TP 6: Design Patterns, Refactoring, Code Smells

Exercise 1

Question 1:

Clean code should be understood by every programmer. The programmer could understand and follow the code with ease. Clean code should me minimalist and don't be duplicated. The same parts of code should not be at different places. Clean code should be as simple as possible. It should have the minimal number of classes. Finally, the tests that are done on clean code should all passed and covered 100% of the code.
The role played by refactoring one's code is to get rid of the technical debt or the potential one, enhance the quality of the code, to make it more understandable and easier to follow for the next programmers.

Question 2:

Yes, I think you can "over-refactor" and do too much. When refactoring, we think we are doing a better code whereas we make it less complete. Sometimes, by refactoring too much, the code can become more complex than before.

Question 3:

A code smell is a coding bad habit. It can indicate problems and future problems. They can represent a deeper problem in the code. Therefore, a code smell can be problematic during the refactoring part. We should bother because code smells can create problems during the refactoring part. Code smells can hide problems that we don't or can't see at the moment.

Question 4:

A few code smells that can be identify in the original GildedRose are:

- Bloaters (Long method): the method updateQuality() is way too long.
- Dispensables (duplicate code in a method): in the method updateQuality(), some statements are written multiple times, it's the case for the "if" statements on quality < 50 and quality > 0 which implies the same changes (quality++ or quality--). We could create methods to add and decrease quality taking these two conditions.

Question 5:

The refactoring techniques that could be used in the GildedRose are:

- Composing methods with methods extraction: the method updateQuality() could be extract in multiples methods.
- Simplifying Conditional Expressions: we can also simplifying the conditional expressions in the updateQuality() method.

Exercise 2:

Question 1:

Design patter are frameworks that can be customized and used to solve coding problem. It represents coding good habits.

Question 1.a:

We should use design pattern when we are having known coding problems. We should use design pattern when we know what our problem is and what is the design pattern we need to use.

Question 1.b:

We should not use design pattern if our code doesn't contain existing coding and software problems. We should not use a design pattern when we don't know what the problem in our code is.

Question 2:

This is a good idea because we don't use the Pizza class to build a pizza but we use the builder. Moreover, it's easier to add toppings in the future.

Question 3:

I implemented a decorator pattern on phones. I have a phone with different properties. Phone can be of 2 types: PhoneHigh (high quality phone) and PhoneLow (low quality phone). Then, I have a Decorator which follows Phone. Finally, I have two concrete decorator that come from Decorator.
The limitations that can occur are about the parent class because a subclass can only have one parent class.

Question 4:

I implement the template method from the behavioral patterns. I implemented the construction of computers that need to get a keyboard, electronic parts and a screen which can be of 2 types, either a normal screen or a touch screen.