Design Patterns - TD2

TD2 initial code

This is a template for the students' assignments.

Course material: icon::mobile[] icon::tablet[] icon::[]laptop http://bit.ly/jmb-cpoa

Assignment info

LAST NAME

BRUEL

First Name

Jean-Michel

Group #

- [x] Teachers
- [ ] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [ ] Innopolis

Requirements

You'll need:

- [x] A GitHub account
- [ ] A Git Bash terminal (if you use Window$)

Try the following command in your terminal to check your `git` environment:

```
git config --global -l
```

Initial tasks

- [x] Click on the Github Classroom link proemptyd by your teacher (in fact, this should be done if you read this).
- [ ] Clone on your machine the Github project generated by Github Classroom.

- ☐ Modify the README file to add your last name, first name and group number.
- ☐ Commit and push using the following message:

commit/push

```
fix #0 Initial task done
```

In the following, every time you'll see à `fix #⋯` text, make sure all your files are committed, and then push your modifications in the distant repo, making sure you used the corresponding message (`fix #⋯`) in one of the `commit` messages.

- If you want to check that you're really ready for `fix #0`, you can run the command in your shell: `make check`.
- If you want to list the ToDos of the day, run `make todos`.

1. The "Chocolate Factory" app

This TD exercise is inspired from the excellent book: "Head First: Design Pattern. Bert Bates, Eric Freeman, Elisabeth Freeman, Kathy Sierra. Editions O'Reilly. 2005."

1.1. Existing application

You're a developer in a chocolate factory simulator company. Chocolate factories use boilers supervised by controllers.

The job of the boiler is to take in chocolate and milk, bring them to a boil, and then pass them on to the next phase of making chocolate bars.

Here is the controller class you have developed:

The `ChocolateBoiler` class

```
public class ChocolateBoiler {
    private boolean empty;
    private boolean boiled;

    public ChocolateBoiler() {
        empty = true;
        boiled = false;
    }

    public void fill() {
        if (isEmpty()) {
            empty = false;
            boiled = false;
            // fill the boiler actions
        }
    }

    public void drain() {
        if (!isEmpty() && isBoiled()) {
            // drain the boiler
            empty = true;
        }
    }

    public void boil() {
        if (!isEmpty() && !isBoiled()) {
            // boil the content
            boiled = true;
        }
    }

    public boolean isEmpty() { return empty;}

    public boolean isBoiled() { return boiled;}
}
```

QUESTION

1. What are the attributes `empty` and `boiled` used for?

You're making horrible nightmares (really?) that you drain in a ocean of chocolate.

QUESTION

1. What could be wrong if two different objects manipulate the same physical boiler?

   To test this scenario, execute the JUnit test `twoChocolateBoilersMightBlowTheFactory`.

2. What should we guaranty to avoid this problem?
3. Find examples where it is important to make sure there is only one instance of certain classes.

commit/push

```
fix #1.1 Initial code
```

1.2. Solution 1

You remember your first coding experience in Java about class variables you propose to use an instance counter to solve the problem.

QUESTION

You first modify the constructor so that it only works when the instance counter is equal to 0. What is wrong in the following excerpt:

ChocolateBoilerCpt.java

```
public class ChocolateBoilerCpt {
    private boolean empty;
    private boolean boiled;
    private static int nbInstance = 0;

    public ChocolateBoilerCpt() {
        empty = true;
        boiled = false;
        if (nbInstance == 0) {
            nbInstance = 1;
            return this;
        }
        else {
            return null;
        }
...
```

1.3. Solution 2

You change your strategy and remember seeing this kind of code:

An idea!

```
public class MyClasse {
    private MyClasse() {...}
}
```

QUESTION

1. Is it allowed to have a private constructor?
2. How can we create an instance under those circumstances? Don't we end-up simply with an unusable class?

TODO*:*

- ▢ Complete the following code to solve the problem:

ChocolateBoilerSafe

```
public class ChocolateBoilerSafe {
    private boolean empty;
    private boolean boiled;
    ...
    ...

            ChocolateBoiler() {
        ...
        ...
        }

    ...
    ...
    ...
    ...

    public void fill() {
        if (isEmpty()) {
            empty = false;
            boiled = false;
            // ... }
        }
        // ...
}
```

- ▢ Write a test that use this class
- ▢ Commit&Push when everything is ready

commit/push

```
fix #1.3 Solution with a private constructor
```

1.4. It’s not over!

It looks like the Chocolate Boiler still has a problem: the `ChocolateBoiler`s `fill()` method was able to start filling the boiler even though a batch of milk and chocolate was already boiling! What happened!?

QUESTION

1. We have two threads, each executing this code. Your job is to play the JVM and determine whether there is a case in which two threads might get ahold of different boiler objects. Use the code magnets to help you study how the code might interleave to create two boiler objects:

Thread 1	Thread 2	Valeur de `uniqueInstance`

Bloc 1

```
public static ChocolateBoilerSafe getInstance() {
```

Bloc 2

```
if (uniqueInstance == null) {
```

Bloc 3

```
uniqueInstance = new ChocolateBoilerSafe();
```

Bloc 4

```
        }
```

Bloc 5

```
        return uniqueInstance;
```

Bloc 6

```
    }
```

1.5. Solution to the multithreading

QUESTION

1. Propose a simple solution to this problem.

1.6. Problem of the solution!!

QUESTION

1. How many times this mechanism is used?
2. Hence, what do you think of this solution?
3. Propose a solution where the instance is created at the "beginning" rather than on demand.

2. Singleton

Congratulations, you manipulated your 2nd pattern: **Singleton**.

Design pattern: **Singleton**

Singleton guaranties that a classe has only one instance and provides a global access to this instance.

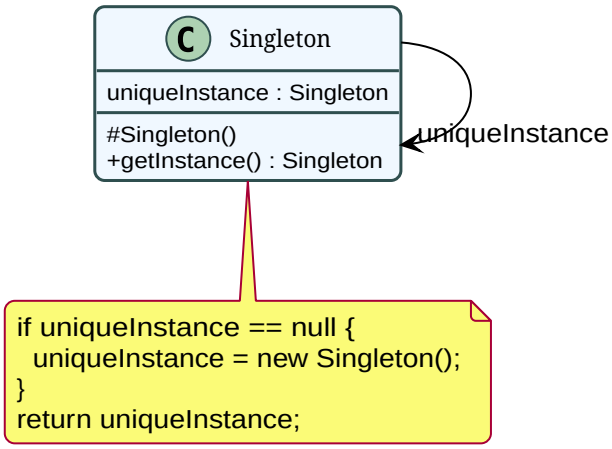

Figure 1. UML model of the Singleton pattern

Appendix A: Still hungry?...

QUESTION

1. What is the difference between a singleton and a global variable?
2. How would you test the Singleton pattern?

TODO:

- ▢ Add some tests to your repo to test the effectiveness of the pattern
- ▢ Commit&Push when everything is ready

commit/push

```
fix #1.3 Solution with a private constructor
```

Contributors

- Jean-Michel Bruel

About...

Baked with {asciidoctorlink} (version 2.0.11) from 'Dan Allen', based on {asciidoc}. 'Licence Creative Commons'.  licence Creative Commons Paternité - Partage à l'Identique 3.0 non transposé.