OOP Design Patterns
Practice Assignment

Ex. 1 Singleton

Implement a Logger Singleton Class, that should allow you to log, ie. print, any message from anywhere inside your app. It should have a log method with the signature: void log(String msg).

Add the class to one of your OOP Exercises you implemented last session, and use it more than once in different classes.

Before you continue!

For the rest of the exercises, you need to get some prepared code to work on.
Open your Git terminal, enter this command to clone the code repo:

```
git clone https://github.com/mnzaki/guc-berlin-met-bootcamp-2015
```

You will find all the code under the repo folder inside design_patterns_code folder.
For each of the following exercises, create a new java project in Eclipse, then add all the files from the exercise folder to the project to work on them.
** These exercises are adapted from the book “Head First, Design Patterns”.

Ex. 2 Factory

Under the folder “design_patterns_code/factory_pattern”, you will find a set of classes describing a problem of a Pizza store.

- There are four types of pizza extending the ***Abstract*** *Pizza* class.
- The class *SimplePizzaFactory* should be able to create a Pizza object of the right type to correspond with the type’s correct subclass.
- The *PizzaStore* class utilizes the factory to create a pizza, prepare it and sell it.
- Skim through the code, understand how it should work, and find the **TODO**s comments, follow them to edit the code so that it works.
- If you think you are done, run the main method inside *PizzaTestDrive* class, make sure you get a delicious hot pizza!

** Make sure you understood how we utilized the Factory design pattern, and which code would have been repeated if we did not use it.

Ex. 3 Iterator

Under the folder "design_patterns_code/iterator_pattern", you will find a set of classes describing a problem of a restaurant.

- The *Menu interface makes sure that each menu has an iterator to list it's items, MenuItem.*
- The *DinnerMenu* class implement the *Menu* interface, you may have other menus, breakfast, drinks, etc.
- The *DinnerMenuIterator* class implements java *Iterator* Interface. Notice that it returns general java class *Object*, that should be casted when returned.
- The *Waitress* iterates a given menu and print it to customers.
- Skim through the code, understand how it should work, and find the **TODO**s comments, follow them to edit the code so that it works.
- If you think you are done, run the main method inside *MenuTestDrive* class, make sure you get a delicious hot pizza!

** Make sure you understood how we utilized the *Iterator* design pattern, and how we can change the custom iterator to iterate in any way we like rather than a linear standard one given by any Array.

Ex. 4 Template

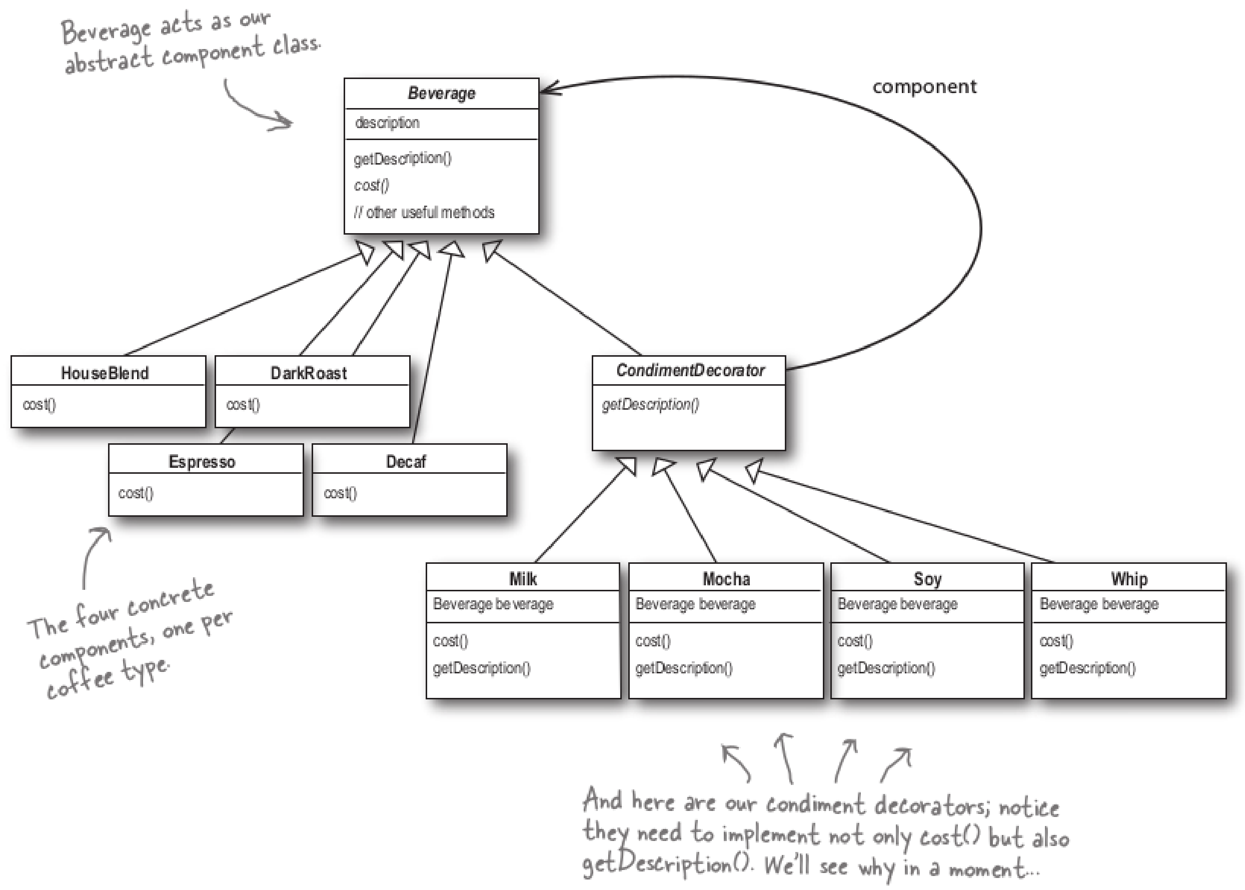
Under the folder "design_patterns_code/template_pattern", you will find a set of classes describing a problem of a Coffeeshop.

- There are two types of caffeine beverages extending the ***Abstract*** *CaffeineBeverage* class.
- *CaffeineBeverage* class has *prepareRecipe()* method with the algorithm to prepare a hot beverage, with some of them implemented and others are left **Abstract**.
- Skim through the code, understand how it should work, and find the **TODO**s comments, follow them to edit the code so that it works.
- If you think you are done, run the main method inside *BeverageTestDrive* class.

** Make sure you understood how we utilized the Template design pattern, and which code would have been repeated if we did not use it.

Ex. 5 Decorator

Under the folder “design_patterns_code/decorator_pattern”, you will find a set of classes describing a problem of another Coffeeshop. The diagram explains them.



- All the classes are subclasses of *Beverage* class, however CondimentDecorator subclasses all should have a Beverage variable to save the last state of the Beverage before adding the a condiment to it.
- Skim through the code, understand how it should work, and find the **TODO**s comments in the *Soy* class which should look like the *Milk* class, follow them to edit the code so that it works.
- If you think you are done, take a look at and run the main method inside *StarbuzzCoffee* class.

** Make sure you understood how we utilized the Decorator design pattern, and how many classes would have been needed instead, one for HouseblendWithMilk, HouseblendWithSoy, …, DecafWithMilk, … etc.