Object Orientation with Design Patterns

Stock Market Simulator

Student Name: Diogo Gomes de Oliveira
Student Number: 2016277
Student Name: Flavia Aparecida da Silva
Student Number: 2016265
Student Name: Jordana Rodrigues
Student Number: 2016363

After researched we had decided to use the follow design patterns:

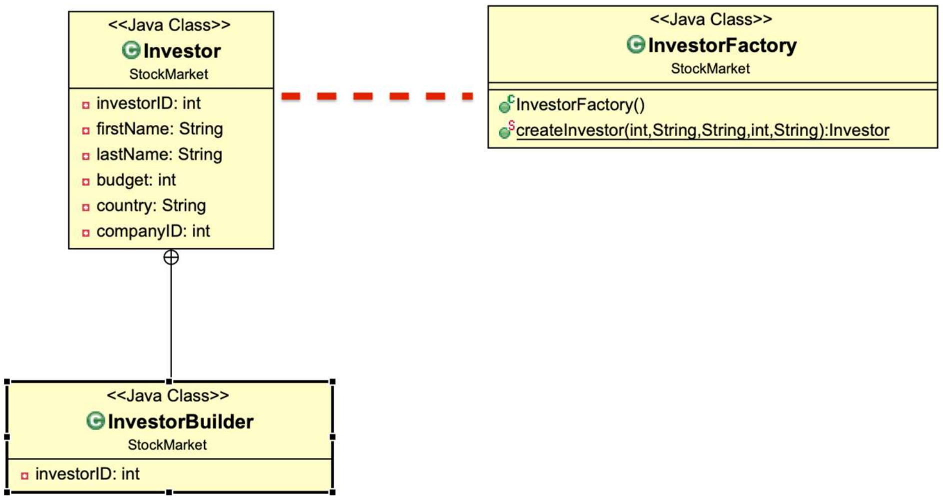
Factory Pattern:

This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

In Factory pattern, the method can deal with the problem of creating objects without having to specify the extract class just using a common interface, it does not have tight coupling with the class hierarchy of the library.

We can use when we have a super class with multiple sub-classes and based on input, we need to return one of the sub-classes. The responsibility of instantiation of a class is taken out from the client program.

In our program we do not have now different sub-classes of the Investors class, but we decided to implement this pattern so if an alteration is made in the future and the client what's different sub-classes of Investors, this part is already done and the implementation will be easier.

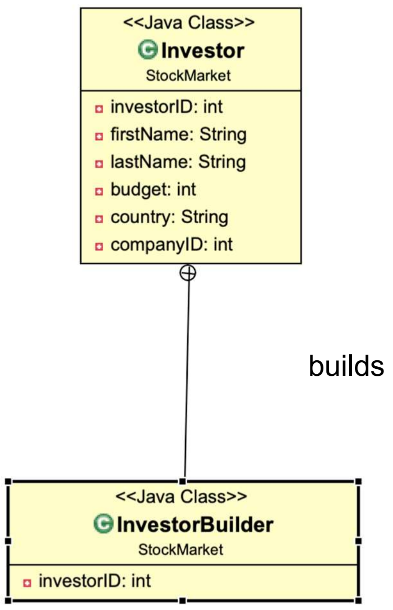


Builder Pattern:

Builder patter can build complex objects using simple objects using a step by step approach. It is one of the creational patterns it solves problem related to object creation and provides one of the best ways to create an object.

The process of constructing an object should be generic so that it can be used to create different representations of the same object, providing a high readable method calls, minimizing the number of parameters in constructor.

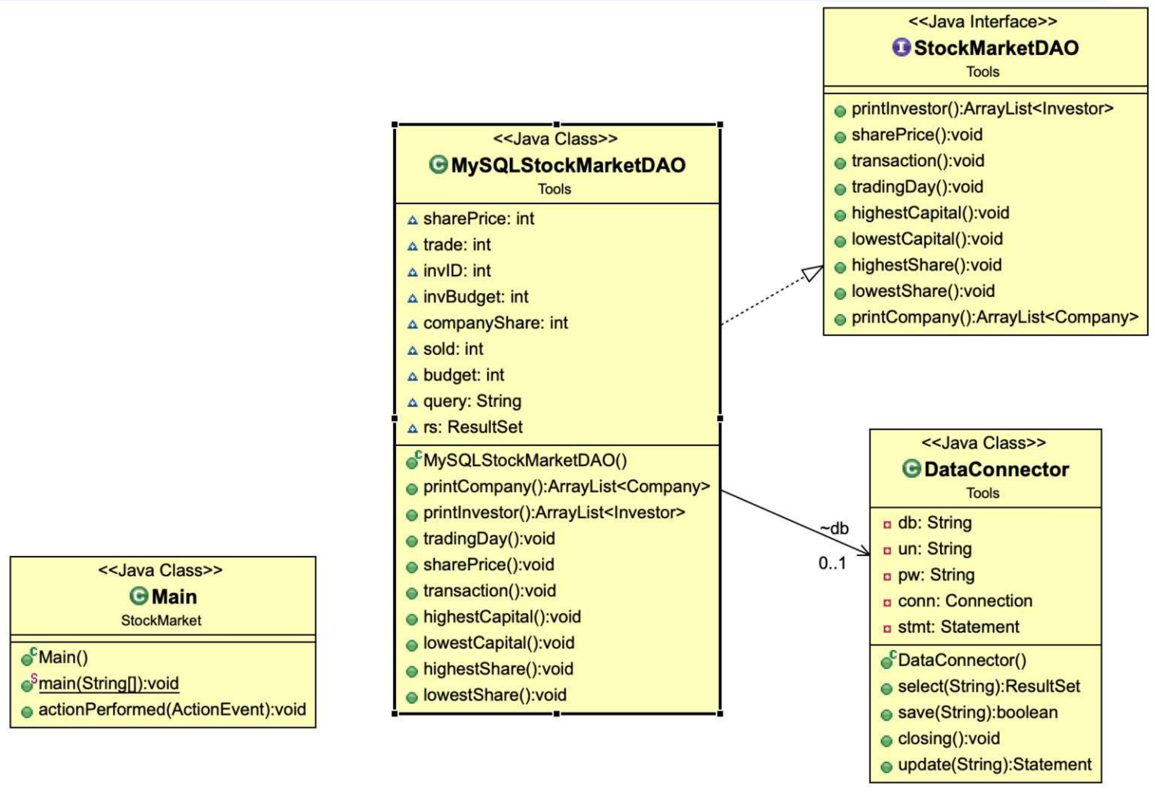
We had decided to use this pattern because it provides a clear separation between the construction and representation of the object, with that in mind we were able to better control the contraction process.



Data Access Object Pattern:

It used to separate the data persistence logic in a separate layer

Data Access Object Patter it is a structural pattern that allows isolation of the application from the persistence layer (in our case the Database). This pattern was chosen because we wanted to hide from the application all the complexities related to the data access. DAO allows both layers to evolve separately without knowing anything about each other.



References:

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

https://www.tutorialspoint.com/design_pattern/factory_pattern.htm.

https://medium.com/@ajinkyabadve/builder-design-patterns-in-java-1ffb12648850.

https://www.geeksforgeeks.org/builder-design-pattern/

https://www.baeldung.com/java-dao-pattern.

https://www.tutorialspoint.com/design_pattern/data_access_object_pattern.htm.