Séquence 3

Écrire une classe

1 Quel est le problème ?

Il existe beaucoup de notions qu'on souhaite utiliser dans des programmes et qui ne correspondent ni aux types de base de Java (int, double, etc), ni aux classes prédéfinies de la librairie (Date, String, etc).

Par exemple, on peut avoir besoin de représenter des personnes. Une personne ne peut se réduire à un nombre ou une chaîne de caractères et il n'existe pas de classe des personnes dans la librairie standard de Java. Dans un tel cas, il faut écrire soi-même la classe.

2 Composants fondamentaux d'une classe

Une classe comporte les éléments suivants :

— Des données stockées dans des variables appelées *attributs*. Elles doivent contenir toute l'information dont le programme a besoin concernant l'entité à représenter.

— Un moyen d'initialiser les données à la création de l'objet. C'est ce qu'on appelle un *constructeur*.

— Des traitements qui utilisent ou modifient les données : ce sont les *méthodes* de l'objet. On distingue les opérations qui utilisent les données en lecture uniquement, sans les modifier, et celles qui changent les données. Les premières sont appelées *accesseurs*, les secondes des *modificateurs*.

3 Exemple des comptes bancaires

Prenons un exemple, celui d'un compte en banque (un peu simplifié).

3.1 Attributs des comptes

On va utiliser trois attributs pour représenter un compte dans nos programmes Java :

— Le nom du titulaire.

— Le numéro de compte.

— Le solde du compte.

Ces trois attributs seront trois variables de type respectif `String`, `int` et `double`. Notons que les `int` de Java sont trop petits pour les numéros de comptes réels utilisés en France qui comportent

beaucoup de chiffres. Nous les utilisons quand même dans cet exemple simplifié. Ces variables sont déclarées directement dans la classe. Voici le début du programme.

```
public class Compte{
  private String titulaire;
  private int numero;
  private double solde;
```

Ces déclarations utilisent le mot-clé `private` : nous n'allons pas expliquer tout de suite à quoi cela sert. Nous allons l'utiliser systématiquement pour tous les attributs jusqu'à nouvel ordre.

3.2 Constructeur

Le rôle du constructeur est d'initialiser les attributs de la classe. Ici, il y a trois attributs : le titulaire et le numéro de compte ne peuvent pas être devinés dans le constructeur en utilisant du code Java. Il faut que cette information soit donnée lors de la création de l'objet. Cela sera fait au moyen de paramètres du constructeur. Le solde pour sa part peut être fixé à 0.0 à la création de l'objet. Cela revient à supposer que les comptes sont vides à leur création. Cela est assez réaliste.

Le constructeur est une sorte de méthode appelée par l'instruction `new` lors de l'instanciation d'une classe. Cette méthode renvoie l'objet créé par new correctement initialisé. Dans la classe, il se distingue des méthodes par le fait qu'il porte le même nom que la classe et que le type des objets renvoyés n'est pas précisé. Dans son code, il n'y a pas de return pour renvoyer l'objet créé. C'est fait automatiquement de façon sous-entendue.

Voici le code du constructeur à ajouter dans la classe :

```
public Compte(String tit, int num){
  titulaire = tit;
  numero = num;
  solde = 0.0;
}
```

On voit que le code Java du constructeur utilise les attributs (à gauche des affectations) et les paramètres du constructeur.

3.3 Méthodes

Il s'agit de déterminer quelles opérations sont nécessaires pour utiliser un compte bancaire. Parmi les opérations usuelles que tout un chacun fait sur son compte, il y a la consultation du solde, le dépôt d'argent et le retrait d'argent. Ces trois opérations doivent nécessairement figurer dans la classe sous forme de trois méthodes (on pourrait à la rigueur ne faire qu'une méthode pour déposer et retirer en considérant qu'un retrait est un dépôt d'un montant négatif).

Faut-il d'autres méthodes ? Pas nécessairement, mais on peut ajouter des méthodes qui permettront de faciliter la mise au point des programmes et/ou d'homogénéiser les pratiques.

On peut proposer des méthodes accesseurs pour le nom du titulaire et le numéro de compte. Cela ne pose pas de problème particulier, même si ça ne correspond pas vraiment à des opérations courantes sur les comptes.

Voici donc un premier lot de méthodes à ajouter à la classe.

```
public double getSolde(){
  return solde;
}
```

```
  public void deposer(double montant){
    if (montant<0){
      throw new IllegalArgumentException("Un␣montant␣ne␣peut␣pas␣être␣négatif");
    }else{
      solde = solde + montant;
    }
  }
  public void retirer(double montant){
    if (montant<0){
      throw new IllegalArgumentException("Un␣montant␣ne␣peut␣pas␣être␣négatif");
    }else{
     solde = solde -montant;
    }
  }
  // optionnelles
  public int getNumero(){
    return numero;
  }
  public String getTitulaire(){
    return titulaire;
  }
```

Deux remarques à propos de ce code : le nom de la méthode `getSolde` : c'est l'usage de nommer `getXXX` une méthode qui renvoie la valeur de l'attribut `XXX`. Eclipse, par exemple, vous propose d'écrire ces méthodes pour vous si vous les voulez.

Seconde remarque : l'exception `IllegalArgumentException` est une exception prédéfinie prévue pour les cas où l'on donne des arguments qui ne conviennent pas à une méthode.

3.4 La méthode toString

Il est d'usage d'écrire une méthode nommée `toString` qui n'a pas de paramètre et renvoie une chaîne de caractère. Cette chaîne doit être un résumé de l'état de l'objet.

```
  public String toString(){
    return "Titulaire:␣" + titulaire + "␣numéro:␣" + numero + "␣solde:␣" +
           solde;
  }
```

Cette méthode est appelée par `System.out.println` et par la concaténation de chaînes `+`. Si on n'écrit pas cette méthode dans la classe, c'est un `toString` par défaut qui est utilisé et qui affiche plus ou moins la référence de l'objet.

Voyons un exemple de création puis d'affichage d'un compte.

```
public class TestCompte{
    public static void main(String[] args){
        Compte c1 = new Compte("Toto",107);
        String str = "le␣compte␣";
        System.out.println(c1);
        str = str + c1;
        System.out.println(str);
    }
}
```

Si l'on écrit une classe sans méthode `toString`, voilà ce qu'affiche le code suivant :

```
> java TestCompte
Compte@6d06d69c
le compte Compte@6d06d69c
```

Avec la méthode `toString` telle qu'écrite ci-dessus, l'affichage est le suivant :

```
> java TestCompte
Titulaire: Toto numéro: 107 solde: 0.0
le compte Titulaire: Toto numéro: 107 solde: 0.0
```

3.5 Les méthodes setXXX

De même qu'on peut écrire une méthode `getXXX` pour accéder à un attribut de la classe, on peut écrire une méthode `setXXX` pour affecter une valeur à cet attribut. Eclipse vous propose de le faire automatiquement pour vous.

Il ne faut pas le faire systématiquement pour tous les attributs, car cela présente un risque de corruption des données et de perturbation de la cohérence des objets instances de la classe.

Dans notre exemple, il **ne faut absolument pas de méthode `setXXX`** :

— pas de méthode `setSolde` parce que le solde ne doit pas évoluer autrement que par des dépôts et des retraits. Il s'agit toujours de faire un + ou un - par rapport à l'état actuel du solde.
— pas de méthode `setTitulaire` car on ne change jamais le titulaire d'un compte en banque. Ce titulaire doit être connu à la création du compte et ne change jamais, c'est un principe de gestion des comptes bancaires.
— pas de méthode `setTitulaire` pour changer le numéro de compte car un compte ne change jamais de numéro.

Ainsi, la présence ou l'absence de méthode `setXXX` doit être bien pesée, contrairement aux méthode `getXXX` qui présentent moins de risque. Il faut quand même réfléchir un peu. Par exemple, s'il y avait un code secret associé à une compte, il faudrait probablement ne pas avoir de méthode `getCodeSecret` pour garantir que le code reste secret.

4 Récapitulation et terminologie

— Les composantes fondamentales d'une classe sont les attributs, le constructeur et les méthodes.
— Les attributs sont des variables Java déclarées et éventuellement initialisées dans la classe. Elles peuvent être utilisées par toutes les méthodes de la classe.
— Le constructeur est un sous-programme appelé par l'instruction `new` juste après la création de l'objet. Il a pour but d'initialiser l'objet, ce qui veut souvent dire affecter des valeurs aux attributs.
— Les méthodes se divisent en accesseurs qui lisent seulement les valeurs des attributs et en modificateurs qui affectent directement ou indirectement des valeurs aux attributs.
— On ne crée par forcément d'accesseur et de modificateurs pour chaque attribut.
— Certains accesseurs et certains modificateurs concernent plusieurs attributs. Par exemple, `toString` est un accesseur qui lit les valeurs des trois attributs de `Compte`.
— Jusqu'à nouvel ordre, tous les attributs seront déclarés avec `private` suivi du type et du nom de l'attribut.

— Il est d'usage d'écrire une méthode `toString` qui renvoie une chaîne de caractère résummant l'état courant de l'objet.

Terminologie :

— Attribut : on parle également de `variable d'instance`
— Constructeur.
— Méthode d'une classe : on parle également de `méthode d'instance` qui sont les méthodes qu'on appelle sur une instance, par opposition aux `méthodes statiques` ou `méthodes de classe` qu'on appelle sur une classe comme par exemple `Integer.parseInt()`.
— Accesseur.
— Getter : méthode `getXXX` qui renvoie la valeur d'un attribut.
— Modificateur.
— Setter : méthode `setXXX` qui affecte une nouvelle valeur à un attribut.

5 Code complet de l'exemple

```
public class Compte{
    private String titulaire;
    private int numero;
    private double solde;
    public Compte(String tit, int num){
        titulaire = tit;
        numero = num;
        solde = 0.0;
    }
    public double getSolde(){
        return solde;
    }
    public void deposer(double montant){
        if (montant<0){
            throw new IllegalArgumentException("Un␣montant␣ne␣peut␣pas␣être␣négatif");
        }else{
            solde = solde + montant;
        }
    }
    public void retirer(double montant){
        if (montant<0){
            throw new IllegalArgumentException("Un␣montant␣ne␣peut␣pas␣être␣négatif");
        }else{
            solde = solde -montant;
        }
    }
    public int getNumero(){
        return numero;
    }
    public String getTitulaire(){
        return titulaire;
    }
    public String toString(){
        return "Titulaire:␣" + titulaire + "␣numéro:␣" + numero + "␣solde:␣" +
            solde;
```

```
    }
}
```