```java
public class PersonalFinanceManager {

    static 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";

        // Constructor
        public PersonalAccount(String accountHolderName) {
            this.accountHolderName = accountHolderName;
            this.accountNumber = generateAccountNumber();
            this.currentBalance = 0.0;
            this.totalIncome = 0.0;
            this.totalExpenses = 0.0;
            totalAccounts++;
        }

        // Static method to generate account number
        public static String generateAccountNumber() {
            return String.format("AC%03d", totalAccounts + 1);
        }

        // Static method to set bank name
        public static void setBankName(String name) {
            bankName = name;
        }

        // Static method to get total accounts
        public static int getTotalAccounts() {
            return totalAccounts;
        }

        // Add income
        public void addIncome(double amount, String description) {
            if (amount <= 0) {
```

```
                System.out.println("Invalid income amount.");
                return;
            }
            currentBalance += amount;
            totalIncome += amount;
            System.out.println(description + ": ₹" + amount + " added to "
+ accountNumber);
        }

        // Add expense
        public void addExpense(double amount, String description) {
            if (amount <= 0) {
                System.out.println("Invalid expense amount.");
                return;
            }
            if (amount > currentBalance) {
                System.out.println("Insufficient balance for expense: " +
description);
                return;
            }
            currentBalance -= amount;
            totalExpenses += amount;
            System.out.println(description + ": ₹" + amount + " deducted
from " + accountNumber);
        }

        // Calculate savings
        public double calculateSavings() {
            return totalIncome - totalExpenses;
        }

        // Display account summary
        public void displayAccountSummary() {
            System.out.println("=== Account Summary ===");
            System.out.println("Bank Name       : " + bankName);
            System.out.println("Account Number  : " + accountNumber);
            System.out.println("Holder Name     : " + accountHolderName);
            System.out.println("Current Balance : ₹" +
String.format("%.2f", currentBalance));
```

```
            System.out.println("Total Income    : ₹" +
String.format("%.2f", totalIncome));
            System.out.println("Total Expenses  : ₹" +
String.format("%.2f", totalExpenses));
            System.out.println("Savings         : ₹" +
String.format("%.2f", calculateSavings()));
            System.out.println("========================\n");
        }
    }

    public static void main(String[] args) {
        // Set bank name
        PersonalAccount.setBankName("Unity Bank");

        // Create accounts
        PersonalAccount acc1 = new PersonalAccount("Alice");
        PersonalAccount acc2 = new PersonalAccount("Bob");
        PersonalAccount acc3 = new PersonalAccount("Charlie");

        // Perform transactions
        acc1.addIncome(10000, "Salary");
        acc1.addExpense(2500, "Rent");
        acc1.addExpense(1200, "Groceries");

        acc2.addIncome(8000, "Freelance");
        acc2.addExpense(3000, "Laptop Purchase");

        acc3.addIncome(15000, "Consulting");
        acc3.addExpense(5000, "Travel");
        acc3.addExpense(2000, "Dining");

        // Display summaries
        acc1.displayAccountSummary();
        acc2.displayAccountSummary();
        acc3.displayAccountSummary();

        // Show static vs instance
        System.out.println("Total Accounts Created : " +
PersonalAccount.getTotalAccounts());
```

```
        System.out.println("Bank Name (Shared)      : " +
PersonalAccount.bankName);
    }
}
```

```
Compiling PersonalFinanceManager.java...
Compilation successful. Running program...

Salary: ?10000.0 added to AC001
Rent: ?2500.0 deducted from AC001
Groceries: ?1200.0 deducted from AC001
Freelance: ?8000.0 added to AC002
Laptop Purchase: ?3000.0 deducted from AC002
Consulting: ?15000.0 added to AC003
Travel: ?5000.0 deducted from AC003
Dining: ?2000.0 deducted from AC003
=== Account Summary ===
Bank Name       : Unity Bank
Account Number  : AC001
Holder Name     : Alice
Current Balance : ?6300.00
Total Income    : ?10000.00
Total Expenses  : ?3700.00
Savings         : ?6300.00
========================

=== Account Summary ===
Bank Name       : Unity Bank
Account Number  : AC002
Holder Name     : Bob
Current Balance : ?5000.00
Total Income    : ?8000.00
Total Expenses  : ?3000.00
Savings         : ?5000.00
========================

=== Account Summary ===
Bank Name       : Unity Bank
Account Number  : AC003
Holder Name     : Charlie
Current Balance : ?8000.00
Total Income    : ?15000.00
Total Expenses  : ?7000.00
Savings         : ?8000.00
========================

Total Accounts Created : 3
Bank Name (Shared)     : Unity Bank
```

```java
//  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.


import java.util.Scanner;

public class OnlineShoppingCartSystem {

    static class Product {
```

```
        private String productId;
        private String productName;
        private double price;
        private String category;
        private int stockQuantity;

        private static int totalProducts = 0;
        private static String[] categories = {"Electronics", "Clothing",
"Books", "Home", "Toys"};

        // Constructor
        public Product(String productName, double price, String category,
int stockQuantity) {
            this.productName = productName;
            this.price = price;
            this.category = category;
            this.stockQuantity = stockQuantity;
            this.productId = generateProductId();
            totalProducts++;
        }

        // Static method to generate product ID
        public static String generateProductId() {
            return String.format("P%03d", totalProducts + 1);
        }

        // Static method to find product by ID
        public static Product findProductById(Product[] products, String
productId) {
            for (Product p : products) {
                if (p != null && p.productId.equals(productId)) {
                    return p;
                }
            }
            return null;
        }

        // Static method to get products by category
        public static void getProductsByCategory(Product[] products,
String category) {
```

```
            System.out.println("Products in category: " + category);
            for (Product p : products) {
                if (p != null && p.category.equalsIgnoreCase(category)) {
                    System.out.println(p.productId + " - " + p.productName
+ " - ₹" + p.price);
                }
            }
        }

        public void reduceStock(int quantity) {
            stockQuantity -= quantity;
        }

        public void increaseStock(int quantity) {
            stockQuantity += quantity;
        }

        public int getStockQuantity() {
            return stockQuantity;
        }

        public String getProductId() {
            return productId;
        }

        public String getProductName() {
            return productName;
        }

        public double getPrice() {
            return price;
        }

        public void displayProduct() {
            System.out.println(productId + " | " + productName + " | ₹" +
price + " | " + category + " | Stock: " + stockQuantity);
        }
    }

    static class ShoppingCart {
```

```
        private String cartId;
        private String customerName;
        private Product[] products;
        private int[] quantities;
        private double cartTotal;
        private int itemCount;

        // Constructor
        public ShoppingCart(String customerName) {
            this.customerName = customerName;
            this.cartId = generateCartId();
            this.products = new Product[20];
            this.quantities = new int[20];
            this.cartTotal = 0.0;
            this.itemCount = 0;
        }

        public static String generateCartId() {
            return "CART" + System.currentTimeMillis();
        }

        public void addProduct(Product product, int quantity) {
            if (product.getStockQuantity() < quantity) {
                System.out.println("Not enough stock for " +
product.getProductName());
                return;
            }
            products[itemCount] = product;
            quantities[itemCount] = quantity;
            product.reduceStock(quantity);
            itemCount++;
            calculateTotal();
            System.out.println("Added " + quantity + " x " +
product.getProductName() + " to cart.");
        }

        public void removeProduct(String productId) {
            for (int i = 0; i < itemCount; i++) {
                if (products[i].getProductId().equals(productId)) {
                    products[i].increaseStock(quantities[i]);
```

```
                    System.out.println("Removed " +
products[i].getProductName() + " from cart.");
                    products[i] = null;
                    quantities[i] = 0;
                    // Shift remaining items
                    for (int j = i; j < itemCount - 1; j++) {
                        products[j] = products[j + 1];
                        quantities[j] = quantities[j + 1];
                    }
                    products[itemCount - 1] = null;
                    quantities[itemCount - 1] = 0;
                    itemCount--;
                    calculateTotal();
                    return;
                }
            }
            System.out.println("Product not found in cart.");
        }

        public void calculateTotal() {
            cartTotal = 0.0;
            for (int i = 0; i < itemCount; i++) {
                cartTotal += products[i].getPrice() * quantities[i];
            }
        }

        public void displayCart() {
            System.out.println("=== Cart Summary for " + customerName + "
===");
            for (int i = 0; i < itemCount; i++) {
                System.out.println(products[i].getProductName() + " x " +
quantities[i] + " = ₹" + (products[i].getPrice() * quantities[i]));
            }
            System.out.println("Total: ₹" + cartTotal);
            System.out.println("==============================");
        }

        public void checkout() {
            displayCart();
```

```
            System.out.println("Checkout complete. Thank you for
shopping!");
            itemCount = 0;
            cartTotal = 0.0;
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // Create products
        Product[] products = new Product[10];
        products[0] = new Product("Laptop", 55000, "Electronics", 5);
        products[1] = new Product("T-Shirt", 499, "Clothing", 20);
        products[2] = new Product("Book - Java", 799, "Books", 15);
        products[3] = new Product("Mixer Grinder", 2999, "Home", 10);
        products[4] = new Product("Toy Car", 399, "Toys", 25);
        products[5] = new Product("Smartphone", 15000, "Electronics", 8);
        products[6] = new Product("Jeans", 999, "Clothing", 12);
        products[7] = new Product("Cookbook", 599, "Books", 10);
        products[8] = new Product("Wall Clock", 799, "Home", 7);
        products[9] = new Product("Puzzle Game", 299, "Toys", 30);

        ShoppingCart cart = new ShoppingCart("Alice");

        while (true) {
            System.out.println("\n=== Online Shopping Menu ===");
            System.out.println("1. View All Products");
            System.out.println("2. View Products by Category");
            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("Choose an option: ");
            int choice = sc.nextInt();
            sc.nextLine(); // consume newline

            switch (choice) {
                case 1:
```

```
                    for (Product p : products) {
                        p.displayProduct();
                    }
                    break;
                case 2:
                    System.out.print("Enter category: ");
                    String cat = sc.nextLine();
                    Product.getProductsByCategory(products, cat);
                    break;
                case 3:
                    System.out.print("Enter Product ID: ");
                    String pid = sc.nextLine();
                    Product prod = Product.findProductById(products, pid);
                    if (prod != null) {
                        System.out.print("Enter quantity: ");
                        int qty = sc.nextInt();
                        cart.addProduct(prod, qty);
                    } else {
                        System.out.println("Product not found.");
                    }
                    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!");
                    return;
                default:
                    System.out.println("Invalid choice.");
            }
        }
    }
```

```
}
```

```
Compilation successful. Running program...


=== Online Shopping Menu ===
1. View All Products
2. View Products by Category
3. Add Product to Cart
4. Remove Product from Cart
5. View Cart
6. Checkout
7. Exit
Choose an option: 1
P001 | Laptop | ?55000.0 | Electronics | Stock: 5
P002 | T-Shirt | ?499.0 | Clothing | Stock: 20
P003 | Book - Java | ?799.0 | Books | Stock: 15
P004 | Mixer Grinder | ?2999.0 | Home | Stock: 10
P005 | Toy Car | ?399.0 | Toys | Stock: 25
P006 | Smartphone | ?15000.0 | Electronics | Stock: 8
P007 | Jeans | ?999.0 | Clothing | Stock: 12
P008 | Cookbook | ?599.0 | Books | Stock: 10
P009 | Wall Clock | ?799.0 | Home | Stock: 7
P010 | Puzzle Game | ?299.0 | Toys | Stock: 30

=== Online Shopping Menu ===
1. View All Products
2. View Products by Category
3. Add Product to Cart
4. Remove Product from Cart
5. View Cart
6. Checkout
7. Exit
Choose an option: 7
Exiting... Thank you!

Program finished. Cleaning up...
```

```
//  Hotel Reservation System
// Topic: Multiple Classes with Complex Interactions
// Problem Statement: Build a hotel reservation management system handling
rooms, guests,
```

```java
// 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.


import java.util.*;

public class HotelReservationSystem {

    static 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, int maxOccupancy) {
```

```
            this.roomNumber = roomNumber;
            this.roomType = roomType;
            this.pricePerNight = pricePerNight;
            this.maxOccupancy = maxOccupancy;
            this.isAvailable = true;
        }

        public boolean isAvailable() {
            return isAvailable;
        }

        public void setAvailability(boolean status) {
            isAvailable = status;
        }

        public String getRoomType() {
            return roomType;
        }

        public String getRoomNumber() {
            return roomNumber;
        }

        public double getPricePerNight() {
            return pricePerNight;
        }

        public void displayRoomInfo() {
            System.out.println("Room " + roomNumber + " | Type: " +
roomType + " | ₹" + pricePerNight + " | Available: " + isAvailable);
        }
    }

    static class Guest {
        private String guestId;
        private String guestName;
        private String phoneNumber;
        private String email;
        private String[] bookingHistory;
        private int bookingCount;
```

```
        public Guest(String guestName, String phoneNumber, String email) {
            this.guestName = guestName;
            this.phoneNumber = phoneNumber;
            this.email = email;
            this.guestId = generateGuestId();
            this.bookingHistory = new String[10];
            this.bookingCount = 0;
        }

        public String getGuestId() {
            return guestId;
        }

        public String getGuestName() {
            return guestName;
        }

        public void addBooking(String bookingId) {
            if (bookingCount < bookingHistory.length) {
                bookingHistory[bookingCount++] = bookingId;
            }
        }

        public void displayGuestInfo() {
            System.out.println("Guest ID: " + guestId + " | Name: " +
guestName + " | Phone: " + phoneNumber + " | Email: " + email);
        }

        private static int guestCounter = 0;

        public static String generateGuestId() {
            return String.format("G%03d", ++guestCounter);
        }
    }

    static 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 = 0.0;
        private static String hotelName = "Tranquil Stay";

        public Booking(Guest guest, Room room, String checkInDate, String
checkOutDate, int nights) {
            this.bookingId = generateBookingId();
            this.guest = guest;
            this.room = room;
            this.checkInDate = checkInDate;
            this.checkOutDate = checkOutDate;
            this.totalAmount = calculateBill(nights);
            totalBookings++;
            hotelRevenue += totalAmount;
            room.setAvailability(false);
            guest.addBooking(bookingId);
        }

        public static String generateBookingId() {
            return String.format("B%03d", totalBookings + 1);
        }

        public double calculateBill(int nights) {
            return room.getPricePerNight() * nights;
        }

        public void cancelReservation() {
            room.setAvailability(true);
            hotelRevenue -= totalAmount;
            System.out.println("Booking " + bookingId + " cancelled.");
        }

        public void displayBookingInfo() {
            System.out.println("Booking ID: " + bookingId + " | Guest: " +
guest.getGuestName() +
```

```
                    " | Room: " + room.getRoomNumber() + " | ₹" +
totalAmount +
                    " | Check-in: " + checkInDate + " | Check-out: " +
checkOutDate);
        }

        public static double getTotalRevenue() {
            return hotelRevenue;
        }

        public static double getOccupancyRate(Room[] rooms) {
            int occupied = 0;
            for (Room r : rooms) {
                if (!r.isAvailable()) occupied++;
            }
            return (double) occupied / rooms.length * 100;
        }

        public static String getMostPopularRoomType(Room[] rooms) {
            Map<String, Integer> typeCount = new HashMap<>();
            for (Room r : rooms) {
                if (!r.isAvailable()) {
                    typeCount.put(r.getRoomType(),
typeCount.getOrDefault(r.getRoomType(), 0) + 1);
                }
            }
            return typeCount.entrySet().stream()
                    .max(Map.Entry.comparingByValue())
                    .map(Map.Entry::getKey)
                    .orElse("N/A");
        }

        public static void setHotelName(String name) {
            hotelName = name;
        }

        public static String getHotelName() {
            return hotelName;
        }
    }
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        Room[] rooms = {
            new Room("101", "Deluxe", 2500, 2),
            new Room("102", "Standard", 1800, 2),
            new Room("103", "Suite", 4000, 4),
            new Room("104", "Standard", 1800, 2),
            new Room("105", "Deluxe", 2500, 2)
        };

        Guest guest1 = new Guest("Alice", "9876543210",
"alice@example.com");
        Guest guest2 = new Guest("Bob", "9123456780", "bob@example.com");

        Booking.setHotelName("Ocean View Resort");

        // Make reservations
        Booking booking1 = new Booking(guest1, rooms[0], "2025-08-28",
"2025-08-30", 2);
        Booking booking2 = new Booking(guest2, rooms[2], "2025-08-29",
"2025-09-01", 3);

        // Display info
        System.out.println("=== Hotel: " + Booking.getHotelName() + "
===");
        booking1.displayBookingInfo();
        booking2.displayBookingInfo();

        System.out.println("\n--- Room Status ---");
        for (Room r : rooms) r.displayRoomInfo();

        System.out.println("\n--- Guest Info ---");
        guest1.displayGuestInfo();
        guest2.displayGuestInfo();

        System.out.println("\n--- Hotel Reports ---");
        System.out.println("Total Revenue: ₹" +
Booking.getTotalRevenue());
```

```
        System.out.println("Occupancy Rate: " +
Booking.getOccupancyRate(rooms) + "%");
        System.out.println("Most Popular Room Type: " +
Booking.getMostPopularRoomType(rooms));
    }
}
```

```
Compilation successful. Running program...

=== Hotel: Ocean View Resort ===
Booking ID: B001 | Guest: Alice | Room: 101 | ?5000.0 | Check-in: 2025-08-28 | Check-out: 2025-08-30
Booking ID: B002 | Guest: Bob | Room: 103 | ?12000.0 | Check-in: 2025-08-29 | Check-out: 2025-09-01

--- Room Status ---
Room 101 | Type: Deluxe | ?2500.0 | Available: false
Room 102 | Type: Standard | ?1800.0 | Available: true
Room 103 | Type: Suite | ?4000.0 | Available: false
Room 104 | Type: Standard | ?1800.0 | Available: true
Room 105 | Type: Deluxe | ?2500.0 | Available: true

--- Guest Info ---
Guest ID: G001 | Name: Alice | Phone: 9876543210 | Email: alice@example.com
Guest ID: G002 | Name: Bob | Phone: 9123456780 | Email: bob@example.com

--- Hotel Reports ---
Total Revenue: ?17000.0
Occupancy Rate: 40.0%
Most Popular Room Type: Suite
```

```
// 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)
```

```java
// • 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.



import java.util.*;

public class StudentGradeManagementSystem {

    static 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;
        }
    }

    static class Student {
        private String studentId;
```

```
        private String studentName;
        private String className;
        private String[] subjects;
        private double[][] marks; // [subject][marks]
        private double gpa;

        private static int totalStudents = 0;
        private static String schoolName = "Greenfield High";
        private static String[] gradingScale = {"A", "B", "C", "D", "F"};
        private static double passPercentage = 40.0;

        public Student(String studentName, String className, String[]
subjects) {
            this.studentName = studentName;
            this.className = className;
            this.subjects = subjects;
            this.studentId = generateStudentId();
            this.marks = new double[subjects.length][];
            totalStudents++;
        }

        public static String generateStudentId() {
            return String.format("S%03d", totalStudents + 1);
        }

        public void addMarks(String subject, double[] subjectMarks) {
            for (int i = 0; i < subjects.length; i++) {
                if (subjects[i].equalsIgnoreCase(subject)) {
                    marks[i] = subjectMarks;
                    return;
                }
            }
            System.out.println("Subject not found for student " +
studentName);
        }

        public void calculateGPA() {
            double total = 0;
            int count = 0;
            for (double[] subjectMarks : marks) {
```

```
                if (subjectMarks != null) {
                    double avg =
Arrays.stream(subjectMarks).average().orElse(0);
                    total += avg;
                    count++;
                }
            }
            gpa = count > 0 ? total / count : 0;
        }

        public void generateReportCard() {
            System.out.println("=== Report Card ===");
            System.out.println("Student ID   : " + studentId);
            System.out.println("Name         : " + studentName);
            System.out.println("Class        : " + className);
            for (int i = 0; i < subjects.length; i++) {
                if (marks[i] != null) {
                    double avg =
Arrays.stream(marks[i]).average().orElse(0);
                    String grade = getGrade(avg);
                    System.out.println(subjects[i] + " - Avg: " +
String.format("%.2f", avg) + " Grade: " + grade);
                } else {
                    System.out.println(subjects[i] + " - No marks
entered");
                }
            }
            System.out.println("GPA          : " + String.format("%.2f",
gpa));
            System.out.println("Promotion    : " +
(checkPromotionEligibility() ? "Eligible" : "Not Eligible"));
            System.out.println("====================\n");
        }

        public boolean checkPromotionEligibility() {
            for (double[] subjectMarks : marks) {
                if (subjectMarks != null) {
                    double avg =
Arrays.stream(subjectMarks).average().orElse(0);
                    if (avg < passPercentage) return false;
```

```
                }
            }
            return true;
        }

        private String getGrade(double percentage) {
            if (percentage >= 90) return gradingScale[0]; // A
            else if (percentage >= 75) return gradingScale[1]; // B
            else if (percentage >= 60) return gradingScale[2]; // C
            else if (percentage >= 40) return gradingScale[3]; // D
            else return gradingScale[4]; // F
        }

        public static void setGradingScale(String[] scale) {
            gradingScale = scale;
        }

        public static double calculateClassAverage(Student[] students) {
            double total = 0;
            int count = 0;
            for (Student s : students) {
                s.calculateGPA();
                total += s.gpa;
                count++;
            }
            return count > 0 ? total / count : 0;
        }

        public static Student[] getTopPerformers(Student[] students, int
count) {
            Arrays.sort(students, (a, b) -> Double.compare(b.gpa, a.gpa));
            return Arrays.copyOfRange(students, 0, Math.min(count,
students.length));
        }

        public static void generateSchoolReport(Student[] students) {
            System.out.println("=== School Report ===");
            System.out.println("School Name       : " + schoolName);
            System.out.println("Total Students    : " + totalStudents);
```

```
            System.out.println("Class Average GPA : " +
String.format("%.2f", calculateClassAverage(students)));
            Student[] toppers = getTopPerformers(students, 3);
            System.out.println("Top Performers:");
            for (Student s : toppers) {
                System.out.println(s.studentName + " - GPA: " +
String.format("%.2f", s.gpa));
            }
            System.out.println("======================\n");
        }
    }

    public static void main(String[] args) {
        String[] subjects = {"Math", "Science", "English"};

        Student s1 = new Student("Alice", "10A", subjects);
        Student s2 = new Student("Bob", "10A", subjects);
        Student s3 = new Student("Charlie", "10A", subjects);

        s1.addMarks("Math", new double[]{85, 90});
        s1.addMarks("Science", new double[]{78, 82});
        s1.addMarks("English", new double[]{88, 91});

        s2.addMarks("Math", new double[]{65, 70});
        s2.addMarks("Science", new double[]{60, 62});
        s2.addMarks("English", new double[]{75, 80});

        s3.addMarks("Math", new double[]{95, 98});
        s3.addMarks("Science", new double[]{92, 94});
        s3.addMarks("English", new double[]{89, 90});

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

        for (Student s : students) {
            s.calculateGPA();
            s.generateReportCard();
        }

        Student.generateSchoolReport(students);
    }
```

```
}
```

```
=== Report Card ===
Student ID   : S001
Name         : Alice
Class        : 10A
Math - Avg: 87.50 Grade: B
Science - Avg: 80.00 Grade: B
English - Avg: 89.50 Grade: B
GPA          : 85.67
Promotion    : Eligible
====================

=== Report Card ===
Student ID   : S002
Name         : Bob
Class        : 10A
Math - Avg: 67.50 Grade: C
Science - Avg: 61.00 Grade: C
English - Avg: 77.50 Grade: B
GPA          : 68.67
Promotion    : Eligible
====================

=== Report Card ===
Student ID   : S003
Name         : Charlie
Class        : 10A
Math - Avg: 96.50 Grade: A
Science - Avg: 93.00 Grade: A
English - Avg: 89.50 Grade: B
GPA          : 93.00
Promotion    : Eligible
====================

=== School Report ===
School Name       : Greenfield High
Total Students    : 3
Class Average GPA : 82.44
Top Performers:
Charlie - GPA: 93.00
Alice - GPA: 85.67
Bob - GPA: 68.67
======================
```

```java
// 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.


import java.util.*;

public class LibraryManagementSystem {

    static class Book {
        String bookId, title, author, isbn, category;
        boolean isIssued;
```

```
        String issueDate, dueDate;

        static int totalBooks = 0;

        public Book(String title, String author, String isbn, String
category) {
            this.bookId = generateBookId();
            this.title = title;
            this.author = author;
            this.isbn = isbn;
            this.category = category;
            this.isIssued = false;
            this.issueDate = "";
            this.dueDate = "";
            totalBooks++;
        }

        public static String generateBookId() {
            return "B" + (totalBooks + 1);
        }

        public void displayBookInfo() {
            System.out.println(bookId + " | " + title + " | " + author + "
| " + category + " | Issued: " + isIssued);
        }
    }

    static class Member {
        String memberId, memberName, memberType, membershipDate;
        Book[] booksIssued;
        double totalFines;

        static int totalMembers = 0;
        static String libraryName = "Central Library";
        static double finePerDay = 2.0;
        static int maxBooksAllowed = 3;

        public Member(String memberName, String memberType, String
membershipDate) {
            this.memberId = generateMemberId();
```

```
            this.memberName = memberName;
            this.memberType = memberType;
            this.membershipDate = membershipDate;
            this.booksIssued = new Book[maxBooksAllowed];
            this.totalFines = 0.0;
            totalMembers++;
        }

        public static String generateMemberId() {
            return "M" + (totalMembers + 1);
        }

        public void issueBook(Book book, String issueDate, String dueDate)
{
            if (book.isIssued) {
                System.out.println("Book already issued.");
                return;
            }
            for (int i = 0; i < booksIssued.length; i++) {
                if (booksIssued[i] == null) {
                    booksIssued[i] = book;
                    book.isIssued = true;
                    book.issueDate = issueDate;
                    book.dueDate = dueDate;
                    System.out.println("Book " + book.bookId + " issued to
" + memberName);
                    return;
                }
            }
            System.out.println("Max book limit reached for " +
memberType);
        }

        public void returnBook(String bookId, String returnDate) {
            for (int i = 0; i < booksIssued.length; i++) {
                if (booksIssued[i] != null &&
booksIssued[i].bookId.equals(bookId)) {
                    double fine = calculateFine(booksIssued[i].dueDate,
returnDate);
                    totalFines += fine;
```

```
                    booksIssued[i].isIssued = false;
                    booksIssued[i].issueDate = "";
                    booksIssued[i].dueDate = "";
                    System.out.println("Book " + bookId + " returned.
Fine: ₹" + fine);
                    booksIssued[i] = null;
                    return;
                }
            }
            System.out.println("Book not found in issued list.");
        }

        public 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[0]), Integer.parseInt(d1[1]) -
1, Integer.parseInt(d1[2]));
                ret.set(Integer.parseInt(d2[0]), Integer.parseInt(d2[1]) -
1, Integer.parseInt(d2[2]));
                long diff = (ret.getTimeInMillis() -
due.getTimeInMillis()) / (1000 * 60 * 60 * 24);
                return diff > 0 ? diff * finePerDay : 0;
            } catch (Exception e) {
                return 0;
            }
        }

        public void renewBook(String bookId, String newDueDate) {
            for (Book b : booksIssued) {
                if (b != null && b.bookId.equals(bookId)) {
                    b.dueDate = newDueDate;
                    System.out.println("Book " + bookId + " renewed. New
due date: " + newDueDate);
                    return;
                }
            }
            System.out.println("Book not found for renewal.");
```

```
        }

        public void displayMemberInfo() {
            System.out.println("Member ID: " + memberId + " | Name: " +
memberName + " | Type: " + memberType + " | Fines: ₹" + totalFines);
        }

        public static void generateLibraryReport(Book[] books, Member[]
members) {
            System.out.println("=== " + libraryName + " Report ===");
            System.out.println("Total Books   : " + Book.totalBooks);
            System.out.println("Total Members : " + totalMembers);
            double totalFine = 0;
            for (Member m : members) totalFine += m.totalFines;
            System.out.println("Total Fines   : ₹" + totalFine);
            System.out.println("==============================");
        }

        public static void getOverdueBooks(Book[] books, String
currentDate) {
            System.out.println("Overdue Books as of " + currentDate +
":");
            for (Book b : books) {
                if (b.isIssued && compareDates(currentDate, b.dueDate) >
0) {
                    System.out.println(b.bookId + " - " + b.title + "
(Due: " + b.dueDate + ")");
                }
            }
        }

        public static int compareDates(String d1, String d2) {
            try {
                String[] a = d1.split("-");
                String[] b = d2.split("-");
                Calendar c1 = Calendar.getInstance();
                Calendar c2 = Calendar.getInstance();
                c1.set(Integer.parseInt(a[0]), Integer.parseInt(a[1]) - 1,
Integer.parseInt(a[2]));
```

```
                c2.set(Integer.parseInt(b[0]), Integer.parseInt(b[1]) - 1,
Integer.parseInt(b[2]));
                return c1.compareTo(c2);
            } catch (Exception e) {
                return 0;
            }
        }

        public static void getMostPopularBooks(Book[] books) {
            System.out.println("Popular Books (Issued):");
            for (Book b : books) {
                if (b.isIssued) {
                    System.out.println(b.bookId + " - " + b.title);
                }
            }
        }
    }

    public static void main(String[] args) {
        Book[] books = {
            new Book("Java Basics", "Herbert Schildt", "ISBN001",
"Programming"),
            new Book("Data Structures", "Mark Allen", "ISBN002", "CS"),
            new Book("Operating Systems", "Galvin", "ISBN003", "CS")
        };

        Member m1 = new Member("Alice", "Student", "2025-01-10");
        Member m2 = new Member("Bob", "Faculty", "2025-02-15");

        m1.issueBook(books[0], "2025-08-01", "2025-08-10");
        m2.issueBook(books[1], "2025-08-05", "2025-08-15");

        m1.returnBook("B1", "2025-08-12"); // 2 days late
        m2.renewBook("B2", "2025-08-20");

        m1.displayMemberInfo();
        m2.displayMemberInfo();

        Member.generateLibraryReport(books, new Member[]{m1, m2});
        Member.getOverdueBooks(books, "2025-08-21");
```

```
        Member.getMostPopularBooks(books);
    }
}
```

```
Book B1 issued to Alice
Book B2 issued to Bob
Book B1 returned. Fine: ?4.0
Book B2 renewed. New due date: 2025-08-20
Member ID: M1 | Name: Alice | Type: Student | Fines: ?4.0
Member ID: M2 | Name: Bob | Type: Faculty | Fines: ?0.0
=== Central Library Report ===
Total Books   : 3
Total Members : 2
Total Fines   : ?4.0
============================
Overdue Books as of 2025-08-21:
B2 - Data Structures (Due: 2025-08-20)
Popular Books (Issued):
B2 - Data Structures

Program finished. Cleaning up...
LibraryManagementSystem.class file deleted successfully.
```

```
//  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)
```

```java
// • 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.


import java.util.*;

public class EmployeePayrollAttendanceSystem {

    static class Employee {
        String empId, empName, department, designation, joinDate;
        double baseSalary;
        boolean[] attendanceRecord;
        String empType;

        static int totalEmployees = 0;
        static String companyName = "TechNova Solutions";
        static double totalSalaryExpense = 0.0;
        static int workingDaysPerMonth = 30;

        public Employee(String empName, String department, String
designation, double baseSalary, String joinDate, String empType) {
            this.empId = generateEmpId();
            this.empName = empName;
            this.department = department;
            this.designation = designation;
            this.baseSalary = baseSalary;
            this.joinDate = joinDate;
            this.empType = empType;
            this.attendanceRecord = new boolean[workingDaysPerMonth];
            totalEmployees++;
```

```
        }

        public static String generateEmpId() {
            return "EMP" + (totalEmployees + 1);
        }

        public void markAttendance(int day, boolean present) {
            if (day >= 1 && day <= workingDaysPerMonth) {
                attendanceRecord[day - 1] = present;
            }
        }

        public double calculateSalary() {
            int presentDays = 0;
            for (boolean present : attendanceRecord) {
                if (present) presentDays++;
            }

            double salary = 0;
            switch (empType) {
                case "Full-time":
                    salary = baseSalary;
                    break;
                case "Part-time":
                    salary = (baseSalary / workingDaysPerMonth) *
presentDays;
                    break;
                case "Contract":
                    salary = baseSalary; // fixed
                    break;
            }

            totalSalaryExpense += salary;
            return salary;
        }

        public double calculateBonus() {
            int presentDays = 0;
            for (boolean present : attendanceRecord) {
                if (present) presentDays++;
```

```
            }
            double attendanceRate = (double) presentDays /
workingDaysPerMonth;
            return attendanceRate >= 0.9 ? 0.10 * baseSalary : 0.0;
        }

        public void generatePaySlip() {
            double salary = calculateSalary();
            double bonus = calculateBonus();
            System.out.println("=== Pay Slip ===");
            System.out.println("Employee ID   : " + empId);
            System.out.println("Name          : " + empName);
            System.out.println("Department    : " + department);
            System.out.println("Designation   : " + designation);
            System.out.println("Type          : " + empType);
            System.out.println("Base Salary   : ₹" + baseSalary);
            System.out.println("Bonus         : ₹" + bonus);
            System.out.println("Total Pay     : ₹" + (salary + bonus));
            System.out.println("================\n");
        }

        public void requestLeave(int day) {
            if (day >= 1 && day <= workingDaysPerMonth) {
                attendanceRecord[day - 1] = false;
                System.out.println(empName + " requested leave on day " +
day);
            }
        }

        public static double calculateCompanyPayroll(Employee[] employees)
{
            totalSalaryExpense = 0;
            for (Employee e : employees) {
                e.calculateSalary();
            }
            return totalSalaryExpense;
        }

        public static void getAttendanceReport(Employee[] employees) {
            System.out.println("=== Attendance Report ===");
```

```
            for (Employee e : employees) {
                int presentDays = 0;
                for (boolean present : e.attendanceRecord) {
                    if (present) presentDays++;
                }
                System.out.println(e.empName + " - Present Days: " +
presentDays);
            }
            System.out.println("==========================\n");
        }
    }

    static class Department {
        String deptId, deptName;
        Employee manager;
        Employee[] employees;
        double budget;

        public Department(String deptName, Employee manager, Employee[]
employees, double budget) {
            this.deptId = generateDeptId();
            this.deptName = deptName;
            this.manager = manager;
            this.employees = employees;
            this.budget = budget;
        }

        public static String generateDeptId() {
            return "DPT" + UUID.randomUUID().toString().substring(0,
4).toUpperCase();
        }

        public double getDepartmentWiseExpenses() {
            double expense = 0;
            for (Employee e : employees) {
                expense += e.calculateSalary() + e.calculateBonus();
            }
            return expense;
        }
```

```
        public void displayDepartmentInfo() {
            System.out.println("=== Department Info ===");
            System.out.println("Department ID   : " + deptId);
            System.out.println("Name           : " + deptName);
            System.out.println("Manager        : " + manager.empName);
            System.out.println("Budget         : ₹" + budget);
            System.out.println("Employees      : ");
            for (Employee e : employees) {
                System.out.println(" - " + e.empName + " (" + e.empType +
")");
            }
            System.out.println("========================\n");
        }
    }

    public static void main(String[] args) {
        Employee e1 = new Employee("Alice", "Engineering", "Developer",
50000, "2025-01-10", "Full-time");
        Employee e2 = new Employee("Bob", "Engineering", "Intern", 15000,
"2025-02-01", "Part-time");
        Employee e3 = new Employee("Charlie", "Engineering", "Consultant",
30000, "2025-03-15", "Contract");

        for (int i = 1; i <= 28; i++) {
            e1.markAttendance(i, true);
            e2.markAttendance(i, i % 2 == 0);
            e3.markAttendance(i, true);
        }

        Department engineering = new Department("Engineering", e1, new
Employee[]{e1, e2, e3}, 200000);

        e1.generatePaySlip();
        e2.generatePaySlip();
        e3.generatePaySlip();

        engineering.displayDepartmentInfo();
        System.out.println("Department Expense: ₹" +
engineering.getDepartmentWiseExpenses());
```

```
        System.out.println("Company Payroll: ₹" +
Employee.calculateCompanyPayroll(new Employee[]{e1, e2, e3}));
        Employee.getAttendanceReport(new Employee[]{e1, e2, e3});
    }
}
```

```
=== Pay Slip ===
Employee ID   : EMP1
Name          : Alice
Department    : Engineering
Designation   : Developer
Type          : Full-time
Base Salary   : ?50000.0
Bonus         : ?5000.0
Total Pay     : ?55000.0
================

=== Pay Slip ===
Employee ID   : EMP2
Name          : Bob
Department    : Engineering
Designation   : Intern
Type          : Part-time
Base Salary   : ?15000.0
Bonus         : ?0.0
Total Pay     : ?7000.0
================

=== Pay Slip ===
Employee ID   : EMP3
Name          : Charlie
Department    : Engineering
Designation   : Consultant
Type          : Contract
Base Salary   : ?30000.0
Bonus         : ?3000.0
Total Pay     : ?33000.0
================

=== Department Info ===
Department ID   : DPT3210
Name            : Engineering
Manager         : Alice
Budget          : ?200000.0
Employees       :
 - Alice (Full-time)
 - Bob (Part-time)
 - Charlie (Contract)
=======================

Department Expense: ?95000.0
Company Payroll: ?87000.0
=== Attendance Report ===
Alice - Present Days: 28
Bob - Present Days: 14
Charlie - Present Days: 28
=========================
```

```java
// 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.


import java.util.*;

public class VehicleFleetManagementSystem {

    static class Vehicle {
        protected String vehicleId;
        protected String brand;
        protected String model;
        protected int year;
        protected double mileage;
        protected String fuelType;
```

```
        protected String currentStatus;

        protected Driver assignedDriver;

        protected static int totalVehicles = 0;
        protected static double fleetValue = 0.0;
        protected static String companyName = "TransFleet Logistics";
        protected static double totalFuelConsumption = 0.0;

        public Vehicle(String brand, String model, int year, double
mileage, String fuelType, String currentStatus) {
            this.vehicleId = generateVehicleId();
            this.brand = brand;
            this.model = model;
            this.year = year;
            this.mileage = mileage;
            this.fuelType = fuelType;
            this.currentStatus = currentStatus;
            totalVehicles++;
        }

        public static String generateVehicleId() {
            return "V" + (totalVehicles + 1);
        }

        public void assignDriver(Driver driver) {
            this.assignedDriver = driver;
            driver.assignedVehicle = this;
            System.out.println(driver.driverName + " assigned to " +
vehicleId);
        }

        public void scheduleMaintenance() {
            currentStatus = "Under Maintenance";
            System.out.println("Vehicle " + vehicleId + " scheduled for
maintenance.");
        }

        public double calculateRunningCost(double fuelRatePerKm) {
            return mileage * fuelRatePerKm;
```

```
        }

        public void updateMileage(double additionalKm) {
            mileage += additionalKm;
            totalFuelConsumption += additionalKm * getFuelEfficiency();
        }

        public boolean checkServiceDue() {
            return mileage >= 10000;
        }

        public double getFuelEfficiency() {
            return fuelType.equalsIgnoreCase("Diesel") ? 0.12 : 0.10;
        }

        public void displayInfo() {
            System.out.println(vehicleId + " | " + brand + " " + model + "
| " + year + " | Mileage: " + mileage + " km | Status: " + currentStatus);
        }
    }

    static class Car extends Vehicle {
        int passengerCapacity;

        public Car(String brand, String model, int year, double mileage,
String fuelType, int passengerCapacity) {
            super(brand, model, year, mileage, fuelType, "Available");
            this.passengerCapacity = passengerCapacity;
        }
    }

    static class Bus extends Vehicle {
        int seatingCapacity;

        public Bus(String brand, String model, int year, double mileage,
String fuelType, int seatingCapacity) {
            super(brand, model, year, mileage, fuelType, "Available");
            this.seatingCapacity = seatingCapacity;
        }
    }
```

```
    static class Truck extends Vehicle {
        double loadCapacity;

        public Truck(String brand, String model, int year, double mileage,
String fuelType, double loadCapacity) {
            super(brand, model, year, mileage, fuelType, "Available");
            this.loadCapacity = loadCapacity;
        }
    }

    static class Driver {
        String driverId;
        String driverName;
        String licenseType;
        Vehicle assignedVehicle;
        int totalTrips;

        public Driver(String driverName, String licenseType) {
            this.driverId = generateDriverId();
            this.driverName = driverName;
            this.licenseType = licenseType;
            this.totalTrips = 0;
        }

        public static String generateDriverId() {
            return "D" + UUID.randomUUID().toString().substring(0,
4).toUpperCase();
        }

        public void completeTrip(double distance) {
            totalTrips++;
            if (assignedVehicle != null) {
                assignedVehicle.updateMileage(distance);
                System.out.println(driverName + " completed trip of " +
distance + " km with " + assignedVehicle.vehicleId);
            }
        }
    }
```

```
    // Static fleet operations
    public static double calculateTotalMaintenanceCost(Vehicle[] vehicles,
double costPerVehicle) {
        double total = 0;
        for (Vehicle v : vehicles) {
            if (v.checkServiceDue()) total += costPerVehicle;
        }
        return total;
    }

    public static double getFleetUtilization(Vehicle[] vehicles) {
        int active = 0;
        for (Vehicle v : vehicles) {
            if (v.currentStatus.equalsIgnoreCase("Available")) active++;
        }
        return (double) active / vehicles.length * 100;
    }

    public static void getVehiclesByType(Vehicle[] vehicles, String type)
{
        System.out.println("Vehicles of type: " + type);
        for (Vehicle v : vehicles) {
            if ((type.equalsIgnoreCase("Car") && v instanceof Car) ||
                (type.equalsIgnoreCase("Bus") && v instanceof Bus) ||
                (type.equalsIgnoreCase("Truck") && v instanceof Truck)) {
                v.displayInfo();
            }
        }
    }

    public static void main(String[] args) {
        Vehicle[] fleet = {
            new Car("Toyota", "Corolla", 2020, 9500, "Petrol", 5),
            new Bus("Volvo", "9400", 2019, 12000, "Diesel", 40),
            new Truck("Tata", "Prima", 2021, 8000, "Diesel", 20.5)
        };

        Driver d1 = new Driver("Alice", "Heavy");
        Driver d2 = new Driver("Bob", "Light");
```

```
        fleet[0].assignDriver(d2);
        fleet[1].assignDriver(d1);

        d1.completeTrip(300);
        d2.completeTrip(150);

        for (Vehicle v : fleet) {
            v.displayInfo();
        }

        System.out.println("Fleet Utilization: " +
getFleetUtilization(fleet) + "%");
        System.out.println("Total Maintenance Cost: ₹" +
calculateTotalMaintenanceCost(fleet, 5000));
        System.out.println("Total Fuel Consumption: " +
Vehicle.totalFuelConsumption + " liters");

        getVehiclesByType(fleet, "Bus");
    }
}
```

```
Bob assigned to V1
Alice assigned to V2
Alice completed trip of 300.0 km with V2
Bob completed trip of 150.0 km with V1
V1 | Toyota Corolla | 2020 | Mileage: 9650.0 km | Status: Available
V2 | Volvo 9400 | 2019 | Mileage: 12300.0 km | Status: Available
V3 | Tata Prima | 2021 | Mileage: 8000.0 km | Status: Available
Fleet Utilization: 100.0%
Total Maintenance Cost: ?5000.0
Total Fuel Consumption: 51.0 liters
Vehicles of type: Bus
V2 | Volvo 9400 | 2019 | Mileage: 12300.0 km | Status: Available

Program finished. Cleaning up...
```

```
// 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.


import java.util.*;

public class HospitalPatientManagementSystem {

    static class Patient {
        String patientId;
        String patientName;
        int age;
```

```
        String gender;
        String contactInfo;
        String[] medicalHistory;
        String[] currentTreatments;

        static int totalPatients = 0;

        public Patient(String patientName, int age, String gender, String
contactInfo) {
            this.patientId = generatePatientId();
            this.patientName = patientName;
            this.age = age;
            this.gender = gender;
            this.contactInfo = contactInfo;
            this.medicalHistory = new String[10];
            this.currentTreatments = new String[5];
            totalPatients++;
        }

        public static String generatePatientId() {
            return "P" + (totalPatients + 1);
        }

        public void updateTreatment(String treatment) {
            for (int i = 0; i < currentTreatments.length; i++) {
                if (currentTreatments[i] == null) {
                    currentTreatments[i] = treatment;
                    return;
                }
            }
        }

        public void dischargePatient() {
            Arrays.fill(currentTreatments, null);
            System.out.println(patientName + " has been discharged.");
        }

        public void displayInfo() {
            System.out.println("Patient ID: " + patientId + " | Name: " +
patientName + " | Age: " + age + " | Gender: " + gender);
```

```
        }
    }

    static class Doctor {
        String doctorId;
        String doctorName;
        String specialization;
        String[] availableSlots;
        int patientsHandled;
        double consultationFee;

        public Doctor(String doctorName, String specialization, String[]
availableSlots, double consultationFee) {
            this.doctorId = generateDoctorId();
            this.doctorName = doctorName;
            this.specialization = specialization;
            this.availableSlots = availableSlots;
            this.consultationFee = consultationFee;
            this.patientsHandled = 0;
        }

        public static String generateDoctorId() {
            return "D" + UUID.randomUUID().toString().substring(0,
4).toUpperCase();
        }

        public void incrementPatientsHandled() {
            patientsHandled++;
        }

        public void displayInfo() {
            System.out.println("Doctor ID: " + doctorId + " | Name: " +
doctorName + " | Specialization: " + specialization);
        }
    }

    static class Appointment {
        String appointmentId;
        Patient patient;
        Doctor doctor;
```

```
        String appointmentDate;
        String appointmentTime;
        String status;
        String type;

        static int totalAppointments = 0;
        static String hospitalName = "CareWell Hospital";
        static double totalRevenue = 0.0;

        public Appointment(Patient patient, Doctor doctor, String
appointmentDate, String appointmentTime, String type) {
            this.appointmentId = generateAppointmentId();
            this.patient = patient;
            this.doctor = doctor;
            this.appointmentDate = appointmentDate;
            this.appointmentTime = appointmentTime;
            this.status = "Scheduled";
            this.type = type;
            totalAppointments++;
            doctor.incrementPatientsHandled();
        }

        public static String generateAppointmentId() {
            return "A" + (totalAppointments + 1);
        }

        public void cancelAppointment() {
            status = "Cancelled";
            System.out.println("Appointment " + appointmentId + " has been
cancelled.");
        }

        public double generateBill() {
            double rate = switch (type) {
                case "Consultation" -> doctor.consultationFee;
                case "Follow-up" -> doctor.consultationFee * 0.5;
                case "Emergency" -> doctor.consultationFee * 1.5;
                default -> doctor.consultationFee;
            };
            totalRevenue += rate;
```

```
            System.out.println("Bill for " + patient.patientName + ": ₹" +
rate);
            return rate;
        }

        public void displayAppointment() {
            System.out.println("Appointment ID: " + appointmentId + " |
Patient: " + patient.patientName +
                    " | Doctor: " + doctor.doctorName + " | Type: " + type
+ " | Status: " + status);
        }

        public static void generateHospitalReport(Appointment[]
appointments) {
            System.out.println("=== " + hospitalName + " Report ===");
            System.out.println("Total Patients     : " +
Patient.totalPatients);
            System.out.println("Total Appointments : " +
totalAppointments);
            System.out.println("Total Revenue      : ₹" + totalRevenue);
            System.out.println("================================");
        }

        public static void getDoctorUtilization(Doctor[] doctors) {
            System.out.println("=== Doctor Utilization ===");
            for (Doctor d : doctors) {
                System.out.println(d.doctorName + " - Patients Handled: "
+ d.patientsHandled);
            }
        }

        public static void getPatientStatistics(Patient[] patients) {
            System.out.println("=== Patient Statistics ===");
            for (Patient p : patients) {
                int activeTreatments = 0;
                for (String t : p.currentTreatments) {
                    if (t != null) activeTreatments++;
                }
                System.out.println(p.patientName + " - Active Treatments:
" + activeTreatments);
```

```
            }
        }
    }

    public static void main(String[] args) {
        Patient p1 = new Patient("Alice", 30, "Female",
"alice@example.com");
        Patient p2 = new Patient("Bob", 45, "Male", "bob@example.com");

        Doctor d1 = new Doctor("Dr. Smith", "Cardiology", new
String[]{"10AM", "2PM"}, 1000);
        Doctor d2 = new Doctor("Dr. Mehta", "Neurology", new
String[]{"11AM", "3PM"}, 1200);

        Appointment a1 = new Appointment(p1, d1, "2025-08-28", "10AM",
"Consultation");
        Appointment a2 = new Appointment(p2, d2, "2025-08-28", "11AM",
"Emergency");

        p1.updateTreatment("Blood Pressure Monitoring");
        p2.updateTreatment("MRI Scan");

        a1.generateBill();
        a2.generateBill();

        a1.displayAppointment();
        a2.displayAppointment();

        p1.dischargePatient();

        Appointment.generateHospitalReport(new Appointment[]{a1, a2});
        Appointment.getDoctorUtilization(new Doctor[]{d1, d2});
        Appointment.getPatientStatistics(new Patient[]{p1, p2});
    }
}
```

```
Bill for Alice: ?1000.0
Bill for Bob: ?1800.0
Appointment ID: A1 | Patient: Alice | Doctor: Dr. Smith | Type: Consultation | Status: Scheduled
Appointment ID: A2 | Patient: Bob | Doctor: Dr. Mehta | Type: Emergency | Status: Scheduled
Alice has been discharged.
=== CareWell Hospital Report ===
Total Patients     : 2
Total Appointments : 2
Total Revenue      : ?2800.0
============================
=== Doctor Utilization ===
Dr. Smith - Patients Handled: 1
Dr. Mehta - Patients Handled: 1
=== Patient Statistics ===
Alice - Active Treatments: 0
Bob - Active Treatments: 1
```