# Introduction of Inheritance

### **Assisted Problems**

1. **Animal Hierarchy**
   * **Description**: Create a hierarchy where Animal is the superclass, and Dog, Cat, and Bird are subclasses. Each subclass has a unique behavior.
   * **Tasks**:
     + Define a superclass Animal with attributes name and age, and a method makeSound().
     + Define subclasses Dog, Cat, and Bird, each with a unique implementation of makeSound().
   * **Goal**: Learn basic inheritance, method overriding, and polymorphism with simple classes.

**Soln**:

import java.util.Scanner;

class Animal {

String name;

int age;

Animal(String name, int age) {

this.name = name;

this.age = age;

}

void makeSound() {

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

}

}

class Dog extends Animal {

Dog(String name, int age) { super(name, age); }

void makeSound() { System.out.println(name + " barks."); }

}

class Cat extends Animal {

Cat(String name, int age) { super(name, age); }

void makeSound() { System.out.println(name + " meows."); }

}

class Bird extends Animal {

Bird(String name, int age) { super(name, age); }

void makeSound() { System.out.println(name + " chirps."); }

}

public class AnimalHierarchy {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter animal type (Dog/Cat/Bird): ");

String type = sc.nextLine();

System.out.print("Enter name: ");

String name = sc.nextLine();

System.out.print("Enter age: ");

int age = sc.nextInt();

Animal animal;

switch (type.toLowerCase()) {

case "dog": animal = new Dog(name, age); break;

case "cat": animal = new Cat(name, age); break;

case "bird": animal = new Bird(name, age); break;

default: animal = new Animal(name, age);

}

animal.makeSound();

}

}

1. **Employee Management System**
   * **Description**: Create an Employee hierarchy for different employee types such as Manager, Developer, and Intern.
   * **Tasks**:
     + Define a base class Employee with attributes like name, id, and salary, and a method displayDetails().
     + Define subclasses Manager, Developer, and Intern with unique attributes for each, like teamSize for Manager and programmingLanguage for Developer.
   * **Goal**: Practice inheritance by creating subclasses with specific attributes and overriding superclass methods.

**Soln**:

import java.util.Scanner;

class Employee {

String name;

int id;

double salary;

Employee(String name, int id, double salary) {

this.name = name;

this.id = id;

this.salary = salary;

}

void displayDetails() {

System.out.println("Name: " + name + ", ID: " + id + ", Salary: " + salary);

}

}

class Manager extends Employee {

int teamSize;

Manager(String name, int id, double salary, int teamSize) {

super(name, id, salary);

this.teamSize = teamSize;

}

void displayDetails() {

super.displayDetails();

System.out.println("Team Size: " + teamSize);

}

}

class Developer extends Employee {

String programmingLanguage;

Developer(String name, int id, double salary, String lang) {

super(name, id, salary);

this.programmingLanguage = lang;

}

void displayDetails() {

super.displayDetails();

System.out.println("Language: " + programmingLanguage);

}

}

class Intern extends Employee {

int duration;

Intern(String name, int id, double salary, int duration) {

super(name, id, salary);

this.duration = duration;

}

void displayDetails() {

super.displayDetails();

System.out.println("Internship Duration: " + duration + " months");

}

}

public class EmployeeSystem {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter employee type (Manager/Developer/Intern): ");

String type = sc.nextLine();

System.out.print("Enter name: ");

String name = sc.nextLine();

System.out.print("Enter ID: ");

int id = sc.nextInt();

System.out.print("Enter salary: ");

double salary = sc.nextDouble();

sc.nextLine();

Employee e;

switch (type.toLowerCase()) {

case "manager":

System.out.print("Enter team size: ");

int size = sc.nextInt();

e = new Manager(name, id, salary, size); break;

case "developer":

System.out.print("Enter programming language: ");

String lang = sc.nextLine();

e = new Developer(name, id, salary, lang); break;

case "intern":

System.out.print("Enter duration (months): ");

int dur = sc.nextInt();

e = new Intern(name, id, salary, dur); break;

default:

e = new Employee(name, id, salary);

}

e.displayDetails();

}

}

1. **Vehicle and Transport System**
   * **Description**: Design a vehicle hierarchy where Vehicle is the superclass, and Car, Truck, and Motorcycle are subclasses with unique attributes.
   * **Tasks**:
     + Define a superclass Vehicle with maxSpeed and fuelType attributes and a method displayInfo().
     + Define subclasses Car, Truck, and Motorcycle, each with additional attributes, such as seatCapacity for Car.
     + Demonstrate polymorphism by storing objects of different subclasses in an array of Vehicle type and calling displayInfo() on each.
   * **Goal**: Understand how inheritance helps in organizing shared and unique features across subclasses and use polymorphism for dynamic method calls.

Soln:

import java.util.Scanner;

class Vehicle {

int maxSpeed;

String fuelType;

Vehicle(int maxSpeed, String fuelType) {

this.maxSpeed = maxSpeed;

this.fuelType = fuelType;

}

void displayInfo() {

System.out.println("Speed: " + maxSpeed + " km/h, Fuel: " + fuelType);

}

}

class Car extends Vehicle {

int seatCapacity;

Car(int maxSpeed, String fuelType, int seatCapacity) {

super(maxSpeed, fuelType);

this.seatCapacity = seatCapacity;

}

void displayInfo() {

super.displayInfo();

System.out.println("Seats: " + seatCapacity);

}

}

class Truck extends Vehicle {

int loadCapacity;

Truck(int maxSpeed, String fuelType, int loadCapacity) {

super(maxSpeed, fuelType);

this.loadCapacity = loadCapacity;

}

void displayInfo() {

super.displayInfo();

System.out.println("Load: " + loadCapacity + " kg");

}

}

class Motorcycle extends Vehicle {

boolean hasCarrier;

Motorcycle(int maxSpeed, String fuelType, boolean hasCarrier) {

super(maxSpeed, fuelType);

this.hasCarrier = hasCarrier;

}

void displayInfo() {

super.displayInfo();

System.out.println("Carrier: " + (hasCarrier ? "Yes" : "No"));

}

}

public class VehicleSystem {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Vehicle[] vehicles = new Vehicle[3];

System.out.println("Enter Car details:");

vehicles[0] = new Car(sc.nextInt(), sc.next(), sc.nextInt());

sc.nextLine();

System.out.println("Enter Truck details:");

vehicles[1] = new Truck(sc.nextInt(), sc.next(), sc.nextInt());

sc.nextLine();

System.out.println("Enter Motorcycle details:");

vehicles[2] = new Motorcycle(sc.nextInt(), sc.next(), sc.nextBoolean());

for (Vehicle v : vehicles) {

v.displayInfo();

System.out.println("-----------");

}

}

}

## Single Inheritance

### **Sample Problem 1: Library Management with Books and Authors**

* + **Description**: Model a Book system where Book is the superclass, and Author is a subclass.
  + **Tasks**:
    - Define a superclass Book with attributes like title and publicationYear.
    - Define a subclass Author with additional attributes like name and bio.
    - Create a method displayInfo() to show details of the book and its author.
  + **Goal**: Practice single inheritance by extending the base class and adding more specific details in the subclass.

Soln:

import java.util.Scanner;

class Book {

String title;

int publicationYear;

Book(String title, int year) {

this.title = title;

this.publicationYear = year;

}

void displayInfo() {

System.out.println("Title: " + title + ", Year: " + publicationYear);

}

}

class Author extends Book {

String name, bio;

Author(String title, int year, String name, String bio) {

super(title, year);

this.name = name;

this.bio = bio;

}

void displayInfo() {

super.displayInfo();

System.out.println("Author: " + name + ", Bio: " + bio);

}

}

public class LibrarySystem {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String title = sc.nextLine();

int year = sc.nextInt(); sc.nextLine();

String name = sc.nextLine();

String bio = sc.nextLine();

Author a = new Author(title, year, name, bio);

a.displayInfo();

}

}

**Sample Problem 2: Smart Home Devices**

* + **Description**: Create a hierarchy for a smart home system where Device is the superclass and Thermostat is a subclass.
  + **Tasks**:
    - Define a superclass Device with attributes like deviceId and status.
    - Create a subclass Thermostat with additional attributes like temperatureSetting.
    - Implement a method displayStatus() to show each device's current settings.
  + **Goal**: Understand single inheritance by adding specific attributes to a subclass, keeping the superclass general.

**Soln**:

import java.util.Scanner;

class Device {

String deviceId, status;

Device(String id, String status) {

this.deviceId = id;

this.status = status;

}

void displayStatus() {

System.out.println("Device ID: " + deviceId + ", Status: " + status);

}

}

class Thermostat extends Device {

int temperatureSetting;

Thermostat(String id, String status, int temp) {

super(id, status);

this.temperatureSetting = temp;

}

void displayStatus() {

super.displayStatus();

System.out.println("Temperature: " + temperatureSetting + "°C");

}

}

public class SmartHome {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Thermostat t = new Thermostat(sc.nextLine(), sc.nextLine(), sc.nextInt());

t.displayStatus();

}

}

## Multilevel Inheritance

**Sample Problem 1: Online Retail Order Management**

* + **Description**: Create a multilevel hierarchy to manage orders, where Order is the base class, ShippedOrder is a subclass, and DeliveredOrder extends ShippedOrder.
  + **Tasks**:
    - Define a base class Order with common attributes like orderId and orderDate.
    - Create a subclass ShippedOrder with additional attributes like trackingNumber.
    - Create another subclass DeliveredOrder extending ShippedOrder, adding a deliveryDate attribute.
    - Implement a method getOrderStatus() to return the current order status based on the class level.
  + **Goal**: Explore multilevel inheritance, showing how attributes and methods can be added across a chain of classes.

**Soln**:

import java.util.Scanner;

class Order {

String orderId, orderDate;

Order(String id, String date) {

orderId = id;

orderDate = date;

}

}

class ShippedOrder extends Order {

String trackingNumber;

ShippedOrder(String id, String date, String track) {

super(id, date);

trackingNumber = track;

}

}

class DeliveredOrder extends ShippedOrder {

String deliveryDate;

DeliveredOrder(String id, String date, String track, String deliveryDate) {

super(id, date, track);

this.deliveryDate = deliveryDate;

}

void getOrderStatus() {

System.out.println("Order ID: " + orderId + " delivered on " + deliveryDate);

}

}

public class RetailSystem {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

DeliveredOrder d = new DeliveredOrder(sc.nextLine(), sc.nextLine(), sc.nextLine(), sc.nextLine());

d.getOrderStatus();

}

}

**Sample Problem 2: Educational Course Hierarchy**

* + **Description**: Model a course system where Course is the base class, OnlineCourse is a subclass, and PaidOnlineCourse extends OnlineCourse.
  + **Tasks**:
    - Define a superclass Course with attributes like courseName and duration.
    - Define OnlineCourse to add attributes such as platform and isRecorded.
    - Define PaidOnlineCourse to add fee and discount.
  + **Goal**: Demonstrate how each level of inheritance builds on the previous, adding complexity to the system.

**Soln**:

import java.util.Scanner;

class Course {

String courseName;

int duration;

Course(String name, int duration) {

courseName = name;

this.duration = duration;

}

}

class OnlineCourse extends Course {

String platform;

boolean isRecorded;

OnlineCourse(String name, int duration, String platform, boolean recorded) {

super(name, duration);

this.platform = platform;

this.isRecorded = recorded;

}

}

class PaidOnlineCourse extends OnlineCourse {

double fee;

double discount;

PaidOnlineCourse(String name, int dur, String plat, boolean rec, double fee, double disc) {

super(name, dur, plat, rec);

this.fee = fee;

this.discount = disc;

}

void showCourseDetails() {

System.out.println("Course: " + courseName + ", Duration: " + duration + " hrs");

System.out.println("Platform: " + platform + ", Recorded: " + isRecorded);

System.out.println("Fee: " + fee + ", Discount: " + discount + "%");

}

}

public class CourseSystem {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

PaidOnlineCourse c = new PaidOnlineCourse(sc.nextLine(), sc.nextInt(), sc.next(), sc.nextBoolean(), sc.nextDouble(), sc.nextDouble());

c.showCourseDetails();

}

}

## Hierarchical Inheritance

**Sample Problem 1: Bank Account Types**

* + **Description**: Model a banking system with different account types using hierarchical inheritance. BankAccount is the superclass, with SavingsAccount, CheckingAccount, and FixedDepositAccount as subclasses.
  + **Tasks**:
    - Define a base class BankAccount with attributes like accountNumber and balance.
    - Define subclasses SavingsAccount, CheckingAccount, and FixedDepositAccount, each with unique attributes like interestRate for SavingsAccount and withdrawalLimit for CheckingAccount.
    - Implement a method displayAccountType() in each subclass to specify the account type.
  + **Goal**: Explore hierarchical inheritance, demonstrating how each subclass can have unique attributes while inheriting from a shared superclass.

**Soln**:

import java.util.Scanner;

class BankAccount {

String accountNumber;

double balance;

BankAccount(String acc, double bal) {

accountNumber = acc;

balance = bal;

}

}

class SavingsAccount extends BankAccount {

double interestRate;

SavingsAccount(String acc, double bal, double rate) {

super(acc, bal);

interestRate = rate;

}

void displayAccountType() {

System.out.println("Savings Account - Acc: " + accountNumber + ", Balance: " + balance + ", Interest: " + interestRate + "%");

}

}

class CheckingAccount extends BankAccount {

double withdrawalLimit;

CheckingAccount(String acc, double bal, double limit) {

super(acc, bal);

withdrawalLimit = limit;

}

void displayAccountType() {

System.out.println("Checking Account - Acc: " + accountNumber + ", Balance: " + balance + ", Limit: " + withdrawalLimit);

}

}

class FixedDepositAccount extends BankAccount {

int termMonths;

FixedDepositAccount(String acc, double bal, int term) {

super(acc, bal);

termMonths = term;

}

void displayAccountType() {

System.out.println("Fixed Deposit - Acc: " + accountNumber + ", Balance: " + balance + ", Term: " + termMonths + " months");

}

}

public class BankSystem {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

SavingsAccount sa = new SavingsAccount(sc.next(), sc.nextDouble(), sc.nextDouble());

CheckingAccount ca = new CheckingAccount(sc.next(), sc.nextDouble(), sc.nextDouble());

FixedDepositAccount fa = new FixedDepositAccount(sc.next(), sc.nextDouble(), sc.nextInt());

sa.displayAccountType();

ca.displayAccountType();

fa.displayAccountType();

}

}

**Sample Problem 2: School System with Different Roles**

* + **Description**: Create a hierarchy for a school system where Person is the superclass, and Teacher, Student, and Staff are subclasses.
  + **Tasks**:
    - Define a superclass Person with common attributes like name and age.
    - Define subclasses Teacher, Student, and Staff with specific attributes (e.g., subject for Teacher and grade for Student).
    - Each subclass should have a method like displayRole() that describes the role.
  + **Goal**: Demonstrate hierarchical inheritance by modeling different roles in a school, each with shared and unique characteristics.

**Soln**:

import java.util.Scanner;

class Person {

String name;

int age;

Person(String name, int age) {

this.name = name;

this.age = age;

}

}

class Teacher extends Person {

String subject;

Teacher(String name, int age, String subject) {

super(name, age);

this.subject = subject;

}

void displayRole() {

System.out.println("Teacher: " + name + ", Age: " + age + ", Subject: " + subject);

}

}

class Student extends Person {

int grade;

Student(String name, int age, int grade) {

super(name, age);

this.grade = grade;

}

void displayRole() {

System.out.println("Student: " + name + ", Age: " + age + ", Grade: " + grade);

}

}

class Staff extends Person {

String department;

Staff(String name, int age, String dept) {

super(name, age);

this.department = dept;

}

void displayRole() {

System.out.println("Staff: " + name + ", Age: " + age + ", Dept: " + department);

}

}

public class SchoolSystem {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Teacher t = new Teacher(sc.nextLine(), sc.nextInt(), sc.next()); sc.nextLine();

Student s = new Student(sc.nextLine(), sc.nextInt(), sc.nextInt()); sc.nextLine();

Staff st = new Staff(sc.nextLine(), sc.nextInt(), sc.next());

t.displayRole();

s.displayRole();

st.displayRole();

}

}

## Hybrid Inheritance (Simulating Multiple Inheritance)

Since Java doesn’t support multiple inheritance directly, hybrid inheritance is typically achieved through **interfaces**.

**Sample Problem 1: Restaurant Management System with Hybrid Inheritance**

* + **Description**: Model a restaurant system where Person is the superclass and Chef and Waiter are subclasses. Both Chef and Waiter should implement a Worker interface that requires a performDuties() method.
  + **Tasks**:
    - Define a superclass Person with attributes like name and id.
    - Create an interface Worker with a method performDuties().
    - Define subclasses Chef and Waiter that inherit from Person and implement the Worker interface, each providing a unique implementation of performDuties().
  + **Goal**: Practice hybrid inheritance by combining inheritance and interfaces, giving multiple behaviors to the same objects.

**Soln**:

import java.util.Scanner;

interface Worker {

void performDuties();

}

class Person {

String name;

int id;

Person(String name, int id) {

this.name = name;

this.id = id;

}

}

class Chef extends Person implements Worker {

Chef(String name, int id) {

super(name, id);

}

public void performDuties() {

System.out.println("Chef " + name + " cooks food.");

}

}

class Waiter extends Person implements Worker {

Waiter(String name, int id) {

super(name, id);

}

public void performDuties() {

System.out.println("Waiter " + name + " serves food.");

}

}

public class RestaurantSystem {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Chef c = new Chef(sc.nextLine(), sc.nextInt()); sc.nextLine();

Waiter w = new Waiter(sc.nextLine(), sc.nextInt());

c.performDuties();

w.performDuties();

}

}

**Sample Problem 2: Vehicle Management System with Hybrid Inheritance**

* + **Description**: Model a vehicle system where Vehicle is the superclass and ElectricVehicle and PetrolVehicle are subclasses. Additionally, create a Refuelable interface implemented by PetrolVehicle.
  + **Tasks**:
    - Define a superclass Vehicle with attributes like maxSpeed and model.
    - Create an interface Refuelable with a method refuel().
    - Define subclasses ElectricVehicle and PetrolVehicle. PetrolVehicle should implement Refuelable, while ElectricVehicle include a charge() method.
  + **Goal**: Use hybrid inheritance by having PetrolVehicle implement both Vehicle and Refuelable, demonstrating how Java interfaces allow adding multiple behaviors.

**Soln**:

import java.util.Scanner;

interface Refuelable {

void refuel();

}

class Vehicle {

int maxSpeed;

String model;

Vehicle(int maxSpeed, String model) {

this.maxSpeed = maxSpeed;

this.model = model;

}

}

class ElectricVehicle extends Vehicle {

ElectricVehicle(int maxSpeed, String model) {

super(maxSpeed, model);

}

void charge() {

System.out.println("Electric Vehicle " + model + " is charging.");

}

}

class PetrolVehicle extends Vehicle implements Refuelable {

PetrolVehicle(int maxSpeed, String model) {

super(maxSpeed, model);

}

public void refuel() {

System.out.println("Petrol Vehicle " + model + " is refueling.");

}

}

public class VehicleHybridSystem {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

ElectricVehicle ev = new ElectricVehicle(sc.nextInt(), sc.next()); sc.nextLine();

PetrolVehicle pv = new PetrolVehicle(sc.nextInt(), sc.next());

ev.charge();

pv.refuel();

}

}