

# Design Patterns - TD4

## TD4 initial code

This is a template for the students' assignments.



Course material:    <http://bit.ly/jmb-cpoa>

## Assignment info

**LAST NAME**

BRUEL

**First Name**

Jean-Michel

**Group #**

☒ Teachers

☐ 1

☐ 2

☐ 3

☐ 4

☐ Innopolis

## Requirements

You'll need:

☒ A [GitHub](#) account

☐ A [Git Bash](#) terminal (if you use Window\$)



Try the following command in your terminal to check your **git** environment:

```
git config --global -l
```

## Initial tasks

☒ Click on the Github Classroom link provided by your teacher (in fact, this should be done if you read this).

☐ Clone on your machine the Github project generated by Github Classroom.

- ❑ Modify the README file to add your last name, first name and group number.
- ❑ Commit and push using the following message:

🔄 `commit/push`

```
fix #0 Initial task done
```



In the following, every time you'll see a `fix #...` text, make sure all your files are committed, and then push your modifications in the distant repo, making sure you used the corresponding message (`fix #...`) in one of the `commit` messages.



- If you want to check that you're really ready for `fix #0`, you can run the command in your shell: `make check`.
- If you want to list the Todos of the day, run `make todos`.

This TD exercise is inspired from the excellent [book](#): "Head First: Design Pattern. Bert Bates, Eric Freeman, Elisabeth Freeman, Kathy Sierra. Editions O'Reilly. 2005."



# 1. Differences between dependency, association, composition, aggregation

Considering the following class diagram:

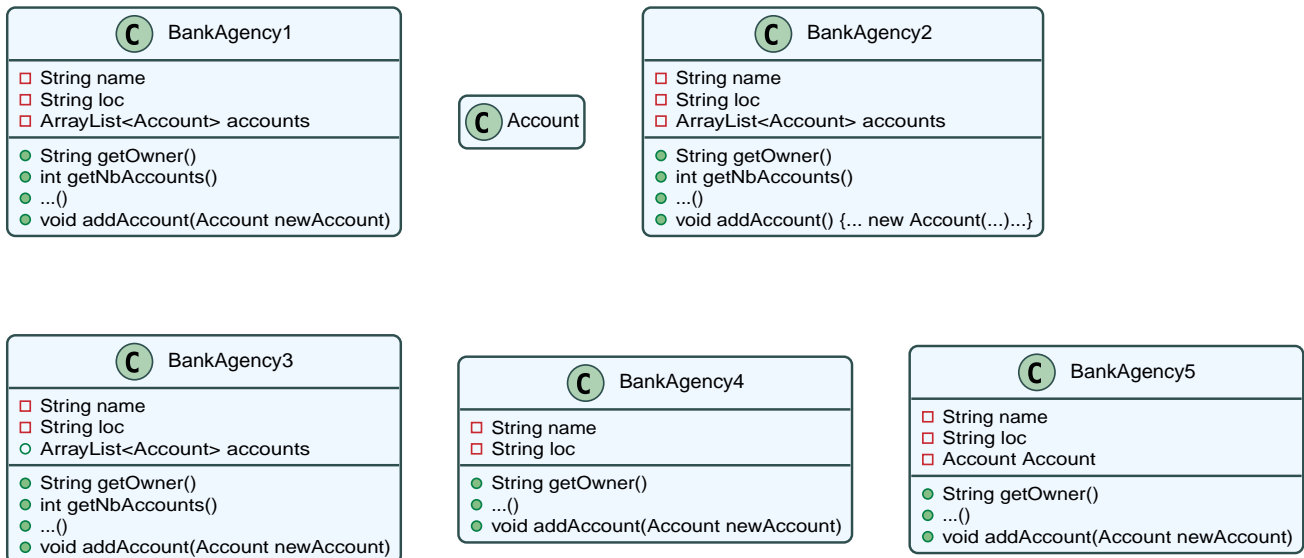


Diagram generated using <http://plantuml.sourceforge.net>.

Figure 1. Partial Class Diagram



### QUESTION

Complete the diagram by adding pertinent relations (dependency, association, composition, aggregation) between classes.

 [commit/push](#)

fix #1.0 Completed class diagram

## 2. Design Pattern

## QUESTION

For each class diagram below (representing known Design Pattern), provide:

- the name of the Design Pattern,
- the missing relationships.

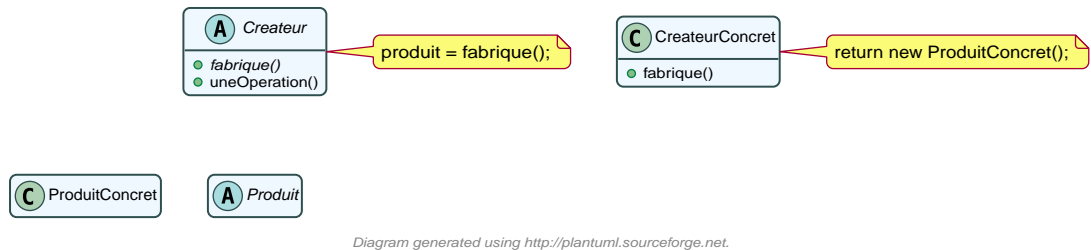


Figure 2. Design Pattern: ...

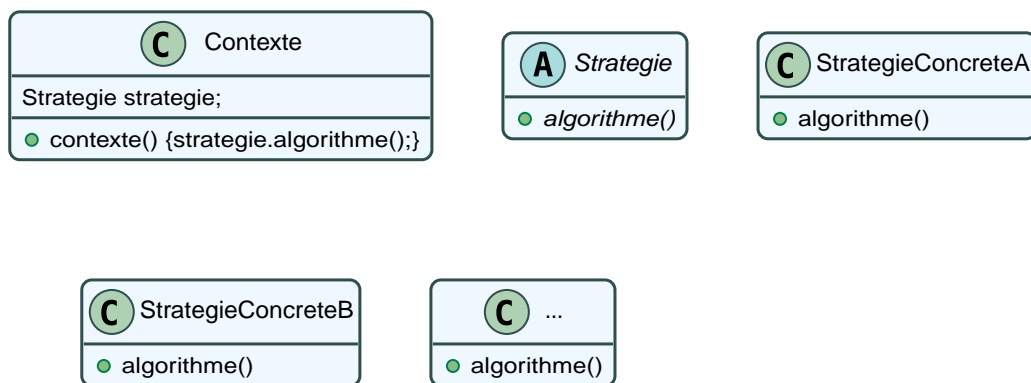


Figure 3. Design Pattern: ...

## 3. State machines

### QUESTION



1. Propose a UML state machine of a traffic light (UK/Russian one?). In a 2nd version, add the safety case of a problem (blinking orange in France).
2. Propose a UML state machine of a chess game party.

## 4. Sequence diagrams

The goal is to document the following banking application (**BankAgencyApp**) from some **Java** code excerpts.



You will have, as a practical work, to refactor this application, the goal is then not to correct the problems but to identify them.

Static method `accountsOfOwner` (from `BankAgencyApp.java`)

```
public static void accountsOfOwner (BankAgency ag, String ownerName) {
    Account [] t;

    t = ag.getAccountsOf(ownerName);
    if (t.length == 0) {
        System.out.println("no account for this name ...");
    } else {
        System.out.println(" " + t.length + " accounts for " + ownerName);
        for (int i=0; i<t.length; i++)
            t[i].print();
    }
}
```



#### QUESTION

Define a sequence diagram illustrating the behavior of this method.

`BankAgencyApp.java`

```
public class BankAgencyApp {

    public static void main(String argv[]) {

        String choice;

        boolean continue ;
        Scanner lect;
        BankAgency myAgency ;

        String name, number;
        Account c;
        double amount;

        lect = new Scanner ( System.in );
        lect.useLocale(Locale.US);

        myAgency = AccesBankAgency.getBankAgency();

        continue = true;
        while (continue) {
            ...
            choice = lect.next();
            choice = choice.toLowerCase();
            switch (choice) {
                case "q" :
                    System.out.println("ByeBye");
                    continue = false;
                    break;
                case "l" :
```

```

        myAgency.print();
        break;
        case "v" :
            System.out.print("Num Account -> ");
            number = lect.next();
            c = myAgency.getAccount(number);
            if (c==null) {
                System.out.println("Account non existing ...");
            } else {
                c.print();
            }
            break;
        case "p" :
            System.out.print("Owner -> ");
            name = lect.next();
            BankAgencyApp.accountsOfOwner (myAgency, name);
            break;
        case "d" :
            ...
            break;
        case "r" :
            ...
            break;
        default :
            ...
            break;
    }
}

public static void accountsOfOwner (BankAgency ag,
    String ownerName) {...}

public static void depositOnAccount (BankAgency ag,
    String numberAccount, double amount) {...}

public static void withdrawFromAccount (BankAgency ag,
    String numberAccount, double amount) {...}
}

```

```
public class AccesBankAgency {  
  
    private AccesBankAgency () {}  
    public static BankAgency getBankAgency () {  
  
        BankAgency ag = new BankAgency("Tinkoff Bank", "Kazan");  
        ...  
    }  
    ...  
}
```

## Appendix A: Still hungry?...



### QUESTION

1. Provide the class diagram of the application
2. Does `AccesBankAgency` remind you of something ?
3. Provide the sequence diagram illustrating the behavior of this application ( `main`). Do not take care of the scanners.
4. Is it possible, in a `Java` code, to make the difference between the aggregation `1 <-> *` and the association `1 -> *`?
5. Commit&Push when everything is ready


 `commit/push`

`fix #Bonus: Here is additional material...`

## Contributors

- [Jean-Michel Bruel](#)

## About...

Baked with {asciidoctorlink} (version `2.0.11`) from 'Dan Allen', based on {asciidoc}. 'Licence Creative Commons'.  [licence Creative Commons Paternité - Partage à l'Identique 3.0 non transposé](#).