1.To develop a vehicle class hierarchy in Java to demonstrate the concept of polymorphism.

Step 1:-Declare a super class called vehicle with data

elements doors,wheels and seats.

Step 2:-Derive another class called car and invoke a

function tostring() to display the variables.

Step 3:-Derive another class calledmotorcycle with same

data and method called setseats() .

Step 4:-Declare another sub class called Truck with 2

constructors and finally assign values to variables.

Step 5:-In the main function, create an object for class

motorcycle and display all details of sub classes through

object.

// A super class called vehicle with data elements doors, wheels and seats

class Vehicle {

// Data elements

int doors;

int wheels;

int seats;

// A constructor to initialize the data elements

public Vehicle(int doors, int wheels, int seats) {

this.doors = doors;

this.wheels = wheels;

this.seats = seats;

}

}

// A sub class called car that inherits from vehicle and overrides the toString method to display the variables

class Car extends Vehicle {

// A constructor to initialize the data elements using the super class constructor

public Car(int doors, int wheels, int seats) {

super(doors, wheels, seats);

}

// A method to display the variables

public String toString() {

return "Car: doors = " + doors + ", wheels = " + wheels + ", seats = " + seats;

}

}

// A sub class called motorcycle that inherits from vehicle and has a method called setSeats to change the number of seats

class Motorcycle extends Vehicle {

// A constructor to initialize the data elements using the super class constructor

public Motorcycle(int doors, int wheels, int seats) {

super(doors, wheels, seats);

}

// A method to change the number of seats

public void setSeats(int seats) {

this.seats = seats;

}

// A method to display the variables

public String toString() {

return "Motorcycle: doors = " + doors + ", wheels = " + wheels + ", seats = " + seats;

}

}

// A sub class called truck that inherits from vehicle and has two constructors

class Truck extends Vehicle {

// A constructor to initialize the data elements using the super class constructor

public Truck(int doors, int wheels, int seats) {

super(doors, wheels, seats);

}

// Another constructor to assign default values to the data elements

public Truck() {

this.doors = 2;

this.wheels = 6;

this.seats = 3;

}

// A method to display the variables

public String toString() {

return "Truck: doors = " + doors + ", wheels = " + wheels + ", seats = " + seats;

}

}

// A class to test the program

public class Test {

public static void main(String[] args) {

// Create an object for class motorcycle and display all details of sub classes through object

Motorcycle m = new Motorcycle(0,2,1);

System.out.println(m); // Display the details of motorcycle

m.setSeats(2); // Change the number of seats of motorcycle

System.out.println(m); // Display the updated details of motorcycle

// Create an object for class car and display its details

Car c = new Car(4,4,5);

System.out.println(c);

// Create an object for class truck using the default constructor and display its details

Truck t = new Truck();

System.out.println(t);

// Create another object for class truck using the parameterized constructor and display its details

Truck t2 = new Truck(4,8,4);

System.out.println(t2);

}

}

2.What is super class and subclass?

[A super class and a subclass are two types of classes that are related by **inheritance**1](https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html). [Inheritance is a concept that allows one class to inherit the attributes and methods from another class1](https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html).

[A super class, also known as a parent class or a base class, is the class that is inherited from1](https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html)[2](https://careerkarma.com/blog/java-super/). [A subclass, also known as a child class or a derived class, is the class that inherits from the super class1](https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html)[2](https://careerkarma.com/blog/java-super/).

For example, in the following code snippet, Vehicle is a super class and Car is a subclass:

class Vehicle {

// Super class attributes and methods

}

class Car extends Vehicle {

// Subclass attributes and methods

}

Copy

The subclass can access the members (fields, methods, and nested classes) of the super class using the **super** keyword[3](https://www.geeksforgeeks.org/super-keyword/)[2](https://careerkarma.com/blog/java-super/). The super keyword can also be used to call the constructor of the super class from the subclass[3](https://www.geeksforgeeks.org/super-keyword/)[2](https://careerkarma.com/blog/java-super/).

For example, in the following code snippet, the Car subclass uses the super keyword to access the brand attribute and the honk method of the Vehicle super class, and to call the Vehicle constructor:

class Vehicle {

protected String brand = "Ford"; // Super class attribute

public void honk() { // Super class method

System.out.println("Tuut, tuut!");

}

}

class Car extends Vehicle {

private String modelName = "Mustang"; // Subclass attribute

public Car() { // Subclass constructor

super(); // Call the super class constructor

System.out.println(super.brand + " " + this.modelName); // Access the super class attribute

super.honk(); // Call the super class method

}

}

3.Can you overload a final method in Java?

[No, you cannot overload a final method in Java1](https://stackoverflow.com/questions/1988077/is-it-possible-to-overload-a-final-method). [A final method is a method that cannot be overridden or hidden by subclasses1](https://stackoverflow.com/questions/1988077/is-it-possible-to-overload-a-final-method). [Overloading a method means defining another method with the same name but different parameters2](https://www.geeksforgeeks.org/can-we-override-final-method-in-java/). [If you try to overload a final method, you will get a compile-time error1](https://stackoverflow.com/questions/1988077/is-it-possible-to-overload-a-final-method)[2](https://www.geeksforgeeks.org/can-we-override-final-method-in-java/).

[The reason why you cannot overload a final method is that final methods follow static binding (early binding), which means that the method definition will be grouped with a body at compile-time itself1](https://stackoverflow.com/questions/1988077/is-it-possible-to-overload-a-final-method). In other words, JVM must know exactly which method to call at compile-time. [To accomplish this, for every final method definition there must be a unique body1](https://stackoverflow.com/questions/1988077/is-it-possible-to-overload-a-final-method).

[However, you can overload the finalize() method, which is a special method that is invoked by the garbage collector before an object is destroyed3](https://stackoverflow.com/questions/29773942/can-finalize-method-be-overloaded-in-java). [The finalize() method can be overloaded without any issue, but it won’t be called by the garbage collector and JVM as they will call the one with no parameters](https://stackoverflow.com/questions/29773942/can-finalize-method-be-overloaded-in-java)

4.Can a method be abstract and final?

[No, a method cannot be abstract and final in Java1](https://www.java67.com/2017/07/can-you-make-abstract-class-method-final-in-java.html). [This is because abstract and final are mutually exclusive concepts1](https://www.java67.com/2017/07/can-you-make-abstract-class-method-final-in-java.html). An abstract method is a method that is declared without an implementation and must be overridden by subclasses[2](https://www.geeksforgeeks.org/difference-between-final-and-abstract-in-java/). [A final method is a method that cannot be overridden or hidden by subclasses2](https://www.geeksforgeeks.org/difference-between-final-and-abstract-in-java/).

[If you try to declare a method as abstract and final, you will get a compile-time error1](https://www.java67.com/2017/07/can-you-make-abstract-class-method-final-in-java.html)[2](https://www.geeksforgeeks.org/difference-between-final-and-abstract-in-java/). For example, the following code snippet will cause an error:

abstract class A {

public abstract final void methodOne(); // Error: illegal combination of modifiers

}

[However, an abstract class can have final methods in Java, but the final methods cannot be abstract1](https://www.java67.com/2017/07/can-you-make-abstract-class-method-final-in-java.html). For example, the following code snippet is valid:

abstract class A {

public final void methodOne() {

// Some implementation

}

}

5.Write the syntax for creating the subclass of a class?

The syntax for creating the subclass of a class in Java is:

class Subclass extends Superclass {

// Subclass fields and methods

}

The keyword **extends** is used to indicate that the subclass inherits from the superclass. The subclass can access the members of the superclass using the **super** keyword. The subclass can also override the methods of the superclass by providing a new implementation with the same signature.

For example, the following code snippet shows how to create a subclass called Dog that inherits from a superclass called Animal:

class Animal {

// Superclass fields and methods

String name;

int age;

public Animal(String name, int age) {

this.name = name;

this.age = age;

}

public void makeSound() {

System.out.println("Animal sound");

}

}

class Dog extends Animal {

// Subclass fields and methods

String breed;

public Dog(String name, int age, String breed) {

super(name, age); // Calling the superclass constructor

this.breed = breed;

}

@Override // Indicating that this method overrides the superclass method

public void makeSound() {

System.out.println("Woof woof"); // Providing a new implementation

}

}

6.Write a program to get the personal details of students such as name, register number, age and department from the base class Student and display those information in a class named Student Display.

// A base class to represent a student

class Student {

// Fields to store the personal details of a student

String name;

int registerNumber;

int age;

String department;

// A constructor to initialize the fields

public Student(String name, int registerNumber, int age, String department) {

this.name = name;

this.registerNumber = registerNumber;

this.age = age;

this.department = department;

}

}

// A class to display the personal details of a student

class StudentDisplay {

// A method to display the personal details of a student

public void display(Student s) {

System.out.println("Name: " + s.name);

System.out.println("Register Number: " + s.registerNumber);

System.out.println("Age: " + s.age);

System.out.println("Department: " + s.department);

}

}

// A class to test the program

public class Main {

public static void main(String[] args) {

// Create a student object using the Student constructor

Student s1 = new Student("Raj", 101, 19, "CSE");

// Create a student display object

StudentDisplay sd = new StudentDisplay();

// Display the personal details of the student using the display method

sd.display(s1);

}

}