```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

/**
 * Demonstrates Functional Programming Paradigm
 * Key concepts: immutability, pure functions, higher-order functions, lambda expressions
 */
public class Exer1_Functional {

    public static void main(String[] args) {
        System.out.println("=== FUNCTIONAL PROGRAMMING PARADIGM ===\n");

        // Sample data: list of student scores
        List<Integer> scores = Arrays.asList(85, 92, 78, 95, 88, 76, 90, 83);

        System.out.println("Original scores: " + scores);

        // 1. Filter scores >= 85 using lambda and streams
        List<Integer> highScores = scores.stream()
            .filter(score -> score >= 85)
            .collect(Collectors.toList());
        System.out.println("\nHigh scores (>= 85): " + highScores);

        // 2. Map: Add 5 bonus points to each score
        List<Integer> bonusScores = scores.stream()
            .map(score -> score + 5)
            .collect(Collectors.toList());
        System.out.println("Scores with +5 bonus: " + bonusScores);

        // 3. Reduce: Calculate average score
        double average = scores.stream()
            .mapToInt(Integer::intValue)
            .average()
            .orElse(0.0);
        System.out.println("\nAverage score: " + String.format("%.2f", average));

        // 4. Pure function example
        int highest = findMax(scores);
        System.out.println("Highest score: " + highest);

        // 5. Function composition
        List<String> grades = scores.stream()
            .map(Exer1_Functional::calculateGrade)
```

```
            .collect(Collectors.toList());
        System.out.println("\nGrades: " + grades);

        // 6. Count passing scores (>= 75)
        long passingCount = scores.stream()
            .filter(score -> score >= 75)
            .count();
        System.out.println("Number of passing scores: " + passingCount);
    }

    // Pure function: always returns same output for same input, no side effects
    public static int findMax(List<Integer> numbers) {
        return numbers.stream()
            .max(Integer::compare)
            .orElse(0);
    }

    // Pure function: converts score to letter grade
    public static String calculateGrade(int score) {
        if (score >= 90) return "A";
        else if (score >= 85) return "B+";
        else if (score >= 80) return "B";
        else if (score >= 75) return "C+";
        else if (score >= 70) return "C";
        else return "F";
    }
}



import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

/**
 * Demonstrates Declarative Programming Paradigm
 * Focus: WHAT to do, not HOW to do it
 * Uses streams and high-level abstractions to describe desired results
 */
public class Exer1_Declarative {

    public static void main(String[] args) {
        System.out.println("=== DECLARATIVE PROGRAMMING PARADIGM ===\n");
```

```
// Sample data: list of products with prices
List<Product> products = Arrays.asList(
    new Product("Laptop", 45000, "Electronics"),
    new Product("Mouse", 500, "Electronics"),
    new Product("Desk", 8000, "Furniture"),
    new Product("Chair", 3500, "Furniture"),
    new Product("Monitor", 12000, "Electronics"),
    new Product("Keyboard", 1500, "Electronics")
);

System.out.println("All Products:");
products.forEach(System.out::println);

// Declarative approach: describe WHAT we want, not HOW to get it

// 1. Get all Electronics products
System.out.println("\n--- Electronics Products ---");
products.stream()
    .filter(p -> p.category.equals("Electronics"))
    .forEach(System.out::println);

// 2. Get products under 5000 pesos
System.out.println("\n--- Products under ₱5000 ---");
products.stream()
    .filter(p -> p.price < 5000)
    .forEach(System.out::println);

// 3. Get product names only
System.out.println("\n--- Product Names ---");
List<String> productNames = products.stream()
    .map(p -> p.name)
    .collect(Collectors.toList());
System.out.println(productNames);

// 4. Calculate total price of all products
double totalPrice = products.stream()
    .mapToDouble(p -> p.price)
    .sum();
System.out.println("\n--- Total inventory value: ₱" +
    String.format("%.2f", totalPrice));

// 5. Find most expensive product
System.out.println("\n--- Most Expensive Product ---");
products.stream()
```

```
            .max((p1, p2) -> Double.compare(p1.price, p2.price))
            .ifPresent(System.out::println);

        // 6. Group products by category and count
        System.out.println("\n--- Products per Category ---");
        products.stream()
            .collect(Collectors.groupingBy(p -> p.category, Collectors.counting()))
            .forEach((category, count) ->
                System.out.println(category + ": " + count + " items"));

        // 7. Check if any product exceeds 40000
        boolean hasExpensive = products.stream()
            .anyMatch(p -> p.price > 40000);
        System.out.println("\n--- Has product over ₱40000: " + hasExpensive);
    }

    // Product class to represent data
    static class Product {
        String name;
        double price;
        String category;

        Product(String name, double price, String category) {
            this.name = name;
            this.price = price;
            this.category = category;
        }

        @Override
        public String toString() {
            return String.format("%s - ₱%.2f (%s)", name, price, category);
        }
    }
}



import java.util.ArrayList;
import java.util.List;

/**
 * Demonstrates Imperative Programming Paradigm
 * Focus: HOW to do something step-by-step
 * Uses explicit control flow, loops, and state mutations
```

```
 */
public class Exer1_Imperative {

    public static void main(String[] args) {
        System.out.println("=== IMPERATIVE PROGRAMMING PARADIGM ===\n");

        // Sample data: array of student scores
        int[] scores = {85, 92, 78, 95, 88, 76, 90, 83};

        System.out.println("Original scores:");
        printArray(scores);

        // 1. Find high scores (>= 85) using imperative approach
        System.out.println("\n--- High Scores (>= 85) ---");
        List<Integer> highScores = new ArrayList<>();
        for (int i = 0; i < scores.length; i++) {
            if (scores[i] >= 85) {
                highScores.add(scores[i]);
            }
        }
        System.out.println(highScores);

        // 2. Add 5 bonus points to each score (mutation)
        System.out.println("\n--- Adding +5 bonus to all scores ---");
        for (int i = 0; i < scores.length; i++) {
            scores[i] = scores[i] + 5;
        }
        printArray(scores);

        // 3. Calculate average score using loop
        System.out.println("\n--- Calculating Average ---");
        int sum = 0;
        for (int i = 0; i < scores.length; i++) {
            sum += scores[i];
        }
        double average = (double) sum / scores.length;
        System.out.println("Average score: " + String.format("%.2f", average));

        // 4. Find highest score using loop
        System.out.println("\n--- Finding Highest Score ---");
        int highest = scores[0];
        for (int i = 1; i < scores.length; i++) {
            if (scores[i] > highest) {
                highest = scores[i];
```

```
        }
    }
    System.out.println("Highest score: " + highest);

    // 5. Convert scores to grades
    System.out.println("\n--- Converting to Letter Grades ---");
    String[] grades = new String[scores.length];
    for (int i = 0; i < scores.length; i++) {
        if (scores[i] >= 90) {
            grades[i] = "A";
        } else if (scores[i] >= 85) {
            grades[i] = "B+";
        } else if (scores[i] >= 80) {
            grades[i] = "B";
        } else if (scores[i] >= 75) {
            grades[i] = "C+";
        } else if (scores[i] >= 70) {
            grades[i] = "C";
        } else {
            grades[i] = "F";
        }
    }
    printGrades(grades);

    // 6. Count passing scores (>= 75)
    System.out.println("\n--- Counting Passing Scores ---");
    int passingCount = 0;
    for (int i = 0; i < scores.length; i++) {
        if (scores[i] >= 75) {
            passingCount++;
        }
    }
    System.out.println("Number of passing scores: " + passingCount);

    // 7. Sort scores in descending order (Bubble Sort)
    System.out.println("\n--- Sorting Scores (Descending) ---");
    sortDescending(scores);
    printArray(scores);

    // 8. Product inventory example with imperative approach
    System.out.println("\n\n=== PRODUCT INVENTORY (Imperative) ===");
    demonstrateProductInventory();
}
```

```
// Helper method to print array
public static void printArray(int[] arr) {
    System.out.print("[");
    for (int i = 0; i < arr.length; i++) {
        System.out.print(arr[i]);
        if (i < arr.length - 1) {
            System.out.print(", ");
        }
    }
    System.out.println("]");
}

// Helper method to print grades
public static void printGrades(String[] grades) {
    System.out.print("[");
    for (int i = 0; i < grades.length; i++) {
        System.out.print(grades[i]);
        if (i < grades.length - 1) {
            System.out.print(", ");
        }
    }
    System.out.println("]");
}

// Bubble sort implementation (imperative)
public static void sortDescending(int[] arr) {
    int n = arr.length;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] < arr[j + 1]) {
                // Swap elements
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

// Product inventory demonstration
public static void demonstrateProductInventory() {
    // Create products using explicit steps
    String[] names = {"Laptop", "Mouse", "Desk", "Chair", "Monitor", "Keyboard"};
    double[] prices = {45000, 500, 8000, 3500, 12000, 1500};
```

```
        String[] categories = {"Electronics", "Electronics", "Furniture", "Furniture", "Electronics",
"Electronics"};

        System.out.println("\nAll Products:");
        for (int i = 0; i < names.length; i++) {
            System.out.println(names[i] + " - ₱" + prices[i] + " (" + categories[i] + ")");
        }

        // Filter Electronics
        System.out.println("\n--- Electronics Products ---");
        for (int i = 0; i < names.length; i++) {
            if (categories[i].equals("Electronics")) {
                System.out.println(names[i] + " - ₱" + prices[i]);
            }
        }

        // Calculate total price
        System.out.println("\n--- Total Inventory Value ---");
        double total = 0;
        for (int i = 0; i < prices.length; i++) {
            total += prices[i];
        }
        System.out.println("Total: ₱" + String.format("%.2f", total));

        // Find most expensive
        System.out.println("\n--- Most Expensive Product ---");
        int maxIndex = 0;
        for (int i = 1; i < prices.length; i++) {
            if (prices[i] > prices[maxIndex]) {
                maxIndex = i;
            }
        }
        System.out.println(names[maxIndex] + " - ₱" + prices[maxIndex]);
    }
}
```