

Unit 4: Inheritance and Polymorphism

Inheritance and reuse

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.

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.

Inheritance

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.

Inheritance: Syntax



```
class SubClassName extends SuperClassName  
{  
    //methods and fields  
}
```

The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

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.




```
// Car.java (superclass)
public class Car {
    private String make;
    private String model;
    private int year;
    private String fuelType;

    public Car(String make, String model, int year, String fuelType) {
        this.make = make;
        this.model = model;
        this.year = year;
        this.fuelType = fuelType;
    }

    public void start() {
        System.out.println("Starting the car");
    }

    public void accelerate() {
        System.out.println("Accelerating the car");
    }

    public void displayInfo() {
        System.out.println("Make: " + make);
        System.out.println("Model: " + model);
        System.out.println("Year: " + year);
        System.out.println("Fuel Type: " + fuelType);
    }
}
```



```
// ElectricCar.java (subclass)
public class ElectricCar extends Car {
    private int batteryCapacity;

    public ElectricCar(String make, String model, int year, int batteryCapacity) {
        super(make, model, year, "ELECTRIC");
        this.batteryCapacity = batteryCapacity;
    }

    @Override
    public void start() {
        System.out.println("Starting the electric car");
    }

    public void chargeBattery() {
        System.out.println("Charging the battery");
    }

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Battery Capacity: " + batteryCapacity + " kWh");
    }
}
```




```
// CarAndElectricCarExample.java
```

```
public class CarAndElectricCarExample {  
    public static void main(String[] args) {  
        Car car = new Car("Toyota", "Corolla", 2020, "GASOLINE");  
        ElectricCar electricCar = new ElectricCar("Tesla", "Model 3", 2022, 75);  
  
        car.start();  
        car.accelerate();  
        car.displayInfo();  
  
        electricCar.start();  
        electricCar.accelerate();  
        electricCar.displayInfo();  
        electricCar.chargeBattery();  
    }  
}
```

Java Inheritance Types

1. Single Inheritance: In single inheritance, one child class inherits from one parent class. This is the most basic type of inheritance in Java.
2. Multilevel Inheritance: In multilevel inheritance, a child class extends another child class, which in turn extends a parent class. This forms a chain of inheritance.
3. Hierarchical Inheritance: In hierarchical inheritance, multiple child classes extend the same parent class.
4. Multiple Inheritance: Java does not support multiple inheritance directly, but it can be achieved through interfaces. In multiple inheritance, a child class can inherit from multiple parent classes.
5. Hybrid Inheritance: Hybrid inheritance is a combination of two or more types of inheritance. It can be achieved through a combination of multilevel inheritance and hierarchical inheritance.

A



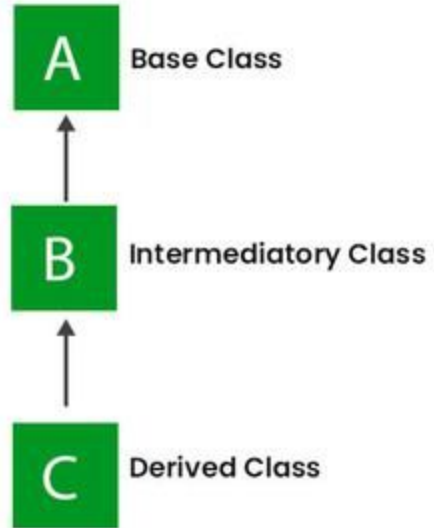
B

Single Inheritance



// Single Inheritance

```
class Animal {  
    void eat() {  
        System.out.println("eating");  
    }  
}  
  
class Dog extends Animal {  
    void bark() {  
        System.out.println("barking");  
    }  
}
```



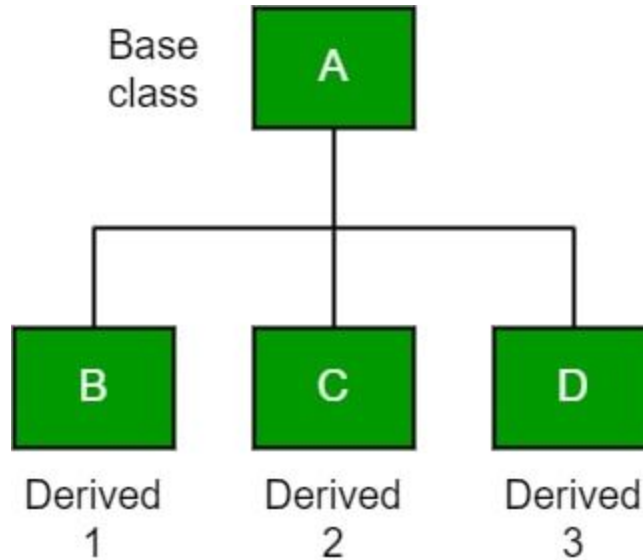
Multilevel Inheritance



// Multilevel Inheritance

```
class Animal {  
    void eat() {  
        System.out.println("eating");  
    }  
}  
  
class Dog extends Animal {  
    void bark() {  
        System.out.println("barking");  
    }  
}  
  
class Puppy extends Dog {  
    void play() {  
        System.out.println("playing");  
    }  
}
```

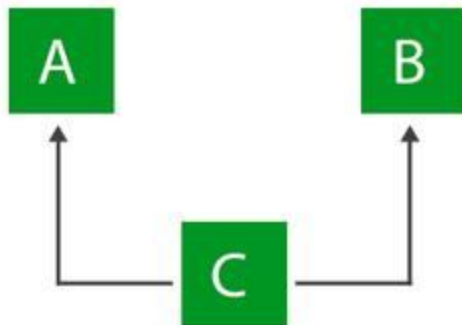
Hierarchical Inheritance





// Hierarchical Inheritance

```
class Animal {  
    void eat() {  
        System.out.println("eating");  
    }  
}  
  
class Dog extends Animal {  
    void bark() {  
        System.out.println("barking");  
    }  
}  
  
class Cat extends Animal {  
    void meow() {  
        System.out.println("meowing");  
    }  
}
```

Multiple Inheritance

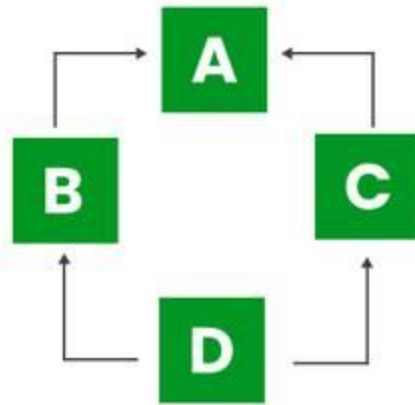


```
// Multiple Inheritance
interface Flyable {
    void fly();
}

interface Walkable {
    void walk();
}

class Bird implements Flyable, Walkable {
    public void fly() {
        System.out.println("flying");
    }

    public void walk() {
        System.out.println("walking");
    }
}
```



Hybrid Inheritance



// Hybrid Inheritance

```
class Animal {  
    void eat() {  
        System.out.println("eating");  
    }  
}  
  
class Mammal extends Animal {  
    void giveBirth() {  
        System.out.println("giving birth");  
    }  
}  
  
class Dog extends Mammal {  
    void bark() {  
        System.out.println("barking");  
    }  
}  
  
class Cat extends Mammal {  
    void meow() {  
        System.out.println("meowing");  
    }  
}
```

Polymorphism

Polymorphism is a fundamental concept in object-oriented programming (OOP), and it's a key feature of the Java programming language. It enables you to write more flexible and reusable code by allowing objects of different classes to be treated as objects of a common superclass.

There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism.

References:

<https://java-tutorial.dev/docs/Java-Object-Oriented-Programming/Polymorphism-In-Java>

<https://www.javatpoint.com/runtime-polymorphism-in-java>

DIY: Method Overriding and Method Overloading

DIY: Compile-Time Polymorphism and Runtime Polymorphism

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.

References: <https://www.javatpoint.com/abstract-class-in-java>

Abstract Classes:

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.

Rules for Java Abstract class



1

An abstract class must be declared with an abstract keyword.

2

It can have abstract and non-abstract methods.

3

It cannot be instantiated.

4

It can have final methods

5

It can have constructors and static methods also.

Abstract Method:

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

Example:

```
abstract void printStatus();//no method body and abstract
```



```
abstract class Bike{
    abstract void run();
}
class Honda4 extends Bike{
void run(){System.out.println("running safely");}
public static void main(String args[]){
    Bike obj = new Honda4();
    obj.run();
}
}
```

Packages

A java package is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

Advantage of Java Package

1. Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2. Java package provides access protection.
3. Java package removes naming collision.

References: <https://www.javatpoint.com/package>

Interface


An interface in Java is a blueprint of a class. It has static constants and abstract methods. It is a blueprint of a behavior and contains static constants and abstract methods. Interfaces are used to achieve abstraction and multiple inheritances in Java.

References: <https://www.javatpoint.com/interface-in-java>

Why use Interface



Example



```
public interface Animal {
    public void animalSound();
    public void sleep();
}

public class Pig implements Animal {
    public void animalSound() {
        System.out.println("The pig says: wee wee");
    }
    public void sleep() {
        System.out.println("The pig is sleeping");
    }
}
```


Lambda

Lambda expression is a new and important feature of Java which was included in Java SE 8. It provides a clear and concise way to represent one method interface using an expression. It is very useful in collection library. It helps to iterate, filter and extract data from collection.

The Lambda expression is used to provide the implementation of an interface which has functional interface. It saves a lot of code. In case of lambda expression, we don't need to define the method again for providing the implementation. Here, we just write the implementation code.

References: <https://www.javatpoint.com/java-lambda-expressions>

Why use Lambda

- To provide the implementation of Functional interface.
- Less coding.

Lambda Expression Syntax

`(argument-list) -> {body}`

Lambda expression is consisted of three components.

- 1) Argument-list: It can be empty or non-empty as well.
- 2) Arrow-token: It is used to link arguments-list and body of expression.
- 3) Body: It contains expressions and statements for lambda expression.

Lambda Expression Parameters

There are three Lambda Expression Parameters are mentioned below:

Zero Parameter

Single Parameter

Multiple Parameters

No Parameter Syntax

```
() -> {  
  //Body of no parameter lambda  
}
```

One Parameter Syntax

```
(p1) -> {  
  //Body of single parameter lambda  
}
```

Two Parameter Syntax

```
(p1,p2) -> {  
  //Body of multiple parameter lambda  
}
```

Example



```
import java.util.ArrayList;

public class LambdaExample {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<>();
        list.add("Apple");
        list.add("Banana");
        list.add("Cherry");

        list.forEach(s -> System.out.println(s));
    }
}
```

Practical

1. Create a Book class with attributes title and author. Add a method displayInfo() that prints the book's title and author.
2. Create a Square class that inherits from a Shape class. Add an attribute sideLength and a method calculateArea() that returns the area of the square.
3. Create a Fruit class with an attribute color. Then, create an Apple class that inherits from Fruit and adds an attribute taste. Write a method displayInfo() in the Apple class that prints the apple's color and taste.
4. Create a Vehicle class with an attribute speed. Then, create a Bicycle class that inherits from Vehicle and adds an attribute gears. Write a method displayInfo() in the Bicycle class that prints the bicycle's speed and number of gears.
5. Create a Calculator class with a method add(int x, int y) that adds two numbers. Then, create a ScientificCalculator class that inherits from Calculator and overrides the add() method to perform multiplication instead of addition. Write a main() method that creates a ScientificCalculator object and calls the add() method.
6. Create an AbstractAnimal abstract class with an abstract method sound(). Then, create a Dog class that extends AbstractAnimal and provides an implementation for the sound() method. Write a main() method that creates a Dog object and calls the sound() method.
7. Create an AbstractShape abstract class with an abstract calculatePerimeter() method. Create a Rectangle class that extends AbstractShape and provides an implementation for calculatePerimeter(). Write a main() method to create a Rectangle object and call calculatePerimeter().
8. Create a Printable interface with a method print(). Then, create a Document class that implements the Printable interface and provides an implementation for the print() method. Write a main() method that creates a Document object and calls the print() method.
9. Create a Comparable interface with a compareTo() method. Create a Student class that implements Comparable and provides an implementation for compareTo(). Write a main() method to create a Student object and call compareTo().
10. Create a MathOperation interface with a method operation(int x, int y). Then, use a lambda function to implement the operation() method to perform addition. Write a main() method that creates a MathOperation object and calls the operation() method.