Compte Rendu TP3

Léo VALETTE - Romain SERPOLLET

Version 1 de l'application

- Question 1:

Modifier le code de sorte que Consommable soit une interface (et non une classe).

```
public interface Consommable {
    public String getNom();
    public int getPrix();
}
```

Modifier en conséquence les classes `Plat` et `Boisson`.

- Reponse:

On modifie les deux classes en mettant `implements` au lieux de `extends

```
public class Plat implements Consommable, Nutrition{
  //classe Plat
}
```

- Question 2:

Modifier la classe de test TestRestaurant avec une méthode main que vous compléterez au fur et à mesure que vous répondrez aux questions pour tester vos solutions.

- Reponse:

La méthode main de la classe `TestRestaurant` contient plusieurs tests qui auront pour but de tester chaque méthode implémentée.

- Question 3:

Compléter le constructeur de la classe `Menu`

- Reponse:

```
public Menu(int prix, Entree e, PlatPrincipal p, Dessert d, Boisson b) throws Exception{
    this.prix = prix;
    items = new ArrayList<Consommable>();
    items.add(e);
    items.add(p);
    items.add(d);
    items.add(b);
    //On doit vérifier que le prix du menu est inférieur à la somme des prix de ses composants
    if(!verifPrixMenu()){
        throw new Exception("Prix incorrect");
    }
}
```

- Question 4:

Compléter la méthode `toString` de cette même classe afin qu'elle affiche la liste des plats et la boisson du menu (par exemple : "Menu composé de Salade verte, Pizza Reine, Tiramisu, Eau, au prix de 15 euros").

- Reponse:

```
public String toString(){
     String message = "Menu composé de ";
     for(Consommable it : items){
         message += it.getNom() + ", ";
     }
     message += "au Prix de " + this.prix + " euros";
     return message;
}
```

- Question 5:

  Ajouter à la classe Menu la méthode privée `private boolean verifPrixMenu()` Cette méthode retourne une valeur vraie uniquement si la somme des prix des items composant un menu est supérieure ou égale au prix du menu qui doit lui-même être strictement positif. Modifier le constructeur en conséquence.

- Reponse:

```
private boolean verifPrixMenu(){
    int sum = 0;
    //on fait la somme des prix des items
    for(Consommable it : items){
        sum += it.getPrix();
    }
    //on renvoie vrai si le prix des items est superieur au prix du menu
    //et si le prix du menu est strictement positif
    return(prix>0 && prix<=sum);
}
```

- Question 6:

La carte du restaurant contient l'ensemble de plats et boissons proposés par le restaurant classés par type (entrées, plats principaux, desserts, boissons) ainsi que les différents menus. La classe Carte joue ce rôle. On peut constater, en lisant son implémentation incomplète, qu'au moment d'ajouter un item dans la carte, on vérifie qu'un item du même nom n'y apparaît pas déjà (afin d'éviter qu'un plat nommé "Tiramisu" soit inséré tandis qu'une entrée du même nom figure déjà dans la carte).

Compléter la méthode `verifCarte` qui effectue cette vérification. (On doit implementer cette methode avec les iterators)

- Reponse:

```
private boolean verifCarte(Consommable c){
    //on creer un iterateur de type Consommable
    Iterator<Consommable> iter;
    //on iter d'abbord sur les entrees
    iter = entrees.iterator();
    //on fait notre boucle tant que l'iterateur a un suivant
    while(iter.hasNext()){
        //on recupere le nom du Consommable pointe par l'iterateur
        //et on test sa valeur avec la valeur de notre Consommable avec la methode
        //equals
        if(iter.next().getNom().equals(c.getNom())){
            return false;
        }
    }
    //puis sur les desserts
    iter = desserts.iterator();
    while(iter.hasNext()){
        if(iter.next().getNom().equals(c.getNom())){
            return false;
        }
    }
    //les plats principaux
    iter = platsPrincipaux.iterator();
    while(iter.hasNext()){
        if(iter.next().getNom().equals(c.getNom())){
            return false;
        }
    }
    //et enfin les boissons
    iter = boissons.iterator();
    while(iter.hasNext()){
        if(iter.next().getNom().equals(c.getNom())){
            return false;
        }
    }

    return true;
}
```

- Question 7:

Une autre vérification est effectuée au moment de l'insertion des menus dans la carte : tous les items (plats, boissons) composant le menu doivent figurer déjà dans la carte.

Réaliser la méthode `verifMenu` qui effectue cette vérification.

- Reponse:

```
public boolean verifMenu(Menu m) {
    //on verifie que chaque item du menu est dans la carte
    for(Consommable c : m.getItems()) {
        if(verifCarte(c)) {
            return false;
        }
    }
    return true;
}
```

- Question 8:

  Ajouter la méthode `public int calculerPrixCommande(Commande c)` à la classe `Carte` calculant le prix d'une commande

- Reponse:

```java
public int caclulerPrixCommade(Commande c) {
    int prix = 0;
    //stock si c'est la fin des menus
    boolean fin = false;
    //stock si il y a eu des menus durant la derniere boucle
    boolean is_there_menu;
    //commande temporaire pour ne pas detruire celle en entree
    Commande commande = new Commande();
    //methode addList permet d'ajouter une liste d'items a une commande
    commande.addList(c.getItemsCommands());
    /*tant qu'on a des menus on verifie pour chaque menu si il est dans la
    commande*/
    while(!fin) {
        is_there_menu = false;
        for(Menu m : menus) {
            /*on a implemente une methode isThereMenu dans la classe commande
            pour savoir si une commande contient un menu passe en parametre*/
            if(commande.isThereMenu(m)) {
                /*on a implemente une methode removeMenu dans la classe commande
                pour supprimer de la commande une fois chque item d'un menu*/
                commande.removeMenu(m);
                prix += m.getPrix();
                is_there_menu = true;
            }
        }
        /*on fait la boucle tant qu'on a eu au moins un menu dans la boucle
        si on a pas eu de menus, c'est la fin.*/
        /*on fait cela pour le cas ou il y aurait plusieurs fois le meme menu
        dans la commande*/
        if(!is_there_menu) {
            fin = true;
        }
    }
    /*une fois qu'on a verifie tous les menus, on peut additionner le prix de
    chaque item restant*/
    for(Consommable a : commande.getItemsCommands()) {
        prix += a.getPrix();
    }
    return prix;
}
```

- Question 9:

Ajouter une méthode d'impression d'une commande.

- Reponse:

  ```java public void imprimerCommande(Commande c) { *// on reutilise la methode caclulerPrixCommade en rajoutant des sorties consoles pour afficher la commande*// int prix = 0; boolean fin = false; boolean is_there_menu; Commande commande = new Commande(); commande.addList(c.getItemsCommands()); System.out.println("_____"); System.out.println("_____Commande_____"); while(!fin) { is_there_menu = false; for(Menu m : menus) { if(commande.isThereMenu(m)) { commande.removeMenu(m); System.out.println(m); prix += m.getPrix(); is_there_menu = true; } } if(!is_there_menu) { fin = true; } } for(Consommable a : commande.getItemsCommands()) { prix += a.getPrix(); System.out.println("Consommable : " + a.getNom() + "\tPrix : " + a.getPrix()); }

  System.out.println("\n\n\nPrix total : " + prix); System.out.println("_____");

  }
  ```

```
___

## <center>V2 La diététique</center>


___


Nous souhaitons apporter à l'application une dimension diététique en permettant aux consommateurs de commander des menus dont ils co

```java
public interface Nutrition {    // S'applique à une portion moyenne
    public int getKcal();          // nombre de Kcal
    public float getGlucides(); // grammes de glucides
}
```

- Question 10:

Modifier en conséquence les classes Plat, Boisson et leurs sous-classes.

- Reponse:

```
 //on ajoute l'implementation
public class Plat implements Consommable, Nutrition{
  public String nom;
  public int prix;
    //on rajoute les variables kcal et glucides
  private int kcal;
  private float glucides;

    //on modifie le constructeur
  public Plat(String nom, int prix, int kcal, float glucides){
     this.nom = nom;
     this.prix = prix;
     this.kcal = kcal;
     this.glucides = glucides;
  }
  public int getPrix(){
    return prix;
  }
  public String getNom(){
    return nom;
  }
    //on rajoute les getters
  public float getGlucides() {
    return this.glucides;
  }
  public int getKcal() {
    return this.kcal;
  }
}
```

1. On doit rajouter l'implementation dans l'intitule de la classe
2. On doit modifier les constructeurs des classes et sous-classes
3. On rajoute les variables dans les classes
4. On rajoute les getters et on modifie les methodes toString()

- Question 11:

Ajouter la méthode `public void proposerMenu(int Kc, int epsilon)` à la classe `Carte`, affichant à l'écran tous les menus qui totalisent un nombre de Kcal égal à Kc (à epsilon Kcal près)

- Reponse:
```

```
public void proposerMenu(int Kc, int epsilon) {
    int kcal;
    //on regarde la valeur nutritionnelle de chaque menu
    for(Menu menu : menus) {
        kcal = menu.getKcal();
        //si il correspond, on l'affiche
        if(kcal<Kc+epsilon && kcal>Kc-epsilon) {
            System.out.println("Un menu adéquat est : ");
            System.out.println(menu);
        }
    }
}
```