Project 1 - Factory
CmpE 250, Data Structures and Algorithms, Fall 2022

Instructor: Özlem Durmaz İncel
TAs: Suzan Ece Ada, Barış Yamansavaşılar
SAs: Batuhan Çelik, Bahadır Gezer, Zeynep Buse Aydın

Due: 17/10/2022, 23:55 Sharp

1 Introduction

In Rumeli Hisarustu, a factory called CMPE250 Factory produces essential products. In the factory, individual units called holders are responsible for handling the products in the factory line. These holders are connected to each other since the product line must meet the given requirements. You are tasked with the organization of this product line.

2 Details

In this project you will be given two classes `Holder`, `Product`; and an interface, `Factory`. You will need to write two new classes `FactoryImpl`, and `Project1`.

2.1 FactoryImpl

- The `Factory` interface should be implemented. The overridden methods should work exactly as described.
- This class has three mandatory fields.

```
private Holder first;
private Holder last;
private Integer size;
```

- You can add any helper methods you like.
- Mandatory fields should be correctly modified inside the overridden methods from the `Factory` interface.

2.2 `Project1`

- The main method should be implemented here. This class is the entry point of your program.

3 Input & Output

3.1 Input

- **Add First -** Adds the product at the beginning of the factory line.

AF	product_{id}	product_{value}

- **Add Last -** Adds the product to the end of the factory line.

AL	product_{id}	product_{value}

- **Remove First -** Removes the first product from the factory line and prints it. Prints `"Factory is empty."` if the there is no product in the factory.

RF

- **Remove Last -** Removes the last product from the factory line and prints it. Prints `"Factory is empty."` if the there is no product in the factory.

RL

- **Find -** Prints the product with the given `product`$_{id}$. If the product is not in the factory line, prints `"Product not found."`

F	product_{id}

- **Update -** Updates the `value` of the product with the given `product`$_{id}$ to `product`$_{value}$. If the product is not in the factory line, prints `"Product not found."`

U	product_{id}	product_{value}

- **Get -** Prints the product in the given `index`. If the index is out of bounds, prints `"Index out of bounds."`

G	index

- **Print -** Prints the entire factory line.

P

Input	Corresponding Output Line	Explanation
P	{}	Initially the factory line is empty.
AF 1 1		`(1,1)` is added to the front.
AL 2 4		`(2,4)` is added to the end.
AL 3 9		`(3,9)` is added to the end.
RF	(1, 1)	First item is removed and printed.
RL	(3, 9)	Last item is removed and printed.
P	{(2, 4)}	Factory line is printed.
AF 4 20		`(4,20)` is added to the front.
AL 5 1		`(5,1)` is added to the end.
P	{(4, 20),(2, 4),(5, 1)}	Factory line is printed.
F 2	(2, 4)	Product with `id` 2 is found and printed.
F 10	Product not found.	Product with `id` does not exist.
U 4 13	(4, 20)	`value` of the product with `id` 4 is updated to 13. The previous product is printed.
U 1 30	Product not found.	Product with `id` 1 does not exist.
G 1	(2, 4)	Product with `index` 1 is printed.
G 3	Index out of bounds.	The factory line has 3 products so `index` 3 is out of bounds.
RF	(4, 13)	First item is removed and printed.
RF	(2, 4)	Same as above.
RF	(5, 1)	Same as above.
RF	Factory is empty.	Cannot remove from the front since the factory is empty.
RL	Factory is empty.	Same as above, but now it's the end.

Table 1: Example Input

Input file path will be given as the first program argument.

3.2 Output

- **Product -** Products will be printed in the format below.

  $(\text{product}_{id},\ \text{product}_{value})$

- **Factory Line -** Factory line will be printed in the format below.

  $\{\ \text{product}_1,\ \ \text{product}_2,\ \ \ldots\ ,\ \ \text{product}_n\ \}$

Output file path will be given as the second program argument.

4 Submission

Your code will be graded automatically. So it's important that you follow the submission instructions. First, all 5 of your source files, and nothing more, should be collected under `Project1/src`. Then, you should zip the `Project1` folder and rename it to `p1_<student`$_{id}$`>.zip`. This zip file will be submitted through moodle.

Please make sure that your program compiles and runs from the terminal line with the `javac` and `java` commands respectively. The target version is 17.

5 Grading

Grading of this project is based on the automatic compilation and run and the success of your code in test cases. If your code compiles and runs with no error, then you will get 10% of the project grade. 40% of the grade will come from unit tests. We will test each method implementation for the methods in the `Factory` interface. Each method implementation will have equal weight. The rest of your grade will be the sum of collected points from each test case. Each test case will have equal weight. Maximum project grade is 100%.

6 Warnings

- All source codes are checked automatically for similarity with other submissions and exercises from previous years. Make sure you write and submit your own code. Any sign of cheating will be penalized by at least -50 points at first attempt and F grade in case of recurrence.
- Make sure you document your code with necessary inline comments and use meaningful variable names. Do not over-comment, or make your variable names unnecessarily long. This is very important for partial grading.
- Do not make changes in `Holder.java`, `Product.java`, and `Factory.java`.
- Do not add any files to the source code except `Holder.java`, `Product.java`, `Factory.java`, `FactoryImpl.java`, and `Project1.java`.
- Make sure that each method in the `Factory` interface does the operations it is supposed to do. These methods will be individually unit tested. For example, if there is only one product in the factory and either `RF` or `RL` are called, then both `first` and `last` fields should be set to `null`.
- All method and field names must be exact.
- Make sure that the white-spaces in your output is correct. You can disregard the ones at the end of the line.