Alexis BERTRAND
Guillaume BRUCHON

TP3 Java OS312

> Vous trouverez ci-joint le code de l'application.

Question 1

Nous avons remplacé la classe abstraite consommable par l'interface demandé, puis nous avons copié les attributs et les méthodes de la classe consommable dans Plat et Boisson. Enfin, nous avons remplacé les appels de super dans les constructeurs de Plat et Boisson par le code du constructeur de l'ancienne classe consommable et utilisé le mot clé `impléments` pour implémenter l'interface.

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


public class Plat implements Consommable{
    private String nom;
    private int prix;
    public Plat(String nom, int prix){
        this.nom = nom;
        this.prix = prix;
    }

    public String getNom(){ return this.nom; }
    public int getPrix(){ return this.prix; }
}
```

Question 2

> Voir fin de ce compte-rendu

Question 3

On ajoute chaque consommable au menu, et on indique le prix :

```
public Menu(int prix, Entrée e, PlatPrincipal p, Dessert d, Boisson b) {
    this.prix=prix;
    items = new ArrayList<Consommable>();
    items.add(e);
    items.add(p);
    items.add(d);
    items.add(b);
}
```

On parcours tous les consommables pour créer une string d'affichage :

```
public String toString(){
    String message = "Menu compose de ";
    for (Consommable i : items)
        message += i.toString() + ", ";
    message += "au prix de " + this.prix/100 + " euros";
    return message;
}
```

Question 5

La fonction `verifPrixMenu` vérifiant que le prix d'un menu est moins chère ou égal à celui des consommables pris un par un. Elle était déjà implémentée :

```
    private boolean verifPrixMenu(){
```

```
            int sommePrix = 0;
            for(Consommable c: this.items){
                sommePrix += c.getPrix();
            }
            return sommePrix >= this.prix && sommePrix > 0;
        }
```

Afin que menu génère une exception si son prix est trop élevé, on ajoute `throws Exception` dans le constructeur et on déclenche cette exception en cas d'erreur (vérification du prix avec `verifPrixMenu`):

```
    public Menu(int prix, Entrée 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);
        if (!verifPrixMenu())
            throw new Exception (this.toString());
    }
```

Maintenant, si l'on veut créer une nouvelle Exception, il faut créer une **nouvelle classe qui hérite d'Exception** :

```
    public class ExceptionPrixMenuIncorrect extends Exception {
        private static final long serialVersionUID = 1L;
        private String menu;
        public ExceptionPrixMenuIncorrect (String menu) {
            this.menu = menu;
        }
        public String getMenu() {
            return menu;
        }
        public String toString() {
            return ("Le prix de : " + menu + " est plus chère que chaque
    élément individuellement...");
        }
    }
```

et on déclenche cette nouvelle classe en cas d'erreur dans le menu :

```
    public Menu(int prix, Entrée 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);
    if (!verifPrixMenu())
        throw new ExceptionPrixMenuIncorrect (this.toString());
}
```

Question 6

On créé un tableau contenant tous les consommables (aussi bien entrée que plat ou dessert) et on itère dans ce tableau **pour vérifier que le consommable possède un nom unique**. Pour chaque élément du tableau, on vérifie que l'élément à insérer n'a pas le même nom. Si l'élément est déjà présent, on retourne `false` , sinon c'est `true` .

```
private boolean verifCarte(Consommable c){
    ArrayList<Consommable> tab = new ArrayList<Consommable>();
    tab.addAll(entrées);
    tab.addAll(platsPrincipaux);
    tab.addAll(desserts);
    tab.addAll(boissons);
    for (Consommable i : tab)
        if (i.getNom().compareTo(c.getNom()) == 0)
            return false;
    return true;
}
```

Question 7

Afin de **vérifier qu'un élément est bien présent dans la carte** avant de l'ajouter au menu, on se sert de la méthode `verifCarte` qui fait l'opération inverse. Quand `verifCarte` retourne `true` , on sait que l'élément n'est pas présent, c'est donc un `false` pour nous ; et inversement …
On va donc appeler `verifCarte` pour chaque élement du menu et vérifier que cet élément est bien présent.

```
private boolean verifMenu(Menu m){
    boolean retour = true;
    for (Consommable i : m.getItems()) {
        retour &= !verifCarte(i);
    }
    return retour;
}
```

Question 8

Afin de **calculer le prix d'une commande**, nous avons eut la démarche suivante :

- On regarde si la commande contient tout les élément d'un menu
    - Si c'est le cas :
        - On supprime de la liste temporaire de commande les éléments contenu dans le menu
        - On augmente le prix de la commande, de la valeur du menu
        - Et on recommence au début (indice=0), car il est possible que l'on ait de nouveaux menus (voir même 2 fois le même menu !)
    - Sinon, on ne fait rien et on passe au menu suivant
- Dès que l'on à fait autant d'itération que la taille restante du menu, sans trouver de nouveau menu, on sait qu'il n'y en a plus.
- On crédite donc du montant total, les consommables individuels restant qui n'appartiennent à aucun menu.

Avec cette méthode, on peut donc avoir plusieurs menu et même plusieurs fois le même menu. De plus on ne touche pas à la commande car on copie préalablement la liste de item afin de pouvoir les supprimer si on trouve un menu.

```
public int calculerPrixCommande(Commande c){
        ArrayList<Consommable> commande = c.getItemsCommandés();
        int indice = 0;
        int prix = 0;

        while ((indice++) < commande.size()) {
            for (Menu menu : menus) {
                if  (commande.containsAll(menu.getItems()) ) {
                    indice = 0;
                    prix += menu.prix;
                    for (Consommable i : menu.getItems())
```

```
                    commande.remove(i);
                break;
            }
        }
    }
    for (Consommable i : commande)
        prix += i.getPrix();

    return prix;
}
```

Tests

Afin d'effectuer des tests, nous avons tout d'abord modifié la méthode `afficherMenu`, car elle n'affichait pas les menu étant donné qu'un consommable n'a pas de `toString`. Pour cela, on utilise la méthode `forEach` des collections, permettant d'appliquer une fonction à chaque élément de la collection. Ici, on applique une fonction anonyme qui permet d'afficher le contenu du consommable.

```
public void afficherMenu(){
    System.out.print("Liste des entrées: ");
    entrées.forEach((Consommable c) -> {System.out.print(c.getNom()+",
");});
    ...
}
```

Ensuite avec la classe `TestRestaurant`, on vérifie que toutes nos méthodes fonctionnent. Pour cela, nous avons implémentés tout les cas.

> Voir les commentaires dans le code TestRestaurant .java qui explique ce qu'il doit se passer

On obtient le résultat suivant, qui correspond bien à ce qui est attendu :

```
--- Ajout des menus ---
Le prix de : Menu compose de Salade verte, Pizza Reine, Tiramisu, Eau,
au prix de 18 euros est plus chère que chaque élément
individuellement...
Menu compose de Salade verte, Pizza Reine, Tiramisu, Eau, au prix de 15
euros
Menu compose de Salade rouge, Pizza Reine, Tiramisu, Eau, au prix de 10
euros
```

```
--- Affichage de la carte et ajout des menus ---
Liste des entrées: Salade verte, Salade composée,
Liste des plats principaux: Pizza Reine, Pizza Margarita, Spaghetti à la
Bolognaise,
Liste des desserts: Tiramisu,
Liste des boissons: Eau,

Liste des menus:
-Menu compose de Salade verte, Pizza Reine, Tiramisu, Eau, au prix de 15
euros

--- Ajout commande ---
46
```

On retrouve ainsi :

- Le menu qui ne peut pas s'ajouter (car trop chère)
- Les consommables qui ont bien étés ajoutés à la carte (et il n'y a pas de doublons)
- Le seul menu qui a pu être ajouté (car pour l'autre tout les éléments n'existaient pas)
- Le prix de la commande contenant 2 Pizza reine, 2 Pizza margarita, 2 salade verte, 2 tiramisu et 2 eau, est de **46€** = 30 + 16
    - 2 * 8€ = 16€ pour les 2 Pizza margarita
    - 2 * 15€ = 30€ pour les 2 menus (Salade verte, Pizza reine, Tiramisu et Eau) ; et non pas le prix article par article.
      Le calcul du prix tiens donc compte des 2 formules présentes.