Design Patterns - TP3

TP3 initial code

This is a template for the students' assignments.

> Course material: 📱📱💻 http://bit.ly/jmb-cpoa

Assignment info

LAST NAME
: DOE

First Name
: John

Group #

- [ ] Teachers
- [ ] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [X] 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 provided 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`.

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."

The *Factory* pattern

QUESTION

- Fully implement the Pizzeria application so that:
  - it implements the Abstract Factory
  - it implements the Singleton (for the factory)
  - the test program below will produce the result below

Start by writing this program and use *QuickFix* to "generate" the code as much as possible.

Rendus attendus

ToDo

- ▢ a `pom.xml` that runs the tests of your application
- ▢ a `build.gradle` that runs the tests of your application
- ▢ the class diagram of your application, in a file named `TP3.plantuml`, placed in a `docs` folder in your repo.

We will use the following pizzas model:

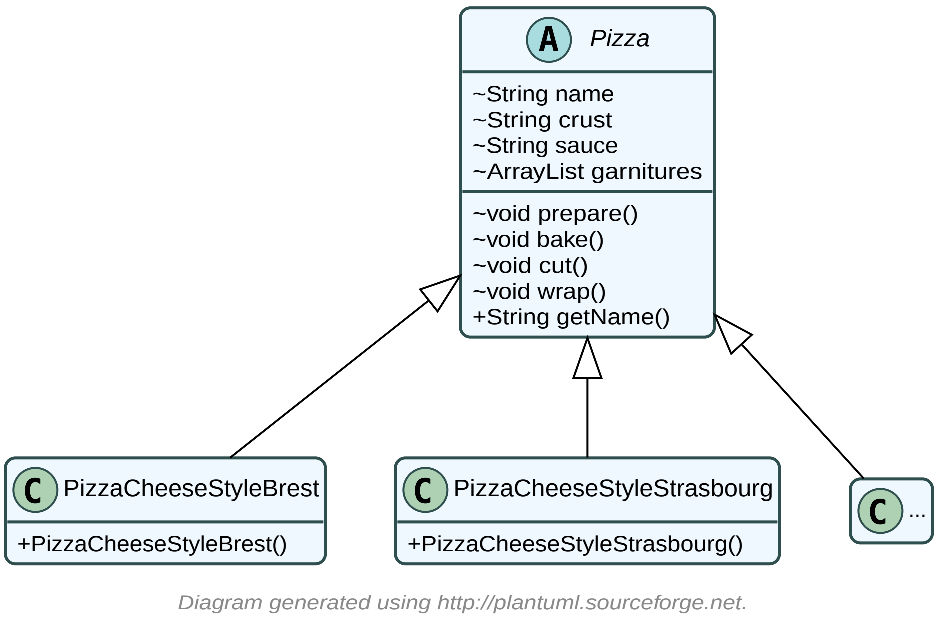

Figure 1. Class diagram of the Pizzas

Testing program

```
public class PizzaTestDrive {
    public static void main(String[] args) {
        Pizzeria shopFromBrest = PizzeriaFactory.getInstance().create("Brest");
        Pizzeria shopFromStrasbourg = PizzeriaFactory.getInstance().create
("Strasbourg");

        Pizza pizza = shopFromBrest.orderPizza("cheese");
        System.out.println("JMB has ordered a " + pizza.getName() + "\n");

        pizza = shopFromStrasbourg.orderPizza("cheese");
        System.out.println("JMI has ordered a " + pizza.getName() + "\n");
    }
}
```

Execution results

```
$ java -jar target/pizzeria-1.0.jar
Preparation of Pizza with Brest style sauce and cheese
Spread the pizza dough...
Add the sauce...
Add the garnitures:
 Parmigiano reggiano
Bake 25 minutes at 180 degrees
Cut the pizza in triangles
Put the pizza in the official box
JMB has ordered a Pizza with Brest style sauce and cheese

Preparation of Pizza Strasbourg style with cheese
Spread the pizza dough...
Add the sauce...
Add the garnitures:
 Mozzarella
Bake 25 minutes at 180 degrees
Cut in square portions
Put the pizza in the official box
JMI has ordered a Pizza Strasbourg style with cheese
```

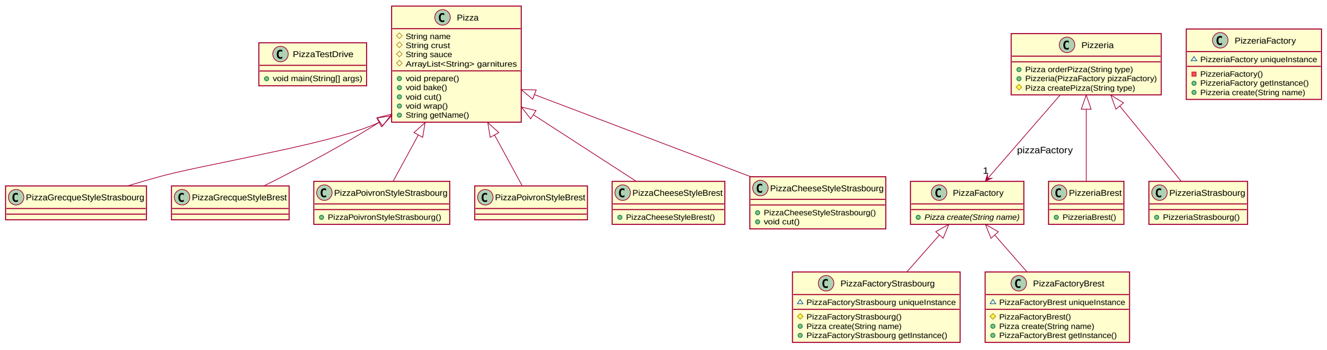
PizzaTestDrive
void main(String[] args)
Pizza
String name
String crust
String sauce
ArrayList<String> garnitures
void prepare()
void bake()
void cut()
void wrap()
String getName()
PizzaGrecqueStyleStrasbourg
PizzaGrecqueStyleBrest
PizzaPoivronStyleStrasbourg
PizzaPoivronStyleStrasbourg()
PizzaPoivronStyleBrest
PizzaCheeseStyleBrest
PizzaCheeseStyleBrest()
PizzaCheeseStyleStrasbourg
PizzaCheeseStyleStrasbourg()
void cut()
Pizzeria
Pizza orderPizza(String type)
Pizzeria(PizzaFactory pizzaFactory)
Pizza createPizza(String type)
PizzeriaFactory
PizzeriaFactory uniqueInstance
PizzeriaFactory()
PizzeriaFactory getInstance()
Pizzeria create(String name)
pizzaFactory
1
PizzaFactory
Pizza create(String name)
PizzeriaBrest
PizzeriaBrest()
PizzeriaStrasbourg
PizzeriaStrasbourg()
PizzaFactoryStrasbourg
PizzaFactoryStrasbourg uniqueInstance
PizzaFactoryStrasbourg()
Pizza create(String name)
PizzaFactoryStrasbourg getInstance()
PizzaFactoryBrest
PizzaFactoryBrest uniqueInstance
PizzaFactoryBrest()
Pizza create(String name)
PizzaFactoryBrest getInstance()

Figure 2. expected Structure

This assessment is graded. The autograding will run the tests via `gradle test` and `maven test`, as well as `test0` and the test of the model. This will constitute 80%of your grade. The remaining 20% will be evaluated by your TA and will focus on the tests (number and quality).

commit/push

```
fix #All: Completed all duties
```

Contributors

- Jean-Michel Bruel

About…