ADD ZERO GROUP
UNCOMMON PEOPLE

OOPs (Object-Oriented Programming)

Object means a real-world entity such as a pen, chair, table, computer, watch, etc.
Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

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

Object

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.

An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory.

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

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.

Advantage of OOPs over Procedure-oriented programming language

1) OOPs makes development and maintenance easier, whereas, in a procedure-oriented programming language, it is not easy to manage if code grows as project size increases.

2) OOPs provides data hiding, whereas, in a procedure-oriented programming language, global data can be accessed from anywhere.

3) OOPs provides the ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

Constructors

In Java, a constructor is a block of codes similar to the method. It is called when a 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.

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 three 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 synchronised

Types of Java constructors

Default Constructor

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

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

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 values to objects.

Copy Constructor

In Java, However, we can copy the values from one object to another like a copy constructor in C++.

```java
public class Prog9 {
    public static void main(String[] args) {
        // creating object using default constructor
        Student s1 = new Student();
            //  creating  object  using  parameterised
   constructor
        Student s2 = new Student(1, "George");
       // creating object using copy constructor
        Student s3 = new Student(s2);
    }
}
class Student{
    int rNo;
    String name;

    // default constructor
    public Student(){}

    //parameterised constructor
    public Student(int rNo, String name) {
        this.rNo = rNo;
        this.name = name;
    }
```

```
    // copy constructor
    public Student(Student s){
        this.rNo = s.rNo;
        this.name = s.name;
    }
}
```

this keyword in Java

In Java, this is a **reference variable** that refers to the current object. this can be used to refer to the current class instance variable.

```
public Student(int rNo, String name) {
    this.rNo = rNo;
    this.name = name;
}
```

Access Modifiers in Java

```
class Student {
    int rNo; // default
    private String name; // private
    protected int age; // protected
    public String contact; // public
}
```

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Getter and Setter Method in Java

```
class Student {
    private int rNo;
    private String name;

    public int getrNo() {
        return rNo;
    }

    public void setrNo(int rNo) {
        this.rNo = rNo;
    }
}
```

static keyword

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

```
public class AddZeroGroup {

    public static void main(String[] args) {


   System.out.println(SavingAccount.interestRate);

    }

}
```

```
class SavingAccount{
    int accNo;
    String custName;

    // static variable
    static int interestRate;
    // static block to initialise static variable
    static {
        interestRate = 4;
    }
}
```

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block

1) 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.

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

2) 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.
- The static method can not use non static data member or call non-static method directly.

Q) Why is the Java main method static?

Ans) It is because the object is not required to call a static method. If it were a non-static method, JVM creates an object first then call main() method that will lead the problem of extra memory allocation.

3) static block

- Is used to initialize the static data member.
- It is executed before the main method at the time of classloading.

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

System.out.println(SavingAccount.interestRate);
    }
}
```

```
class SavingAccount{
    int accNo;
    String custName;

    // static variable
    static int interestRate;

    // static block to initialise static variable
    static {
        interestRate = 4;
    }

    // static method to print interest rate
    public static void printInterestRate(){
        System.out.println(interestRate);
    }
}
```

Encapsulation

Encapsulation in Java is a *process of wrapping code and data together into a single unit*, for example, a capsule which is mixed of several medicines.

We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

It is a way to achieve **data hiding** in Java because other class will not be able to access the data through the private data members.

Inheritance

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

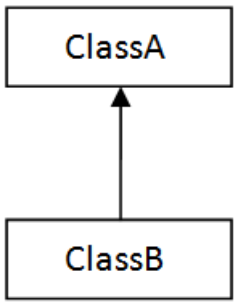
- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability

Terms used in Inheritance

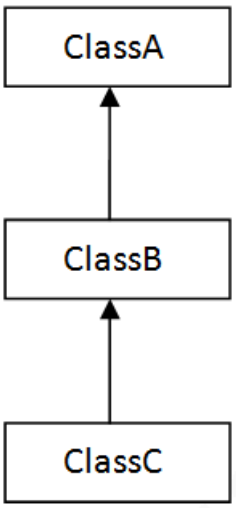
- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent 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 **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.

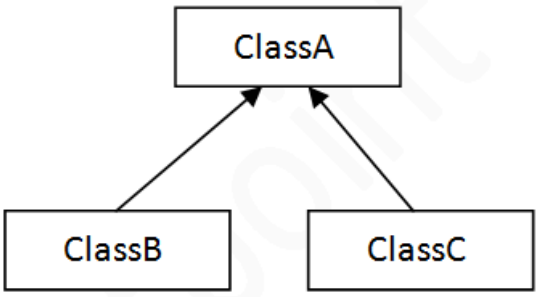
Types of inheritance in java



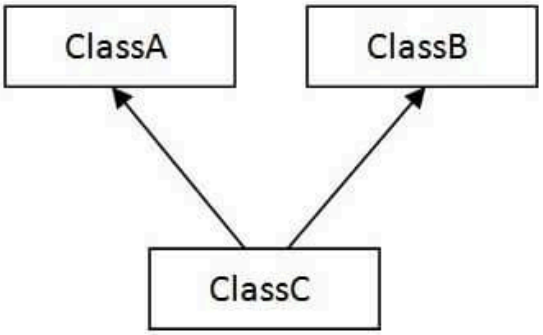
1) Single



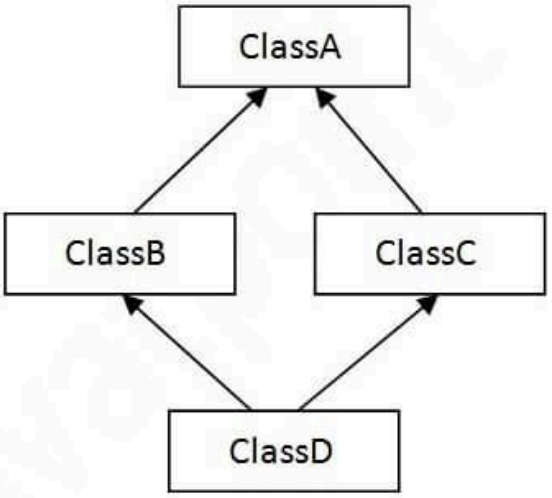
2) Multilevel



3) Hierarchical



4) Multiple



5) Hybrid

Single Inheritance

When a class inherits another class, it is known as a *single inheritance*.

```
class Person{
    String name;
    int age;
}
class Student extends Person{
    int rNo;
    int sem;
}
```

Multilevel Inheritance

When there is a chain of inheritance, it is known as *multilevel inheritance*.

```
class Person{
    String name;
    int age;

}

class Employee{
    int eId;
    int salary;
}

class Professor extends Employee{
    String[] subjects;
}
```

Hierarchical Inheritance

When two or more classes inherits a single class, it is known as *hierarchical inheritance*

```
class Person{
    String name;
    int age;

}
class Student extends Person{
    int rNo;
    int sem;
}

class Professor extends Person{
    String[] subjects;
}
```

Hybrid Inheritance

Two or more inheritance at the same time.

```
class Person{
    String name;
    int age;

}

class Employee extends Person{
    int eId;
    int salary;
}
```

```
class Professor extends Employee{
    String[] subjects;
}

class Student extends Person{
    int rNo;
    int sem;
}
```

Q) 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 a 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.

Aggregation

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

```
class Address{
    int hNo;
    String soc;
    String city;

    public Address(int hNo, String soc, String city) {
        this.hNo = hNo;
        this.soc = soc;
```

```
        this.city = city;
    }
}

class Student{
    int rNo;
    int sem;
    Address address;

     public Student(int rNo, int sem, int hNo, String
   soc, String city) {
        this.rNo = rNo;
        this.sem = sem;
        this.address = new Address(hNo, soc, city);
    }
}
```

Super Keyword

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

```
class Person{
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

```
class Student extends Person{
    int rNo;
    int sem;

     public Student(int rNo, int sem, String name, int
   age) {
         super(name, age);
         this.rNo = rNo;
         this.sem = sem;
     }
}
```

Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

Polymorphism

1. Method Overloading (compile time)

If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.

There are two ways to overload the method in java

1. By changing number of arguments

2. By changing the data type

```
public static void sum(int a, int b){
    System.out.println(a+b);
}
// by changing No. of arguments
public static void sum(int a, int b, int c){
    System.out.println(a+b);
}
// by changing types of arguments
public static void sum(double a, double b){
    System.out.println(a+b);
}

//Not possible by changing return type of method
public static double sum(double a, double b){
    return a+b;
}
```

Note: Method Overloading is not possible by changing the return type of method

2. Method Overriding (Run time)

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Rules for Java Method Overriding

1. The method must have the same name as in the parent class

2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).

```
class Bank{
    int accNo;
    double balance;

    public void printInterest(){
        System.out.println(4);
    }
}

class SBI extends Bank{
    @Override
    public void printInterest(){
        System.out.println(5);
    }
}
```

Final Keyword

The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

1) final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

```
final int noOfDaysInWeek = 7;
```

2) Java final method

If you make any method as final, you cannot override it.

```
public final void printGoodMorning(){
    System.out.println("Good morning");
}
```

3) Java final class

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

```
final class Bank{
    int accNo;
    double balance;
}
```

Q) Is final method inherited?

Ans) Yes, final method is inherited but you cannot override it

Abstraction

Abstraction is a process of hiding the implementation details and showing only functionality to the user.
Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery. Abstraction lets you focus on what the object does instead of how it does it.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

Abstract class

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 and static methods also.

- It can have final methods which will force the subclass not to change the body of the method.

Abstract Method in Java

A method which is declared as abstract and does not have implementation is known as an abstract method.

```
abstract class Person{
    private String name;
    private int age;

    // abstract method
    abstract public void printSomething();

    // normal method
    public void printGoodMorning(){
        System.out.println("good morning");
    }
}
```

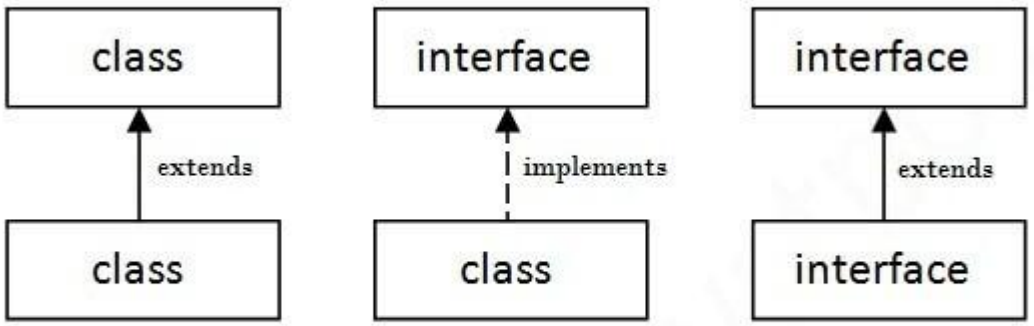
Interface

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods. The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

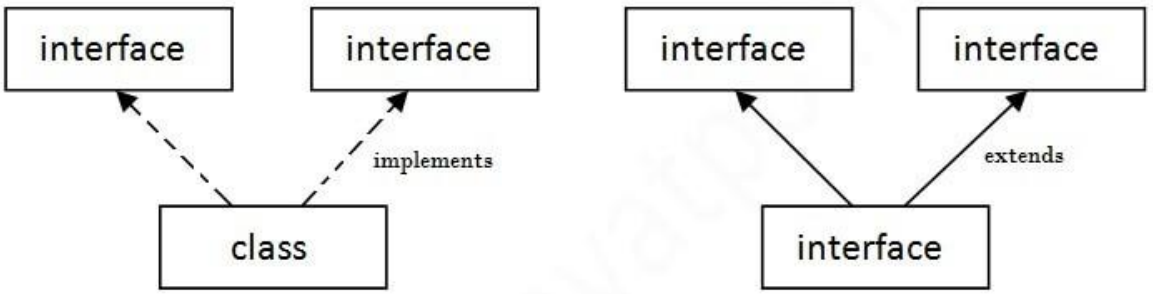
In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body. Java Interface also represents the IS-A relationship. It cannot be instantiated just like the abstract class.

Since Java 8, we can have **default and static methods** in an interface.

The relationship between classes and interfaces



Multiple inheritance in Java by interface



Multiple Inheritance in Java

Q) Multiple inheritance is not supported through class in java, but it is possible by an interface, why?

Multiple inheritance is not supported in the case of class because of ambiguity. However, it

is supported in case of an interface because there is no ambiguity. It is because its implementation is provided by the implementation class.

Default Method in Interface

Since Java 8, we can have method body in interface. But we need to make it default method.

Static Method in Interface

Since Java 8, we can have static method in interface. Let's see an example:

```
interface Bank{
    static final int interestRate = 4;
    public void calculateInterestRate();

    // static method - since java 8
    static void printSomething(){
        System.out.println("something");
    }

    // default method - since java 8
    default void printAnything(){
        System.out.println("anything");
    }
}
```

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();
	}

Enums

The **Enum in Java** is a data type which contains a fixed set of constants.

It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY) , directions (NORTH, SOUTH, EAST, and WEST), season (SPRING, SUMMER, WINTER, and AUTUMN or FALL), colors (RED, YELLOW, BLUE, GREEN, WHITE, and BLACK) etc. According to the Java naming conventions, we should have all constants in capital letters. So, we have enum constants in capital letters.

```
enum MonthEnum{

    JANUARY, FEBRUARY, MARCH, APRIL }
```

ADD ZERO GROUP

UNCOMMON PEOPLE