2 Instruction – Loops

Preliminaries

Exercises, or parts, that are marked with a * are extra material for the interested student.

Remember:

- Start every file you write with a line of comment with your name(s) and the date.
- Work in pairs, if possible.
- Keep your code neat from the beginning on, with proper indentation, etc.
- Have each exercise checked by one of your instructors.

2.1 Simple repetition

1. Consider the following class:

```
class Repetition {
    int k;
    int sum;

    void computeSum() {
        k = 0;
        sum = 0;
        while (k<=10) {
            sum = sum + k;
            k = k+1;
        }
        System.out.println("The sum is "+sum);
    }
}
```

2. Use this class as an inspiration to write a program (including a main method etc.) that computes the *factorial* (Dutch: *faculteit*) of 12, that is, $1 * 2 * 3 * \ldots * 12$.
3. Modify this program such that it asks for a number and then computes the factorial of this number.
4. Write a program that reads positive numbers until you enter a 0. Then it should print the product of these numbers (without the last 0). What does your program do with negative numbers? Make a sensible decision.
5. Extend this program such that it also prints the greatest number of the sequence. You may assume that the sequence is not empty.
6. And have it also print the smallest number. Is this harder?

2.2 Accumulation variables

1. Write a program that reads non-zero integer numbers until the user enters a 0. Then it should print the product of these numbers (without the last 0).
2. Extend this program such that it also prints the greatest number and the smallest number of the sequence. You may assume that the sequence is not empty.
3. Have it also print the average.

2.3 Standard deviation

Write a program that computes the standard deviation σ for a sequence of integers not equal to 0, terminated by a 0. The 0 is not considered part of the sequence. Use the following formula.

$$\sigma = \sqrt{\frac{1}{N}(\sum_{n=1}^{N} x_i^2) - \bar{x}^2}$$

For a derivation of the formula from the more usual definition see, e.g., the wikipedia page on "standard deviation".

2.4 Guessing game

1. Write a program that plays a guessing game with the user. (S)he should guess an integer number between 0 and 1000 that is secretly input by someone else. The program allows as many guesses as necessary and gives helpful replies to each guess ("lower", "higher"). When the player guesses right, the program should output: "Congratulations, ... was the right guess!" and then terminate.

   NB If you don't want to have someone else put in the secret number, use `Math.random()` to provide a random number between 0.0 and 1.0. If, for example, you want int `secretNumber` to get a value between between 0 and 1000 you can use `secretNumber = (int)(Math.random() * 1001)`. The effect of `(int)` is that the double `Math.random() * 1001` is rounded down to an int.

2. Change the program to also count the number of guesses and output this number, adding "You used ... guesses."

3. Change the program such that after 10 guesses it informs the player that the player has lost the game, gives the number and terminates.

4. Challenge: reverse roles and write a program that *guesses* a number that you have in mind, informing the computer at each guess whether it was too low, too high or correct.

2.5 Prime numbers

An integer is prime if it is greater than 1 and its only divisors are 1 and itself: 2, 3 , 5, 7, 11, 13, 17, ...

1. Write a program that asks for and reads an integer from input and outputs whether or not it is prime.

   Example:
   Type a number
   121
   121 is not prime

   Use the modulo operator %, which gives the remainder on division: 7%3 evaluates to 1.

2. Write a program that outputs the first primes up to a desired number.

   Example:
   Up to which number you want the primes?
   23
   2 3 5 7 11 13 17 19

3. Write a program that outputs the first primes up to a certain amount.

   Example:
   ```
   How many of the primes do you want?
   10
   2 3 5 7 11 13 17 19 23 27
   ```

2.6 Finding the largest integer

The built-in numeric types of Java are limited in the size of the numbers you can store. Try this by adding two large integer numbers, e.g., 1234567 and 2345678.

1. We want to find the largest possible int experimentally. As you have seen, adding two positive numbers of which the sum is too large for an int, results in a *negative* number. Use this information to write a loop to find the largest possible int.
2. Do the same for the smallest negative number.
3. These values can be obtained directly with the expressions `Integer.MAX_VALUE` and `Integer.MIN_VALUE`.
4. * As you see, these values are not each other's opposite. Try to find a reason for this. In your head or on the web (e.g., the wikipedia article on *two's complement*).

2.7 Floating Around

The floating point types in Java (float and double) can store much larger values than int can, but this comes at a price: they have limited *precision*.

1. What happens when you add 1 to a very large `double` number?
2. There are not only gaps between large numbers, but also gaps between smaller numbers. There are a lot of real numbers that cannot be represented with Java floating point numbers. Write a program that calculates $x + 0.2 - x$ for $x = 1$. Try it also for $x = 0$ and $x = 10$ and for other values than 0.2, e.g., 0.5 and 0.1. Can you explain this?
3. Write a program that calculates $x + 0.2 - x$ for $x = 0, 1, 2, \ldots, n$ for $n = 50$.
4. Do you see a pattern? Experiment more with this program. Improve the "visualization" by printing only a "+" sign when the result is greater than .2 and "–" when it is smaller and use larger numbers for n. Print the pluses and minuses after one another, on one line, or print a new line when the sign changes. Print the values for i where the sign changes. Can you explain (a bit of) this?