

BankAccount

Question 1 : À quoi correspond l'attribut `balance` ? Pourquoi a-t-on utilisé le mot clé `private` afin de le rendre inaccessible à l'extérieur de la classe ?

L'attribut `balance` correspond au crédit du compte. C'est un attribut d'instance car chaque compte a son propre crédit. On utilise le mot-clé `private` car on ne souhaite pas que l'utilisateur de la classe puisse augmenter ou diminuer le crédit d'un compte sans passer par les méthodes `deposit` et `withdraw`. En effet, La valeur de crédit est quelque chose que l'on souhaite contrôler et ne doit pas être accessible directement par l'utilisateur de la classe.

Question 2 : Quel est le rôle du mot-clé `this`. Est-ce que son utilisation est indispensable ?

Le mot-clé `this` sert à indiquer qu'on manipule des attributs d'instance. Il n'est pas indispensable car il n'y a pas de variables ou arguments dans les méthodes ayant le même nom que les attributs manipulés. Il n'y a donc pas d'ambiguïté dans le code et le mot-clé `this` peut être enlevé sans que cela change le comportement de la classe.

Question 3 : Quel est le rôle de chaque méthode ?

Les rôles des méthodes sont les suivants :

- `deposit` : sert à déposer un montant d'argent égal à `depositAmount` en augmentant le crédit du compte d'autant.
- `withdraw` : sert à retirer un montant d'argent égal à `withdrawAmount` en crédit le crédit du compte d'autant.
- `getNumber` : permet d'accéder au numéro du compte.
- `getBalance` : permet d'accéder au crédit du compte.

Question 4 : Est-ce que cela vous paraît être une utilisation normale de la classe `BankAccount` ?

Dans le code écrit par le programmeur, le compte est débiter d'un montant négatif ce qui revient à augmenter

son crédit et donc est de facto un dépôt. Ce n'est pas une utilisation normale de la classe. Il y a plusieurs façons de régler le problème :

- Rajouter une assertion en changeant la méthode `deposit` en :

```
public void withdraw(double withdrawAmount)
{
    assert withdrawAmount >= 0;
    this.balance -= withdrawAmount;
}
```

- Rajouter un test ne faisant le dépôt que s'il est positif. La nouvelle méthode `deposit` renvoie un booléen afin d'indiquer le succès ou non du dépôt :

```
public boolean withdraw(double withdrawAmount)
{
    if (withdrawAmount < 0)
        return false;
    this.balance -= withdrawAmount;
    return true;
}
```

Question 5 : On souhaiterait rajouter une méthode `String toString()` à la classe afin de convertir le compte en une chaîne de caractère au format suivant : "Compte crédit : 123\$" (exemple d'un compte ayant le numéro 12 et un crédit de 123). Que faut-il changer à la classe `BankAccount` ?

Il suffit d'ajouter la méthode suivante :

```
public String toString(){
    return "Compte crédit : " + getBalance() + "$";
}
```

Question 6 : On souhaiterait empêcher les retraits (`withdraw`) entraînant un crédit négatif. Un retrait entraînant un solde négatif ne serait tout simplement pas exécuté. Comment peut-on changer la classe `BankAccount` pour réaliser cela ? Faut-il changer la signature de la méthode `withdraw` ?

Comme pour la méthode `withdraw`, il y a plusieurs façons de régler le problème :

- Rajouter une assertion en changeant la méthode `deposit` en :

```
public void withdraw(double depositAmount)
{
```

```

assert depositAmount ==> 0;
this.balance += depositAmount;
}

```

- Rajouter un test ne faisant le dépôt que s'il est positif. La nouvelle méthode `deposit` renvoie un booléen afin d'indiquer le succès ou non du dépôt :

```

public boolean deposit(double depositAmount)
{
    if (depositAmount < 0)
        return false;
    this.balance += depositAmount;
    return true;
}

```

Question 7 : On souhaiterait rajouter un attribut à la classe permettant de lier une personne à un compte en banque. Comment peut-on changer la classe `BankAccount` pour réaliser cela ?

Il suffit de rajouter un attribut à la classe. Cet attribut peut être une chaîne de caractère `String` `ownerName` représentant le nom de la personne ou bien directement un objet `Person` `owner` de type `Person` définie par exemple par la classe suivante :

```

public class Person {
    String firstName;
    String lastName;

    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public String toString() {
        return firstName + " " + lastName;
    }
}

```

Flacons (bonus)

Question 8 : Quels qualificateurs, `public`, `final` ou `private`, doit-on donner aux attributs ?

- `capacity` : la capacité d'une bouteille ne pouvant pas changer l'attribut doit être `final`. On peut mettre l'attribut en `private` pour limiter son accès mais c'est moins indispensable quand celui ci est déjà `final`.

- **totalVolume** : le volume de liquide contenu dans une bouteille peut changer (via les méthodes qui versent du liquide dans la bouteille ou la méthode qui boit son contenu), l'attribut *ne* doit donc *pas* être **final**. On ne souhaite pas que l'utilisateur de l'objet puisse changer directement le volume contenu dans la bouteille. par conséquent on met l'attribut **private**.
- **concentration** : la concentration de sirop dans une bouteille peut changer (via les méthodes qui versent du liquide dans la bouteille), l'attribut *ne* doit donc *pas* être **final**. On ne souhaite pas que l'utilisateur de l'objet puisse changer directement la concentration de sirop de la bouteille. par conséquent on met l'attribut **private**.
- **label** : on peut considérer qu'on a pas le droit de changer les étiquettes, auquel cas l'attribut doit être **final**. On peut mettre l'attribut en **private** pour limiter son accès mais c'est moins indispensable quand celui ci est déjà **final**.

Question 9 : Est-il raisonnable d'ajouter des méthodes pour accéder aux attributs privées ?

Il n'est pas raisonnable de mettre en place des méthodes publiques pour modifier (*setters*) les attributs car on les a mis en privé justement pour éviter des modifications. Pour l'accès en lecture, l'implémentation de *getters* est raisonnable mais pas indispensable.

Question 10 : Implémenter la classe **Bottle**.

```
public class Bottle {
    private final double capacity;
    private double totalVolume;
    private double concentration;
    private final String label;

    public Bottle(double capacity,
                  double totalVolume,
                  double concentration,
                  String label) {
        this.capacity = capacity;
        this.totalVolume = totalVolume;
        this.concentration = concentration;
        this.label = label;
    }

    private double getSyrupVolume(){
        return concentration * totalVolume;
    }

    void pourLiquide(double volume, double concentration){
        double newSyrupVolume = getSyrupVolume()
```

```

        + concentration*volume;
    this.totalVolume += volume;
    this.concentration = newSyrupVolume/totalVolume;
}

public void pourSyrup(double volume){
    pourLiquid(volume,1);
}

public void pourWater(double volume){
    pourLiquid(volume, 0);
}

public void transvase(Bottle pouringBottle, double volume){
    volume = pouringBottle.drink(volume);
    this.pourLiquid(volume, pouringBottle.concentration);
}

public double drink(double volume){
    volume = Math.min(volume, this.totalVolume);
    this.totalVolume -= volume;
    return volume;
}

public String toString(){
    return "Étiquette : " + this.label + " Volume : "
        + this.totalVolume + " Concentration "+
        this.concentration;
}

}

```