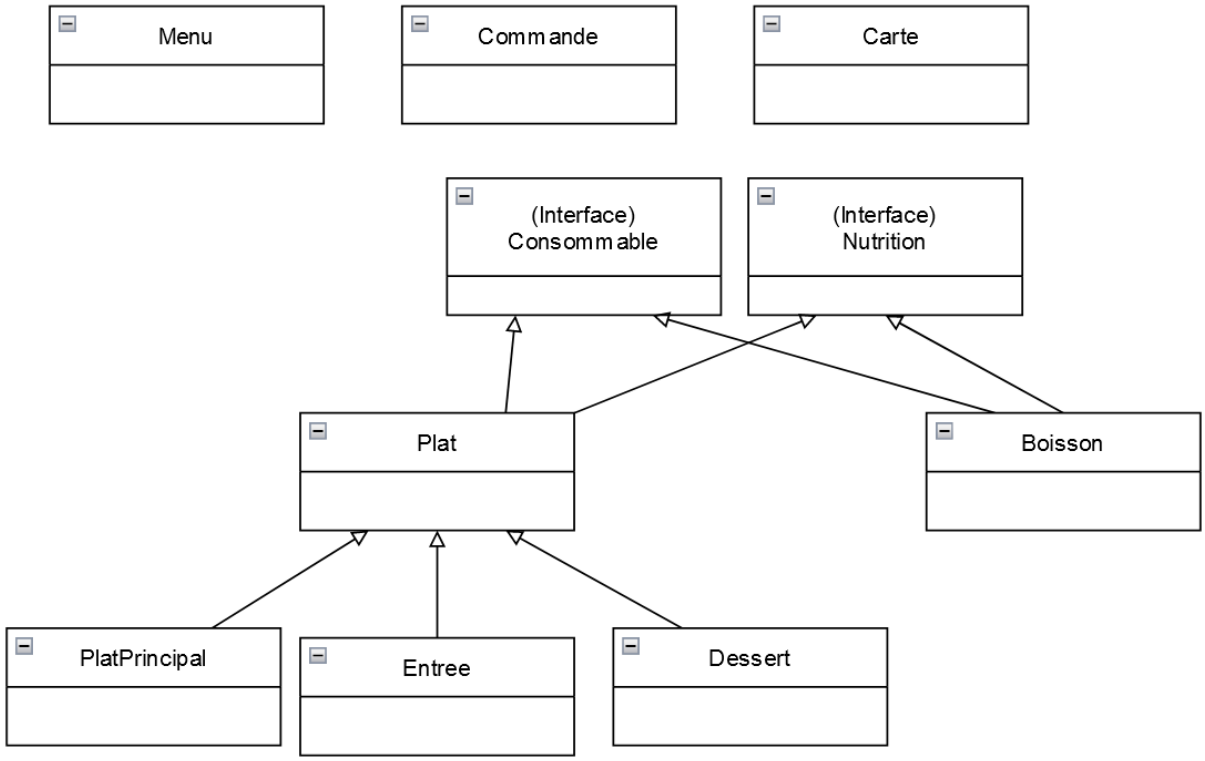
Eloi Donval
Martin Boulin
Loïc Savornin

CS312 - Compte-rendu TP3

1 – Diagramme de classe UML du projet

Pour se représenter plus facilement les différentes classes (et interfaces) du projets, nous avons fait un diagramme UML à l'aide de draw.io :



2 – Codes des questions

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();
}
```

Eloi Donval
Martin Boulin
Loïc Savornin

Modifier en conséquence les classes Plat et Boisson.

Nous ajoutons donc aux classes Plat et Boisson leurs attributs et modifions leurs constructeurs :

Classe Boisson :

```
    private String nom;
    private int prix; // en cents d'euros

    public Boisson(String nom, int prix, int volume) {
        this.nom = nom;
        this.prix = prix;
        this.volume = volume;
    }
```

Classe Plat :

```
    private String nom;
    private int prix; // en cents d'euros

    public Plat(String nom, int prix) {
        this.nom = nom;
        this.prix = prix;
    }
```

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.

Le main étant assez long, il est préférable d'aller le voir directement dans son fichier.

Question 3

Compléter le constructeur de la classe Menu.

```
public Menu(int prix, Entree e, PlatPrincipal p, Dessert d, Boisson b) throws Exception {
    this.prix = prix;
    items.add(e);
    items.add(p);
    items.add(d);
    items.add(b);
    if (verifPrixMenu()) {
        throw new Exception("ERREUR dans le prix lors de la creation du Menu");
    }
}
```

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").

Eloi Donval
Martin Boulin
Loïc Savornin

Classe Menu :

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

```
    public boolean verifPrixMenu() {
        boolean prixIncorect = false;
        int prixTotal = 0; //prix des consommables cummulés

        for (Consommable item : items) {
            prixTotal += item.getPrix();
        }
        prixIncorect = (this.prix <= 0 || prixTotal <= this.prix);

        return prixIncorect;
    }
```

Constructeur :

```
public Menu(int prix, Entree e, PlatPrincipal p, Dessert d, Boisson b) throws Exception {
    this.prix = prix;
    items.add(e);
    items.add(p);
    items.add(d);
    items.add(b);
    if (verifPrixMenu()) {
        throw new Exception("ERREUR dans le prix lors de la creation du Menu");
    }
}
```

Eloi Donval
Martin Boulin
Loïc Savornin

Question 6

Compléter la méthode verifCarte qui effectue cette vérification (dans Carte).

```
    // Vérifie qu'il n'y a pas d'homonymes dans la carte
    // utiliser equals et des iterateurs
    private boolean verifCarte(Consommable c) {
        String nomItem = c.getNom();
        Iterator<Consommable> iterateurBoisson = this.boissons.iterator();
        Iterator<Consommable> iterateurPlats =
this.platsPrincipaux.iterator();
        Iterator<Consommable> iterateurEntrees = this.entrees.iterator();
        Iterator<Consommable> iterateurDesserts = this.desserts.iterator();

        return verifNom(nomItem, iterateurBoisson) //on vérifie que l'homonyme
potentiel n'est dans aucune des listes (Boisson/Plats/Entrees/Desserts)
                && verifNom(nomItem, iterateurPlats)
                && verifNom(nomItem, iterateurEntrees)
                && verifNom(nomItem, iterateurDesserts);
    }
```

Question 7

Réaliser la méthode verifMenu qui effectue cette vérification. (dans Carte)

```
    // Vérifie que les plats et boissons du menu sont bien dans la carte
    private boolean verifMenu(Menu menu) {
        boolean verif = true;
        ArrayList<Consommable> items = menu.getItems();
        for (Consommable item : items) {
            verif = verif && (this.boissons.contains(item) // verif à 1 si
l'item est dans la carte)
                    || this.desserts.contains(item)
                    || this.platsPrincipaux.contains(item)
                    || this.entrees.contains(item));
        }

        return verif;
    }
```

Eloi Donval
Martin Boulin
Loïc Savornin

Question 8

Ajouter la méthode public int calculerPrixCommande(Commande c) à la classe Carte calculant le prix d'une commande selon le principe ci-dessus.

```
    public int calculerPrixCommande(Commande c) {
        int prix = 0;
        ArrayList<Consommable> itemsFinaux = new
ArrayList<Consommable>(c.getItemsCommandes());
        ArrayList<Menu> menusFinaux = new ArrayList<Menu>();

        for (Menu menu : this.menus) {// on regarde si chaque menu est présent
dans la commande
            boolean menuPresent = true;
            for (Consommable item : menu.items) {
                menuPresent = menuPresent &&
c.getItemsCommandes().contains(item);
            }
            if (menuPresent) {
                menusFinaux.add(menu);
                for (Consommable conso : menu.items) {
                    itemsFinaux.remove(conso);
                }
            }
        }
        for (Consommable item : itemsFinaux) {
            prix += item.getPrix();
        }
        for (Menu menu : menusFinaux) {
            prix += menu.getPrix();
        }

        return prix;
    }
```

Question 9

Ajouter une méthode d'impression d'une commande. (dans Carte)

```
    //Affiche tous les objets de la commande et son prix
    public void afficherRecapitulatifCommande(Commande commande) {
        String message = "Commande composée de : ";
        for (Consommable item : commande.getItemsCommandes()) {
            message += item.getNom() + ", ";
        }
        message += "au prix de " + calculerPrixCommande(commande) + "€";
        System.out.println(message);
    }
```

Eloi Donval
Martin Boulin
Loïc Savornin

Question 10

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

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

Nous ajoutons les attributs et modifions les constructeurs de ces classes :

```
    protected int kcal = -1; // nombre de Kcal
    protected float glucides = -1; // en grammes
```

```
    public Plat(String nom, int prix, int kcal, float glucides) {
        this.nom = nom;
        this.prix = prix;
        this.kcal = kcal;
        this.glucides = glucides;
    }
```

```
    public Boisson(String nom, int prix, int volume, int kcal, float glucides) {
        this.nom = nom;
        this.prix = prix;
        this.volume = volume;
        this.kcal = kcal;
        this.glucides = glucides;
    }
```

```
    public PlatPrincipal(String nom, int prix, int kcal, float glucides) {
        super(nom, prix, kcal, glucides);
    }
```

```
    public Dessert(String nom, int prix, int kcal, float glucides) {
        super(nom, prix, kcal, glucides);
    }
```

```
    public Entree(String nom, int prix, int kcal, float glucides) {
        super(nom, prix, kcal, glucides);
    }
```

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).

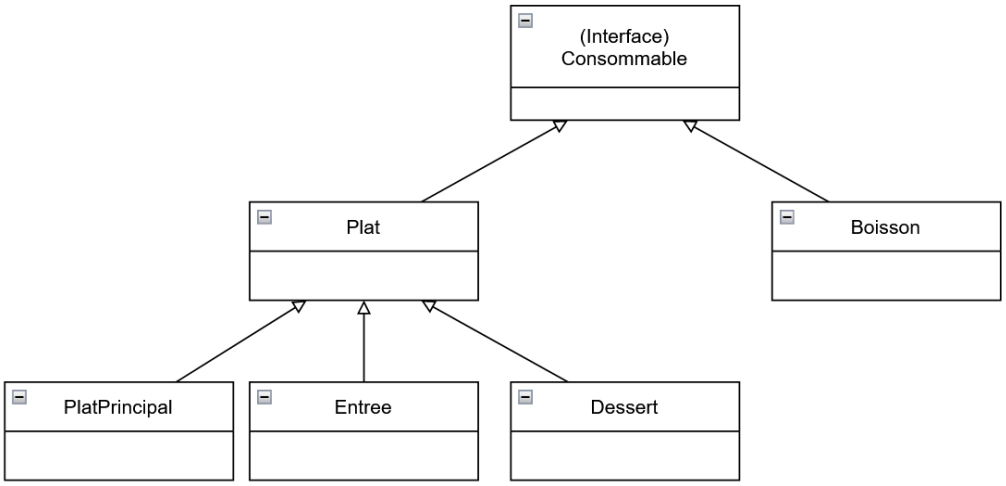
Nous avons eu des difficultés lié à l'héritage, aux interfaces, et à la définition de la classe Commande, sur cette dernière question, le code est donc incomplet, mais voici ce que nous avions commencés :

```
    public void proposerMenu(int Kc, int epsilon) {
        ArrayList<Menu> menus = new ArrayList<Menu>();
        String message;
        int kcal = 0;
        for (Menu menu : this.menus) {  //Il  faudrait utiliser des méthodes
de Nutrition alors que j'ai une liste de Consommable, je vois pas comment
faire
            // for (Nutrition consommable : (ArrayList<Nutrition>)
menu.getItems()) {
            // kcal += consommable.getKcal();
            // }
        }
        message = "Les menus ayant " + Kc + "±" + epsilon + " kcal sont :" +
menus;
        System.out.println(message);
    }
```

3 – Analyse classe abstraite versus interface

Pour cette application, il aurait été préférable de mettre une classe abstraite plutôt qu'une interface pour `Consommable`.

En regardant une partie de l'UML on peut voir que Consommable est la classe la plus haute :



Dans Plat et dans Boisson nous définissons les attributs nom et prix, et redéfinissons leurs getteurs. Nous avons donc exactement le même code copié collé dans les 2 classes, alors que si Consommable avait été une classe abstraite nous n'aurions pas eu à faire une telle chose (grâce à l'héritage des attributs&méthodes).

De même il aurait été plus simple de fusionner les classes Consommable et Nutrition, ce qui aurait permis de faire facilement la dernière question (n°11).