Warning: This course has migrated to a new server https://tmc.mooc.fi (Only affects people using the server https://tmc.mooc.fi/mooc)
What this means:

- Old accounts at https://tmc.mooc.fi/mooc will not work anymore. You have to create a new account at https://tmc.mooc.fi.
- Old submissions at https://tmc.mooc.fi/mooc will not be migrated to the new server. Theywill still be visible at the old server for now.
- The old server https://tmc.mooc.fi/mooc will be shut down at some point next year, but there is a brand new Object-Oriented Programming with Java course coming up! This course is just a placeholder between the old and the new course.
- This placeholder course is found at https://tmc.mooc.fi/org/mooc. Notice the /org/ part in the middle.
- If you were doing the course on the old server and want to continue where you left off there, please resubmit all your exercises to the new server.
- Remember to change your account name, password and server address in Netbeans' TMC Settings to correspond the account you have on the new server. The correct server address is "https://tmc.mooc.fi/org/mooc".

Object-Oriented Programming with Java, part I

Authors: Arto Hellas, Matti Luukkainen
Translators to English: Emilia Hjelm, Alex H. Virtanen, Matti Luukkainen, Virpi Sumu, Birunthan Mohanathas

The course is maintained by the Agile Education Research Group

1. The program and the source code

1.1 Source code

A computer program is composed of commands written in the *source code*. A computer generally runs *commands* in the source code from *top to bottom* and *from left to right*. Source code is saved in a textual format and will be *executed* somehow.

1.2 Commands

Computers execute different *operations*, or actions, based on the commands. For example, when printing the text "Hello world!" on the screen, it is done by the command `System.out.println`.

```
System.out.println("Hello world!");
```

The `System.out.println` command prints the string given inside the brackets on the screen. The suffix `ln` is short for the word *line*. Therefore, this command prints out a line. This means that after the given string has been printed, the command will also print a line break.

1.3 Compiler and interpreter

Computers do not directly understand the programming language we are using. We need a *compiler* between the source code and the computer. When we are programming using the command line interface, the command `javac Hello.java` will compile the `Hello.java` file into *bytecode*, which can be executed using the Java interpreter. To run the compiled program, you can use the command `java Hello` where Hello is the name of the original source code file.

When using a modern development environment (more on this later), it will take care of compiling the source code. When we choose to run the program, the development environment will compile and execute the program. All development environments compile source code while it is being written by the programmer, which means that simple errors will be noticed before executing the program.

1.4 Components of commands

1.4.1 Semicolon

A semicolon `;` is used to separate different commands. The compiler and the interpreter both ignore line breaks in the source code, so we could write the entire program on a single line.

In the example below we will use the `System.out.print` command, which is similar to the `System.out.println` command except that it will not print a line break after printing the text.

Example of how the semicolons are used

```
System.out.print("Hello "); System.out.print("world");
System.out.print("!");
```

```
Hello world!
```

Even though neither the compiler nor the interpreter need line breaks in the source code, they are very important when considering human readers of the source code. Line breaks are required to divide source code in a clear manner. Readability of source code will be emphasized throughout this course.

1.4.2 Parameters (information passed to commands)

The information processed by a command are the *parameters of a command*. They are passed to the command by placing them between `()` brackets that follow the command name. For example, the `System.out.print` command is given the text *hello* as a parameter as follows: `System.out.print("hello")`.

1.4.3 Comments

Comments are a useful way to make notes in the source code for yourself and others. Everything on a line after two forward slashes `//` is treated as a comment.

1.4.4 Example of using comments

```
// We will print the text "Hello world"
System.out.print("Hello world");

System.out.print(" and all the people of the world."); // We print more text to the same line.

// System.out.print("this line will not be executed, because it is commented out");
```

The last line of the example introduces a particularly handy use for comments: you can comment out code instead of completely deleting it if you want to temporarily try out something.

1.5 More about printing

As we can see from the examples above, there are two commands for printing.

- `System.out.print` prints the text without the line break at the end
- `System.out.println` prints the text and the line break

The printed text can contain both traditional characters and special characters. The most important special character is `\n`, which stands for a line break. There are also other special characters.

```
System.out.println("First\nSecond\nThird");
```

When executed, the example above prints:

```
First
Second
Third
```

2. Main program body

The body for a program named "Example" is as follows:

```
public class Example {
    public static void main(String[] args) {
        // program code
    }
}
```

The program is stored in a text file named after the program with the *.java* extension. For a program named *Example*, the file should be named `Example.java`.

The execution of the program begins at the part marked with the *// program code* comment above. During our first week of programming, we will limit ourselves to this part. When we are talking about commands such as printing, we need to write the commands into the program body. For example: `System.out.print("Text to be printed");`

```
public class Example {
    public static void main(String[] args) {
        System.out.print("Text to be printed");
    }
}
```

From this point on, the main program body will be omitted from the examples.

3. Getting to know your development environment

Programming these days takes place in development environments almost without exceptions. The development environment provides several tools and features to assist the programmer. Although the development environment does not write the program on behalf of the programmer, it contains several handy features such as hinting about mistakes in code and assisting the programmer to visualize the structure of the program.

In this course, we will use the NetBeans development environment. A guide for using NetBeans is available here.

Until you become familiar with NetBeans, follow the guides and steps precisely. Most of the following exercises show what needs to be printed to the screen for the program to function correctly.

Note: Do not do the exercises by writing code and then clicking the test button. You should also execute the code manually (green arrow) and observe the result on the screen. This is especially useful if an exercise fails to pass the tests.

In the following exercises, we will practice the use of NetBeans and printing of text on the screen.

Remember to read the guide on using NetBeans before you continue!

Please answer to our survey: here. It will take less than five minutes.

Exercise 1: Name

Create a program that prints your name to the screen.

The program output should resemble the following:

```
Jane Doe
```

Exercise 2: Hello world! (And all the people of the world)

Create a program that prints out the following:

```
Hello world!
(And all the people of the world)
```

Exercise 3: Spruce

Create a program that prints the following:

```
    *
   ***
  *****
 *******
*********
    *
```

Note: You probably wrote `System.out.println("...")` quite a few times. Try typing only *sout* on an empty line in NetBeans and then press the tab key. What happened? This tip will save a lot of your time in the future!

4. Variables and assignment

4.1 Variables and data types

A *variable* is one of the most important concepts in computer programming. A variable should be imagined as a box in which you can store information. The information stored in a variable always has a type. These types include text (*String*), whole numbers (*int*), decimal numbers (*double*), and truth values (*boolean*). A *value* can be assigned to a variable using the equals sign (=).

```
int months = 12;
```

In the statement above, we assign the value 12 to the variable named `months` whose data type is integer (`int`). The statement is read as "the variable `months` is assigned the *value* 12".

The value of the variable can be appended to a string with the plus + sign as shown in the following example.

```
String text = "includes text";
int wholeNumber = 123;
double decimalNumber = 3.141592653;
boolean isTrue = true;

System.out.println("The variable's type is text. Its value is " + text);
System.out.println("The variable's type is integer. Its value is  " + wholeNumber);
System.out.println("The variable's type is decimal number. Its value is " + decimalNumber);
System.out.println("The variable's type is truth value. Its value is " + isTrue);
```

Printing:

```
The variable's type is text. Its value is includes text
The variable's type is integer. Its value is 123
The variable's type is decimal number. Its value is 3.141592653
The variable's type is truth value. Its value is true
```

A variable holds its value until it is assigned a new one. Note that the variable type is written only when the variable is first declared in the program. After that we can use the variable by its name.

```
int wholeNumber = 123;
System.out.println("The variable's type is integer. Its value is  " + wholeNumber);

wholeNumber = 42;
System.out.println("The variable's type is integer. Its value is  " + wholeNumber);
```

The output is:

```
The variable's type is integer. Its value is 123
The variable's type is integer. Its value is 42
```

4.2 Variable data types are immutable

When a variable is declared with a data type, it cannot be changed later. For example, a text variable cannot be changed into an integer variable and it cannot be assigned integer values.

```
String text = "yabbadabbadoo!";
text = 42; // Does not work! :(
```

Integer values can be assigned to decimal number variables, because whole numbers are also decimal numbers.

```
double decimalNumber = 0.42;
decimalNumber = 1; // Works! :)
```

Exercise 4: Varying variables

The exercise file initially contains a program which prints:

```
Chickens:
3
Bacon (kg):
5.5
A tractor:
There is none!

In a nutshell:
3
5.5
There is none!
```

Change the program in the specified places so that it will print:

```
Chickens:
9000
Bacon (kg):
0.1
A tractor:
Zetor

In a nutshell:
9000
0.1
Zetor
```

4.3 Allowed and descriptive variable names

There are certain limitations on the naming of our variables. Even though umlauts, for example, can be used, it is better to avoid them, because problems might arise with character encoding. For example, it is recommended to use A instead of Ä.

Variable names must not contain certain special characters like exclamation marks (!). Space characters cannot be used, either, as it is used to separate commands into multiple parts. It is a good idea to replace the space character using a *camelCase* notation. **Note:** The first character is always written in lower case when using the camel case notation.

```
int camelCaseVariable = 7;
```

Variable names can contain numbers as long it does not start with one. Variable names cannot be composed solely of numbers, either.

```
int 7variable = 4; // Not allowed!
int variable7 = 4; // A valid, but not descriptive variable name
```

Variable names that have been defined before cannot be used. Command names such as `System.out.print` cannot be used, either.

```
int camelCase = 2;
int camelCase = 5; // Not allowed, the variable camelCase is already defined!
```

It is strongly recommended to name variables so that their purpose can be understood without comments and without thinking. Variable names used in this course **must** be descriptive.

4.3.1 Valid variable names

- lastDay = 20
- firstYear = 1952
- name = "Matti"

4.3.2 Invalid variable names

- last day of the month = 20
- 1day = 1952
- watchout! = 1910
- 1920 = 1

5. Calculation

The calculation operations are pretty straightforward: `+`, `-`, `*` and `/`. A more peculiar operation is the modulo operation `%`, which calculates the remainder of a division. The order of operations is also pretty straightforward: the operations are calculated from left to right taking the parentheses into account.

```
int first = 2;   // variable of whole number type is assigned the value 2
int second = 4;  // variable of whole number type is assigned the value 4
int sum = first + second;  // variable of whole number type is assigned the value of first + second
                           //     (which means 2 + 4)

System.out.println(sum); // the value of the sum of variables is printed
```

```
int calcWithParens = (1 + 1) + 3 * (2 + 5);  // 23
int calcWithoutParens = 1 + 1 + 3 * 2 + 5;   // 13
```

The parentheses example above can also be done step by step.

```
int calcWithParens = (1 + 1);
calcWithParens = calcWithParens + 3 * (2 + 5);  // 23

int calcWithoutParens = 1 + 1;
calcWithoutParens = calcWithoutParens + 3 * 2;
calcWithoutParens = calcWithoutParens + 5;      // 13
```

Calculation operations can be used almost anywhere in the program code.

```
int first = 2;
int second = 4;

System.out.println(first + second);
System.out.println(2 + second - first - second);
```

5.1 Floating point numbers (decimal numbers)

Calculating the division and remainder of whole numbers is a little trickier. A floating point number (decimal number) and integer (whole number) often get mixed up. If all the variables in a calculation operation are integers, the end result will also be an integer.

```
int result = 3 / 2;  // result is 1 (integer) because 3 and 2 are integers as well
```

```
int first = 3:
int second = 2;
double result = first / second;  // the result is again 1 because first and second are integers
```

The remainder can be calculated using the remainder operation (`%`). For example, the calculation `7 % 2` yields 1.

```
int remainder = 7 % 2;  // remainder is 1 (integer)
```

If either the dividend or the divisor (or both!) is a floating point number (decimal number) the end result will also be a floating point number.

```
double whenDividendIsFloat = 3.0 / 2;  // result is: 1.5
double whenDivisorIsFloat = 3 / 2.0;   // result is: 1.5
```

If needed, integers can be converted to floating point using the type cast operation `(double)` as follows:

```
int first = 3;
int second = 2;
double result1 = (double)first / second;  // result is: 1.5

double result2 = first / (double)second;  // result is: 1.5

double result3 = (double)(first / second);  // result is: 1
```

In the last example calculation, the result is rounded incorrectly because the calculation between the integers is done before the type cast to a floating point number.

If the quotient is assigned to a variable of integer type, the result will be an integer as well.

```
int integerResultBecauseTypeIsInteger = 3.0 / 2;  // quotient is automatically integer: 1
```

The next example will print "1.5" because the dividend is transformed into a floating point number by multiplying it with a floating point number (1.0 * 3 = 3.0) before the division.

```
int dividend = 3;
int divisor = 2;

double quotient = 1.0 * dividend / divisor;
System.out.println(quotient);
```

What does the following code print?

```
int dividend = 3;
int divisor = 2;

double quotient = dividend / divisor * 1.0;
System.out.println(quotient);
```

From now on, make sure that you name your variables that follow good conventions like the variables in the examples above.

Exercise 5: Seconds in a year

Create a program that counts how many seconds there are in a year. You can assume that a year consists of 365 days (therefore the year is not a leap year).

The program should print the following:

```
There are X seconds in a year.
```

X should be replaced with the calculation of your program.

6. Concatenation or combining strings

Let us take a closer look on combining strings with the `+` operator.

If the `+` operator is used between two strings, a new string is created with the two strings combined. Note the clever use of space characters in the values of the variables below!

```
String greeting = "Hi ";
String name = "John";
String goodbye = ", and goodbye!";

String sentence = greeting + name + goodbye;

System.out.println(sentence);
```

```
Hi John, and goodbye!
```

If a string is on either side of the `+` operator, the other side is converted to a string and a new string is created. For example, the integer `2` will be converted into the string "2" and then combined with the other string.

```
System.out.println("there is an integer --> " + 2);
System.out.println(2 + " <-- there is an integer");
```

What we learned earlier about the order of operations is still valid:

```
System.out.println("Four: " + (2 + 2));
System.out.println("But! Twenty-two: " + 2 + 2);
```

```
Four: 4
But! Twenty-two: 22
```

Using this information, we can print a mix of strings and values of variables:

```
int x = 10;

System.out.println("variable x has the following value: " + x);

int y = 5;
int z = 6;

System.out.println("y has the value  " + y + " and z has the value " + z);
```

This program obviously prints:

```
variable x has the following value: 10
y has the value 5 and z has the value 6
```

Exercise 6: Addition

Create a program to calculate the sum of two numbers. At the beginning of the program two variables are introduced and those variables hold the numbers to be summed. Feel free to use other variables if you need to.

For example, if the variables hold numbers 5 and 4, the program should output:

```
5 + 4 = 9
```

If the variables hold numbers 73457 and 12888, the program output should be:

```
73457 + 12888 = 86345
```

Exercise 7: Multiplication

Create a program similar to the previous one except that it multiplies the two numbers instead of adding them.

For example, if the variables hold numbers 2 and 8, the program output should be:

```
2 * 8 = 16
```

If the variables hold numbers 277 and 111, the program output should be:

```
277 * 111 = 30747
```

What is the biggest multiplication that your program is able to calculate?

7. Reading user input

So far our programs have been rather one-sided. Next we will learn how to read *input* from the user. We will use a special *Scanner* tool to read the user input.

Let us add the *Scanner* to our existing main program body. Do not worry if the main program body seems obscure as we will continue to write our code in the part marked *// program code*.

```
import java.util.Scanner;

public class ProgramBody {
    public static void main(String[] args) {
        Scanner reader = new Scanner(System.in);

        // program code
    }
}
```

7.1 Reading a string

The following code reads the user's name and prints a greeting:

```
System.out.print("What is your name? ");
String name = reader.nextLine(); // Reads a line of input from the user and assigns it
                                 //     to the variable called name

System.out.println("Hi, " + name);
```

```
What is your name? John
Hi, John
```

The program above combined along with the main program body is shown below. The name of the program is *Greeting*, which means that it must be located in a file named `Greeting.java`.

```
import java.util.Scanner;

public class Greeting {
    public static void main(String[] args) {
        Scanner reader = new Scanner(System.in);

        System.out.print("Who is greeted: ");
        String name = reader.nextLine(); // Reads a line of input from the user and assigns it
                                         //     to the variable called name
        System.out.print("Hi " + name);
    }
}
```

When the program above is executed, you can type the input. The output tab in NetBeans (at the bottom) looks as follows when the program has finished (the user inputs the name "John").

```
run:
Who is greeted: John
Hi John
BUILD SUCCESSFUL (total time: 6 seconds)
```

7.2 Reading integers

Our Scanner tool is not good for reading integers, so we will use another special tool to read an integer. The command `Integer.parseInt` converts the string given to it into an integer. The command's parameter is given between brackets and it returns an integer that can be assigned to an integer variable.

Basically, we are joining two commands together. First we read the input as a string from the user and immediately give it to the command `Integer.parseInt`.

```
System.out.print("Type an integer: ");
int number = Integer.parseInt(reader.nextLine());

System.out.println("You typed " + number);
```

Next we will ask the user to give us his name and age. The program body is included this time.

```
import java.util.Scanner;

public class NameAndAgeGreeting {
    public static void main(String[] args) {
        Scanner reader = new Scanner(System.in);

        System.out.print("Your name: ");
        String name = reader.nextLine();   // Reads a line from the users keyboard

        System.out.print("How old are you: ");
        int age = Integer.parseInt(reader.nextLine()); // Reads a string variable from the keyboard and transfers it to an integer

        System.out.println("Your name is: " + name + ", and you are " + age + " years old, nice to meet you!");
    }
}
```

7.3 Summary

The program body for interaction with the user is as follows:

```
import java.util.Scanner;
public class ProgramName {
    public static void main(String[] args) {
        Scanner reader = new Scanner(System.in);

        // code here
    }
}
```

Reading a string:

```
String text = reader.nextLine();
```

Reading an integer:

```
int number = Integer.parseInt(reader.nextLine());
```

Exercise 8: Adder

Create a program that asks the user for two integers and then prints their sum.

```
Type a number: 6
Type another number: 2

Sum of the numbers: 8
```

In this example the user input is marked in red color. From now on the red color will indicate user input in examples.

Exercise 9: Divider

Create a program that asks the user for two integers and prints their quotient. Make sure that `3 / 2 = 1.5`. If the decimal part is missing, take another look at 6.1 Floating point numbers (decimal numbers) to find the solution.

```
Type a number: 3
Type another number: 2

Division: 3 / 2 = 1.5
```

Exercise 10: Calculating the circumference

The circumference of a circle is calculated using the formula `2 * pi * radius`. Create a program that asks the user for the radius and then calculates the circumference using the given radius. Java already contains the value of pi in the `Math.PI` variable, which you can use in your calculation.

```
Type the radius: 20

Circumference of the circle: 125.66370614359172
```

Exercise 11: Bigger number

Create a program that asks the user for two integers and then prints the larger of the two.

Tip: When you write `Math.` (that is, Math followed by a dot) in NetBeans, it shows you a bunch of available mathematical calculations. For example, `Math.cos(10)` calculates the cosine of the number 10. Try to find a suitable tool in `Math` to complete this exercise! If you cannot find anything suitable or do not know how to complete the exercise, skip the exercise for now. We will return to the matter later on.

```
Type a number: 20
Type another number: 14

The bigger number of the two numbers given was: 20
```

Exercise 12: Sum of the ages

Create a program that asks for the names and ages of two users. After that the program prints the sum of their ages.

```
Type your name: Matti
Type your age: 14

Type your name: Arto
Type your age: 12

Matti and Arto are 26 years old in total.
```

Exercise 13: NHL statistics, part 1

A ready-made component `NHLStatistics` is included along with the exercise files for this excercise. It can be used to fetch and see NHL players' score data, including their number of played games, goals, assists, points, and penalty amount.

The main program imports (i.e. enables the use of) the component by adding the following line to the beginning of the file: `import nhlstats.NHLStatistics;`. The next example prints the top 10 players based on points:

```java
import java.util.Scanner;
import nhlstats.NHLStatistics;

public class Mainprogram {
    public static void main(String[] args) throws Exception {
        Scanner reader = new Scanner(System.in);

        System.out.println("Top ten by points");
        NHLStatistics.sortByPoints();
        NHLStatistics.top(10);
    }
}
```

It will print (this was the situation on the 9th of January 2012):

```
Top ten by points
Henrik Sedin          VAN       43 11 + 38= 49  36
Phil Kessel           TOR       41 24 + 24= 48  10
Claude Giroux         PHI       36 18 + 30= 48  16
Joffrey Lupul         TOR       41 19 + 28= 47  36
Daniel Sedin          VAN       42 18 + 29= 47  32
Steven Stamkos        TBL       40 28 + 17= 45  34
Marian Hossa          CHI       41 17 + 27= 44  14
Evgeni Malkin         PIT       33 16 + 28= 44  30
Jordan Eberle         EDM       41 17 + 26= 43   6
Jason Pominville      BUF       41 14 + 29= 43   8
```

The name, abbreviation of the club, matches played, assists, points and penalties of players are printed.

The first command `NHLStatistics.sortByPoints()` sorts the list of NHL players by the points they have gathered. The second command `NHLStatistics.top(10);` prints the ten first players from the list. Any integer can be given as a parameter.

Similarly the players can be printed ordered by the goals or assists they have made, or by penalty minutes they have been given. First, we call the command to sort the players:

```java
NHLStatistics.sortByPoints();     // orders the players by points
NHLStatistics.sortByGoals();      // orders the players by goals
NHLStatistics.sortByAssists();    // orders the players by assists
NHLStatistics.sortByPenalties();  // orders the players by penalty minutes
```

After that the players are printed with the command `top` using the number of players to be printed as its parameter.

It is also possible to use the component to request the statistics of a certain player:

```java
NHLStatistics.searchByPlayer("Jaromir Jagr");  // prints stats of Jaromir Jagr
NHLStatistics.searchByPlayer("Koivu");         // prints stats of Mikko Koivu and Saku Koivu
NHLStatistics.searchByPlayer("Teemu");         // prints stats of all players named Teemu
```

The component can also print the statistics of all players in a club:

```
NHLStatistics.teamStatistics("NYR");  // Statistics of New York Rangers
```

The order of the printed club statistics can be changed using a `sortBy...()` first.

The name of the club must be given using the official three letter abbreviation. You can check the abbreviations here . The statistics component prints a list of the available abbreviations if you request the statistics of an invalid club.

Create a program that does the following tasks into the main program body. The tasks must be done in exactly the same order as listed below. Do the tasks in the program body one after another without deleting tasks you have already done.

Note: When you first run the program, the execution might take a while because the information is downloaded from the web. Execution should be quick after the first run.

The program must do the following:

- Print the top ten players based on goals
- Print the top 25 players based on penalty amounts
- Print the statistics for Sidney Crosby
- Print the statistics for Philadelphia Flyers (abbreviation: PHI). Note in which order the players are printed in and why that might be!
- Print the players in Anaheim Ducks (abbreviation: ANA) ordered by points

After you have successfully submitted the exercise, you can play with the code as you wish!

8. Conditional statements and truth values

So far, our programs have progressed from one command to another in a straightforward manner. In order for the program to *branch* to different execution paths based on e.g. user input, we need conditional statements.

```
int number = 11;

if (number > 10) {
    System.out.println("The number was greater than 10");
}
```

The condition `(number > 10)` evaluates into a truth value; either `true` or `false`. The `if` command only handles truth values. The conditional statement above is read as "if the number is greater than 10".

Note that the `if` statement is not followed by semicolon as the condition path continues after the statement.

After the condition, the opening curly brace `{` starts a new *block*, which is executed if the condition is true. The *block* ends with a closing curly brace `}`. Blocks can be as long as desired.

The comparison operators are:

- `>` Greater than
- `>=` Greater than or equal to
- `<` Less than
- `<=` Less than or equal to
- `==` Equals
- `!=` Not equal

```
int number = 55;

if (number != 0) {
    System.out.println("The number was not equal to 0");
}

if (number >= 1000) {
    System.out.println("The number was greater than or equal to 1000");
}
```

A block can contain any code including other `if` statements.

```
int x = 45;
int number = 55;

if (number > 0) {
    System.out.println("The number is positive!");
    if (number > x) {
        System.out.println(" and greater than the value of variable x");
        System.out.println("after all, the value of variable x is " + x);
    }
}
```

The comparison operators can also be used outside the `if` statements. In such case the truth value will be stored in a truth value variable.

```
int first = 1;
int second = 3;

boolean isGreater = first > second;
```

In the example above the boolean (i.e. a truth value) variable `isGreater` now includes the truth value *false*.

A boolean variable can be used as a condition in a conditional sentence.

```
int first = 1;
int second = 3;

boolean isLesser = first < second;

if (isLesser) {
    System.out.println(first + " is less than " + second + "!");
}
```

```
1 is less than 3!
```

8.1 Code indentation

Note that the commands in the block following the `if` statement (i.e. the lines after the curly brace, `{`) are not written at the same level as the `if` statement itself. They should be **indented** slightly to the right. Indentation happens when you press the tab key, which is located to the left of q key. When the block ends with the closing curly brace, indentation ends as well. The closing curly brace `}` should be on the same level as the original `if` statement.

The use of indentation is crucial for the readability of program code. During this course and generally everywhere, you are expected to indent the code properly. NetBeans helps with the correct indentation. You can easily indent your program by pressing shift, alt, and f simultaneously.

8.2 else

If the truth value of the comparison is false, another optional block can be executed using the `else` command.

```
int number = 4;

if (number > 5) {
    System.out.println("Your number is greater than five!");
} else {
    System.out.println("Your number is equal to or less than five!");
}
```

```
Your number is equal to or less than five!
```

Exercise 14: A positive number

Create a program that asks the user for a number and tells if the number is positive (i.e. greater than zero).

```
Type a number: 5

The number is positive.
```

```
Type a number: -2

The number is not positive.
```

Are you certain that your code is indented correctly?

Reread the section on code indentation. Observe what happens when you press shift, alt and f simultaneously! The same automatic indentation functionality can also be used using the menu bar by selecting Source and then Format.

Exercise 15: Age of majority

Create a program that asks for the user's age and tells whether the user has reached the age of majority (i.e. 18 years old or older).

```
How old are you? 12

You have not reached the age of majority yet!
```

```
How old are you? 32

You have reached the age of majority!
```

Exercise 16: Even or odd?

Create a program that asks the user for a number and tells whether the number is even or odd.

```
Type a number: 2
Number 2 is even.
```

```
Type a number: 7
Number 7 is odd.
```

Hint: The number's remainder when dividing by 2 tells whether the number is even or odd. The remainder can be obtained with the `%` operator.

8.3 else if

If there are more than two conditions for the program to check, it is recommended to use the `else if` command. It works like the `else` command, but with an additional condition. `else if` comes after the `if` command. There can be multiple `else if` commands.

```
int number = 3;

if (number == 1) {
    System.out.println("The number is one.");
} else if (number == 2) {
    System.out.println("The number is two.");
} else if (number == 3) {
    System.out.println("The number is three!");
} else {
    System.out.println("Quite a lot!");
}
```

```
The number is three!
```

Let us read out loud the example above: If number is one, print out "The number is one.". Otherwise if the number is two, print out "The number is two.". Otherwise if the number is three, print out "The number is three!". Otherwise print out "Quite a lot!".

8.4 Comparing strings

Strings cannot be compared using the equality operator (==). For string comparison, we use the `equals`. command, which is always associated with the string to compare.

```
String text = "course";

if (text.equals("marzipan")) {
    System.out.println("The variable text contains the text marzipan");
} else {
    System.out.println("The variable text does not contain the text marzipan");
}
```

The `equals` command is always attached to the string variable with a dot in between. A string variable can also be compared to another string variable.

```
String text = "course";
String anotherText = "horse";

if (text.equals(anotherText)) {
    System.out.println("The texts are the same!");
} else {
    System.out.println("The texts are not the same!");
}
```

When comparing strings, it is crucial to make sure that both string variables have been assigned some value. If a value has not been assigned, the program execution terminates with a *NullPointerException* error, which means that variable has no value assigned to it (*null*).

Exercise 17: Greater number

Create a program that asks the user for two numbers and prints the greater of those two. The program should also handle the case in which the two numbers are equal.

Example outputs:

```
Type the first number: 5
Type the second number: 3

Greater number: 5
```

```
Type the first number: 5
Type the second number: 8

Greater number: 8
```

```
Type the first number: 5
Type the second number: 5

The numbers are equal!
```

Exercise 18: Grades and points

Create a program that gives the course grade based on the following table.

Points	Grade
0–29	failed
30–34	1
35–39	2
40–44	3
45–49	4
50–60	5

Example outputs:

```
Type the points [0-60]: 37

Grade: 2
```

```
Type the points [0-60]: 51

Grade: 5
```

8.5 Logical operations

The condition statements can be made more complicated using logical operations. The logical operations are:

- `condition1 && condition2` is true if both conditions are true.
- `condition1 || condition2` is true if either of the conditions are true.
- `!condition` is true if the condition is false.

Below we will use the AND operation `&&` to combine two individual conditions in order to check if the value of the variable is greater than 4 **and** less than 11 (i.e. in the range 5 - 10).

```
System.out.println("Is the number between 5-10?");
int number = 7;

if (number > 4 && number < 11) {
    System.out.println("Yes! :)");
} else {
    System.out.println("Nope :(")
}
```

```
Is the number between 5-10?
Yes! :)
```

Next up is the OR operation `||`, which will be used to check if the value is less than 0 **or** greater than 100. The condition evaluates to true if the value fulfills either condition.

```
System.out.println("Is the number less than 0 or greater than 100?");
int number = 145;

if (number < 0 || number > 100) {
    System.out.println("Yes! :)");
} else {
    System.out.println("Nope :(")
}
```

```
Is the number less than 0 or greater than 100?
Yes! :)
```

Now we will use the negation operation `!` to negate the condition:

```
System.out.println("Is the string equal to 'milk'?");
String text = "water";

if (!(text.equals("milk"))) {  // true if the condition text.equals("milk") is false
    System.out.println("No!");
} else {
    System.out.println("Yes")
}
```

```
Is the text equal to 'milk'?
No!
```

For complicated conditions, we often need parentheses:

```
int number = 99;

if ((number > 0 && number < 10) || number > 100 ) {
    System.out.println("The number was in the range 1-9 or it was over 100");
} else {
    System.out.println("The number was equal to or less than 0 or it was in the range 10-99");
}
```

```
The number was equal to or less than 0 or it was in the range 10-99
```

Exercise 19: Age check

Create a program that asks for the user's age and checks that it is reasonable (at least 0 and at most 120).

```
How old are you? 10
OK
```

```
How old are you? 55
OK
```

```
How old are you? -3
Impossible!
```

```
How old are you? 150
Impossible!
```

Exercise 20: Usernames

Create a program that recognizes the following users:

Username	Password
alex	mightyducks
emily	cat

The program should check for the username and password as follows:

```
Type your username: alex
Type your password: mightyducks
You are now logged into the system!
```

```
Type your username: emily
Type your password: cat
You are now logged into the system!
```

```
Type your username: emily
Type your password: dog
Your username or password was invalid!
```

Note: Remember that you cannot compare strings with the `==` operation!

Exercise 21: Leap year

A year is a leap year if it is divisible by 4. But if the year is divisible by 100, it is a leap year only when it is also divisible by 400.

Create a program that checks whether the given year is a leap year.

```
Type a year: 2011
The year is not a leap year.
```

```
Type a year: 2012
The year is a leap year.
```

```
Type a year: 1800
The year is not a leap year.
```

```
Type a year: 2000
The year is a leap year.
```

9. Introduction to loops

Conditional statements allow us to execute different commands based on the conditions. For example, we can let the user login only if the username and password are correct.

In addition to conditions we also need repetitions. We may, for example, need to keep asking the user to input a username and password until a valid pair is entered.

The most simple repetition is an infinite loop. The following code will print out the string *I can program!* forever or "an infinite number of times":

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

In the example above, the `while (true)` command causes the associated block (i.e. the code between the curly braces `{}`) to be *looped* (or repeated) infinitely.

We generally do not want an infinite loop. The loop can be interrupted using e.g. the `break` command.

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

    System.out.print("Continue? ('no' to quit)? ");
    String command = reader.nextLine();
    if (command.equals("no")) {
        break;
    }
}

System.out.println("Thank you and see you later!");
```

Now the loop progresses like this: First, the program prints *I can program!*. Then, the program will ask the user if it should continue. If the user types *no*, the `break` command is executed and the loop is interrupted and *Thank you and see you again!* is printed.

```
I can program!
Continue? ('no' to quit)?yeah
I can program!
Continue? ('no' to quit)? jawohl
I can program!
Continue? ('no' to quit)? no
Thank you and see you again!
```

Many different things can be done inside a loop. Next we create a simple calculator, which performs calculations based on commands that the user enters. If the command is *quit*, the `break` command will be executed to end the loop. Otherwise two numbers are asked. Then, if the initial command was *sum*, the program calculates and prints the sum of the two numbers. If the command was *difference*, the program calculates and prints the difference of the two numbers. If the command was something else, the program reports that the command was unknown.

```
System.out.println("welcome to the calculator");

while (true) {
    System.out.print("Enter a command (sum, difference, quit): ");
    String command = reader.nextLine();
    if (command.equals("quit")) {
        break;
    }

    System.out.print("enter the numbers");
    int first = Integer.parseInt(reader.nextLine());
    int second = Integer.parseInt(reader.nextLine());

    if (command.equals("sum") ) {
        int sum = first + second;
        System.out.println( "The sum of the numbers is " + sum );
    } else if (command.equals("difference")) {
        int difference = first - second;
        System.out.println("The difference of the numbers is " + difference);
    } else {
        System.out.println("Unknown command");
    }
}

System.out.println("Thanks, bye!");
```

Exercise 22: Password

In this exercise we create a program that asks the user for a password. If the password is right, a secret message is shown to the user.

```
Type the password: turnip
Wrong!
Type the password: beetroot
Wrong!
Type the password: carrot
Right!

The secret is: jryy qbar!
```

The program will be done in three steps.

Exercise 22.1: Asking for the password

The initial exercise template defines the variable `String password` with a value of `carrot`. Do not change this password! You should make the program ask the user to enter a password and then compare it with the value in the variable `password`. Remember what that there is a special way to compare strings!

```
Type the password: turnip
Wrong!
```

```
Type the password: carrot
Right!
```

```
Type the password: potato
Wrong!
```

Exercise 22.2: Asking for the password until the user gives the correct one

Modify the program so that it asks the user to type a password until it gets the correct one. Implement this using a `while-true` loop statement. The loop statement can be interrupted if and only if the entered password matches the value of the `password` variable.

```
Type the password: turnip
Wrong!
Type the password: beetroot
Wrong!
Type the password: carrot
Right!
```

Exercise 22.3: Secret message

Add your own secret message to the program and show it to the user when the password is correct. Your message can be whatever you want!

```
Type the password: turnip
Wrong!
Type the password: rutabaga
Wrong!
Type the password: carrot
Right!
```

```
The secret is: jryy qbar!
```

The secret above has been encrypted using the Rot13 algorithm. During this course we will implement our own encryption program.

Exercise 23: Temperatures

You will get the `Graph` component along with the exercise template. `Graph` draws graphs based on numbers that are given to it. You can give it numbers as follows:

```
Graph.addNumber(13.0);
```

We will create a program that draws a graph based on daily temperatures given to it.

Exercise 23.1: Asking for numbers

Create a program that asks the user to input floating point numbers (`double`) and then adds the numbers to the graph. Use the `while-true` structure again.

Note: To read a `double`, use: `double number = Double.parseDouble(reader.nextLine());`

Exercise 23.2: Checking

Improve your program so that temperatures below -30 degrees or over +40 degrees are ignored and not added to the graph.

Exercise 24: NHL statistics, part 2

We will continue using the NHL component introduced earlier and create a program that the user can use to query for statistics.

The program is structured similarly to the Calculator example program above. The program body is as follows:

```
public static void main(String[] args) throws Exception {
    Scanner reader = new Scanner(System.in);

    System.out.println("NHL statistics service");
    while (true) {
        System.out.println("");
        System.out.print("command (points, goals, assists, penalties, player, club, quit): ");
        String command = reader.nextLine();

        if (command.equals("quit")) {
            break;
        }

        if (command.equals("points")) {
            // print the top ten playes sorted by points
        } else if (command.equals("goals")) {
            // print the top ten players sorted by goals
        } else if (command.equals("assists")) {
            // print the top ten players sorted by assists
        } else if (command.equals("penalties")) {
            // print the top ten players sorted by penalties
        } else if (command.equals("player")) {
            // ask the user for the player name and print the statistics for that player
        } else if (command.equals("club")) {
            // ask the user for the club abbreviation and print the statistics for the club
            // note: the statistics should be sorted by points
            //     (players with the most points are first)
        }
    }
}
```

The program asks the user to give commands and then executes the operation that is associated with the given command. The commands are: *points, goals, assists, penalties, player, club, quit*.

You should write code in the parts marked with comments.

Here is an example demonstrating the program in action:

```
NHL statistics service

command (points, goals, assists, penalties, player, club): assists
Henrik Sedin           VAN        43 11 + 38= 49  36
Erik Karlsson          OTT        43  6 + 35= 41  24
Claude Giroux          PHI        36 18 + 30= 48  16
Pavel Datsyuk          DET        41 13 + 30= 43  10
Brian Campbell         FLA        42  3 + 30= 33   4
Daniel Sedin           VAN        42 18 + 29= 47  32
Jason Pominville       BUF        41 14 + 29= 43   8
Nicklas Backstrom      WSH        38 13 + 29= 42  22
Joffrey Lupul          TOR        41 19 + 28= 47  36
Evgeni Malkin          PIT        33 16 + 28= 44  30

command (points, goals, assists, penalties, player, club): player
which player: Jokinen
Olli Jokinen           CGY        43 12 + 21= 33  32
Jussi Jokinen          CAR        40  4 + 19= 23  30

command (points, goals, assists, penalties, player, club): club
which club: DET
Pavel Datsyuk          DET        41 13 + 30= 43  10
Johan Franzen          DET        41 16 + 20= 36  34
Valtteri Filppula      DET        40 14 + 21= 35  10
Henrik Zetterberg      DET        41  8 + 24= 32  14
// and more players

command (points, goals, assists, penalties, player, club): quit
```

Note: When you first run the program, the execution might take a while because the information is downloaded from the internet. Execution should be quick after the first run.

Help: IRCnet/Matrix #mooc.fi | News:

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

TIETOJENKÄSITTELYTIETEEN LAITOS
INSTITUTIONEN FÖR DATAVETENSKAP
DEPARTMENT OF COMPUTER SCIENCE