**Inheritance in Java**

**Super Class:** The class whose features are inherited is known as superclass(or a base class or a parent class).

**Sub 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.

**Syntax :**

class derived-class extends base-class

{

//methods and fields

}

**Inheritance Program**

class Animal {

// field and method of the parent class

String name;

public void eat()

{

System.out.println("I can eat");

}

}

// inherit from Animal

class Dog extends Animal {

// new method in subclass

public void display() {

System.out.println("My name is " + name);

}

}

class Main {

public static void main(String[] args) {

// create an object of the subclass

Dog labrador = new Dog();

// access field of superclass

labrador.name = "Rohu";

labrador.display();

// call method of superclass

// using object of subclass

labrador.eat();

}

}

## **Types of inheritance**

### **1. Single Inheritance**

### **2. Multilevel Inheritance**

### **3. Hierarchical Inheritance**

### **4. Multiple Inheritance**

### **5. Hybrid Inheritance**

### **Multilevel Inheritance**

// Java program to illustrate the

// concept of Multilevel inheritance

import java.io.\*;

import java.lang.\*;

import java.util.\*;

class one

{

public void geeks()

{

System.out.println("Geeks");

}

}

class two extends one {

public void for()

{

System.out.println("for");

}

}

class three extends two {

public void geek()

{

System.out.println("Geek");

}

}

// Drived class

public class Main {

public static void main(String[] args)

{

three g = new three();

g.geeks();

g.for();

g.geek();

}}

Hierarchical Inheritance:

// 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();

D obj\_D = new D();

obj\_D.print\_A();

obj\_D.print\_D();

}

}

### **Multiple Inheritance**

interface Backend {

// abstract class

public void connectServer();

}

class Frontend {

public void responsive(String str) {

System.out.println(str + " can also be used as frontend.");

}

}

// Language extends Frontend class

// Language implements Backend interface

class Language extends Frontend implements Backend {

String language = "Java";

// implement method of interface

public void connectServer() {

System.out.println(language + " can be used as backend language.");

}

public static void main(String[] args) {

// create object of Language class

Language java = new Language();

java.connectServer();

// call the inherited method of Frontend class

java.responsive(java.language);

}

}

### **Hybrid Inheritance**

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 the superclass of Earth class.

SolarSystem the superclass of Mars class.

Earth and Mars are subclasses of SolarSystem class.

Moon is the subclass of both Earth and SolarSystem classes

**Program**

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);

}

}

## **Method Overriding in Java Inheritance**

// method in the superclass

public void eat() {

System.out.println("I can eat");

}

}

// Dog inherits Animal

class Dog extends Animal {

// overriding the eat() method

@Override

public void eat() {

System.out.println("I eat dog food");

}

// new method in subclass

public void bark() {

System.out.println("I can bark");

}

}

class Main {

public static void main(String[] args) {

// create an object of the subclass

Dog labrador = new Dog();

// call the eat() method

labrador.eat();

labrador.bark();

}

}

## **super Keyword in Java Inheritance**

// Dog inherits Animal

class Dog extends Animal {

// overriding the eat() method

@Override

public void eat() {

// call method of superclass

super.eat();

System.out.println("I eat dog food");

}

// new method in subclass

public void bark() {

System.out.println("I can bark");

}

}

class Main {

public static void main(String[] args) {

// create an object of the subclass

Dog labrador = new Dog();

// call the eat() method

labrador.eat();

labrador.bark();

}

}

# **Java Method Overriding**

class Animal {

public void displayInfo() {

System.out.println("I am an animal.");

}

}

class Dog extends Animal {

@Override

public void displayInfo() {

System.out.println("I am a dog.");

}

}

class Main {

public static void main(String[] args) {

Dog d1 = new Dog();

d1.displayInfo();

}

}

# **Java Method Overloading**

class MethodOverloading {

private static void display(int a){

System.out.println("Arguments: " + a);

}

private static void display(int a, int b){

System.out.println("Arguments: " + a + " and " + b);

}

public static void main(String[] args) {

display(1);

display(1, 4);

}

}

**Type conversion**

Widening or Automatic Type Conversion

class Test

{

public static void main(String[] args)

{

int i = 100;

// automatic type conversion

long l = i;

// automatic type conversion

float f = l;

System.out.println("Int value "+i);

System.out.println("Long value "+l);

System.out.println("Float value "+f);

}

}

Narrowing or Explicit Conversion

//Java program to illustrate incompatible data

// type for explicit type conversion

public class Test

{

public static void main(String[] argv)

{

char ch = 'c';

int num = 88;

ch = num;

}

}

//Java program to illustrate explicit type conversion

class Test

{

public static void main(String[] args)

{

double d = 100.04;

//explicit type casting

long l = (long)d;

//explicit type casting

int i = (int)l;

System.out.println("Double value "+d);

//fractional part lost

System.out.println("Long value "+l);

//fractional part lost

System.out.println("Int value "+i);

}

}

## **Type conversion from int to String**

class Main {

public static void main(String[] args) {

// create int type variable

int num = 10;

System.out.println("The integer value is: " + num);

// converts int to string type

String data = String.valueOf(num);

System.out.println("The string value is: " + data);

}

}

## **Type conversion from String to int**

class Main {

public static void main(String[] args) {

// create string type variable

String data = "10";

System.out.println("The string value is: " + data);

// convert string variable to int

int num = Integer.parseInt(data);

System.out.println("The integer value is: " + num);

}

}