**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.

**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

**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**

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();

}

}

**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.

Example of default constructor that displays the default values

class Student3{

int id;

String name;

//method to display the value of id and name

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

public static void main(String args[]){

//creating objects

Student3 s1=new Student3();

Student3 s2=new Student3();

//displaying values of the object

s1.display();

s2.display();

}

}

**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.

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();

}

}

**JAVA STATIC KEYWORD**

The static keyword in Java is used for memory management mainly. We can apply static keyword with variables, methods, blocks. The static keyword belongs to the class than an instance of the class.

The static can be:

Variable (also known as a class variable)

Method (also known as a class method)

Block

**1) 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.

**Advantages of static variable**

It makes your program memory efficient (i.e., it saves memory).

Understanding the problem without static variable

class Student{

int rollno;

String name;

String college="ITS";

}

Suppose there are 500 students in my college, now all instance data members will get memory each time when the object is created. All students have its unique rollno and name, so instance data member is good in such case. Here, "college" refers to the common property of all objects. If we make it static, this field will get the memory only once.

**Example of static variable**

//Java Program to demonstrate the use of static variable

class Student{

int rollno;//instance variable

String name;

static String college ="ITS";//static variable

//constructor

Student(int r, String n){

rollno = r;

name = n;

}

//method to display the values

void display (){System.out.println(rollno+" "+name+" "+college);}

}

//Test class to show the values of objects

public class TestStaticVariable1{

public static void main(String args[]){

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

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

//we can change the college of all objects by the single line of code

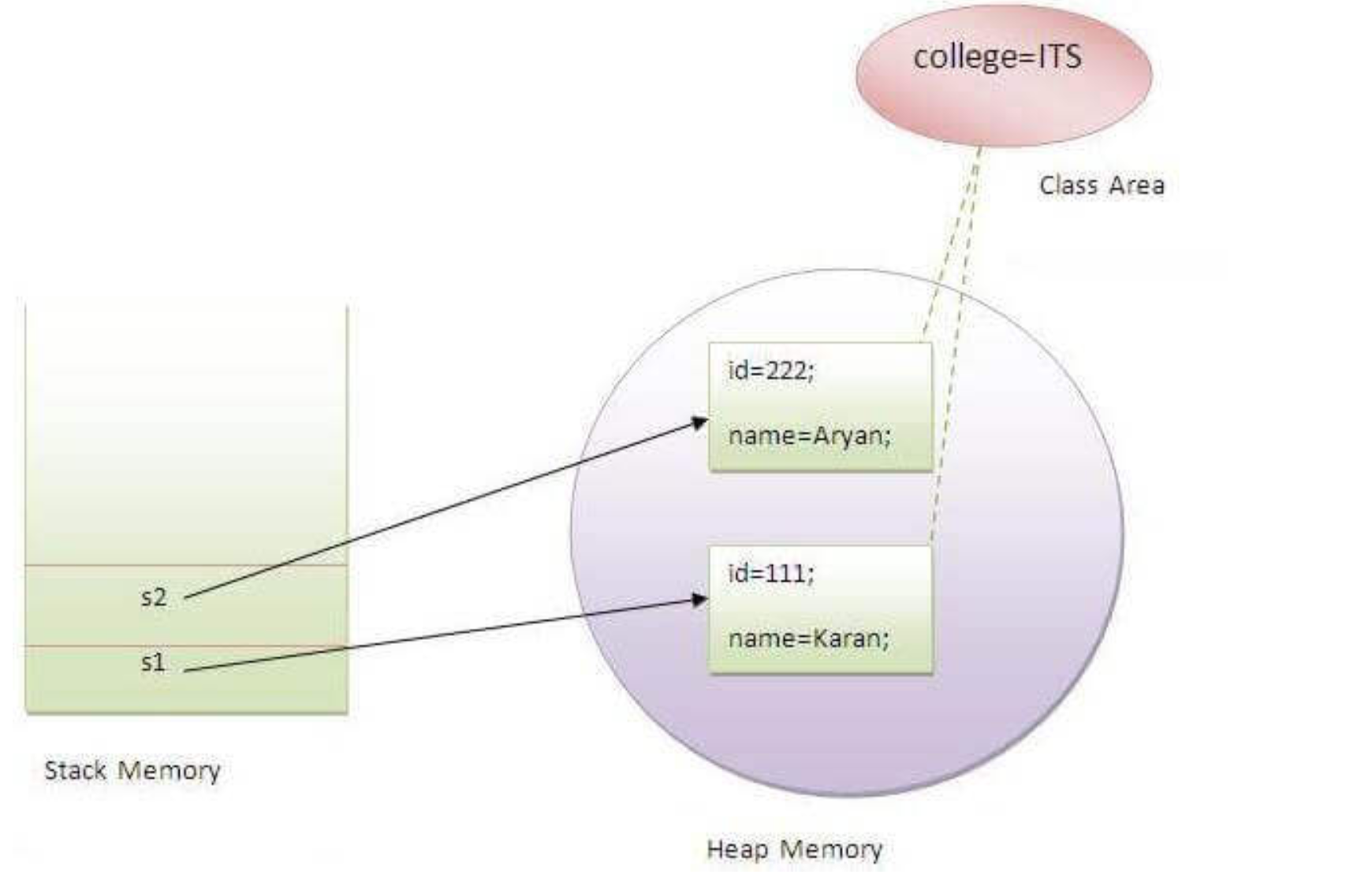
//Student.college="BBDIT";

s1.display();

s2.display();

}

}



**Program of the counter without static variable**

In this example, we have created an instance variable named count which is incremented in the constructor. Since instance variable gets the memory at the time of object creation, each object will have the copy of the instance variable. If it is incremented, it won't reflect other objects. So each object will have the value 1 in the count variable.

//Java Program to demonstrate the use of an instance variable

//which get memory each time when we create an object of the class.

class Counter{

int count=0;//will get memory each time when the instance is created

Counter(){

count++;//incrementing value

System.out.println(count);

}

public static void main(String args[]){

//Creating objects

Counter c1=new Counter();

Counter c2=new Counter();

Counter c3=new Counter();

}

}

**Program of counter by static variable**

As we have mentioned above, static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value.

//Java Program to illustrate the use of static variable which

//is shared with all objects.

class Counter2{

static int count=0;//will get memory only once and retain its value

Counter2(){

count++;//incrementing the value of static variable

System.out.println(count);

}

public static void main(String args[]){

//creating objects

Counter2 c1=new Counter2();

Counter2 c2=new Counter2();

Counter2 c3=new Counter2();

}

}

**Java static method**

If you apply static keyword with any method, it is known as static method.

A static method belongs to the class rather than the object of a class.

A static method can be invoked without the need for creating an instance of a class.

A static method can access static data member and can change the value of it.

**Example of static method**

//Java Program to demonstrate the use of a static method.

class Student{

int rollno;

String name;

static String college = "ITS";

//static method to change the value of static variable

static void change(){

college = "BBDIT";

}

//constructor to initialize the variable

Student(int r, String n){

rollno = r;

name = n;

}

//method to display values

void display(){System.out.println(rollno+" "+name+" "+college);}

}

//Test class to create and display the values of object

public class TestStaticMethod{

public static void main(String args[]){

Student.change();//calling change method

//creating objects

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

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

Student s3 = new Student(333,"Sonoo");

//calling display method

s1.display();

s2.display();

s3.display();

}

}

**INHERITANCE IN JAVA**

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**

**Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.

**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.

**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.

**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

{

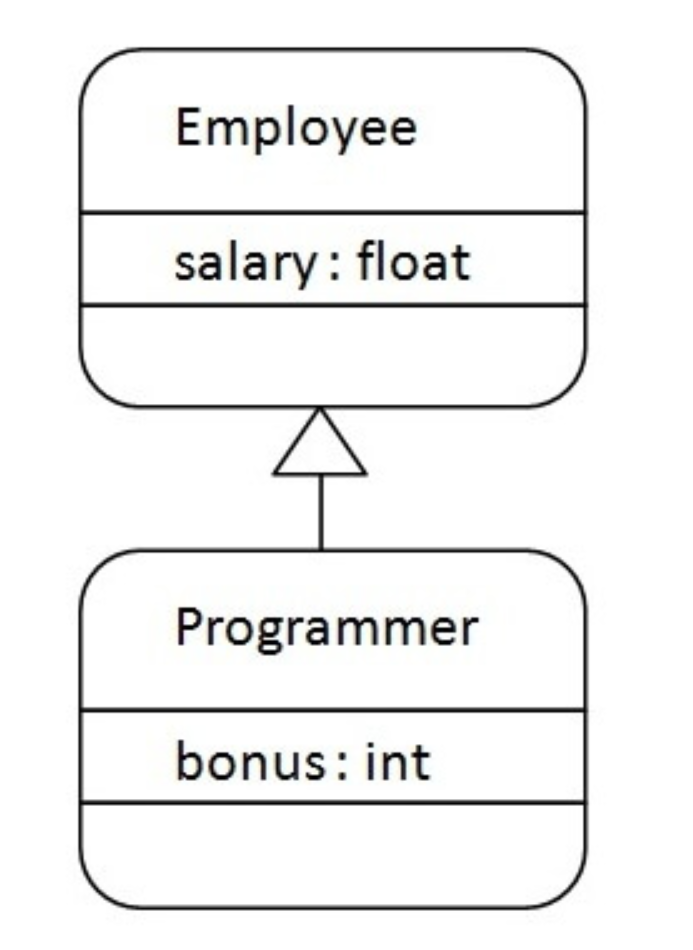
//methods and fields

}

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.

Java Inheritance Example



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 is:"+p.bonus);

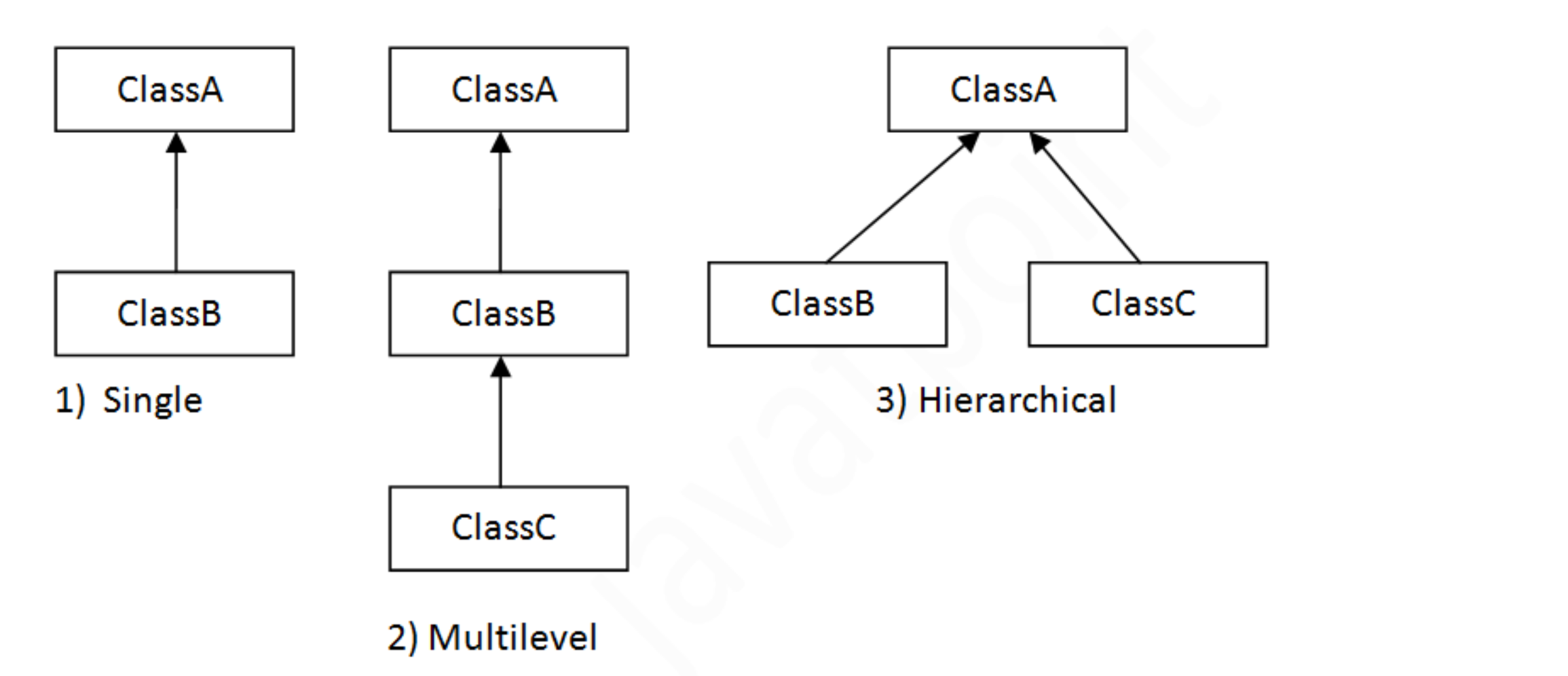
}

}

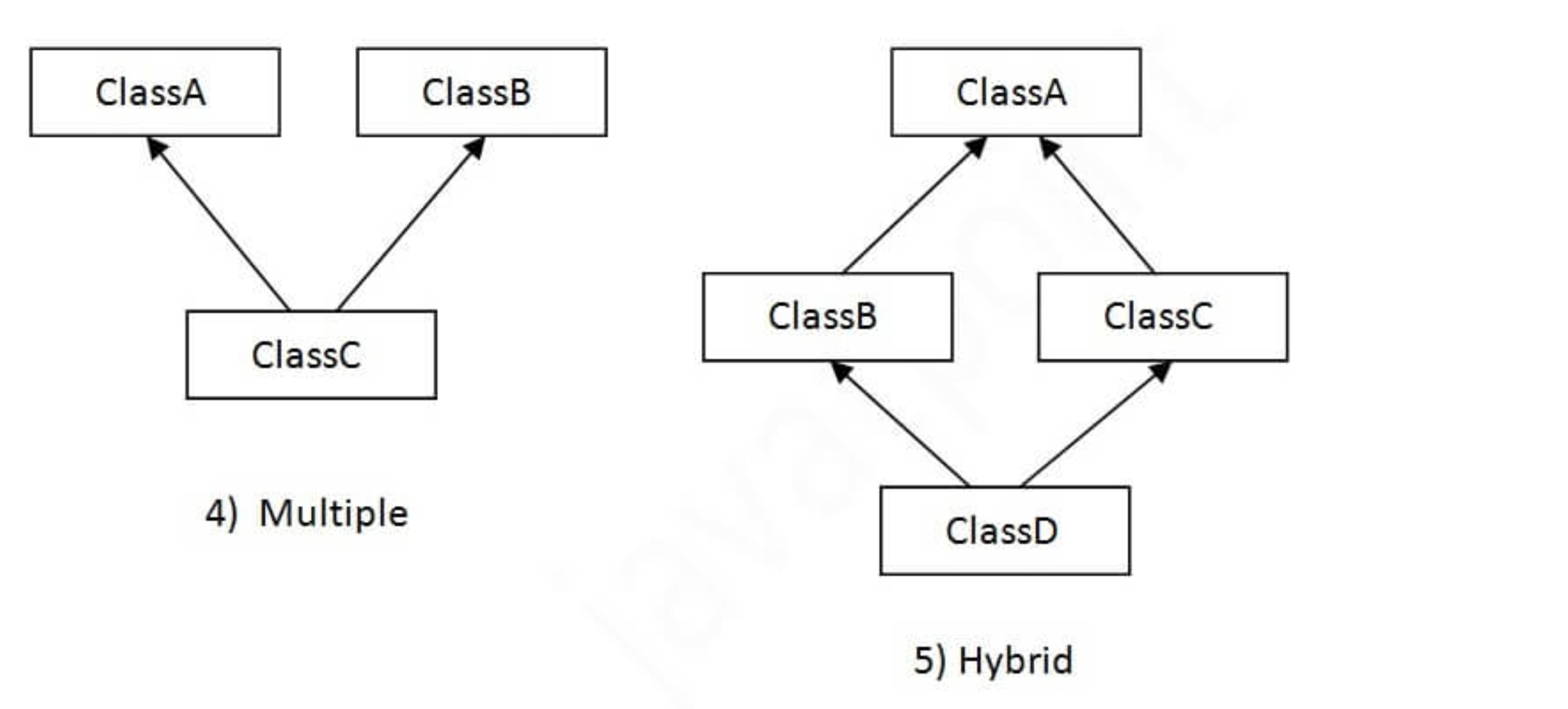
**Types of inheritance in java**

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. We will learn about interfaces later.



When one class inherits multiple classes, it is known as multiple inheritance. For Example:



**Single Inheritance Example**

When a class inherits another class, it is known as a single inheritance. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

File: TestInheritance.java

class Animal{

void eat(){System.out.println("eating...");}

}

class Dog extends Animal{

void bark(){System.out.println("barking...");}

}

class TestInheritance{

public static void main(String args[]){

Dog d=new Dog();

d.bark();

d.eat();

}}

**Multilevel Inheritance Example**

When there is a chain of inheritance, it is known as multilevel inheritance. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

File: TestInheritance2.java

class Animal{

void eat(){System.out.println("eating...");}

}

class Dog extends Animal{

void bark(){System.out.println("barking...");}

}

class BabyDog extends Dog{

void weep(){System.out.println("weeping...");}

}

class TestInheritance2{

public static void main(String args[]){

BabyDog d=new BabyDog();

d.weep();

d.bark();

d.eat();

}}

**Hierarchical Inheritance Example**

When two or more classes inherits a single class, it is known as hierarchical inheritance. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

File: TestInheritance3.java

class Animal{

void eat(){System.out.println("eating...");}

}

class Dog extends Animal{

void bark(){System.out.println("barking...");}

}

class Cat extends Animal{

void meow(){System.out.println("meowing...");}

}

class TestInheritance3{

public static void main(String args[]){

Cat c=new Cat();

c.meow();

c.eat();

//c.bark();//C.T.Error

}}

**Why multiple inheritance is not supported in java?**

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

class A{

void msg(){System.out.println("Hello");}

}

class B{

void msg(){System.out.println("Welcome");}

}

class C extends A,B{//suppose if it were

public static void main(String args[]){

C obj=new C();

obj.msg();//Now which msg() method would be invoked?

}

}