

252-0027-00L: Introduction to Programming

Exercise Sheet 3

Submission: October 06, 2026, 23:59

In this exercise, you will finally write Java programs yourself! Check out the new template with IntelliJ by selecting *Main Menu* → *Git* → *Pull...*, as in Exercise 2.

Note that in this exercise you will have several independent programs in the same IntelliJ project. Therefore, before you start a program, make sure that you have selected the correct file or opened it in the editor. *Do not forget to comment your program code!*

Task 1: Debugging in IntelliJ

1. Open the file `FollerVehler.java` in the folder `u03` → `FollerVehler` → `src` → `main` → `java`.
2. Try to run the program. IntelliJ will show you the error messages.
3. First try to find the errors only by reading the code and thinking about how they could be fixed. (Tip: Pay attention to parentheses and braces, capitalization, semicolons, and method names.)
4. Fix the errors step by step until `FollerVehler.java` compiles and runs without errors. IntelliJ continues to show you the error messages and supports you in fixing the problems along the way.

When all errors have been fixed, the program should output the following:

```
Hello world
Gefaellet Ihnen dieses Programm?

Ich habe es selbst geschrieben.
```

Task 2: Swiss Flag

Instead of using a template, in this task you will write a program from scratch yourself. IntelliJ will, however, take some of the typing work off your hands.

1. Create a new Java file “SwissFlag.java”. To do this, right-click the `src` → `main` → `java` folder and select *New* → *Java Class*. Enter the name of the class (“SwissFlag.java”) and confirm with the Enter key. Then select *Add* if you are asked whether you want to add the file to Git.
2. First create an empty `main` method, as you have seen in other programs.
3. Extend the program so that it prints the Swiss flag in the console. The flag could look roughly as follows, but you may also design a larger or nicer version:

[illegible]

Divide the program into several methods that are called by the main method. This ensures that there are fewer repetitions of code fragments, which makes changing the program much easier.

Task 3: Input and Randomness

In this task, you work with Java input and output and get to know the classes `Scanner` and `Random` more closely. The classes `Scanner` and `Random` must be imported. This can be done with `import java.util.Scanner` and `import java.util.Random`, as shown in the lecture.

1. Write a program “Adder.java” that reads in two integers and prints their sum. You should use the Scanner class for this, as shown in the lecture. The program should ask for the first number:

Geben Sie Zahl 1 ein:

then for the second number:

Geben Sie Zahl 2 ein:

and finally, if you enter for example "25" and "1999", output the following:

$$25 + 1999 = 2024$$



Figure 1: A die with 20 sides

You can assume that only integers are entered. Test the program with different numbers.

2. Write a program `Wuerfel.java` that simulates a die. A positive integer `N` should be read in; it represents the number of sides of the die. The usual die has 6 sides, but dice with 12, 16, 20 (see Figure 1) and more sides also exist. Each side carries a different number, ranging from 1 to `N` (including `N`).

The program should simulate the roll by using the `Random` class. One possible run of the program could look as follows:

Wie viele Seiten hat Ihr Wuerfel?

The user enters a number, e.g. 20; after that, the die is rolled:

Es wurde eine 17 gewuerfelt!

3. Write a program `ChatBot.java` that simulates interaction with an AI chatbot. The program should greet the user, ask for their name and age, and then reveal the lucky number. The lucky number should be chosen randomly by using the `Random` class. One possible run of the program could look as follows:

Guten Tag! Ich bin ChatBot, der beste Chatbot, den es gibt! Wie heissen Sie?

A name is entered, e.g. Alice; after that, the chatbot responds to this input:

Sehr erfreut Alice! Wie alt sind Sie?

An age is entered, e.g. 23; after that, the chatbot responds to this input:

Mittels dieser Informationen habe ich Ihre Glueckszahl gefunden! Die Glueckszahl lautet 42.

Task 4: Calculations

1. Complete "SharedDigit.java". In the main method, two int variables a and b are declared and initialized with a value between 10 and 99 (inclusive). The program should assign a certain value to an int variable r. If a and b have a digit in common, then r is assigned the common digit (if a and b have both digits in common, then any one digit may be assigned). If there is no common digit, then -1 should be assigned. You do not need a loop for this program.

Examples:

- If a: 34 and b: 53, then r: 3
- If a: 10 and b: 22, then r: -1
- If a: 66 and b: 66, then r: 6
- If a: 34 and b: 34, then r: 3 or 4

Test your solution with a equal to 34 and b equal to 43. What does your program output?

2. Complete "SumPattern.java". In the main method, three int variables a, b, and c are declared and initialized with arbitrary values. If the sum of two of the variables gives the third, suppose $a + c == b$, then the method should output "Moeglich. a + c == b" (where the values for a, b, and c must be inserted). If this is not the case, then the method should output "Unmoeglich."

Examples:

- If a: 4, b: 10, c: 6, then "Moeglich. 4 + 6 == 10" or "Moeglich. 6 + 4 == 10" is output.
 - If a: 2, b: 12, c: 0, then "Unmoeglich." is output.
3. Complete "AbsoluteMax.java". In the main method, three int variables a, b, and c are declared and initialized with arbitrary values. The program should assign to an int variable r the largest absolute value of a, b, and c.

Task 5: EBNF

Note: This EBNF task is also available as a [Moodle quiz](#). In some parts, the Moodle quiz contains additional exercises and is therefore also helpful for students who intend to work only on the paper version.

In this task, you again create various EBNF descriptions. Save these as usual in the text file "EBNF.txt", which is located in the "u03" → "resources" folder. You can edit the file directly in IntelliJ.

1. Create a description <pyramid> which allows, as legal symbols, exactly those words that consist of a sequence of strictly increasing digits followed by a sequence of strictly decreasing digits. Examples are: 14, 121, 1221, 1341.
You may assume that the empty word is also allowed. (As a challenge, you can try to exclude the empty word.)

2. Create a description `<digitSum>` which allows, as legal symbols, exactly those natural numbers whose digit sum is an even number. (As a challenge, you can try to exclude the empty word.)
3. Create a description `<xyz>` which allows exactly all words consisting of X, Y, and Z in which every X has at least one Y in the subword to its left and to its right. Examples are: Z, YXY, YXXY, ZYXYY.
4. Create a description `<term>` which allows, as legal symbols, exactly all well-formed arithmetic terms consisting of positive integers, variables (x , y , z), addition, and parentheses. Parenthesized terms must contain at least one addition. Examples are: $1 + 4$, $(1 + 4)$, $1 + (3 + 4)$, $(1 + 1) + x + 5$.