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.

Why Do We Need Java Inheritance?

- **Code Reusability:** The code written in the Superclass is common to all subclasses. Child classes can directly use the parent class code.
- **Method Overriding:** Method Overriding is achievable only through Inheritance. It is one of the ways by which Java achieves Run Time Polymorphism.
- **Abstraction:** The concept of abstract where we do not have to provide all details is achieved through inheritance. Abstraction only shows the functionality to the user.

Important Terminologies Used in Java Inheritance

- **Class:** 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.
- **Super Class/Parent Class:** The class whose features are inherited is known as a superclass(or a base class or a parent class).
- **Sub Class/Child Class:** The class that inherits the other class is known as a subclass(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.

existing class. By doing this, we are reusing the fields and methods of the existing class.

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.

Syntax :

```
class DerivedClass extends BaseClass
{
   //methods and fields
}
```

Inheritance in Java Example

Example: In the below example of inheritance, class Bicycle is a base class, class MountainBike is a derived class that extends the Bicycle class and class Test is a driver class to run the program.

Java

```java
// Java program to illustrate the
// concept of inheritance

// base class
class Bicycle {
    // the Bicycle class has two fields
    public int gear;
    public int speed;

    // the Bicycle class has one constructor
    public Bicycle(int gear, int speed)
    {
        this.gear = gear;
        this.speed = speed;
    }

    // the Bicycle class has three methods
    public void applyBrake(int decrement)
    {
```

```
    {
        speed += increment;
    }

    // toString() method to print info of Bicycle
    public String toString()
    {
        return ("No of gears are " + gear + "\n"
                + "speed of bicycle is " + speed);
    }
}

// derived class
class MountainBike extends Bicycle {

    // the MountainBike subclass adds one more field
    public int seatHeight;

    // the MountainBike subclass has one constructor
    public MountainBike(int gear, int speed,
                        int startHeight)
    {
        // invoking base-class(Bicycle) constructor
        super(gear, speed);
        seatHeight = startHeight;
    }

    // the MountainBike subclass adds one more method
    public void setHeight(int newValue)
    {
        seatHeight = newValue;
    }

    // overriding toString() method
    // of Bicycle to print more info
    @Override public String toString()
    {
        return (super.toString() + "\nseat height is "
                + seatHeight);
    }
}

// driver class
public class Test {
    public static void main(String args[])
    {

        MountainBike mb = new MountainBike(3, 100, 25);
        System.out.println(mb.toString());
```

Output

```
No of gears are 3
speed of bicycle is 100
seat height is 25
```

In the above program, when an object of MountainBike class is created, a copy of all methods and fields of the superclass acquires memory in this object. That is why by using the object of the subclass we can also access the members of a superclass.

Please note that during inheritance only the object of the subclass is created, not the superclass. For more, refer to Java Object Creation of Inherited Class.

Example 2: In the below example of inheritance, class Employee is a base class, class Engineer is a derived class that extends the Employee class and class Test is a driver class to run the program.

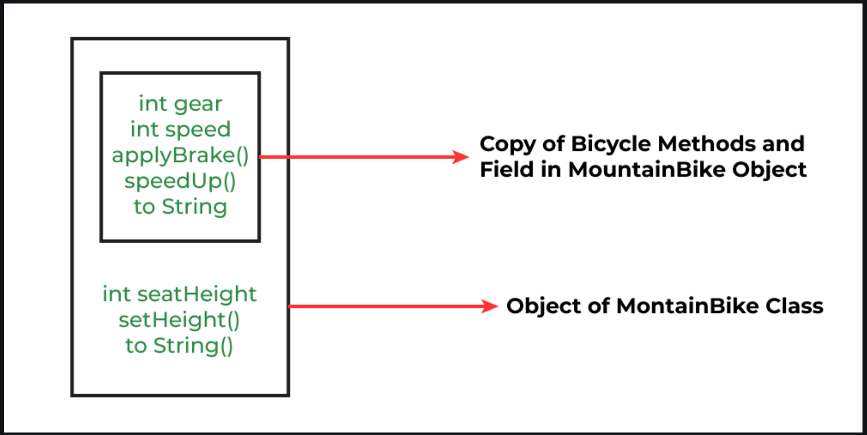
Java

```
// Java Program to illustrate Inheritance (concise)

import java.io.*;

// Base or Super Class
class Employee {
    int salary = 60000;
}

// Inherited or Sub Class
class Engineer extends Employee {
    int benefits = 10000;
}

// Driver Class
class Gfg {
    public static void main(String args[])
    {
        Engineer E1 = new Engineer();
}
```

Output

```
Salary : 60000
Benefits : 10000
```

Illustrative image of the program:


int gear
int speed
applyBrake()
speedUp()
to String
Copy of Bicycle Methods and Field in MountainBike Object
int seatHeight
setHeight()
to String()
Object of MontainBike Class

In practice, inheritance, and polymorphism are used together in Java to achieve fast performance and readability of code.

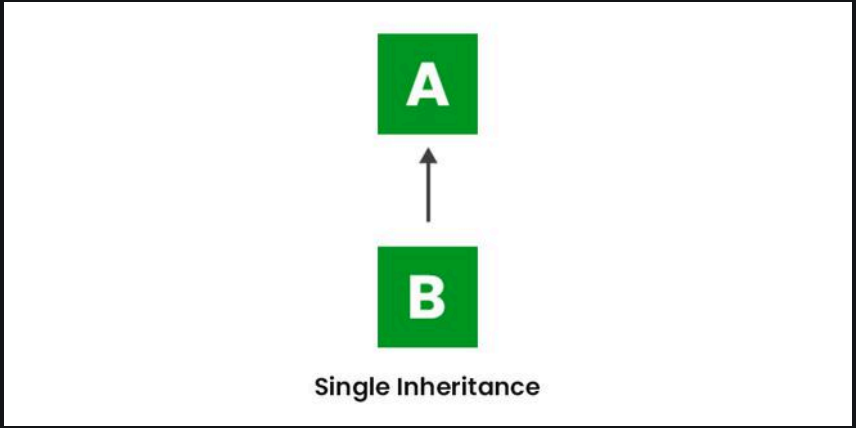
Java Inheritance Types

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, subclasses inherit the features of one superclass. In the



Single inheritance

Java

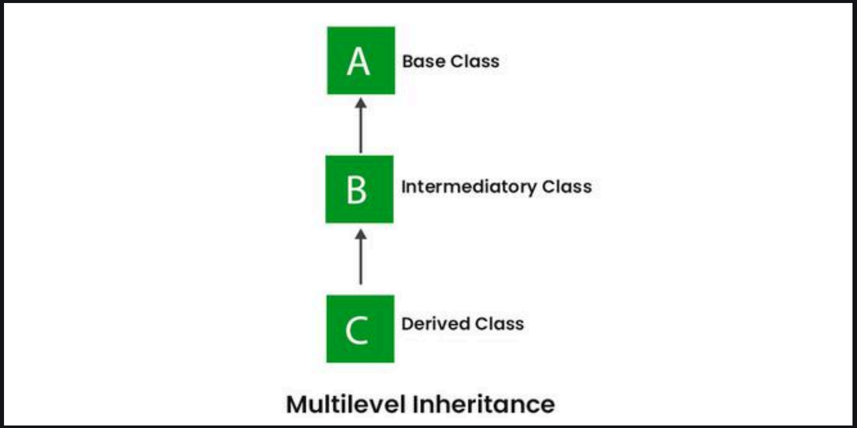
```java
// Java program to illustrate the
// concept of single inheritance
import java.io.*;
import java.lang.*;
import java.util.*;

// Parent class
class One {
    public void print_geek()
    {
        System.out.println("Geeks");
    }
}

class Two extends One {
    public void print_for() { System.out.println("for"); }
}

// Driver class
public class Main {
      // Main function
    public static void main(String[] args)
    {
        Two g = new Two();
        g.print_geek();
        g.print_for();
        g.print_geek();
```

Output

```
Geeks
for
Geeks
```

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.



Multilevel Inheritance

Java

```java
// Java program to illustrate the
// concept of Multilevel inheritance
import java.io.*;
import java.lang.*;
import java.util.*;

class One {
```

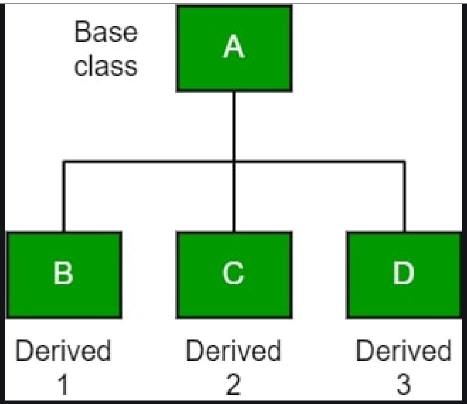
```java
    }
}

class Two extends One {
    public void print_for() { System.out.println("for"); }
}

class Three extends Two {
    public void print_geek()
    {
        System.out.println("Geeks");
    }
}

// Drived class
public class Main {
    public static void main(String[] args)
    {
        Three g = new Three();
        g.print_geek();
        g.print_for();
        g.print_geek();
    }
}
```

Output

```
Geeks
for
Geeks
```

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.



Java

```
// Java program to illustrate the
// concept of Hierarchical  inheritance

class A {
    public void print_A() { System.out.println("Class A"); }
}

class B extends A {
    public void print_B() { System.out.println("Class B"); }
}

class C extends A {
    public void print_C() { System.out.println("Class C"); }
}

class D extends A {
    public void print_D() { System.out.println("Class D"); }
}

// Driver Class
public class Test {
    public static void main(String[] args)
    {
        B obj_B = new B();
        obj_B.print_A();
        obj_B.print_B();

        C obj_C = new C();
        obj_C.print_A();
        obj_C.print_C();
```

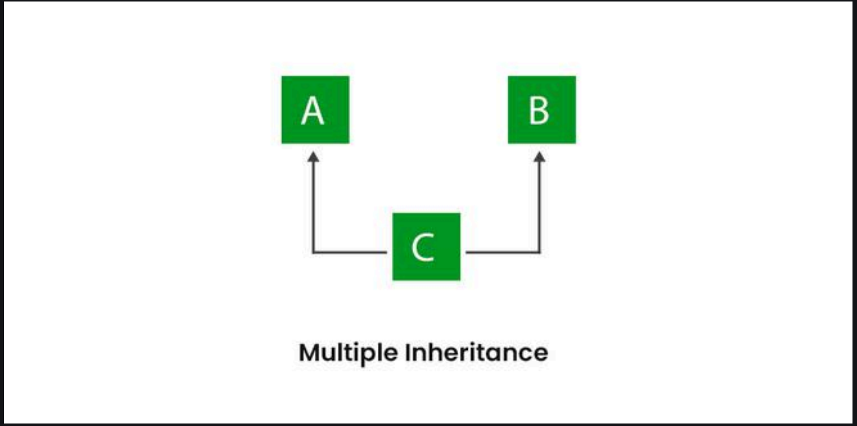
```
    }
}
```

Output

```
Class A
Class B
Class A
Class C
Class A
Class 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. In the image below, Class C is derived from interfaces A and B.



Multiple Inheritance

Java

```
import java.util.*;

interface One {
    public void print_geek();
}

interface Two {
    public void print_for();
}

interface Three extends One, Two {
    public void print_geek();
}
class Child implements Three {
    @Override public void print_geek()
    {
        System.out.println("Geeks");
    }

    public void print_for() { System.out.println("for"); }
}

// Drived class
public class Main {
    public static void main(String[] args)
    {
        Child c = new Child();
        c.print_geek();
        c.print_for();
        c.print_geek();
    }
}
```

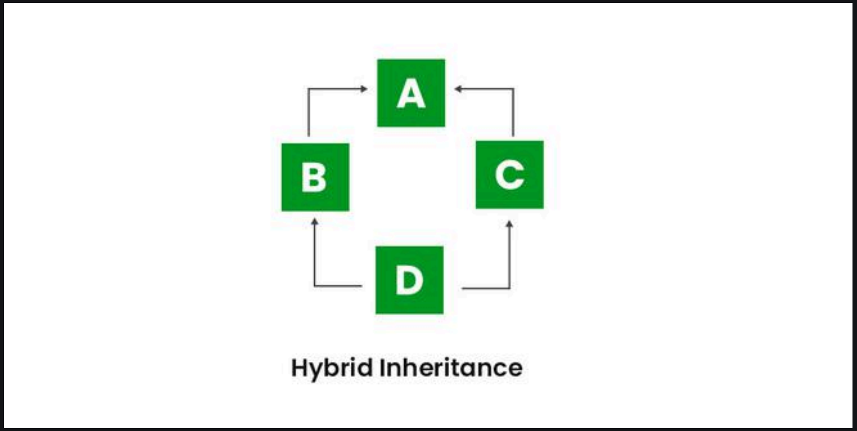
Output

```
Geeks
for
Geeks
```

5. Hybrid Inheritance

It is a mix of two or more of the above types of inheritance. Since Java

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.



Hybrid Inheritance

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.

Java

```
public class SolarSystem {
}
public class Earth extends SolarSystem {
}
public class Mars extends SolarSystem {
}
public class Moon extends Earth {
}
```

- SolarSystem is the superclass of Earth class.
- SolarSystem is the superclass of Mars class.
- Earth and Mars are subclasses of SolarSystem class.
- Moon is the subclass of both Earth and SolarSystem classes.

Java

```
class SolarSystem {
}
class Earth extends SolarSystem {
}
class Mars extends SolarSystem {
}
public class Moon extends Earth {
    public static void main(String args[])
    {
        SolarSystem s = new SolarSystem();
        Earth e = new Earth();
        Mars m = new Mars();

        System.out.println(s instanceof SolarSystem);
        System.out.println(e instanceof Earth);
        System.out.println(m instanceof SolarSystem);
    }
}
```

Output

```
true
true
true
```

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.

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:

1. 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.
2. 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.
3. Class Hierarchy: Inheritance allows for the creation of a class hierarchy, which can be used to model real-world objects and their relationships.

Java Arrays Java Strings Java OOPs Java Collection Java 8 Tutorial Java Multithreading Java Exceptio

methods of the superclass, which allows them to change their behavior in different ways.

Disadvantages of Inheritance in Java:

1. Complexity: Inheritance can make the code more complex and harder to understand. This is especially true if the inheritance hierarchy is deep or if multiple inheritances is used.
2. Tight Coupling: Inheritance creates a tight coupling between the superclass and subclass, making it difficult to make changes to the superclass without affecting the subclass.

Conclusion

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

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.

FAQs in Inheritance

1. What is Inheritance Java?

Inheritance is a concept of OOPs where one class inherits from another class that can reuse the methods and fields of the parent class.

2. What are the 4 types of inheritance in Java?

There are Single, Multiple, Multilevel, Hierarchical and Hybrid

3. What is the use of extend keyword?

Extend keyword is used for inheriting one class into another.

4. What is an example of inheritance in Java?

A real-world example of Inheritance in Java is mentioned below:

> *Consider a group of vehicles. You need to create classes for Bus, Car, and Truck. The methods fuelAmount(), capacity(), applyBrakes() will*