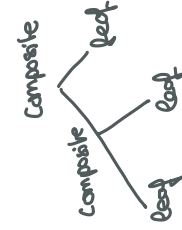


(Golf pdf design pattern)



Q1. Qu'affiche ce programme ?

Q2. Représentez sur un diagramme la *structure* des classes de ce programme et les associations qui les lient entre elles.

Q3. Représentez sur un diagramme les dépendances qui existent entre les classes de cet exemple.

Q4. Représentez sur un diagramme l'état de la mémoire (e.g. instances de classes présentes) du programme au moment d'évaluer « e ».

programme au moment d'évaluer « C ».

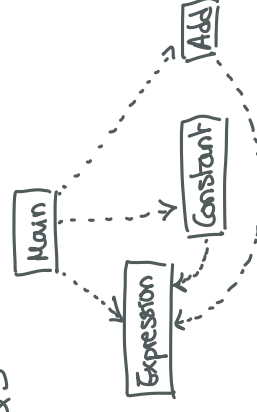
Q5. Représentez sur un diagramme la dynamique des échanges entre objets ; on souhaite modéliser tout ce qui se passe au cours de l'invocation dans le main à « e.eval() ».

Q6. Expliquez les liens de cohérence entre ces diagrammes.

Q7. Peut-on fusionner l'information de ces diagrammes en un seul ? Est-ce souhaitable ?

Q1. Ce programme affiche

28



op() static  
⇒ A.op():

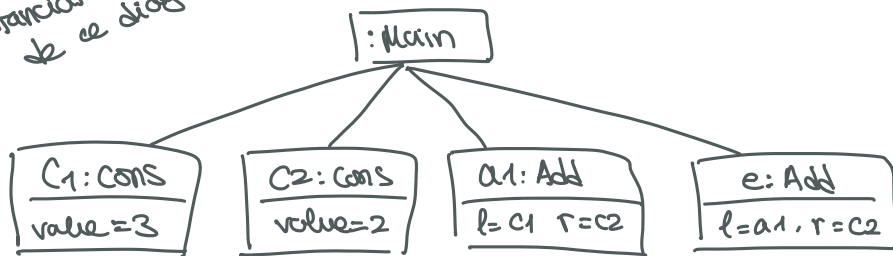
$$\frac{A}{\phi L}$$

50() non static

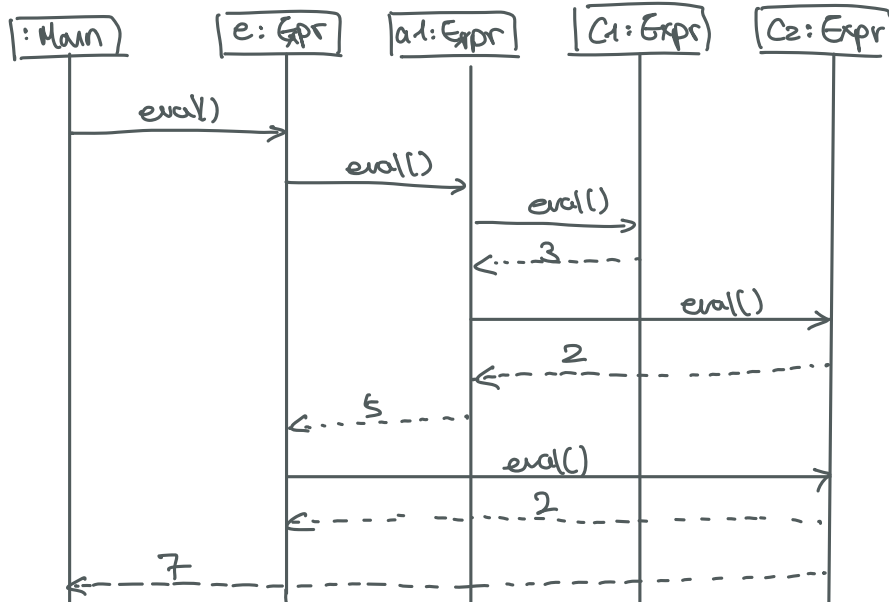
diagramme  
objets  
Q4

instanciation  
de ce diagramme

⇒ A a = new A();  
a.op(c);



Q5



Q6.

Q7 Non.

si on veut changer un truc d'une variable  
il suffit de changer la factory.  
pas tout le code.

Dans l'esprit du design pattern « Factory », on propose d'utiliser plutôt cette variante du code :

```

MainFac.java
package il;

public class MainFac {
    public static void main(String[] args) {
        Expression c1 = ExprFactory.createConstant(3);
        Expression c2 = ExprFactory.createConstant(2);

        Expression a1 = ExprFactory.createAdd(c1, c2);
        Expression e = ExprFactory.createAdd(a1, c2);

        System.out.println("e="+e.eval());
    }
}
  
```

```

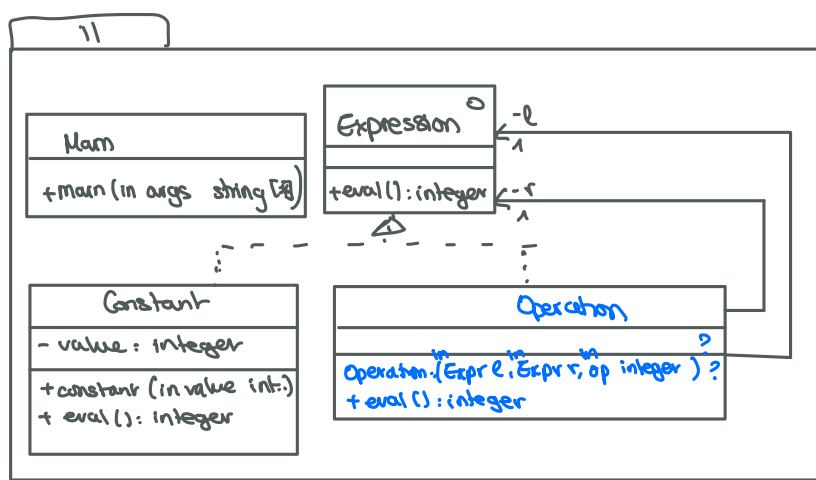
ExprFactory.java
package il;

public class ExprFactory {
    public static Expression
        createConstant(int v) {
        return new Constant(v);
    }
    public static Expression
        createAdd(Expression l, Expression r) {
        return new Add(l,r);
    }
}
  
```

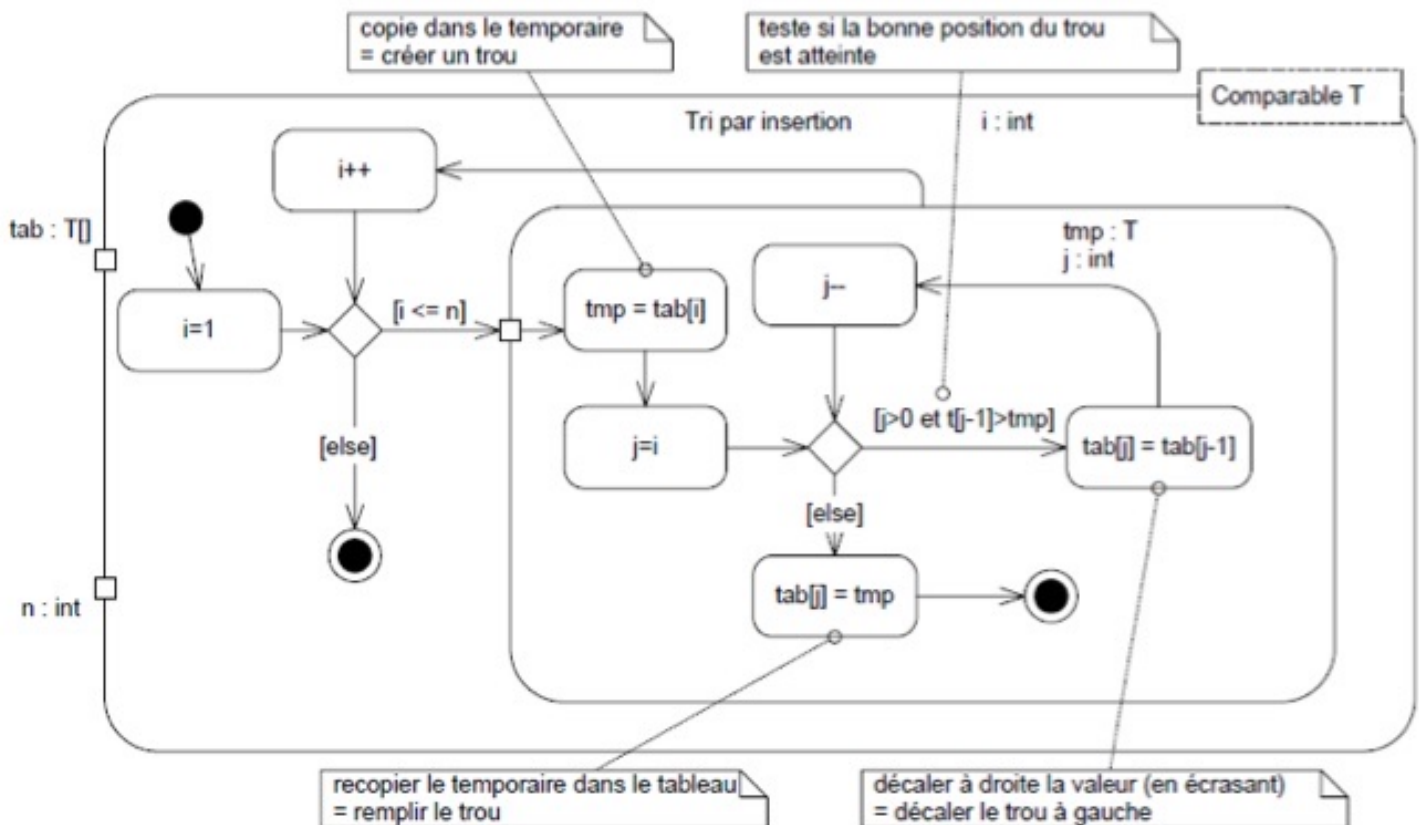
Q8. Quel est l'impact le comportement du programme ainsi que sur les diagrammes précédents de cette modification ?

Main connaît que interface et factory  
On cache les classes concretes au main, il manipule que des expressions

Q9. On souhaite étendre l'application en introduisant la multiplication. Quelle représentation est la plus appropriée à votre avis pour raisonner sur les changements à introduire dans le code ?



`return new (Operation (l, r, Add))`



Comparez le diagramme d'activité UML et le code java correspondant.

```

public static <T extends Comparable<T>> void triSelect(T[] tab) {
    int n = tab.length;
    for (int i = 1; i <= n; i++) {
        T tmp = tab[i];
        int j;
        for (j = i; j > 0 && tab[j - 1].compareTo(tmp) > 0; j--) {
            tab[j] = tab[j - 1];
        }
        tab[j] = tmp;
    }
}

```

Q11 – Quels sont les qualités et inconvénients de ces deux façons de présenter les choses ? Code et diagramme contiennent-ils la même information ? Quelle présentation préférer pour communiquer avec un collègue développeur Java ? Avec un informaticien qui ne parle pas Java ? Avec un mathématicien ? Avec un ordinateur ? Laquelle de ces deux représentations est la plus facile à construire ?

Code

mathématicien : aucun des deux.

machin : code 0 et 1

diff : modèle  
code

diagramme un morceau des infos  
à un niveau donné.

TD2

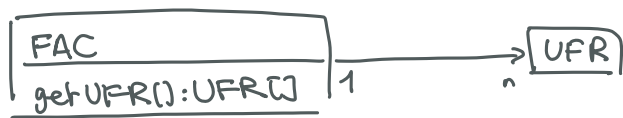
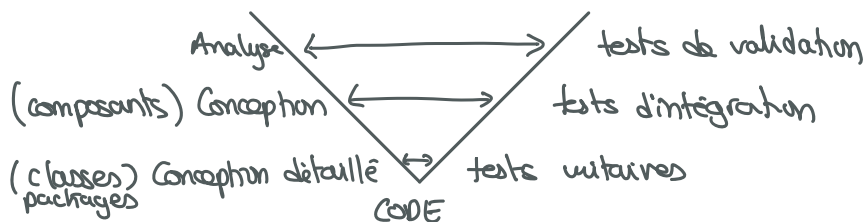


Diagramme de classe



Diagramme de use case

Diagramme de séquence  
+ test validations



③ Non

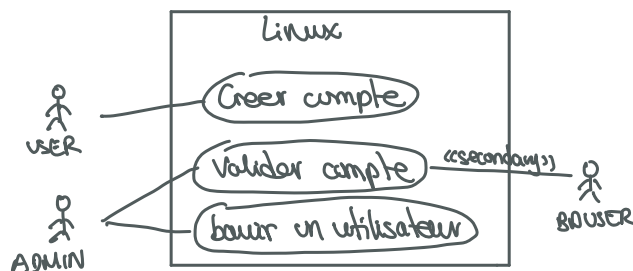
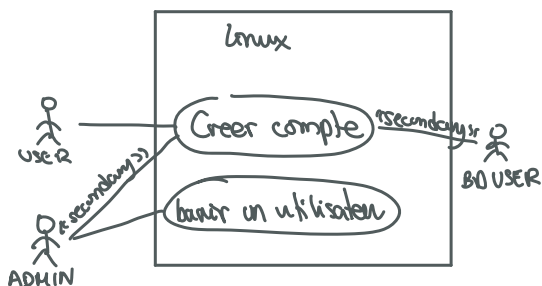
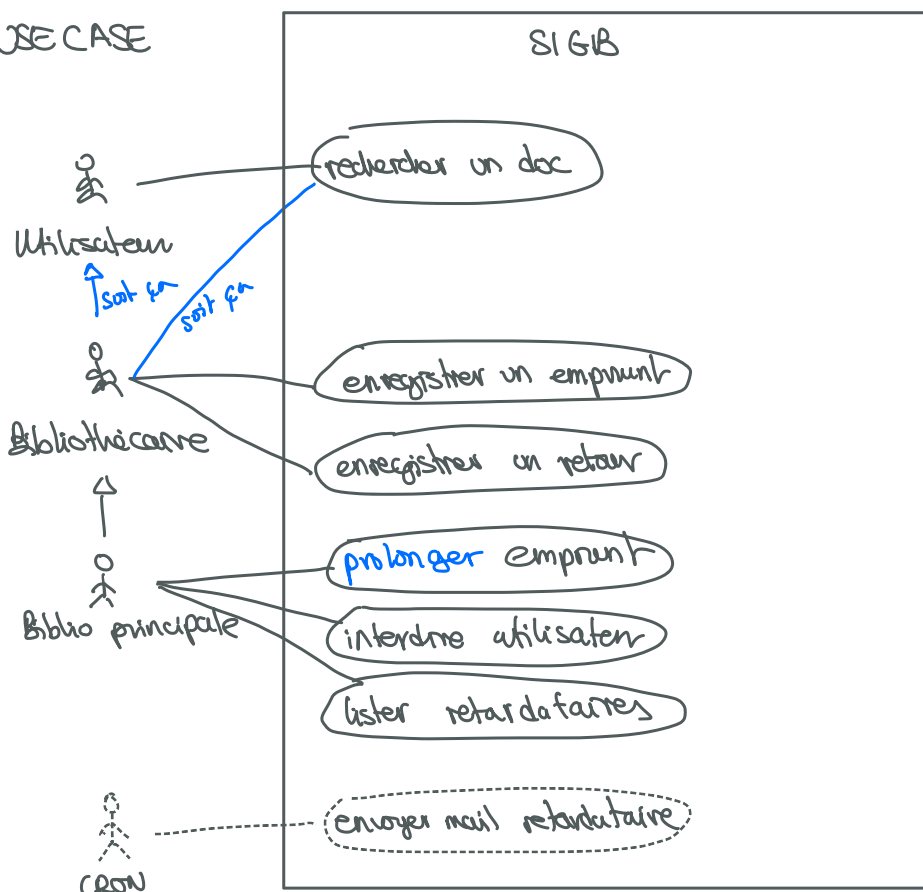
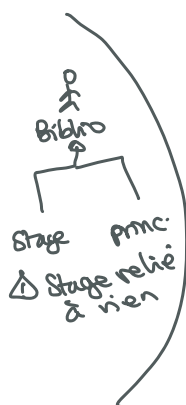
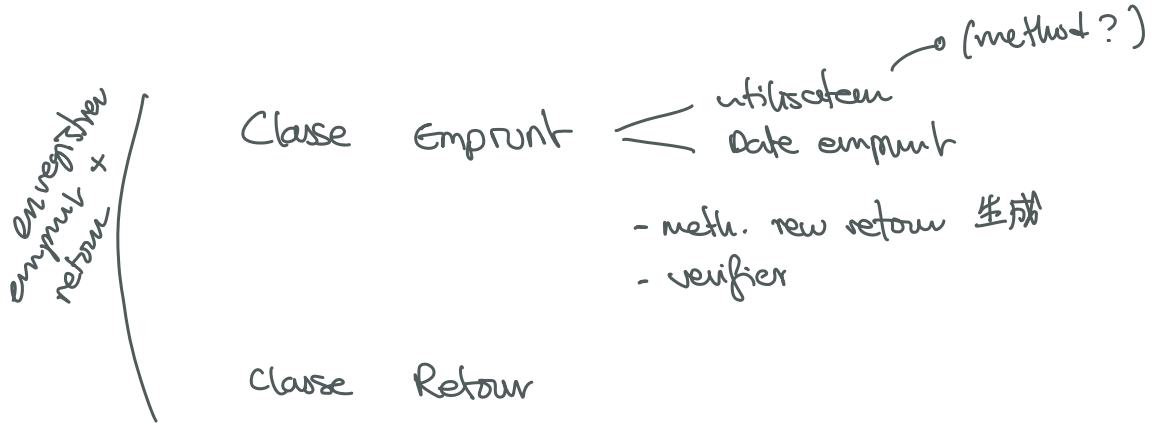
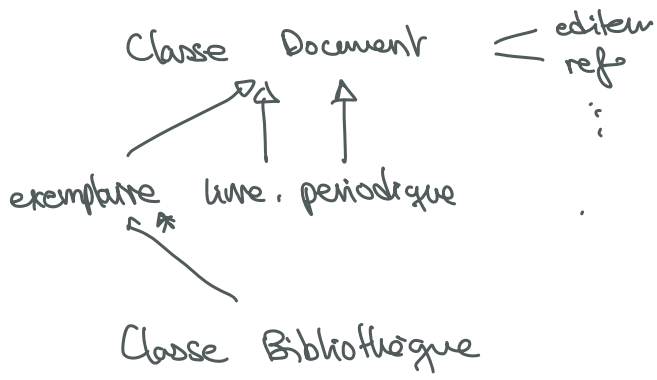


Diagramme USE CASE





Classe USER — attribut :

ABONÉ

