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

10. Changing variables

We usually want to change the value of an existing variable. This can be done using the normal assignment statement. In the next example, we increase the value of the variable `age` by one:

```
int age = 1;

System.out.println(age);  // prints 1
age = age + 1;            // the new value of age is the old value of age + 1
System.out.println(age);  // prints 2
```

The `age = age + 1` statement increments the value of the variable `age` by one. It is also possible to increment a variable by one as bellow:

```
int age = 1;

System.out.println(age);  // prints 1
age++;                    // means the same as age = age + 1
System.out.println(age);  // prints 2
```

Another example:

```
int length = 100;

System.out.println(length);  // prints 100
length = length - 50;
System.out.println(length);  // prints 50
length = length * 2;
System.out.println(length);  // prints 100
length = length / 4;
System.out.println(length);  // prints 25
length--;                    // means the same as length = length - 1;
System.out.println(length);  // prints 24
```

Exercise 25: Sum of three numbers

Create a program that asks the user for three numbers and then prints their sum. Use the following structure in your program:

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

// WRITE YOUR PROGRAM HERE
// USE ONLY THE VARIABLES sum, reader AND read!

System.out.println("Sum: " + sum);
```

```
Type the first number: 3
Type the second number: 6
Type the third number: 12

Sum: 21
```

Exercise 26: Sum of many numbers

Create a program that reads numbers from the user and prints their sum. The program should stop asking for numbers when user enters the number 0. The program should be structured like this:

```
Scanner reader = new Scanner(System.in);
int sum = 0;
while (true) {
    int read = Integer.parseInt(reader.nextLine());
    if (read == 0) {
        break;
    }

    // DO SOMETHING HERE

    System.out.println("Sum now: " + sum);
}

System.out.println("Sum in the end: " + sum);
```

The program should work as follows:

```
3
Sum now: 3
2
Sum now: 5
1
Sum now: 6
1
Sum now: 7
0
Sum in the end: 7
```

11. More loops

we have previously learned to use repetition using the `while (true)` loop, which repeats commands until the `break` command is used.

The `break` command is not the only way to end a loop. A common structure for a loop is `while (condition)`, where the condition can be any statement with a truth value. This means that the condition works exactly like conditions in an `if` statements.

In the following example, we print the numbers 1, 2, ..., 10. When the value of the variable `number` increases above 10, the condition of the `while` statement is no longer true and the loop ends.

```
int number = 1;

while (number < 11) {
    System.out.println(number);
    number++;  // number++ means the same as number = number + 1
}
```

The example above can be read "as long as the variable number is less than 11, print the variable and increment it by one".

Above, the variable `number` was incremented in each iteration of the loop. Generally the change can be anything, meaning that the variable used in the condition does not always need to be incremented. For example:

```
int number = 1024;

while (number >= 1) {
    System.out.println(number);
    number = number / 2;
}
```

A few programming-part1 tips

- You can find all the NetBeans tips here.
- **Auto-completion of your code**

  If you have, for example, the variable `String familyName;` in your code, you do not need to always write `familyName`. Try what happens when you type in `f` and then press CTRL and space simultaneously. You can also use auto-completion with commands like `while` by typing in `w` and then CTRL + space.

- **sout**

  Remember that you can get the text *System.out.println("")* by typing *sout* and pressing the *tab* key (located to the left of the q key)

Complete the following exercises using the `while` statement:

Exercise 27: From one to a hundred

Create a program that prints the integers (whole numbers) from 1 to 100.

The program output should be:

```
1
2
3
(many rows of numbers here)
98
99
100
```

Exercise 28: From hundred to one

Create a program that prints the integers (whole numbers) from 100 to 1.

The program output should be:

```
100
99
98
(many rows of numbers here)
3
2
1
```

Tip: Assign the variable you use in the condition of the loop a initial value of 100 and then subtract one on each iteration of the loop.

Exercise 29: Even numbers

Create a program that prints all even numbers between 2 and 100.

```
2
4
6
(many rows of numbers here)
96
```

```
98
100
```

Exercise 30: Up to a certain number

Create a program that prints all whole numbers from 1 to the number the user enters.

```
Up to what number? 3
1
2
3
```

```
Up to what number? 5
1
2
3
4
5
```

Tip: The number you read from the user now works as the upper limit in the condition of the `while` statement. Remember that in Java `a <= b` means *a is less than or equal to b*.

Exercise 31: Lower limit and upper limit

Create a program that asks the user for the first number and the last number and then prints all numbers between those two.

```
First: 5
Last: 8
5
6
7
8
```

If the first number is greater than the last number, the program prints nothing:

```
First: 16
Last: 12
```

Note: Remember that the lower and upper limits can also be negative!

11.1 Assignment operations

Because changing the value of a variable is a very common operation, Java has special assignment operations for it.

```
int length = 100;

length += 10;  // same as length = length + 10;
length -= 50;  // same as length = length - 50;
```

When performing the assignment operation on an existing variable, it is written as `variable operation= change`, for example `variable += 5`. Note that a variable must be defined before you can assign a value to it. Defining a variable is done by specifying the variable type and the name of the variable.

The following example will not work because the type of the variable `length` has not been defined.

```
length = length + 100;  // error!
length += 100;          // error!
```

When the type is defined, the operations will also work.

```
int length = 0;
length = length + 100;
length += 100;

// the variable length now holds the value 200
```

There are also other assignment operations:

```
int length = 100;

length *= 10;   // same as length = length * 10;
length /= 100;  // same as length = length / 100;
length %= 3;    // same as length = length % 3;

// the variable length now holds the value 1
```

Often during a loop, the value of a variable is calculated based on repetition. The following program calculates 3*4 somewhat clumsily as the sum 3+3+3+3:

```
int result = 0;

int i = 0;
while (i < 4) {
   result = result + 3;
   i++;  // means the same as i = i + 1;
}
```

In the beginning `result = 0`. During the loop, the value of the variable is incremented by 3 on each iteration. Because there are 4 iterations, the value of the variable is 3*4 in the end.

Using the assignment operator introduced above, we can achieve the same behavior as follows:

```
int result = 0;

int i = 0;
while (i < 4) {
   result += 3;  // this is the same as result = result + 3;
   i++;          // means the same as i = i+1;
}
```

Exercise 32: The sum of a set of numbers

Create a program that calculates the sum 1+2+3+...+n where n is a number entered by the user.

Example outputs:

```
Until what? 3
Sum is 6
```

The calculation above was: 1+2+3 = 6.

```
Until what? 7
Sum is 28
```

The calculation above was: 1+2+3+4+5+6+7 = 28.

Hint: Create the program using the `while` statement. Use a helper variable in your program to remember how many times the block has been executed. Use also another helper variable to store the sum. During each execution add the value of the helper variable that counts the executions to the variable in which you should collect the sum.

Exercise 33: The sum between two numbers

Similar to the previous exercise, except that the program should ask for both the lower and upper bound. You can assume that the users first gives the smaller number and then the greater number.

Example outputs:

```
First: 3
Last: 5
The sum 12
```

```
First: 2
Last: 8
The sum is 35
```

Exercise 34: Factorial

Create a program that calculates the factorial of the number n. The factorial n! is calculated using the formula 1*2*3*...*n. For example 4! = 1*2*3*4 = 24. Additionally, it is defined that 0! = 1.

Example outputs:

```
Type a number: 3
Factorial is 6
```

```
Type a number: 10
Factorial is 3628800
```

Exercise 35: Sum of the powers

Create a program that calculates the sum of $2^0+2^1+2^2+...+2^n$, where n is a number entered by the user. The notation 2^i means raising the number 2 to the power of i, for example 2^4 = 2*2*2*2 = 16. In Java we cannot write a^b directly, but instead we can calculate the power with the command `Math.pow(number, power)`. Note that the command returns a number of `double` type (i.e. floating point number). A double can

be converted into the `int` type (i.e. whole number) as follows: `int result = (int)Math.pow(2, 3)`. This assigns the value of 2^3 to variable result.

Example outputs:

```
Type a number: 3
The result is 15
```

```
Type a number: 7
The result is 255
```

11.2 Infinite loops

One of the classic errors in programming is to accidentally create an infinite loop. In the next example we try to print "Never again shall I program an eternal loop!" 10 times:

```
int i = 0;

while (i < 10) {
    System.out.println("Never again shall I program an eternal loop!");
}
```

The variable `i`, which determines is supposed to index the loops, is initially set to 0. The block is looped as long as the condition `i < 10` is true. But something funny happens. Because the value of the variable `i` is never changed, the condition stays true forever.

11.3 Ending a while loop

So far, we have used the while loop with a structure similar to this:

```
int i = 1;
while (i < 10) {
    // Some code.
    i++;
}
```

With the structure above, the variable `i` remembers the number of times the the loop has been executed. The condition to end the loop is based on comparing the value of `i`.

Let us now recall how a while loop is stopped. Ending a while loop does not always need to be based on the amount of loops. The next example program asks for the user's age. If the

given age is not in the range 5-85, the program prints a message and asks for the user's age again. As you can see, the condition for the while loop can be any expression that results in a boolean (truth value).

```
System.out.println("Type your age: ");

int age = Integer.parseInt(reader.nextLine());

while (age < 5 || age > 85) {  // age less than 5 OR greater than 85
    System.out.println("You are lying!");
    if (age < 5) {
        System.out.println("You are so young that you cannot know how to write!");
    } else if (age > 85) {
        System.out.println("You are so old that you cannot know how to use a computer!");
    }

    System.out.println("Type your age again: ");
    age = Integer.parseInt(reader.nextLine();
}

System.out.println("Your age is " + age);
```

The program could also have been implemented using the good old `while (true)` structure:

```
System.out.println("Type your age ");
int age;
while (true) {
    age = Integer.parseInt(reader.nextLine());

    if (age >= 5 && age <= 85) {  // age between 5 AND 85
        break;  // end the loop
    }

    System.out.println("You are lying!");
    if (age < 5) {
        System.out.println("You are so young that you cannot know how to write!");
    } else {  // that means age is over 85
        System.out.println("You are so old that you cannot know how to use a computer!");
    }

    System.out.println("Type your age again: ");
}

System.out.println("Your age is " + age);
```

Exercise 36: Loops, ending and remembering

This set of exercises will form one larger program when put together. We create the program by adding features exercise by exercise. If you do not finish all the exercises you can still send them to be reviewed by the exercise robot. To do that, click the "submit" button, which has a picture of an arrow and is located on the right of the testing button. Even though the exercise robot complains about tests in the incomplete exercises, you will still get points for the parts you have completed.

Note: from now on every sub-exercise of a larger exercise (like 36.1) has the same value as an exercise without sub-exercises. It means that exercise 36 as a whole corresponds to five normal exercises.

Exercise 36.1: Reading numbers

Create a program that asks the user to input numbers (integers). The program prints "Type numbers" until the user types the number -1. When the user types the number -1, the program prints "Thank you and see you later!" and ends.

```
Type numbers:
5
2
4
-1
Thank you and see you later!
```

Exercise 36.2: The sum of the numbers

Develop your number reading program by adding the following feature: the program should print the sum of the numbers entered by the user (without the number -1).

```
Type numbers:
5
2
4
-1
Thank you and see you later!
The sum is 11
```

Exercise 36.3: Summing and counting the numbers

Develop your number reading and summing program by adding the following feature: the program should print how many numbers the user typed (without the number -1).

```
Type numbers:
5
2
4
-1
Thank you and see you later!
The sum is 11
How many numbers: 3
```

Exercise 36.4: Counting the average

Develop your number reading, summing and counting program by adding the following feature: the program should print the average of the numbers the user

typed (without the number -1).

```
Type numbers:
5
2
4
-1
Thank you and see you later!
The sum is 11
How many numbers: 3
Average: 3.666666666666
```

Exercise 36.5: Even and odd numbers

Develop your program by adding the following feature: the program should print the number of even and odd numbers that the user typed (without the number -1).

```
Type numbers:
5
2
4
-1
Thank you and see you later!
The sum is 11
How many numbers: 3
Average: 3.666666666666
Even numbers: 2
Odd numbers: 1
```

Note: creating a program in small steps

In these exercises we actually created one single program, but programming happened in very small steps. This is **ALWAYS** the preferred way to program.

When you are programming something, no matter if it is an exercise or a project of your own, it is advised to do it in very tiny pieces. Do not ever try to solve the whole problem in one go. Start with something easy, something you know that you can do. In this recent set of exercises, for example, we focused first on stopping the program when the user types -1. When one part of the program is complete and working, we can move on to work out the solution for the next sub-problem of the big main problem.

Some of the exercises in this course are sliced into smaller pieces like the set of exercises we just introduced. Usually the pieces need to be sliced again into smaller pieces depending on the problem. It is advised that you execute the whole program after almost every new line of code you write. This enables you to be sure that your solution is going in the right and working direction.

12. Methods

We have so far used many different commands of Java: assignment, calculations, comparison, if structures and while structures. We have been using a "command" `System.out.println()` to print text. We can also count the maximum of two numbers with the help of the "command" `Math.max()`. We are also familiar with `reader.nextLine()`, usually seen together with `Integer.parseInt()`.

If we take a closer look, we notice that those commands differ from if and while (etc). The first difference is that after the command there are brackets () and sometimes an input for the command inside those brackets. Actually, the commands ending with brackets are not called commands, but **methods**.

Technically speaking, a method is a piece of code that can be called from different places of the program code. The line of code `System.out.println("I am a parameter given to the method!")` means that we call a method that actually handles the printing. After the method has been executed we go back to where we called the method, and continue executing. The input given to the method inside the brackets is called a *method parameter*.

In addition to a parameter, the method can also have a return value, for example, a familiar line of code:

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

includes two method calls. First the inner method `reader.nextLine` is called. That method has the integer typed by the user as a return value. Next the outer method `Integer.parseInt` is called. As a parameter for that method there is the string of characters that was received from the `reader.nextLine` method as a return value. The return value for the method `Integer.parseInt` is the string of characters transformed into an integer (whole number).

Method names also seem to include a dot, for example `reader.nextLine()`. Actually the method name starts after the dot, here it is `nextLine()`. The first part of the command that comes before the dot shows whose method is in question. Here the method belongs to the reader, which means that we have the *reader's* method *nextLine*. Later we will learn more precisely about the owner of the method (or the name on the left side of the dot). An attentive reader will notice that the method `System.out.println()` has two dots. Here, the method name is println and `System.out` is the owner of the method. Roughly `System.out` means the computer monitor.

This far we have been using ready-made methods from Java libraries. Next we will learn how to create our own methods.

13. Self-written methods

This far we have been using a programming style where code is written (and read and executed) from top to bottom.

It was mentioned before that "a method is a piece of code that can be called from different places of the program code". Ready-made methods of Java have been used since our very first program.

In addition to using these ready-made methods programmers can write their own methods for programs to call. In the real world, it is really exceptional if the program does not include any self-written methods. From now on almost every program we create during this course will include self-written methods.

The methods are written in the program body outside the main's braces ({ and }) but still inside the outermost braces, for example like this: :

```
import java.util.Scanner;

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

    // self-written methods
}
```

Let us create a method `greet`.

```
public static void greet() {
    System.out.println("Greetings from the world of methods!");
}
```

And let us place it in the right spot.

```
import java.util.Scanner;

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

    // self-written methods
    public static void greet() {
        System.out.println("Greetings from the world of methods!");
    }
}
```

In order to define a new method we need to write two things. In the first row of the method definition, you will find the name of the method, in this case *greet*. On the left side of the name you will find the definitions `public static void`. On the next line, the code block marked by the braces ({ and }). Inside it, the method's code, or the commands that will be executed when the method is called. Our method `greet` only writes one line of text to the screen.

It is easy to call a self-written method. It happens by writing the method name, brackets () and a semicolon. In the next example main (or the main program) calls for our method, first once and then several times.

```
import java.util.Scanner;

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

        // program code
        System.out.println("Let us try if we can get to the method world:");
        greet();

        System.out.println("It seems like we can, let us try again:");
        greet();
        greet();
        greet();
    }

    // self-written methods
    public static void greet() {
        System.out.println("Greetings from the world of methods!");
    }
}
```

When the program is executed, we see the following output:

```
Let us try if we can get to the method world:
Greetings from the world of methods!
It seems like we can, let us try again:
Greetings from the world of methods!
Greetings from the world of methods!
Greetings from the world of methods!
```

What is noteworthy here is the execution order of the program code. The execution starts with the main program's (or main's) lines of code, from top to bottom, one by one. When the line of code to be executed happens to be a method call, the lines of code in the method block are executed again one by one. When the method block ends, the execution continues from the place where the method was called. To be exact, the execution continues from the next line after the original method call.

To be even more exact, the main program is also a method. When the program starts, the operation system calls for the main method. That means that the main method is the starting point of the program and the execution starts from the first code line of main. The program execution ends when it reaches the end of main.

From now on when we introduce methods, we will not point out that they need to be written in the right place inside the program code. For example, a method cannot be defined inside another method.

Exercise 37: Printing out text

Create a method `printText` that prints the following string of characters: "In the beginning there were the swamp, the hoe and Java." and a line break.

```
public static void main(String[] args) {
    printText();
}

public static void printText() {
    // write your code here
}
```

The program output:

```
In the beginning there were the swamp, the hoe and Java.
```

Exercise 38: Many prints

Develop the program by adding the following feature: the main program should ask the user how many times the text should be printed (meaning how many times the method is called).

```
public static void main(String[] args) {
    // ask the user how many times the text should be printed
    // use the while structure to call the printText method several times
}

public static void printText() {
    // write your code here
}
```

The program output:

```
How many?
7
In the beginning there were the swamp, the hoe and Java.
In the beginning there were the swamp, the hoe and Java.
In the beginning there were the swamp, the hoe and Java.
In the beginning there were the swamp, the hoe and Java.
In the beginning there were the swamp, the hoe and Java.
In the beginning there were the swamp, the hoe and Java.
In the beginning there were the swamp, the hoe and Java.
```

Note: you should print the assisting question *How many?* on its own line!

13.1 Method parameters

We can make our methods more useful by giving it *parameters*! Parameters are variables that we define inside brackets in the first line, just after the method name. When the method is called, the parameters are assigned values.

In the next example we define a method with a parameter, its name will be `greet` and its parameter will be a variable of the type String called `name`.

```
public static void greet(String name) {
    System.out.println("Hi " + name + ", greetings from the world of methods!");
}
```

Let us next call the `greet` method so that on the first try we give its parameter the value `Matt` and on the second try `Arthur`.

```
public static void main(String[] args) {
    greet("Matt");
    greet("Arthur");
}
```

```
Hi Matt, greetings from the world of methods!
Hi Arthur, greetings from the world of methods!
```

More complicated expressions can also be used as a parameter for our self-written methods, the same way we used them together with the ready-made `System.out.println()` method.

```
public static void main(String[] args) {
    String name1 = "Anne";
    String name2 = "Green";
    greet( name1 + " " + name2 );

    int age = 24;
    greet("John " + age + " years");
}
```

```
Hi Anne Green, greetings from the world of methods!
Hi John 24 years, greetings from the world of methods!
```

In both cases the method has only one parameter. The value for the parameter is calculated before calling the method. In the first case the parameter value comes from the String concatenation (a cool word that means putting the text together) `name1 + " " + name2`. The value for the concatenation is *Anne Green*. In the second case we get the parameter value from the String concatenation `"John " + age + " years"`.

13.2 Many parameters

A method can be defined to have more than one parameter. In this case, the parameters are always listed in the same order.

```
public static void greet(String name, String greetingsFrom) {
    System.out.println("Hi " + name + ", greetings from " + greetingsFrom);
}
```

```
String who = "Matt";
String greetings = "Alabama";

greet(who, greetings);
greet(who, greetings + " from Nevada");
```

In the last `greet` function (or method) call the second parameter is formed by concatenating (or adding) the text “from Nevada” to the variable `greetings`. This is done before the actual function call.

```
Hi Matt, greetings from Alabama
Hi Matt, greetings from Alabama from Nevada
```

13.3 Method calling another method

Methods can also be called outside of main. Methods can call each other! Let us create a method `greetManyTimes` that greets the user many times getting assistance from the method `greet`:

```
public static void greet(String name) {
    System.out.println("Hi " + name + ", greetings from the world of methods!");
}

public static void greetManyTimes(String name, int times) {
    int i = 0;
    while ( i < times ) {
        greet(name);
        i++;
    }

}

public static void main(String[] args) {
    greetManyTimes("Anthony", 3);
    System.out.println("and");
    greetManyTimes("Martin", 2);
}
```

Output:

```
Hi Anthony, greetings from the world of methods!
Hi Anthony, greetings from the world of methods!
Hi Anthony, greetings from the world of methods!
and
Hi Martin, greetings from the world of methods!
Hi Martin, greetings from the world of methods!
```

Exercise 39: Printing

Exercise 39.1: Printing stars

Create a method `printStars` that prints the given amount of stars and a line break.

Create the method in the following body:

```
private static void printStars(int amount) {
    // you can print one star with the command
    // System.out.print("*");
    // call this command amount times
}

public static void main(String[] args) {
    printStars(5);
    printStars(3);
    printStars(9);
}
```

The program output:

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

Note: you can return exercises that contain many parts to the exercise robot even though you are not finished with all parts. In that case, the robot complains about tests in the unfinished parts of the exercise, but gives you points for all tests that pass.

Exercise 39.2: Printing a square

Create a method `printSquare(int sideSize)` that prints a square using our previous method printStars. The method call `printSquare(4)`, for example, prints the following:

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

Note: in order to complete the exercise it is not enough that the outprint looks good. Inside the printSquare method the printing must be done using the printStars method.

When you are in the middle of making your program, you should verify the correctness of your methods by writing some test code into your main method.

Exercise 39.3: Printing a rectangle

Create a method `printRectangle(int width, int height)` that prints a rectangle using the printStars method. The call `printRectangle(17,3)`, for example, has the following output:

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

Exercise 39.4: Printing a left-aligned triangle

Create the method `printTriangle(int size)` that prints a triangle using the printStars method. The method call `printTriangle(4)`, for example, has the following output:

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

Exercise 40: Printing Like A Boss

Exercise 40.1: Printing stars and whitespaces

Create a method `printWhitespaces(int size)` that prints the given amount of whitespaces. Them method should not print a line break.

Reimplement or copy the method `printStars(int size)` from the previous exercise.

Exercise 40.2: Printing a right-aligned triangle

Create the method `printTriangle(int size)` that prints a triangle using the mehtods `printWhitespaces` and `printStars`. **Note:** do not print anything in the method itself, just call the helper methods to do the actual printing.

For example, the method call `printTriangle(4)` has the following output:

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

Exercise 40.3: Printing a Christmas tree

Create the method `xmasTree(int height)` that prints a Christmas tree using the mehtods `printWhitespaces` and `printStars`. A Christmas tree consists of a triangle of given height and a stand. The stand is two stars tall and three stars wide and it is located in the center of the bottom of the triangle. **Note:** do not print anything in the method itself, just call the helper methods to do the actual printing.

The method call `xmasTree(4)`, for example, has the following output:

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

The method call `xmasTree(10)` has the following output:

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

Second note: You don't need to worry about heights below 3!

Exercise 41: Guessing a number game

In this exercise the following game is created:

```
Guess a number: 73
The number is lesser, guesses made: 1
Guess a number: 22
The number is greater, guesses made: 2
Guess a number: 51
The number is greater, guesses made: 3
Guess a number: 62
The number is greater, guesses made: 4
Guess a number: 68
The number is greater, guesses made: 5
Guess a number: 71
The number is lesser, guesses made: 6
Guess a number: 70
Congratulations, your guess is correct!
```

Exercise 41.1: Guessing a number

The program that comes with the exercise contains a command called `drawNumber`. It draws a number, which is in the range 0 to 100 (both 0 and 100 are possible). Create a program that draws a number. Then the user has the chance to guess once, what the number is. The program should to print "The number is lesser", "The number is greater" or "Congratulations, your guess is correct!" depending on the number the user typed.

```
Guess a number: 12
The number is greater
```

```
Guess a number: 66
The number is lesser
```

```
Guess a number: 42
Congratulations, your guess is correct!
```

Exercise 41.2: Repeated guessing

Develop your program by adding the following functionality: the guessing should be made repeatedly until the user types the right number. Note that you need to draw the number by using the `drawNumber` command *before the repetition*. Why? What happens if you draw the number inside the repetition?

In the example below, the command call `drawNumber` returned the value 83.

```
Guess a number: 55
The number is greater
Guess a number: 85
The number is lesser
Guess a number: 77
The number is greater
Guess a number: 81
The number is greater
Guess a number: 83
Congratulations, your guess is correct!
```

Exercise 41.3: Counting the guesses

Develop your program by adding the following functionality: the program needs to include a variable of type int, which is used to count the guesses the user has made. The program should always print the number of guesses along with the answer.

```
Guess a number: 55
The number is greater, guesses made: 1
Guess a number: 85
```

```
The number is lesser, guesses made: 2
Guess a number: 77
The number is greater, guesses made: 3
Guess a number: 81
The number is greater, guesses made: 4
Guess a number: 83
Congratulations, your guess is correct!
```

Exercise 42: A text-based user interface for the Hangman game

Your friend has programmed a Hangman game for you, but the game lacks the user inferface. The Hangman has the following methods:

- *hangman.gameOn()*
  Shows if the game is on
- *hangman.printStatus()*
  Prints the game status. Shows how many guesses have been made and the letters that have not been used yet.
- *hangman.printWord()*
  Prints the word the user tries to guess. The letters that have not been guessed yet are hidden as question marks, like "v?ri?ble".
- *hangman.printMan()*
  Prints the Hangman.
- *hangman.guess(String letter)*
  Guesses the letter that is given as a parameter.

You will get a program body from the exercise robot. It already contains some functionalities:

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

        System.out.println("************");
        System.out.println("* Hangman *");
        System.out.println("************");
        System.out.println("");
        printMenu();
        System.out.println("");

        // ADD YOUR IMPLEMENTATION HERE

    System.out.println("Thank you for playing!");
```

In addition to the program body, you will get the method called `printMenu`:

```
    public static void printMenu() {
        System.out.println(" * menu *");
        System.out.println("quit   - quits the game");
        System.out.println("status  - prints the game status");
        System.out.println("a single letter uses the letter as a guess");
```

```
        System.out.println("an empty line prints this menu");
    }
```

The exercise is completed in small steps.

Exercise 42.1: Loops and ending loops

Create a loop in the program that works as a base for the rest of the user interface. Ask the user to submit the command inside the loop. If the command is "quit", break the loop.

Use the command `hangman.gameOn` as the condition for the while structure. The loop should look like:

```
while (hangman.gameOn()) {
    String command = reader.nextLine();
    // ...
}
```

In the next set (week) of exercises, we will find out what this peculiar-looking condition for ending the loop is about.

This far the program should produce the following output:

```
************
* Hangman *
************

 * menu *
quit   - quits the game
status - prints the game status
a single letter uses the letter as a guess
an empty line prints this menu

Type a command:
do not quit

Type a command:
quit
Thank you for playing!
```

Exercise 42.2: Printing the status

If the user gives the command "status", print the status using the method `hangman.printStatus()`.

```
************
* Hangman *
************

* menu *
quit   - quits the game
status - prints the game status
```

```
a single letter uses the letter as a guess
an empty line prints this menu

Type a command:
status
You have not made any guesses yet.
Unused letters: abcdefghijklmnopqrstuvwxyz

Type a command:
quit
Thank you for playing!
```

Exercise 42.3: Making a guess

If the user types in a single letter as a command, use it to make a guess. Guessing a letter occurs in the method `hangman.guess(command)`. The guessing command has its own printing functionality, which it uses to print more information about the guess.

Hint: finding out if the command is a single letter is done as follows:

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

if(command.length() == 1) {  // command has only one letter, so it must be a guess
    hangman.guess(command);
}
```

```
...
Type a command:
a
The letter a is not in the word.

Type a command:
b
The letter b is not in the word.

Type a command:
c
The letter c was found in the word!

Type a command:
quit
Thank you for playing!
```

Exercise 42.4: Printing out the menu

If the user types an empty string of characters, meaning a string that has zero length, you need to call the method `printMenu`. Note that the method `printMenu` is not in the Hangman game but in your own program.

Note: checking if the string is empty is done as follows:

```
String winnie = "the pooh";
if(winnie.isEmpty()) {
    System.out.println("String was empty");
} else {
```

```
    System.out.println("I found something!");
}
```

Exercise 42.5: Printing the man and the word

If the user has not typed the command `quit`, you should call the Hangman game commands `hangman.printMan()` and `hangman.printWord()` at the end of the loop.

```
...
Type a command:
a
The letter a is not in the word.
 ____
 |
 |
 |
 |
/|\
Word to be guessed: ????

Type a command:
m
The letter m was found in the word!
 ____
 |
 |
 |
 |
/|\
Word to be guessed: m???

Type a command:
quit
Thank you for playing!
```

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

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

TIETOJENKÄSITTELYTIETEEN LAITOS
INSTITUTIONEN FÖR DATAVETENSKAP
DEPARTMENT OF COMPUTER SCIENCE