Lab 2 - Extension of Name Management with Methods, Class Variables and Arrays

This lab introduces essential object-oriented programming concepts in Java through three progressive tasks. Students will learn encapsulation by implementing private variables with getter/setter methods, work with arrays and static variables for data management, and practice method design with validation. The exercises build from enhancing the Lab 1 name management system to creating grade calculators and bank account simulators, emphasizing proper class design, data protection, and modular programming principles.

Contents

1. Task 1: Enhanced Name Management System

Extend your Java program from Lab 1 to use proper object-oriented programming principles. This task builds on the Person class you created and adds encapsulation, class variables, and arrays.

Modify your name management program as follows:
- Make all instance variables in the Person class private
- Create getter and setter methods to access these variables
- Add validation in the setter for month (must be between 1-12)
- Replace the three separate person variables with an array of Person objects
- Add a static class variable to count the total number of Person objects
- Extend the menu to include option 5: display the total person count

Detailed requirements:
- The inputs in the individual input fields themselves do not need to be checked again for correctness, i.e., letters, numbers, special characters, meaning the inputs in the individual input fields must be meaningful and do not need to be checked by your program. However, all instance variables must now be protected from direct external access, i.e., they get the private attribute and access to them is done via so-called getter and setter methods. Only the month input needs to be exemplarily checked for plausibility.
- Introduce a class variable that counts the number of persons and outputs it on request, i.e., with the input of the number 5 in the selection menu, the number of persons is output. (This is still constant in this task, but could become significant in a later extension.)
- Furthermore, the individual persons are now to be stored in an array and no longer in separate variables, to enable an extension to a realistic number of persons.

1.1. Preparation

Again, first clarify the task by drawing the Person class according to UML notation and then creating a structure chart or flowchart that solves this task. Then define all necessary classes, methods, and variables in Java.

1.2. Assistance

The following code snippets demonstrate key concepts needed for this task:

Private variables and getter/setter methods:

```java
public class Person {
    private String firstName;
    private String lastName;
    private int day, month, year;
    private static int personCount = 0;

    // Constructor
    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
        personCount++; // Increment when new person is created
    }

    // Getter methods
    public String getFirstName() {
        return firstName;
    }

    // Setter with validation
    public void setMonth(int month) {
        if (month >= 1 && month <= 12) {
            this.month = month;
        } else {
            System.out.println("Invalid month! Must be between 1-12.");
        }
    }

    // Static method to get person count
    public static int getPersonCount() {
        return personCount;
    }
}
```

Using arrays instead of separate variables:

```java
Person[] persons = new Person[3];
persons[0] = new Person("John", "Doe");
persons[1] = new Person("Jane", "Smith");
persons[2] = new Person("Bob", "Johnson");
```

2. Task 2: Student Grade Calculator

Create a Java program that manages student grades using arrays and methods. This task will help you practice working with arrays, methods, and calculations.

Write a program that:

- Stores grades for 5 students in an array
- Calculates the average grade
- Finds the highest and lowest grades
- Counts how many students passed (grade >= 60)
- Displays all results in a clear format

Your program should:

1. Ask the user to enter 5 student grades (0-100)
2. Store these grades in an array
3. Use separate methods for each calculation:
   - `int calculateAverage(int[] grades)` - returns the average
   - `int findHighest(int[] grades)` - returns the highest grade
   - `int findLowest(int[] grades)` - returns the lowest grade
   - `int countPassing(int[] grades)` - returns number of passing grades
   - `int[] deduplicatedGrades(int[] grades)` - returns a deduplicated array of occurred grades
4. Display all results with appropriate labels

2.1. Requirements

- Use an array to store the grades
- Create separate methods for each calculation
- Use loops to process the array
- Display results with clear formatting

2.2. Assistance

Array declaration and initialization:

```java
int[] grades = new int[5];

// Reading grades into array
Scanner scanner = new Scanner(System.in);
for (int i = 0; i < grades.length; i++) {
    System.out.print("Enter grade for student " + (i + 1) + ": ");
    grades[i] = scanner.nextInt();
}
```

Method example for calculating average:

```java
public static double calculateAverage(int[] grades) {
    int sum = 0;
    for (int grade : grades) {
        sum += grade;
    }
    return (double) sum / grades.length;
}
```

3. Task 3: Simple Bank Account Simulator

Create a simple bank account class that demonstrates encapsulation and method design. This task focuses on object-oriented programming principles.

Create a `BankAccount` class with the following features:

- Private balance variable
- Account holder name
- Methods for deposit, withdrawal, and balance inquiry
- Transaction validation (no negative deposits, sufficient funds for withdrawal)

Your program should:

1. Create a BankAccount object
2. Display a menu with options:
   - 1: Check balance
   - 2: Make deposit
   - 3: Make withdrawal
   - 4: Display account info
   - 0: Exit
3. Handle user input and call appropriate methods
4. Validate all transactions and display appropriate messages

3.1. Requirements

- Use private variables with public methods
- Validate all inputs (positive deposits, sufficient funds)
- Display clear success/error messages
- Use proper encapsulation principles

3.2. Assistance

Basic BankAccount class structure:

```java
public class BankAccount {
    private double balance;
    private String accountHolder;

    public BankAccount(String name, double initialBalance) {
        this.accountHolder = name;
        this.balance = initialBalance;
    }

    public boolean deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            return true;
        }
        return false;
    }

    public boolean withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            return true;
        }
        return false;
    }

    public double getBalance() {
        return balance;
    }
}
```

4. Lab Execution

If your program is not yet working without issue, we will try to correct this during the course of the lab. With good preparation, this should not be a problem. Every student is required to be able to explain their thought process at the beginning of the lab. By the end of the lab, the task needs to be completed. Of course, we will support you, but your personal commitment must also be clearly recognizable! Julian Moldenhauer, Furkan Yildirim, and Emily Antosch wish you lots of fun and success!