Oops

"Oops" is a common acronym in programming that stands for "Object-Oriented Programming." Object-oriented programming is a programming paradigm that revolves around the concept of objects, which can contain data (in the form of fields or attributes) and code (in the form of procedures or methods). It emphasizes the organization of code into reusable and modular components.

In Java, object-oriented programming is fundamental to the language's design. Here's why OOP is important in Java:

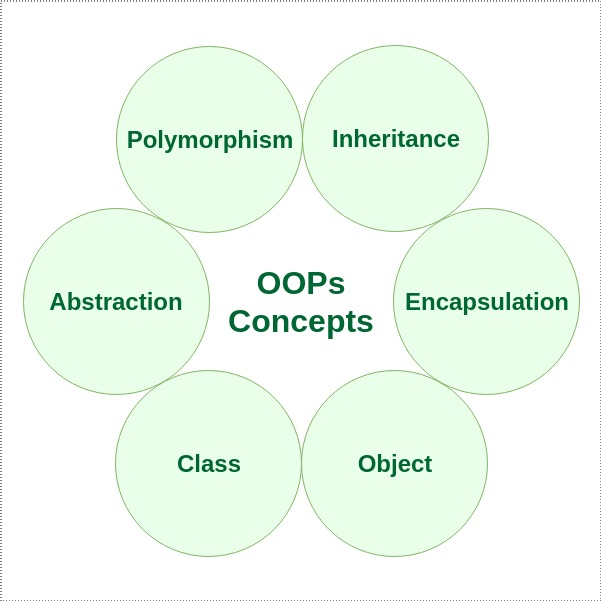
1. **Encapsulation**: Java supports encapsulation, which means the bundling of data (attributes) and methods (functions) that operate on the data into a single unit, known as a class. This helps in hiding the internal state of objects from the outside world and only exposing the necessary functionalities, thus enhancing security and maintainability of code.
2. **Inheritance**: Java allows classes to inherit attributes and methods from other classes. This promotes code reusability and allows for the creation of a hierarchy of classes, where common attributes and behaviors can be defined in a superclass and specialized behavior can be added in subclasses.
3. **Polymorphism**: Java supports polymorphism, which allows objects of different classes to be treated as objects of a common superclass. This feature enables flexibility in code design, as it allows methods to be defined in terms of their superclass and overridden in subclasses to provide specific implementations.
4. **Abstraction**: Abstraction allows programmers to represent real-world entities as classes with attributes and methods that model their essential features and behaviors, while hiding the unnecessary details. This simplifies complex systems by focusing on relevant aspects and ignoring irrelevant ones.
5. **Modularity**: Object-oriented programming encourages the modular design of software, where complex systems are broken down into smaller, manageable units (objects/classes). This promotes code organization, maintenance, and collaboration among developers.

Using Object-Oriented Programming (OOP) in Java offers several benefits that contribute to better software design, development, and maintenance:

* Modularity: OOP allows you to break down your software into smaller, self-contained modules (objects or classes), which makes it easier to understand and manage your codebase. Each module can encapsulate data and behavior, providing clear boundaries and reducing complexity.
* Code Reusability: With OOP, you can create reusable components (classes) that can be easily reused in different parts of your program or even in other projects. This saves development time and effort and promotes consistency across your codebase
* Encapsulation: OOP enables you to encapsulate the internal state of an object and hide its implementation details from the outside world. This protects the integrity of the data and allows you to change the internal implementation without affecting the external interfaces, enhancing code security and maintainability.
* Inheritance: Java supports inheritance, which allows you to create new classes (subclasses) based on existing ones (superclasses). This promotes code reuse by inheriting common attributes and behaviors from a superclass and extending or modifying them in subclasses. It also facilitates the creation of class hierarchies, enabling better organization and abstraction of concepts.
* Polymorphism: Polymorphism allows objects of different types to be treated uniformly through a common interface. In Java, polymorphism is achieved through method overriding and method overloading. This flexibility enables you to write code that can work with objects of different types, making your programs more adaptable and extensible.
* Abstraction: OOP supports abstraction, allowing you to model real-world entities as abstract classes or interfaces that define a set of common behaviors without specifying their concrete implementations. This simplifies complex systems by focusing on essential features and hiding unnecessary details, improving code clarity and maintainability.
* Ease of Maintenance: OOP promotes a modular and organized code structure, which makes it easier to maintain and extend your software over time. Changes or updates to one part of the codebase are less likely to have unintended consequences on other parts, leading to fewer bugs and faster development cycles.

OOPs

* Object-Oriented Programming or Java OOPs concept refers to languages that use objects in programming
* The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.



Class

* A class in Java is a set of objects which shares common characteristics/ behavior and common properties/ attributes.
* It is a user-defined blueprint or prototype from which objects are created
* For example, Student is a class while a particular student named Sonam is an object.

Properties of Java Classes

* 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 a group of methods.
* A Class in Java can contain:
  + Data member
  + Method
  + Constructor
  + Nested Class
  + Interface

Example- access\_modifier class <class\_name>

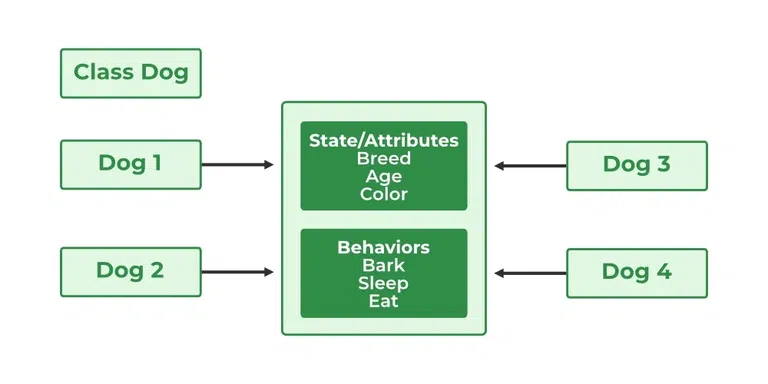
Java Objects

* An object in Java is a basic unit of Object-Oriented Programming and represents real-life entities
* Objects are the instances of a class that are created to use the attributes and methods of a class
* A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :
  + State: It is represented by attributes of an object. It also reflects the properties of an object.
  + Behavior: It is represented by the methods of an object. It also reflects the response of an object with other objects.
  + Identity: It gives a unique name to an object and enables one object to interact with other objects.



**Declaring Objects (Also called instantiating a class)**

* When an object of a class is created, the class is said to be instantiated.
* All the instances share the attributes and the behavior of the class.
* But the values of those attributes, i.e. the state are unique for each object.
* A single class may have any number of instances.
* For reference variables , the type must be strictly a concrete class name.



**Initializing a Java object**

* The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory
* The new operator also invokes the class constructor.

**Ways to Create an Object of a Class**

There are four ways to create objects in Java. Strictly speaking, there is only one way(by using a new keyword), and the rest internally use a new keyword.

1 . Using new keyword

creating object of class Test

Test t = new Test();

2. Using Class.forName(String className) method

* There is a pre-defined class in java.lang package with name Class.
* The forName(String className) method returns the Class object associated with the class with the given string name.
* We have to give a fully qualified name for a class. On calling the new Instance() method on this Class object returns a new instance of the class with the given string name.

// creating object of public class Test

// consider class Test present in com.p1 package

Test obj = (Test)Class.forName("com.p1.Test").newInstance();

3. Using clone() method

clone() method is present in the Object class.

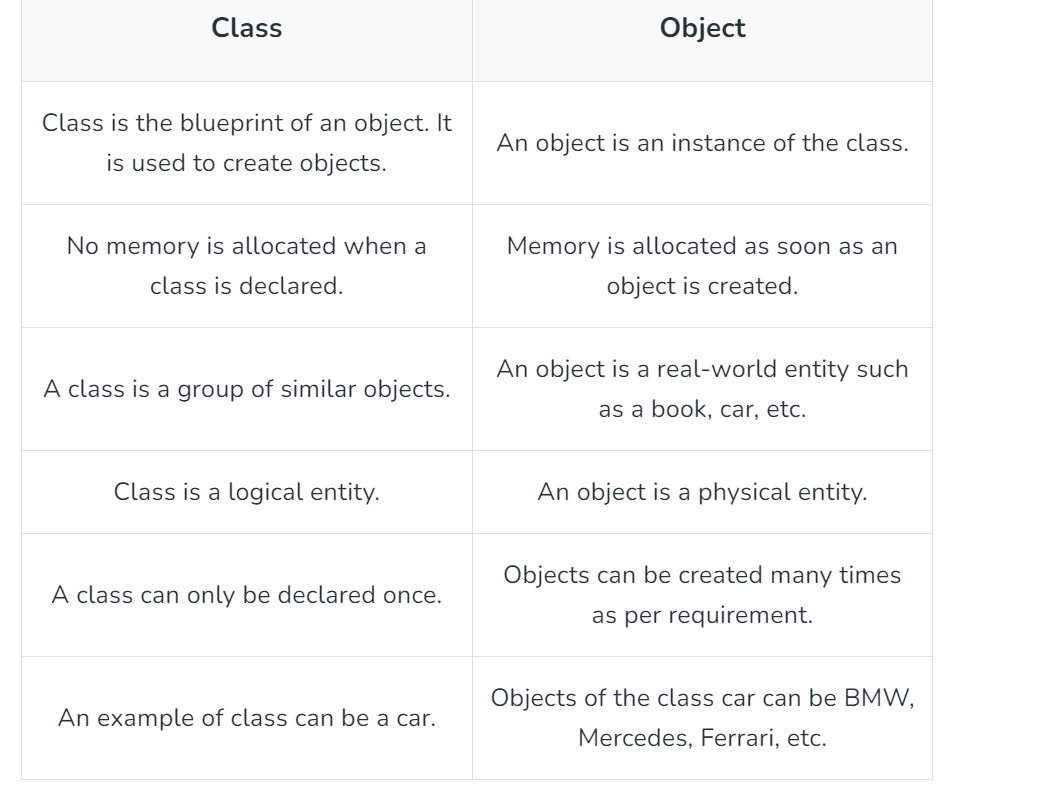
It creates and returns a copy of the object.

4. Deserialization

**Anonymous Objects in Java**

* Anonymous objects are objects that are instantiated but are not stored in a reference variable.
* They are used for immediate method calls.
* They will be destroyed after method calling.
* They are widely used in different libraries.

new EventHandler()



Why Java is not a purely Object-Oriented Language?

* Pure Object Oriented Language or Complete Object Oriented Language are Fully Object Oriented Language that supports or have features that treats everything inside the program as objects.
* It doesn’t support primitive datatype(like int, char, float, bool, etc.).
* There are seven qualities to be satisfied for a programming language to be pure object-oriented. They are:
* Encapsulation/Data Hiding
* Inheritance
* Polymorphism
* Abstraction
* **All predefined types are objects**
* All user defined types are objects
* **All operations performed on objects must be only through methods exposed at the objects**

Explanation:

1.**Primitive Data Type ex. int, long, bool, float, char, etc as Objects:**

Smalltalk is a “pure” object-oriented programming language unlike Java and C++ as there is no difference between values that are objects and values that are primitive types. In Smalltalk, primitive values such as integers, booleans, and characters are also objects. In Java, we have predefined types as non-objects (primitive types).

**2.The static keyword:** When we declare a class as static, then it can be used without the use of an object in Java. If we are using static function or static variable then we can’t call that function or variable by using dot(.) or class object defying object-oriented feature.

**3. Wrapper Class**: Wrapper class provides the mechanism to convert primitive into object and object into primitive. In Java, you can use Integer, Float, etc. instead of int, float etc. We can communicate with objects without calling their methods. ex. using arithmetic operators.

Even using Wrapper classes does not make Java a pure OOP language, as internally it will use the operations like Unboxing and Autoboxing. So if you create Integer instead of int and do any mathematical operation on it, under the hoods Java is going to use primitive type int only.

Java Constructors

* Java constructors or constructors in Java is a terminology used to construct something in our programs.
* A constructor in Java is a special method that is used to initialize objects.
* The constructor is called when an object of a class is created.
* It can be used to set initial values for object attributes.

# What are 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 the constructor, memory for the object is allocated in the memory. It is a special type of method that is used to initialize the object.
* Every time an object is created using the new() keyword, at least one constructor is called.

Note: It is not necessary to write a constructor for a class. It is because the java compiler creates a default constructor (constructor with no arguments) if your class doesn’t have any.

**How Java Constructors are Different From Java Methods?**

* Constructors must have the same name as the class within which it is defined it is not necessary for the method in Java.
* Constructors do not return any type while method(s) have the return type or void if does not return any value.
* Constructors are called only once at the time of Object creation while method(s) can be called any number of times.

**class Animal**

**{**

**.......**

**// A Constructor**

**Animal() {**

**}**

**.......**

**}**

**// We can create an object of the above class**

**// using the below statement. This statement**

**// calls above constructor.**

**Animal obj = new Animal();**

The first line of a constructor is a call to super() or this(), (a call to a constructor of a super-class or an overloaded constructor),

if you don’t type in the call to super in your constructor the compiler will provide you with a non-argument call to super at the first line of your code, the super constructor must be called to create an object:

If you think your class is not a subclass it actually is, every class in Java is the subclass of a class object even if you don’t say extends object in your class definition.

# Need of Constructors in Java

* Think of a Box.
* If we talk about a box class then it will have some class variables (say length, breadth, and height).
* But when it comes to creating its object(i.e Box will now exist in the computer’s memory),

then can a box be there with no value defined for its dimensions? The answer is No.

* So constructors are used to assign values to the class variables at the time of object creation, either explicitly done by the programmer or by Java itself (default constructor).

# When Java Constructor is called?

* Each time an object is created using a new() keyword, at least one constructor (it could be the default constructor) is invoked to assign initial values to the data members of the same class.

Rules for writing constructors are as follows:

* The constructor(s) of a class must have the same name as the class name in which it resides.
* A constructor in Java can not be **abstract, final, static, or Synchronized.**
* Access modifiers can be used in constructor declaration to control its access i.e which other class can call the constructor.
* So far, we have learned constructors are used to initialize the object’s state. Like methods, a constructor also contains a collection of statements(i.e. instructions) that are executed at the time of Object creation.

# Types of Constructors in Java

Now is the correct time to discuss the types of the constructor, so primarily there are three types of constructors in Java are mentioned below:

* Default Constructor
* Parameterized Constructor
* Copy Constructor

1. Default Constructor in Java

* A constructor that has no parameters is known as default the constructor.
* A default constructor is invisible.
* And if we write a constructor with no arguments, the compiler does not create a default constructor. It is taken out.
* It is being overloaded and called a parameterized constructor.
* The default constructor changed into the parameterized constructor.
* But Parameterized constructor can’t change the default constructor.
* The default constructor can be implicit or explic it.
* If we don’t define explicitly, we get an implicit default constructor.
* If we manually write a constructor, the implicit one is overridded.

**Note: Default constructor provides the default values to the object like 0, null, etc. depending on the type.**

# 2. Parameterized Constructor in Java

A constructor that has parameters is known as parameterized constructor. If we want to initialize fields of the class with our own values, then use a parameterized constructor.

**Remember: Does constructor return any value?**

**There are no “return value” statements in the constructor, but the constructor returns the current class instance. We can write ‘return’ inside a constructor.**

Now the most important topic that comes into play is the strong incorporation of OOPS with constructors known as constructor overloading. Just like methods, we can overload constructors for creating objects in different ways. The compiler differentiates constructors on the basis of the number of parameters, types of parameters, and order of the parameters.

**3. Copy Constructor in Java**

Unlike other constructors copy constructor is passed with another object which copies the data available from the passed object to the newly created object.

**Note**: In Java,there is no such inbuilt copy constructor available like in other programming languages such as C++, instead we can create our own copy constructor by passing the object of the same class to the other instance(object) of the class.

// Java Program for Copy Constructor

import java.io.\*;

class Person {

// data members of the class.

String name;

int id;

// Parameterized Constructor

Person(String name, int id)

{

this.name = name;

this.id = id;

}

// Copy Constructor

Person(Person obj2)

{

this.name = obj2.name;

this.id = obj2.id;

}

}

class Driver {

public static void main(String[] args)

{

// This would invoke the parameterized constructor.

System.out.println("First Object");

Person p1 = new Person("Avni", 68);

System.out.println("Name :" + p1.name

+ " and Id :" + p1.id);

System.out.println();

// This would invoke the copy constructor.

Person p2 = new Person(p1);

System.out.println(

"Copy Constructor used Second Object");

System.out.println("Name :" + p2.name

+ " and Id :" + p2.id);

}

}

1. What is a constructor in Java?

A constructor in Java is a special method used to initialize objects.

2. Can a Java constructor be private?

Yes, a constructor can be declared private. A private constructor is used in restricting object creation.

3.Constructor chaining???

**Inheritance in Java**

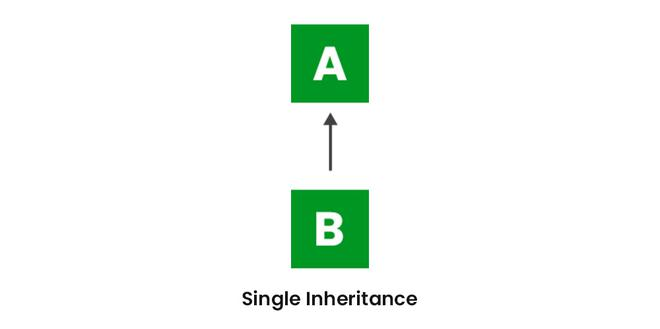
* Java, Inheritance is an important pillar of OOP(Object-Oriented Programming).
* It is the mechanism in Java by which one class is allowed to inherit the features(fields and methods) of another class.
* In Java, Inheritance means creating new classes based on existing ones.
* A class that inherits from another class can reuse the methods and fields of that class.
* In addition, you can add new fields and methods to your current class as well.

**How to Use Inheritance in Java?**

* The extends keyword is used for inheritance in Java. Using the extends keyword indicates you are derived from an existing class. In other words, “extends” refers to increased functionality.
* Java Inheritance Types
* Below are the different types of inheritance which are supported by Java.
  1. Single Inheritance
  2. Multilevel Inheritance
  3. Hierarchical Inheritance
  4. Multiple Inheritance
  5. Hybrid Inheritance

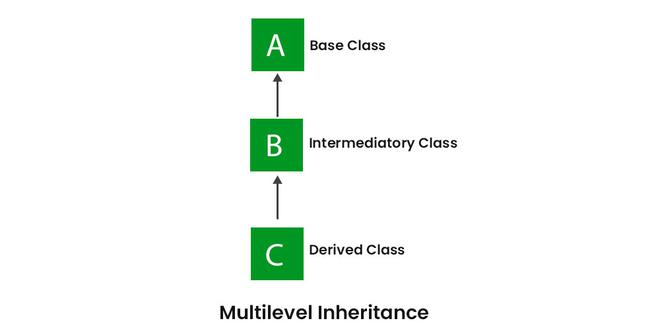
**1. Single Inheritance**

* In single inheritance, a sub-class is derived from only one super class.
* It inherits the properties and behavior of a single-parent class.
* Sometimes, it is also known as simple inheritance. In the below figure, ‘A’ is a parent class and ‘B’ is a child class.
* The class ‘B’ inherits all the properties of the class ‘A’.



**2. Multilevel Inheritance**

* In Multilevel Inheritance, a derived class will be inheriting a base class, and as well as the derived class also acts as the base class for other classes.
* In the below image, class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C.
* In Java, a class cannot directly access the grandparent’s members.

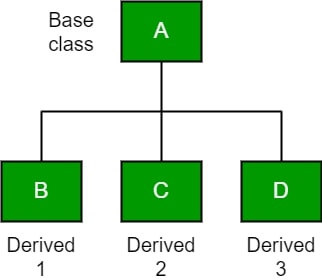
****

**3. Hierarchical Inheritance**

In Hierarchical Inheritance, one class serves as a superclass (base class) for more than

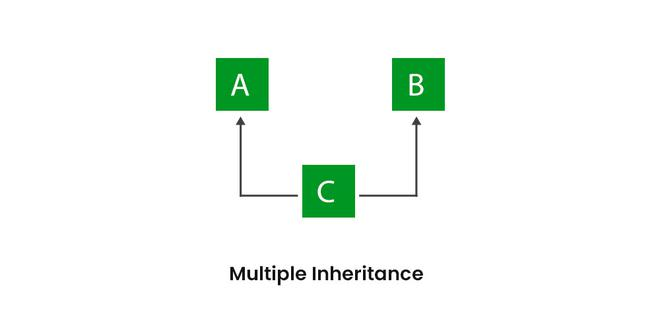
one subclass. In the below image, class A serves as a base class for the derived classes

B, C, and D.



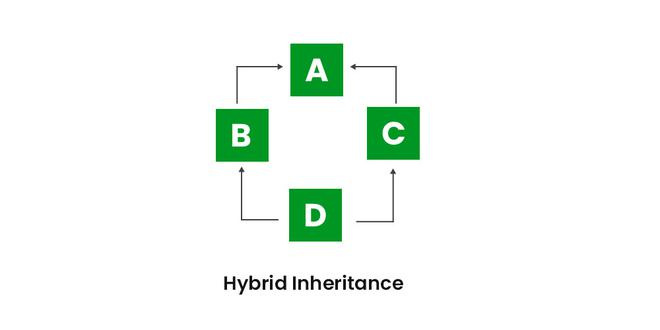
**4. Multiple Inheritance (Through Interfaces)**

* In Multiple inheritances, one class can have more than one superclass and inherit features from all parent classes.
* Please note that Java does not support multiple inheritances with classes.
* In Java, we can achieve multiple inheritances only through Interfaces.



5. Hybrid Inheritance

* It is a mix of two or more of the above types of inheritance.
* Since Java doesn’t support multiple inheritances with classes, hybrid inheritance involving multiple inheritance is also not possible with classes.
* In Java, we can achieve hybrid inheritance only through Interfaces if we want to involve multiple inheritance to implement Hybrid inheritance.
* However, it is important to note that Hybrid inheritance does not necessarily require the use of Multiple Inheritance exclusively.
* It can be achieved through a combination of Multilevel Inheritance and Hierarchical Inheritance with classes, Hierarchical and Single Inheritance with classes.
* Therefore, it is indeed possible to implement Hybrid inheritance using classes alone, without relying on multiple inheritance type.



## **Java IS-A type of Relationship**

IS-A is a way of saying: This object is a type of that object. Let us see how the extends keyword is used to achieve inheritance.

**What Can Be Done in a Subclass?**

In sub-classes we can inherit members as is, replace them, hide them, or supplement them with new members:

* The inherited fields can be used directly, just like any other fields.
* We can declare new fields in the subclass that are not in the superclass.
* The inherited methods can be used directly as they are.
* We can write a new instance method in the subclass that has the same signature as the one in the superclass, thus overriding it (as in the example above, toString() method is overridden).
* We can write a new static method in the subclass that has the same signature as the one in the superclass, thus hiding it.
* We can declare new methods in the subclass that are not in the superclass.
* We can write a subclass constructor that invokes the constructor of the superclass, either implicitly or by using the keyword super.

**Advantages Of Inheritance in Java:**

* **Code Reusability**: Inheritance allows for code reuse and reduces the amount of code that needs to be written. The subclass can reuse the properties and methods of the superclass, reducing duplication of code.
* **Abstraction:** Inheritance allows for the creation of abstract classes that define a common interface for a group of related classes. This promotes abstraction and encapsulation, making the code easier to maintain and extend.
* **Class Hierarchy:** Inheritance allows for the creation of a class hierarchy, which can be used to model real-world objects and their relationships.
* **Polymorphism:** Inheritance allows for polymorphism, which is the ability of an object to take on multiple forms. Subclasses can override the methods of the superclass, which allows them to change their behaviour in different ways.

**Conclusion**

Let us check some important points from the article are mentioned below:

* **Default superclass**: Except Object class, which has no superclass, every class has one and only one direct superclass (single inheritance). In the absence of any other explicit superclass, every class is implicitly a subclass of the Object class.
* **Superclass can only be one:** A superclass can have any number of subclasses. But a subclass can have only one superclass. This is because Java does not support multiple inheritances with classes. Although with interfaces, multiple inheritances are supported by Java.
* **Inheriting Constructors:** A subclass inherits all the members (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.
* **Private member inheritance:** A subclass does not inherit the private members of its parent class. However, if the superclass has public or protected methods(like getters and setters) for accessing its private fields, these can also be used by the subclass.

Abstraction in Java

* Abstraction in Java is the process in which we only show essential details/functionality to the user. The non-essential implementation details are not displayed to the user.
* In this article, we will learn about abstraction and what abstract means.

Simple Example to understand Abstraction:

* Television remote control is an excellent example of abstraction. It simplifies the interaction with a TV by hiding the complexity behind simple buttons and symbols, making it easy without needing to understand the technical details of how the TV functions.

# What is Abstraction in Java?

In Java, abstraction is achieved by interfaces and abstract classes. We can achieve 100% abstraction using interfaces.

Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details. The properties and behaviours of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.

Abstraction Real-Life Example:

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of a car or applying brakes will stop the car, but he does not know how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of the accelerator, brakes, etc in the car. This is what abstraction is.

# Java Abstract classes and Java Abstract methods

* An abstract class is a class that is declared with an abstract keyword.
* An abstract method is a method that is declared without implementation.
* An abstract class may or may not have all abstract methods. Some of them can be concrete methods
* A method-defined abstract must always be redefined in the subclass, thus making overriding compulsory or making the subclass itself abstract.
* Any class that contains one or more abstract methods must also be declared with an abstract keyword.
* There can be no object of an abstract class. That is, an abstract class can not be directly instantiated with the new operator.
* An abstract class can have parameterized constructors and the default constructor is always present in an abstract class.

# Algorithm to implement abstraction in Java

Determine the classes or interfaces that will be part of the abstraction.

Create an abstract class or interface that defines the common behaviours and properties of these classes.

Define abstract methods within the abstract class or interface that do not have any implementation details.

Implement concrete classes that extend the abstract class or implement the interface.

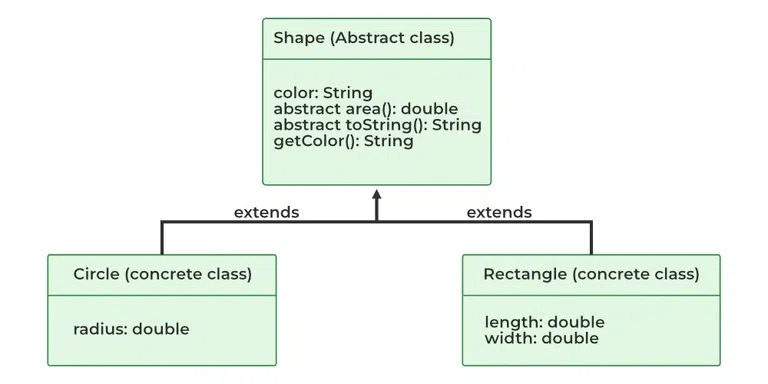
Override the abstract methods in the concrete classes to provide their specific implementations.

Use the concrete classes to implement the program logic.

# When to use abstract classes and abstract methods?

There are situations in which we will want to define a superclass that declares the structure of a given abstraction without providing a complete implementation of every method. Sometimes we will want to create a superclass that only defines a generalization form that will be shared by all of its subclasses, leaving it to each subclass to fill in the details.

Consider a classic “shape” example, perhaps used in a computer-aided design system or game simulation. The base type is “shape” and each shape has a color, size, and so on. From this, specific types of shapes are derived(inherited)-circle, square, triangle, and so on — each of which may have additional characteristics and behaviours. For example, certain shapes can be flipped. Some behaviours may be different, such as when you want to calculate the area of a shape. The type hierarchy embodies both the similarities and differences between the shapes.



// Java Program to implement

// Java Abstraction

// Abstract Class declared

abstract class Animal {

private String name;

public Animal(String name) { this.name = name; }

public abstract void makeSound();

public String getName() { return name; }

}

// Abstracted class

class Dog extends Animal {

public Dog(String name) { super(name); }

public void makeSound()

{

System.out.println(getName() + " barks");

}

}

// Abstracted class

class Cat extends Animal {

public Cat(String name) { super(name); }

public void makeSound()

{

System.out.println(getName() + " meows");

}

}

// Driver Class

public class AbstractionExample {

// Main Function

public static void main(String[] args)

{

Animal myDog = new Dog("Buddy");

Animal myCat = new Cat("Fluffy");

myDog.makeSound();

myCat.makeSound();

}

}

Interface

* Interfaces are another method of implementing abstraction in Java.
* The key difference is that, by using interfaces, we can achieve 100% abstraction
* in Java classes.
* In Java or any other language, interfaces include both methods and variables
* but lack a method body. Apart from abstraction, interfaces can also be used to
* implement interfaces in Java.
* Implementation: To implement an interface we use the keyword “implements” with class.

// Define an interface named Shape

interface Shape {

double calculateArea(); // Abstract method for

// calculating the area

}

// Implement the interface in a class named Circle

class Circle implements Shape {

private double radius;

// Constructor for Circle

public Circle(double radius) { this.radius = radius; }

// Implementing the abstract method from the Shape

// interface

public double calculateArea()

{

return Math.PI \* radius \* radius;

}

}

// Implement the interface in a class named Rectangle

class Rectangle implements Shape {

private double length;

private double width;

// Constructor for Rectangle

public Rectangle(double length, double width)

{

this.length = length;

this.width = width;

}

// Implementing the abstract method from the Shape

// interface

public double calculateArea() { return length \* width; }

}

// Main class to test the program

public class Main {

public static void main(String[] args)

{

// Creating instances of Circle and Rectangle

Circle myCircle = new Circle(5.0);

Rectangle myRectangle = new Rectangle(4.0, 6.0);

// Calculating and printing the areas

System.out.println("Area of Circle: "

+ myCircle.calculateArea());

System.out.println("Area of Rectangle: "

+ myRectangle.calculateArea());

}

}

**Advantages of Abstraction**

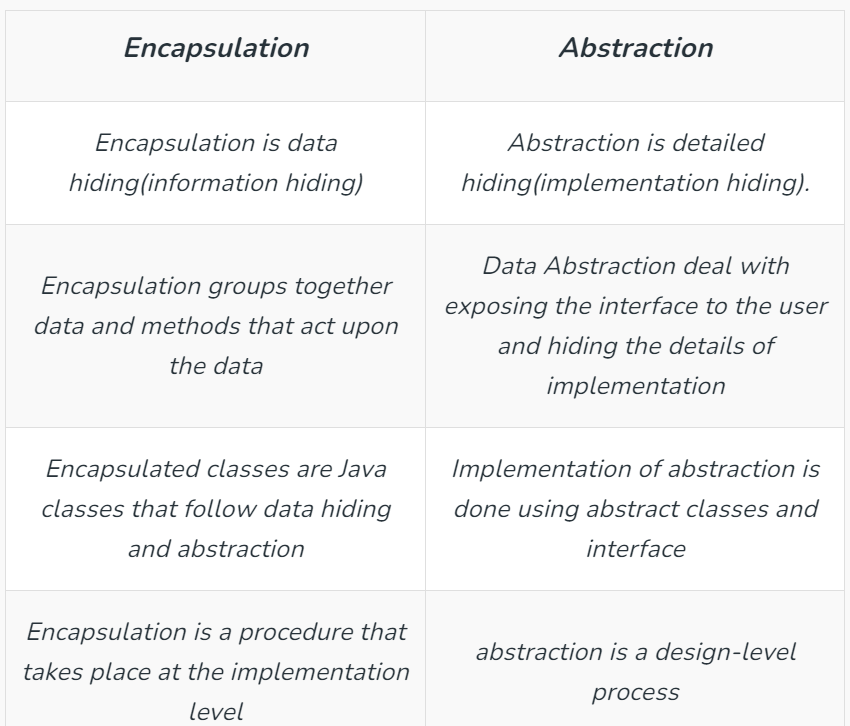
Here are some advantages of abstraction:

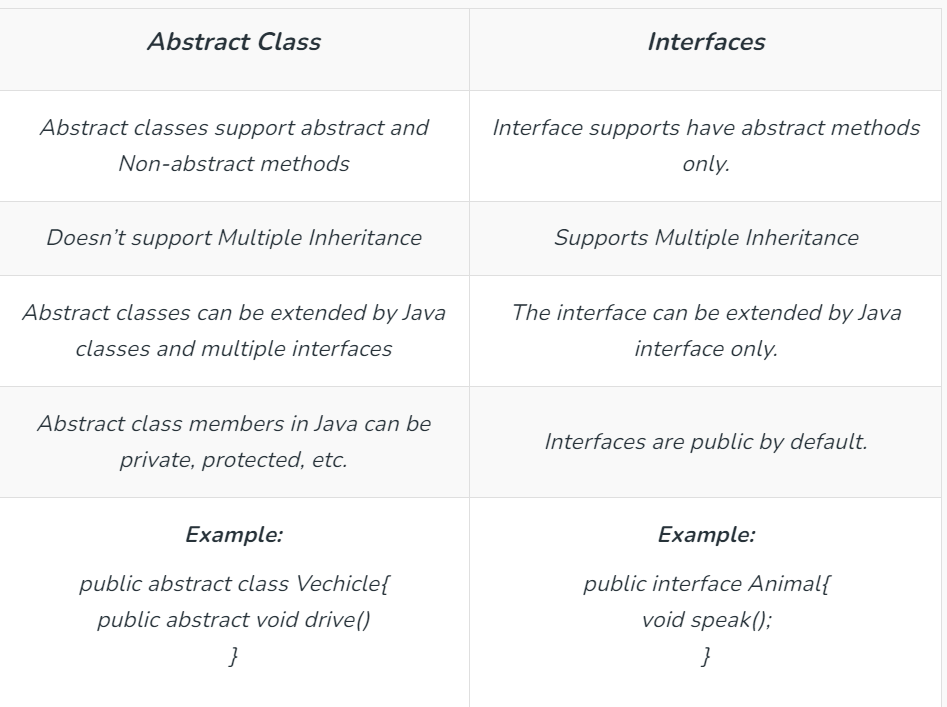
* It reduces the complexity of viewing things.
* Avoids code duplication and increases reusability.
* Helps to increase the security of an application or program as only essential details are provided to the user.
* It improves the maintainability of the application.
* It improves the modularity of the application.
* The enhancement will become very easy because without affecting end-users we can able to perform any type of changes in our internal system.
* Improves code reusability and maintainability.
* Hides implementation details and exposes only relevant information.
* Provides a clear and simple interface to the user.
* Increases security by preventing access to internal class details.
* Supports modularity, as complex systems can be divided into smaller and more manageable parts.
* Abstraction provides a way to hide the complexity of implementation details from the user, making it easier to understand and use.
* Abstraction allows for flexibility in the implementation of a program, as changes to the underlying implementation details can be made without affecting the user-facing interface.
* Abstraction enables modularity and separation of concerns, making code more maintainable and easier to debug.

**Disadvantages of Abstraction in Java**

Here are the main disadvantages of abstraction in Java:

* Abstraction can make it more difficult to understand how the system works.
* It can lead to increased complexity, especially if not used properly.
* This may limit the flexibility of the implementation.
* Abstraction can add unnecessary complexity to code if not used appropriately, leading to increased development time and effort.
* Abstraction can make it harder to debug and understand code, particularly for those unfamiliar with the abstraction layers and implementation details.
* Overuse of abstraction can result in decreased performance due to the additional layers of code and indirection.





Encapsulation in Java

* Encapsulation in Java is a fundamental concept in object-oriented programming (OOP) that refers to the bundling of data and methods that operate on that data within a single unit, which is called a class in Java.
* Java Encapsulation is a way of hiding the implementation details of a class from outside access and only exposing a public interface that can be used to interact with the class.
* In Java, encapsulation is achieved by declaring the instance variables of a class as private, which means they can only be accessed within the class.
* To allow outside access to the instance variables, public methods called getters and setters are defined, which are used to retrieve and modify the values of the instance variables, respectively.
* By using getters and setters, the class can enforce its own data validation rules and ensure that its internal state remains consistent.