TD6 - Design Pattern Itérateur

Génie logiciel

Justin Bossard

20 novembre 2024

Première implémentation

1. On créer la classe `Plat` :

```java
public class Plat {
    private String nom;
    private String description;
    private boolean vegetarien;
    private double prix;

    public Plat(String nom, String description, boolean vegetarien, double prix) {
        this.nom  = nom;
        this.description = description;
        this.vegetarien = vegetarien;
        this.prix = prix;
    }

    public String getNom() {
        return nom;
    }

    public String getDescription() {
        return description;
    }

    public boolean isVegetarien() {
        return vegetarien;
    }

    public double getPrix() {
        return prix;
    }
}
```

2. On créer la classe `MenuCreperie` :

```java
import java.util.ArrayList;
import java.util.List;

public class MenuCreperie implements Menu {
    private List<Plat> plats;

    public MenuCreperie() {
        plats = new ArrayList<Plat>();
```

```
        ajouterPlat("Galette Annie", "Jambon blanc, emmental", false, 10.90);
        ajouterPlat("Galette Jeannette", "Jambon blanc, œuf, emmental", false, 12.50);
        ajouterPlat("Galette Paulette", "Jambon blanc, œuf, emmental, champignons à la
        ↪  crème", false, 13.50);
        ajouterPlat("Galette Germaine", "Légumes au choix : fondue de poireaux, épinards,
        ↪  ratatouille Maison selon la saison ou champignons à la crème", true, 11.90);
    }

    public void ajouterPlat(String nom, String description, boolean vegetarien, double
    ↪  prix) {
        Plat plat = new Plat(nom, description, vegetarien, prix);
        plats.add(plat);
    }

    public List<Plat> getPlats() {
        return plats;
    }
}
```

Puis la classe `MenuCafeteria` :

```
public class MenuCafeteria {
    private final static int MAX_PLATS = 5;
    private int nombreDePlats = 0;
    private Plat[] plats;

    public MenuCafeteria() {
        plats = new Plat[MAX_PLATS];
        ajouterPlat("Salade du Forez", "Saucisson chaud lyonnais artisanal, Râpées de
        ↪  pomme de terre maison", false, 11.50);
        ajouterPlat("Salade bergère", "Fourme fondue, Crottin de chèvre chaud",false,
        ↪  11.50);
        ajouterPlat("Salade bien-être", "Assortiment de crudités",true, 11.50);
    }

    public void ajouterPlat(String nom, String description, boolean vegetarien, double
    ↪  prix) {
        Plat plat = new Plat(nom, description, vegetarien, prix);
        if (nombreDePlats >= MAX_PLATS) {
            System.err.println("Il a y déjà " + MAX_PLATS + " dans le menu, ajout
            ↪  impossible");
        } else {
            plats[nombreDePlats] = plat;
            nombreDePlats++;
        }
    }

    public Plat[] getPlats() {
        return plats;
    }

    public int getNombrePlats() {
        return nombreDePlats;
    }
}
```

3. On créer une classe `Serveuse` :

```java
public class Serveuse {

    private MenuCreperie menuCreperie;
    private MenuCafeteria menuCafeteria;

    public Serveuse(MenuCreperie menuCreperie, MenuCafeteria menuCafeteria) {
        this.menuCreperie = menuCreperie;
        this.menuCafeteria = menuCafeteria;
    }

    public void afficherMenu() {
        // Cafétéria
        System.out.println("Menu de la cafétéria");
        for(int i = 0 ; i < menuCafeteria.getNombrePlats(); i++) {
            System.out.println("\nNom : " + menuCafeteria.getPlats()[i].getNom());
            System.out.println(menuCafeteria.getPlats()[i].getDescription());
            System.out.println("Prix : " + menuCafeteria.getPlats()[i].getPrix());
            System.out.println("Végétarien : " +
            ↪  menuCafeteria.getPlats()[i].isVegetarien());
        }
        // Crêperie
        System.out.println("\n\nMenu de la crêperie");
        for(int i = 0 ; i < menuCreperie.getPlats().size(); i++) {
            System.out.println("\nNom : " + menuCreperie.getPlats().get(i).getNom());
            System.out.println(menuCreperie.getPlats().get(i).getDescription());
            System.out.println("Prix : " + menuCreperie.getPlats().get(i).getPrix());
            System.out.println("Végétarien : " +
            ↪  menuCreperie.getPlats().get(i).isVegetarien());
        }
    }
}
```

On la teste de la manière suivante :

```java
public class TestServeuse {
    public static void main(String[] args) {
        MenuCreperie menuCreperie = new MenuCreperie();
        MenuCafeteria menuCafeteria = new MenuCafeteria();

        Serveuse serveuse = new Serveuse(menuCreperie, menuCafeteria);
        serveuse.afficherMenu();
    }
}
```

4. - `MenuCreperie` et `MenuCafeteria` sont des implémentations concrètes pour une interface non définie.

     Faux, il s'agit de classe implémentant directement un menu. Il est possible de réaliser un interface `Menu` simplement, mais cela n'est pas fait ici.

   - La Serveuse n'implémente pas l'API Java `Waitresse`.

     Vrai, on n'utilise pas cette API.

   - Si `MenuCafeteria` utilise un HashMap alors le code de Serveuse devra être revu.

     Vrai, car on ne pourrait pas afficher le menu de la même manière.

   - L'implémentation de `Serveuse` respecte la notion d'encapsulation.

Vrai, ses attributs sont privés, et ne sont accessible que par la méthode `afficherMenu()`.

- Nous avons du code dupliqué et une troisième implémentation imposerait une nouvelle boucle.

  Vrai, entre `MenuCreperie` et `MenuCafeteria`, pour ajouter et obtenir les plats.

- L'implémentation n'utilise pas un métalangage (XML/JSON) afin d'assurer l'interopérabilité des menus.

  Vrai, on n'utilise à aucun moment ces langages pour représenter les menus.

5. Cette implémentation comporte plusieurs défauts :
   - Il y a du code dupliqué entre `MenuCreperie` et `MenuCafeteria`. Les listes ne sont pas faites de la même manière, d'un point de vue de l'abstraction cela est discutable.
   - L'affichage et l'accès au menu est pénible, il serait plus pertinent d'utiliser un `Iterator`.

Deuxième implémentation

6. Pour encapsuler l'itération, on introduit l'interface `Iterateur`, structurée comme telle :

```java
public interface Iterateur {
    boolean encore();
    Plat suivant();
}
```

7. On implémente la classe `IterateurMenuCafeteria` comme telle :

```java
public class IterateurMenuCafeteria implements Iterateur {
    private Plat[] plats;
    private int position;

    public IterateurMenuCafeteria(Plat[] plats) {
        this.plats = plats;
        this.position = 0;
    }

    @Override
    public boolean encore() {
        return position < plats.length && plats[position] != null;
    }

    @Override
    public Plat suivant() {
        if (encore()) {
            return plats[position++];
        }
        return null;
    }
}
```

8. On ajoute la méthode `creerIterateur()` à la classe `MenuCafeteria` :

```java
public Iterateur creerIterateur() {
    return new IterateurMenuCafeteria(plats);
}
```

9. Pour la crêperie, on commence par créer la classe `IterateurMenuCreperie` :

```java
import java.util.List;

public class IterateurMenuCreperie implements Iterateur {
```

```
    private List<Plat> plats;
    private int position;

    public IterateurMenuCreperie(List<Plat> plats) {
        this.plats = plats;
        this.position = 0;
    }

    @Override
    public boolean encore() {
        return position < plats.size();
    }

    @Override
    public Plat suivant() {
        if (encore()) {
            return plats.get(position++);
        }
        return null;
    }
}
```

Puis on ajoute la méthode `creerIterateur` à la classe `MenuCreperie` :

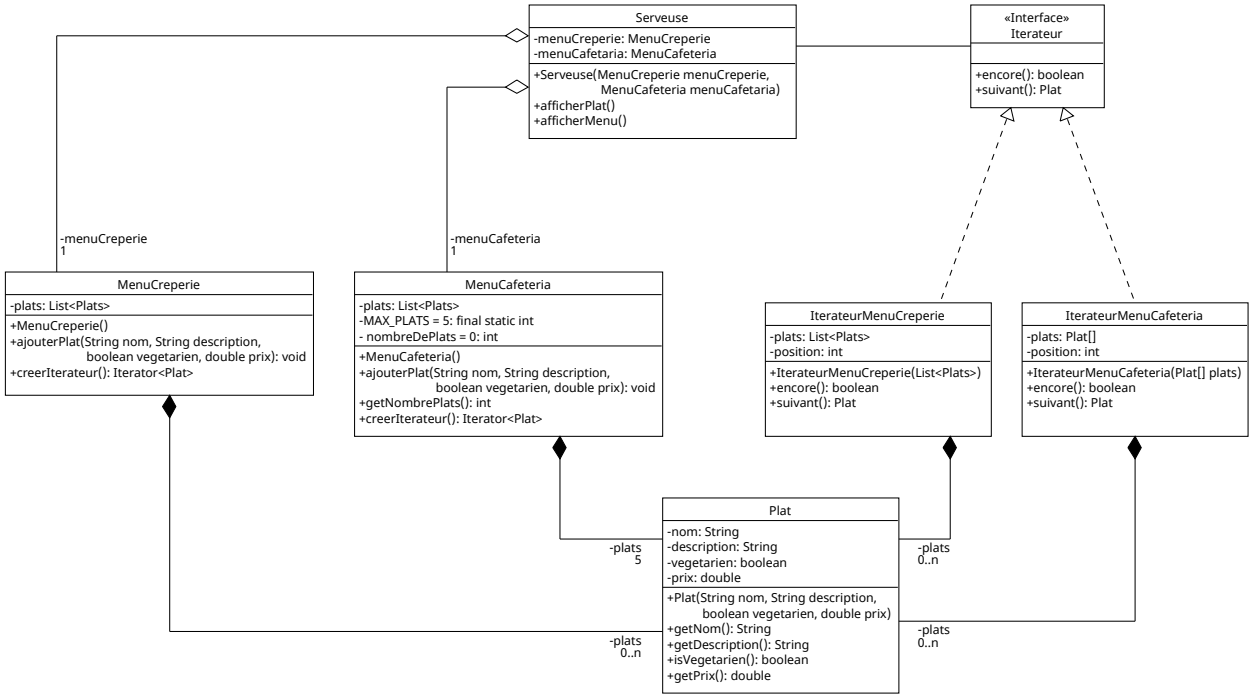
```
public Iterateur creerIterateur() {
    return new IterateurMenuCreperie(plats);
}
```

10. On modifie à présent le code de `Serveuse` pour prendre en compte l'itérateur, et on en profite pour factoriser le code :

```
public void afficherPlat(Plat plat) {
    System.out.println("\nNom : " + plat.getNom());
    System.out.println(plat.getDescription());
    System.out.println("Prix : " + plat.getPrix());
    System.out.println("Végétarien : " + plat.isVegetarien());
}

public void afficherMenu() {
    Plat plat;

    // Cafétaria
    System.out.println("Menu de la caféteria");
    Iterateur itCafeteria = menuCafetaria.creerIterateur();

    while (itCafeteria.encore()) {
        plat = itCafeteria.suivant();
        afficherPlat(plat);
    }
    // Crêperie
    System.out.println("\n\nMenu de la crêperie");
    Iterateur itCreperie = menuCreperie.creerIterateur();
    while (itCreperie.encore()) {
        plat = itCreperie.suivant();
        afficherPlat(plat);
    }
```

```
}
```

11. On a le diagramme de classe suivant :



Troisième implémentation

12. 1. On modifie la méthode `creerIterateur` de la classe `MenuCreperie` :

```java
public Iterator<Plat> creerIterateur() {
    return plats.iterator();
}
```

2. On modifie maintenant la classe `IterateurMenuCafeteria` pour utiliser également l'interface `Iterator` de Java :

```java
public class IterateurMenuCafeteria implements Iterator<Plat> {
    private Plat[] plats;
    private int position;

    public IterateurMenuCafeteria(Plat[] plats) {
        this.plats = plats;
        this.position = 0;
    }

    @Override
    public boolean hasNext() {
        if (position >= plats.length || plats[position] == null) {
            return false;
        }
        return true;
    }

    @Override
    public Plat next() {
        Plat plat = plats[position];
        position++;
        return plat;
    }

    @Override
    public void remove() {
        if (position <= 0) {
            throw new IllegalStateException("Suppression impossible à cette
            ↪ position");
        }
        if (plats[position-1] != null) {
            for (int i = position-1; i < (plats.length-1); i++) {
                plats[i] = plats[i+1];
            }
            plats[plats.length-1] = null;
        }
    }
}
```

L'emploi d'un itérateur permet de rendre inutiles les méthodes `getPlats()` des deux menus.

13. On créer l'interface `Menu` de la manière suivante :

```java
public interface Menu {
    public void ajouterPlat(String nom, String description, boolean vegetarien, double
    ↪ prix);
    public Iterator<Plat> creerIterateur();
```

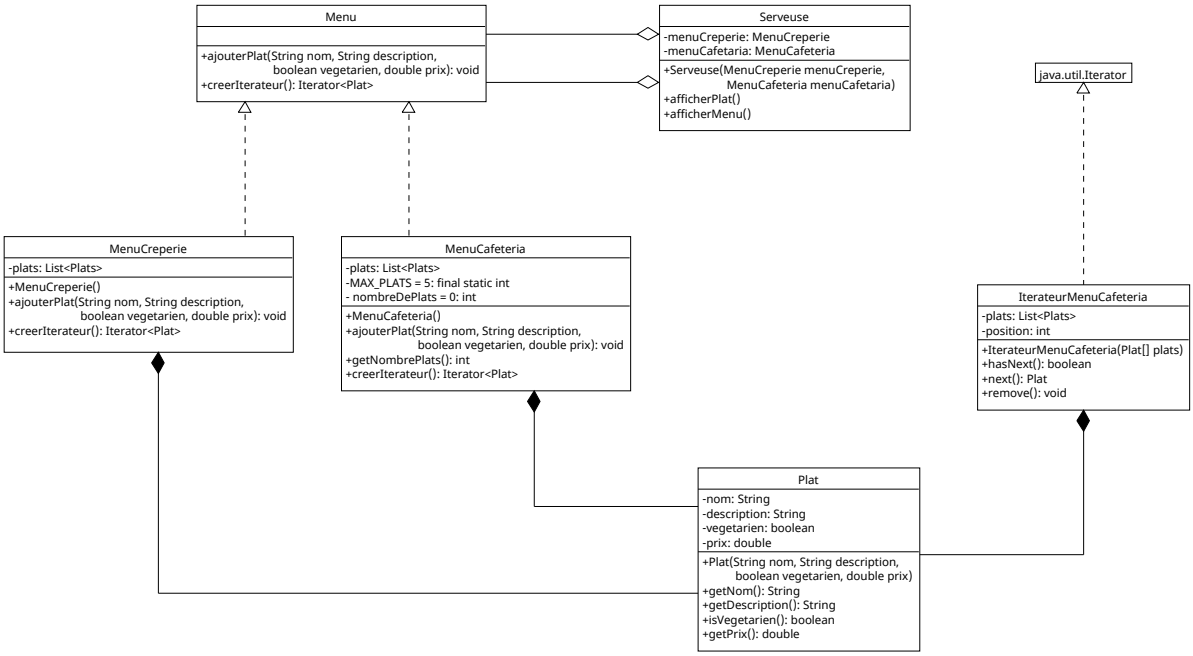
```
}
```

Puis on modifie les menus en ajoutant la directive `@Override` aux méthodes `ajouterPlat()` et `creerIterateur()` des menus.

14. On modifie la classe `Serveuse` comme suit :

```
public class Serveuse {

    private MenuCreperie menuCreperie;
    private MenuCafeteria menuCafeteria;

    public Serveuse(MenuCreperie menuCreperie, MenuCafeteria menuCafetaria) {
        this.menuCreperie = menuCreperie;
        this.menuCafeteria = menuCafetaria;
    }

    public void afficherPlat(Plat plat) {
        System.out.println("\nNom : " + plat.getNom());
        System.out.println(plat.getDescription());
        System.out.println("Prix : " + plat.getPrix());
        System.out.println("Végétarien : " + plat.isVegetarien());
    }

    public void afficherMenu() {
        // Cafétéria
        System.out.println("Menu de la cafétéria");

        Iterator<Plat> iterCafeteria = menuCafeteria.creerIterateur();
        while (iterCafeteria.hasNext()) {
            Plat plat = iterCafeteria.next();
            afficherPlat(plat);
        }
        // Crêperie
        System.out.println("\n\nMenu de la crêperie");

        Iterator<Plat> iterCreperie = menuCreperie.creerIterateur();
        while (iterCreperie.hasNext()) {
            Plat plat = iterCreperie.next();
            afficherPlat(plat);
        }
    }
}
```

15. 1. À partir de mon code source, on a le diagramme de classe suivant :



2. À partir du schéma fourni :