

Lab 1 - Name Management with Java

In this lab, the first knowledge in the Java programming language should be consolidated. You need the first four lectures to be able to complete this lab. A name management system should be programmed with which you can modify a list of names according to certain rules.

Contents

1. Task 1: Name Management System	1
1.1. Preparation	1
1.2. Assistance	1
2. Task 2: Number Guessing Game	3
2.1. Requirements	3
2.2. Assistance	3
3. Task 3: Roman Numeral Converter	5
3.1. How Roman Numerals Work	5
3.2. Your Task	5
3.3. Requirements	5
3.4. Assistance	6
4. Lab Execution	6

1. Task 1: Name Management System

This program allows different people to change their name every three years. The program to be created should be able to run on different hardware platforms without recompilation. Therefore, create a Java program that makes this possible. Proceed as follows:

- First, create a class called Person that contains the following attributes: first_name, last_name, name_change_date (separated by day, month, and year).
- Now write a Java program that first outputs the Hamburg greeting for “Good day!” on the console.
- Then your program creates three objects of the Person class, named: person_1, person_2, and person_3.
- Then your program asks which person (1, 2, or 3) wants to change their name.
- Inputting 0 leads to program termination and input 4 displays the stored attributes of all objects. If the input is 1, 2, or 3, it asks for the new first and last name and the day, month, and year from when this change should be valid. This date must be checked regarding the three-year limit. If the check is successful, the change is made in the corresponding object and the result is displayed. If it is not successful, an error message is output and the program continues.
- The inputs in the individual input fields themselves do not need to be checked 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. In a real project, this preprocessing would be checked by another program and is therefore not the subject of this task.

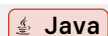
1.1. Preparation

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 in Java.

1.2. Assistance

The following program, which demonstrates console input, can be used to solve this task:

```
1 import java.util.Scanner;  
2
```



```
3  public class InputExample {
4      public static void main(String[] args) {
5          // Console input
6          Scanner scanner = new Scanner(System.in);
7
8          System.out.print("Enter something here: ");
9          String input = scanner.nextLine();
10         System.out.println("You entered \"" + input + "\"");
11
12         // If a number is needed for case distinction (e.g., age of majority):
13         System.out.print("Enter your age: ");
14         int age = scanner.nextInt();
15         if (age >= 18) {
16             System.out.println("Of age.");
17         } else {
18             System.out.println("Not yet of age.");
19         }
20
21         scanner.close();
22     }
23 }
```

2. Task 2: Number Guessing Game

Create a Java program that implements a number guessing game where the computer generates a random number and the player tries to guess it. This task will help you practice loops, conditionals, random number generation, and user input handling in Java.

The program should work as follows:

- Generate a random number between 1 and 100 (inclusive)
- Display a welcome message explaining the game rules
- Repeatedly ask the user to guess the number
- For each guess, provide feedback:
 - “Too small!” if the guess is lower than the target number
 - “Too big!” if the guess is higher than the target number
 - “You guessed it!” if the guess is correct
- Keep track of the number of attempts
- When the correct number is guessed, display a congratulatory message and the number of attempts it took
- The program ends when the correct number is guessed

2.1. Requirements

- Use the Random class for generating random numbers
- Use Scanner class for user input
- Use a loop structure to keep asking for guesses until correct
- Display clear and user-friendly messages
- Assume the user will always enter valid integers (no input validation needed)

2.2. Assistance

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

Random number generation:

```
1 import java.util.Random;
2
3 Random rand = new Random();
4 int randomNumber = rand.nextInt(100) + 1; // generates 1-100
```

Simple user input with Scanner:

```
1 import java.util.Scanner;
2
3 Scanner scanner = new Scanner(System.in);
4 System.out.print("Enter your guess: ");
5 int userGuess = scanner.nextInt();
```

Basic loop structure:

```
1 boolean gameRunning = true;
2 while (gameRunning) {
3     // Game logic here
```

```
4     if (userGuess == randomNumber) {  
5         gameRunning = false; // End the game  
6     }  
7 }
```

3. Task 3: Roman Numeral Converter

Create a Java program that converts Roman numerals to integers. This task will help you practice working with strings, character processing, and conditional logic.

Roman numerals use seven different symbols with specific values:

- I = 1
- V = 5
- X = 10
- L = 50
- C = 100
- D = 500
- M = 1000

3.1. How Roman Numerals Work

Most of the time, Roman numerals are written from largest to smallest (left to right) and you simply add up the values:

- “III” = 1 + 1 + 1 = 3
- “XII” = 10 + 1 + 1 = 12
- “LVIII” = 50 + 5 + 1 + 1 + 1 = 58

However, there are special cases where a smaller numeral appears before a larger one, meaning you subtract instead of add:

- “IV” = 5 - 1 = 4 (not “IIII”)
- “IX” = 10 - 1 = 9
- “XL” = 50 - 10 = 40
- “XC” = 100 - 10 = 90
- “CD” = 500 - 100 = 400
- “CM” = 1000 - 100 = 900

3.2. Your Task

Write a program that:

1. Asks the user to enter a Roman numeral as a string
2. Converts it to an integer using the rules above
3. Displays the result

Test your program with these examples:

- “III” should give 3
- “IV” should give 4
- “IX” should give 9
- “LVIII” should give 58
- “MCMXCIV” should give 1994

3.3. Requirements

- Use Scanner for input
- Process the string character by character
- Handle both normal addition and subtraction cases
- Display clear output

3.4. Assistance

Getting individual characters from a string:

```
1 String roman = "XIV";  
2 char firstChar = roman.charAt(0); // Gets 'X'  
3 int length = roman.length();      // Gets 3
```

 Java


Simple approach - check current and next character:

```
1 String roman = "IV";  
2 int total = 0;  
3  
4 for (int i = 0; i < roman.length(); i++) {  
5     char current = roman.charAt(i);  
6  
7     // Check if we need to subtract (when current < next)  
8     if (i < roman.length() - 1) {  
9         char next = roman.charAt(i + 1);  
10        // Add your logic here to compare current and next  
11    }  
12 }
```

 Java

Getting the value of a Roman numeral character:

```
1 // You can use if-else statements or a switch  
2 int getValue(char c) {  
3     if (c == 'I') return 1;  
4     if (c == 'V') return 5;  
5     if (c == 'X') return 10;  
6     // ... continue for other characters  
7     return 0;  
8 }
```

 Java

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!