TD2

Idil SAGLAM Groupe 4 Numéro étudiant: 22015094

1. Manipuler des confitures

Question 1 :

Écrivez la classe `Confiture` avec un constructeur public adapté.

Réponse 1 :

Nous créons une classe `Confiture` avec des attributs privés`fruit` de type `String`, `proportion` de type `int` et `cal` de type `int`. Nous créons aussi le constructeur dans la classe `Confiture` qui prendre des paramètres dans le même ordre donnée en question.

```
public class Confiture {
    private String fruit;
    private int proportion;
    private int cal;

    Confiture(String fruit, int proportion, int cal){
        this.fruit = fruit;
        this.proportion = proportion;
        this.cal = cal;
    }
}
```

Question 2 :

Écrivez un deuxième constructeur qui ne prend en argument que la nature du fruit et le nombre de calories ; la proportion sera initialisée à 50. (Pensez que vous pouvez réutiliser le premier constructeur en fixant un paramètre)

Réponse 2 :

Nous créons un deuxième constructeur avec deux paramètres `nature` de type `String` et `cal` de type `int`. Ce constructeur va initialiser la valeur de l'attribut `proportion` à 50 par défault.
Nous utilisons le mot-clé `this` pour faire appel à un autre constructeur de la même classe.

```
Confiture(String nature,int cal){
    this(nature, 50, cal);
}
```

Question 3 :

Écrivez une méthode publique d'objet (c.à.d non statique) `description()` et qui renvoie une chaîne de caractères le décrivant (par exemple : "Confiture de fraise, 50% de fruit, 120 calories aux 100 grammes").

Réponse 3 :

```
public String description(){
    return String.format(
        "Confiture de %s, %d%% de fruit, %d calories aux 100 grammes.",
        this.fruit,
        this.proportion,
        this.cal
    );
}
```

Question 4 :

Dans une méthode `main` située dans une nouvelle classe `Test`, créez un objet de type `Confiture` et affichez sa description.

Réponse 4 :

Nous créons une classe `Test` où nous créons ainsi la méthode `main`. Dans le méthode main nous créons une variable de `confiture` de type `Confiture` et nous affichons sa description en utilisant la méthode `description` de la classe `Confiture`.

```
public class Test {
    public static void main(String[] args) {
        Confiture confiture = new Confiture("Fraise", 50, 120);
        System.out.println(confiture.description());
    }
}
```

Question 5 :

Dans la classe `Confiture`, écrivez une méthode publique d'objet qui prend en argument une quantité en grammes, et donne le nombre de calories correspondant à cette quantité pour cette confiture. (Il faut simplement faire un calcul qui respecte les proportions)

Réponse 5 :

Dans la classe `Confiture` nous créons la méthode `calculerCal` qui prend un argument de type `int` et qui a comme type de retour `int`. Pour garder les proportions, nous calculons d'abord les calories par gramme et nous le multiplions par la valaur du paramètre `gramme` de ce méthode.

```
public int calculerCal(int gramme){
    return this.cal / 100 * gramme;
```

```
}
```

Question 6 :

Écrivez une méthode de signature `public boolean egal(Confiture c)` qui s'adresse à une confiture courante et qui regarde si oui ou non elle a les mêmes attributs que la confiture `c`. (pour savoir si deux objets `s1` et `s2` de type String sont égaux, utilisez de préférence l'expression `s1.equals(s2)` qui est fournie par java et retourne un booléen)

Réponse 6 :

Nous créons un méthode `egal` pour tester l'égalité de deux objets `Confiture`. Pour que deux objets `Confitures` soient égaux, il faut que les trois attributs des deux soient tous égaux entre eux-même.

```
public boolean egal(Confiture c){
    return (
        c.fruit.equals(this.fruit) &&
        c.proportion == this.proportion &&
        c.cal == this.cal
    );
}
```

Question 7 :

On écrit le bout de code suivant situé dans la méthode `main` de la classe `Test`. Quelles lignes ne compilent pas, que produisent les autres ?

```
Confiture c1 = new Confiture ("fraise", 50 , 120);
Confiture c2 = new Confiture ("fraise", 50 , 120);
System.out.println(c1.egal(c2));
System.out.println(c1 == c2 ;
System.out.println(c1.fruit);
```

Réponse 7 :

5éme ligne des instructions donnés ne compile pas parce que `fruit` est un attribut privés qui a une portée de la classe elle-même. Le fait d'appeler un attribut privé en dehors de sa porté (en dehors de la classe qu'il est défini) ne compliera pas.

Question 8 :

On voudrait que l'attribut `fruit` ne puisse pas être modifié, même par une méthode de la classe `Confiture`; comment faire ?

Réponse 8 :

Pour pouvoir déclarer un attribut que nous ne pouvons pas modifier (inmutable), il faut qu'on ajoute `final` devant sa déclaration dans la classe.

Question 9 :

Écrivez une méthode qui retourne la valeur de l'attribut `fruit`. Écrivez-en une qui permet de modifier l'attribut `cal`. De quelles familles sont ces méthodes ?

Réponse 9 :

Nous écrivons les deux méthodes `getFruit` et `setCal` pour retourner la valeur de la variable `fruit` et pour changer la valeur de la variable `cal` dans la classe `Confiture` comme suivant:

```
public String getFruit(){
    return this.fruit;
}

public void setCal(int cal){
    this.cal = cal;
}
```

La première méthode `getFruit` sera une méthode `getter` (de la famille getteurs), la deuxième méthode `setCal` sera une méthode `setter` (de la famille `setteur`).

Question 10 :

En fait la valeur calorique dépend principalement de la quantité de sucre, qui est de 387 Kca pour 100 g, la valeur calorique du fruit est négligeable. Stockez cette valeur dans une variable adéquate

Réponse 10 :

Nous allons ajouter un attribut `sucre` de type `int` dans la classe `Confiture`. Nous changons aussi au premier contructeur (`Confiture(String fruit, int proportion, int cal)`) la ligne suivante pour calculer le quantité de sucre et initialiser ce nouveau attribut :

```
this.sucre = 387 * this.proportion /100;
```

Question 11 :

Écrivez un modifieur de proportion en précisant son domaine d'utilisation.

Réponse 11 :

Nous créons une méthode `setProportion` qui change la valeur de l'attribut privé `proportion` de la classe `Confiture`. Cette méthode peut être utile en cas de modification de quantité fruit dans un confiture.

```
public void setProportion(int proportion){
    this.proportion = proportion;
}
```

1.2 Mettre les confitures en pots

Question 1 :

Écrivez la classe Pot avec un constructeur public adapté

Réponse 1 :

Nous créons une classe `Pot` qui a comme attribut `confiture` de type `Confiture` qui indique le confiture dans le pot et `quantiteConfiture` de type `int` qui indique le quantité de confiture dans le pot.

```
public class Pot {
    public Confiture confiture;
    public int quantiteConfiture;
}
```

Question 2 :

Écrivez une méthode publique `description` qui renvoie une chaîne de caractères le décrivant. Remarquez qu'il n'y a pas d'ambiguïté avec la méthode description de `Confiture`, et que vous pouvez l'utiliser pour définir celle-ci.

Réponse 2 :

Nous créons la méthode `description()` dans la classe `Pot` comme suivant:

```
public String description(){
    return String.format(
        "Pot de %d grammes contient %s",
        this.quantiteConfiture,
        this.confiture.description()
    );
}
```

Cette méthode ne posera aucun problème avec celle de la classe `Confiture` comme elles ne sont pas définis dans la même classe.

Question 3 :

On veut numéroter les pots de confitures, à partir de 1, dans l'ordre de leur création. Cette numérotation doit se faire de manière transparente, c'est-à-dire que l'utilisateur n'aura pas à intervenir : les numéros seront affectés de manière automatique à la création des pots. Mettez en place ce mécanisme.

Réponse 3:

Dans la classe `Pot` nous ajoutons un attribut statique `COUNTER` de type `int` qui a comme but de compter la création des pots. Et le fait que ça soit statique, permet de garder la valeur de la variable pendant tout l'éxécution du code. Ainsi nous ajoutons un attribut non statique `id` de tpy `int` qui signifie l'identifiant du `Pot`.

```
private static int COUNTER = 0;
final int id&
```

ainsi dans le constructeur; nous ajoutons les lignes suivantes:

```
COUNTER++;
this.id = COUNTER;
```

nous incrémentons la valeur de l'attribut `COUNTER` et nous initialisons la valeur de l'attribut `id` avec la valeur de `COUNTER`.

Question 4 :

Écrivez une méthode qui retourne le dernier numéro attribué. Cette méthode est-elle statique ou non statique ?

Réponse 4 :

Dans le classe `Pot` nous définissons une méthode `dernierNumero()` ce méthode doit être non statique parce qu'il ne dépend pas d'un pot, nous pouvons même l'appeler avant la création d'aucun pot.

```
public static int dernierNumero(){
    return COUNTER;
}
```

Question 5 :

Dans votre classe Test créez plusieurs instances de Pot, affichez leurs descriptions, puis affichez le dernier numéro attribué.

Réponse 5 :

Dans la classe `Test` nous ajoutons les lignes suivantes:

```
Pot pot = new Pot(new Confiture("Fraise",50,120),50);
System.out.printf("Dernier numéro de pot après création d'un premier pot
%d\n", Pot.dernierNumero());
Pot pot2 = new Pot(new Confiture("fraise",50,120),20);
```

```
System.out.printf("Dernier numéro de pot après création d'un second pot:
%d\n", Pot.dernierNumero());
Pot pot3 = new Pot(new Confiture("Orange",90,1040),40);
System.out.printf("Dernier numéro de pot après création d'un troisième
pot: %d\n", Pot.dernierNumero());
```

2. Témperature

Question 1 :

Définissez une classe `Temperature`, décrite par un double représentant la température, et un `String` représentant l'unité.

Réponse 1 :

Nous créons une class `Temperature` qui a comme attribut `temp` de type `double` et `unite` de type `String`.

```
public class Temperature {
    public double temp;
    public String unite;
}
```

Question 2 :

Définissez un constructeur sans arguments qui lorsqu'il est utilisé produit un objet Temperature à zéro Kelvin.

Réponse 2 :

Dans la classe `Temperature` nous créons le constructeur suivant.

```
Temperature(){
    this.temp = 0;
    this.unite = "Kelvin";
}
```

Question 3 :

Définissez un deuxième constructeur prenant en argument un `double` et un `String` (Si l'unité n'est pas reconnue elle sera interprétée en Kelvin).

Réponse 3 :

Dans la classe `Temperature` nous créons un constructeur qui prends deux paramètres. Pour unifier la casse de l'unité nous initialisons la valeur de l'attribut `unite` à l'aide d'un `switch case`.

```
Temperature(double temp, String unite){
    this.temp = temp;
    switch(unite.toLowerCase()){
        case "celcius":
            this.unite="Celcius";
            break;
        case "fahrenheit":
            this.unite="Fahrenheit";
            break;
        default:
            this.unite = "Kelvin";
            break;
    }
}
```

Question 4 :

Définissez un troisième constructeur prenant en argument une Temperature et réalisant une copie de celle-ci.

Réponse 4 :

Dans la classe `Temperature` nous créons une troisième constructeur comme suivant:

```
Temperature(Temperature t) {
    this.temp = t.temp;
    this.unite = t.unite;
}
```

Question 5 :

Définissez des méthodes permettant d'afficher et de modifier chaque élément d'une Temperature (sans vous poser de questions de conversions, c'est abordé dans la suite).

Réponse 5 :

Dans la classe `Temperature` nous créons les méthodes suivante:

```
public double getTemp() {
    return temp;
}

public String getUnite() {
    return unite;
}

public void setUnite(String unite) {
    this.unite = unite;
```

```
}

public void setTemp(double temp) {
    this.temp = temp;
}
```

Question 6 :

Définissez une méthode privée `conversionKC`, non statique, produisant un nouvel objet `Temperature`. Lorsque this est bien en Kelvin le résultat sera sa conversion en Celsius, sinon elle ne produira pas de nouvel objet. On rappelle la formule TC = TK – 273.15 .

Réponse 6 :

Dans la classe `Temperature` nous créons une méthode non-statique `conversionKC` comme suivant:

```
private Temperature conversionKC(){
    if(this.unite.equals("Kelvin")){
        return  new Temperature(this.temp - 273.15,"Celsius");
    }
    return null;
}
```

Nous vérifions d'abord que l'unité est de type `Kelvin`, si c'est le cas nous appliqons la formule en créant un nouveau objet de type `Tempaerature`.

Question 7 :

Supposons que l'on ait écrit suffisamment de méthodes de conversions sur le modèle de la précédente (on rappelle par exemple la formule TF = 9/5∗TC +32). Écrivez une méthode read qui prend en argument une unité et renvoie la valeur numérique d'un objet Temperature dans l'unité spécifiée en argument.

Réponse 7 :

Pour convertir Fahrenheit en Celcius, il faut appliquer la formule donnée en question. Comme `unité` est un variable de type `String`,pour voir l'égalite, nous devons utiliser la méthode `String.equals(String2)` qui renvoie `vrai` si les deux variables (String et String2) sont égaux et qui renvoie `false` au contraire. Nous vérifons d'abord que l'unité est bien égale à `Fahrenheit` si c'est le cas nous appliqons le formule de conversion. Sinon nous retournons `null`.

```
private Temperature conversionFC(){
    if(this.unite.equals("Fahrenheit")){
        return new Temperature(5./9 * (this.temp - 32),"Celsius";
    }
    return null;
 }
```

La méthode `read` qui prend un paramètre `unite` de type `String` et qui retourne la valeur numérique dans l'unité passé en paramètre s'écrit comme ceci:

```
public double read(String unite){
    if(this.unite.equals(unite)){
        return this.temp;
    }
    if(unite.equals("Celcius") && this.unite.equals("Fahrenheit")){
        return conversionFC().temp;
    }
    if(unite.equals("Celcius") && this.unite.equals("Kelvin")){
        return conversionKC().temp;
    }
    if(unite.equals("Fahrenheit") && this.unite.equals("Kelvin")){
        return conversationKF().temp;
    }
    if(unite.equals("Kelvin") && this.unite.equals("Fahrenheit")){
        return conversationFK().temp;
    }
    if(unite.equals("Celcius") && this.unite.equals("Fahrenheit")){
        return conversationCF().temp;
    }
    return conversationCK().temp;
}
```

Pour cela dans la classe `Temperature` nous définissions quatre méthodes privées de conversions `conversionKF()`, `conversionFK()`, `conversionCK()` et `conversionCF()` comme suivant:

```
private Temperature conversationFK(){
    if(this.unite.equals("Fahrenheit")){
        return new Temperature((((this.temp-32) * 5) / 9) +
273.15,"Kelvin");
    }
    return null;
}

private Temperature conversationKF(){
    if(this.unite.equals("Kelvin")){
        return new Temperature((this.temp-273.15)*1.8000,"Fahrenheit");
    }
    return null;
}
private Temperature conversationCF(){
    if(this.unite.equals("Celcius")){
        return new Temperature((this.temp*1.8000+32.00),"Fahrenheit");
    }
    return null;
}

private Temperature conversationCK(){
```

```
    if(this.unite.equals("Celcius")){
        return new Temperature(this.temp+273.15,"Kelvin");
    }
    return null;
}
```

Question 8 :

Comment tester l'égalité de deux objets Temperatures ?

Réponse 8 :

Pour tester l'égalité des deux objets `Temperature` nous redéfinissons la méthode `equals(Object o)` de la classe `Object`. Pour ce faire, nous déclarons la méthode `public boolean toString(Object o)` avec un décorateur `@Override`. Ce méthode peut s'écrire comme suivant:

```
@Override
public boolean equals(Object o){
    if(o instanceof Temperature){
        Temperature to = (Temperature)o;
        return (to.unite.equals(this.unite) && to.temp == this.temp);
    }
    return false;
}
```

Nous testons d'abord si l'objet de type `Object` passé en paramètre est aussi un instance de `Temperature`. Si ce n'est pas le cas, il n'y a aucun moyen que ces deux objets soient égaux. Si c'est le cas nous testons tous les attributs les deux classes pour vérifier tous les attributs soient égaux entre eux.

Question 9 :

Définir une méthode plusBasseQue permettant de comparer deux Temperatures.

Réponse 9 :

Dans la classe `Temperature` nous écrivons la méthode `plusBasseQue` comme suivant:

```
public boolean plusBasseQue(Temperature t) {
    if(this.unite.equals(t.unite)) {
        return(this.temp < t.temp);
    }
    return (this.read(t.unite) < t.temp);
}
```

Nous testons d'abord si les deux unités sont égaux. Si c'est le cas nous faisons un simple comparaison des valeurs numériques des deux temperatures. Si ce n'est pas le cas, nous les convertissons à `Celcius` et nous faisons la comparaison des valeurs numériques.