

1 BankAccount

Dans cette exercice, on va travailler sur la classe `BankAccount` ci-dessous qui permet de gérer un compte en banque avec retrait et dépôt.

```
public class BankAccount {  
  
    private double balance;  
    private int number;  
  
    private static int lastAccountNumber = 0;  
  
    public BankAccount(double initialBalance)  
    {  
        this.balance = initialBalance;  
        this.number = lastAccountNumber++;  
    }  
  
    public void deposit(double depositAmount)  
    {  
        this.balance += depositAmount;  
    }  
  
    public void withdraw(double withdrawAmount)  
    {  
        this.balance -= withdrawAmount;  
    }  
  
    public int getAccountNumber()  
    {  
        return this.number;  
    }  
  
    public double getBalance()  
    {  
        return this.balance;  
    }  
  
}
```

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?
2. Quel est le rôle du mot-clé `this`. Est-ce que son utilisation est indispensable?
3. Quel est le rôle du mot-clé `static` dans la définition de `lastAccountNumber`?
4. Quel est le rôle de chaque méthode?

5. Un programmeur a écrit le code suivant qui utilise la classe `BankAccount` :

```
public class MyBank{
    public static void main(String[] args){

        BankAccount account = new BankAccount(700);
        account.withdraw(-500);

        System.out.print("BankAccount " + account.getNumber());
        System.out.println("has a balance of " + account.getBalance());
    }
}
```

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

6. On souhaiterait rajouter une méthode `String toString()` à la classe afin de convertir le compte en une chaîne de caractère au format suivant : "Compte numéro : 12 crédit : 123\$" (exemple d'un compte ayant le numéro 12 et un crédit de 123). Que faut-t-il changer à la classe `BankAccount` ?
7. 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` ?
8. 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 ?

2 Application de la classe Point

On va réutiliser la classe `Point` qu'on a définie dans un exercice précédent (avec quelques méthodes supplémentaires). Le code de la classe `Point` est le suivant :

```
public class Point {
    public final int x;
    public final int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public double distanceTo(Point p){
        return Math.hypot(this.x - p.x, this.y - p.y);
    }

    public Point translate(int dx, int dy) {
        return new Point(x + dx, y + dy);
    }

    public String toString(){
        return "(" + this.x + "," + this.y + ")";
    }

    public boolean equals(Point p){
```

```

    return this.x == p.x && this.y == p.y;
}
}

```

On veut créer une classe pour représenter une ligne polygonale. Une ligne polygonale est une figure géométrique formée d'une suite de segments de droites reliant une suite de points. Une ligne polygonale est donc définie par un certain nombre de points qu'on appelle ses sommets. On va définir une classe `PolygonalChain` pour représenter de tels objets. Les sommets y seront mémorisés sous forme de tableau de points.

Cette classe `PolygonalChain` comportera donc :

- un attribut :
 - `Point[] vertices` : le tableau des points qui sont les sommets de la ligne polygonale.
- des constructeurs :
 - `PolygonalChain(int size)` : constructeur d'une ligne polygonale à `size` sommets, les sommets n'étant pas définis.
 - `PolygonalChain(Point [] vertices)` : constructeur d'une ligne polygonale ayant les sommets indiqués en argument.
- des méthodes :
 - `Point getVertex(int index)` : renvoie le sommet numéro `index` de la ligne.
 - `void setVertex(int index, Point p)` : fixe la valeur du sommet numéro `index` à `p`.
 - `String toString()` : renvoie une expression de la ligne polygonale sous forme de texte. Pour une ligne polygonale composée des points (1,1), (2,2) et (-3,3), la méthode `toString` devra retourner la chaîne de caractères "`[(1,1), (2,2), (-3,3)]`".
 - `void translate(int dx, int dy)` qui applique à chaque sommet la ligne polygonale une translation de vecteur (dx, dy) .
 - `double length()` qui renvoie la longueur de la ligne polygonale, c'est-à-dire la somme des longueurs des segments qui la composent.
 - `boolean isClosed()` qui renvoie `true` si la chaîne est fermée, c'est-à-dire que son premier sommet est égal à son dernier sommet.

3 Vecteur d'entiers

Écrivez la classe `Vector` qui permet de gérer un tableau dont la capacité augmente automatiquement quand celui-ci est plein. Cette classe contient deux attributs : un tableau d'entiers `array` et un entier `size`. La longueur du tableau `array` peut être supérieure à `size`. Néanmoins, les entiers réellement présents dans le vecteur sont stockés dans les `size` premières cases du tableau `array`. Le constructeur prend en paramètre la longueur initiale du tableau `array`, c'est-à-dire la capacité initiale du vecteur. Si la longueur du tableau `array` ne permet plus de conserver tous les éléments du vecteur, elle est automatiquement augmentée à l'aide de la méthode `ensureCapacity`. La classe fournit les méthodes suivantes :

- `void ensureCapacity(int capacity)` fait en sorte que le tableau `array` puisse contenir `capacity` éléments. Si la capacité actuelle du vecteur est inférieure à `capacity`, la capacité est augmentée. La nouvelle capacité doit être égale à $\max(\text{capacity}, 2 \times \text{capacité actuelle})$. Les nouvelles cases du tableau `array` sont initialisées à zéro. Le nombre d'éléments (c'est-à-dire la valeur de `size`) n'est pas modifié.
- `void resize(int size)` modifie la taille du vecteur. Si la capacité est inférieure à `size`, elle est augmentée et les nouvelles cases sont initialisées à zéro.
- `int size()` retourne la taille actuelle du vecteur.
- `boolean isEmpty()` teste si le vecteur est vide.
- `void add(int value)` ajoute l'entier `value` à la fin du vecteur.
- `void set(int index, int value)` affecte l'élément `value` à la position `index` dans le tableau. Si le tableau contient moins de `index+1` éléments, la méthode ne fait rien.

— `int get(int index)` retourne l'élément à la position `index` dans le vecteur. Si le vecteur contient moins de `index+1` éléments, la méthode retourne 0.