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

14. More about methods

14.1 Methods and visibility of variables

Let us try to change from within a method the value of a variable located in the main program.

```
// main program
public static void main(String[] args) {
    int number = 1;
    addThree();
}

// method
public static void addThree() {
    number = number + 3;
}
```

Unfortunately this program will not work, because the method cannot "see" the variable `number` located in the main program.

This holds generally. Variables defined in the main program are not visible for other methods. Also, the other way is similar: variables defined in methods are not visible for other methods or the main program. The only way to give information to a method from the outside is to use parameters.

```
// main program
public static void main(String[] args) {
    int number = 1;
    System.out.println("Main program variable number holds the value: " + number);
    addThree(number);
    System.out.println("Main program variable number holds the value: " + number);
}

// method
public static void addThree(int number) {
    System.out.println("Method parameter number holds the value: " + number);
    number = number + 3;
    System.out.println("Method parameter number holds the value: " + number);
}
```

In the program above the method addThree has a parameter called `number`. This parameter is *copied* (duplicated) for the method to use. When the program above is executed we see the following output:

```
Main program variable number holds the value: 1
Method parameter number holds the value: 1
```

```
Method parameter number holds the value: 4
Main program variable number holds the value:  1
```

The number we gave as a parameter to the method was *copied* for the method to use. If we would like the main program to be able to use the new value generated by the method, the method needs to return that value.

14.2 Return values

A method can return a value. In the examples above, methods have not been returning anything. This is expressed by writing *void* in the first line of the method, just before it's name.

```
public static void addThree() {
  ...
```

When defining a method that returns a value, we also have to define the type of the return value. We can define the type of the return value by writing it just before the name of the method. Next, we have a method that always returns the number 10 (type `int`). Returning a value is accomplished with the command return:

```
public static int alwaysReturnTen() {
    return 10;
}
```

If we want to use the returned value later, we have to catch the return value and store it into a variable:

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

    System.out.println( "method returned the number " + number );
}
```

The return value of the method is assigned to a variable of type `int` just like any other integer. The return value can also be a part of a sentence:

```
double number = 4 * alwaysReturnTen() + (alwaysReturnTen() / 2) - 8;

System.out.println( "calculation total " + number );
```

Every variable type we have seen this far can be used as a return value:

```
public static void methodThatReturnsNothing() {
  // method body
}

public static int methodThatReturnsInteger() {
  // method body, needs a return statement
}

public static String methodThatReturnsText() {
```

```
  // method body, needs a return statement
}

public static double methodThatReturnsFloatingpoint() {
  // method body, needs a return statement
}
```

If the method is defined to have a return value, it also has to return a value. The following method is incorrect:

```
public static String wrongMethod() {
    System.out.println("I tell you that I will return a String but I do not!");
}
```

In the following example, we define a method for calculating a sum. Then, we use the method to calculate 2 + 7. The return value (returned after the method call) is assigned to a variable called `sumNumbers`.

```
public static int sum(int first, int second) {
    return first + second;
}
```

Method call:

```
int sumNumbers = sum(2, 7);
// sumNumbers now holds the value 9
```

Let us expand the example program so that the user types the numbers.

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

    System.out.print("Type the first number: ");
    int first = Integer.parseInt( reader.nextLine() );

    System.out.print("Type the second number: ");
    int second = Integer.parseInt( reader.nextLine() );

    System.out.print("Total: " + sum(first,second) );
}

public static int sum(int first, int second) {
    return first + second;
}
```

As we can see, the return value of the method does not always need to be assigned to a variable. It can also act as a part of the printing command just like any other integer value.

In the next example, we call the method sum using integers that we get as return values from the method `sum`.

```
int first = 3;
int second = 2;

sum(sum(1, 2), sum(first, second));
// 1) the inner methods are executed:
```

```
//    sum(1, 2) = 3   and sum(first, second) = 5
// 2) the outer method is executed:
//    sum(3, 5) = 8
```

14.3 The method's own variables

The following method calculates the average of the numbers the method gets as parameters. The method uses helper variables `sum` and `average`. The method's own variables can be introduced just like any other variables.

```
public static double average(int number1, int number2, int number3) {

    int sum = number1 + number2 + number3;
    double average = sum / 3.0;

    return average;
}
```

Exercise 43: Sum of numbers

Create the method `sum` that calculates the sum of numbers the method receives as parameters.

Place the method in the following program body:

```
public static int sum(int number1, int number2, int number3, int number4) {
    // write program code here
    // remember that the method needs a return in the end
}

public static void main(String[] args) {
    int answer = sum(4, 3, 6, 1);
    System.out.println("sum: " + answer);
}
```

Example output:

```
sum: 14
```

Note: if an exercise involves a method *returning* something, it means that the return type needs to be defined for the method, and that the method needs to return a value of that type using the `return` command. In this case, the method does not print (or use the command `System.out.println(..)`), the method caller handles printing, here, the main program..

Exercise 44: Least

Create the method `least`, which returns the least of the numbers given as parameters. If the parameters are equal, you can decide which one is returned.

```
public static int least(int number1, int number2) {
    // write program code here
    // do not print anything inside the method

    // method needs a return in the end
}

public static void main(String[] args) {
    int answer =  least(2, 7);
    System.out.println("Least: " + answer);
}
```

Example output:

```
Least: 2
```

Exercise 45: Greatest

Create the method `greatest`, which gets three integers as parameters and then returns the greatest of them. If there are several parameters that are equally great, you can decide which one is returned. Printing should be done in the main program.

```
public static int greatest(int number1, int number2, int number3) {
    // write your code here
}

public static void main(String[] args) {
    int answer =  greatest(2, 7, 3);
    System.out.println("Greatest: " + answer);
}
```

Example output:

```
Greatest: 7
```

Exercise 46: Average of given numbers

Create the method `average`, which calculates the average of the numbers it gets as parameters. Inside the method you should use the method `sum` as a helper!

Place the method in the following program body:

```
public static double average(int number1, int number2, int number3, int number4) {
    // write your code here
}

public static void main(String[] args) {
    double answer = average(4, 3, 6, 1);
    System.out.println("average: " + answer);
}
```

Program output:

```
average: 3.5
```

Make sure you remember how you can transform a whole number (`int`) into a decimal number (`double`)!

15. Strings of characters

In this section, we take a closer look at strings of characters in Java, which are called `Strings`. We have already used variables of `String` type when printing, and learned how to compare Strings. Comparing two strings is performed by *calling* the `equals()` method of the string.

```
String animal = "Dog";

if( animal.equals("Dog") ) {
    System.out.println(animal + " says bow-wow");
} else if ( animal.equals("Cat") ) {
    System.out.println(cat + " says meow meow");
}
```

It is possible to ask the string how many characters long it is by writing `.length()` after it's name. In other words, we are calling its length() method.

```
String banana = "banana";
String cucumber = "cucumber";
String together = banana + cucumber;

System.out.println("The length of banana is " + banana.length());
System.out.println("The length of  cucumber is " + cucumber.length());
System.out.println("The word " + together + " length is " + together.length());
```

In the above code, the method `length()` is called for three different strings. The call `banana.length()` calls only the method that gives the length of the string `banana`, while `cucumber.length()` calls the method that gives the length of the string `cucumber` etc. The left part before the dot says whose method is called.

Java has a special data type, called `char`, to be used for characters. A `char` variable can store only one character. A string variable can return a character from a specific location in itself with the method `charAt()` that uses the index of the location as a parameter. Note that counting the index of the character starts from zero!

```
String word = "Supercalifragilisticexpialidocious";

char character = word.charAt(3);
System.out.println("The 4th character of the word is " + character); //prints "e"
```

The characters in a string are numbered (indexed) starting from 0. This means that we can reach the last character in a string with number (or index) "the length of the word minus one", or `word.charAt(word.length()-1)`. The following example will make the program crash, because we are trying to get a character from an index that does not exist.

```
char character = word.charAt(word.length());
```

A tip for using NetBeans

- You can find all the NetBeans tips here
- **Renaming**

  Variables, methods and classes (we will learn about these in the next set) need to have descriptive names. Often, a name is not that describing and needs to be changed. In NetBeans, it is really easy to rename things. Just select and "paint" the name you want to change with the mouse. Then press ctrl and r simultaneously, and write the new name.

Exercise 47: The length of a name

Create a program that asks for the user's name and says how many characters the name contains.

```
Type your name: Paul
Number of characters: 4
```

```
Type your name: Catherine
Number of characters: 9
```

Note!Your program should be structured so that you put the calculating of the name length in it's own method: `public static int calculateCharacters(String text)`. The tests will be testing both the method `calculateCharacters` and the program overall.

Exercise 48: First character

Create a program that asks for the user's name and gives the first character.

```
Type your name: Paul
First character: P
```

```
Type your name: Catherine
First character: C
```

Note! Your program should be structured so that you put the search for the first character in its own method: `public static char firstCharacter(String text)`. The tests will be testing both the method `firstCharacter` and the program overall

Exercise 49: Last character

Create a program that asks for the user's name and gives the last character.

```
Type your name: Paul
Last character: l
```

```
Type your name: Catherine
Last character: e
```

Note! Your program should be structured so that you put the search for the last character in its own method: `public static char lastCharacter(String text)`. The tests will be testing both the method `lastCharacter` and the program overall.

Exercise 50: Separating first characters

Create a program that asks for the user's name and gives its first, second and third characters separately. If the name length is less than three, the program prints nothing. You do not need to create methods in this exercise.

```
Type your name: Paul
1. character: P
2. character: a
3. character: u
```

```
Type your name: me
```

Note: watch closely at the output in this and the following exercise. The print needs to contain a space after the dot and the colon!

Exercise 51: Separating characters

Create a program that asks for the user's name and gives its characters separately. You do not need to create methods in this exercise.

```
Type your name: Paul
1. character: P
2. character: a
3. character: u
4. character: l
```

Hint: using a `while` loop helps in this exercise!

```
Type your name: Catherine
1. character: C
2. character: a
3. character: t
4. character: h
5. character: e
6. character: r
7. character: i
8. character: n
9. character: e
```

Exercise 52: Reversing a name

Create a program that asks for the user's name and prints it in reverse order. You do not need to create a separate method for this.

```
Type your name: Paul
In reverse order: luaP
```

```
Type your name: Catherine
In reverse order: enirehtaC
```

Hint: You can print one character using the command `System.out.print()`

15.1 Other methods for strings

We often want to read only a specific part of a string. A method in the String class called `substring` makes this possible. It can be used in two ways:

```
String word = "Supercalifragilisticexpialidocious";
System.out.println(word.substring(14)); //prints "listicexpialidocious"
System.out.println(word.substring(9,20)); //prints "fragilistic"
```

We can store the return value in a variable, because the return value of the `substring` method is of type `String`.

```
String book = "Mary Poppins";
String endpart = book.substring(5);
System.out.println("Harry " + endpart); // prints "Harry Poppins"
```

Methods in the String class also make it possible to search for a specific word in text. For example, the word "or" can be found in the word "Horse". A method called `indexOf()` searches for the word given as a parameter in a string. If the word is found, it returns the starting index (location), remember that the numbering starts from 0 of the word. If the word is not found, the method returns the value -1.

```
String word = "aesthetically";

int index = word.indexOf("tic"); // index value will be 6
System.out.println(word.substring(index)); //prints "tically"

index = word.indexOf("ally"); //index value will be 9
System.out.println(word.substring(index)); //prints "ally"

index = word.indexOf("book"); // string "aesthetically" does not include "book"
System.out.println(index); //prints -1
System.out.println(word.substring(index)); //error!
```

Exercise 53: First part

Create a program that prints the first part of a word. The program asks the user for the word and the length of the first part. Use the `substring` method in your program.

```
Type a word: example
Length of the first part: 4
Result: exam
```

```
Type a word: example
Length of the first part: 6
Result: exampl
```

Exercise 54: The end part

Create a program that prints the end part of a word. The program asks the user for the word and the length of the end part. Use the `substring` method in your program.

```
Type a word: example
Length of the end part: 4
Result: mple
```

```
Type a word: example
Length of the end part: 6
Result: xample
```

Exercise 55: A word inside a word

Create a program that asks the user for two words. Then the program tells if the second word is included in the first word. Use String method `indexOf` in your program.

```
Type the first word: glitter
Type the second word: litter
The word 'litter' is found in the word 'glitter'.
```

```
Type the first word: glitter
Type the second word: clean
The word 'clean' is not found in the word 'glitter'.
```

Note: Make your program outputs (prints) match exactly the example above!

Exercise 56: Reversing text

Create the method `reverse` that puts the given string in reversed order. Use the following program body for the method:

```
public static String reverse(String text) {
    // write your code here
}

public static void main(String[] args) {
    System.out.print("Type in your text: ");
    String text = reader.nextLine();
    System.out.println("In reverse order: " + reverse(text));
}
```

Hint: you probably need to build the reversed string character by character in your method. You can use a String-type variable as a helper during the building process. In the beginning, the helper variable should have an empty string of characters as a value. After this, new characters are added to the string one by one.

```
String help = "";

// ...
// adding a character to the help variable
help = help + character;
```

Program output:

```
Type a text: example
elpmaxe
```

16. Object

Strings and integers have some differences. Integers are "just values", they can be used in calculations and they can be printed on the screen:

```
int x = 1;
int y = 2;

y = 3*x;

System.out.println( "value of y now: " + y );
```

Strings are a bit "cleverer" and for example know how long they are:

```
String word1 = "Programming";
String word2 = "Java";

System.out.println( "String "+ word1 +" length: " + word1.length() );

System.out.println( "String "+ word2 +" length: " + word2.length() );
```

Program output:

```
String Programming length: 11
String Java length: 4
```

We can determine the length by calling the String method `length()`. Strings have other methods as well. Integers (or whole numbers, variables of type `int`) have no methods at all. They do not

"know" anything.

Strings are *objects*, or "something that has methods and a value". Later we will see many other objects as well.

As we can see in the previous example, an object's methods are called by adding a dot and a method call after the name of the object:

```
word1.length()    // String object's name is word1 and its method length() is called
word2.length()    // String object's name is word2 and its method length() is called
```

The method call is made explicitly to the object. In the above example, we have two objects and first we call the `length()` method of the String object `word1` and then do the same for the object `word2`.

Our old friend `reader` is also an object:

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

Even though readers and strings are both objects, they are not very similar. For example, readers (Scanners) have the `nextLine()` method, but Strings do not. In the Java programming language, objects must be "born", in other words created with the `new` command. Strings are objects that make an exception to this rule! -- There are two ways to create a String object:

```
String banana = new String("Banana");
String carrot = "carrot";
```

Both of the commands above create a new String object. Using the `new` command when creating a String objects is uncommon.

The object's "type" is called a *class*. The class of a string of characters is called `String` and the class of readers is called `Scanner`. Later we learn much more about classes and objects.

17. ArrayList or an "object container"

Often during programming, we would like to keep many different strings in memory. A very bad idea would be to define a variable for each of them: :

```
String word1;
String word2;
String word3;
// ...
String word10;
```

This would be such a good-for-nothing solution that it does not almost need an explanation -- think of this approach for a word count of 100 or 1000!

Just like other modern programming languages, Java gives us different tools to store many objects neatly in our programs. Now, we take a closer look at *ArrayList*, which is probably the

most used object container in Java.

The following lines of code make use of an ArrayList that holds specifically objects of type String. A couple of strings are stored into the list.

```
import java.util.ArrayList;

public class ListProgram {

    public static void main(String[] args) {
        ArrayList<String> wordList = new ArrayList<String>();

        wordList.add("First");
        wordList.add("Second");
    }
}
```

In the above main program method, the first row creates a new ArrayList called `wordList`, which can be used as a container for String variables. The type of the ArrayList is `ArrayList<String>`, which means that the ArrayList is meant for storing Strings. The list is created using the command `new ArrayList<String>();`.

Note: to make the ArrayList work, we must first write an import statement at the beginning of the program either `import java.util.ArrayList;` or `import java.util.*;`

When the list is created, two strings are added by calling the list method `add`. The list will not run out of space, so theoretically the list can contain any amount of Strings (as long as they fit in the computer's memory).

Internally an ArrayList is -- as its name suggests -- a list. The added strings automatically go to the end of the ArrayList.

17.1 Methods of ArrayLists

ArrayList provides us with many useful methods:

```
public static void main(String[] args) {
    ArrayList<String> teachers = new ArrayList<String>();

    teachers.add("Anthony");
    teachers.add("Barto");
    teachers.add("Paul");
    teachers.add("John");
    teachers.add("Martin");
    teachers.add("Matt");

    System.out.println("the number of teachers " + teachers.size() );

    System.out.println("first teacher on the list " + teachers.get(0));
    System.out.println("third teacher on the list " + teachers.get(2));

    teachers.remove("Barto");

    if (teachers.contains("Barto")) {
        System.out.println("Barto is on the teachers list");
    } else {
        System.out.println("Barto is not on the teachers list");
```

```
    }
}
```

First a list of strings is created and then 6 names added to it. `size` tells us the amount of strings in the list. **Note:** when the method is called, the call should have the following format: `teachers.size()`. First comes the name of the object, then follows a dot followed by the name of the method.

The strings will be in the list in the order in which they were added to it. By calling the method `get(i)`, we get the value from the index (location) i in the list. The indexing of items in the list starts from 0. This means that the first added string is located at index 0, the second at index 1, and so on.

We can remove strings from lists through the method `remove`. The method can be used in two ways. First, `remove("characters")` removes the string given as a parameter. Second, `remove(3)` removes the 4th String from the list.

At the end of the example, the method `contains` is called. This method is used for asking the list if it contains the string given as a parameter. If it does, the method returns the value `true`.

Program output:

```
the number of teachers 6
first teacher on the list Anthony
third teacher on the list Paul
Barto is not on the teachers list
```

Note! The methods `remove` and `contains` assume that the objects stored in the ArrayList do have an `equals` method. We will get back to this later in the course.

17.2 Going through an ArrayList

In the following example 4 names are added to the list. Then the whole list is printed:

```
public static void main(String[] args) {
    ArrayList<String> teachers = new ArrayList<String>();

    teachers.add("Anthony");
    teachers.add("Paul");
    teachers.add("John");
    teachers.add("Martin");

    System.out.println( teachers.get(0) );
    System.out.println( teachers.get(1) );
    System.out.println( teachers.get(2) );
    System.out.println( teachers.get(3) );
}
```

This solution works, but is really clumsy. What if there were more items in the list? Or less? What if we would not know how many items there are?

First, we create a temporary version:

```
public static void main(String[] args) {
    ArrayList<String> teachers = new ArrayList<String>();

    teachers.add("Anthony");
    teachers.add("Paul");
    teachers.add("John");
    teachers.add("Martin");
    teachers.add("Matt");

    int place = 0;
    System.out.println( teachers.get(place) );
    place++;
    System.out.println( teachers.get(place) );  // place = 1
    place++;
    System.out.println( teachers.get(place) );  // place = 2
    place++;
    System.out.println( teachers.get(place) );  // place = 3
}
```

Using our old friend the `while` command, we can increment the variable `place` by one until it gets too big:

```
public static void main(String[] args) {
    ArrayList<String> teachers = new ArrayList<String>();

    teachers.add("Anthony");
    teachers.add("Paul");
    teachers.add("John");
    teachers.add("Martin");
    teachers.add("Matt");

    int place = 0;
    while ( place < teachers.size() )  // remember why place <= teachers.size() doesn't work?
        System.out.println( teachers.get(place) );
        place++;
    }
}
```

Now, printing works regardless of the amount of items in the list.

Using a while loop, and "self indexing" the locations in the list, is usually not the best way to go through a list. A much more recommended way is to use the for-each loop described below.

17.3 for-each

Even though the command is usually referred to as for-each, the real name of the command is only `for`. There are two versions of for, the traditional and the "for-each". The latter is used now.

Going through items in an ArrayList with for-each is easy:

```
public static void main(String[] args) {
    ArrayList<String> teachers = new ArrayList<String>();

    teachers.add("Anthony");
    teachers.add("Paul");
    teachers.add("John");
    teachers.add("Martin");
    teachers.add("Matt");
```

```
    for (String teacher : teachers) {
        System.out.println( teacher );
    }
}
```

As we can see, the indexes of the list can be ignored if we go through the content of the list "automatically".

In the code block of the for command (inside { }) a variable `teacher` is used. It is defined in the for row, on the left side of the colon. What happens is that every item in the list `teachers` becomes the value of the variable `teacher`, one by one. It means that when for is entered, the first `teacher` is *Anthony*, the second execution of for makes the `teacher` become *Paul* etc.

Even though the `for` command might seem a bit strange at first, you should definitely get used to use it!

Exercise 57: Words

Create a program that asks the user to input words until the user types in an empty String. Then the program prints the words the user gave. *Try the for repetition sentence here*. Use an `ArrayList` structure in your program. ArrayList is defined like this:

```
ArrayList<String> words = new ArrayList<String>();
```

```
Type a word: Mozart
Type a word: Schubert
Type a word: Bach
Type a word: Sibelius
Type a word: Liszt
Type a word:
You typed the following words:
Mozart
Schubert
Bach
Sibelius
Liszt
```

Note: an empty String can be detected this way:

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

if ( word.isEmpty() ) {  // could also be: word.equals("")
   // word was empty, meaning that the user only pressed enter
}
```

Exercise 58: Recurring word

Create a program that asks the user to input words until the user gives the same word twice. Use an `ArrayList` structure in your program. ArrayList is defined like this:

```
ArrayList<String> words = new ArrayList<String>();
```

```
Type a word: carrot
Type a word: celery
Type a word: turnip
Type a word: rutabaga
Type a word: celery
You gave the word celery twice
```

Hint: Remember that ArrayList has the method `.contains()`

17.4 Ordering, reversing and shuffling a list

Items in an ArrayList are easy to order by size. Ordering by size means an alphabetic order when the list items are of type String. Ordering is done as follows:

```
public static void main(String[] args) {
    ArrayList<String> teachers = new ArrayList<String>();

    // ...

    Collections.sort(teachers);

    for (String teacher : teachers) {
        System.out.println( teacher );
    }
}
```

Output:

```
Anthony
Barto
John
Martin
Matt
Paul
```

We give the list as a parameter for the method `Collections.sort`. The import line `import java.util.Collections;` or `import java.util.*;` needs to be at the beginning of the program in order to get tools of Collections working in our program.

Collections also includes other useful methods:

- `shuffle` shuffles the list items, can be useful for example in games
- `reverse` reverses the order of list items

Exercise 59: Words in reverse order

Create a program that asks the user to input words, until the user gives an empty string. Then the program prints the words the user gave in reversed order, the last word is printed first etc.

```
Type a word: Mozart
Type a word: Schubert
Type a word: Bach
Type a word: Sibelius
Type a word: Liszt
Type a word:
You typed the following words:
Liszt
Sibelius
Bach
Schubert
Mozart
```

Exercise 60: Words in alphabetical order

Create a similar program as the previous one, but in which the words are printed in alphabetical order.

```
Type a word: Mozart
Type a word: Schubert
Type a word: Bach
Type a word: Sibelius
Type a word: Liszt
Type a word:
You typed the following words:
Bach
Liszt
Mozart
Schubert
Sibelius
```

17.5 ArrayList as a parameter for a method

ArrayList can be given to a method as a parameter:

```
public static void print(ArrayList<String> list) {
    for (String word : list) {
        System.out.println( word );
    }
}

public static void main(String[] args) {
    ArrayList<String> list = new ArrayList<String>();
    list.add("Java");
    list.add("Python");
    list.add("Ruby");
    list.add("C++");

    print(list);
}
```

The type of the parameter is defined as an ArrayList of String variables the same way a String ArrayList is defined:

Note that the name of the parameter can be anything:

```
public static void print(ArrayList<String> printed) {
    for (String word : printed) {
        System.out.println( word );
    }
}

public static void main(String[] args) {
    ArrayList<String> programmingLanguages = new ArrayList<String>();
    programmingLanguages.add("Java");
    programmingLanguages.add("Python");
    programmingLanguages.add("Ruby");
    programmingLanguages.add("C++");

    ArrayList<String> countries = new ArrayList<String>();
    countries.add("Finland");
    countries.add("Sweden");
    countries.add("Norway");

    print(programmingLanguages);    // method is given the list programmingLanguages
as a parameter

   print(countries);                //  method is given the list countries as a parameter
}
```

The program now includes two lists, *programmingLanguages* and *countries*. First the printing method is given the list *programmingLanguages*. The method `print` internally refers to the list given as a parameter with the name *printed*! Next, the printing method is given the list *countries*. Now, the method uses again the name *printed* referring to the parameter list.

Exercise 61: Amount of items in a list

Create the method `public static int countItems(ArrayList<String> list)` that returns the number of the items in the list. Your method should not print anything. Use a `return` statement to return the number as shown in the following example:

```
ArrayList<String> list = new ArrayList<String>();
list.add("Hallo");
list.add("Ciao");
list.add("Hello");
System.out.println("There are this many items in the list:");
System.out.println(countItems(list));
```

```
There are this many items in the list:
3
```

Inside the method, it is possible to influence the items in the parameter list. In the following example, the method `removeFirst` --as the name suggests-- removes the first string from the list. What would happen if the list was empty?

```
public static void print(ArrayList<String> printed) {
    for (String word : printed) {
        System.out.println( word );
    }
}

public static void removeFirst(ArrayList<String> list) {
    list.remove(0);  // removes the first item (indexed 0)
}

public static void main(String[] args) {
    ArrayList<String> programmingLanguages = new ArrayList<String>();
    programmingLanguages.add("Pascal");
    programmingLanguages.add("Java");
    programmingLanguages.add("Python");
    programmingLanguages.add("Ruby");
    programmingLanguages.add("C++");

    print(programmingLanguages);

    removeFirst(programmingLanguages);

    System.out.println();  // prints an empty line

   print(programmingLanguages);
}
```

Output:

```
Pascal
Java
Python
Ruby
C++

Java
Python
Ruby
C++
```

Similarly a method could, for example, add more strings to the list it received as a parameter.

Exercise 62: Remove last

Create the method `public static void removeLast(ArrayList<String> list)`, which removes the last item from the list. Example code:

```
ArrayList<String> brothers = new ArrayList<String>();
brothers.add("Dick");
brothers.add("Henry");
brothers.add("Michael");
brothers.add("Bob");

System.out.println("brothers:");
System.out.println(brothers);

// sorting brothers
brothers.sort();

// removing the last item
removeLast(brothers);

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

Example output:

```
brothers:
[Dick, Henry, Michael, Bob]
[Bob, Dick, Henry]
```

As we notice from the example above, an ArrayList can be printed as it is. The print formatting is not usually what is sought after, so we are forced to handle the printing ourself. For example, with the help of the `for` command.

17.6 Numbers in an ArrayList

ArrayLists can be used to store any type of values. If the stored variables are of integer type, there are a couple of details to remember. An integer ArrayList is defined like this: `ArrayList<Integer>`, instead of writing `int` you must write `Integer`.

The method `remove` does not work like expected when the list consists of `int` numbers::

```
public static void main(String[] args) {
    ArrayList<Integer> numbers = new ArrayList<Integer>();

    numbers.add(4);
    numbers.add(8);
```

```
    // tries to remove the number from the index 4, does not work as expected!
    numbers.remove(4);

    // this removes the number 4 from the list
    numbers.remove(Integer.valueOf(4));
}
```

`numbers.remove(4)` tries to remove the item in the index 4 from the list. There are only 2 items in the list, so the command generates an error. We must use a slightly more complicated command if the number 4 needs to be removed: `numbers.remove( Integer.valueOf(4) );`

ArrayLists can also be used to store `double`s (decimal numbers) and characters (`char` variables). The lists can be defined as follows:

```
ArrayList<Double> doubles = new ArrayList<Double>();
ArrayList<Character> characters = new ArrayList<Character>();
```

Exercise 63: Sum of the numbers

Create the method `sum`, which receives a list of numbers (`ArrayList<Integer>`) as a parameter and then calculates the sum of the items in that list.

Create the method using the following program body:

```
public static int sum(ArrayList<Integer> list) {
    // write your code here
}

public static void main(String[] args) {
    ArrayList<Integer> list = new ArrayList<Integer>();
    list.add(3);
    list.add(2);
    list.add(7);
    list.add(2);

    System.out.println("The sum: " + sum(list));

    list.add(10);

    System.out.println("the sum: " + sum(list));
}
```

Program output:

```
The sum: 14
The sum: 24
```

Exercise 64: Average of numbers

Create the method `average`, which receives a list of numbers (`ArrayList<Integer>`) as a parameter and then calculates the average of the items in that list.

Note: the method should use the method `sum` from the previous exercise to calculate the sum of the parameters.

Create the method using the following program body:

```
public static double average(ArrayList<Integer> list) {
    // write your code here
}

public static void main(String[] args) {
    ArrayList<Integer> list = new ArrayList<Integer>();
    list.add(3);
    list.add(2);
    list.add(7);
    list.add(2);

    System.out.println("The average is: " + average(list));
}
```

Program output:

```
The average is: 3.5
```

17.7 ArrayList as return value of a method

ArrayList can also be returned from a method as a return value. In the next example, a method creates an ArrayList, adds three integers into the list and then returns the list. Pay attention to how the main program assigns the list returned by the method as a value into a variable that has the same type as the return value:

```
public class Main {

    public static ArrayList<Integer> addNumbersToList(int num1, int num2, int num3){
        ArrayList<Integer> list = new ArrayList<Integer>();

        list.add(num1);
        list.add(num2);
        list.add(num3);

        return list;
    }

    public static void main(String[] args) {
        ArrayList<Integer> numbers = addNumbersToList(3, 5, 2);

        for (int number : numbers) {
```

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

Exercise 65: The lengths of the Strings

Create the method `lengths` that gets a list of String variables as a parameter and returns an ArrayList that contains the lengths of the Strings in the same order as the original list.

```
public static ArrayList<Integer> lengths(ArrayList<String> list) {
    // write your code here
}

public static void main(String[] args) {
    ArrayList<String> list = new ArrayList<String>();
    list.add("Hallo");
    list.add("Moi");
    list.add("Benvenuto!");
    list.add("badger badger badger badger");
    ArrayList<Integer> lengths = lengths(list);

    System.out.println("The lengths of the Strings: " + lengths);
}
```

Program output:

```
The lengths of the Strings: [5, 3, 10, 27]
```

Exercise 66: The Greatest

Create the method `greatest`, which receives a list of numbers (`ArrayList<Integer>`) as a parameter and then returns the greatest number in the list as a return value.

```
public static int greatest(ArrayList<Integer> list) {
    // write your code here
}

public static void main(String[] args) {
    ArrayList<Integer> list = new ArrayList<Integer>();
    list.add(3);
    list.add(2);
    list.add(7);
    list.add(2);

    System.out.println("The greatest number is: " + greatest(list));
}
```

Program output:

```
The greatest number is: 7
```

Exercise 67: Variance

Create the method `variance`, which receives a list of integers as a parameter and then returns the sample variance of that list. You can check how a sample variance is calculated in Wikipedia, under "Population variance and sample variance".

Note: the method should use the method `average` of exercise 64 to calculate the average of the parameters. The method should be called only once!

```
public static double variance(ArrayList<Integer> list) {
    // write your code here
}

public static void main(String[] args) {
    ArrayList<Integer> list = new ArrayList<Integer>();
    list.add(3);
    list.add(2);
    list.add(7);
    list.add(2);

    System.out.println("The variance is: " + variance(list));
}
```

Program output:

```
The variance is: 5.666667
```

(The average of the numbers is 3.5, so the sample variance is ((3 - 3.5)² + (2 - 3.5)² + (7 - 3.5)² + (2 - 3.5)²)/(4 - 1) ? 5,666667.)

Note while testing your program! Sample variance for a list that contains only one item is not defined! It causes a division by zero in the formula. Java considers the result of division by zero as a *Not a Number* (`NaN`).

18. Using truth values

A variable of type truth value (`boolean`) can only have two values: either *true* or *false*). Here is an example on how to use boolean variables:

```
int num1 = 1;
int num2 = 5;

boolean firstGreater = true;

if (num1 <= num2) {
    firstGreater = false;
}

if (firstGreater==true) {
    System.out.println("num1 is greater");
} else {
    System.out.println("num1 was not greater");
}
```

First, we assign the truth value variable `firstGreater` the value *true*. The first if sentence checks whether `num1` is less or equal to `num2`. If it is, we change the value of firstGreater to *false*. The later if sentence selects which string to print based on the truth value.

As a matter of fact, using a truth value in a conditional sentence is easier than the description in the previous example. We can write the second if sentence as follows:

```
if (firstGreater) {  // means the same as firstGreater==true
    System.out.println("num1 was greater");
} else {
    System.out.println("num1 was not greater");
}
```

If we want to check if the boolean variable holds the value true, we do not need to write `==true`, just writing the name of the variable is enough!

If we want to check if the boolean variable holds the value false, we can check that using the negation operation ! (exclamation mark):

```
if (!firstGreater) {  // means the same as firstGreater==false
    System.out.println("num1 was not greater");
} else {
    System.out.println("num1 was greater");
}
```

18.1 Methods that return a truth value

Truth values come in especially handy when we want to write methods that check for validity. Let us create a method that checks if the list it gets as a parameter includes only positive numbers (here 0 is considered positive). The method returns the information as a boolean (i.e. truth value).

```
public static boolean allPositive(ArrayList<Integer> numbers) {
    boolean noNegative = true;

    for (int number : numbers) {
        if (number < 0) {
            noNegative = false;
        }
```

```
    }

    // if one of the numbers on the list had a value that is below zero, noNegatives becomes false.

    return noNegative;
}
```

The method has a boolean helper variable called `noNegative`. First we assign the helper variable the value true. The method checks all numbers on the list one by one. If at least one number is less than 0, we assign the helper variable the value false. In the end the method returns the value of the helper variable. If no negative numbers were found, it has the value true, otherwise it has the value false.

The method is used as follows:

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

    ArrayList<Integer> numbers = new ArrayList<Integer>();
    numbers.add(3);
    numbers.add(1);
    numbers.add(-1);

    boolean result = allPositive(numbers);

    if (result) {  // means the same as result == true
        System.out.println("all numbers are positive");
    } else {
        System.out.println("there is at least one negative number");
    }
}
```

Usually it is not necessary to store the answer into a variable. We can write the method call directly as the condition:

```
ArrayList<Integer> numbers = new ArrayList<Integer>();
numbers.add(4);
numbers.add(7);
numbers.add(12);
numbers.add(9);

if (allPositive(numbers)) {
    System.out.println("all numbers are positive");
} else {
    System.out.println("there is at least one negative number");
}
```

18.2 The return command and ending a method

The execution of a method is stopped immediately when a command called `return` is executed. Using this information to our advantage, we write the `allPositive` method easier to understand.

```
public static boolean allPositive(ArrayList<Integer> numbers) {
    for (int number : numbers) {
        if (number < 0) {
            return false;
        }
    }
```

```
    // if the execution reached this far, no negative numbers were found
    // so we return true
    return true;
}
```

When we are going through the list of numbers and we find a negative number, we can exit the method by returning false. If there are no negative numbers on the list, we get to the end and therefore can return the value true. We now got rid of the helper variable inside the method!

Exercise 68: Is the number more than once in the list?

Create the method `moreThanOnce` that gets a list of integers and an integer (i.e. number) as parameter. If the number appears on the list *more than once* the method returns `true` and otherwise `false`.

The program body is the following:

```
public static boolean moreThanOnce(ArrayList<Integer> list, int number) {
    // write your code here
}

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

    ArrayList<Integer> list = new ArrayList<Integer>();
    list.add(3);
    list.add(2);
    list.add(7);
    list.add(2);

    System.out.println("Type a number: ");
    int number = Integer.parseInt(reader.nextLine());
    if (moreThanOnce(list, number)) {
        System.out.println(number + " appears more than once.");
    } else {
        System.out.println(number + " does not appear more than once.");
    }
}
```

```
Type a number: 2
2  appears more than once
```

```
Type a number: 3
3 does not appear more than once.
```

Exercise 69: Palindrome

Create the method `palindrome` that checks if a string is a palindrome (reads the same forward and backward).

The method can use the method `reverse` (from assignment number 56. Reversing text) as a helper. The method type is `boolean`, so it returns either `true` (the string is a palindrome) or `false` (the string is not a palindrome).

```
public static boolean palindrome(String text) {
    // write your code here
}

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

    System.out.println("Type a text: ");
    String text = reader.nextLine();
    if (palindrome(text)) {
       System.out.println("The text is a palindrome!");
    } else {
       System.out.println("The text is not a palindrome!");
    }
}
```

Example outputs:

```
Type a text: madam
The text is a palindrome!
```

```
Type a word: example
The text is not a palindrome!
```

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

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

TIETOJENKÄSITTELYTIETEEN LAITOS
INSTITUTIONEN FÖR DATAVETENSKAP
DEPARTMENT OF COMPUTER SCIENCE