Part 8

Short recap

Learning Objectives

- You will brush up on the contents of the parts 1-7

Below there are a few exercises that go over the material introduced in the parts 1-7. The assignments are identical to some exercises that have appeared earlier in the material. If you have completed some of the exercises before, please solve them again without looking at your earlier solution.

Programming exercise:

Cubes

Points 1/1

Write a program that reads strings from the user until the user inputs the string "end". As long as the input is not "end" the program should handle the input as an integer and print the cube of the integer (meaning number * number * number). Below are some sample outputs

Sample output

```
3
27
-1
-1
11
```

```
1331
end
```

Sample output

```
end
```

Exercise submission instructions

How to see the solution

Programming exercise:

Average of positive numbers

Points 1/1

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

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

Below a few examples of the program's output

Sample output

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

Sample output

```
0
```

Cannot calculate the average

Sample output

```
-3
1
0
1.0
```

Sample output

```
1
1
0
1.0
```

Exercise submission instructions

How to see the solution

Programming exercise:

Liquid containers (3 parts)

Points 3/3

NB! By submitting a solution to a part of an exercise which has multiple parts, you can get part of the exercise points. You can submit a part by using the 'submit' button on NetBeans. More on the programming exercise submission instructions: Exercise submission instructions.

Let's create an interactive program to control two liquid containers. The containers are named "first" and "second". Each are capable of containing 100 liters of liquid at a time.

The program offers the functionality to add, move and remove liquid. Using the "add" command will add liquid to the first container, "move"

will move liquid from the first container to the second container and "remove" will remove liquid from the second container.

The commands must be defined as following:

- `add amount` adds the amount of liquid specified by the parameter to the first container. The inserted amount must be specified as an integer. The container can't hold more than a hundred liters and everything added past that will go to waste.
- `move amount` moves the amount of liquid specified by the parameter from the first container to the second container. The given amount must be specified as an integer. If the program is requested to move more liquid than than the first container currently holds, move all the remaining liquid. The second container can't hold more than one hundred liters of liquid and everything past that will go to waste.
- `remove amount` removes the amount of liquid specified by the parameter from the second container. If the program is requested to remove more liquid than the container currently holds, remove all the remaining liquid.

After every command the program will print the contents of both containers. You don't have to take negative values into consideration.

All the functionality must be added to the method `main` in the class `LiquidContainers` (do not create new methods). The template already contains a loop which exits the program with the command "quit".

A reminder of how to split a string below.

```
String input = scan.nextLine();
String[] parts = input.split(" ");

String command = parts[0];
int amount = Integer.valueOf(parts[1]);
```

Part 1: Adding

Implement the functionality to add liquid to the first container. The user interface should work as follows:

Sample output

```
First: 0/100
Second: 0/100
add 5

First: 5/100
Second: 0/100
add 25

First: 30/100
Second: 0/100
add 60

First: 90/100
Second: 0/100
add 1000

First: 100/100
Second: 0/100
add -5

First: 100/100
Second: 0/100
quit
```

Part 2: Moving

Implement the functionality to move liquid from the first container to the second. The user interface should work as follows:

Sample output

```
First: 0/100
Second: 0/100
add 1000

First: 100/100
Second: 0/100
move 50
```

First: 50/100
Second: 50/100
add 100

First: 100/100
Second: 50/100
move 100

First: 0/100
Second: 100/100
quit

Second example:

Sample output

First: 0/100
Second: 0/100
move 30

First: 0/100
Second: 0/100
add 10

First: 10/100
Second: 0/100
move -5

First: 10/100
Second: 0/100
move 20

First: 0/100
Second: 10/100
move 10

First: 0/100
Second: 10/100
quit

Part 3: Removing

Implement the functionality to remove liquid from the second container. The user interface should work as follows:

Sample output

```
First: 0/100
Second: 0/100
remove 10

First: 0/100
Second: 0/100
add 20

First: 20/100
Second: 0/100
remove 5

First: 20/100
Second: 0/100
move 15

First: 5/100
Second: 15/100
remove 5

First: 5/100
Second: 10/100
remove 20

First: 5/100
Second: 0/100
quit
```

Exercise submission instructions

How to see the solution

Programming exercise:

Liquid Containers 2.0 (2 parts)

Points 2/2

NB! By submitting a solution to a part of an exercise which has multiple parts, you can get part of the exercise points. You can submit a part by using the 'submit' button on NetBeans. More on the programming exercise submission instructions: Exercise submission instructions.

Let's redo the previous program for handling two liquid containers. This time we'll create a class `Container`, which is responsible for managing the contents of a container.

Part 1: Container

Make a class called `Container`. The class must have a constructor which does not take any parameters, and the following methods:

- `public int contains()` which returns the amount of liquid in a container as an integer.
- `public void add(int amount)` which adds the amount of liquid given as a parameter to the container. If the amount is negative, no liquid is added.

A container can hold maximum of 100 units of liquid.

- `public void remove(int amount)` which removes the amount of liquid given as a parameter from the container. If the amount is negative, no liquid is removed. A container can never hold less than 0 units of liquid.
- `public String toString()` which returns the container as a string formatted "*amount of liquid*/100", for example "32/100".

The class should work as follows:

```
Container container = new Container();
System.out.println(container);

container.add(50);
System.out.println(container);
System.out.println(container.contains());

container.remove(60);
System.out.println(container);
```

```
container.add(200);
System.out.println(container);
```

Sample output

```
0/100
50/100
50
0/100
100/100
```

Part 2: Functionality

Copy the user interface you implemented for the previous example, and modify it to use the new `Container` class. The main method in the class `LiquidContainers2` must start the program.

Below is some sample output. The user interface should work as follows:

Sample output

```
First: 0/100
Second: 0/100
remove 10

First: 0/100
Second: 0/100
add 20

First: 20/100
Second: 0/100
remove 5

First: 20/100
Second: 0/100
move 15

First: 5/100
Second: 15/100
remove 5
```

First: 5/100
Second: 10/100
remove 20

First: 5/100
Second: 0/100
quit

Exercise submission instructions

How to see the solution

Programming exercise:

To do list (2 parts)

Points
2/2

NB! By submitting a solution to a part of an exercise which has multiple parts, you can get part of the exercise points. You can submit a part by using the 'submit' button on NetBeans. More on the programming exercise submission instructions: Exercise submission instructions.

In this exercise we are going to create a program that can be used to create and modify a to-do list. The final product will work in the following manner.

Sample output

Command: add
Task: go to the store
Command: add
Task: vacuum clean
Command: list
1: go to the store
2: vacuum clean
Command: completed
Which task was completed? 2
Task go to the store tehty

```
Command: list
1: go to the store
Command: add
Task: program
Command: list
1: go to the store
2: program
Command: stop
```

We will build the program in parts.

Part 1: TodoList

Create a class called `TodoList`. It should have a constructor without parameters and the following methods:

- `public void add(String task)` - add the task passed as a parameter to the todo list.
- `public void print()` - prints the exercises. Each task has a number associated with it on the print statement — use the task's index here (+1).
- `public void remove(int number)` - removes the task associated with the given number; the number is the one seen associated with the task in the print.

```
TodoList list = new TodoList();
list.add("read the course material");
list.add("watch the latest fool us");
list.add("take it easy");

list.print();
list.remove(2);

System.out.println();
list.print();
```

Sample output

```
1: read the course material
2: watch the latest fool us
3: take it easy

1: read the course material
2: take it easy
```

NB! You may assume that the `remove` method is given a number that corresponds to a real task. The method only has to correctly work once after each print call.

Another example:

```
TodoList list = new TodoList();
list.add("read the course material");
list.add("watch the latest fool us");
list.add("take it easy");
list.print();
list.remove(2);
list.print();
list.add("buy rasins");
list.print();
list.remove(1);
list.remove(1);
list.print();
```

Sample output

```
1: read the course material
2: watch the latest fool us
3: take it easy
1: read the course material
2: take it easy
1: read the course material
2: take it easy
3: buy rasins
1: buy rasins
```

Part 2: User interface

Next, implement a class called `UserInterface`. It should have a constructor with two parameters. The first parameter is an instance of the class `TodoList`, and the second is an instance of the class `Scanner`. In addition to the constructor, the class should have the method `public void start()` that is used to start the text user interface. The text UI works with an eternal looping statement (`while-true`), and it must offer the following commands to the user:

- The command `stop` stops the execution of the loop, after which the execution of the program advances out of the `start` method.
- The command `add` asks the user for the next task to be added. Once the user enters this task, it should be added to the to-do list.
- The commmand `list` prints all the tasks on the to-do list.
- The command `remove` asks the user to enter the id of the task to be removed. When this has been entered, the specified task should be removed from the list of tasks.

Below is an example of how the program should work.

Sample output

```
Command: add
To add: write an essay
Command: add
To add: read a book
Command: list
1: write an essay
2: read a book
Command: remove
Which one is removed? 1
Command: list
1: read a book
Command: remove
Which one is removed? 1
Command: list
Command: add
To add: stop
Command: list
1: stop
Command: stop
```

NB! The user interface is to use the TodoList and Scanner that are passed as parameters to the constructor.

Exercise submission instructions

How to see the solution

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

→ 2. Hash Map

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