Larger programming exercises

To conclude the seventh part, you'll do a few more extensive exercises. There is no predefined structure for these tasks — think about the classes and objects that will help you complete the task while you're completing it.

Programming exercise:

Grade statistics (4 parts)

Points 4/4

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 create a program for printing statistics for points in course. The program receives points (integers from zero to one hundred) as input, based on which it prints statistics about grades. Reading of input stops when the user enters the number -1. Numbers that are not within the interval [0-100] should not be taken into account when calculating the statistics.

A string read from the user can be converted to an integer using the `Integer` class' method `valueOf`. It works as follows:

```java
String numberAsString = "3";
int number = Integer.valueOf(numberAsString);

System.out.println(numberAsString + 7);
System.out.println(number + 7);
```

Sample output

```
37
10
```

Part 1: Point averages

Write a program that reads integers representing course point totals from the user. Numbers between [0-100] are acceptable and the number -1 ends the reading of input. Other numbers are erroneous input, which should be ignored. When the user enters the number -1, the program should print the average of the point totals that were input.

Sample output

```
Enter point totals, -1 stops:
-42
24
42
72
80
52
-1
Point average (all): 54.0
```

Sample output

```
Enter point totals, -1 stops:
50
51
52
-1
Point average (all): 51.0
```

Part 2: Point average for points giving a passing grade

Extend the program, such that it in addition to giving the point average of all totals also provides the point average for points giving a passing grade.

A passing grade is achieved by getting a minimum of 50 course points. You may assume that the user always provides at least one integer between [0-100]. If there are no numbers giving a passing grade, the program should print a line "-" where the average would be.

Sample output

```
Enter point totals, -1 stops:
-42
24
42
72
80
52
-1
Point average (all): 54.0
Point average (passing): 68.0
```

Sample output

```
Enter point totals, -1 stops:
49
48
47
-1
Point average (all): 48.0
Point average (passing): -
```

Part 3: Pass percentage

Extend the program from the previous part, such that it also print the pass percentage. The pass percentage is calculated using the formula *100 * passing / participants*.

Sample output

```
Enter point totals, -1 stops:
49
48
47
-1
Point average (all): 48.0
Point average (passing): -
Pass percentage: 0.0
```

Sample output

```
Enter point totals, -1 stops:
102
-4
33
77
99
1
-1
Point average (all): 52.5
Point average (passing): 88.0
Pass percentage: 50.0
```

Part 4: Grade distribution

Extend the program, such that it also prints the grade distribution. The grade distribution is as follows:

points	grade
< 50	failed, i.e. 0
< 60	1
< 70	2
< 80	3

< 90	4
>= 90	5

Each point total is converted to a grade based on the above table. If a point total isn't within [0-100], it should be ignored.

The grade distribution is printed out as stars. E.g. if there is one point total giving the grade 5, then it should print the row *5: **. If there are no point totals giving a particular grade, then no stars should be printed for it. In the sample below this is true for e.g. the grade 4.

Sample output

```
Enter point totals, -1 stops:
102
-2
1
33
77
99
-1
Point average (all): 52.5
Point average (passing): 88.0
Pass percentage: 50.0
Grade distribution:
5: *
4:
3: *
2:
1:
0: **
```

Exercise submission instructions

How to see the solution

Programming exercise:

Recipe search (4 parts)

Points
4/4

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 allows for searching for recipes based on their name, cooking time, or the name of an ingredient. The program should read the recipes from a file that the user provides. *It might be a good idea to brush up on reading information from files (part 4) before beginning*

Each recipe consists of three or more rows in a recipe file. The first row is for the name of the recipe, the second the cooking time (an integer), and the third and possibly following rows list the ingredients used in the recipe. An empty row follows the last ingredient row. There can be many recipes in a single file. Below, an example file containing recipes is described.

Sample output

```
Pancake dough
60
milk
egg
flour
salt
butter

Meatballs
20
ground meat
egg
breadcrumbs

Tofu rolls
30
tofu
```

```
rice
water
carrot
cucumber
avocado
wasabi
```

The program will be implemented in parts. First we'll create the possibility to read and list recipes. After that we'll add the functionality to search for recipes based on their name, cooking time, or the name of an ingredient.

There is a file called `recipes.txt` supplied with the exercise base. You can use it for testing purposes. *Notice that the program should not list the ingredients of the recipes, but they will be used in the search functionality.*

Part 1: Reading and listing recipes

First create the functionality to read and list recipes. The user interface of the program is described below. You may assume that the user only enters files that exist. Below we assume that the example recipes given earlier in the exercise description are stored in the file `recipes.txt`.

Sample output

```
File to read: recipes.txt

Commands:
list - lists the recipes
stop - stops the program

Enter command: list

Recipes:
Pancake dough, cooking time: 60
Meatballs, cooking time: 20
Tofu rolls, cooking time: 30

Enter command:  stop
```

Part 2: Finding recipes by name

Make it possible to find recipes by their names. Finding by name is done with the command `find name`, after which the user is asked for the name that is used to search. The search should print all the recipes whose names contain the string given by the user.

Sample output

```
File to read: recipes.txt

Commands:
list - lists the recipes
stop - stops the program
find name - searches recipes by name

Enter command: list

Recipes:
Pancake dough, cooking time: 60
Meatballs, cooking time: 20
Tofu rolls, cooking time: 30

Enter command: find name
Searched word: roll

Recipes:
Tofu rolls, cooking time: 30

Enter command:  stop
```

Part 3: Searching for recipes by cooking time

Next, implement the possibility to find recipes based on their cooking time. This is done with the command `find cooking time`, after which the user is asked for the longest acceptable cooking time. The program should react by printing all the recipes whose cooking times don't exceed the cooking time given by the user (so equal or less time).

Sample output

```
File to read: recipes.txt

Commands:
list - lists the recipes
stop - stops the program
find name - searches recipes by name
find cooking time - searches recipes by cooking time

Enter command: find cooking time
Max cooking time: 30

Recipes:
Meatballs, cooking time: 20
Tofu rolls, cooking time: 30

Enter command: find cooking time
Max cooking time: 15

Recipes:

Enter command: find name
Searched word: roll

Recipes:
Tofu rolls, cooking time: 30

Enter command:  stop
```

Part 4: Finding recipes based on their ingredients

Finally, add the functionality to search for recipes based on their ingredients. This is done by choosing the command `find ingredient`, after which the user is asked for a string. The program should then print all the recipes that contain the specified string. Notice that with this option the given string must match exactly the ingredient that is searched for (e.g. "ugar" will return different results than "sugar").

Sample output

```
File to read: recipes.txt

Commands:
list - lists the recipes
stop - stops the program
find name - searches recipes by name
find cooking time - searches recipes by cooking time
find ingredient - searches recipes by ingredient

Enter command: find cooking time
Max cooking time: 30

Recipes:
Meatballs, cooking time: 20
Tofu rolls, cooking time: 30

Enter command: find ingredient
Ingredient: sugar

Recipes:
Pancake dough, cooking time: 60

Enter command: find ingredient
Ingredient: egg

Recipes:
Pancake dough, cooking time: 60
Meatballs, cooking time: 20

Enter command: find ingredient
Ingredient: gg

Recipes:

Enter command:  stop
```

Exercise submission instructions

How to see the solution

Programming exercise:
Big year (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.

This exercise is worth three one-part exercises

In this exercise you will design and implement a database for bird-watchers. The database contains birds, each of which has a name (string) and a name in Latin (string). The database also counts the observations of each bird.

The program must implement the following commands:

- `Add` - adds a bird
- `Observation` - adds an observation
- `All` - prints all birds
- `One` - prints one bird
- `Quit` - ends the program

Incorrect input must also be handled. The following is an example of the program functionality:

Sample output

```
? Add
Name: Crow
Name in Latin: Corvus Corvus
? Add
Name: Hawk
Name in Latin: Dorkus Dorkus
? Observation
Bird? Hawk
? Observation
Bird? Lion
Not a bird!
```

```
? Observation
Bird? Hawk
? All
Hawk (Dorkus Dorkus): 2 observations
Crow (Corvus Corvus): 0 observations
? One
Bird? Hawk
Hawk (Dorkus Dorkus): 2 observations
? Quit
```

NB You're free to structure your program the way you want. We only test that the `main` method of the `mainProgram` class works as shown above. You will most likely find it useful to use classes that are descriptive of the problem domain.

Exercise submission instructions

How to see the solution

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

→ 4. Conclusion

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

In this part: