UNIVERSITE GRENOBLE ALPES
L3 MIASHS

INF F5 — PROG. AVANCEE ET STRUCTURES DE DONNEES

Session 1 — Janvier 2017

Tous documents autorisés — durée 3 heures
Le barème indiqué est indicatif et peut légèrement varier lors de la correction.

Suivi diététique

Nous nous intéressons à la programmation d'une partie d'une application de suivi diététique. Il s'agit plus précisément de quantifier les apports énergétiques d'aliments entrant dans la composition de plats qui constituent à leur tour des repas. Nous nous limitons aux apports en énergie mesurés en kilocalories (kcal), et en glucides, lipides et protides mesurés en grammes (g) et ne tenons pas compte des apports en vitamines ou en autres nutriments tels que les sels minéraux ou les oligo-éléments. A des fins de simplification, la seule mesure de quantité utilisée est le gramme (g) pour les quantités même lorsqu'il s'agit de quantités liquides habituellement mesurées en millilitres. Nous ignorons enfin, toujours pour simplifier ce sujet, les transformations (perte d'eau, altérations chimiques, …) pouvant intervenir dans la préparation et la cuisson des plats.

Nous distinguons parmi les aliments simples les épices, pour lesquelles nous admettons que l'apport en énergie, et en glucides, lipides et protides est négligeable, des autres aliments pour lesquels ces quatre valeurs doivent être connues pour une ration de 100g. Les plats sont des aliments élaborés qui sont représentés dans le système par les ingrédients les composant : on se propose de baser cette représentation sur l'utilisation d'une *Map* dont les clés sont des aliments et les valeurs des quantités exprimées en g. Il est remarquable qu'un plat puisse éventuellement admettre dans sa composition d'autres plats et non uniquement des aliments de base. Enfin nous considérons des repas constitués de plusieurs plats et d'une boisson d'accompagnement.

Etant donné que la prise en compte des apports en glucides est tout à fait similaire à la prise en compte des apports en lipides et en protides, il n'est demandé dans les questions suivantes de ne se préoccuper que des méthodes relatives au calcul des apports en glucides.

1 Classe `AlimentAbstrait` (2,5 points)

```
package dietetique;

public abstract class AlimentAbstrait implements Comparable<AlimentAbstrait> {
	private String nom;

	public AlimentAbstrait() {…}
	public AlimentAbstrait(String nom) {…}

	public String getNom() {…}
	public abstract int getEnergie();
	public abstract double getGlucides();
	public abstract double getLipides();
	public abstract double getProtides();

	public int compareTo(Object o) {…}

	public String toString() {…}
}
```

La classe `AlimentAbstrait` permet de représenter des aliments de manière abstraite. Cette classe admet comme sous-classes les classes `Epice`, `Aliment` et `Plat` (voir questions 2, 3 et 4).

1.1 Donner la définition des constructeurs de cette classe.

1.2 Donner la définition de l'accesseur en lecture `getNom`.

1.3 Donner la définition de la méthode `compareTo` : un aliment abstrait est considéré inférieur (respectivement équivalent, supérieur) à un autre si son nom est inférieur (respectivement équivalent, supérieur) au nom de l'autre.

1.4 Donner la définition de la méthode `toString` : à ce niveau d'abstraction, la représentation textuelle d'un aliment se limite au nom de celui-ci.

2 Classe `Epice` (1,5 points)

```
package dietetique;

public class Epice extends AlimentAbstrait {

	public Epice(String nom) {…}

	public int getEnergie() {…}
	public double getGlucides() {…}
	public double getLipides() {…}
	public double getProtides() {…}
}
```

La classe `Epice` permet de représenter des épices, c'est-à-dire des aliments pour lesquels les apports sont considérés nuls.

2.1 Donner la définition du constructeur de cette classe.

2.2 Donner la définition des méthodes `getEnergie` et `getGlucides` de cette classe qui renvoient des valeurs nulles.

3 Classe `Aliment` (3 points)

```
package dietetique;

public class Aliment extends AlimentAbstrait {
	private int energie;    // apport en kcal pour 100g
	private double protides; // apport en g pour 100g
	private double lipides;  // apport en g pour 100g
	private double glucides; // apport en g pour 100g

	public Aliment(String nom, int energie, double glucides, double lipides,
                                                                double protides) {…}

	public int getEnergie() {…}
	public double getGlucides() {…}
	public double getLipides() {…}
	public double getProtides() {…}

	public String toString() {…}
}
```

La classe `Aliment` permet de représenter des aliments simples pour lesquels on connaît les apports en énergie, et en glucides, lipides et protides. Ces apports sont donnés pour une ration de 100g.

3.1 Donner la définition du constructeur de cette classe. Si la somme des apports en glucides, lipides et protides est supérieure à 100g, une exception `IllegalArgumentException` doit être levée.

3.2 Donner la définition des méthodes `getEnergie` et `getGlucides` de cette classe.

3.3 Donner la définition de la méthode `toString` qui doit retourner une valeur similaire aux exemples suivants :

```
"beurre(711 kcal, glucides : 0.5g, lipides : 83.0g, protides : 0.7g, pour 100g)"
"lait(45 kcal, glucides : 1.6g, lipides : 3.2g, protides : 4.5g, pour 100g)"
"farine(310 kcal, glucides : 11.7g, lipides : 2.0g, protides : 61.3g, pour 100g)"
"pâtes(375 kcal, glucides : 75.5g, lipides : 12.6g, protides : 1.4g, pour 100g)"
"vin rouge(65 kcal, glucides : 0.15g, lipides : 0.0g, protides : 0.0g, pour 100g)"
```

4 Classe `Plat` (8 points)

```
package dietetique;
import java.util.*;

public class Plat extends AlimentAbstrait {
	private Map<AlimentAbstrait,Double> ingredients;
	private int poids;

	public Plat(String nom) {…}
	public Plat(String nom, int nbIngredients) {…}

	public boolean add(AlimentAbstrait aliment, double quantite) {…}

	public int getTotalEnergie() {…}
	public double getTotalGlucides() {…}
	public double getTotalLipides() {…}
	public double getTotalProtides() {…}

	public int getEnergie() {…}
	public double getGlucides() {…}
	public double getLipides() {…}
	public double getProtides() {…}

	public String toString() {…}
	public String composition() {…}
}
```

La classe `Plat` permet de représenter des aliments élaborés à partir d'autres aliments. Les aliments entrant dans la composition d'un plat ainsi que leurs quantités sont référencés dans la `Map ingredients`. L'attribut `poids` est destiné à donner le poids d'une part de ce plat, ce qui correspond à la somme des quantités indiquées pour chaque ingrédient.

4.1 Donner la définition des constructeurs de cette classe. Bien que ce nombre puisse toujours augmenter, le paramètre `nbIngredients` permet de spécifier le nombre d'ingrédients initialement prévu lors de la création d'un plat. Si ce nombre est omis, la valeur par défaut est 10. Si ce nombre est négatif, une exception `IllegalArgumentException` doit être levée.

4.2 Donner la définition de la méthode `add` qui permet d'ajouter un ingrédient à ce plat en en spécifiant la quantité. Si cette quantité est négative ou nulle, une exception `IllegalArgumentException` doit être levée. Si l'ingrédient est déjà présent, l'ajout est impossible et la méthode retourne `false`, elle retourne `true` si l'ajout a pu avoir lieu.

4.3 Donner la définition des méthodes `getTotalEnergie` et `getTotalGlucides` de cette classe. Ces méthodes retournent respectivement les apports en énergie et en glucides d'une part du plat (dont le poids peut être différent de 100g).

4.4 Donner la définition des méthodes `getEnergie` et `getGlucides` de cette classe. Ces méthodes retournent respectivement les apports en énergie et en glucides d'une ration de 100g du plat (dont le poids d'une part peut être différent de 100g).

4.5 Donner la définition de la méthode `toString` qui doit retourner une valeur similaire aux exemples suivants (les apports correspondent à ceux d'une part et non d'une ration de 100g) :

```
"pâtes au beurre(poids : 110g, 446 kcal, glucides : 75.55g, lipides : 20.9g,
protides : 1.47g)"
"béchamel(poids : 553g, 540 kcal, glucides : 4.37g, lipides : 49.4g, protides :
28.91g)"
"pâtes béchamel(poids : 120g, 972 kcal, glucides : 80.33g, lipides : 67.24g,
protides : 33.37g)"
```

4.6 Donner la définition de la méthode `composition` qui doit retourner une valeur similaire aux exemples suivants :

```
"Composition de pâtes au beurre :
pâtes : 100.0g
beurre : 10.0g"
"Composition de béchamel :
beurre : 40.0g
farine : 10.0g
sel : 2.0g
poivre : 1.0g
lait : 500.0g"
"Composition de pâtes béchamel :
pâtes : 100.0g
béchamel : 20.0g"
```

5 Classe `Repas` (5 points)

```
package dietetique;
import java.util.*;

public class Repas {
	private List<Plat> plats;
	private Aliment boisson;
	private int qteBoisson;

	public Repas(Aliment boisson, int quantite) {…}

	public void add(Plat p) {…}

	public String toString() {…}
}
```

La classe `Repas` permet de représenter des repas composés de plats et accompagnés d'une boisson dont on spécifie la quantité à la création. Les plats composant un repas sont référencés dans une liste. Les apports d'un repas sont la somme des apports de chaque plat le constituant auxquels s'ajoutent les apports de la boisson l'accompagnant.

5.1 Donner la définition du constructeur de cette classe. Si la quantité de boisson est négative, une exception `IllegalArgumentException` doit être levée. Si la quantité de boisson est nulle, la boisson doit être `null`, sinon une exception `IllegalArgumentException` doit être levée.

5.2 Donner la définition de la méthode `add` qui permet d'ajouter un plat à ce repas. Un plat peut entrer plusieurs fois dans la constitution d'un repas.

5.3 Donner la définition de la méthode `toString` qui doit donner un résultat similaire aux exemples suivants :

```
"Repas :
pâtes beurre
pâtes béchamel
Boisson : vin rouge, 200g
(1548 kcal, glucides : 156.18g, lipides : 88.14g, protides : 34.84g)"
"Repas :
pâtes beurre
Pas de boisson
(446 kcal, glucides : 75.55g, lipides : 20.9g, protides : 1.47g)"
```

Rappels et indications

- Certaines valeurs ont été arrondies à 2 chiffres décimaux dans la transcription des exemples, il n'est pas demandé de programmer ces arrondis.
- Le constructeur `Double(double d)` permet de construire un `Double` dont la valeur correspond à `d`. La méthode `doubleValue()` de la classe `Double` permet d'obtenir la valeur du `Double` considéré sous la forme d'un `double`.
- `IllegalArgumentException` est une sous-classe de `RunTimeException`.