

**Java OOP Introduction: Classes and Objects - Homework Assignments**

**Programming Assignment Instructions**

1. Each assignment should be implemented in a separate Java file

2. Follow proper naming conventions for classes, variables, and methods 3. Include proper documentation and comments explaining your code

4. Test your programs with multiple test cases

5. Submit both source code (.java files) and output screenshots

6. Handle exceptions appropriately where necessary

7. Use proper encapsulation with private variables and public methods 8. Demonstrate the use of both static and instance members

**Assignment 1: Personal Finance Manager**

**Topic**: Basic Class Creation and Object Usage

**Problem Statement**: Create a personal finance management system to track income, expenses, and savings for individuals.

**Requirements**:

● Create a PersonalAccount class with private attributes: accountHolderName (String), accountNumber (String), currentBalance (double), totalIncome (double), totalExpenses (double)

● Include static variables: totalAccounts (int), bankName (String)

● Implement methods: addIncome(double amount, String description), addExpense(double amount, String description), calculateSavings(), displayAccountSummary()

● Create static methods: setBankName(String name), getTotalAccounts(), generateAccountNumber()

● In the main method, create at least 3 different personal accounts, perform various transactions, and display account summaries

1



● Demonstrate the difference between static and instance variables by showing how bankName is shared across all accounts while individual balances are unique

**Deliverables**: Complete Java program with test cases showing multiple account operations and proper OOP implementation.

SOLUTION:

import java.util.Scanner;

class PersonalAccount {

private String accountHolderName;

private String accountNumber;

private double currentBalance;

private double totalIncome;

private double totalExpenses;

private static int totalAccounts = 0;

private static String bankName = "Default Bank";

public PersonalAccount(String accountHolderName) {

this.accountHolderName = accountHolderName;

this.accountNumber = generateAccountNumber();

this.currentBalance = 0.0;

this.totalIncome = 0.0;

this.totalExpenses = 0.0;

totalAccounts++;

}

public static void setBankName(String name) {

bankName = name;

}

public static String getBankName() {

return bankName;

}

public static int getTotalAccounts() {

return totalAccounts;

}

private static String generateAccountNumber() {

return "ACC" + (1000 + totalAccounts + 1);

}

public void addIncome(double amount, String description) {

try {

if (amount <= 0) {

throw new IllegalArgumentException("Income amount must be greater than zero.");

}

currentBalance += amount;

totalIncome += amount;

System.out.println("Income Added: Rs." + amount + " | Description: " + description);

} catch (IllegalArgumentException e) {

System.out.println("Error: " + e.getMessage());

}

}

public void addExpense(double amount, String description) {

try {

if (amount <= 0) {

throw new IllegalArgumentException("Expense amount must be greater than zero.");

}

if (amount > currentBalance) {

throw new IllegalArgumentException("Insufficient balance for this expense!");

}

currentBalance -= amount;

totalExpenses += amount;

System.out.println("Expense Added: Rs." + amount + " | Description: " + description);

} catch (IllegalArgumentException e) {

System.out.println("Error: " + e.getMessage());

}

}

public double calculateSavings() {

return totalIncome - totalExpenses;

}

public void displayAccountSummary() {

System.out.println("\n====== Account Summary ======");

System.out.println("Bank Name : " + bankName);

System.out.println("Account Holder : " + accountHolderName);

System.out.println("Account Number : " + accountNumber);

System.out.println("Total Income : Rs." + totalIncome);

System.out.println("Total Expenses : Rs." + totalExpenses);

System.out.println("Current Balance : Rs." + currentBalance);

System.out.println("Savings : Rs." + calculateSavings());

System.out.println("==============================\n");

}

}

public class PersonalFinanceManager {

public static void main(String[] args) {

PersonalAccount.setBankName("Aayush National Bank");

PersonalAccount acc1 = new PersonalAccount("Aayush Rai");

PersonalAccount acc2 = new PersonalAccount("Rahul Verma");

PersonalAccount acc3 = new PersonalAccount("Priya Sharma");

acc1.addIncome(25000, "Salary Credit");

acc1.addExpense(5000, "Groceries");

acc1.addExpense(2000, "Movie Tickets");

acc2.addIncome(30000, "Freelance Project");

acc2.addExpense(8000, "Rent Payment");

acc2.addExpense(3000, "Shopping");

acc3.addIncome(15000, "Part-time Job");

acc3.addExpense(2000, "Electricity Bill");

acc3.addExpense(1000, "Snacks");

acc1.displayAccountSummary();

acc2.displayAccountSummary();

acc3.displayAccountSummary();

System.out.println("Total Accounts Created: " + PersonalAccount.getTotalAccounts());

System.out.println("\nBank Name is shared among all accounts: " + PersonalAccount.getBankName());

}

}

OUTPUT:

Income Added: Rs.25000.0 | Description: Salary Credit

Expense Added: Rs.5000.0 | Description: Groceries

Expense Added: Rs.2000.0 | Description: Movie Tickets

Income Added: Rs.30000.0 | Description: Freelance Project

Expense Added: Rs.8000.0 | Description: Rent Payment

Expense Added: Rs.3000.0 | Description: Shopping

Income Added: Rs.15000.0 | Description: Part-time Job

Expense Added: Rs.2000.0 | Description: Electricity Bill

Expense Added: Rs.1000.0 | Description: Snacks

====== Account Summary ======

Bank Name : Aayush National Bank

Account Holder : Aayush Rai

Account Number : ACC1001

Total Income : Rs.25000.0

Total Expenses : Rs.7000.0

Current Balance : Rs.18000.0

Savings : Rs.18000.0

==============================

====== Account Summary ======

Bank Name : Aayush National Bank

Account Holder : Rahul Verma

Account Number : ACC1002

Total Income : Rs.30000.0

Total Expenses : Rs.11000.0

Current Balance : Rs.19000.0

Savings : Rs.19000.0

==============================

====== Account Summary ======

Bank Name : Aayush National Bank

Account Holder : Priya Sharma

Account Number : ACC1003

Total Income : Rs.15000.0

Total Expenses : Rs.3000.0

Current Balance : Rs.12000.0

Savings : Rs.12000.0

==============================

Total Accounts Created: 3

Bank Name is shared among all accounts: Aayush National Bank

**Assignment 2: Online Shopping Cart System Topic**: Object Relationships and Method Interaction

**Problem Statement**: Develop an online shopping cart system that manages products and customer purchases.

**Requirements**:

● Create a Product class with attributes: productId (String), productName (String), price (double), category (String), stockQuantity (int)

● Create a ShoppingCart class with attributes: cartId (String), customerName (String), products (Product array), quantities (int array), cartTotal (double) ● Include static variables in Product class: totalProducts (int), categories (String array)

● Implement methods in ShoppingCart: addProduct(Product product, int quantity), removeProduct(String productId), calculateTotal(), displayCart(), checkout()

● Create static methods in Product class: findProductById(Product[] products, String productId), getProductsByCategory(Product[] products, String category)

● Create a menu-driven system allowing users to browse products, add/remove items from cart, and checkout

● Demonstrate object interaction where ShoppingCart objects contain and manipulate Product objects

**Deliverables**: Complete shopping cart system with at least 10 different products and comprehensive testing of all functionalities.

SOLUTION:

import java.util.Scanner;

class Product {

String productId;

String productName;

double price;

String category;

int stockQuantity;

static int totalProducts = 0;

public Product(String productId, String productName, double price, String category, int stockQuantity) {

this.productId = productId;

this.productName = productName;

this.price = price;

this.category = category;

this.stockQuantity = stockQuantity;

totalProducts++;

}

public static Product findProductById(Product[] products, String productId) {

for (Product product : products) {

if (product.productId.equalsIgnoreCase(productId)) {

return product;

}

}

return null;

}

}

class ShoppingCart {

Product[] products;

int[] quantities;

int count;

double cartTotal;

public ShoppingCart() {

products = new Product[50];

quantities = new int[50];

count = 0;

cartTotal = 0.0;

}

public void addProduct(Product product, int quantity) {

if (product == null || quantity <= 0 || product.stockQuantity < quantity) {

System.out.println("Cannot add product!");

return;

}

for (int i = 0; i < count; i++) {

if (products[i].productId.equals(product.productId)) {

quantities[i] += quantity;

product.stockQuantity -= quantity;

calculateTotal();

System.out.println("Added " + quantity + " more " + product.productName + " to cart.");

return;

}

}

products[count] = product;

quantities[count] = quantity;

product.stockQuantity -= quantity;

count++;

calculateTotal();

System.out.println(product.productName + " added to cart.");

}

public void removeProduct(String productId) {

for (int i = 0; i < count; i++) {

if (products[i].productId.equalsIgnoreCase(productId)) {

products[i].stockQuantity += quantities[i];

for (int j = i; j < count - 1; j++) {

products[j] = products[j + 1];

quantities[j] = quantities[j + 1];

}

products[count - 1] = null;

quantities[count - 1] = 0;

count--;

calculateTotal();

System.out.println("Product removed.");

return;

}

}

System.out.println("Product not found in cart.");

}

public void calculateTotal() {

cartTotal = 0;

for (int i = 0; i < count; i++) {

cartTotal += products[i].price \* quantities[i];

}

}

public void displayCart() {

if (count == 0) {

System.out.println("\nCart is empty.");

return;

}

System.out.println("\n=== Shopping Cart ===");

for (int i = 0; i < count; i++) {

System.out.println(products[i].productId + " | " + products[i].productName + " | Rs." + products[i].price + " | Qty: " + quantities[i]);

}

System.out.println("Cart Total: Rs." + cartTotal);

}

public void checkout() {

if (count == 0) {

System.out.println("Cart is empty. Cannot checkout.");

return;

}

displayCart();

System.out.println("\nThank you for shopping!");

products = new Product[50];

quantities = new int[50];

count = 0;

cartTotal = 0.0;

}

}

public class OnlineShoppingSystem {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Product[] products = {

new Product("P101", "Laptop", 55000, "Electronics", 10),

new Product("P102", "Smartphone", 25000, "Electronics", 15),

new Product("P103", "Headphones", 2000, "Electronics", 20),

new Product("P104", "T-Shirt", 600, "Clothing", 30),

new Product("P105", "Jeans", 1200, "Clothing", 25),

new Product("P106", "Novel", 400, "Books", 40),

new Product("P107", "Notebook", 50, "Books", 100),

new Product("P108", "Rice Bag", 1200, "Groceries", 50),

new Product("P109", "Milk Packet", 60, "Groceries", 100),

new Product("P110", "Snacks Pack", 100, "Groceries", 80)

};

ShoppingCart cart = new ShoppingCart();

int choice;

do {

System.out.println("\n=== Online Shopping Menu ===");

System.out.println("1. View All Products");

System.out.println("2. Search Product by ID");

System.out.println("3. Add Product to Cart");

System.out.println("4. Remove Product from Cart");

System.out.println("5. View Cart");

System.out.println("6. Checkout");

System.out.println("7. Exit");

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

choice = sc.nextInt();

sc.nextLine();

switch (choice) {

case 1:

System.out.println("\n=== Available Products ===");

for (Product p : products) {

System.out.println(p.productId + " | " + p.productName + " | Rs." + p.price + " | Stock: " + p.stockQuantity);

}

break;

case 2:

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

String searchId = sc.nextLine();

Product found = Product.findProductById(products, searchId);

if (found != null) {

System.out.println(found.productId + " | " + found.productName + " | Rs." + found.price + " | Stock: " + found.stockQuantity);

} else {

System.out.println("Product not found.");

}

break;

case 3:

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

String addId = sc.nextLine();

Product prod = Product.findProductById(products, addId);

if (prod == null) {

System.out.println("Product not found!");

break;

}

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

int qty = sc.nextInt();

cart.addProduct(prod, qty);

break;

case 4:

System.out.print("Enter Product ID to Remove: ");

String removeId = sc.nextLine();

cart.removeProduct(removeId);

break;

case 5:

cart.displayCart();

break;

case 6:

cart.checkout();

break;

case 7:

System.out.println("Exiting... Thank you!");

break;

default:

System.out.println("Invalid choice!");

}

} while (choice != 7);

sc.close();

}

}

OUTPUT:

=== Online Shopping Menu ===

1. View All Products

2. Search Product by ID

3. Add Product to Cart

4. Remove Product from Cart

5. View Cart

6. Checkout

7. Exit

Enter choice: 4

Enter Product ID to Remove: 103

Product not found in cart.

=== Online Shopping Menu ===

1. View All Products

2. Search Product by ID

3. Add Product to Cart

4. Remove Product from Cart

5. View Cart

6. Checkout

7. Exit

Enter choice: 1

=== Available Products ===

P101 | Laptop | Rs.55000.0 | Stock: 10

P102 | Smartphone | Rs.25000.0 | Stock: 15

P103 | Headphones | Rs.2000.0 | Stock: 20

P104 | T-Shirt | Rs.600.0 | Stock: 30

P105 | Jeans | Rs.1200.0 | Stock: 25

P106 | Novel | Rs.400.0 | Stock: 40

P107 | Notebook | Rs.50.0 | Stock: 100

P108 | Rice Bag | Rs.1200.0 | Stock: 50

P109 | Milk Packet | Rs.60.0 | Stock: 100

P110 | Snacks Pack | Rs.100.0 | Stock: 80

=== Online Shopping Menu ===

1. View All Products

2. Search Product by ID

3. Add Product to Cart

4. Remove Product from Cart

5. View Cart

6. Checkout

7. Exit

Enter choice: 2

Enter Product ID: 105

Product not found.

=== Online Shopping Menu ===

1. View All Products

2. Search Product by ID

3. Add Product to Cart

4. Remove Product from Cart

5. View Cart

6. Checkout

7. Exit

Enter choice: 7

Exiting... Thank you!

2



**Assignment 3: Hotel Reservation System**

**Topic**: Multiple Classes with Complex Interactions

**Problem Statement**: Build a hotel reservation management system handling rooms, guests, and bookings.

**Requirements**:

● Create a Room class with attributes: roomNumber (String), roomType (String), pricePerNight (double), isAvailable (boolean), maxOccupancy (int) ● Create a Guest class with attributes: guestId (String), guestName (String), phoneNumber (String), email (String), bookingHistory (String array) ● Create a Booking class with attributes: bookingId (String), guest (Guest object), room (Room object), checkInDate (String), checkOutDate (String), totalAmount (double)

● Include static variables: totalBookings (int), hotelRevenue (double), hotelName (String)

● Implement reservation management methods: makeReservation(), cancelReservation(), checkAvailability(), calculateBill() ● Create static methods for reporting: getOccupancyRate(), getTotalRevenue(), getMostPopularRoomType()

● Implement a complete booking workflow from room search to checkout

**Deliverables**: Full hotel management system with multiple room types, guest management, and comprehensive booking operations.

SOLUTION:

import java.util.\*;

class Room {

private String roomNumber;

private String roomType;

private double pricePerNight;

private boolean isAvailable;

private int maxOccupancy;

public Room(String roomNumber, String roomType, double pricePerNight, boolean isAvailable, int maxOccupancy) {

this.roomNumber = roomNumber;

this.roomType = roomType;

this.pricePerNight = pricePerNight;

this.isAvailable = isAvailable;

this.maxOccupancy = maxOccupancy;

}

public String getRoomNumber() { return roomNumber; }

public String getRoomType() { return roomType; }

public double getPricePerNight() { return pricePerNight; }

public boolean isAvailable() { return isAvailable; }

public void setAvailable(boolean available) { isAvailable = available; }

public int getMaxOccupancy() { return maxOccupancy; }

}

class Guest {

private String guestId;

private String guestName;

private String phoneNumber;

private String email;

private ArrayList<String> bookingHistory;

public Guest(String guestId, String guestName, String phoneNumber, String email) {

this.guestId = guestId;

this.guestName = guestName;

this.phoneNumber = phoneNumber;

this.email = email;

this.bookingHistory = new ArrayList<>();

}

public String getGuestId() { return guestId; }

public String getGuestName() { return guestName; }

public void addBooking(String bookingId) { bookingHistory.add(bookingId); }

}

class Booking {

private String bookingId;

private Guest guest;

private Room room;

private String checkInDate;

private String checkOutDate;

private double totalAmount;

private static int totalBookings = 0;

private static double hotelRevenue = 0;

private static String hotelName = "Grand Palace";

public Booking(String bookingId, Guest guest, Room room, String checkInDate, String checkOutDate, double totalAmount) {

this.bookingId = bookingId;

this.guest = guest;

this.room = room;

this.checkInDate = checkInDate;

this.checkOutDate = checkOutDate;

this.totalAmount = totalAmount;

totalBookings++;

hotelRevenue += totalAmount;

guest.addBooking(bookingId);

}

public String getBookingId() { return bookingId; }

public Room getRoom() { return room; }

public static double getTotalRevenue() { return hotelRevenue; }

public static int getTotalBookings() { return totalBookings; }

public static String getHotelName() { return hotelName; }

}

class HotelReservationSystem {

private static ArrayList<Room> rooms = new ArrayList<>();

private static ArrayList<Booking> bookings = new ArrayList<>();

private static Scanner sc = new Scanner(System.in);

public static void initializeRooms() {

rooms.add(new Room("101", "Single", 1500, true, 1));

rooms.add(new Room("102", "Double", 2500, true, 2));

rooms.add(new Room("103", "Suite", 5000, true, 4));

rooms.add(new Room("104", "Double", 2500, true, 2));

rooms.add(new Room("105", "Single", 1500, true, 1));

}

public static Room checkAvailability(String type) {

for (Room room : rooms) {

if (room.getRoomType().equalsIgnoreCase(type) && room.isAvailable()) {

return room;

}

}

return null;

}

public static void makeReservation() {

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

String guestId = sc.next();

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

String guestName = sc.next();

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

String phone = sc.next();

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

String email = sc.next();

Guest guest = new Guest(guestId, guestName, phone, email);

System.out.print("Enter Room Type (Single/Double/Suite): ");

String roomType = sc.next();

Room room = checkAvailability(roomType);

if (room == null) {

System.out.println("No available rooms of this type!");

return;

}

System.out.print("Enter Check-in Date: ");

String checkIn = sc.next();

System.out.print("Enter Check-out Date: ");

String checkOut = sc.next();

System.out.print("Enter Number of Nights: ");

int nights = sc.nextInt();

double totalAmount = room.getPricePerNight() \* nights;

String bookingId = "B" + (bookings.size() + 1);

Booking booking = new Booking(bookingId, guest, room, checkIn, checkOut, totalAmount);

bookings.add(booking);

room.setAvailable(false);

System.out.println("Booking Successful! ID: " + bookingId + " | Total: Rs." + totalAmount);

}

public static void cancelReservation() {

System.out.print("Enter Booking ID to Cancel: ");

String bookingId = sc.next();

Booking found = null;

for (Booking b : bookings) {

if (b.getBookingId().equalsIgnoreCase(bookingId)) {

found = b;

break;

}

}

if (found == null) {

System.out.println("Booking not found!");

return;

}

found.getRoom().setAvailable(true);

bookings.remove(found);

System.out.println("Booking " + bookingId + " cancelled successfully!");

}

public static void displayReport() {

System.out.println("Hotel: " + Booking.getHotelName());

System.out.println("Total Bookings: " + Booking.getTotalBookings());

System.out.println("Total Revenue: Rs." + Booking.getTotalRevenue());

}

public static void main(String[] args) {

initializeRooms();

while (true) {

System.out.println("\n=== Hotel Reservation System ===");

System.out.println("1. Make Reservation");

System.out.println("2. Cancel Reservation");

System.out.println("3. Display Report");

System.out.println("4. Exit");

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

int choice = sc.nextInt();

switch (choice) {

case 1: makeReservation(); break;

case 2: cancelReservation(); break;

case 3: displayReport(); break;

case 4: System.exit(0);

default: System.out.println("Invalid choice!");

}

}

}

}

OUTPUT:

=== Hotel Reservation System ===

1. Make Reservation

2. Cancel Reservation

3. Display Report

4. Exit

Enter choice: 1

Enter Guest ID: 101

Enter Name: aayush

Enter Phone: 9898000000

Enter Email: abc123@yahoo.com

Enter Room Type (Single/Double/Suite): 2

No available rooms of this type!

=== Hotel Reservation System ===

1. Make Reservation

2. Cancel Reservation

3. Display Report

4. Exit

Enter choice: 2

Enter Booking ID to Cancel: 103

Booking not found!

=== Hotel Reservation System ===

1. Make Reservation

2. Cancel Reservation

3. Display Report

4. Exit

Enter choice: 3

Hotel: Grand Palace

Total Bookings: 0

Total Revenue: Rs.0.0

=== Hotel Reservation System ===

1. Make Reservation

2. Cancel Reservation

3. Display Report

4. Exit

Enter choice: 4

3



**Assignment 4: Student Grade Management System Topic**: Static vs Instance Members and Data Processing

**Problem Statement**: Create a comprehensive student grade management system for a school. **Requirements**:

● Create a Student class with attributes: studentId (String), studentName (String), className (String), subjects (String array), marks (double 2D array), gpa (double) ● Include static variables: totalStudents (int), schoolName (String), gradingScale (String array), passPercentage (double)

● Create a Subject class with: subjectCode (String), subjectName (String), credits (int), instructor (String)

● Implement methods: addMarks(String subject, double marks), calculateGPA(), generateReportCard(), checkPromotionEligibility() ● Create static methods: setGradingScale(),

calculateClassAverage(Student[] students),

getTopPerformers(Student[] students, int count),

generateSchoolReport()

● Include grade categorization (A, B, C, D, F) based on percentage ranges ● Create a system to handle multiple classes and generate comparative reports

**Deliverables**: Complete grade management system with statistical analysis and reporting capabilities for multiple students and subjects.

SOLUTION:

import java.util.\*;

class Subject {

String subjectCode;

String subjectName;

int credits;

String instructor;

public Subject(String subjectCode, String subjectName, int credits, String instructor) {

this.subjectCode = subjectCode;

this.subjectName = subjectName;

this.credits = credits;

this.instructor = instructor;

}

}

class Student {

private String studentId;

private String studentName;

private String className;

private String[] subjects;

private double[][] marks;

private double gpa;

static int totalStudents = 0;

static String schoolName = "ABC International School";

static String[] gradingScale = {"A", "B", "C", "D", "F"};

static double passPercentage = 40.0;

public Student(String studentId, String studentName, String className, String[] subjects) {

this.studentId = studentId;

this.studentName = studentName;

this.className = className;

this.subjects = subjects;

this.marks = new double[subjects.length][3];

this.gpa = 0.0;

totalStudents++;

}

public void addMarks(String subject, double[] subjectMarks) {

for (int i = 0; i < subjects.length; i++) {

if (subjects[i].equalsIgnoreCase(subject)) {

marks[i] = subjectMarks;

break;

}

}

}

public void calculateGPA() {

double total = 0;

int totalSubjects = subjects.length;

for (int i = 0; i < totalSubjects; i++) {

double sum = 0;

for (double m : marks[i]) sum += m;

double average = sum / marks[i].length;

total += average;

}

gpa = total / totalSubjects / 10;

}

public String getGrade(double percentage) {

if (percentage >= 90) return gradingScale[0];

else if (percentage >= 75) return gradingScale[1];

else if (percentage >= 60) return gradingScale[2];

else if (percentage >= 40) return gradingScale[3];

else return gradingScale[4];

}

public void generateReportCard() {

System.out.println("\n==============================");

System.out.println("Report Card - " + studentName);

System.out.println("Student ID: " + studentId + " | Class: " + className);

System.out.println("School: " + schoolName);

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

double totalMarks = 0;

double maxMarks = 0;

for (int i = 0; i < subjects.length; i++) {

double sum = 0;

for (double m : marks[i]) sum += m;

double percentage = sum / marks[i].length;

totalMarks += percentage;

maxMarks += 100;

System.out.println(subjects[i] + " : " + String.format("%.2f", percentage) + "% | Grade: " + getGrade(percentage));

}

double overallPercentage = (totalMarks / subjects.length);

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

System.out.println("Overall GPA: " + String.format("%.2f", gpa));

System.out.println("Overall Percentage: " + String.format("%.2f", overallPercentage) + "%");

System.out.println("Final Grade: " + getGrade(overallPercentage));

System.out.println("Promotion Status: " + (overallPercentage >= passPercentage ? "Promoted" : "Not Promoted"));

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

}

public static void setGradingScale(String[] newScale) {

gradingScale = newScale;

}

public static double calculateClassAverage(Student[] students) {

double total = 0;

for (Student s : students) {

s.calculateGPA();

total += s.gpa;

}

return total / students.length;

}

public static void getTopPerformers(Student[] students, int count) {

Arrays.sort(students, (a, b) -> Double.compare(b.gpa, a.gpa));

System.out.println("\nTop " + count + " Performers:");

for (int i = 0; i < count && i < students.length; i++) {

System.out.println((i + 1) + ". " + students[i].studentName + " - GPA: " + String.format("%.2f", students[i].gpa));

}

}

public static void generateSchoolReport(Student[] students) {

System.out.println("\n=========== School Report ===========");

System.out.println("Total Students: " + totalStudents);

double avgGPA = calculateClassAverage(students);

System.out.println("Average GPA: " + String.format("%.2f", avgGPA));

getTopPerformers(students, 3);

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

}

}

public class StudentGradeManagementSystem {

public static void main(String[] args) {

String[] subjects = {"Math", "Science", "English", "Computer"};

Student s1 = new Student("S001", "Aayush Rai", "10-A", subjects);

Student s2 = new Student("S002", "Rahul Verma", "10-A", subjects);

Student s3 = new Student("S003", "Priya Singh", "10-A", subjects);

s1.addMarks("Math", new double[]{90, 85, 88});

s1.addMarks("Science", new double[]{92, 89, 95});

s1.addMarks("English", new double[]{80, 78, 85});

s1.addMarks("Computer", new double[]{95, 97, 96});

s2.addMarks("Math", new double[]{75, 70, 72});

s2.addMarks("Science", new double[]{78, 80, 79});

s2.addMarks("English", new double[]{65, 68, 70});

s2.addMarks("Computer", new double[]{85, 83, 82});

s3.addMarks("Math", new double[]{55, 60, 58});

s3.addMarks("Science", new double[]{50, 55, 52});

s3.addMarks("English", new double[]{62, 64, 61});

s3.addMarks("Computer", new double[]{70, 72, 68});

s1.calculateGPA();

s2.calculateGPA();

s3.calculateGPA();

s1.generateReportCard();

s2.generateReportCard();

s3.generateReportCard();

Student[] students = {s1, s2, s3};

Student.generateSchoolReport(students);

}

}

OUTPUT:

==============================

Report Card - Aayush Rai

Student ID: S001 | Class: 10-A

School: ABC International School

==============================

Math : 87.67% | Grade: B

Science : 92.00% | Grade: A

English : 81.00% | Grade: B

Computer : 96.00% | Grade: A

--------------------------------

Overall GPA: 8.92

Overall Percentage: 89.17%

Final Grade: B

Promotion Status: Promoted

==============================

==============================

Report Card - Rahul Verma

Student ID: S002 | Class: 10-A

School: ABC International School

==============================

Math : 72.33% | Grade: C

Science : 79.00% | Grade: B

English : 67.67% | Grade: C

Computer : 83.33% | Grade: B

--------------------------------

Overall GPA: 7.56

Overall Percentage: 75.58%

Final Grade: B

Promotion Status: Promoted

==============================

==============================

Report Card - Priya Singh

Student ID: S003 | Class: 10-A

School: ABC International School

==============================

Math : 57.67% | Grade: D

Science : 52.33% | Grade: D

English : 62.33% | Grade: C

Computer : 70.00% | Grade: C

--------------------------------

Overall GPA: 6.06

Overall Percentage: 60.58%

Final Grade: C

Promotion Status: Promoted

==============================

=========== School Report ===========

Total Students: 3

Average GPA: 7.51

Top 3 Performers:

1. Aayush Rai - GPA: 8.92

2. Rahul Verma - GPA: 7.56

3. Priya Singh - GPA: 6.06

=====================================

4



**Assignment 5: Library Management System with Fine Calculation**

**Topic**: Real-world Application with Business Logic

**Problem Statement**: Develop a comprehensive library management system with member management and fine calculations.

**Requirements**:

● Create a Book class with: bookId (String), title (String), author (String), isbn (String), category (String), isIssued (boolean), issueDate (String), dueDate (String)

● Create a Member class with: memberId (String), memberName (String), memberType (String), booksIssued (Book array), totalFines (double), membershipDate (String) ● Include static variables: totalBooks (int), totalMembers (int), libraryName (String), finePerDay (double), maxBooksAllowed (int)

● Implement methods: issueBook(), returnBook(), calculateFine(), renewBook(), searchBooks(), reserveBook()

● Create different member types (Student, Faculty, General) with different borrowing privileges

● Include static methods: generateLibraryReport(), getOverdueBooks(), getMostPopularBooks()

● Implement automatic fine calculation based on overdue days

**Deliverables**: Full library system with member management, book operations, fine calculations, and comprehensive reporting features.

SOLUTION:

import java.util.\*;

class Book {

private String bookId, title, author, isbn, category, issueDate, dueDate;

private boolean isIssued;

static int totalBooks = 0;

static String libraryName = "City Library";

static double finePerDay = 10.0;

static int maxBooksAllowed = 3;

Book(String bookId, String title, String author, String isbn, String category) {

this.bookId = bookId;

this.title = title;

this.author = author;

this.isbn = isbn;

this.category = category;

this.isIssued = false;

this.issueDate = "";

this.dueDate = "";

totalBooks++;

}

public String getBookId() { return bookId; }

public String getTitle() { return title; }

public boolean isIssued() { return isIssued; }

public void setIssued(boolean issued) { this.isIssued = issued; }

public void setIssueDate(String date) { this.issueDate = date; }

public void setDueDate(String date) { this.dueDate = date; }

public String getDueDate() { return dueDate; }

}

class Member {

private String memberId, memberName, memberType, membershipDate;

private Book[] booksIssued;

private int issuedCount = 0;

private double totalFines;

static int totalMembers = 0;

Member(String memberId, String memberName, String memberType, String membershipDate) {

this.memberId = memberId;

this.memberName = memberName;

this.memberType = memberType;

this.membershipDate = membershipDate;

this.booksIssued = new Book[Book.maxBooksAllowed];

this.totalFines = 0.0;

totalMembers++;

}

public String getMemberId() { return memberId; }

public String getMemberName() { return memberName; }

public Book[] getBooksIssued() { return booksIssued; }

public int getIssuedCount() { return issuedCount; }

public double getTotalFines() { return totalFines; }

public boolean issueBook(Book book, String issueDate, String dueDate) {

if (issuedCount >= Book.maxBooksAllowed || book.isIssued()) return false;

booksIssued[issuedCount++] = book;

book.setIssued(true);

book.setIssueDate(issueDate);

book.setDueDate(dueDate);

return true;

}

public boolean returnBook(String bookId, String returnDate) {

for (int i = 0; i < issuedCount; i++) {

if (booksIssued[i] != null && booksIssued[i].getBookId().equals(bookId)) {

double fine = calculateFine(booksIssued[i].getDueDate(), returnDate);

totalFines += fine;

booksIssued[i].setIssued(false);

booksIssued[i] = booksIssued[issuedCount - 1];

booksIssued[issuedCount - 1] = null;

issuedCount--;

return true;

}

}

return false;

}

public boolean renewBook(String bookId, String newDueDate) {

for (int i = 0; i < issuedCount; i++) {

if (booksIssued[i] != null && booksIssued[i].getBookId().equals(bookId)) {

booksIssued[i].setDueDate(newDueDate);

return true;

}

}

return false;

}

private double calculateFine(String dueDate, String returnDate) {

try {

String[] d1 = dueDate.split("-");

String[] d2 = returnDate.split("-");

Calendar due = Calendar.getInstance();

Calendar ret = Calendar.getInstance();

due.set(Integer.parseInt(d1[2]), Integer.parseInt(d1[1]), Integer.parseInt(d1[0]));

ret.set(Integer.parseInt(d2[2]), Integer.parseInt(d2[1]), Integer.parseInt(d2[0]));

long diff = ret.getTimeInMillis() - due.getTimeInMillis();

long days = diff / (1000 \* 60 \* 60 \* 24);

return days > 0 ? days \* Book.finePerDay : 0.0;

} catch (Exception e) {

return 0.0;

}

}

}

class Library {

private ArrayList<Book> books = new ArrayList<>();

private ArrayList<Member> members = new ArrayList<>();

public void addBook(Book book) { books.add(book); }

public void addMember(Member member) { members.add(member); }

public Book searchBookById(String bookId) {

for (Book book : books) if (book.getBookId().equals(bookId)) return book;

return null;

}

public void generateLibraryReport() {

System.out.println("Library: " + Book.libraryName);

System.out.println("Total Books: " + Book.totalBooks);

System.out.println("Total Members: " + Member.totalMembers);

}

public void getOverdueBooks(String currentDate) {

System.out.println("Overdue Books as of " + currentDate + ":");

for (Member m : members) {

for (Book b : m.getBooksIssued()) {

if (b != null && b.isIssued()) {

double fine = calculateFine(b.getDueDate(), currentDate);

if (fine > 0) System.out.println(b.getTitle() + " - Fine: Rs." + fine);

}

}

}

}

private double calculateFine(String dueDate, String returnDate) {

try {

String[] d1 = dueDate.split("-");

String[] d2 = returnDate.split("-");

Calendar due = Calendar.getInstance();

Calendar ret = Calendar.getInstance();

due.set(Integer.parseInt(d1[2]), Integer.parseInt(d1[1]), Integer.parseInt(d1[0]));

ret.set(Integer.parseInt(d2[2]), Integer.parseInt(d2[1]), Integer.parseInt(d2[0]));

long diff = ret.getTimeInMillis() - due.getTimeInMillis();

long days = diff / (1000 \* 60 \* 60 \* 24);

return days > 0 ? days \* Book.finePerDay : 0.0;

} catch (Exception e) {

return 0.0;

}

}

}

public class LibraryManagementSystem {

public static void main(String[] args) {

Library library = new Library();

Book b1 = new Book("B001", "Java Basics", "James Gosling", "12345", "Programming");

Book b2 = new Book("B002", "Python Crash", "Guido Van Rossum", "54321", "Programming");

Book b3 = new Book("B003", "DSA Handbook", "Narsimha", "67890", "Algorithms");

library.addBook(b1);

library.addBook(b2);

library.addBook(b3);

Member m1 = new Member("M001", "Aayush Rai", "Student", "01-01-2025");

Member m2 = new Member("M002", "Ravi Kumar", "Faculty", "15-02-2025");

library.addMember(m1);

library.addMember(m2);

m1.issueBook(b1, "01-08-2025", "15-08-2025");

m1.issueBook(b2, "01-08-2025", "15-08-2025");

m2.issueBook(b3, "02-08-2025", "12-08-2025");

m1.returnBook("B001", "25-08-2025");

library.generateLibraryReport();

library.getOverdueBooks("30-08-2025");

}

}

OUTPUT:

Library: City Library

Total Books: 3

Total Members: 2

Overdue Books as of 30-08-2025:

Python Crash - Fine: Rs.150.0

DSA Handbook - Fine: Rs.180.0

5



**Assignment 6: Employee Payroll and Attendance System Topic**: Complex Business Logic with Multiple Object Types

**Problem Statement**: Create an integrated employee management system handling payroll, attendance, and performance tracking.

**Requirements**:

● Create an Employee class with: empId (String), empName (String), department (String), designation (String), baseSalary (double), joinDate (String), attendanceRecord (boolean array for 30 days)

● Create a Department class with: deptId (String), deptName (String), manager (Employee), employees (Employee array), budget (double)

● Include static variables: totalEmployees (int), companyName (String), totalSalaryExpense (double), workingDaysPerMonth (int)

● Implement methods: markAttendance(int day, boolean present), calculateSalary(), calculateBonus(), generatePaySlip(),

requestLeave()

● Create different employee types with different salary calculation methods (Full-time, Part-time, Contract)

● Include static methods: calculateCompanyPayroll(),

getDepartmentWiseExpenses(), getAttendanceReport()

● Implement performance-based bonus calculation and leave management

**Deliverables**: Complete HR management system with payroll processing, attendance tracking, and performance evaluation capabilities.

SOLUTION:

import java.util.\*;

class Employee {

String empId, empName, department, designation, joinDate, empType;

double baseSalary;

boolean[] attendanceRecord = new boolean[30];

static int totalEmployees = 0;

static String companyName = "Tech Solutions Pvt Ltd";

static double totalSalaryExpense = 0;

static int workingDaysPerMonth = 30;

Employee(String empId, String empName, String department, String designation, double baseSalary, String joinDate, String empType) {

this.empId = empId;

this.empName = empName;

this.department = department;

this.designation = designation;

this.baseSalary = baseSalary;

this.joinDate = joinDate;

this.empType = empType;

totalEmployees++;

}

void markAttendance(int day, boolean present) {

if (day >= 1 && day <= 30) attendanceRecord[day - 1] = present;

}

int getTotalPresentDays() {

int count = 0;

for (boolean present : attendanceRecord) if (present) count++;

return count;

}

int getAllowedLeaves() {

switch (empType.toLowerCase()) {

case "full-time": return 4;

case "part-time": return 2;

case "contract": return 1;

default: return 2;

}

}

double calculateSalary() {

int presentDays = getTotalPresentDays();

double perDaySalary = baseSalary / workingDaysPerMonth;

double salary = perDaySalary \* presentDays;

double bonus = calculateBonus();

double total = salary + bonus;

totalSalaryExpense += total;

return total;

}

double calculateBonus() {

if (getTotalPresentDays() >= 28) return baseSalary \* 0.2;

else if (getTotalPresentDays() >= 25) return baseSalary \* 0.1;

else return 0;

}

void generatePaySlip() {

System.out.println("\n--- Pay Slip ---");

System.out.println("Company: " + companyName);

System.out.println("Employee: " + empName + " (" + empId + ")");

System.out.println("Designation: " + designation);

System.out.println("Base Salary: Rs." + baseSalary);

System.out.println("Present Days: " + getTotalPresentDays());

System.out.println("Bonus: Rs." + calculateBonus());

System.out.println("Net Salary: Rs." + calculateSalary());

}

void requestLeave(int leaveDays) {

if (leaveDays <= getAllowedLeaves()) System.out.println(empName + " leave approved for " + leaveDays + " days.");

else System.out.println(empName + " leave request denied! Allowed only " + getAllowedLeaves() + " days.");

}

}

class Department {

String deptId, deptName;

Employee manager;

Employee[] employees;

double budget;

Department(String deptId, String deptName, Employee manager, Employee[] employees, double budget) {

this.deptId = deptId;

this.deptName = deptName;

this.manager = manager;

this.employees = employees;

this.budget = budget;

}

double getDepartmentExpenses() {

double total = 0;

for (Employee e : employees) total += e.calculateSalary();

return total;

}

}

public class EmployeePayrollSystem {

static double calculateCompanyPayroll(Employee[] employees) {

double total = 0;

for (Employee e : employees) total += e.calculateSalary();

return total;

}

static void getDepartmentWiseExpenses(Department[] departments) {

for (Department d : departments)

System.out.println("Department: " + d.deptName + " | Expenses: Rs." + d.getDepartmentExpenses());

}

static void getAttendanceReport(Employee[] employees) {

for (Employee e : employees)

System.out.println(e.empName + ": Present " + e.getTotalPresentDays() + " days");

}

public static void main(String[] args) {

Employee e1 = new Employee("E001", "Aayush Rai", "IT", "Developer", 60000, "01-04-2024", "full-time");

Employee e2 = new Employee("E002", "Priya Sharma", "HR", "HR Manager", 50000, "15-03-2023", "full-time");

Employee e3 = new Employee("E003", "Rohan Gupta", "IT", "Tester", 30000, "12-02-2024", "part-time");

Employee[] employees = {e1, e2, e3};

Department d1 = new Department("D001", "IT", e1, new Employee[]{e1, e3}, 300000);

Department d2 = new Department("D002", "HR", e2, new Employee[]{e2}, 150000);

Department[] departments = {d1, d2};

e1.markAttendance(1, true);

e1.markAttendance(2, true);

e1.markAttendance(3, false);

e2.markAttendance(1, true);

e2.markAttendance(2, true);

e3.markAttendance(1, false);

e3.markAttendance(2, true);

e1.generatePaySlip();

e2.generatePaySlip();

e3.generatePaySlip();

System.out.println("\n--- Company Payroll ---");

System.out.println("Total Payroll: Rs." + calculateCompanyPayroll(employees));

System.out.println("\n--- Department Expenses ---");

getDepartmentWiseExpenses(departments);

System.out.println("\n--- Attendance Report ---");

getAttendanceReport(employees);

e1.requestLeave(3);

e3.requestLeave(3);

}

}

OUTPUT:

--- Pay Slip ---

Company: Tech Solutions Pvt Ltd

Employee: Aayush Rai (E001)

Designation: Developer

Base Salary: Rs.60000.0

Present Days: 2

Bonus: Rs.0.0

Net Salary: Rs.4000.0

--- Pay Slip ---

Company: Tech Solutions Pvt Ltd

Employee: Priya Sharma (E002)

Designation: HR Manager

Base Salary: Rs.50000.0

Present Days: 2

Bonus: Rs.0.0

Net Salary: Rs.3333.3333333333335

--- Pay Slip ---

Company: Tech Solutions Pvt Ltd

Employee: Rohan Gupta (E003)

Designation: Tester

Base Salary: Rs.30000.0

Present Days: 1

Bonus: Rs.0.0

Net Salary: Rs.1000.0

--- Company Payroll ---

Total Payroll: Rs.8333.333333333334

--- Department Expenses ---

Department: IT | Expenses: Rs.5000.0

Department: HR | Expenses: Rs.3333.3333333333335

--- Attendance Report ---

Aayush Rai: Present 2 days

Priya Sharma: Present 2 days

Rohan Gupta: Present 1 days

Aayush Rai leave approved for 3 days.

Rohan Gupta leave request denied! Allowed only 2 days.

6



**Assignment 7: Vehicle Fleet Management System Topic**: Inheritance Simulation and Resource Management

**Problem Statement**: Build a vehicle fleet management system for a transportation company. **Requirements**:

● Create a base Vehicle class with: vehicleId (String), brand (String), model (String), year (int), mileage (double), fuelType (String), currentStatus (String) ● Create specific vehicle types: Car, Bus, Truck classes with unique attributes (seatingCapacity for Bus, loadCapacity for Truck, etc.)

● Include static variables: totalVehicles (int), fleetValue (double), companyName (String), totalFuelConsumption (double)

● Implement methods: assignDriver(), scheduleMaintenance(), calculateRunningCost(), updateMileage(), checkServiceDue() ● Create a Driver class with: driverId, driverName, licenseType, assignedVehicle, totalTrips

● Include static methods: getFleetUtilization(),

calculateTotalMaintenanceCost(), getVehiclesByType()

● Implement trip management and fuel consumption tracking

**Deliverables**: Comprehensive fleet management system with vehicle tracking, driver assignment, and operational cost analysis.

SOLUTION:

import java.util.\*;

class Vehicle {

String vehicleId, brand, model, fuelType, currentStatus;

int year;

double mileage;

static int totalVehicles = 0;

static double fleetValue = 0;

static String companyName = "SRM Logistics";

static double totalFuelConsumption = 0;

Vehicle(String vehicleId, String brand, String model, int year, double mileage, String fuelType) {

this.vehicleId = vehicleId;

this.brand = brand;

this.model = model;

this.year = year;

this.mileage = mileage;

this.fuelType = fuelType;

this.currentStatus = "Available";

totalVehicles++;

}

void updateMileage(double newMileage) {

mileage = newMileage;

System.out.println("Updated mileage for " + vehicleId + ": " + mileage + " km");

}

void scheduleMaintenance() {

System.out.println("Maintenance scheduled for Vehicle ID: " + vehicleId);

}

boolean checkServiceDue() {

return mileage > 15000;

}

double calculateRunningCost(double fuelPricePerLitre, double fuelUsed) {

totalFuelConsumption += fuelUsed;

return fuelPricePerLitre \* fuelUsed;

}

}

class Car extends Vehicle {

int seatingCapacity;

Car(String vehicleId, String brand, String model, int year, double mileage, String fuelType, int seatingCapacity) {

super(vehicleId, brand, model, year, mileage, fuelType);

this.seatingCapacity = seatingCapacity;

}

}

class Bus extends Vehicle {

int seatingCapacity;

Bus(String vehicleId, String brand, String model, int year, double mileage, String fuelType, int seatingCapacity) {

super(vehicleId, brand, model, year, mileage, fuelType);

this.seatingCapacity = seatingCapacity;

}

}

class Truck extends Vehicle {

double loadCapacity;

Truck(String vehicleId, String brand, String model, int year, double mileage, String fuelType, double loadCapacity) {

super(vehicleId, brand, model, year, mileage, fuelType);

this.loadCapacity = loadCapacity;

}

}

class Driver {

String driverId, driverName, licenseType;

Vehicle assignedVehicle;

int totalTrips = 0;

Driver(String driverId, String driverName, String licenseType) {

this.driverId = driverId;

this.driverName = driverName;

this.licenseType = licenseType;

}

void assignVehicle(Vehicle vehicle) {

assignedVehicle = vehicle;

vehicle.currentStatus = "Assigned to " + driverName;

System.out.println(driverName + " assigned to vehicle: " + vehicle.vehicleId);

}

void completeTrip(double fuelUsed) {

if (assignedVehicle != null) {

assignedVehicle.mileage += fuelUsed \* 10; // Approx mileage increment

totalTrips++;

System.out.println(driverName + " completed a trip. Total trips: " + totalTrips);

} else {

System.out.println("Driver " + driverName + " has no assigned vehicle.");

}

}

}

public class VehicleFleetManagement {

static ArrayList<Vehicle> fleet = new ArrayList<>();

static ArrayList<Driver> drivers = new ArrayList<>();

static void addSampleData() {

fleet.add(new Car("C1", "Maruti", "Swift", 2022, 5000, "Petrol", 5));

fleet.add(new Bus("B1", "Volvo", "9400", 2021, 30000, "Diesel", 45));

fleet.add(new Truck("T1", "Tata", "Xenon", 2020, 45000, "Diesel", 8.5));

drivers.add(new Driver("D1", "Amit", "LMV"));

drivers.add(new Driver("D2", "Ravi", "HMV"));

}

static void displayFleet() {

System.out.println("\n---- Vehicle Fleet ----");

for (Vehicle v : fleet) {

System.out.println(v.vehicleId + " | " + v.brand + " " + v.model + " | " + v.currentStatus);

}

}

static void assignDriverToVehicle(String driverId, String vehicleId) {

Driver d = null;

Vehicle v = null;

for (Driver dr : drivers)

if (dr.driverId.equals(driverId)) d = dr;

for (Vehicle vh : fleet)

if (vh.vehicleId.equals(vehicleId)) v = vh;

if (d != null && v != null) d.assignVehicle(v);

else System.out.println("Invalid Driver or Vehicle ID!");

}

static void getFleetUtilization() {

long assigned = fleet.stream().filter(v -> !v.currentStatus.equals("Available")).count();

double utilization = ((double) assigned / fleet.size()) \* 100;

System.out.println("Fleet Utilization: " + utilization + "%");

}

public static void main(String[] args) {

addSampleData();

Scanner sc = new Scanner(System.in);

while (true) {

System.out.println("\n--- Vehicle Fleet Management ---");

System.out.println("1. Display Fleet");

System.out.println("2. Assign Driver");

System.out.println("3. Complete Trip");

System.out.println("4. Check Service Due");

System.out.println("5. Fleet Utilization");

System.out.println("6. Exit");

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

int ch = sc.nextInt();

switch (ch) {

case 1:

displayFleet();

break;

case 2:

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

String did = sc.next();

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

String vid = sc.next();

assignDriverToVehicle(did, vid);

break;

case 3:

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

String drId = sc.next();

System.out.print("Fuel used in litres: ");

double fuel = sc.nextDouble();

for (Driver d : drivers)

if (d.driverId.equals(drId)) d.completeTrip(fuel);

break;

case 4:

for (Vehicle v : fleet)

if (v.checkServiceDue())

System.out.println(v.vehicleId + " needs servicing.");

break;

case 5:

getFleetUtilization();

break;

case 6:

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

sc.close();

return;

default:

System.out.println("Invalid choice!");

}

}

}

}

OUTPUT:

--- Vehicle Fleet Management ---

1. Display Fleet

2. Assign Driver

3. Complete Trip

4. Check Service Due

5. Fleet Utilization

6. Exit

Enter choice: 1

---- Vehicle Fleet ----

C1 | Maruti Swift | Available

B1 | Volvo 9400 | Available

T1 | Tata Xenon | Available

--- Vehicle Fleet Management ---

1. Display Fleet

2. Assign Driver

3. Complete Trip

4. Check Service Due

5. Fleet Utilization

6. Exit

Enter choice: 2

Enter Driver ID: 101

Enter Vehicle ID: B1

Invalid Driver or Vehicle ID!

--- Vehicle Fleet Management ---

1. Display Fleet

2. Assign Driver

3. Complete Trip

4. Check Service Due

5. Fleet Utilization

6. Exit

Enter choice: 6

Exiting...

7



**Assignment 8: Hospital Patient Management System Topic**: Advanced Object Relationships and Data Management

**Problem Statement**: Develop a hospital patient management system with appointments, treatments, and billing.

**Requirements**:

● Create a Patient class with: patientId (String), patientName (String), age (int), gender (String), contactInfo (String), medicalHistory (String array), currentTreatments (String array)

● Create a Doctor class with: doctorId (String), doctorName (String), specialization (String), availableSlots (String array), patientsHandled (int), consultationFee (double)

● Create an Appointment class with: appointmentId (String), patient (Patient), doctor (Doctor), appointmentDate (String), appointmentTime (String), status (String)

● Include static variables: totalPatients (int), totalAppointments (int), hospitalName (String), totalRevenue (double)

● Implement methods: scheduleAppointment(), cancelAppointment(), generateBill(), updateTreatment(), dischargePatient()

● Create different appointment types (Consultation, Follow-up, Emergency) with different billing rates

● Include static methods: generateHospitalReport(), getDoctorUtilization(), getPatientStatistics()

● Implement patient history tracking and treatment management

**Deliverables**: Complete hospital management system with patient records, appointment scheduling, doctor management, and billing integration.

SOLUTION:

import java.util.\*;

class Patient {

String patientId, patientName, gender, contactInfo;

int age;

String[] medicalHistory;

String[] currentTreatments;

static int totalPatients = 0;

Patient(String patientId, String patientName, int age, String gender, String contactInfo) {

this.patientId = patientId;

this.patientName = patientName;

this.age = age;

this.gender = gender;

this.contactInfo = contactInfo;

this.medicalHistory = new String[10];

this.currentTreatments = new String[5];

totalPatients++;

}

}

class Doctor {

String doctorId, doctorName, specialization;

String[] availableSlots;

int patientsHandled;

double consultationFee;

Doctor(String doctorId, String doctorName, String specialization, double consultationFee) {

this.doctorId = doctorId;

this.doctorName = doctorName;

this.specialization = specialization;

this.consultationFee = consultationFee;

this.availableSlots = new String[] { "10:00 AM", "11:00 AM", "12:00 PM" };

this.patientsHandled = 0;

}

}

class Appointment {

String appointmentId;

Patient patient;

Doctor doctor;

String appointmentDate, appointmentTime, status;

String appointmentType;

double billAmount;

static int totalAppointments = 0;

static double totalRevenue = 0;

Appointment(String appointmentId, Patient patient, Doctor doctor, String date, String time, String type) {

this.appointmentId = appointmentId;

this.patient = patient;

this.doctor = doctor;

this.appointmentDate = date;

this.appointmentTime = time;

this.appointmentType = type;

this.status = "Scheduled";

this.billAmount = calculateBill();

doctor.patientsHandled++;

totalAppointments++;

totalRevenue += billAmount;

}

private double calculateBill() {

switch (appointmentType.toLowerCase()) {

case "consultation": return doctor.consultationFee;

case "follow-up": return doctor.consultationFee \* 0.5;

case "emergency": return doctor.consultationFee \* 2;

default: return doctor.consultationFee;

}

}

void cancelAppointment() {

if (!status.equals("Cancelled")) {

status = "Cancelled";

totalRevenue -= billAmount;

System.out.println("Appointment " + appointmentId + " cancelled successfully!");

} else {

System.out.println("Appointment is already cancelled!");

}

}

void generateBill() {

System.out.println("\n--- BILL DETAILS ---");

System.out.println("Appointment ID: " + appointmentId);

System.out.println("Patient: " + patient.patientName);

System.out.println("Doctor: " + doctor.doctorName);

System.out.println("Type: " + appointmentType);

System.out.println("Bill Amount: ₹" + billAmount);

}

}

public class HospitalManagementSystem {

static String hospitalName = "CityCare Hospital";

static ArrayList<Patient> patients = new ArrayList<>();

static ArrayList<Doctor> doctors = new ArrayList<>();

static ArrayList<Appointment> appointments = new ArrayList<>();

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Pre-adding doctors

doctors.add(new Doctor("D101", "Dr. Sharma", "Cardiology", 1000));

doctors.add(new Doctor("D102", "Dr. Verma", "Neurology", 1200));

doctors.add(new Doctor("D103", "Dr. Mehta", "Orthopedics", 900));

while (true) {

System.out.println("\n=== " + hospitalName + " Management System ===");

System.out.println("1. Add Patient");

System.out.println("2. Schedule Appointment");

System.out.println("3. Cancel Appointment");

System.out.println("4. Generate Bill");

System.out.println("5. Generate Hospital Report");

System.out.println("6. Exit");

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

int choice = sc.nextInt();

sc.nextLine();

switch (choice) {

case 1 -> addPatient(sc);

case 2 -> scheduleAppointment(sc);

case 3 -> cancelAppointment(sc);

case 4 -> generateBill(sc);

case 5 -> generateHospitalReport();

case 6 -> {

System.out.println("Exiting... Thank you!");

sc.close();

return;

}

default -> System.out.println("Invalid choice! Try again.");

}

}

}

static void addPatient(Scanner sc) {

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

String id = sc.nextLine();

System.out.print("Enter Patient Name: ");

String name = sc.nextLine();

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

int age = sc.nextInt();

sc.nextLine();

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

String gender = sc.nextLine();

System.out.print("Enter Contact Info: ");

String contact = sc.nextLine();

patients.add(new Patient(id, name, age, gender, contact));

System.out.println("Patient added successfully!");

}

static void scheduleAppointment(Scanner sc) {

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

String appId = sc.nextLine();

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

String pid = sc.nextLine();

Patient patient = findPatient(pid);

if (patient == null) {

System.out.println("Patient not found!");

return;

}

System.out.println("Available Doctors:");

for (int i = 0; i < doctors.size(); i++)

System.out.println((i + 1) + ". " + doctors.get(i).doctorName + " (" + doctors.get(i).specialization + ")");

System.out.print("Choose Doctor (1-" + doctors.size() + "): ");

int docChoice = sc.nextInt();

sc.nextLine();

Doctor doctor = doctors.get(docChoice - 1);

System.out.print("Enter Appointment Date (DD-MM-YYYY): ");

String date = sc.nextLine();

System.out.print("Enter Appointment Time: ");

String time = sc.nextLine();

System.out.print("Enter Appointment Type (Consultation/Follow-up/Emergency): ");

String type = sc.nextLine();

Appointment appointment = new Appointment(appId, patient, doctor, date, time, type);

appointments.add(appointment);

System.out.println("Appointment scheduled successfully!");

}

static void cancelAppointment(Scanner sc) {

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

String id = sc.nextLine();

Appointment app = findAppointment(id);

if (app != null) app.cancelAppointment();

else System.out.println("Appointment not found!");

}

static void generateBill(Scanner sc) {

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

String id = sc.nextLine();

Appointment app = findAppointment(id);

if (app != null) app.generateBill();

else System.out.println("Appointment not found!");

}

static void generateHospitalReport() {

System.out.println("\n=== HOSPITAL REPORT ===");

System.out.println("Hospital Name: " + hospitalName);

System.out.println("Total Patients: " + Patient.totalPatients);

System.out.println("Total Appointments: " + Appointment.totalAppointments);

System.out.println("Total Revenue: ₹" + Appointment.totalRevenue);

Doctor topDoctor = doctors.stream().max(Comparator.comparingInt(d -> d.patientsHandled)).orElse(null);

if (topDoctor != null)

System.out.println("Top Doctor: " + topDoctor.doctorName + " (" + topDoctor.patientsHandled + " patients)");

}

static Patient findPatient(String id) {

return patients.stream().filter(p -> p.patientId.equals(id)).findFirst().orElse(null);

}

static Appointment findAppointment(String id) {

return appointments.stream().filter(a -> a.appointmentId.equals(id)).findFirst().orElse(null);

}

}

OUTPUT:

=== CityCare Hospital Management System ===

1. Add Patient

2. Schedule Appointment

3. Cancel Appointment

4. Generate Bill

5. Generate Hospital Report

6. Exit

Enter choice: 1

Enter Patient ID: 101

Enter Patient Name: aayush

Enter Age: 19

Enter Gender: m

Enter Contact Info: 980000000

Patient added successfully!

=== CityCare Hospital Management System ===

1. Add Patient

2. Schedule Appointment

3. Cancel Appointment

4. Generate Bill

5. Generate Hospital Report

6. Exit

Enter choice: 2

Enter Appointment ID: 101

Enter Patient ID: 101

Available Doctors:

1. Dr. Sharma (Cardiology)

2. Dr. Verma (Neurology)

3. Dr. Mehta (Orthopedics)

Choose Doctor (1-3): 3

Enter Appointment Date (DD-MM-YYYY): 09-09-2025

Enter Appointment Time: 06:00

Enter Appointment Type (Consultation/Follow-up/Emergency): c

Appointment scheduled successfully!

=== CityCare Hospital Management System ===

1. Add Patient

2. Schedule Appointment

3. Cancel Appointment

4. Generate Bill

5. Generate Hospital Report

6. Exit

Enter choice: 6

Exiting... Thank you!

8



**Submission Guidelines**

1. **Code Organization**: Each assignment should have a main class and all required supporting classes

2. **Documentation**: Include JavaDoc comments for all classes and methods 3. **Testing**: Provide comprehensive test cases demonstrating all functionalities 4. **Output**: Include sample output showing the program execution with different scenarios 5. **Error Handling**: Implement proper exception handling for invalid inputs and edge cases 6. **Design**: Use proper OOP principles, including encapsulation, static/instance member usage, and object relationships

7. **Deadline**: Each assignment should be completed within one week of the assignment 8. **Format**: Submit as .java files with proper naming conventions

(Assignment1\_StudentName.java, etc.)

9