Part 2

Repeating functionality

Learning Objectives

- You are familiar with loops and know how to create a program that contains one.
- You know how to use the `break` command to end a loop's execution.
- You know how to use `continue` command to return to the beginning of a loop.
- You are able to create a program that reads inputs from a user until a specific input is given. For example, the number 0 or the string "end", after which the program prints something about the provided inputs (e.g., the input count, and in the case of numbers their sum and average).

A computer's processor, which specializes in executing commands, is capable of executing — in a modern computer — over a billion (machine code) commands in a second.

In this section, we'll get used to writing often-repeated program code with the help of loops.

Let's motivate ourselves to use loops. Below you'll find an example of a program that asks the user for five numbers and calculates their sum.

```
Scanner scanner = new Scanner(System.in);
int sum = 0;

System.out.println("Input a number: ");
sum = sum + Integer.valueOf(scanner.nextLine());

System.out.println("Input a number: ");
sum = sum + Integer.valueOf(scanner.nextLine());

System.out.println("Input a number: ");
sum = sum + Integer.valueOf(scanner.nextLine());
```

```
System.out.println("Input a number: ");
sum = sum + Integer.valueOf(scanner.nextLine());

System.out.println("Input a number: ");
sum = sum + Integer.valueOf(scanner.nextLine());

System.out.println("The sum of the numbers is " + sum);
```

It does the job, but not elegantly. What if the program had to read a hundred, or perhaps a thousand numbers and print their sum? What if the program had to read three numbers only?

The problem can be solved with a loop which keeps track of both the sum and the number of times input has been read. The program that prints the sum of five numbers now looks as follows

```
Scanner scanner = new Scanner(System.in);

int numbersRead = 0;
int sum = 0;

while (true) {
    if (numbersRead == 5) {
        break;
    }

    System.out.println("Input number");
    sum = sum + Integer.valueOf(scanner.nextLine());
    numbersRead = numbersRead + 1;
}

System.out.println("The sum of the numbers is " + sum);
```

Next off we will get familiar with loops.

Loops and Infinite Loops

A loop consists of an expression that determines whether or not the code within the loop should be repeated, along with a block containing the source code to be repeated. A loop takes the following form.

```
while (_expression_) {
    // The content of the block wrapped in curly brackets
    // The block can have an unlimited amount of content
}
```

We'll use the value `true` as the loop's expression for now. This way, the loop's execution is always continued when the program arrives at the point that decides whether it should be repeated or not. This happens when the execution of the program first arrives at the loop expression for the first time, and also when it reaches the end of the loop's block.

The loop execution proceeds line-by-line. The following program outputs *I can program* an infinite number of times.

```
while (true) {
    System.out.println("I can program!");
}
```

A program that runs infinitely does not end on its own. In NetBeans, it can be shut down by clicking the red button located on the left side of the output window.

Ending a Loop

The loop can be broken out of with command 'break'. When a computer executes the command 'break', the program execution moves onto the next command following the loop block.

The example below is a program that prints numbers from one to five. Note how the variable that's used within the loop is defined before the loop. This way the variable can be incremented inside the loop and the change sticks between multiple iterations of the loop.

```
int number = 1;

while (true) {
    System.out.println(number);
    if (number >= 5) {
        break;
    }
```

```
    number = number + 1;
}

System.out.println("Ready!");
```

Sample output

```
1
2
3
4
5
Ready!
```

Breaking out of the loop occurs when a user enters a specified input or whenever a calculation performed in the loop ends in the desired result. These kinds of programs contain both a loop used to define a section to be repeated and also a conditional expression used to check whether or not the condition to exit the loop has been fulfilled.

Users can also be asked for input within a loop. The variables that are commonly used in loops (such as Scanner readers) are defined before the loop, whereas variables (such as the value read from the user) that are specific to the loop are defined within it.

In the example below, the program asks the user whether to exit the loop or not. If the user inputs the string "y", the execution of the program moves to the command following the loop block, after which the execution of the program ends.

```
Scanner scanner = new Scanner(System.in);

while (true) {
    System.out.println("Exit? (y exits)");
    String input = scanner.nextLine();
    if (input.equals("y")) {
        break;
    }

    System.out.println("Ok! Let's carry on!");
}
```

```
System.out.println("Ready!");
```

The program in the example works as follows. The user's inputs are marked in red.

Sample output

```
Exit? (y exits)
no
Ok! Let's carry on!
Exit? (y exits)
non
Ok! Let's carry on!
Exit? (y exits)
y
Ready!
```

Programming exercise:

Carry on?

Points 1/1

Write a program by using the loop example that asks "Shall we carry on?" until the user inputs the string "no".

Sample output

```
Shall we carry on?
yes
Shall we carry on?
ye
Shall we carry on?
y
Shall we carry on?
no
```

Exercise submission instructions

How to see the solution

In the previous example, the program read inputs of type string from the user. The program can also be implemented with other types of variables. The program below asks numbers from the user until the user inputs a zero.

```
Scanner scanner = new Scanner(System.in);

while (true) {
    System.out.println("Input a number, 0 to quit");
    int command = Integer.valueOf(scanner.nextLine());
    if (command == 0) {
        break;
    }

    System.out.println("You input " + command);
}

System.out.println("Done, thank you!");
```

The output of the program can be as follows:

Sample output

```
Input a number, 0 to quit
5
You input 5
Input a number, 0 to quit
-2
You input -2
Input a number, 0 to quit
0
Done, thank you!
```

Programming exercise:

Are we there yet?

Points 1/1

Write a program, according to the preceding example, that asks the user to input values until they input the value 4.

Sample output

```
Give a number:
5
Give a number:
744
Give a number:
22
Give a number:
-1
Give a number:
4
```

Exercise submission instructions

How to see the solution

Returning to the Start of the Loop

When the execution reaches the end of the loop, the execution starts again from the start of the loop. This means that all the commands in the loop have been executed. You can also return to the beginning from other places besides the end with the command `continue`. When the computer executes the command `continue`, the execution of the program moves to the beginning of the loop.

The example below demonstrates the use of the `continue` command. The program asks the user to input positive numbers. If the user inputs a

negative number or a zero, the program prints the message "Unfit number, try again", after which the execution returns to the beginning of the loop. In the previous example, the program read inputs of type string from the user. Similar programs with different input types are also possible. In the example below, the user is asked for numbers until they input a zero.

```
Scanner scanner = new Scanner(System.in);

while (true) {
    System.out.println("Insert positive integers");
    int number = Integer.valueOf(scanner.nextLine());

    if (number <= 0) {
        System.out.println("Unfit number! Try again.");
        continue;
    }

    System.out.println("Your input was " + number);
}
```

The program in the example above is repeated infinitely since the `break` command used for exiting the loop is not used. To exit the loop, the `break` command must be added to it.

In the example below, the program is modified in such a way that the user is asked to input positive numbers. If the user inputs a negative number, the program informs them that the number was unfit and returns to the beginning of the loop. If the number was zero, the program exits the loop.

```
Scanner scanner = new Scanner(System.in);

while (true) {
    System.out.println("Input positive numbers.");
    int number = Integer.valueOf(scanner.nextLine());

    if (number == 0) {
        break;
    }

    if (number < 0) {
        System.out.println("Unfit number! Try again.");
        continue;
    }
```

```
    System.out.println("Your input was " + number);
}
```

Quiz:

Exiting loop and limit

Points: 1/1

What does the program below print?

```
public static void main(String[] args) {
    int number = 0;

    while (true) {
        number = number + 1;

        if (number >= 5) {
            break;
        }

        if (number < 5) {
            continue;
        }

        System.out.print(number + " ");
    }

    System.out.print(number + " ");
}
```

Your answer:

Answer

1 2 3 4 5

Your answer is correct

Answered

Tries remaining: 1

Programming exercise:

Only positives

Points
1/1

Write a program that asks the user for numbers. If the number is negative (smaller than zero), the program prints for user "Unsuitable number" and asks the user for a new number. If the number is zero, the program exits the loop. If the number is positive, the program prints the number to the power of two.

Sample output

```
Give a number:
5
25
Give a number:
4
16
Give a number:
-3
Unsuitable number
Give a number:
1
1
Give a number:
0
```

Exercise submission instructions

How to see the solution

In the previous exercise, you made a program that asks the user for numbers. If the user entered a negative number, the program would inform them that the number was unfit, and if the user entered a zero,

the program would exit. A possible solution to the exercise is the following.

```
Scanner scanner = new Scanner(System.in);

while (true) {
    System.out.println("Input a number");
    int number = Integer.valueOf(scanner.nextLine());

    if (number == 0) {
        break;
    }

    if (number < 0) {
        System.out.println("Unfit number");
        continue;
    }

    System.out.println(number * number);
}
```

The program could be made by modifying the if-statement to another form. In the example below, the conditionals have been combined to replace separate if-statements.

```
Scanner scanner = new Scanner(System.in);

while (true) {
    System.out.println("Input a number");
    int number = Integer.valueOf(scanner.nextLine());

    if (number == 0) {
        break;
    } else if (number < 0) {
        System.out.println("Unfit number");
        continue;
    }

    System.out.println(number * number);
}
```

Which of the previous examples was more clear?

Let's examine the clarity of the previous programs through an example. Below, the program asks the user for a number. If the number is negative, the user is informed that the number is unfit and the execution of the program goes to the beginning of the loop. If the number is zero, the program exits the loop. In other cases the program prints the square of the number, i.e., the number times itself.

```
Scanner scanner = new Scanner(System.in);

while (true) {
    System.out.println("Input a number ");
    int number = Integer.valueOf(scanner.nextLine());

    if (number < 0) {
        System.out.println("Unfit number");
        continue;
    }

    if (number == 0) {
        break;
    }

    System.out.println(number * number);
}
```

This program can also be done by combining the if-statements. In that case, the implementations would be the following.

```
Scanner scanner = new Scanner(System.in);

while (true) {
    System.out.println("Input a number ");
    int number = Integer.valueOf(scanner.nextLine());

    if (number < 0) {
        System.out.println("Unfit number");
    } else if (number == 0) {
        break;
    } else {
        System.out.println(number * number);
    }
}
```

Let's examine the previous programs with comments. Before each command, there's a comment that aims to explain what's happening in the program. Below is a program that's written with separate if-statements.

```
// The task is to read an input from the user
Scanner scanner = new Scanner(System.in);

// The task is to repeat the block until the block is exited
while (true) {
    // The task is to ask the user for an input
    System.out.println("Input a number ");
    // The task is to read a number from the user
    int number = Integer.valueOf(scanner.nextLine());

    // The task is to guard and prevent unfit numbers
    // for further processing
    if (number < 0) {
        System.out.println("Unfit number");
        continue;
    }

    // The task is to check if the loop should be exited
    if (number == 0) {
        break;
    }

    // The task is to print the square of the number
    System.out.println(number * number);
}
```

Note that every if-statement has a single, clear task.

When we comment on a program containing combined if-statements, the comments take the following form.

```
// The task is to read an input from the user
Scanner scanner = new Scanner(System.in);

// The task is to repeat the block until the block is exited
while (true) {
    // The task is to ask the user for an input
    System.out.println("Input a number ");
    // The task is to read a number from the user
    int number = Integer.valueOf(scanner.nextLine());
```

```
    // The purpose of the if-else if-else block?
    // The task is the processing of the number?
    if (number < 0) {
        System.out.println("Unfit number");
    } else if (number == 0) {
        break;
    } else {
        System.out.println(number * number);
    }
}
```

We notice that it's difficult to define a single, clear task for `if-else if-else`-block. During the design and implementation of a program, it's desirable to aim for a situation in which every part of the program has a a single, clear task. This theme repeats throughout the course.

Calculation with Loops

Loops are used in computing many different things. For example, programs that process indefinite numbers of user-inputted values make use of loops. These kinds of programs typically print out some sort of statistics about the numbers that were read or other inputs after the end loop.

For the program to print out information from the loop execution after the loop, the information must be saved and modified during the loop.

If the variable used to store the data is introduced within the loop, the variable is only available within that loop and nowhere else.

Let's create a program to count and print out the number of ones entered by the user. Let's first create a non-working version and examine the action of the blocks.

```
Scanner scanner = new Scanner(System.in);

// The task is to read an input from the user
while (true) {

    // The task is to keep count of number ones
    int ones = 0;

    System.out.println("Input a number (0 exits): ");
```

```
    // The task is to read a user inputted number
    int number = Integer.valueOf(scanner.nextLine());

    // The task is to exit the loop if the user
    // has inputted zero
    if (number == 0) {
        break;
    }

    // The task is to increase the amount of ones
    // if the user inputs a number one
    if (number == 1) {
        ones = ones + 1;
    }
}

// The task is to print out the total of ones
// This doesn't work because the variable ones has been
// introduced within the loop
System.out.println("The total of ones: " + ones);
```

The previous program does not work because the variable `ones` is introduced within the loop, and an attempt is made to use it after the loop at the end of the program. The variable only exists inside the loop. If the print statement `System.out.println("The total of ones: " + ones);` was inside the loop, the program would work, but not in the desired way. Let's examine this next.

```
Scanner scanner = new Scanner(System.in);

// The task is to read an input from the user
while (true) {

    // The task is to keep count of number ones
    int ones = 0;

    System.out.println("Insert a number (0 exits): ");
    // The task is to read a user inputted number
    int number = Integer.valueOf(scanner.nextLine());

    // The task is to exit the loop if the user
    // has inputted zero
    if (number == 0) {
        break;
    }
```

```
    // The task is to increase the amount of ones
    // if the user inputs a number one
    if (number == 1) {
        ones = ones + 1;
    }

    // The task is to print out the total of ones
    System.out.println("The total of ones: " + ones);
}
```

The example above works, but not in a way we hoped it would. Below the example output of the program

Sample output

```
Insert a number (0 exits)
5
The total of ones: 0
Insert a number (0 exits)
1
The total of ones: 1
Insert a number (0 exits)
1
The total of ones: 1
Insert a number (0 exits)
2
The total of ones: 0
Insert a number (0 exits)
0
```

If you wish to use a variable after a loop, it needs to be introduced before the loop.

In the example below, the program computes the total of number ones inputted. The inputs are read until the user inputs a zero after which the program prints the total count of number ones entered. The program uses variable `ones` to keep track of the number ones.

```
Scanner scanner = new Scanner(System.in);

// The task is to keep track of number ones
```

```
int ones = 0;

// The task is to read an input from the user
while (true) {
    System.out.println("Give a number (end with 0): ");
    // The task is to read a user inputted number
    int number = Integer.valueOf(scanner.nextLine());

    // The task is to exit the loop if the user
    // has inputted zero
    if (number == 0) {
        break;
    }

    // The task is to increase the amount of ones
    // if the user inputs a number one
    if (number == 1) {
        ones = ones + 1;
    }
}

// The task is to print out the total of ones
System.out.println("The total of ones: " + ones);
```

Below is an example output of the program.

Sample output

```
Give a number (end with 0):
1
Give a number (end with 0):
2
Give a number (end with 0):
1
Give a number (end with 0):
-1
Give a number (end with 0):
0
Total of ones: 2
```

Programming exercise:

Number of Numbers

Points
1/1

Write a program that reads values from the user until they input a 0. After this, the program prints the total number of inputted values. The zero that's used to exit the loop should not be included in the total number count.

Example output of the program:

Sample output

```
Give a number:
5
Give a number:
22
Give a number:
9
Give a number:
-2
Give a number:
0
Number of numbers: 4
```

Exercise submission instructions

How to see the solution

Programming exercise:

Number of negative numbers

Points
1/1

Write a program that reads values from the user until they input a 0. After this, the program prints the total number of inputted values that

are negative. The zero that's used to exit the loop should not be included in the total number count.

Example output of the program:

Sample output

```
Give a number:
5
Give a number:
22
Give a number:
9
Give a number:
-2
Give a number:
0
Number of negative numbers: 1
```

Exercise submission instructions

How to see the solution

The programs written in the previous exercises have read input from the user and kept track of the count of certain types of numbers. In the next exercise, the requested sum of numbers is not much different — this time, rather than keeping track of the number of values entered, you add the number entered by the user to the sum.

Programming exercise:

Sum of Numbers

Points 1/1

Write a program that reads numbers from the user until the user inputs a number 0. After this the program outputs the sum of the

numbers. The number zero does not need to be added to the sum, even if it does not change the results.

Example output of the program:

Sample output

```
Give a number:
5
Give a number:
22
Give a number:
9
Give a number:
-2
Give a number:
0
Sum of the numbers: 34
```

Exercise submission instructions

How to see the solution

Sometimes you need to use multiple variables. The example below shows a program which reads numbers from the user until the user writes 0. Then the program prints the number of positive and negative numbers given, and the percentage of positive numbers from all numbers given.

```
Scanner reader = new Scanner(System.in);

// For saving number of numbers
int numberOfPositives = 0;
int numberOfNegatives = 0;

// For repeatedly asking for numbers
while (true) {
    System.out.println("Give a number (0 to stop): ");
    // For reading user input
    int numberFromUser = Integer.valueOf(reader.nextLine());
```

```
    // For breaking the loop when user writes 0
    if (numberFromUser == 0) {
        break;
    }

    // For increasing numberOfPositives by one
    // when user gives a positive number
    if (numberFromUser > 0) {
        numberOfPositives = numberOfPositives + 1;
    }

    // For increasing numberOfNegatives by one
    // when user gives a negative number
    if (numberFromUser < 0) {
        numberOfNegatives = numberOfNegatives + 1;
    }

    // Also could have used..
    // if (numberFromUser > 0) {
    //     numberOfPositives = numberOfPositives + 1;
    // } else {
    //     numberOfNegatives = numberOfNegatives + 1;
    // }

}

// For printing the number of positive numbers
System.out.println("Positive numbers: " + numberOfPositives);
// For printing the number of negative numbers
System.out.println("Negative numbers: " + numberOfNegative)s;

// For printing the percentage of positive numbers from all numbers

if (numberOfPositives + numberOfNegatives > 0) {
    // Print only if user has given numbers
    // to avoid dividing by zero
    double percentageOfPositives = 100.0 * numberOfPositives / (numberOfPositiv
    System.out.println("Percentage of positive numbers: " + percentageOfPositiv
}
```

Programming exercise:

Number and sum of numbers

Points 1/1

Write a program that asks the user for input until the user inputs 0. After this the program prints the amount of numbers inputted and the sum of the numbers. The number zero does not need to be added to the sum, but adding it does not change the results.

You need two variables to keep track of the information. Use one for keeping track of the numbers inputted and other for keeping track of the sum

Example output of the program:

Sample output

```
Give a number:
5
Give a number:
22
Give a number:
9
Give a number:
-2
Give a number:
0
Number of numbers: 4
Sum of the numbers: 34
```

Exercise submission instructions

How to see the solution

Programming exercise:

Average of numbers

Points
1/1

Write a program that asks the user for input until the user inputs 0. After this, the program prints the average of the numbers. The number zero does not need to be counted to the average. You may assume that the user inputs at least one number.

The average of the numbers can be calculated by dividing the sum of numbers with the amount of the numbers

Example output of the program:

Sample output

```
Give a number:
5
Give a number:
22
Give a number:
9
Give a number:
-2
Give a number:
0
Average of the numbers: 8.5
```

Exercise submission instructions

How to see the solution

Programming exercise:

Average of positive numbers

Points
1/1

Write a program that asks the user for input until the user inputs 0. After this, the program prints the average of the positive numbers (numbers that are greater than zero).

If no positive number is inputted, the program prints "Cannot calculate the average"

Below a few examples of the programs output

Sample output

```
3
5
1
-3
0
3.0
```

Sample output

```
0
Cannot calculate the average
```

Sample output

```
-3
1
0
1.0
```

Sample output

```
1
1
0
1.0
```

Exercise submission instructions

How to see the solution

You have reached the end of this section! Continue to the next section:

→ 3. More loops

Remember to check your points from the ball on the bottom-right corner of the material!

In this part:

Source code of the material

This course is created by the Agile Education Research -research group of the University of Helsinki.

Credits and about the material.

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

MASSIIVISET AVOIMET VERKKOKURSSIT
MASSIVE OPEN ONLINE COURSES · MOOC.FI