

Programmation Orientée Objets avec Java

Chapitre 2 : Objets et Classes

Stéphane Malandain / Yassin Rekik
Semestre d'automne 2023

1 Exercice : Cohérence

La classe proposée ci-dessous permet de créer un compte bancaire : Une règle de cohérence voudrait qu'un compte soit en tout temps positif ou égal à zéro.

Trouvez 3 exemples d'utilisations qui permettent de corrompre l'état d'un compte bancaire et proposez une solution pour éviter ce genre de problème.

```
1  public class Account {
2
3      public int amount;
4      private String owner;
5
6      public Account(String owner, int amount) {
7          this.amount = amount;
8          this.owner = owner;
9      }
10     public static Account of(String owner, int amount) {
11         if (amount < 0) {
12             throw new RuntimeException("Amount must be positive");
13         }
14         return new Account(owner, amount);
15     }
16     public int amount() {
17         return this.amount;
18     }
19     public void deposit(int value) {
20         this.amount += value;
21     }
22     public void withdraw(int value) {
23         if (value <= 0) {
24             throw new RuntimeException("Amount must be positive");
25         } else if (this.amount - value < 0) {
26             throw new RuntimeException("operation impossible: capital would have become negative");
27         }
28         this.amount -= value;
29     }
30 }
```

2 Exercice : Bank Account

Réalisez des fonctionnalités sur des comptes bancaires. Nommez votre classe `Account`. Voici la liste des fonctionnalités:

- création d'un compte sans argent pour un propriétaire donné (à l'aide d'un constructeur);
- création d'un compte pour un propriétaire donné en spécifiant l'argent qu'il a initialement. Un compte ne peut pas être créé avec un montant négatif (à l'aide d'un constructeur);
- retirer de l'argent d'un compte;
- déposer de l'argent sur un compte;
- virer de l'argent d'un compte à un autre;
- pour toutes les opérations, le solde d'un compte ne doit jamais être inférieur à -2000.-;
- récupérer le nom complet du propriétaire à l'aide d'une chaîne de caractères;
- comparer si deux comptes sont égaux. Ils le sont s'ils ont le même propriétaire et le même montant;

3 Exercice : Transvasement

Vous devez développer une fonctionnalité permettant de faire le transvasement d'un récipient d'origine vers d'autres récipients de destination.

3.1 version "fonction" - classe *ContainerHelper*

Cette version doit être réalisée à l'aide d'une méthode et un nombre arbitraire d'arguments

- Le premier argument est le récipient d'origine
- Le deuxième argument est le premier récipient de destination (ces deux premiers arguments sont obligatoires)
- Les arguments suivants sont les récipients secondaires

Voici un exemple d'utilisation :

- `transfer(10, 3, 3, 3) -> {3, 3, 3}` (les trois récipients de destination sont remplis à ras bord)
- `transfer(10, 3, 3, 3, 3) -> {3, 3, 3, 1}`
- `transfer(5, 3) -> {3}`
- `transfer(0, 3) -> {0}`
- `transfer(100, 10, 40, 30, 50) -> {10, 40, 30, 20}`

Utilisez un nombre arbitraire d'arguments tout en obligeant l'utilisateur à renseigner au moins deux arguments.

3.2 version objet - classe *Container*

Réalisez un système de classes permettant une utilisation suivante :

```

1 public class MainContainer{
2
3     Run | Debug
4     public static void main(String[] args) {
5
6         Container origin = Container.withCapacity(capacity: 10); // récipient vide par défaut
7         origin.fillToTheMax(); // remplissage
8         Container destination1 = new Container(capacity: 5); // idem que Container.withCapacity(5);
9         destination1.fillWith(value: 2);
10        Container destination2 = Container.withCapacity(capacity: 3);
11        Container destination3 = Container.withCapacity(capacity: 10);
12        origin.fillTo(destination1, destination2, destination3);
13        assert destination1.isFull() ;
14        assert destination2.isFull() ;
15        assert destination2.remaining() == 0 ;
16        destination2.remove(2);
17        assert destination2.remaining() == 2 : " remain more or less than 2";
18        assert !destination3.isFull() ;
19        assert destination3.quantity() == 4;
20        destination3.flush();
21        assert destination3.quantity() == 0;
22    }
23

```

Remarques:

- Utilisez un nombre minimal de champs
- Exposez que les fonctionnalités utiles à l'utilisateur, les autres doivent rester **privées**
- La méthode `fillTo` doit accepter au minimum un récipient
- Gérez les états incohérents du mieux possible

4 Exercice : Tableaux dynamiques

4.1 Tableau statique pseudo dynamique

Réalisez plusieurs fonctionnalités sur des tableaux statiques d'entiers:

- Une fonctionnalité qui prend un tableau d'entier et ajoute un nouvel entier au début du tableau (append),
- Une seconde fonctionnalité qui retourne l'élément de tête (head),
- Une troisième qui retourne le tableau sans l'élément de tête (tail),
- Une dernière qui concatène deux tableaux (concat). Utilisez des fonctionnalités écrites plus haut.

Ecrivez-les dans un fichier nommé `DynArray.java`. Exemple d'utilisation :

```

public static void main(String[] args) {
    int[] is = {1, 2, 2, -1, 5};
    int[] is2 = append(is, value: 10);
    // is2 = {10, 1, 2, 2, -1, 5};
    int j = head(is2);
    // j = 10
    int[] tail = pop(is);
    // tail = {1, 2, 2, -1, 5};
    int[] result = concat(is, new int[]{4,3,2});
    // result = {1, 2, 2, -1, 5, 4, 3, 2}
}

```

4.2 DynArray version objet

Reprenez l'exercice précédent (3.1, DynArray) et réalisez une version objet de celle-ci.

DynArray est maintenant une classe instanciable. Utilisez un tableau statique comme champ privé (représentation interne). Vous pouvez donc reprendre vos fonctionnalités déjà réalisées. N'exposez que les méthodes visibles ci-dessous. Remarquez que la classe est immutable. Par exemple, la méthode `append` ne modifie pas l'objet `is` mais retourne une nouvelle version de l'objet.

Ajoutez également les nouvelles fonctionnalités (`of`, `reversed`, `size`, `equals`, `toString`).

```

public static void main(String[] args) {
    DynArray is = DynArray.of(...values: 1, 2, 2, -1, 5);
    // is = [1, 2, 2, -1, 5];
    DynArray is2 = is.append(value: 10);
    // is2 = [10, 1, 2, 2, -1, 5];
    int j = is2.head();
    // j = 10
    DynArray tail = is2.pop();
    // tail = [1, 2, 2, -1, 5];
    DynArray rev = tail.reversed();
    // tail = [5, -1, 2, 2, 1]
    int size = rev.size();
    // size = 5
    boolean areEquals = is.equals(tail);
    // areEquals = true
    DynArray result = is2.concat( DynArray.of(...values: 4,3,2) );
    // result = [10, 1, 2, 2, -1, 5, 4, 3, 2]
    String representation = result.toString();
    // representation = [10, 1, 2, 2, -1, 5, 4, 3, 2]
}

```