TP n° 4
Licence Informatique (L2)
« Programmation objet avancée »

F. BERTRAND

Année universitaire 2020–2021

1 Hiérarchie d'animaux

L'objectif de cet exercice est de montrer l'utilisation d'une hiérarchie d'interfaces représentant un sorte d'héritage multiple.

À partir de l'archive `zoo1.zip`, réaliser les étapes suivantes **en conservant, pour chaque étape, les classes et les interfaces développées** :

1. en utilisant des classes (pas d'interfaces), créer une hiérarchie permettant de stocker dans un tableau une instance de chacune des classes présentes dans l'archive. Le programme devra afficher :
   — le nombre de reptiles ;
   — le nombre d'animaux terrestres.
2. Ajouter sous forme d'interface les classes que vous n'avez pas pu définir sous forme de classes abstraites et modifiez vos classes en conséquence.
3. Les deux interfaces créées dans la question précédente n'apportent aucune contrainte aux classes qui les implémentent. D'autre part, nous avons privilégié certains types qui demeurent des classes (ex. `Mammifere`) et d'autres qui ont été créés sous forme d'interfaces (ex. `Aquatique`).

   Dans un souci d'unification proposer une solution où les types `Animal`, `Mammifere`, `Reptile`, `Aquatique` et `Terrestre` seront des interfaces et déclareront une méthode de test sur la propriété qu'il possède (ex. `Animal` imposera qu'on définisse une méthode `String donneNom()`, `Mammifere` une méthode `boolean estCarnivore()`.

2 Utilisation d'une interface pour le tri d'éléments

On souhaite pouvoir trier une liste de comptes bancaires (cf. les deux classes fournies `CompteBancaireInitial` et `Client` sur Moodle) en utilisant la méthode `sort()` de la classe `java.util.Collections`. Cette méthode, pour réaliser son tri, s'appuie sur la comparaison d'élements deux par deux définie par une méthode `compareTo()`. Pour être certain que les objets à comparer appartiennent à une classe ayant implémenté cette méthode `compareTo()`, le paramètre de `sort()` est du type `Comparable`, interface déclarant une méthode `compareTo()`. Ainsi la classe des objets à trier doit implémenter cette interface et donc définir une méthode `compareTo()`.

Travail à réaliser :

1. Modifier la classe `CompteBancaire` pour qu'elle implémente l'interface `Comparable`. La méthode `compareTo()` qui sera définie utilisera le numéro de compte comme critère de comparaison. À l'aide d'un programme de test, créer plusieurs comptes et

vérifier que la méthode `sort()` trie correctement une liste de comptes passée en paramètre.

2. Puis proposer une solution plus générique permettant l'utilisation de différents critères de tri (ex. nom du détenteur, montant du solde, ...) en définissant trois classes implémentant l'interface `Comparator<CompteBancaire>` :
   — `ComparateurNumeroCompte`;
   — `ComparateurNomDetenteur`;
   — `ComparateurMontantSolde`.
   (la méthode `equals` déclarée dans l'interface n'aura pas à être implémentée car nous bénéficions d'une version par défaut héritée de la classe `Object`).

3 Clonage d'objets

Soient les deux classes `EnteteCommande` et `Commande` déclarées dans `Commande.java` (cf. code fourni) dont le code est présenté ci-dessous :

```
1  package clonage_pb;
2
3  public class EnteteCommande {
4      // nom du client qui a passe la commande
5      private String nomClient;
6      // numero de la commande
7      private int numeroCommande;
8
9      public EnteteCommande(String nomClient, int numero) {
10         this.nomClient = nomClient;
11         this.numeroCommande = numero;
12     }
13
14     public String donneNomClient() {
15         return this.nomClient;
16     }
17
18     public void changeNomClient(String nomClient) {
19         this.nomClient = nomClient;
20     }
21
22     public int donneNumeroCommande() {
23         return this.numeroCommande;
24     }
25
26     public void changeNumeroCommande(int numeroCommande) {
27         this.numeroCommande = numeroCommande;
28     }
29
30     public String toString() {
31         return ("numero = " + this.numeroCommande
32                 + " effectu'ee par : " + this.nomClient);
33     }
34 }
```

```
1  package clonage_pb;
2
3  /**
4   * Classe representant une commande
5   */
6  public class Commande {
7      // compteur du nombre de commandes creees
8      private static int nombreCommandes = 0;
9      private EnteteCommande entete;
10
11     // le constructeur est inaccessible pour obliger l'utilisateur
12     // a utiliser la methode creerCommande pour creer des commandes
13     private Commande(String nomClient, int numeroCommande) {
14         this.entete = new EnteteCommande(nomClient, numeroCommande);
15     }
16
17     // permet la creation de commande
18     public static Commande creerCommande(String nomClient) {
19         // utilisation de la variable de classe nombreCommandes
```

```
        // pour attribuer un nouveau numéro à chaque commande
        return new Commande(nomClient, ++nombreCommandes);
    }

    // permet d'obtenir le nom du client d'une commande
    public String donneNomClient() {
        return this.entete.donneNomClient();
    }

    // permet de modifier le client d'une commande
    public void changeNomClient(String nomClient) {
        this.entete.changeNomClient(nomClient);
    }

    // permet d'obtenir le numero d'une commande
    public int donneNumeroCommande() {
        return this.entete.donneNumeroCommande();
    }

    // permet la copie de commandes
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }

    public String toString() {
        return ("Commande " + this.entete);
    }
}
```

Et la classe de test `TestClonageCommande` :

```
package clonage_pb;

public class TestClonageCommandeV1 {
    public static void main(String[] args) throws Exception {
        // création de deux commandes
        Commande cmd1 = Commande.creerCommande("Martin");
        Commande cmd2 = Commande.creerCommande("Durand");

        System.out.println(cmd1);
        System.out.println(cmd2);
        // creation de la commande cmd3 par clonage de la commande cmd2
        Commande cmd3 = (Commande) cmd2.clone();
        // changement du client initial de cette commande
        cmd3.changeNomClient("Dupont");

        System.out.println("Apres la copie de la commande");
        System.out.println(cmd2);
        System.out.println(cmd3);
    }
}
```

Travail à réaliser :

1. Exécutez la méthode `main` de la classe `TestClonageCommandeV1` et constatez le problème. Indiquer pour quelle raison l'exception `CloneNotSupportedException` a été lancée par la machine virtuelle. Modifier le programme pour que celle-ci n'apparaisse plus.
2. Maintenant observer l'état de la commande n° 2 avant et après la modification de la commande n° 3 obtenue par clonage de la n° 2 en identifiant le problème. La figure 1 est là pour vous aider à comprendre le problème :

   Pour rappel, la méthode `clone`, par défaut, se contente de recopier les valeurs des attributs de l'objet qu'elle duplique et si ces attributs sont des références alors elle recopie les adresses des objets référencés mais ne duplique pas ceux-ci...

   Résoudre ce problème en modifiant la méthode `clone()` de la classe `Commande` et en ajoutant une méthode `clone` dans la classe `EnteteCommande`.

   Tester votre solution en exécutant le `main` de la classe `TestClonageCommandeV2`.
3. À ce stade, vous devez avoir résolu le problème précédent. Néanmoins votre solution doit avoir pris comme hypothèse que la méthode `changeNomClient` de la classe

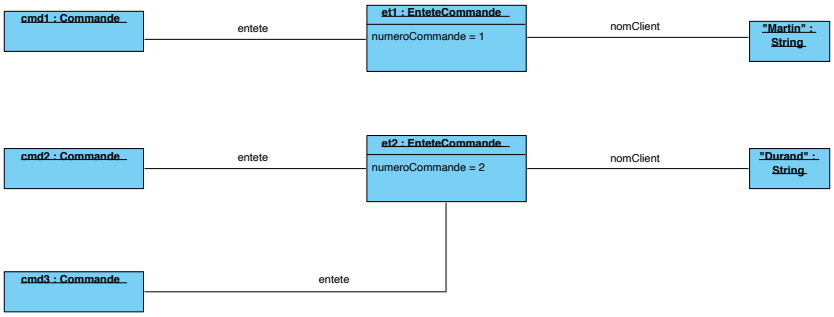

FIGURE 1 – Les différentes instances de `Commande` après clonage

`EnteteCommande` change l'attribut `nomClient` en lui affectant une nouvelle instance de `String` et elle ne modifie donc pas l'objet référencé par `nomClient`. Elle ne le pourrait d'ailleurs pas car la classe `String` ne crée que des instances **immutables** c'est-à-dire qu'une fois créées, ces instances ne peuvent plus être modifiées.

Pour vérifier cette hypothèse, donner à `nomClient` le type `StringBuffer` (classe permettant la modification de ses instances) et modifier le constructeur et la méthode `changeNomClient` en conséquence :

```
1  public EnteteCommande(String nomClient, int numero)
2  {
3    this.nomClient = new StringBuffer(nomClient);
4    this.numeroCommande = numero;
5  }
6
7  void changeNomClient(String nomClient)
8  {
9    this.nomClient.replace(0,this.nomClient.length(),nomClient);
10 }
```

Exécuter la méthode `main` de la classe `TestClonageCommandeV3` et vérifier que votre solution fonctionne toujours. Si ce n'est pas le cas, modifier votre code (principalement la méthode `clone` de la classe `EnteteCommande`) en conséquence.

4. Maintenant il ne doit subsister qu'un seul problème : comment obtenir un nouveau numéro de commande lorsqu'on clone une commande existante ? Modifier votre code pour résoudre ce problème.