
INF 4067 - UML ET DESIGN PATTERNS

Devoir 2 - Implémentation des patterns de Structuration

Author

CHEDJOUN KENGUEP Dave 20U2757

Département Informatique - Master 1 - Genie Logiciel

Octobre - Novembre 2023

Diagrammes UML et résultats

Dans cette 2ème partie de notre devoir nous implémenterons les designs patterns de structuration. Pour se faire, nous avons pour chaque implémentation, des versions de codes ainsi que des versions de diagrammes correspondants à chaque code. Nous vous les présenterons ci dessous en vous donnant, une brève description, des screenshots des résultats et les images avec les modèles UML. Toutes les implémentations se trouvent sur le dépôt gitHub https://github.com/DavePhil/INF4067_Implementation_Patterns_Structuration.git.

1 Patron Adapter

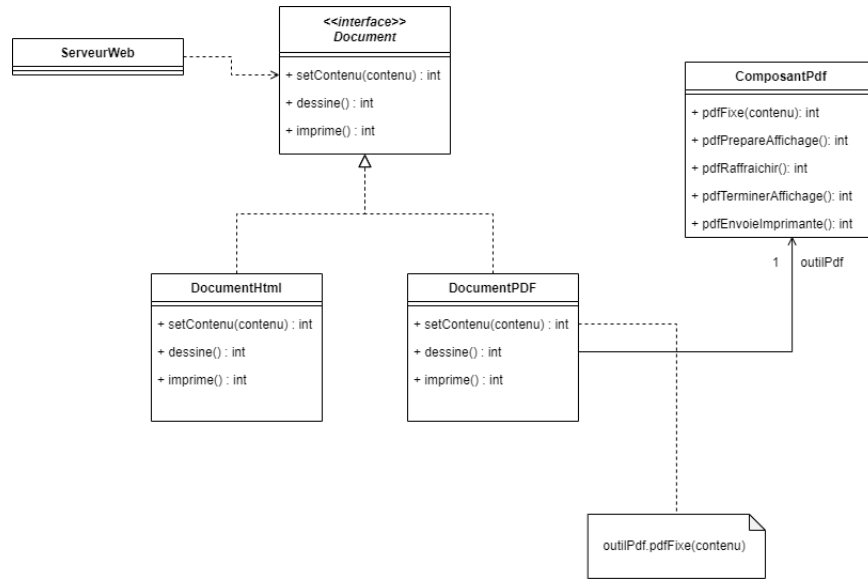
Le patron adapter est un patron, qui permet la collaboration d'instance des classes qui ont des interfaces non compatibles. Son but est de convertir l'interface d'une classe existante en une autre attendue par le client également existants afin qu'il puisse travailler ensemble. Trois exemples du cours ont été fait pour implémenter ce patron. Nous aurons une version du premier exemple (Adaptation Document), le second exemple (Adaptation de classe Gestion qui utilise les piles, pour utiliser les listes chaînées) avec deux versions et enfin un troisième exemple (l'adaptation d'une interface pour les carré qu'elle gère les rectangles). Les participants à ce patron sont:

- l'interface: Qui introduit la signature des méthodes de l'objet
- le client : Programme ou classe qui interagit avec les objets qui répondent à l'interface
- l'adapteur : Implémente les méthodes de l'adapter en invoquant les méthodes de l'adaptée
- l'adaptée: Objet dont l'interface doit être adaptée pour correspondre à l'interface

1.1 adapter_1

Il s'agit ici de l'implémentation de l'adaptation d'une interface qui traite les documents HTML, mais ayant désormais des composants PDF, on aimerait que par cette interface l'on traite les Documents PDF. Son implémentation se trouve sur le dépôt git donné plus haut et sur la branche **adapter_1**

1.1.1 Modèle



1.1.2 Résultats

```

// DocumentHtml.java
public class DocumentHtml implements Document {
    private int contenu;

    public void setContenu(int contenu) {
        this.contenu = contenu;
    }

    public void dessine() {
        System.out.println("Document HTML dessiné");
    }

    public void imprime() {
        System.out.println("Document HTML imprimé");
    }
}

// DocumentPDF.java
public class DocumentPDF implements Document {
    private int contenu;

    public void setContenu(int contenu) {
        this.contenu = contenu;
    }

    public void dessine() {
        System.out.println("Document PDF dessiné");
    }

    public void imprime() {
        System.out.println("Document PDF imprimé");
    }
}

// Main.java
public class Main {
    public static void main(String[] args) {
        Document documentPdf = new DocumentPDF();
        documentPdf.setContenu(1);
        documentPdf.dessine();
        documentPdf.imprime();

        Document documentHtml = new DocumentHtml();
        documentHtml.setContenu(2);
        documentHtml.dessine();
        documentHtml.imprime();
    }
}
  
```

Run: Main()

```

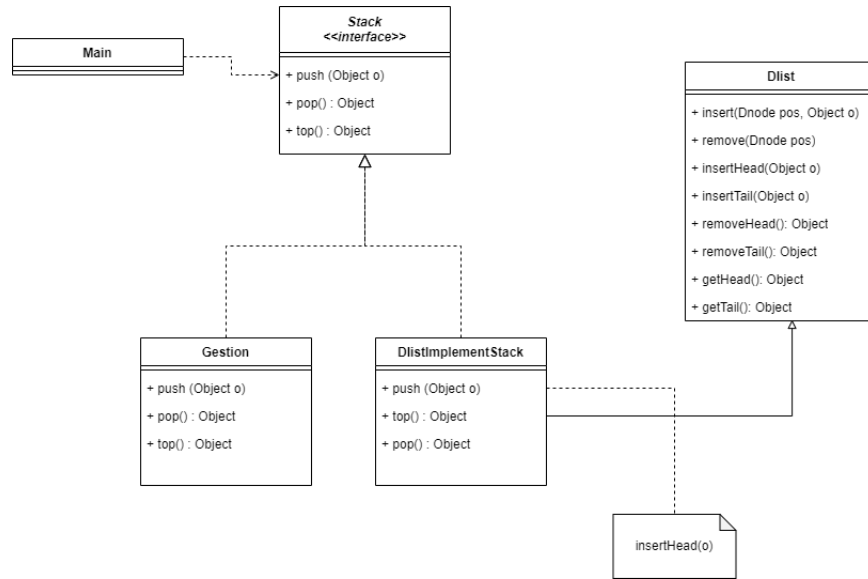
===== Document HTML =====
File le contenu : Very Good exercice au format HTML
Imprime le fichier HTML

===== Document PDF =====
File le contenu : Very Good exercice au format PDF
Imprime le fichier PDF
===== Document PDF =====
  
```

1.2 adapter_2

Il s'agit ici d'avoir une classe DList qui définit une liste doublement chaînée et d'une autre classe (Gestion) qui sait manipuler les piles, d'utiliser la classe Dlist dans Gestion sans modifier Dlist. Ici, nous choisissons comme façon de faire, d'utiliser l'héritage et l'utilisation d'une interface sur la classe Adapteur. L'implémentation de problème se trouve sur la branche **adapter_2**

1.2.1 Modèle



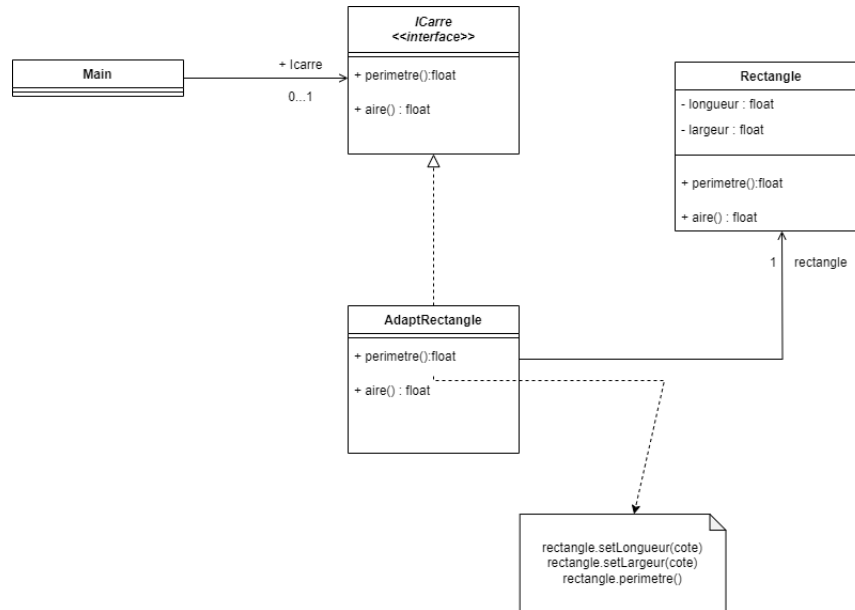
1.2.2 Résultats

The screenshot shows an IDE with a Java project. The **Main** class is visible, which implements the **Stack** interface. The code in **Main** creates a **DlistImplementStack** object and calls `push`, `top`, and `pop` methods. The output window shows the execution results, confirming that the program runs successfully and the stack operations are performed correctly.

1.3 adapter_3

Il s'agit ici d'adapter un programme utilisant une classe qui sait gérer les carrés, de tel sorte qu'elle puisse gérer les rectangles. Son implémentation se trouve sur la branche **adppter_3** du dépôt git.

1.3.1 Modèle



1.3.2 Résultats

```
1 // Adapter réalisé par SÉBASTIEN BENOIST (avec 2022/21) Hecart Denis Legault
2 // Classe cliente qui utilise les objets de l'interface
3
4 A Adapter
5
6 public class Main {
7     A Adapter
8     public static void main(String[] args) {
9         System.out.println("===== debut =====");
10        float cotecote = 20;
11        ICarré icarré = new AdaptRectangle(cotecote);
12        System.out.println("Aire: " + icarré.aire());
13        System.out.println("Perimètre: " + icarré.perimetre());
14        System.out.println("Fin =====");
15    }
16 }
```

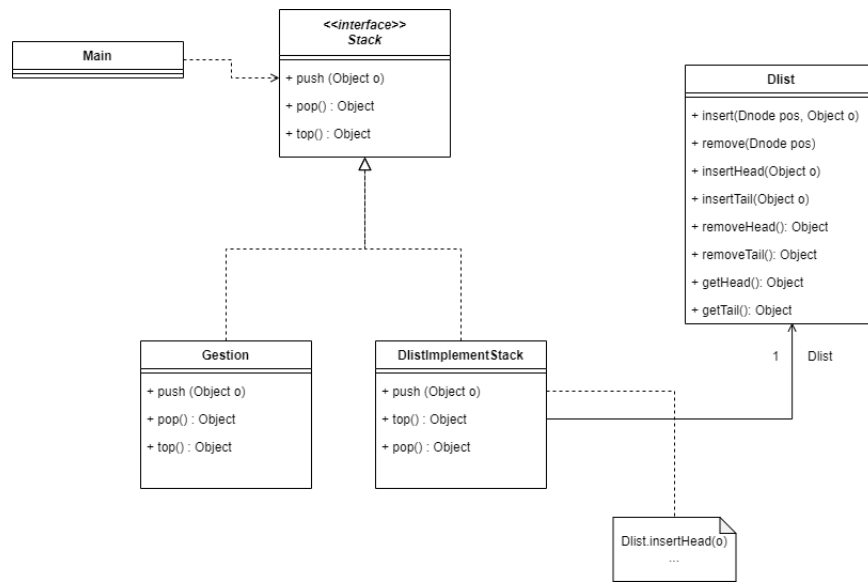
The screenshot shows the IDE output for the **Main()** method:

```
===== debut =====
Aire: 400.0
Perimètre: 200.0
===== Fin =====
Process finished with exit code 0
```

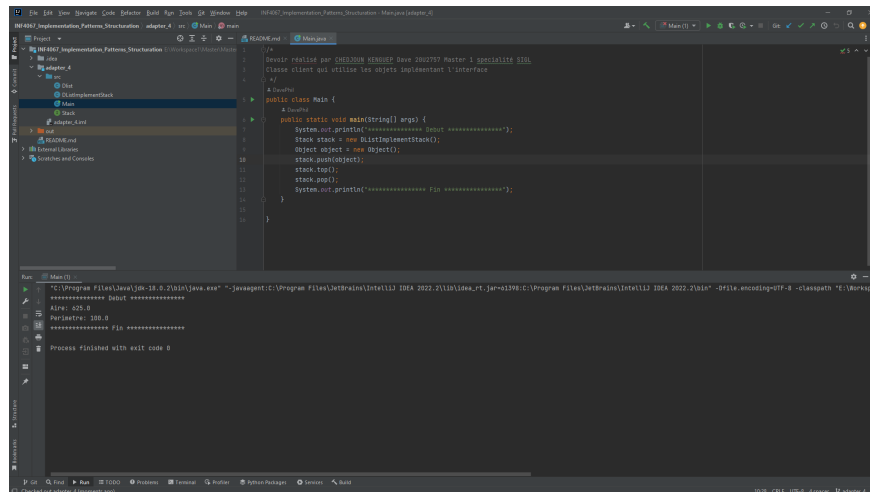
1.4 adapter_4

Ici, il s'agissait d'implémenter une deuxième version pour l'adapter_2. Nous utilisons ici plutôt qu'un héritage, une relation entre la classe adapter et la classe adaptée. L'implémentation se trouve sur la branche **adapter_4**.

1.4.1 Modèle



1.4.2 Résultats



2 Patron Composite

Le patron composite, permet de combiner des objets en structure plus grande. il permet de représenter une hiérarchie de composants. Par exemple, nous pouvons utiliser le patron composite pour représenter une hiérarchie d'organisations, une hiérarchie de menus ou une hiérarchie de fichiers. Permettre aux clients de traiter des objets individuels et des compositions d'objets de la même manière. Cela peut simplifier le code client et le rendre plus générique. Ces participants sont:

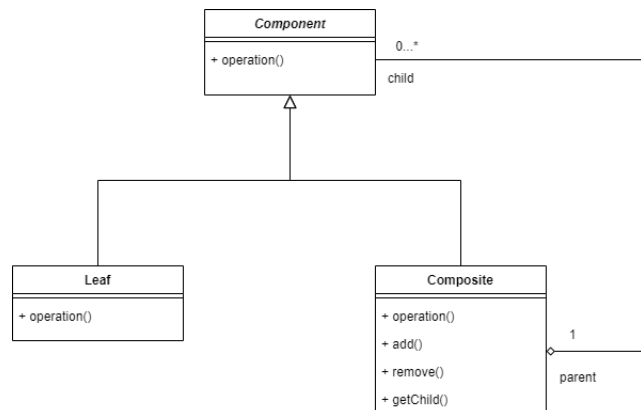
- Le composant : déclare une interface commune à tous les objets. Déclare le comportement par défaut pour toutes les classes.
- Leaf: C'est la feuille. Elle implémente le comportement élémentaire

- Composite : définit le comportement des composants ayant des fils, stocke les fils et implémente les opérations nécessaires à leur gestion.
- Le client: si le receveur est une feuille la requête est directement traité. Sinon le composite retransmet à tous ses fils.

2.1 composite_1

Cette première implémentation représente, l'implémentation du modèle générique du composite. Elle se trouvera sur la branche **composite_1** du dépôt git du devoir.

2.1.1 Modèle



2.1.2 Résultats

```

// Component.java
public interface Component {
    void operation();
}

// Leaf.java
public class Leaf implements Component {
    @Override
    public void operation() {
        System.out.println("Leaf operation");
    }
}

// Composite.java
public class Composite implements Component {
    private List<Component> children = new ArrayList<>();

    @Override
    public void operation() {
        System.out.println("Composite operation");
        for (Component child : children) {
            child.operation();
        }
    }

    public void add(Component child) {
        children.add(child);
    }

    public void remove(Component child) {
        children.remove(child);
    }

    public Component getChild(int index) {
        return children.get(index);
    }
}

// Main.java
public class Main {
    public static void main(String[] args) {
        Composite composite = new Composite();
        composite.add(new Leaf());
        composite.add(new Leaf());
        composite.add(new Leaf());
        composite.operation();
        System.out.println("Le nombre de composite est : " + composite.children.size());
        composite.operation();
        System.out.println("Le nombre de composite est : " + composite.children.size());
    }
}
  
```

Run Results:

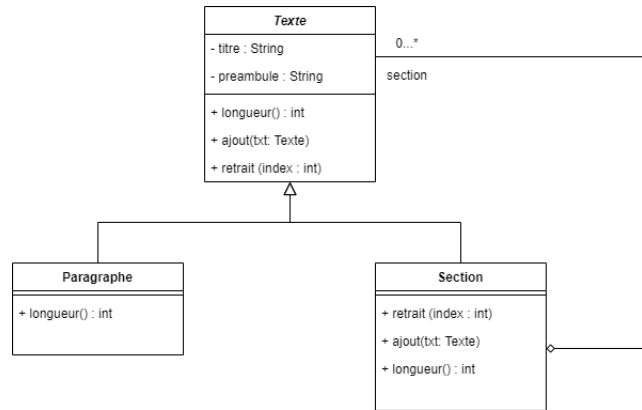
```

Le nombre de composite est : 3
Premiere Operation
Deuxieme Operation
Troisieme Operation
Le nombre de composite est : 3
Premiere Operation
Deuxieme Operation
Troisieme Operation
Le nombre de composite est : 3
  
```

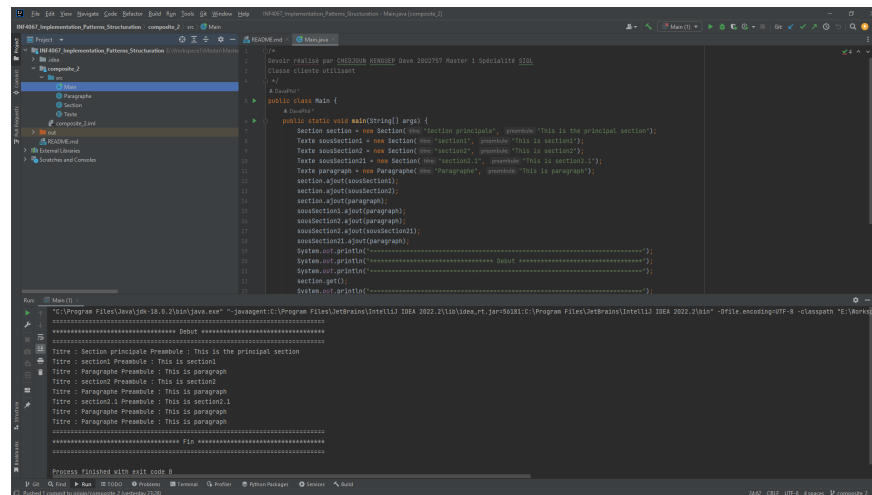
2.2 composite_2

La deuxième version du composite, correspond au deuxième exemple du cours, ou considère une section qui est un texte, et pouvant avoir plusieurs sous sections avec des paragraphes. l'implémentation se trouve sur la branche **composite_2**.

2.2.1 Modèle



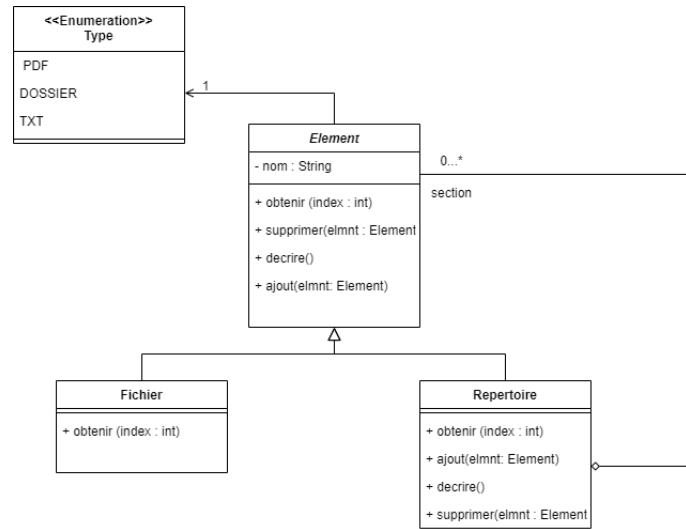
2.2.2 Résultats



2.3 composite_3

Il a été question ici, de résoudre un problème de gestion de fichiers. le système gère des répertoires, qui peuvent avoir des sous répertoires, ou alors des fichiers. de type PDF ou TXT. son implémentation se trouve sur la branche **composite_3**

2.3.1 Modèle



2.3.2 Résultats

The screenshot displays the development environment with the following components:

- Project Explorer:** Shows the project structure including `compo3`, `compo3_1`, `compo3_2`, `compo3_3`, `compo3_4`, `compo3_5`, `compo3_6`, `compo3_7`, `compo3_8`, `compo3_9`, `compo3_10`, `compo3_11`, `compo3_12`, `compo3_13`, `compo3_14`, `compo3_15`, `compo3_16`, `compo3_17`, `compo3_18`, `compo3_19`, `compo3_20`, `compo3_21`, `compo3_22`, `compo3_23`, `compo3_24`, `compo3_25`, `compo3_26`, `compo3_27`, `compo3_28`, `compo3_29`, `compo3_30`, `compo3_31`, `compo3_32`, `compo3_33`, `compo3_34`, `compo3_35`, `compo3_36`, `compo3_37`, `compo3_38`, `compo3_39`, `compo3_40`, `compo3_41`, `compo3_42`, `compo3_43`, `compo3_44`, `compo3_45`, `compo3_46`, `compo3_47`, `compo3_48`, `compo3_49`, `compo3_50`, `compo3_51`, `compo3_52`, `compo3_53`, `compo3_54`, `compo3_55`, `compo3_56`, `compo3_57`, `compo3_58`, `compo3_59`, `compo3_60`, `compo3_61`, `compo3_62`, `compo3_63`, `compo3_64`, `compo3_65`, `compo3_66`, `compo3_67`, `compo3_68`, `compo3_69`, `compo3_70`, `compo3_71`, `compo3_72`, `compo3_73`, `compo3_74`, `compo3_75`, `compo3_76`, `compo3_77`, `compo3_78`, `compo3_79`, `compo3_80`, `compo3_81`, `compo3_82`, `compo3_83`, `compo3_84`, `compo3_85`, `compo3_86`, `compo3_87`, `compo3_88`, `compo3_89`, `compo3_90`, `compo3_91`, `compo3_92`, `compo3_93`, `compo3_94`, `compo3_95`, `compo3_96`, `compo3_97`, `compo3_98`, `compo3_99`, `compo3_100`.
- Source Editor:** Contains the `main` method of the `compo3` class, which demonstrates the use of the `Repertoire` and `Fichier` classes. The code includes comments in French and performs operations like adding, removing, and describing elements.
- Run Console:** Shows the output of the program execution, including the creation of a repertoire, adding files, and the final state of the repertoire.