Part 3

Lists

Learning Objectives

- You are familiar with the list structure and know how to use a list in a program.
- You are familiar with the concept of an index, you can add values to a list, and you know how to retrieve information from a list's indices.
- You know how to iterate over a list with multiple different loop types.
- You know how to check if a value exists in a list, and also know how to remove values from a list.
- You are aware of the list being a reference-type variable, and become familiar with using lists as method parameters.

In programming, we often encounter situations where we want to handle many values. The only method we've used so far has been to define a separate variable for storing each value. This is impractical.

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

The solution presented above is useless in effect — consider a situation in which there are thousands of words to store.

Programming languages offer tools to assist in storing a large quantity of values. We will next take a peek at perhaps the single most used tool in Java, the ArrayList, which is used for storing many values that are of the same type.

ArrayList is a pre-made tool in Java that helps dealing with lists. It offers various methods, including ones for adding values to the list, removing values from it, and also for the retrieval of a value from a specific place in the list. The concrete implementations — i.e., how the list is actually programmed — has been abstracted behind the methods, so that a programmer making use of a list doesn't need to concern themselves with its inner workings.

Using and Creating Lists

For an ArrayList to be used, it first needs be imported into the program. This is achieved by including the command `import java.util.ArrayList;` at the top of the program. Below is an example program where an ArrayList is imported into the program.

```
// import the list to make it available to the program
import java.util.ArrayList;

public class Program {

    public static void main(String[] args) {
        // no implementation yet
    }
}
```

Creating a new list is done with the command `ArrayList<Type> list = new ArrayList<>()`, where *Type* is the type of the values to be stored in the list (e.g. `String`). We create a list for storing strings in the example below.

```
// import the list so the program can use it
import java.util.ArrayList;

public class Program {

    public static void main(String[] args) {
        // create a list
        ArrayList<String> list = new ArrayList<>();

        // the list isn't used yet
    }
}
```

The type of the ArrayList variable is `ArrayList`. When a list variable is initialized, the type of the values to be stored is also defined in addition to the variable type — **all the variables stored in a given list are of the same type**. As such, the type of an ArrayList that stores strings is `ArrayList<String>`. A new list is created with the command `new ArrayList<>();`.

Defining the Type of Values That a List Can Contain

When defining the type of values that a list can include, the first letter of the element type has to be capitalized. A list that includes int-type variables has to be defined in the form `ArrayList<Integer>`; and a list that includes double-type variables is defined in the form `ArrayList<Double>`.

The reason for this has to do with how the ArrayList is implemented. Variables in Java can be divided into two categories: value type (primitive) and reference type (reference type) variables. **Value-type** variables such as `int` or `double` hold their actual values. **Reference-type** variables such as `ArrayList`, in contrast, contain a reference to the location that contains the value(s) relating to that variable.

Value-type variables can hold a very limited amount of information, whereas references can store a near limitless amount of it.

You'll find examples below of creating lists that contain different types of values.

```
ArrayList<Integer> list = new ArrayList<>();
list.add(1);
```

```
ArrayList<Double> list = new ArrayList<>();
list.add(4.2);
```

```
ArrayList<Boolean> list = new ArrayList<>();
list.add(true);
```

```
ArrayList<String> list = new ArrayList<>();
list.add("String is a reference-type variable");
```

Once a list has been created, ArrayList assumes that all the variables contained in it are reference types. Java automatically converts an `int` variable into `Integer` when one is added to a list, and the same occurs when a variable is retrieved from a list. The same conversion occurs for `double`-type variables, which are converted to `Double`. This means that even though a list is defined to contain `Integer`-type variables, variables of type `int` can also be added to it.

```
ArrayList<Integer> integers = new ArrayList<>();
int integer = 1;
integers.add(integer);

ArrayList<Double> doubles = new ArrayList<>();
double d = 4.2;
doubles.add(d);
```

We'll be returning to this theme since the categorization of variables into value and reference types affects our programs in other ways as well.

Adding to a List and Retrieving a Value from a Specific Place

The next example demonstrates the addition of a few strings into an ArrayList containing strings. Addition is done with the list method `add`, which takes the value to be added as a parameter. We then print the value at position zero. To retrieve a value from a certain position, you use the list method `get`, which is given the place of retrieval as a parameter.

To call a list method you first write the name of the variable describing the list, followed by a dot and the name of the method.

```
// import list so that the program can use it
import java.util.ArrayList;

public class WordListExample {

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

```
        // create the word list for storing strings
        ArrayList<String> wordList = new ArrayList<>();

        // add two values to the word list
        wordList.add("First");
        wordList.add("Second");

        // retrieve the value from position 0 of the word list, and print it
        System.out.println(wordList.get(0));
    }
}
```

Sample output

```
First
```

As can be seen, the `get` method retrieves the first value from the list when it is given the parameter `0`. This is because **list positions are counted starting from zero**. The first value is found by `wordList.get(0)`, the second by `wordList.get(1)`, and so on.

```
import java.util.ArrayList;

public class WordListExample {

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

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

        System.out.println(wordList.get(1));
    }
}
```

Sample output

```
Second
```

Programming exercise:

Third element

Points
1/1

The exercise contains a base that asks the user for strings and adds them to a list. The program stops reading when the user enters an empty string. The program then prints the first element of the list.

Your assignment is to modify the program so that instead of the first value, the third value on the list is printed. Remember that programmers start counting from zero! The program is allowed to malfunction if there are fewer than three entries on the list, so you don't need to prepare for such an event at all.

Sample output

```
Tom
Emma
Alex
Mary

Alex
```

Sample output

```
Emma
Alex
Mary

Mary
```

Exercise submission instructions

How to see the solution

Programming exercise:

Second plus third

Points
1/1

In the exercise template there is a program that reads integers from the user and adds them to a list. This ends when the user enters 0. The program then prints the first value on the list.

Modify the program so that instead of the first value, the program prints the sum of the second and third numbers. The program is allowed to malfunction if there are fewer than three entries on the list, so you don't need to prepare for such an event at all.

Sample output

```
1
3
5
7
0
8
```

Sample output

```
2
3
4
0
7
```

Exercise submission instructions

How to see the solution

Retrieving Information from a "Non-Existent" Place

If you try to retrieve information from a place that does not exist on the list, the program will print a `IndexOutOfBoundsException` error. In the example below, two values are added to a list, after which there is an attempt to print the value at place two on the list.

```
import java.util.ArrayList;

public class Example {

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

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

        System.out.println(wordList.get(2));
    }
}
```

Since the numbering (i.e., **indexing**) of the list elements starts with zero, the program isn't able to find anything at place two and its execution ends with an error. Below is a description of the error message caused by the program.

Sample output

```
Exception in thread "main" java.lang.IndexOutOfBoundsException:
Index: 2, Size: 2
at java.util.ArrayList.rangeCheck(ArrayList.java:653)
at java.util.ArrayList.get(ArrayList.java:429)
at Example.main(Example.java:(line))
Java Result: 1
```

The error message provides hints of the internal implementation of an ArrayList object. It lists all the methods that were called leading up to the error. First, the program called the `main` method, whereupon ArrayList's `get` method was called. Subsequently, the `get` method of ArrayList called

the `rangeCheck` method, in which the error occurred. This also acts as a good illustration of proper method naming. Even if we'd never heard of the `rangeCheck` method, we'd have good reason to guess that it checks if a searched place is contained within a given desired range. The error likely occurred because this was not the case.

Programming exercise:

IndexOutOfBoundsException

Points 1/1

A list is extremely useful for storing the values of variables for later use. That said, making mistakes is also relatively easy with a list.

There is a program that uses a list in the exercise template. Modify it so that its execution always produces the error `IndexOutOfBounds`. The user should not have to give any inputs to the program (e.g. write something on the keyboard)

You can also see a means for going through the values of a list — we will return to this topic a bit later.

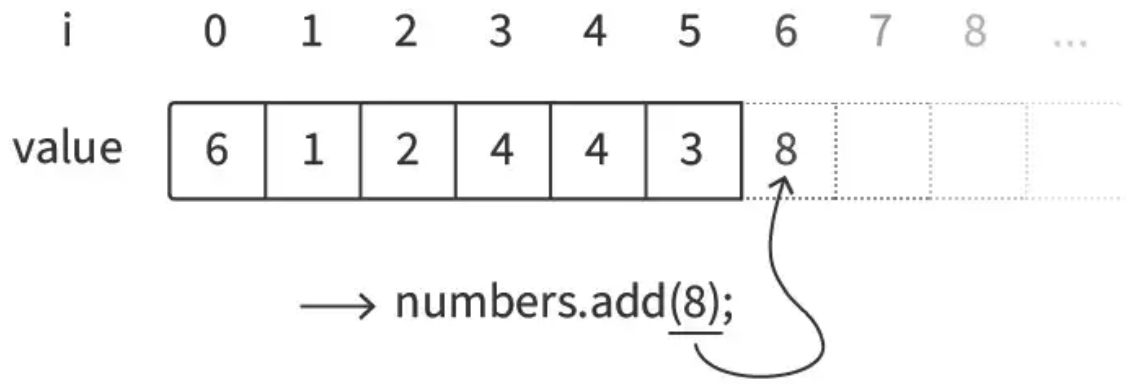
Exercise submission instructions

How to see the solution

A Place in a List Is Called an Index

Numbering places, i.e., indexing, always begins with zero. The list's first value is located at index 0, the second value at index 1, the third value at index 2, and so on. In programs, an index is denoted with a variable called `i`.

In the list above, the first value is 6 and the second value 1. If a new value was added to the list by calling the `add` method of `numbers` with 8 as parameter, the number 8 would be placed at index 6. It would be the seventh number in the list.



Similarly, by calling the method `get` with the parameter 4 the fifth number in the list would be retrieved.

Quiz:
ArrayList and index

Points:
1/1

Examine the program below.

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

researchers.add("Madeline");
researchers.add("Mike");
researchers.add("Lawrence");
```

What is the index of the string "Madeline"?

Select the correct answer

4

3

2

1

✓ 0

The answer is correct

Answered Tries remaining: 1

Importing multiple premade Java tools into the program

Each tool offered by Java has a name and location. The program can use a tool after it has been imported with the `import` command. The command is given the location and the name of the desired class. For example, the use of an ArrayList necessitates placing the command `import java.util.ArrayList;` to the top of the program.

```
import java.util.ArrayList;

public class ListProgram {

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

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

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

The same is generally true for all Java classes. We've been using the Scanner class for reading input by importing it with `import java.util.Scanner;`.

Bringing of multiple tools to use is straightforward. The tools to be imported are simply listed at the top of the program.

```
import java.util.ArrayList;
import java.util.Scanner;

public class ListProgram {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        ArrayList<String> wordList = new ArrayList<>();

        wordList.add("First");
        wordList.add(scanner.nextLine());
    }
}
```

Iterating Over a List

We'll next be examining methods that can be used to go through the values on a list. Let's start with a simple example where we print a list containing four values.

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

teachers.add("Simon");
teachers.add("Samuel");
teachers.add("Ann");
teachers.add("Anna");

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

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

Sample output

```
Simon
Samuel
Ann
Anna
```

The example is obviously clumsy. What if there were more values on the list? Or fewer? What if we didn't know the number of values on the list?

The number of values on a list is provided by the list's **size** method which returns the number of elements the list contains. The number is an integer (`int`), and it can be used as a part of an expression or stored in an integer variable for later use.

```
ArrayList<String> list = new ArrayList<>();
System.out.println("Number of values on the list: " + list.size());

list.add("First");
System.out.println("Number of values on the list: " + list.size());

int values = list.size();

list.add("Second");
System.out.println("Number of values on the list: " + values);
```

Sample output

```
Number of values on the list: 0
Number of values on the list: 1
Number of values on the list: 1
```

Programming exercise:

List size

Points

1/1

In the exercise template is a program that reads input from the user. Modify its working so that when the program quits reading, the program prints the number of values on the list.

Sample output

```
Tom
Emma
Alex
Mary

In total: 4
```

Sample output

```
Juno
Elizabeth
Mason
Irene
Olivia
Liam
Ida
Christopher
Mark
Sylvester
Oscar

In total: 11
```

NB! Be sure to use the `size` method of the list.

Exercise submission instructions

How to see the solution

Iterating Over a List Continued

Let's make a new version of the program that prints each index manually. In this intermediate version we use the `index` variable to keep track of the place that is to be outputted.

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

teachers.add("Simon");
teachers.add("Samuel");
teachers.add("Ann");
teachers.add("Anna");

int index = 0;

if (index < teachers.size()) {
    System.out.println(teachers.get(index)); // index = 0
    index = index + 1; // index = 1
}

if (index < teachers.size()) {
    System.out.println(teachers.get(index)); // index = 1
    index = index + 1; // index = 2
}

if (index < teachers.size()) {
    System.out.println(teachers.get(index)); // index = 2
    index = index + 1; // index = 3
}

if (index < teachers.size()) {
    System.out.println(teachers.get(index)); // index = 3
    index = index + 1; // index = 4
}

if (index < teachers.size()) {
    // this will not be executed since index = 4 and teachers.size() = 4
    System.out.println(teachers.get(index));
    index = index + 1;
}
```

We can see that there's repetition in the program above.

We can convert the `if` statements into a `while` loop that is repeated until the condition `index < teachers.size()` no longer holds (i.e., the value of the variable `index` grows too great).

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

teachers.add("Simon");
teachers.add("Samuel");
teachers.add("Ann");
teachers.add("Anna");

int index = 0;
// Repeat for as long as the value of the variable `index`
// is smaller than the size of the teachers list
while (index < teachers.size()) {
    System.out.println(teachers.get(index));
    index = index + 1;
}
```

Now the printing works regardless of the number of elements.

The for-loop we inspected earlier used to iterate over a known number of elements is extremely handy here. We can convert the loop above to a `for`-loop, after which the program looks like this.

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

teachers.add("Simon");
teachers.add("Samuel");
teachers.add("Ann");
teachers.add("Anna");

for (int index = 0; index < teachers.size(); index++) {
    System.out.println(teachers.get(index));
}
```

Sample output

```
Simon
Samuel
Ann
Anna
```

The index variable of the for-loop is typically labelled `i`:

```
for (int i = 0; i < teachers.size(); i++) {
    System.out.println(teachers.get(i));
}
```

Let's consider using a list to store integers. The functionality is largely the same as in the previous example. The greatest difference has to do with the initialization of the list — the type of value to be stored is defined as `Integer`, and the value to be printed is stored in a variable called `number` before printing.

```
ArrayList<Integer> numbers = new ArrayList<>();

numbers.add(1);
numbers.add(2);
numbers.add(3);
numbers.add(4);

for (int i = 0; i < numbers.size(); i++) {
    int number = numbers.get(i);
    System.out.println(number);
    // alternatively: System.out.println(numbers.get(i));
}
```

Sample output

```
1
2
3
4
```

Printing the numbers in the list in reverse order would also be straightforward.

```
ArrayList<Integer> numbers = new ArrayList<>();

numbers.add(1);
numbers.add(2);
numbers.add(3);
numbers.add(4);

int index = numbers.size() - 1;
while (index >= 0) {
```

```
    int number = numbers.get(index);
    System.out.println(number);
    index = index - 1;
}
```

Sample output

```
4
3
2
1
```

The execution of the program is visualized below. However, the visualization does not show the internal state of the ArrayList (i.e., the values contained by it).

```
1  import java.util.ArrayList;
2
3  public class RepeatStatement {
4     public static void main(String[] args) {
5        ArrayList<Integer> numbers = new ArrayList<>();
6
7        numbers.add(1);
8        numbers.add(2);
9        numbers.add(3);
10       numbers.add(4);
11
12       int index = numbers.size() - 1;
13       while (index >= 0) {
14          int number = numbers.get(index);
15          System.out.println(number);
16          index = index - 1;
17       }
18    }
19 }
```

main:5

Name	Value

Output

Try and recreate the previous example with the for loop!

Notice about the following exercises

The next exercises are meant for learning to use lists and indices. Even if you could complete the exercises without a list, concentrate on training to use it. The functionality in the exercises is to be implemented after reading the input numbers.

Programming exercise:

Last in list

Points 1/1

In the exercise template there is a program that reads inputs from the user and adds them to a list. Reading is stopped once the user enters an empty string.

Your task is to modify the method to print the last read value after it stops reading. Print the value that was read last from the list. Use the method that tells the size of a list to help you.

Sample output

```
Tom
Emma
Alex
Mary

Mary
```

Sample output

```
Juno
Elizabeth
Mason
Irene
Olivia
Liam
Ida
Christopher
Mark
Sylvester
Oscar

Oscar
```

Exercise submission instructions

How to see the solution

Programming exercise:

First and last

Points 1/1

In the exercise template there is a program that reads inputs from the user and adds them to a list. Reading is stopped once the user enters an empty string.

Modify the program to print both the first and the last values after the reading ends. You may suppose that at least two values are read into the list.

Sample output

```
Tom
Emma
Alex
Mary
```

Tom
Mary

Sample output

Juno
Elizabeth
Mason
Irene
Olivia
Liam
Ida
Christopher
Mark
Sylvester
Oscar

Juno
Oscar

Exercise submission instructions

How to see the solution

Programming exercise:

Remember these numbers

Points
1/1

The exercise template contains a base that reads numbers from the user and adds them to a list. Reading is stopped once the user enters the number -1.

Expand the functionality of the program so that after reading the numbers, it prints all the numbers received from the user. The number used to indicate stopping should not be printed.

Sample output

```
72
2
8
11
-1
72
2
8
11
```

Exercise submission instructions

How to see the solution

Programming exercise:

Only these numbers

Points 1/1

The exercise template contains a base that reads numbers from the user and adds them to a list. Reading is stopped once the user enters the number -1.

Expand the program to ask for a start and end indices once it has finished asking for numbers. After this the program shall prints all the numbers in the list that fall in the specified range (between the indices given by the user, inclusive). You may assume that the user gives indices that match some numbers in the list.

Sample output

```
72
2
8
11
-1
```

```
From where? 1
To where? 2
2
8
```

Sample output

```
72
2
8
11
-1
From where? 0
To where? 2
72
2
8
```

Exercise submission instructions

How to see the solution

Programming exercise:

Greatest in list

Points 1/1

The exercise template contains a base that reads numbers from the user and adds them to a list. Reading is stopped once the user enters the number -1.

Continue developing the program so that it finds the greatest number in the list and prints its value after reading all the numbers. The programming should work in the following manner.

Sample output

```
72
2
8
93
11
-1

The greatest number: 93
```

You can use the source code below as an example. It is used to find the smallest number.

```
// assume we have a list that contains integers

int smallest = list.get(0);

for(int i = 0; i < list.size(); i++) {
    int number = list.get(i);
    if (smallest > number) {
        smallest = number;
    }
}

System.out.println("The smallest number: " + smallest);
```

Exercise submission instructions

How to see the solution

Programming exercise:

Index of

Points

1/1

The exercise template contains a base that reads numbers from the user and adds them to a list. Reading is stopped once the user enters the number -1.

Expand the program by adding a functionality that asks the user for a number, and reports that number's index in the list. If the number is not found, the program should not print anything.

Sample output

```
72
2
8
8
11
-1

Search for? 2
2 is at index 1
```

Sample output

```
72
2
8
8
11
-1

Search for? 8
8 is at index 2
8 is at index 3
```

Exercise submission instructions

How to see the solution

Programming exercise:
Index of smallest

Points
1/1

Write a program that reads numbers from the user. When number 9999 is entered, the reading process stops. After this the program will print the smallest number in the list, and also the indices where that number is found. Notice: the smallest number can appear multiple times in the list.

Sample output

```
72
2
8
8
11
9999

Smallest number: 2
Found at index: 1
```

Sample output

```
72
44
8
8
11
9999

Smallest number: 8
Found at index: 2
Found at index: 3
```

Hint: combine the programs you wrote for the exercises "Greatest number in the list" and "Index of the requested number". First find the smallest number, and then find the index of that number.

Exercise submission instructions

How to see the solution

Iterating Over a List with a For-Each Loop

If you don't need to keep track of the index as you're going through a list's values, you can make use of the **for-each** loop. It differs from the previous loops in that it has no separate condition for repeating or incrementing.

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


teachers.add("Simon");
teachers.add("Samuel");
teachers.add("Ann");
teachers.add("Anna");

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

In practical terms, the for-each loop described above hides some parts of the for-loop we practiced earlier. The for-each loop would look like this if implemented as a for-loop:

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

teachers.add("Simon");
teachers.add("Samuel");
teachers.add("Ann");
teachers.add("Anna");
for (int i = 0; i < teachers.size(); i++) {
    String teacher = teachers.get(i);
    // contents of the for each loop:
    System.out.println(teacher);
}
```

In practice, the for-each loop examines the values of the list in order one at a time. The expression is defined in the following format: `for (TypeOfVariable nameOfVariable: nameOfList)`, where `TypeOfVariable` is the list's element type, and `nameOfVariable` is the variable that is used to store each value in the list as we go through it.

```java
import java.util.ArrayList;

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

        teachers.add("Simon");
        teachers.add("Samuel");
        teachers.add("Ann");
        teachers.add("Anna");

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

main:5

Name	Value

Output

Prev 1 / 20 Next

Programming exercise:

Sum of a list

Points 1/1

The exercise template contains a base that reads numbers from the user and adds them to a list. Reading is stopped once the user enters the number -1.

Modify the program so that after reading the numbers it calculates and prints the sum of the numbers in the list.

Sample output

72
2
8
11
-1

Sum: 93

Exercise submission instructions

How to see the solution

Programming exercise:

Average of a list

Points
1/1

The exercise template contains a base that reads numbers from the user and adds them to a list. Reading is stopped once the user enters the number -1.

When reading ends, calculate the average of the numbers in it, and then print that value.

Sample output

72
2
8
11
-1

Average: 23.25

Exercise submission instructions

How to see the solution

Removing from a List and Checking the Existence of a Value

The list's **remove** method removes the value that is located at the index that's given as the parameter. The parameter is an integer.

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

list.add("First");
list.add("Second");
list.add("Third");

list.remove(1);

System.out.println("Index 0 so the first value: " + list.get(0));
System.out.println("Index 1 so the second value: " + list.get(1));
```

Sample output

```
Index 0 so the first value: First
Index 1 so the second value: Third
```

If the parameter given to `remove` is the same type as the values in the list, but not an integer, (integers are used to remove from a given index), it can be used to remove a value directly from the list.

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

list.add("First");
list.add("Second");
list.add("Third");

list.remove("First");

System.out.println("Index 0 so the first value: " + list.get(0));
System.out.println("Index 1 so the second value: " + list.get(1));
```

Sample output

```
Index 0 so the first value: Second
Index 1 so the second value: Third
```

If the list contains integers, you cannot remove a number value by giving an `int` type parameter to the remove method. This would remove the number from the index that the parameter indicates, instead of an element on the list that has the same value as the parameter. To remove an integer type value you can convert the parameter to Integer type; this is achieved by the `valueOf` method of the Integer class.

```
ArrayList<Integer> list = new ArrayList<>();

list.add(15);
list.add(18);
list.add(21);
list.add(24);

list.remove(2);
list.remove(Integer.valueOf(15));

System.out.println("Index 0 so the first value: " + list.get(0));
System.out.println("Index 1 so the second value: " + list.get(1));
```

Sample output

```
Index 0 so the first value: 18
Index 1 so the second value: 24
```

Quiz:

Remove a number

Points:

1/1

What number is removed from the list?

```
ArrayList<Integer> numbers = new ArrayList<>();

numbers.add(3);
numbers.add(1);
```

```
numbers.add(6);
numbers.add(0);

numbers.remove(3);
```

Select the correct answer

- 3
- 1
- 6
- ✓ 0
- something else

The answer is correct

Answered Tries remaining: 1

Quiz:
Remove a number, part 2

Points:
1/1

What number is removed from the list?

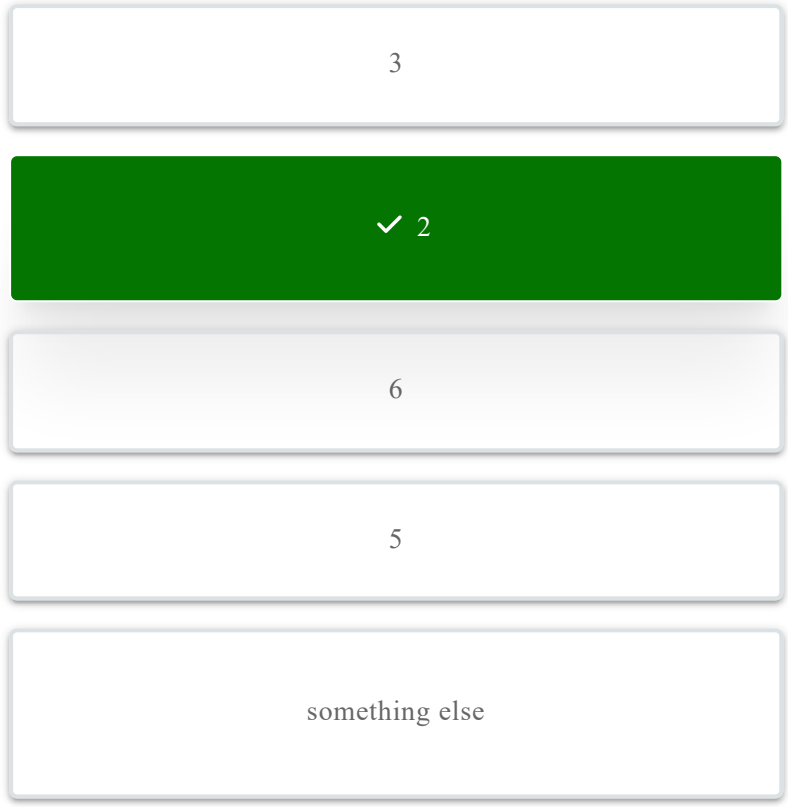
```
ArrayList<Integer> numbers= new ArrayList<>();

numbers.add(2);
numbers.add(6);
numbers.add(5);
```

```
numbers.add(3);

numbers.remove(Integer.valueOf(2));
```

Select the correct answer



The answer is correct

Answered Tries remaining: 1

The list method **contains** can be used to check the existence of a value in the list. The method receives the value to be searched as its parameter, and it returns a boolean type value (`true` or `false`) that indicates whether or not that value is stored in the list.

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

list.add("First");
list.add("Second");
list.add("Third");

System.out.println("Is the first found? " + list.contains("First"));

boolean found = list.contains("Second");
if (found) {
```

```
    System.out.println("Second was found");
}

// or more simply
if (list.contains("Second")) {
    System.out.println("Second can still be found");
}
```

Sample output

```
Is the first found? true
Second was found
Second can still be found
```

Programming exercise:

On the list?

Points

/

In the exercise template there is a program that reads inputs from the user until an empty string is entered. Add the following functionality to it: after reading the inputs one more string is requested from the user. The program then tell whether that string was found in the list or not.

Sample output

```
Tom
Emma
Alex
Mary

Search for? Mary
Mary was found!
```

Sample output

```
Tom
Emma
Alex
Mary
```

```
Search for? Logan
Logan was not found!
```

List as a Method Parameter

Like other variables, a list can be used as a parameter to a method too. When the method is defined to take a list as a parameter, the type of the parameter is defined as the type of the list and the type of the values contained in that list. Below, the method `print` prints the values in the list one by one.

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

We're by now familiar with methods, and it works in the same way here. In the example below we use the `print` method that was implemented above.

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

```
strings.add("First");
strings.add("Second");
strings.add("Third");

print(strings);
```

Sample output

```
First
Second
Third
```

The chosen parameter in the method definition is not dependent on the list that is passed as parameter in the method call. In the program that calls `print`, the name of the list variable is `strings`, but inside the method `print` the variable is called `list` — the name of the variable that stores the list could also be `printables`, for instance.

It's also possible to define multiple variables for a method. In the example the method receives two parameters: a list of numbers and a threshold value. It then prints all the numbers in the list that are smaller than the second parameter.

```
public static void printSmallerThan(ArrayList<Integer> numbers, int threshold)
    for (int number: numbers) {
        if (number < threshold) {
            System.out.println(number);
        }
    }
}
```

```
ArrayList<Integer> list = new ArrayList<>();

list.add(1);
list.add(2);
list.add(3);
list.add(2);
list.add(1);
```

```
printSmallerThan(list, 3);
```

Sample output

```
1
2
2
1
```

Programming exercise:

Print in range

Points

/

Create the method `public static void printNumbersInRange(ArrayList<Integer> numbers, int lowerLimit, int upperLimit)` in the exercise template. The method prints the numbers in the given list whose values are in the range [lowerLimit, upperLimit]. A few examples of using the method are supplied below.

```
ArrayList<Integer> numbers = new ArrayList<>();
numbers.add(3);
numbers.add(2);
numbers.add(6);
numbers.add(-1);
numbers.add(5);
numbers.add(1);

System.out.println("The numbers in the range [0, 5]");
printNumbersInRange(numbers, 0, 5);

System.out.println("The numbers in the range [3, 10]");
printNumbersInRange(numbers, 3, 10);
```

Sample output

```
The numbers in the range [0, 5]
3
2
```

```
5
1
The numbers in the range [3, 10]
3
6
5
```

As before, a method can also return a value. The methods that return values have the type of the return value in place of the `void` keyword, and the actual returning of the value is done by the `return` command. The method below returns the size of the list.

```
public static int size(ArrayList<String> list) {
    return list.size();
}
```

You can also define own variables for methods. The method below calculates the average of the numbers in the list. If the list is empty, it returns the number -1.

```
public static double average(ArrayList<Integer> numbers) {
    if (numbers.size() == 0) {
        return -1.0;
```

```
    }

    int sum = 0;
    for (int number: numbers) {
        sum = sum + number;
    }

    return 1.0 * sum / numbers.size();
}
```

Programming exercise:

Sum

Points

/

Create the method `public static int sum(ArrayList<Integer> numbers)` in the exercise template. The method is to return the sum of the numbers in the parameter list.

```
ArrayList<Integer> numbers = new ArrayList<>();
numbers.add(3);
numbers.add(2);
numbers.add(6);
numbers.add(-1);
System.out.println(sum(numbers));

numbers.add(5);
numbers.add(1);
System.out.println(sum(numbers));
```

Sample output

```
10
16
```

On Copying the List to a Method Parameter

Earlier we have used integers, floating point numbers, etc. as method parameters. When variables such as `int` are used as method parameters, the value of the variable is copied for the method's use. The same occurs in the case that the parameter is a list.

Lists, among practically all the variables that can store large amounts of information, are *reference-type variables*. This means that the value of the variable is a reference that points to the location that contains the information.

When a list (or any reference-type variable) is copied for a method's use, the method receives the value of the list variable, i.e., a *reference*. In such a case the **method receives a reference to the real value of a reference-type variable**, and the method is able to modify the value of the original reference type variable, such as a list. In practice, the list that the method receives as a parameter is the same list that is used in the program that calls the method.

Let's look at this briefly with the following method.

```
public static void removeFirst(ArrayList<Integer> numbers) {
    if (numbers.size() == 0) {
        return;
    }

    numbers.remove(0);
}
```

```
ArrayList<Integer> numbers = new ArrayList<>();
numbers.add(3);
numbers.add(2);
numbers.add(6);
numbers.add(-1);

System.out.println(numbers);

removeFirst(numbers);

System.out.println(numbers);

removeFirst(numbers);
removeFirst(numbers);
removeFirst(numbers);

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

Sample output

```
[3, 2, 6, -1]
[2, 6, -1]
[]
```

Programming exercise:

Remove last

Points

/

Create the method `public static void removeLast(ArrayList<String> strings)` in the exercise template. The method should remove the last value in the list it receives as a parameter. If the list is empty, the method does nothing.

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

strings.add("First");
strings.add("Second");
strings.add("Third");

System.out.println(strings);

removeLast(strings);
```

```
removeLast(strings);

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

Sample output

```
[First, Second, Third]
[First]
```

A Summary of List Methods

The ArrayList contains a bunch of useful methods. The method always operates on the list object that is connected to the method call — this connection is established with a dot. The example below illustrates that a program can contain multiple lists, which also holds true for other variables. Two separate lists are created below.

```
ArrayList<String> exercises1 = new ArrayList<>();
ArrayList<String> exercises2 = new ArrayList<>();

exercises1.add("Ada Lovelace");
exercises1.add("Hello World! (Ja Mualima!)");
exercises1.add("Six");

exercises2.add("Adding a positive number");
exercises2.add("Employee's pension insurance");
```

```
System.out.println("The size of list 1: " + exercises1.size());
System.out.println("The size of list 2: " + exercises2.size());

System.out.println("The first value of the first list " + exercises1.get(0));
```

Sample output

```
The size of list 1: 3
The size of list 2: 2
The first value of the first list Ada Lovelace
The last value of the second list Employee's pension insurance
```

Each list is its own separate entity, and the list methods always concern the list that was used to call the method. A summary of some list methods is provided below. It's assumed that the created list contains variables of type string.

- Adding to a list is done with the method `add` that receives the value to be added as a parameter.

```
ArrayList<String> list = new ArrayList<>();
list.add("hello world!");
```

- The number of elements in a list can be discovered with the non-parameterized method `size`; it returns an integer.

```
ArrayList<String> list = new ArrayList<>();
int size = list.size();
System.out.println(size);
```

- You can retrieve a value from a certain index with the method `get` that is given the index at which the value resides as a parameter.

```
ArrayList<String> list = new ArrayList<>();
list.add("hello world!");
String string = list.get(0);
System.out.println(string);
```

- Removing elements from a list is done with the help of `remove`. It receives as a parameter either the value that is to be removed, or the index of the value to be removed.

```
ArrayList<String> list = new ArrayList<>();
// remove the string "hello world!"
list.remove("hello world!");
 // remove the value at index 3
list.remove(3);
```

- Checking for the existence of a value is done with the method `contains`. It's provided the value being searched for as a parameter, and it returns a boolean value.

```
ArrayList<String> list = new ArrayList<>();
boolean found = list.contains("hello world!");
```

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

→ 3. Arrays

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

In this part:

Source code of the material

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

Credits and about the material.

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

MASSIIVISET AVOIMET VERKKOKURSSIT
MASSIVE OPEN ONLINE COURSES · MOOC.FI