**Polymorphism in Java**

The word polymorphism means having many forms. In simple words, we can define Java Polymorphism as the ability of a message to be displayed in more than one form.

**Real-life Illustration of Polymorphism in Java:** A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, and an employee. So the same person possesses different behaviors in different situations. This is called polymorphism.

**What is Polymorphism in Java?**

Polymorphism is considered one of the important features of Object-Oriented Programming. Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations. The word “poly” means many and “morphs” means forms, So it means many forms.

**Types of Java Polymorphism**

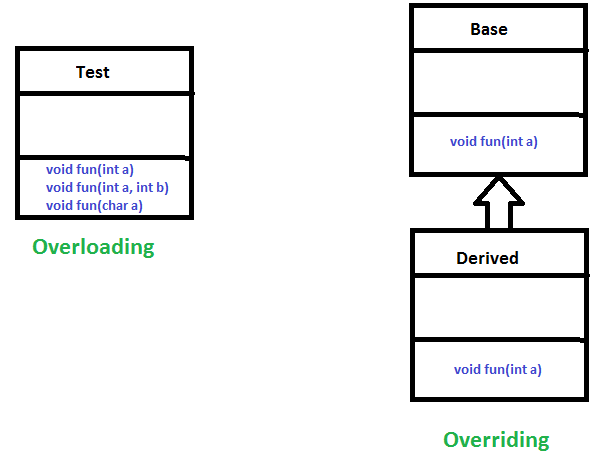
In Java Polymorphism is mainly divided into two types:

* Compile-time Polymorphism
* Runtime Polymorphism

**Compile-Time Polymorphism in Java**

It is also known as static polymorphism. This type of polymorphism is achieved by function overloading or operator overloading.

***Note:****But Java doesn’t support the Operator Overloading.*



**Method Overloading**

When there are multiple functions with the same name but different parameters then these functions are said to be **overloaded**. Functions can be overloaded by changes in the number of arguments or/and a change in the type of arguments.

**Example 1:**

|  |
| --- |
| // Java Program for Method overloading  // By using Different Types of Arguments  // Class 1  // Helper class  class Helper {      // Method with 2 integer parameters      static int Multiply(int a, int b)      {          // Returns product of integer numbers          return a \* b;      }      // Method 2      // With same name but with 2 double parameters      static double Multiply(double a, double b)      {          // Returns product of double numbers          return a \* b;      }  }  // Class 2  // Main class  class GFG {      // Main driver method      public static void main(String[] args)      {          // Calling method by passing          // input as in arguments          System.out.println(Helper.Multiply(2, 4));          System.out.println(Helper.Multiply(5.5, 6.3));      }  } |

**Output**

8

34.65

**Example 2:**

Java

|  |
| --- |
| // Java program for Method Overloading  // by Using Different Numbers of Arguments  // Class 1  // Helper class  class Helper {      // Method 1      // Multiplication of 2 numbers      static int Multiply(int a, int b)      {          // Return product          return a \* b;      }      // Method 2      // // Multiplication of 3 numbers      static int Multiply(int a, int b, int c)      {          // Return product          return a \* b \* c;      }  }  // Class 2  // Main class  class GFG {      // Main driver method      public static void main(String[] args)      {          // Calling method by passing          // input as in arguments          System.out.println(Helper.Multiply(2, 4));          System.out.println(Helper.Multiply(2, 7, 3));      }  } |

**Output**

8

42

**Subtypes of Compile-time Polymorphism**

**1. Function Overloading**

It is a feature in C++ where multiple functions can have the same name but with different parameter lists. The compiler will decide which function to call based on the number and types of arguments passed to the function.

**2. Operator Overloading**

It is a feature in C++ where the operators such as +, -, \*, etc. can be given additional meanings when applied to user-defined data types.

**3. Template**

it is a powerful feature in C++ that allows us to write generic functions and classes. A template is a blueprint for creating a family of functions or classes.

**Runtime Polymorphism in Java**

It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding. **Method overriding**, on the other hand, occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be **overridden**.

**Example**

|  |
| --- |
| // Java Program for Method Overriding  // Class 1  // Helper class  class Parent {      // Method of parent class      void Print()      {          // Print statement          System.out.println("parent class");      }  }  // Class 2  // Helper class  class subclass1 extends Parent {      // Method      void Print() { System.out.println("subclass1"); }  }  // Class 3  // Helper class  class subclass2 extends Parent {      // Method      void Print()      {          // Print statement          System.out.println("subclass2");      }  }  // Class 4  // Main class  class GFG {      // Main driver method      public static void main(String[] args)      {          // Creating object of class 1          Parent a;          // Now we will be calling print methods          // inside main() method          a = new subclass1();          a.Print();          a = new subclass2();          a.Print();      }  } |

**Output**

subclass1

subclass2

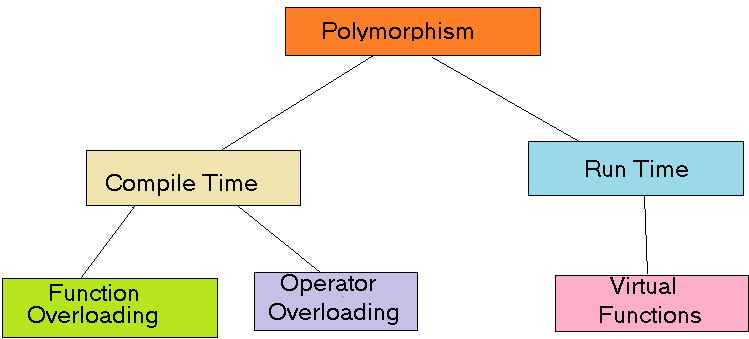
**Explanation of the above code:**

*Here in this program, When an object of a child class is created, then the method inside the child class is called. This is because The method in the parent class is overridden by the child class. Since The method is overridden, This method has more priority than the parent method inside the child class. So, the body inside the child class is executed.*

**Subtype of Run-time Polymorphism**

**i. Virtual functions**

It allows an object of a derived class to behave as if it were an object of the base class. The derived class can override the virtual function of the base class to provide its own implementation. The function call is resolved at runtime, depending on the actual type of the object.



Polymorphism in Java is a concept that allows objects of different classes to be treated as objects of a common class. It enables objects to behave differently based on their specific class type.

**Advantages of Polymorphism in Java**

1. Increases code reusability by allowing objects of different classes to be treated as objects of a common class.
2. Improves readability and maintainability of code by reducing the amount of code that needs to be written and maintained.
3. Supports dynamic binding, enabling the correct method to be called at runtime, based on the actual class of the object.
4. Enables objects to be treated as a single type, making it easier to write generic code that can handle objects of different types.

**Disadvantages of Polymorphism in Java**

1. Can make it more difficult to understand the behavior of an object, especially if the code is complex.
2. This may lead to performance issues, as polymorphic behavior may require additional computations at runtime.

**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.
* **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. 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 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 -= decrement;      }      public void speedUp(int increment)      {          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.

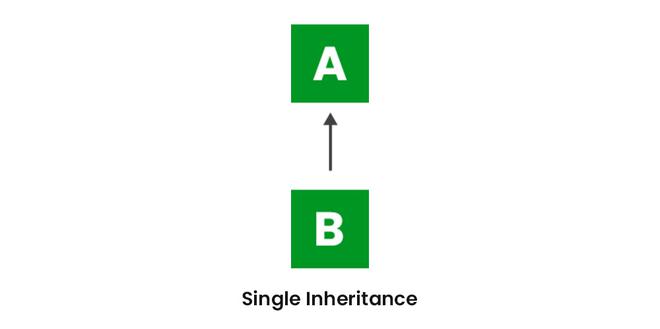
**Java Inheritance Types**

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 image below, class A serves as a base class for the derived class B.



|  |
| --- |
| 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 {      public static void main(String[] args)      {          Two g = new Two();          g.print\_geek();          g.print\_for();          g.print\_geek();      }  } |

**Output**

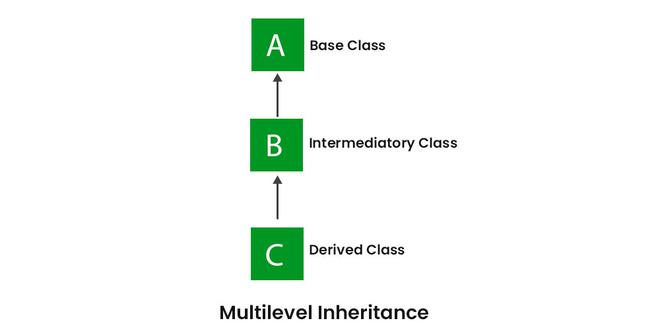
Geeks

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.



|  |
| --- |
| import java.io.\*;  import java.lang.\*;  import java.util.\*;  class One {      public void print\_geek()      {          System.out.println("Geeks");      }  }  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**

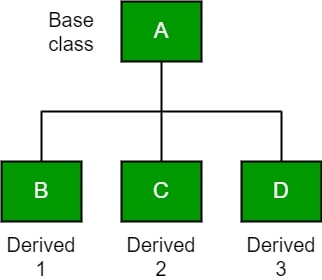
Geeks

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.



|  |
| --- |
| 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();            D obj\_D = new D();          obj\_D.print\_A();          obj\_D.print\_D();      }  } |

**Output**

Class A

Class B

Class A

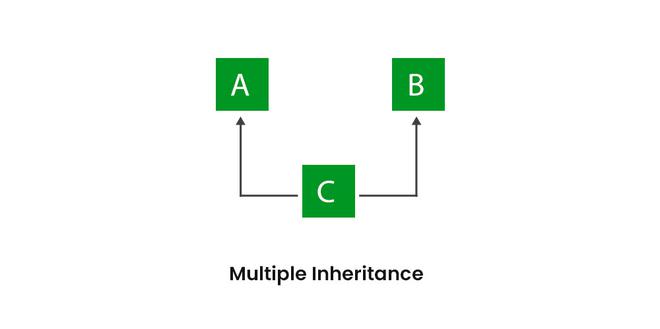
Class C

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.



|  |
| --- |
| import java.io.\*;  import java.lang.\*;  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**

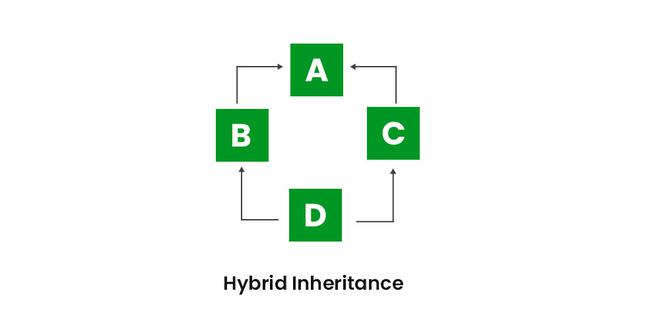
Geeks

for

Geeks

**5. Hybrid Inheritance**

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

Java

|  |
| --- |
| public class SolarSystem {  }  public class Earth extends SolarSystem {  }  public class Mars extends SolarSystem {  }  public class Moon extends Earth {  } |

Now, based on the above example, in Object-Oriented terms, the following are true:-

* 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.
* 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:**

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.
4. 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 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.

**Difference between Inheritance and Polymorphism:**

| **Inheritance** | **Polymorphism** |
| --- | --- |
| Inheritance is one in which a new class is created (derived class) that inherits the features from the already existing class(Base class). | Whereas polymorphism is that which can be defined in multiple forms. |
| It is basically applied to classes. | Whereas it is basically applied to functions or methods. |
| Inheritance supports the concept of reusability and reduces code length in object-oriented programming. | Polymorphism allows the object to decide which form of the function to implement at compile-time (overloading) as well as run-time (overriding). |
| Inheritance can be single, hybrid, multiple, hierarchical and multilevel inheritance. | Whereas it can be compiled-time polymorphism (overload) as well as run-time polymorphism (overriding). |
| It is used in pattern designing. | While it is also used in pattern designing. |
| **Example :**  The class bike can be inherit from the class of two-wheel vehicles, which is turn could be a subclass of vehicles. | **Example :**  The class bike can have method name set\_color(), which changes the bike’s color based on the name of color you have entered. |