JAVA PHA FILE

Module 2

**Q1. University – Department – Professor (One-to-Many Relationship)**

**Problem**

In real-world university systems, a **Department** manages several **Professors**, but each

professor belongs to exactly one department. Instead of keeping all details inside a single

class, we use separate entities for better modularity and scalability. This represents a

**One-to-Many relationship** (one department → many professors).

**Class Specifications:**

• Department class:

o Private attributes: deptName (String), hodName (String),

professors (List<Professor>)

o Constructors: default, parameterized (deptName, hodName)

o Getters/Setters for all fields

o Method: addProfessor(Professor p)

o Override toString() to return department + HOD details + professor

list

• Professor class:

o Private attributes: name (String), employeeId (String),

specialization (String)

o Constructors: default, parameterized (name, employeeId,

specialization)

o Getters/Setters

o Override toString() to return professor details

**Input format:**

Enter Department details (deptName,hodName)

Computer Science,Dr. Mehta

Enter number of professors

2

Enter professor details (name,employeeId,specialization)

Arjun,P101,AI

Neha,P102,ML

**Output format:**

Department: Computer Science

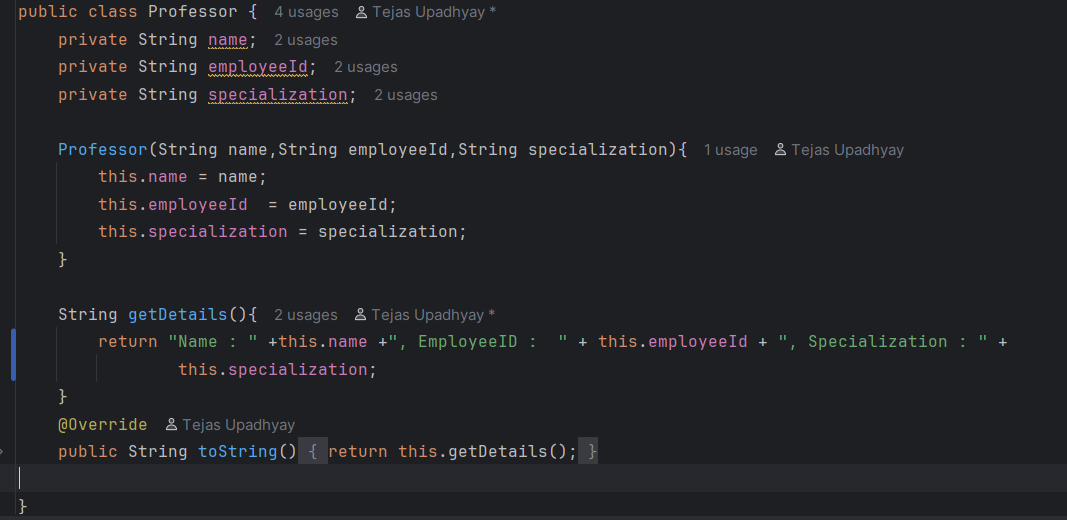
HOD: Dr. Mehta

Professors:

Name: Arjun, ID: P101, Specialization: AI

Name: Neha, ID: P102, Specialization: ML

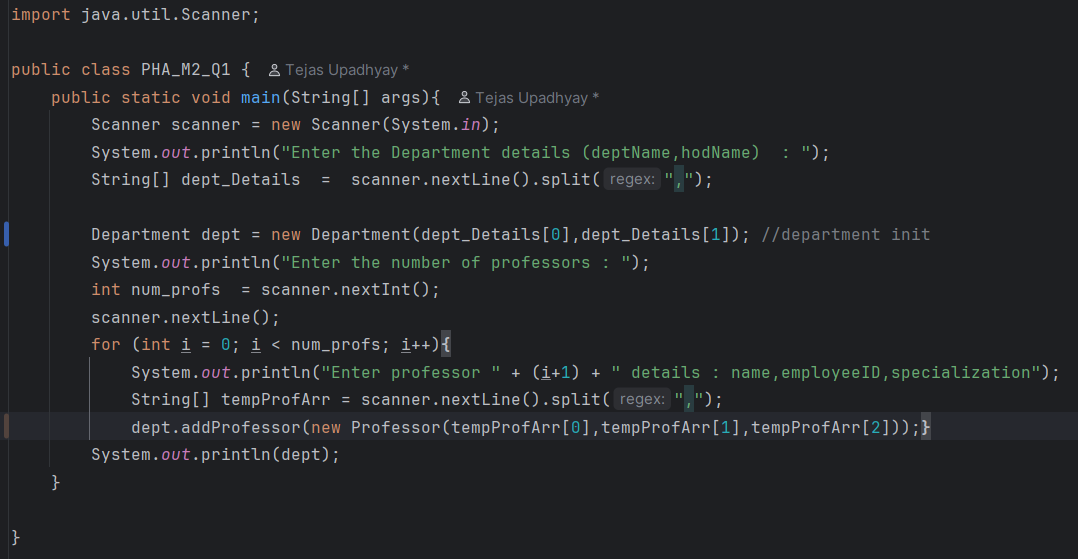
**Solution:**

**Professor Class:  
**

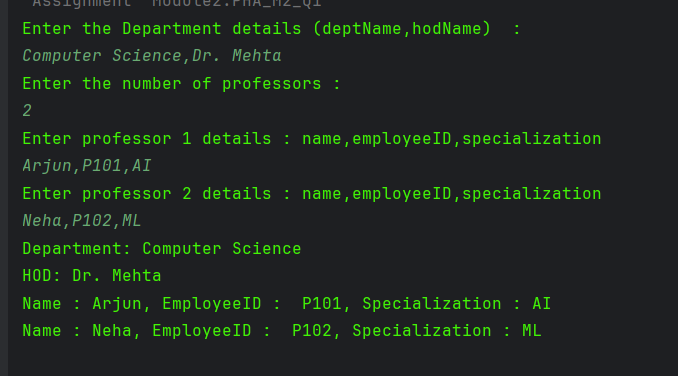
**Department Class:**

****

**Main Class:**

****

**Output**

****

**Q2. Flight Booking System (Abstraction + Inheritance)**

**Problem**

Airline systems have **different types of flights** — domestic and international. While both

share some common attributes (flight number, airline, fare), the **fare calculation rules**

**differ**. Using **abstraction**, we can define a generic Flight and implement details in

subclasses.

**Class Specifications:**

• Abstract Flight class:

o Private attributes: flightNumber (String), airline (String), fare

(double)

o Constructor (flightNumber, airline, fare)

o Abstract method: calculateFare()

o Override toString() → flight details with final fare

• DomesticFlight class extends Flight:

o Fare = base fare + 10% tax

• InternationalFlight class extends Flight:

o Fare = base fare + 25% tax

**Input format:**

Enter flight type,number,airline,fare

Domestic,AI202,Air India,5000

International,QF101,Qantas,20000

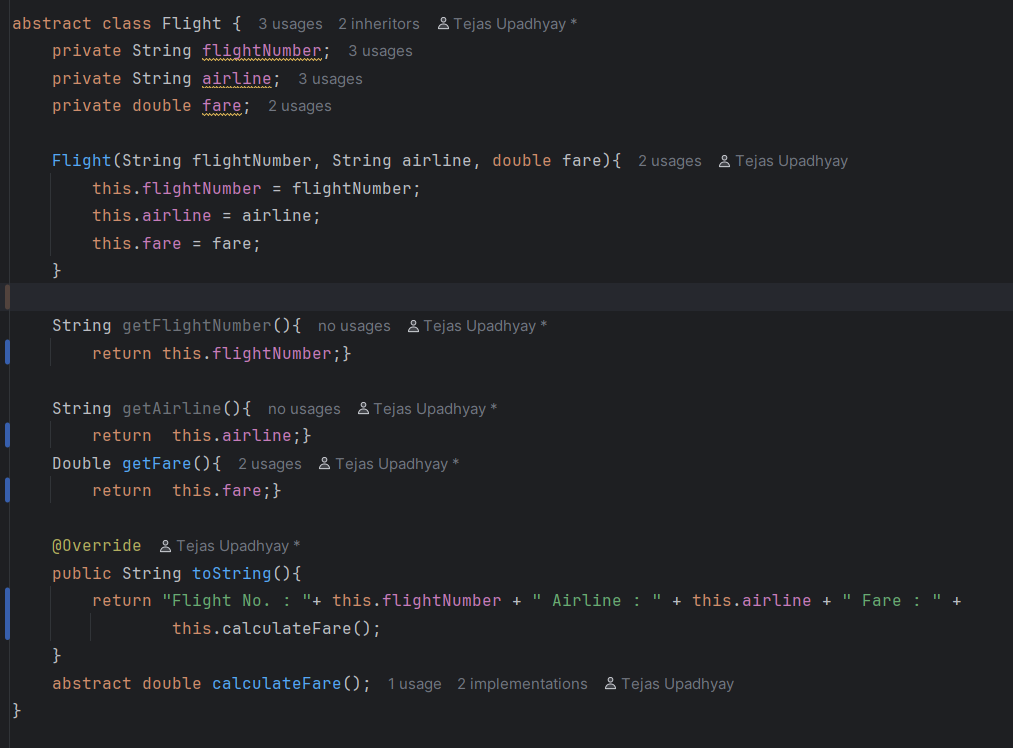
**Output format:**

Flight No: AI202 Airline: Air India Fare: 5500.0

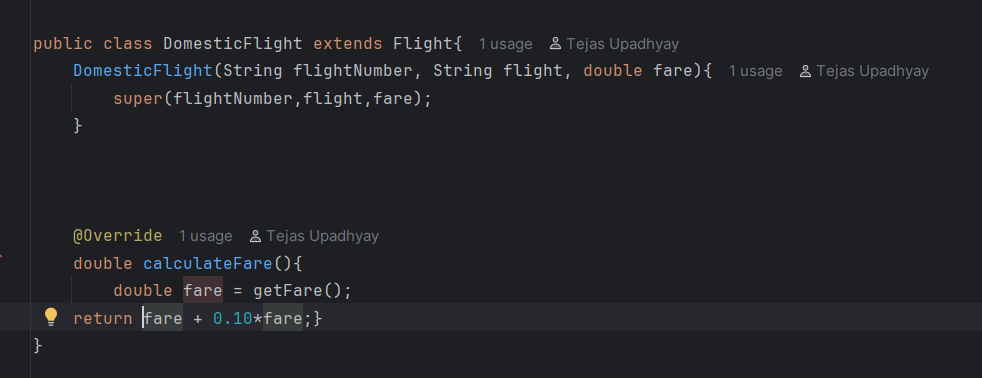
Flight No: QF101 Airline: Qantas Fare: 25000.0

**Solution**

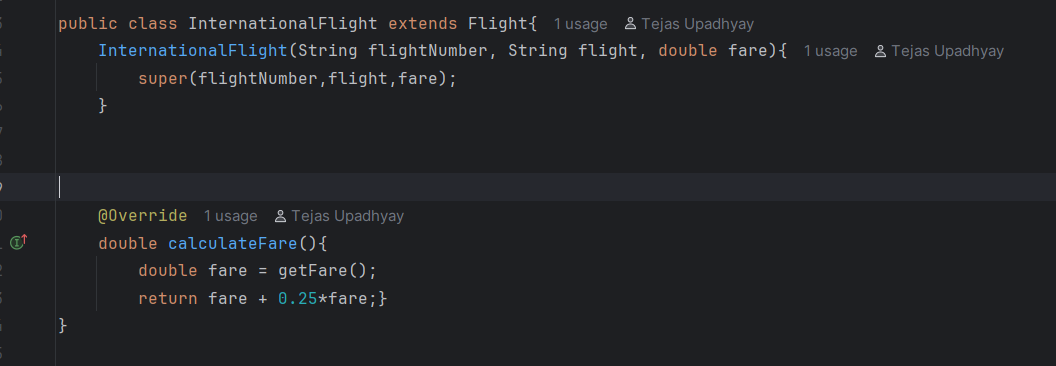
**Flight Class**

****

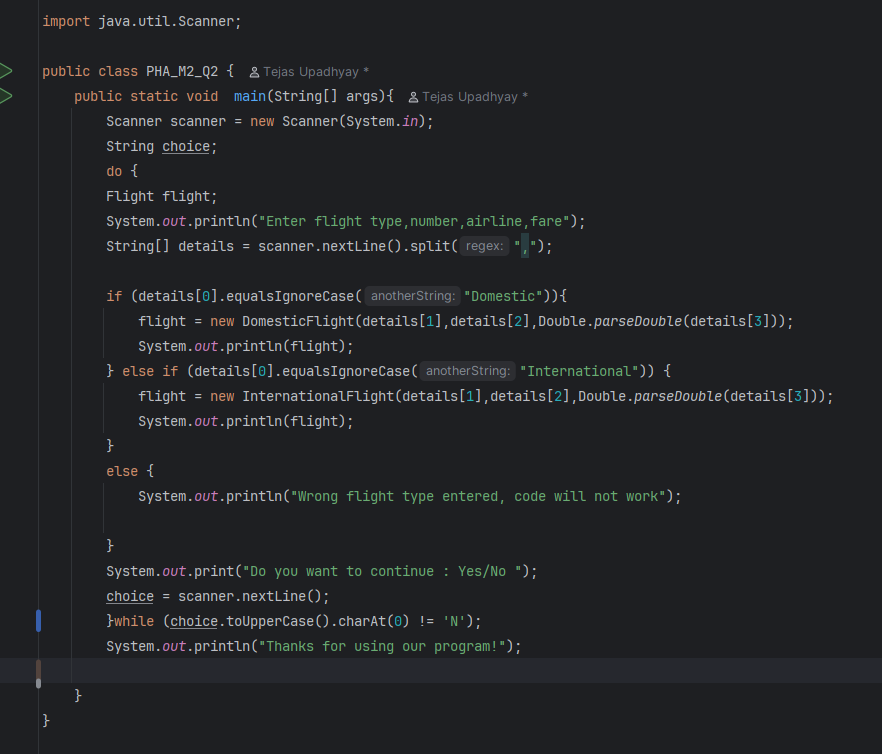
**Domestic Flight**

****

**InternationalFlight**

****

**Main Class**

****

**Output**

****

**Q3. Employee Payroll System (Constructor Chaining + Overriding)**

**Problem**

A company maintains **different types of employees**. A normal employee only has basic

salary, while managers earn an additional bonus. This requires **inheritance, constructor**

**chaining, and method overriding** for salary calculation.

**Class Specifications:**

• Employee class:

o Attributes: name, id, basicSalary

o Constructors: default, (name, id, basicSalary)

o Method: calculateSalary() → returns basicSalary

o Override toString() → employee details + salary

• Manager class extends Employee:

o Extra field: bonus

o Constructor calls super + initializes bonus

o Override calculateSalary() → salary = basicSalary + bonus

**Input format:**

Employee,Ravi,E101,30000

Manager,Seema,M202,40000,5000

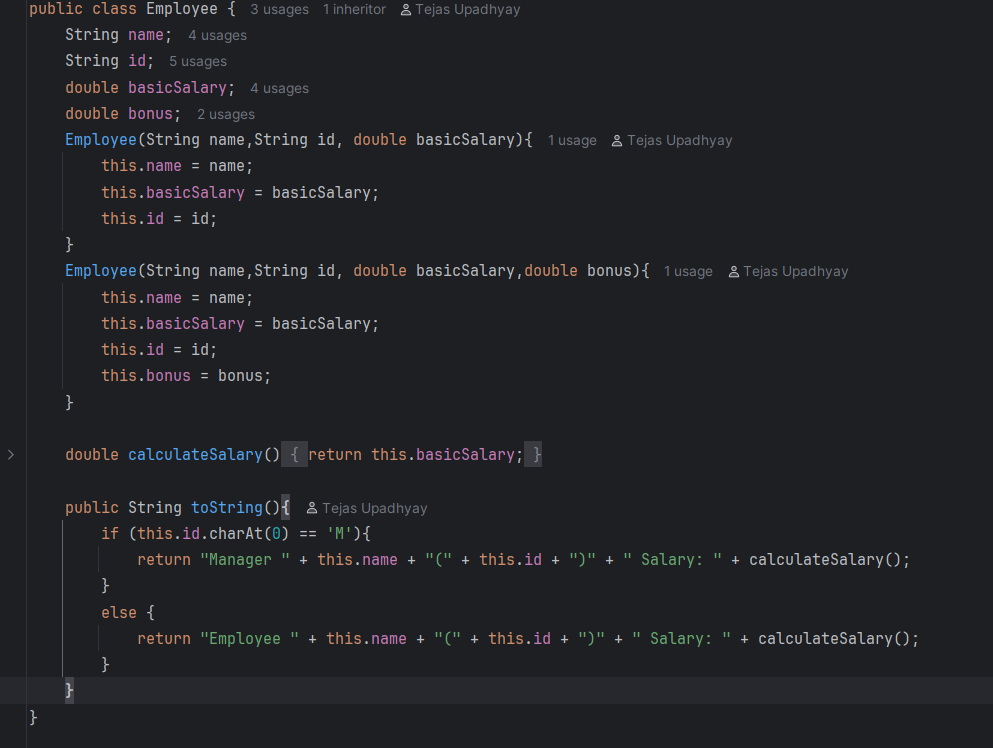
**Output format:**

Employee Ravi (E101) Salary: 30000.0

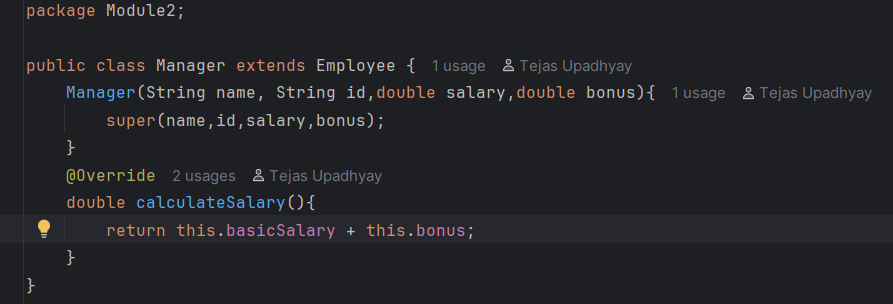
Manager Seema (M202) Salary: 45000.0

**Solution:**

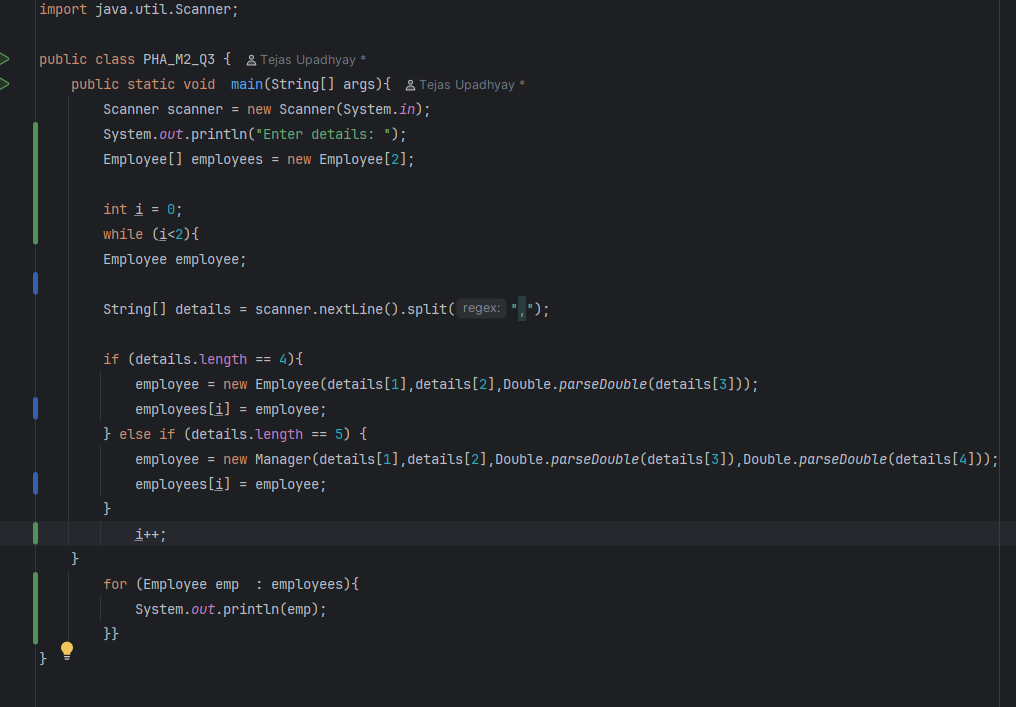
**Employee Class:**

****

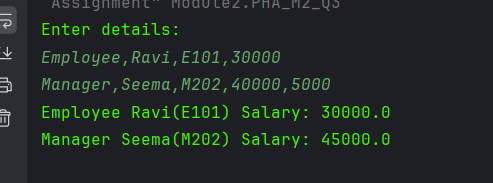
**Manager Class:**

****

**Main Class**

****

**Output**

****

**Q4. Online Shopping Cart (Aggregation)**

**Problem**

In e-commerce, an order contains multiple products. Instead of storing everything in one

class, we model Order and Product. This represents **aggregation** because Order

contains Products but both exist independently.

**Class Specifications:**

• Product class:

o Fields: productName, price, quantity

o Constructors + Getters/Setters

o toString() → productName x qty = total

• Order class:

o Fields: orderId, products (List<Product>)

o Method: calculateTotal()

o Override toString() to print order summary

**Input format:**

ORD101

3

Laptop,50000,1

Mouse,500,2

Keyboard,1500,1

**Output format:**

Order ID: ORD101

Products:

Laptop x1 = 50000

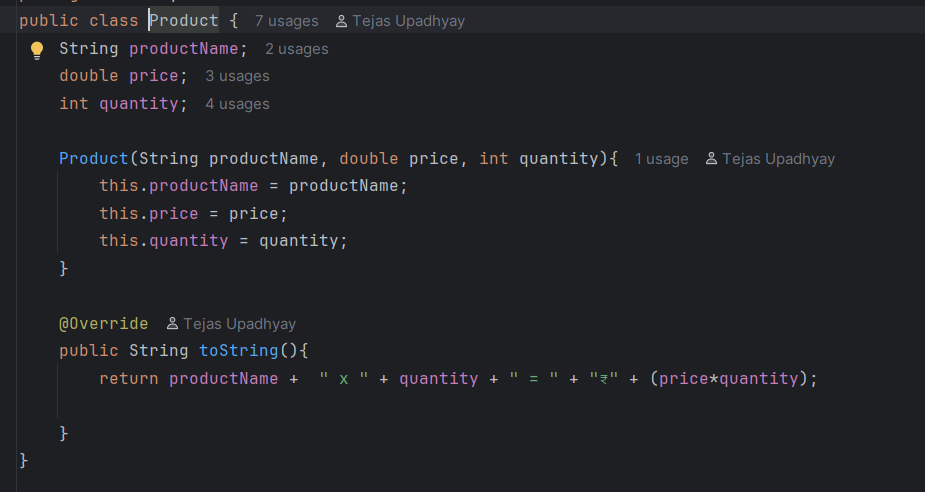
Mouse x2 = 1000

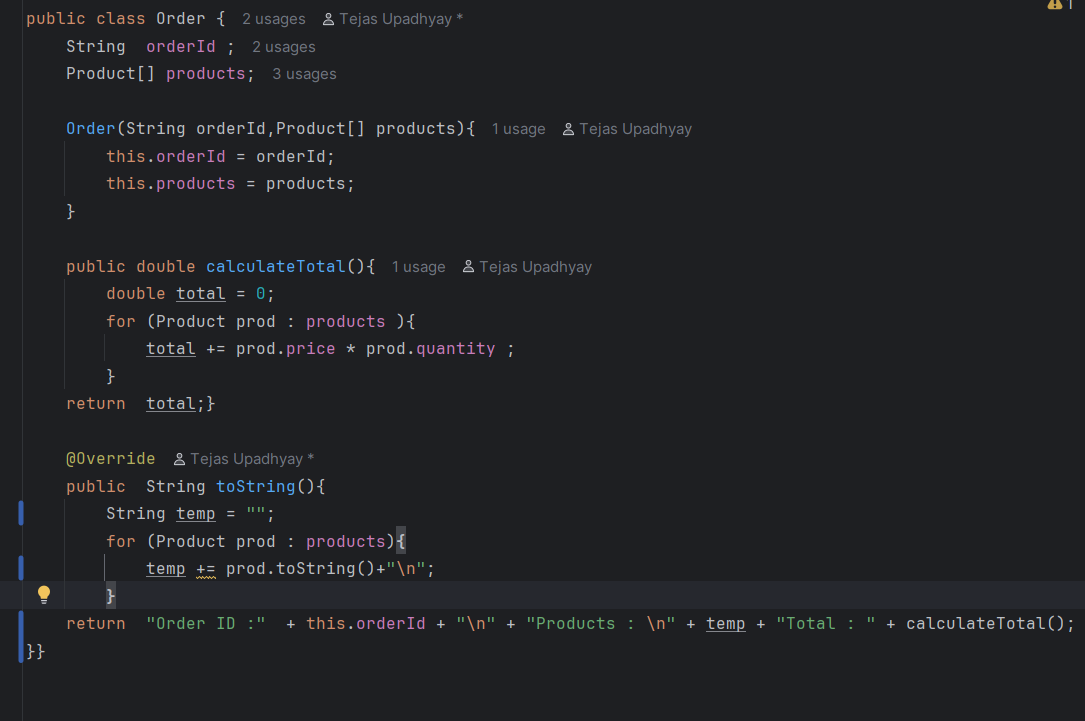
Keyboard x1 = 1500

Total: 52500

**Solution:**

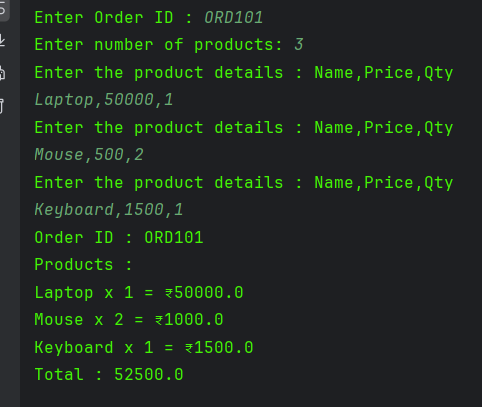
**Product Class:**

****

**Order Class:  
**

**Main Class:**

**Output:**

****

**Q5. Hospital Management (Multi-level Inheritance)**

**Problem**

A hospital tracks different roles. A Person can be a Doctor, and a doctor can

specialize as a Surgeon. This models a **multi-level inheritance** hierarchy.

**Class Specifications:**

• Person: name, age

• Doctor extends Person: specialization

• Surgeon extends Doctor: surgeryType

• Constructors + toString() chain

**Input format:**

John,40,Cardiology,Heart Surgery

**Output format:**

Name: John

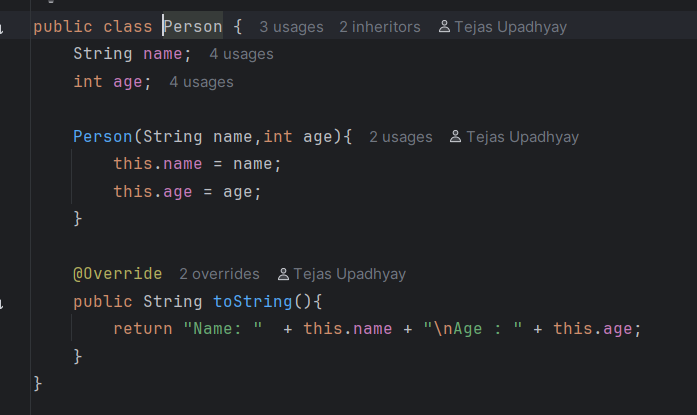
Age: 40

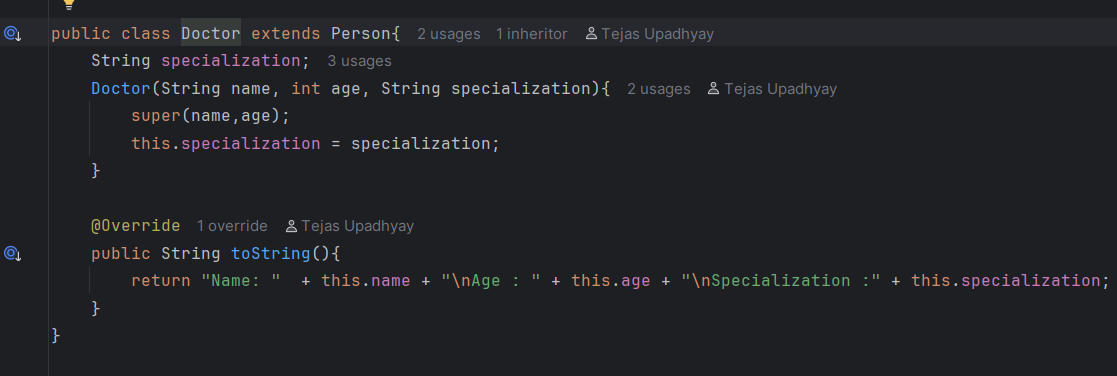
Specialization: Cardiology

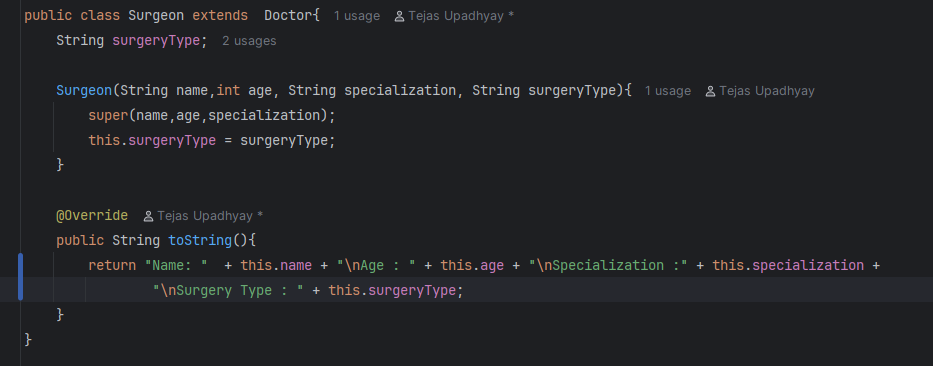
Surgery Type: Heart Surgery

**Solution:**

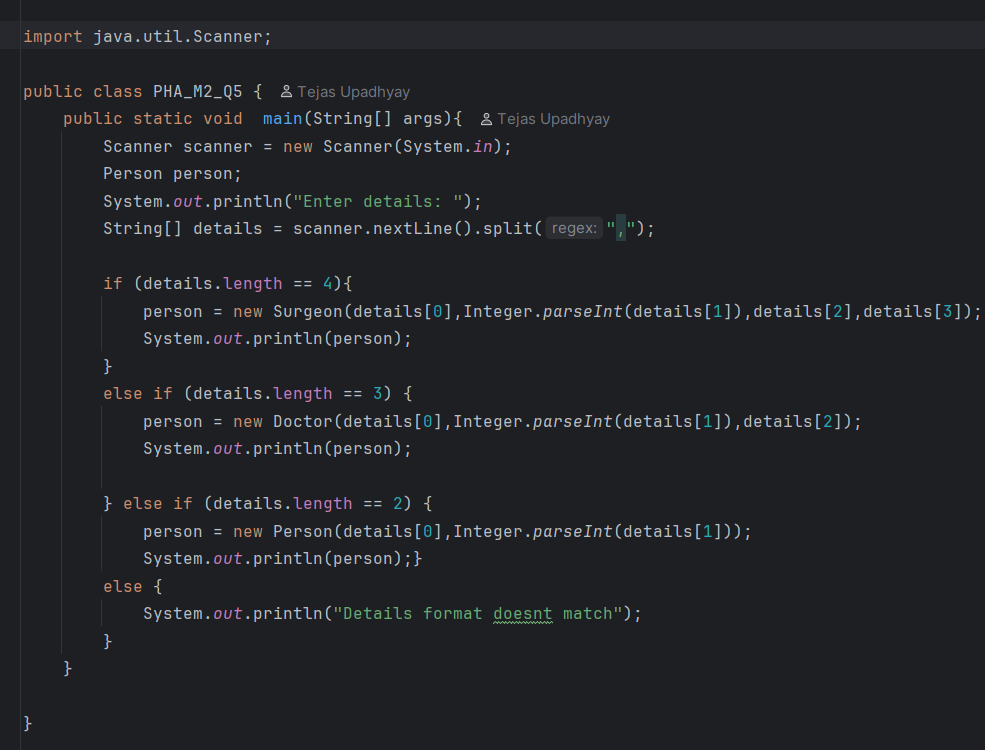
**Person Class:**

****

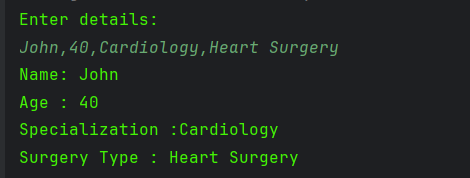
**Doctor Class:**

**Surgeon Class:**

**Main Class:**

****

**Output:**

****

**Q6. Library – Book – Author (One-to-One Relationship)**

**Problem**

Each book has a single author. We separate author details into another class for better

modularity. This forms a **one-to-one relationship**.

**Class Specifications:**

• Author class: name, email, gender

• Book class: title, price, author

• toString() in Book prints Author details via composition

**Input format:**

Effective Java,550,Joshua Bloch,jbloch@abc.com,M

**Output format:**

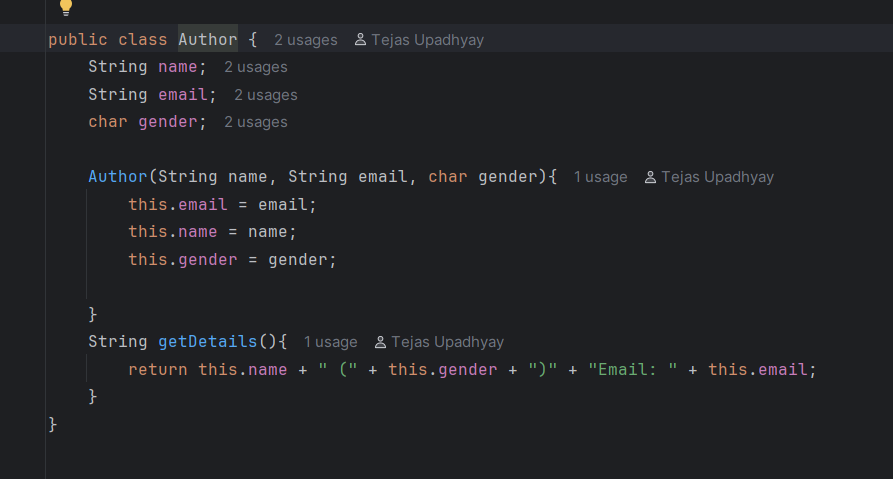
Book: Effective Java

Price: 550

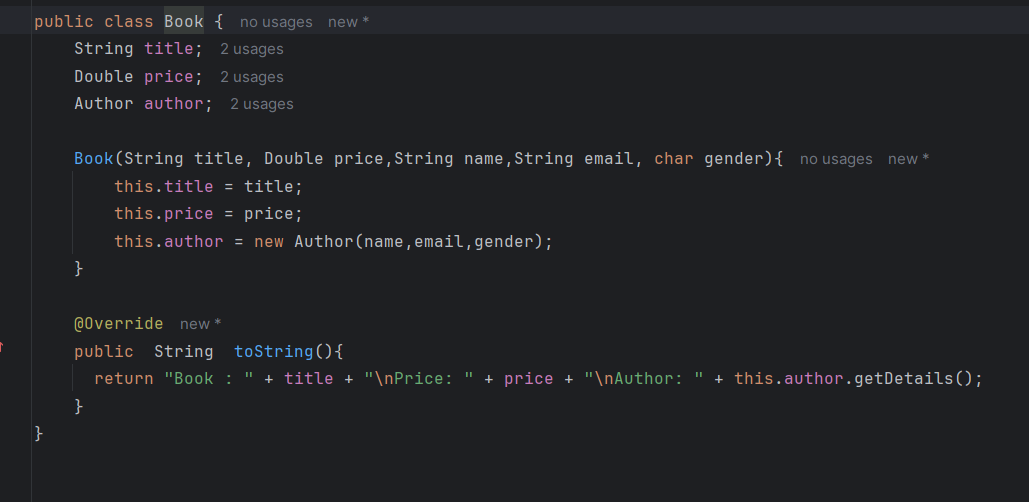
Author: Joshua Bloch (M), Email: [jbloch@abc.com](mailto:jbloch@abc.com)

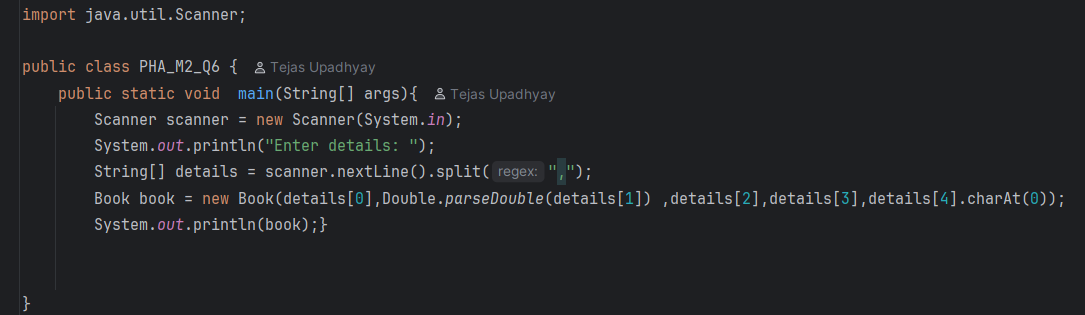
**Solution:**

**Author Class:**

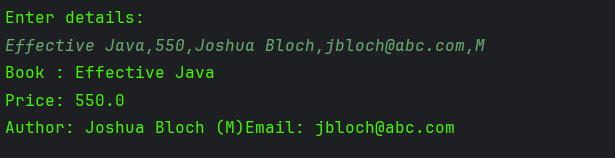
****

**Book Class:**

****

**Main Class:**

**Output:**

****

**Q7. Sports League (Polymorphism – Method Overriding)**

**Problem**

A sports league awards points differently depending on the sport. Cricket and Football

teams follow different rules. We use **inheritance + method overriding** to handle this

polymorphism.

**Class Specifications:**

• Team base class: name, matchesPlayed, wins, draws

• CricketTeam and FootballTeam override calculatePoints()

o Cricket: win=2, draw=1

o Football: win=3, draw=1

**Input:**

Cricket,India,10,6,2

Football,Barcelona,8,6,1

**Output:**

Team: India (Cricket) Points: 14

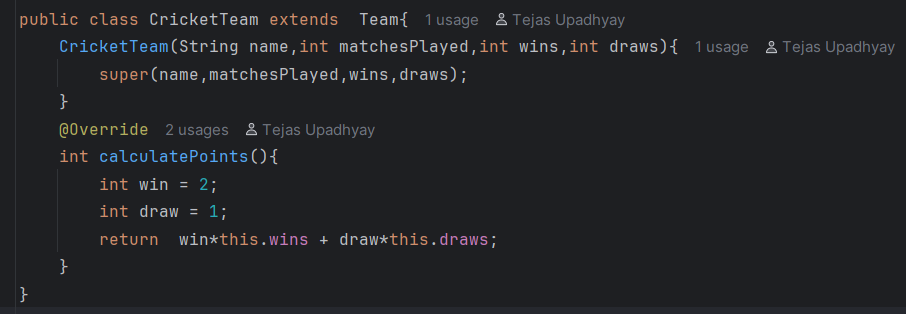
Team: Barcelona (Football) Points: 19

**Solution:**

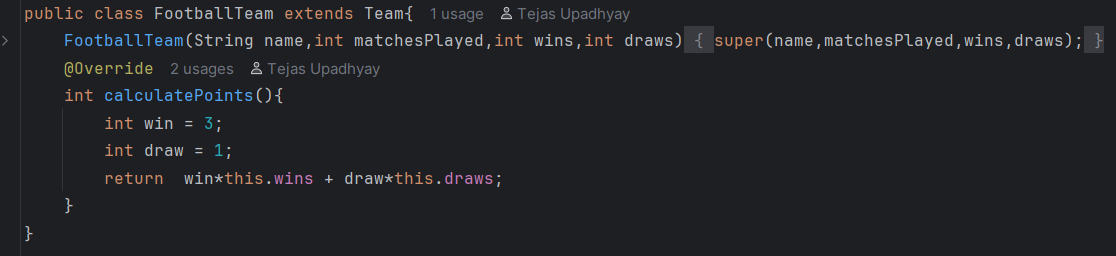
**Team Class:**

****

**Cricket Team Class:**

****

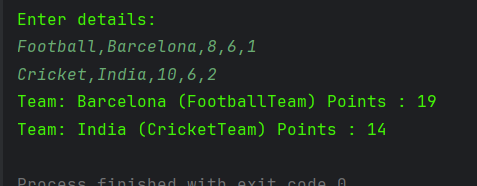
**Football Team Class:**

****

**Main Class:**

****

**Output**

****

**Q8. Loan Management System (Abstraction + Polymorphism)**

**Problem**

Banks offer multiple loan types with different interest rates. Using **abstraction**, define a

generic loan and calculate interest differently in subclasses.

**Class Specifications:**

• Abstract Loan: principal, rate, time, abstract calculateInterest()

• HomeLoan → 8%

• CarLoan → 10%

• Formula: SI = (P*R*T)/100

**Input format:**

Home,500000,8

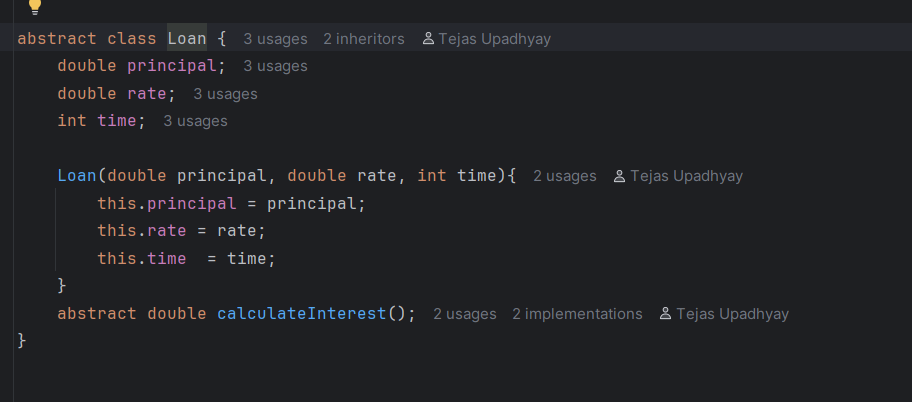
Car,300000,5

**Output:**

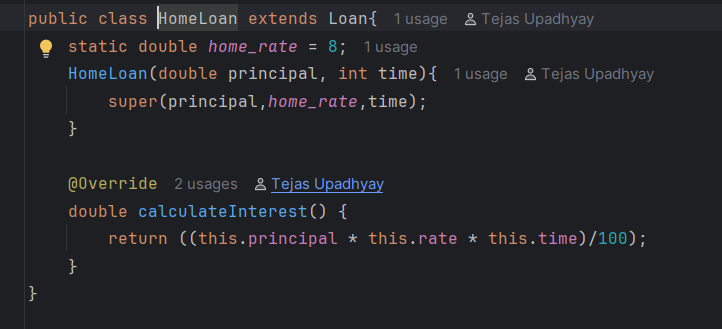
Home Loan Interest: 120000.0

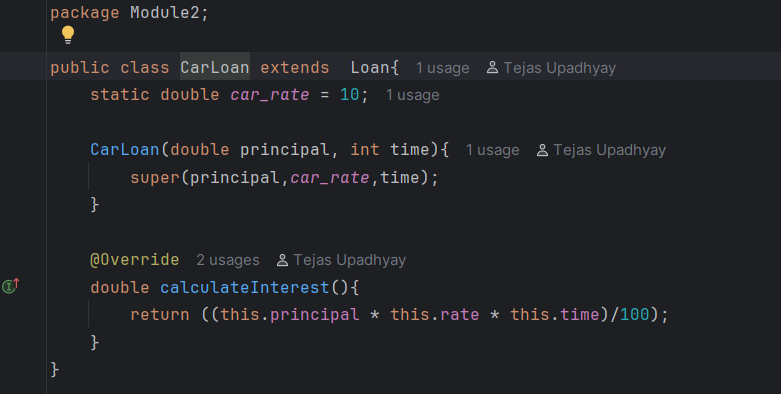
Car Loan Interest: 150000.0

**Solution**

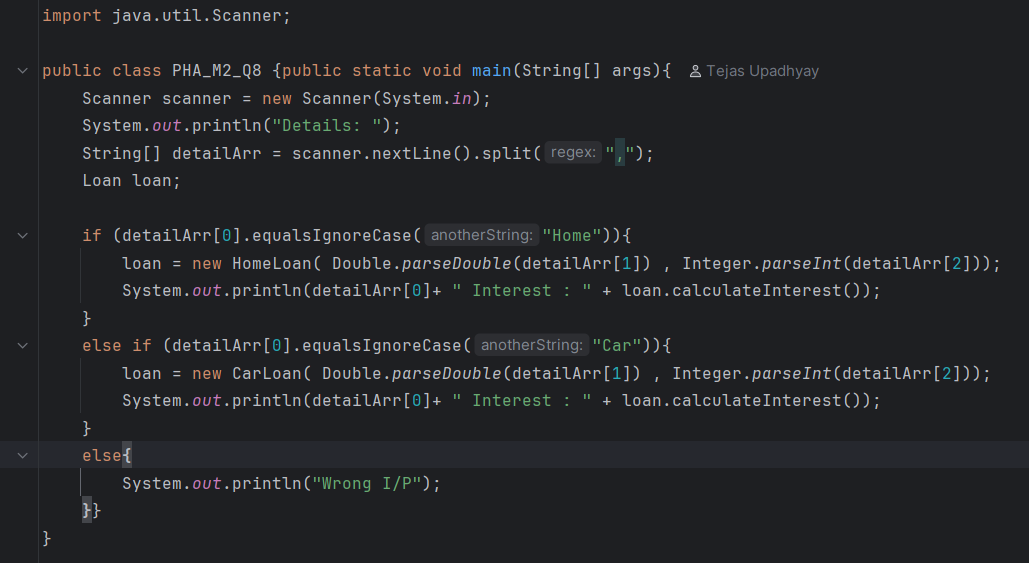
**Loan Class:  
**

**Home Loan Class:**

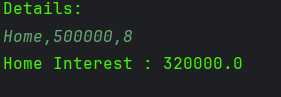
****

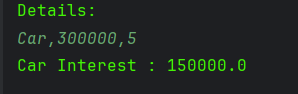
**Car Loan Class:  
**

**Main Class:**

****

**Output:**

****

****

**Q9. Online Course Platform (Association + Inheritance)**

**Problem**

An online platform sells courses. Students can enroll in normal or premium mode.

Premium students get extra discount. This uses **association + inheritance**.

**Class Specifications:**

• Course: courseName, duration

• Student: name, enrolledCourse (Course)

• PremiumStudent extends Student: discount

**Input:**

Java,3 months

Arjun,Java

Meena,Java,20

**Output:**

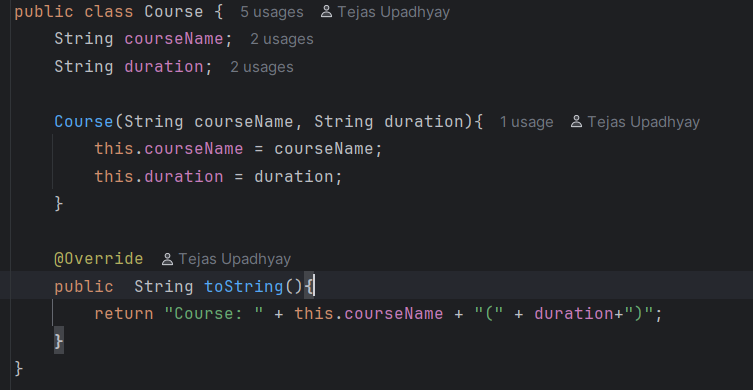
Student: Arjun Course: Java (3 months)

Premium Student: Meena Course: Java (3 months) Discount:

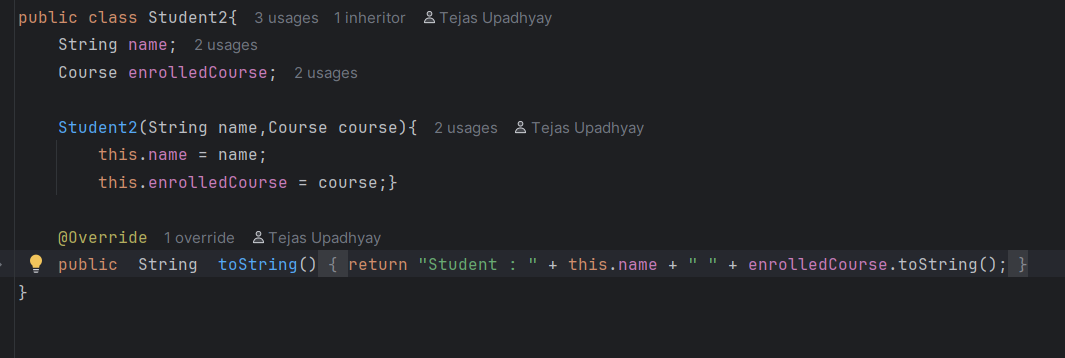
20%

**Solution:**

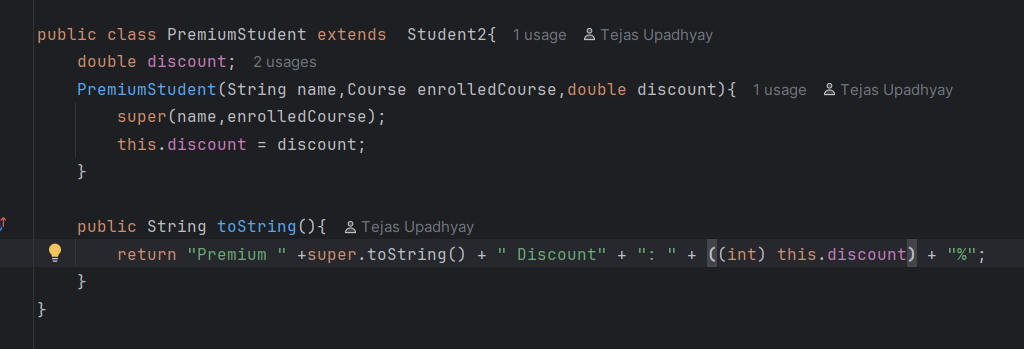
**Course Class:**

****

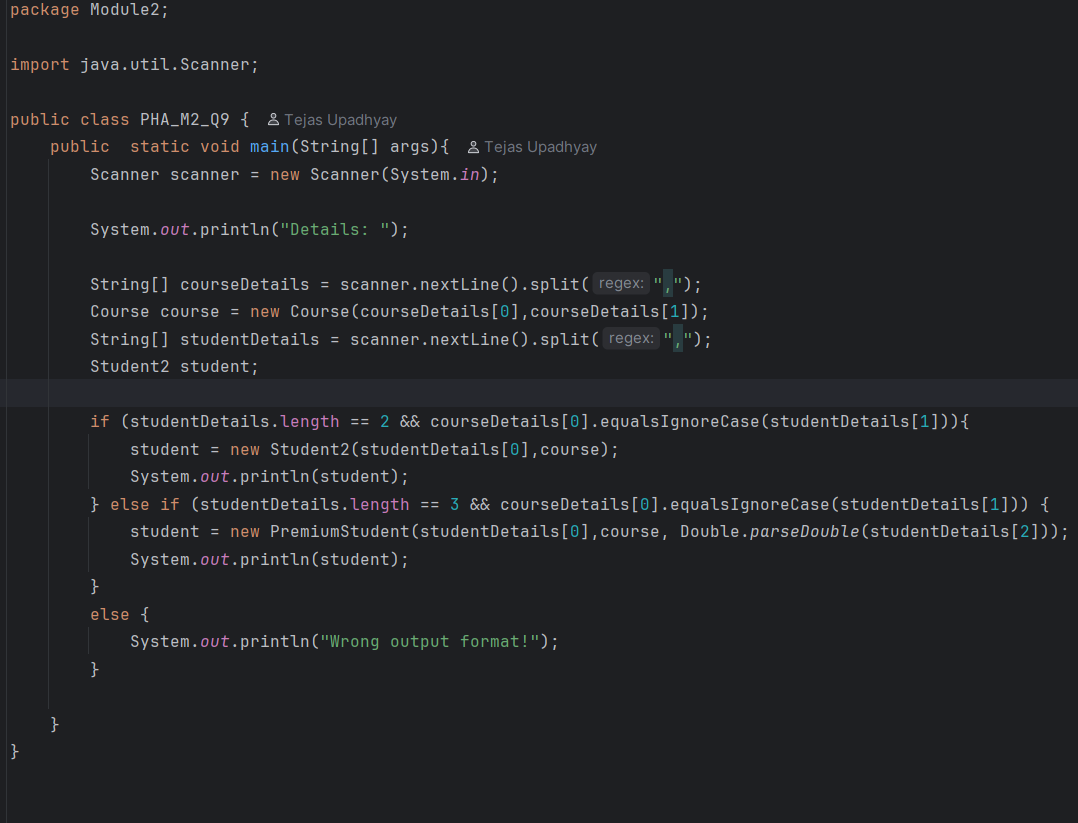
**Student Class:**

****

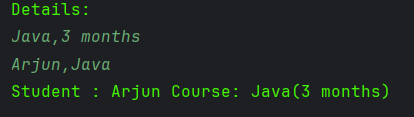
**Premium Student Class:**

****

**Main Class:**

****

**Output:**

****

****

**Q10. Smart Home Devices (Interfaces + Polymorphism)**

**Problem**

Smart homes support multiple devices like fans and lights. Each device must implement

standard operations (turnOn, turnOff). This demonstrates **interfaces and**

**polymorphism**.

**Class Specifications:**

• Interface Device: turnOn(), turnOff()

• Fan, Light implement Device

**Input format:**

Fan

Light

**Output format:**

Fan is now ON

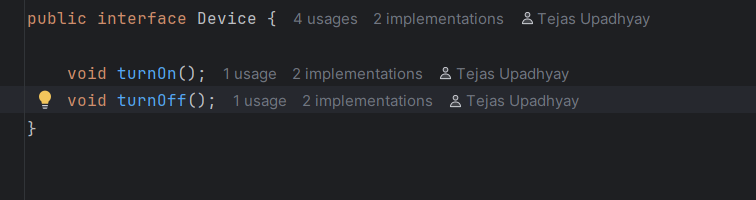
Fan is now OFF

Light is now ON

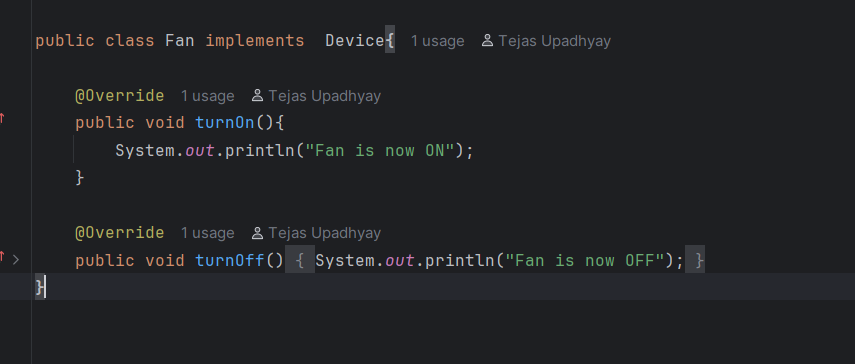
Light is now OFF

**Solution:**

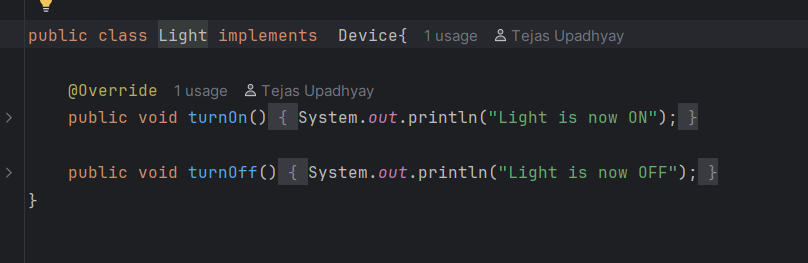
**Device Interface:**

****

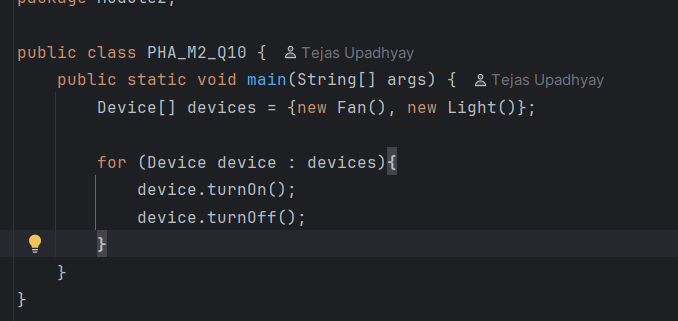
**Fan Class:**

****

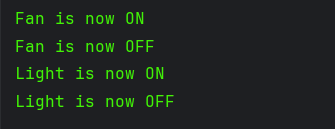
**Light Class:**

****

**Main Class:**

****

**Output:**

****

**Q11. University Hostel Allocation (One-to-One Relationship)**

**Problem**

Each student in a university hostel gets exactly **one room**, and each room is allocated to

**only one student**. This models a strict **one-to-one relationship**.

**Class Specifications:**

• Student: name, roll, course, room (Room)

• Room: roomNumber, block, type (Single/Double)

• Constructors + getters/setters

• toString() in Student prints room details

**Input:**

Ravi,101,CSE

A101,Block-B,Single

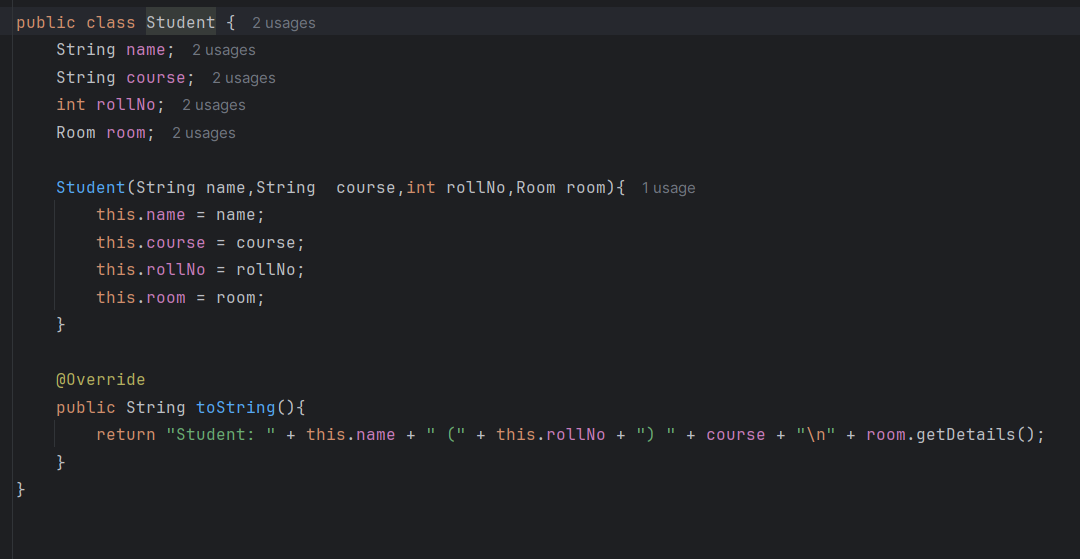
**Output:**

Student: Ravi (101) CSE

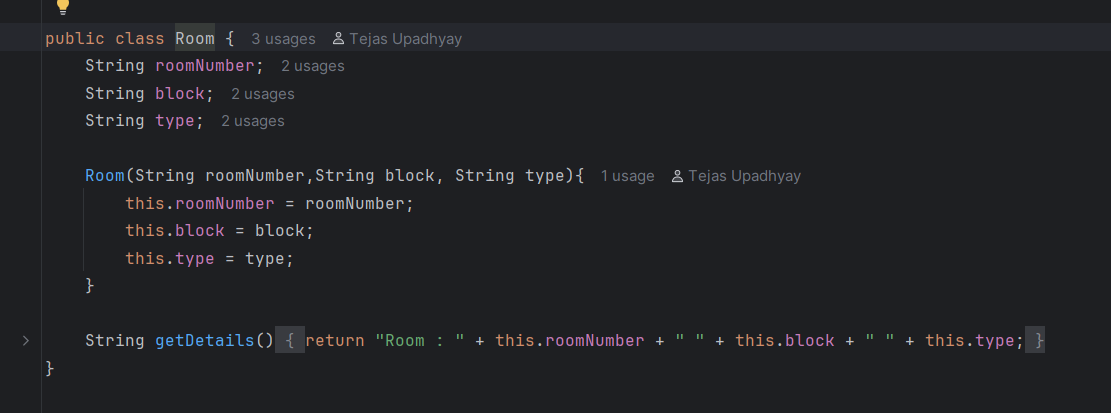
Room: A101 Block-B Single

**Solution:**

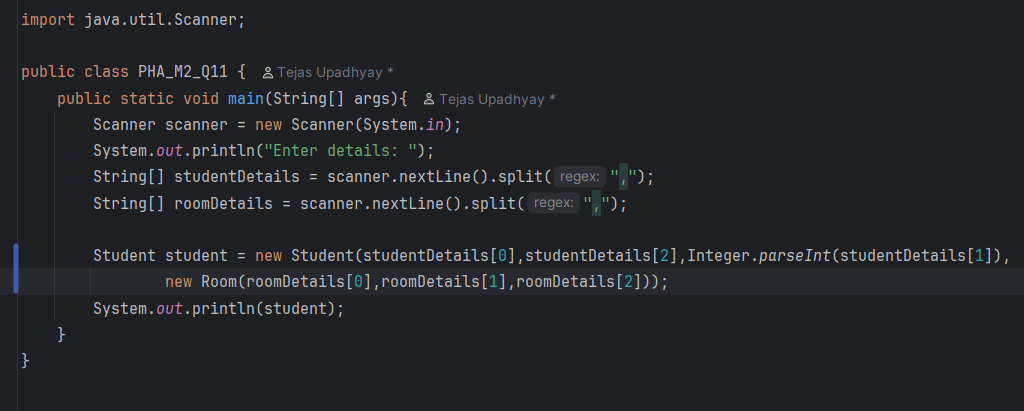
**Student Class:**

****

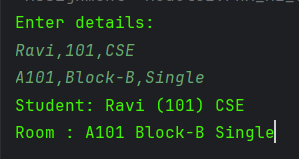
**Room Class:**

****

**Main Class:**

****

**Output:**



**Q12. Vehicle Rental System (Inheritance + Overriding)**

**Problem**

Rental companies offer cars and bikes. Both are vehicles but their rental cost calculation

differs.

**Class Specifications:**

• Vehicle: regNo, brand, baseRate

• Car extends Vehicle: rate = baseRate \* 1.5

• Bike extends Vehicle: rate = baseRate \* 1.2

**Input:**

Car,KA01AA1234,Toyota,1000

Bike,KA05BB6789,Honda,500

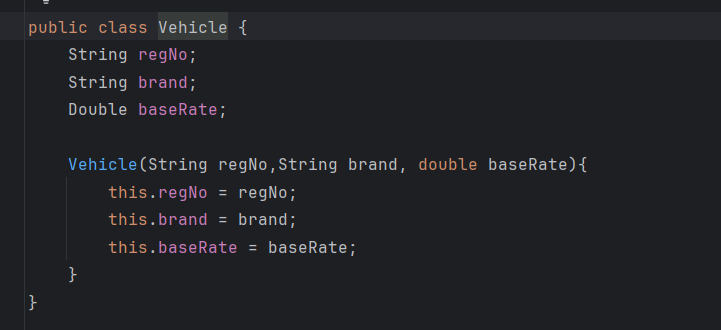
**Output:**

Car KA01AA1234 Toyota Rent: 1500.0

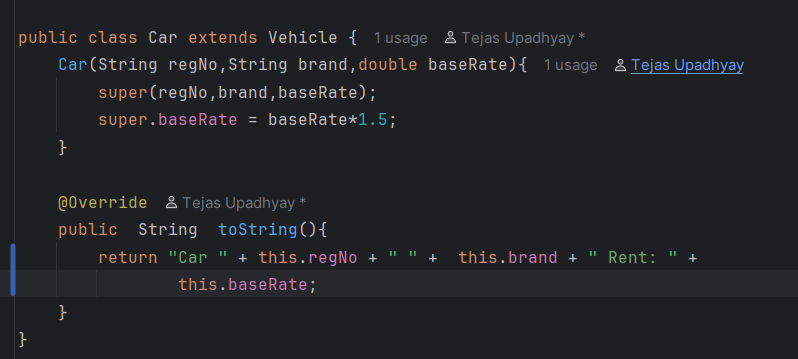
Bike KA05BB6789 Honda Rent: 600.0

**Solution:**

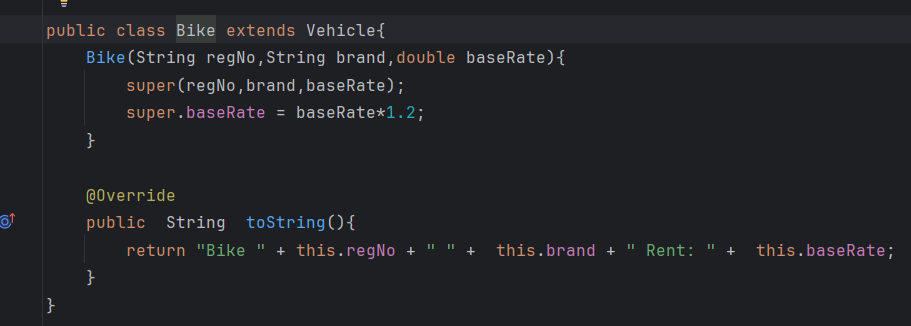
**Vehicle Class:**

****

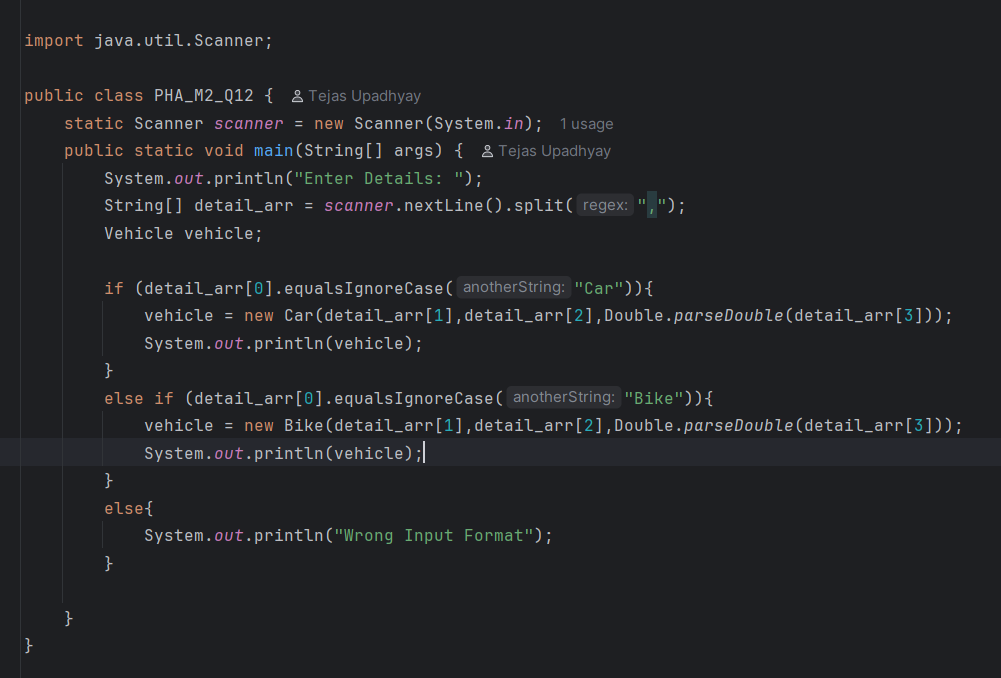
**Car Class:**

****

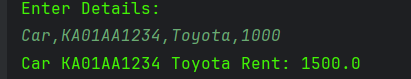
**Bike Class:**

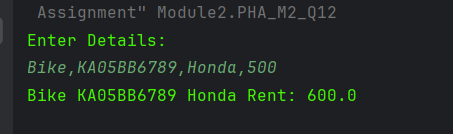
****

**Main Class:**

****

**Output:**

****

****

**Q13. Hotel Reservation System (Aggregation)**

**Problem**

A hotel reservation includes multiple guests under a single booking.

**Class Specifications:**

• Guest: name, age, ID proof

• Reservation: reservationId, roomType, guests (List<Guest>)

**Input:**

R101,Deluxe,2

Amit,25,ID123

Sara,22,ID456

**Output:**

Reservation ID: R101 Room: Deluxe

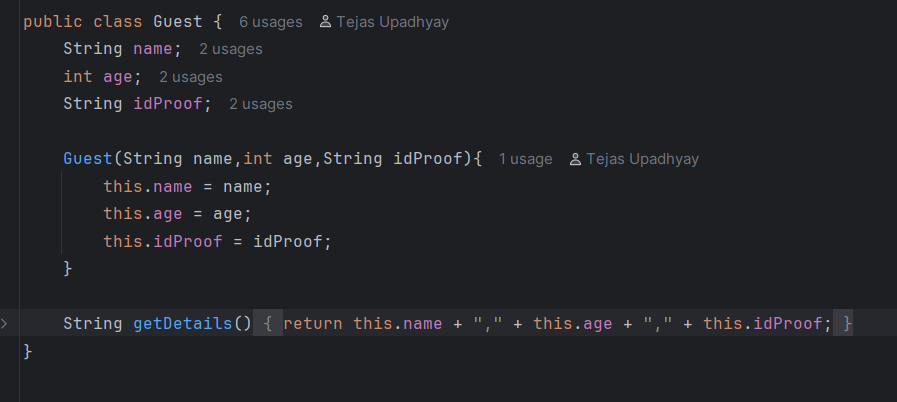
Guests:

Amit,25,ID123

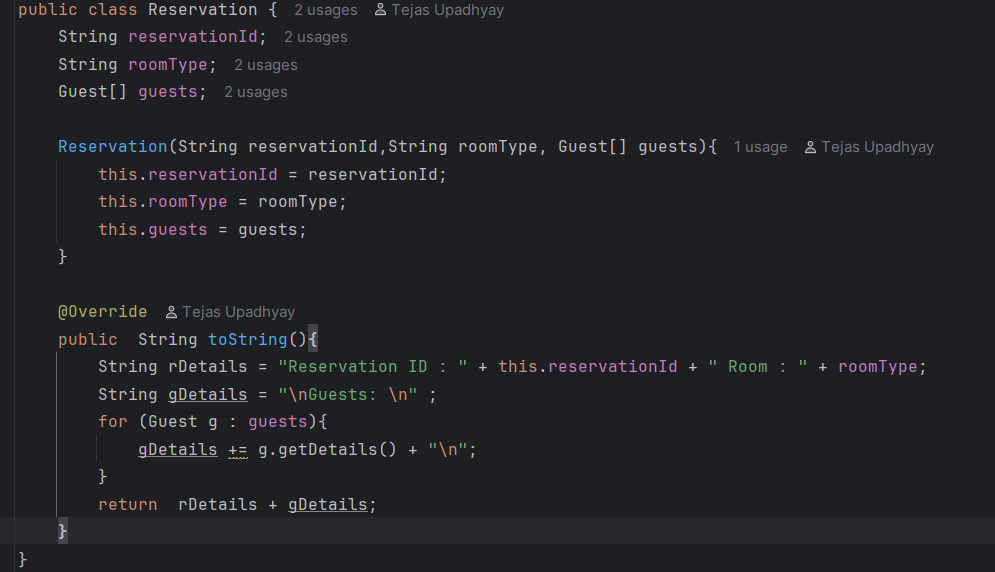
Sara,22,ID456

**Solution:**

**Guest Class:**

****

**Reservation Class:**

****

**Main Class:**

****

**Output:**

****

**Q14. Banking ATM Simulation (Encapsulation)**

**Problem**

Simulate an ATM where users can deposit, withdraw, and check balance. Balance must

be private (encapsulation).

**Class Specifications:**

• Account: accNo, holderName, private balance

• Methods: deposit(), withdraw(), getBalance()

**Input:**

3

deposit 1000

withdraw 500

getBalance

**Output:**

Deposited: 1000.0

Withdrawn: 500.0

Balance: 500.0

**Solution:**

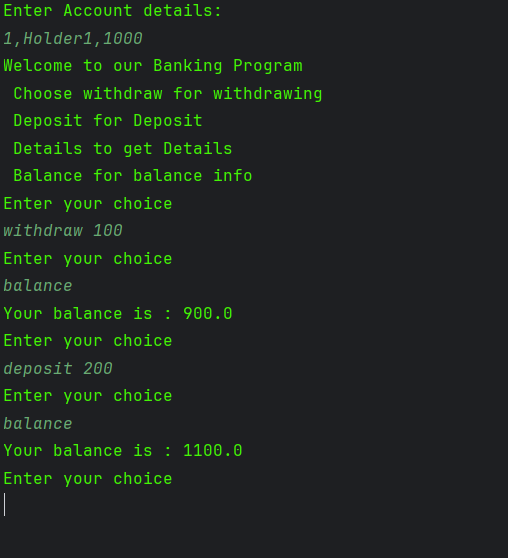
**Reservation Class:**

****

**Main Class:**

****

**Output:**

****

**Q15. Passport – Citizen (One-to-One Relationship)**

**Problem**

A passport belongs to exactly one citizen. This forms a **strict one-to-one relationship**.

**Class Specifications:**

• Citizen: name, dob, address, passport (Passport)

• Passport: passportNo, issueDate, expiryDate

• toString() in Citizen prints passport details

**Input:**

Ravi,01-01-1990,Delhi

P123456,01-01-2020,01-01-2030

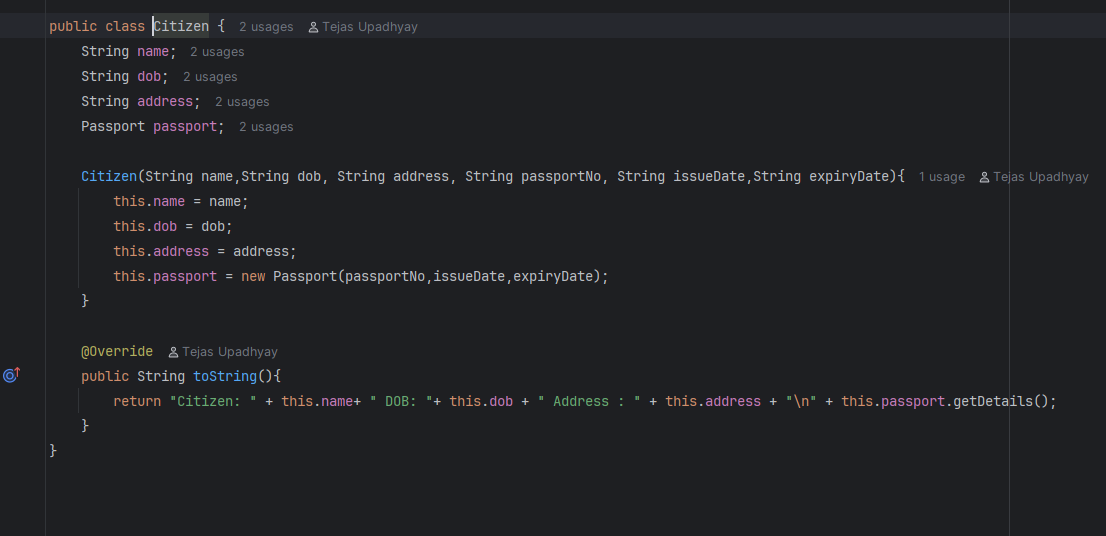
**Output:**

Citizen: Ravi DOB: 01-01-1990 Address: Delhi

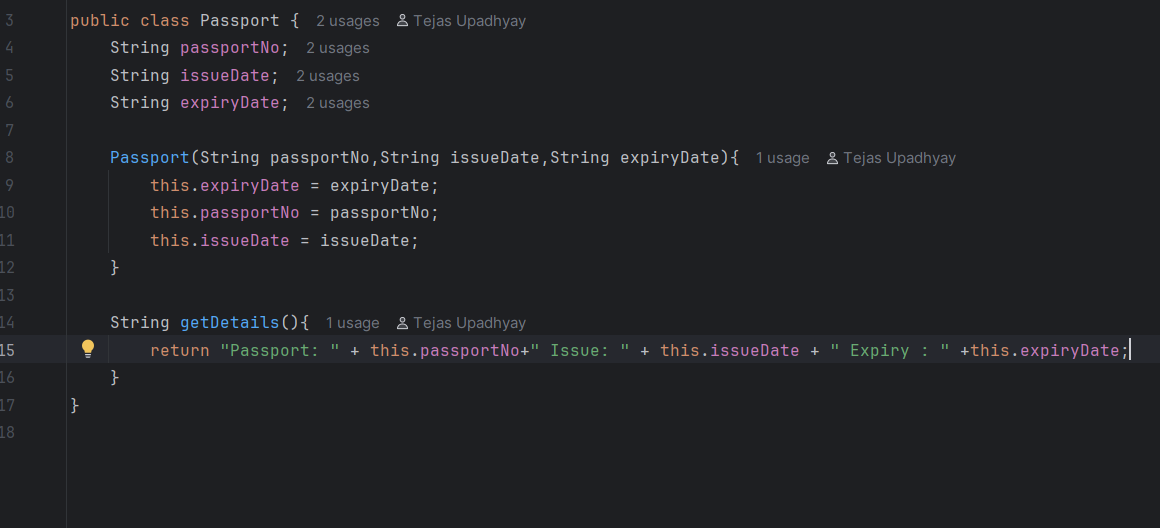
Passport: P123456 Issue: 01-01-2020 Expiry: 01-01-2030

**Solution**

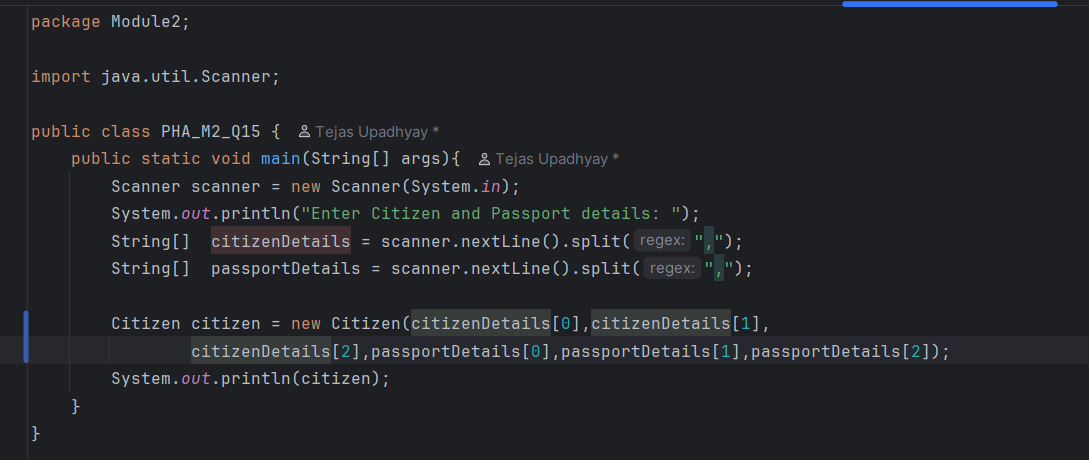
**Citizen Class:**

****

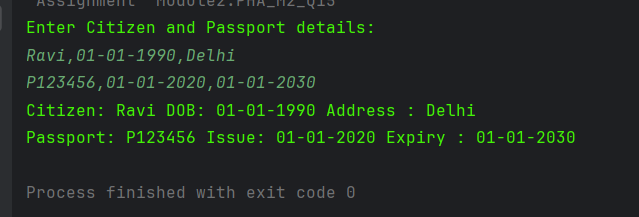
**Passport Class:**

****

**Main Class:**

****

**Output:**

****