy
* One more thing u declare var as static u can call them in main method directly without creating an obj and calling through that reference

|  |
| --- |
| public class staticvar  {      static int x=12;      public static void main(String[] args) {          staticvar obj=new staticvar();          obj.x=50;          System.out.println(obj.x);          System.out.println(x);          /\* here see the obj.x and x prints the same value           bcz              1)first thing here normally object creates new instance of class but here as x is a static variable eventhoug new instance ante obj is created there wont be new allocation of memomory for x as it is static so the obj.x will be pointing to same memory location.x is maintained as only one copy that is why if u change the value by using obj.x the value of x is also changed           \*/      }  } |

Static method

STATIC METHODS:

    A static method can be invoked(called or executed) without the need for creating an instance of

       a class.

       refer static\_method program once

|  |
| --- |
| public class static\_method {      static void hansi()      {          int x=10;          System.out.println("from static mtethod");      }      public static void main(String args[])      {          hansi();//called without creating an object          System.out.println("from main method");      }  } |

1. Write short notes on a) final class b) abstract class( CO2)
2. **Write a note on different access specifiers.** ( CO2)

**Access Modifiers** - controls the access level

## Access Modifiers

For **classes**, you can use either public or default:

|  |  |  |
| --- | --- | --- |
| **Modifier** | **Description** | **Try it** |
| public | The class is accessible by any other class | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_mod_public) |
| default | The class is only accessible by classes in the same package. This is used when you don't specify a modifier. You will learn more about packages in the [Packages chapter](https://www.w3schools.com/java/java_packages.asp) | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_mod_default) |

For **attributes, methods and constructors**, you can use the one of the following:

|  |  |  |
| --- | --- | --- |
| **Modifier** | **Description** | **Try it** |
| public | The code is accessible for all classes | [Try it »](https://www.w3schools.com/java/tryjava_multi.asp?filename=demo_mod_public2&multi=demo_mod_public2_multi) |
| private | The code is only accessible within the declared class | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_access_mod) |
| default | The code is only accessible in the same package. This is used when you don't specify a modifier. You will learn more about packages in the [Packages chapter](https://www.w3schools.com/java/java_packages.asp) | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_mod_default2) |
| protected | The code is accessible in the same package and **subclasses**. You will learn more about subclasses and superclasses in the [Inheritance chapter](https://www.w3schools.com/java/java_inheritance.asp) |  |

1. How do you achieve run time polymorphism in java? Illustrate with an example. ( CO2)

Refer 6th question

1. Explain Dynamic Method Dispatch (DMD) with example. ( CO2)

Method overriding is one of the ways in which Java supports Runtime Polymorphism. Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.

sir methodoverloading==compiletimepolymoerphism==static polymorphism

methodoverriding==runtimepolymorphism==dynamic method dispatch