Exercises: Design Patterns

This document defines the exercises for the "Java Advanced" course @ Software University.

1. Singleton Design Pattern

https://en.wikipedia.org/wiki/Singleton_pattern

The singleton pattern is a software design pattern that restricts the instantiation of a class to one "single" instance. This is useful when exactly one object is needed to coordinate actions across the system.

2. Prototype Design Pattern

https://en.wikipedia.org/wiki/Prototype_pattern

The prototype pattern is a creational design pattern in software development. It is used when the type of objects to create is determined by a prototypical instance, which is cloned to produce new objects.

3. Factory Design Pattern

https://en.wikipedia.org/wiki/Factory_method_pattern

In the Factory pattern, we create an object without exposing the creation logic to the client and refer to a newly created object using a common interface.

4. Builder Design Pattern

https://en.wikipedia.org/wiki/Builder_pattern

It is used to construct a complex object step by step and the final step will return the object. The process of constructing an object should be generic so that it can be used to create different representations of the same object.

5. Command Design Pattern

https://en.wikipedia.org/wiki/Command_pattern

In the Command pattern, there is a Command object that encapsulates a request by binding together a set of actions on a specific receiver. It does so by exposing just one method `execute()` that causes some actions to be invoked on the receiver.

6. Strategy Design Pattern

https://en.wikipedia.org/wiki/Strategy_pattern

Strategy is a behavioral design pattern that lets you define a family of algorithms, but each of them into a separate class, and make their objects interchangeable.

Exercises

1. Singleton

Create a **Hashcode** class that has a **String point** field and its only instance. There is also a **constructor** that initializes the point, as well as a **getInstance()** method based on the **Singleton** Design Pattern. Create a **HashcodeDemo** class in which to test the Hashcode class method. You can take the real hashcode with the **.hashCode()** method.

2. Employee Record

You are given the **EmployeeRecord** class, which contains **int id**, **String name**, **String designation**, **double salary,** and **String address**, as well as two constructors that you do not need to change. Add a **showRecord()** method to return the data. Your task, using a **Prototype** Design Pattern, is to create a **Prototype** interface that has a single **getClone()** method. The **EmployeeRecord** class must **implement** the interface. Create a **PrototypeDemo** class in which you will test your logic.

3. Cake Factory

The **CakeInterface** and the **abstract class Cake** that implements it are given. Your task is to create 4 types of different cakes – **BiscuitCake**, **ChocolateCake**, **SpinachCake,** and **WhiteCake**, that use the methods of the **Cake** class. Then, with the help of **Factory** Design Pattern, create a class **CakeFactory**, in there will be a single method - **createCake**, according to the 4 parameters of the cake and the type (**Chocolate**, **White**, **Spinach**, **Biscuit**). The last thing is to make a **Pastryshop** class that contains an **instance** of **CakeFactory**. The idea of this class is to **place an order** - **create** the cake**, use the three methods** and **return** it.

4. Phonebook

Given the **Contact** and **Phonebook** classes, your task is to familiarize yourself with their structure, then you need to refactor the **Contact** class relying on the **Builder** Design Pattern.

Next, write a program to execute the following commands until "**END**" is received:

"**INFO**" - Reads a name from the console and prints the complete contact information.

"**DELETE**" - Reads a name from the console and deletes the contact from the phone book.

"**PHONEBOOK**" - Prints the complete information of all contacts.

"**END**" - The program ends its operation.