WEEK 2

ONDERWERPEN

- design principles
- design patterns
- strategy, observer, decorator and factory pattern

OPGAVE 1: ENCRYPTIE EN HASHING

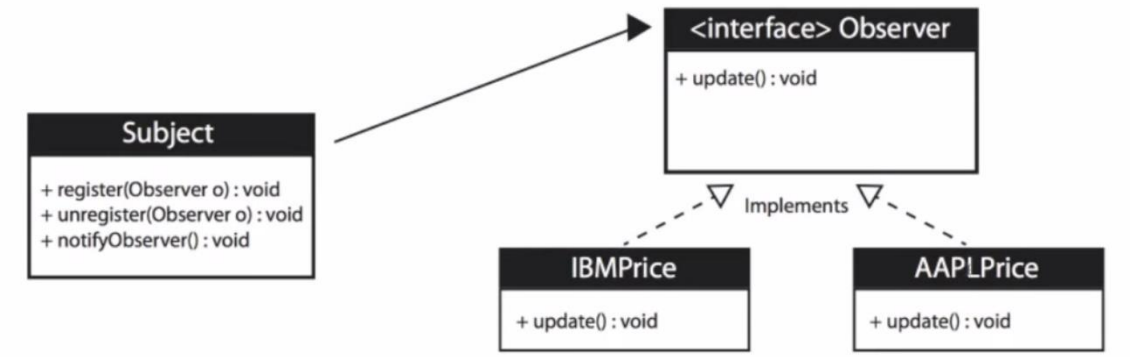
Ontwerp een klasse diagram voor de volgende applicatie, waarbij je uiteraard gebruik maakt van een of meer design patterns.

Een computer heeft verschillende netwerk interfaces: Ethernet, Wifi, Bluetooth en seriële poort (RS232). Op al deze interfaces (poorten) kunnen berichten binnekomen. De applicatie moet deze berichten in het geheugen plaatsen en daarna opslaan in een file.

Bij het opslaan heeft de gebruiker steeds een keuze: het bericht versleutelen of een hash-waarde genereren voor dit bericht. Bij het versleutelen zijn er verschillende mogelijkheden: de gebruiker kan kiezen uit AES, RSA of Blowfish. Ook voor het hash-algoritme heeft de gebruiker een keuze: hij kan kiezen uit MD5 of SHA-256. In je uitwerking gaat het om het gebruik van patterns, overige details (GUI, filehandling, etc.) hoef je niet uit te werken.

OPGAVE 2: STOCK EXCHANGE

In the second lecture of week 2 the Observer Pattern was discussed, using the simulation of a stock exchange as an example. You can find some Java source files on Blackboard.

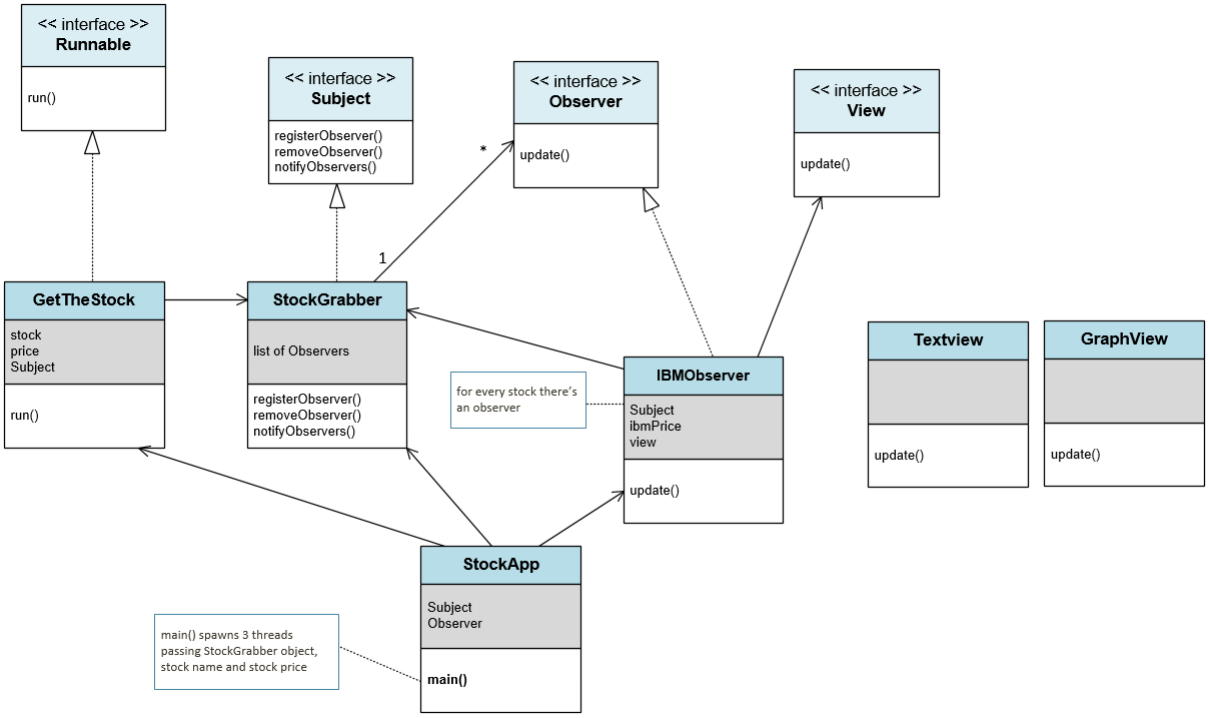


Make an application that simulates the stock exchange, using the class diagram shown below and with the following requirements:

- all stock prices must be shown in a GUI
- use JavaFX as GUI framework
- stock listings must be presented in two different views: text and graph
- the user can switch between these two views dynamically
- of course we want to see historical data as well

- in the future more view types may be added
- for every stock (at least 3) there must be a separate observer
- generation of stock prices is done multi-threaded
- every 2 seconds a thread generates new prices for the shares of a single company, e.g. something like (you are allowed to adapt this):

```
double randNum = (Math.random() * (.1)) - .05;
DecimalFormat df = new DecimalFormat("#.##");
price = Double.valueOf(df.format((price + randNum)));
```



OPGAVE 3: PIZZA STORE

In the lecture week 2-2 the Decorator Pattern was discussed, using the simulation of a pizza store as an example.

On Blackboard you'll find code for a pizza store. Try running this code.

Our pizza store now introduces different sizes of pizza's: S, M and L. Therefore they have added methods setSize() and getSize() to the Pizza class. Of course, toppings now have different prices too!

Alter the decorator classes to handle this change in requirements.

OPGAVE 4: FACTORY PATTERN

Make a - preferably original - application of either a factory method pattern or an abstract factory pattern.

Draw a class diagram of your implementation.

Motivate why the used pattern is appropriate for this application/problem.