UEF4.3. Programmation Orientée Objet
EXAMEN FINAL- Corrigé
02 heures-Documents interdits

Exo1 (6pts)

Exercice 1 (6pts): Voici le code de 3 classes (dans 3 fichiers différents).

```
public class A {
 private int x = 2;
   public void m(int x) {
    this.x=this.x*x;
    System.out.print("A avec x="+this.x);
   }
}
public class B extends A {
private int x = 3;
 public void m(int x) {
    super.m(x);
    this.x=this.x+x;
   System.out.println("B avec x="+x);
 } }
```

```
public class Test {
 public static void
          main(String[]args) {
 A a = new A();
 B b = (B) a;
 b.m();
 }
}
```

Question 1 : Que provoque l'exécution du code de la classe Test ? Choisissez l'une de ces réponses ***en justifiant*** votre réponse.

(1) Ce code ne compile pas.
(2) Il affiche "A avec x=4".
(3) Il affiche "A avec x=4 B avec x=2"
Réponse 2pts= 1 la réponse 4+ 1pt l'explication

(4) Il y a une erreur à l'exécution. cast impossible car a référence un objet de type A

Question 2 : Nous modifions l'entête de la classe A comme suit :

```
class A implements Serializable
```

Qu'affiche l'exécution de la classe Test avec le nouveau code suivant :

```
import java.io.*;
public class Test {
public static void main(String[] args) {
ObjectInputStream in=null;
ObjectOutputStream out=null;
try {
 out = new ObjectOutputStream(
          new BufferedOutputStream(
          new FileOutputStream(
          new File("test.dat"))));
```

```
A a= new A();
a.m(3);
out.writeObject(a);
out.writeObject(new B());
out.close();

in = new ObjectInputStream(
          new BufferedInputStream(
          new FileInputStream(
          new File("test.dat"))));
```

```
((A)in.readObject()).m(2);
((B)in.readObject()).m(2);
in.close();
} catch (ClassNotFoundException e) {
        e.printStackTrace();
} catch (FileNotFoundException e) {
        e.printStackTrace();
} catch (IOException e) {
        e.printStackTrace();
}
}}
```

Réponse 2pts= 0,5*4

A avec x=6 A avec x=12 A avec x=4 B avec x=2

Question 3 : Nous modifions le code des classe B et Test comme suit :

```
class B extends A {
 private int x = 3;
 public void m(int z) {
  final int y=2;
  class In{
   private int x =1;
   public void ml(int k) {
        this.x+=k*y;
        B.this.x+=this.x;
    }
   }
  In i= new In();
  i.ml(z);
  System.out.println("B avec x="+x);
 }
}
```

```
public class Test {
  public static void main(String[] args) {
   A a = new B();
   a.m(2);
}
}
```

Qu'affiche l'exécution de la classe Test ?

Réponse 2pts B avec x=8

Exercice 2 (14 points)

On veut réaliser un programme permettant la gestion de recettes et de menus dans le cadre d'un régime alimentaire. Une recette a un nom, une durée de préparation en minutes, un degré de difficulté (très facile, facile, difficile) et une liste d'ingrédients dont chacun est défini par un aliment et une quantité. Chaque aliment est caractérisé par son nom, une unité de mesure (gramme ou litre) et le nombre de calories pour 100 unités de mesure. Certaines recettes se composent, en plus de leurs ingrédients, d'une ou plusieurs recettes de base à préparer au préalable (exemple: pour réaliser une tarte aux fraises, il faut d'abord préparer une pâte sablée et une crème pâtissière). Grâce à ce programme, l'utilisateur peut ajouter une nouvelle recette ou supprimer une recette existante. Il peut aussi composer et sauvegarder des menus journaliers. Pour ajouter un menu, il doit indiquer le nombre de calories à ne pas dépasser et sélectionner un certain nombre de recettes. Le programme peut alors valider le menu ou signaler à l'utilisateur que le nombre maximal des calories indiqué est dépassé.

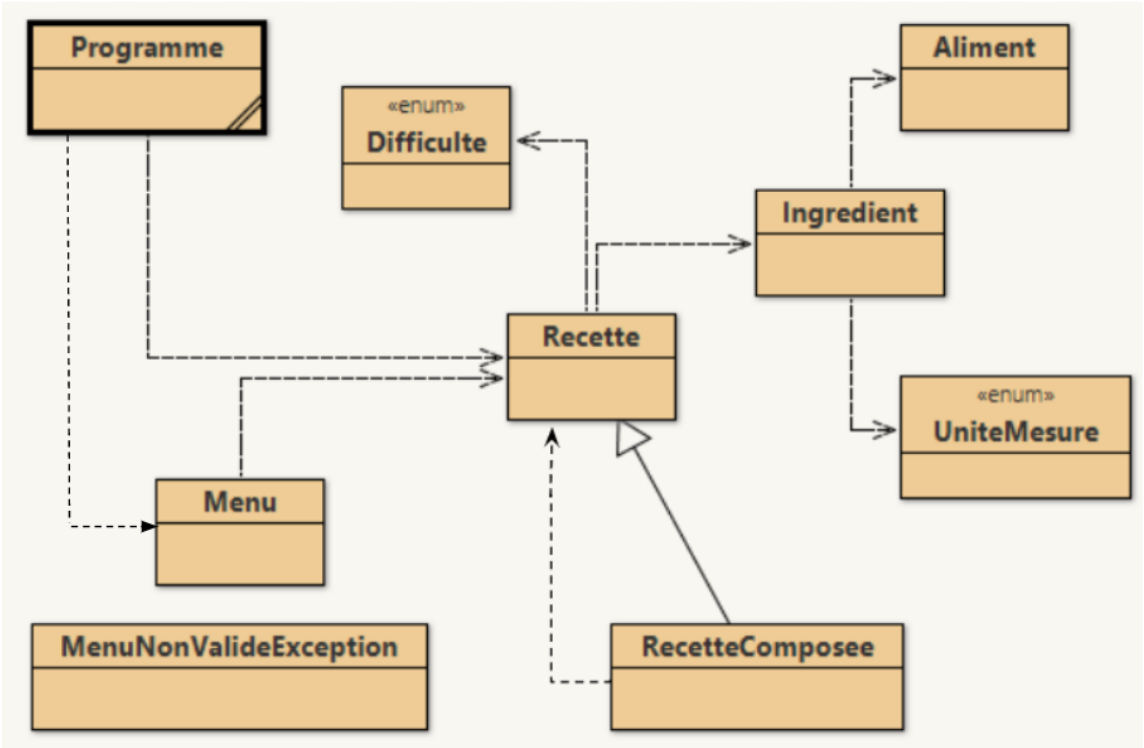
1. Proposez une conception orientée objet pour ce programme en traçant un diagramme des classes illustrant les relations entre elles (utilisation, héritage, implémentation d'interface), et en indiquant dans un tableau les attributs et les méthodes nécessaires (hormis les constructeurs, les getters et les setters). **//6.75 points**

   Abstraction : 0.25 *9 (nombre de classes) = 2.25

   Héritage : 0.5

   Utilisation : 0.25 (tout ou rien selon la sol car on comptabilise l'utilisation dans les tableaux à travers les attributs)

   3



Tableaux : 0.25* 15 = 3.75

Enum UniteMesure
GRAMME, LITRE **// 0.25**

Enum Difficulte
TRESFACILE, FACILE, DIFFICILE **// 0.25**

class Aliment	
Attribut // 0.25	**description**
private String nomAliment private UniteMesure unite	

private int nbCalories	
Methode	**Description**

class Ingredient	
Attribut // 0.25	**description**
private Aliment aliment private double quantite	
Methode // 0.25	**Description**
int calculerCalories()	Calcule le nombre de calories de l'ingrédient en fonction de l'Aliment et de la quantité

class Recette	
Attribut // 0.25	**description**
protected String nomRecette protected int duree Difficulte degreDifficulte Collection <Ingredient> listeDesIngredients	
Methode // 0.25	**Description**
public int calculerCalories()	Elle additionne les calories des ingrédients

class RecetteComposee extends Recette	
Attribut // 0.25	**description**
Collection<Recette> recettesDeBase	Les recettes de base qui entrent dans la composition de cette recette
Methode // 0.25	**Description**
public int calculerCalories()	redéfinition de la méthode héritée de Recette. Elle additionne les calories des ingrédients et celles des recettes de base.

class Menu

Attribut // 0.25	**description**
Collection <Recette> selectionDeRecettes	
Methode // 0.25	**Description**
public int calculerCalories ()	calcul la somme des calories des recettes qui le composent

class Programme	
Attribut // 0.25	**description**
Collection <Recette> toutesLesRecettes Collection <Menu> tousLesMenus	
Methode // 0.25 + // 0.25	**Description**
public void ajouterRecette (Recette r) public void supprimerRecette (Recette r) public void ajouterMenu (Menu r, int maxCalories) throws MenuNonValideException	La méthode ajouterMenu calcul la somme des calories d'un menu et lance une exception si celui-ci dépasse le maximum indiqué par le client

MenuNonValideException extends exception //0.25

2. Pour chacun des cas ci-dessous: **//3.75**
   - Donnez l'instruction java permettant de déclarer et instancier la collection la plus adéquate pour contenir les recettes.
   - Expliquez les conditions nécessaires pour le bon fonctionnement du programme et donnez le code java des méthodes nécessaires.

Cas 2.1. Il n'existe pas deux recettes ayant le même nom. **//0.75**

Réponse:

- instanciation: **//0.25**

```
Set <Recette> toutesLesRecettes = new HashSet<Recette>
```

- Conditions de bon fonctionnement: Redéfinir dans la classe Recette les méthodes hashCode et equals
- Code source des méthodes equals et hashCode: **//0.25*2**

```
public int hashCode(){
   return nom.hashCode();
   }
public boolean equals(Object autreRecette){
```

```
            if (nom.equals(((Recette)autreRecette).nom))
                return true ;
            else return false;
        }
```

Cas 2.2. Il n'existe pas deux recettes ayant le même nom et les recettes sont triées de la moins calorique à la plus calorique. **//2**

instanciation: **//0.25**

```
Set <Recette> toutesLesRecettes = new TreeSet<Recette>
```

Conditions de bon fonctionnement:

- Les classes Ingrédient, Recette et RecetteComposee doivent implémenter une méthode qui calcule le nombre de calories
- La classe Recette doit implémenter l'interface Comparable

```
public class Recette implements Comparable<Recette>
```
//0.25

- La classe Recette doit redéfinir la méthode compareTo de façon retourner un zéro si deux recettes ont le même nom. Si leurs noms sont différents, elle retourne la différence entre le nombre de calories.

Code source des méthodes nécessaires:

//Méthode calculerCalories de la classe Ingrédient **//0.25**

```
public int calculerCalories(){
          return aliment.getCalories()*quantite/100;}
              }
```

// Méthode calculerCalories de la classe Recette: **//0.25**

```
public int calculerCalories(){
             int somme = 0;
             for (int i = 0; i<ingredients.size(); i++)
          somme = somme + ingredients.get(i).calculerCalories();
             return somme;
             }
```

// Méthode calculerCalories de la classe RecetteComposee: **//0.5**

```
public int calculerCalories(){
   int sommeCalIng = 0;
   for (int i = 0; i<ingredients.size(); i++)
   sommeCalIng = sommeCalIng + ingredients.get(i).calculerCalories();
       int sommeCalRecDeBase = 0;
       for ( int j = 0; j<recettesDeBase.size(); j++)
sommeCalRecDeBase=sommeCalRecDeBase+recettesDeBase.get(j).calculerCalories()
;
       return  sommeCalIng+sommeCalRecDeBase;
}
```

// Méthode compareTo de la classe Recette **//0.5**

```
public int compareTo(Recette autreRecette){
```

```
if(nom.equals(autreRecette.nom))return 0;
else return calculerCalories() - autreRecette.calculerCalories();
}
```

Remarque: la méthode de parcours des collections dépend de son type dans cette solution, nous avons utilisé des ArrayList.

Cas 2.3. Les recettes sont classées par degré de difficulté, et pour chaque degré de difficulté, elles sont triées suivant l'ordre décroissant de la durée de préparation. **//1pt**

- Instanciation: Map<Difficulte, TreeSet<recette>> toutesLesRecettes = new HashMap<Difficulte, TreeSet<recette>> **//0.5**
- Conditions de bon fonctionnement: la classe Recette doit implémenter l'interface Comparable et redéfinir la méthode compareTo de façon à classer les recettes par ordre décroissant de la durée **//0.25**
- Code source de la méthode compareTo: **//0.25**

```
public int compareTo(Recette autreRecette){
return (autreRecette.duree-duree);
}
```

3. Donnez le code source de la méthode *ajouterMenu* qui reçoit un menu et un nombre de calories en paramètres, vérifie si le menu est valide avant de l'ajouter, et alerte l'utilisateur s'il ne l'est pas. **//1**

```
public boolean ajouterMenu(Menu menu, int maxCalories)//0.25
throws MenuNonValideException{//0.25
      if (menu.calculerCalories ()>maxCalories)  throw new
MenuNonValideException();//0.25
      else return (tousLesMenus.add(menu));//0.25
}
```

4. Pour améliorer notre programme, on désire guider l'utilisateur dans la composition de ses menus: dans le cas où un menu dépasse le nombre de calories fixé, le programme propose pour chaque recette sélectionnée, des substituts moins caloriques pour chaque ingrédient ou recette de base (exemple: dans la tarte aux fraises, remplacer la crème pâtissière par du yaourt). **//2.5 pts**

- Expliquez (sans donner le code source) ce que vous devez modifier dans votre conception afin de satisfaire ce besoin **//2**

- Rendre Recette abstract **//0.25**
- Créer une sous classe RecetteDeBase **//0.25**
- Créer une interface Remplaçable implémentée par Ingredient et RecetteDeBase et contenant la méthode **//0.5**
  public Collection<Remplaçable> getSubstitut();
- Ajouter dans les classes Ingredient et RecetteDeBase un attribut de type Collection<Remplaçable> représentant les substituts **//0.25*2**

- Implémenter la méthode *public Collection<Remplaçable> getSubstitut();* dans les deux classes **//0.25*2**

Remarque: les recettes n'étant pas modifiables, les tableaux peuvent être utilisés au lieu des collections.

- Donnez le nouveau diagramme des classes. **//0.5 (juste pour le schéma car toutes les modification ont été expliquées plus haut)**