FRÓÐSKAPARSETUR FØROYA
UNIVERSITY OF THE FAROE ISLANDS

Design Patterns with Java (5041.24)

Assignment 02 – Decorator & Factory Patterns

Deadline 24. jan. 2025 23:59:59

01: Profiling with Decorators

Description

This exercise uses the Decorator Pattern for adding a timing aspect to an existing component whose source code we have no access to.

Download the Starter Code from the Assignment02.zip the starter code is **ProfilingWithDecorators.** The code in the "designpatterns5041.assignment02.library" package can NOT be modified.

It seems that the library computes rather slowly when invoked as in main.java. We would very much like to instrument the **ComputeOperation** class with code to measure the execution time of the **compute()** method.

Unfortunately, this is an external library for which we cannot modify the source code, so we need to figure out an alternate way of decorating the existing library.

Task

1. Define an appropriate abstract ComputeOperationDecorator class for ComputeOperation .

2. Write a concrete Timing decorator deriving from ComputeOperationDecorator.
    - You can use System.currentTimeMillis(); to get current time (long timeElapsed = finish – start)

3. Test your implementation by modifying Main.java accordingly.

When you run your complete solution the last part of the output produced should be like the following:

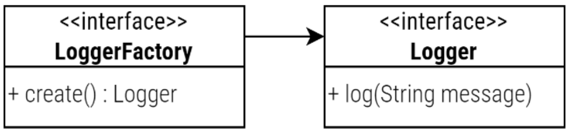
```
...(previous numbers omitted)
74
76
78
80
82
84
86
88
90
92
94
96
98
Execution time was 5344ms
```

4. Using DrawIO create a class diagram that shows the classes in the project and how they are associated. Try and show where the decorator pattern is applied. Use the Cheate Sheet in moodel for guideance.

02: The Logger Factory

Description

You are building a **ApiService** that calls an API to do some work. To be sure the API does what it is supposed to you have added logging to the **ApiService**. The logger you are using implements a Logger interface and is created using a **LoggerFactory**.



Download the Starter Code from the Assignment02.zip the starter code is **TheLoggerFactory.**

Task

1. Implement the **ConsoleLogger** and the **ConsoleLoggerFactory**. Then run the main.java and you should get an output like this:

```
getStuff1 called
Doing work
Work done
getStuff1 is done and took 317ms
getStuff2 called
Doing work
Work done
getStuff2 is done and took 1139ms
getStuff3 called
Doing work
Work done
getStuff3 is done and took 1940ms
```

2. You have discovered that one of your rivals has started stealing your logs to be able to figure out how you've built such an amazing piece of software. To stop them you've decided to RickRoll them by building a **RickRollingLogger**.
   This **RickRollingLogger** does not log what you send to it, but instead just outputs a random line of the chorus of the song "Never Gonna Give You Up" by Rick Astley.
   https://www.azlyrics.com/lyrics/rickastley/nevergonnagiveyouup.html
   Your assignment is to build the **RickRollingLogger** and the **RickRollingLoggerFactory**. Then exchange the **ConsoleLoggerFactory** out with the **RickRollingLoggerFactory**. You should get an output like this:

```
Never gonna say goodbye
Never gonna tell a lie and hurt you
Never gonna let you down
Never gonna make you cry
Never gonna give you up
Never gonna run around and desert you
Never gonna let you down
Never gonna give you up
Never gonna say goodbye
Never gonna let you down
Never gonna say goodbye
Never gonna say goodbye
```

5. Using DrawIO create the ClassDiagram of your setup. This diagram should resemble the Abstract Factory Pattern Diagram as much as possible. Use the Class Diagram CheatSheet on moodle for help.

Submission

The assignment solution must be submitted (pushed) in your assigned student repo in the folder Assignments/Assignment**{Number}**, which you will find in the course's Bitbucket Team.

The program code must contain a functional program with no compilation errors. The codes should be well structured, understandable and easy to read which include:

- Follow naming conventions. (e.g., descriptive names for class, method, variable and others).
- Use Clear and Consistent Formatting (e.g., correct indentations, spacing and bracket placement)
- Be as self documenting as possible, so only add comments if needed. No need to add java doc for every class and method.

Perform a formal submission in Moodle, via the intended submission box, by submitting a text file with your repo url in it!

NB! Changes made in the repo after the deadline will not be taken into consideration. Please contact the teach if submission deadline has elapsed.