Programming 1

Patrick Keller (patrick.keller@uni.lu)

Lab 2 – Data Types, Decision Structures & Code Smells

2021-10-01

Submission (Preliminary: 10/10, Final: 17/10)
The mandatory exercises for **Code Submission** are **3, 4, 6, 8** (marked with an ❗). All other exercises on this sheet are optional but still highly recommended! The **Explainer Video** 🎥 for this sheet must be realized on **Exercise 3 ("Speedlimit checks!")**. A Flipgrid invitation link will be posted on Moodle.

1 Data Types

In this part (exercises 1 & 2), you are not allowed to use any `if-then(-else)`, `switch/case` statements or `?:` constructions.

Exercise 1 – Let it be different! #LogicalOperator #Boolean
The "exclusive or" (xor) operator, denoted by $\oplus$, is defined by the following truth table:

a	b	a $\oplus$ b
false	false	false
false	true	true
true	false	true
true	true	false

1° On a piece of paper, try to find a formula expressing the xor operator ($\oplus$), using only simple logical operators, i.e., *and* ($\wedge$, in Java `&&`), *or* ($\vee$, in Java `||`) and *not* ($\neg$, in Java `!`). Make sure that the result of your formula matches the above truth table!

2° Write a program which defines two boolean variables `a` and `b`. For each of the 4 lines of the above truth table, adjust the values of `a` and `b` and show the result of $a \oplus b$ using your formula.

3° Java provides already an xor operator, look it up in the Java documentation or any other resource you prefer. Extend your program and use the built-in xor operator to verify your formulas results.

Exercise 2 – Go with the (over-)flow
The numeric data types have certain limits. Since the memory is finite, it is not possible to store infinitely large or infinitely precise numbers. (like π for example)

1° Using the `byte` data type, whose domain is $[-128; 127]$, show that an overflow occurs once the limits are reached. Consider both the top and the bottom limit of the domain. What do you observe, and how would you explain this?

2° If you use `int` as a type, it is no problem to store values like -129 and $+128$. However int has it's limits too, lookup those limits and write a program that demonstrates what happens when you exceed them. Is there a type that can store even larger numbers?

3° The decimal types `float` and `double` behave a bit differently on their limits. Those limitations show in precision rather than the magnitude of the value. Write a simple program that adds two double numbers - a very small one and a very big one - and print the result. Is the result what you would have expected?

2 Decision Structures

Exercise 3 – ❗ 🎥 Speedlimits checks! #If

There are general speed limits based on the type of the road, for Luxembourg those road speed limits are as follows:

Road Type	Type id	Speedlimit
motorway	1	130
ordinary	2	90
town	3	50
calmed	4	30

We assigned a "type id" to each road type to simplify the usage of the program that will be written!

1° Write a program that reads a road type id (as listed in the table) and the drivers traveling speed. Both values are read from standard input.
The program checks whether the driver is exceeding the speedlimit or not and outputs *"Speed OK for <roadtype> road."* or *"Too fast for <roadtype> road.* <excess> over the limit!" - where <roadtype> should be replaced by the corresponding road type name and <excess> should be replaced by the amount the speedlimit is exceeded.
If the road type is not valid, output *"Unknown road type: <typeid>"*, where typeid is the id that the user entered.

2° The fine you will get when exceeding a speedlimit depends on a percentage that is exceeded. The ranges can be found in the table below.
Extend your program to also output the fine that the driver will receive based on the excess speed that was measured.

Excess range	fine (in Eur)
[0 – 5]%	0
]5 – 30]%	50
]30 – 50]%	150
> 50%	500

Exercise 4 – ❗ Divisible by three! #ConditionalOperator

It is known that an integer number is divisible by 3, if and only if, the sum of the numbers digits is divisible by 3.
Write a program that reads an integer number from standard input, calculates the sum of the digits and outputs whether or not it is divisible by three. To simplify the implementation, we assume that the number has at most 4 digits.
You are allowed to use the modulo operator (% in Java) to check if the **sum of digits** is divisible by three. (Do not use it to check the number directly!)
Use the conditional operator (?:) instead of if-then-else.

Exercise 5 – Sorting

Write a program that reads three integers from standard input and displays them in ascending order.

1° First, provide a solution with nested `if`-statements.
2° Now, provide another solution without nesting.

Exercise 6 – ❗ Wait a few seconds ...

1° Write a program that reads a time consisting of hours, minutes and seconds from standard input. The user may also specify the clock format (12-hour clock with AM/PM or 24-hour clock). Be sure to provide sanity checks for the indicated values (e.g. minutes are comprised between 0 and 59).
2° Display the indicated time in the format `hh:mm:ss`. Make sure to use leading zeros for numbers less than 10.
3° Advance the time by five second. For instance, if the input time is `09:59:59`, the new time will be `10:00:04`.
4° Display the new time.

Exercise 7 – Rectangles

Write a program that determines whether a point $P(x_P; y_P)$ is inside a rectangle. The program reads the coordinates of P from standard input. It also reads the coordinates of the rectangle from standard input. A rectangle can be defined by a quadruple of coordinates $(x_{min}, y_{min}, x_{max}, y_{max})$ (we only consider rectangles whose sides are parallel to the axes).

ℹ Such a check may be used in game development for collision detection between sprites.

Exercise 8 – ❗ Crowdfunding Rewards #Switch

On crowdfunding platforms like Kickstarter or Indiegogo, it is common to offer a set of rewards to people who back a project by pledging a certain amount of money.

In this exercise, the user shall be able to enter the amount of money he wants to pledge. The available amounts are 10€, 20€, 50€, 100€, 200€and 500€.

Each reward tier contains of course the previous rewards - e.g. donating 50€includes the rewards of the 10€ and 20€ donation aswell! The following rewards would be given to the supporters of a project:

10 €	Beer with TAs
20 €	Your name in ASCII art
50 €	Public display of your donation on Moodle
100 €	Uni.lu USB Stick of 128GB
200 €	– Uni.lu Smartphone cover – Keychain "I love Programming 1"
500 €	– Autograph of your professors – Five free lunches at the "Food house"

Using a `switch` statement to display **all the rewards that the donator receives**, based on the entered amount. Try to minimize the number of `System.out.println` calls for each donation amount. If the donation amount is not contained in the above table, display some error message. (e.g. 25 € donations are not allowed!)

3 Code Styles & Code Smells

Exercise 9 – Style matters

In the repository you will find the **ILoveProgramming.java**, which is supposed to print 20 times *"I love programming!"*. But for some reason it only prints it once. Can you spot the error? Try to fix it.

ℹ In this toy example it is rather easy to find the error but if you write realistic code, it can quickly become overwhelmingly complex. In that case it is important to keep variable names representative but also short. Formatting the code in a consistent manner can save a lot of headaches, to that end there are specific guidelines. (e.g. Google Java Style Guide)