TP Collections (1) - equals, hascode, sets

Marianne Simonot

.

1 Savoir définir l'égalité dans une classe (`equals`)

Considérons les 2 classes suivantes qui définissent respectivement des personnes et des adresses. (A télécharger sur Moodle)

```
public class Adresse {
    private int numeroRue;
    private String rue, ville , pays;

    public Adresse(int numeroRue, String rue,
                   String ville , String pays) {
        this.numeroRue = numeroRue;
        this.rue = rue;
        this. ville = ville ;
        this.pays = pays;
    }
   @Override
    public String toString() {
        return numeroRue + "␣" + rue + "␣,␣" + ville
                    + "␣,␣" + pays + "\n";
    }

}
```

```
public class Personne {
    private static int compteur = 0;
    private final int identifiant ;
    private String nom, prenom;
    private Adresse adresse;

    public Personne(String nom, String prenom,
                    Adresse adresse) {
        this.nom = nom;
        this.prenom = prenom;
        this.adresse = adresse;
        identifiant = compteur;
        compteur++;
    }
  @Override
    public String toString() {
        return identifiant + nom + prenom + adresse;
    }
}
```

Aide-mémoire

`equals`

On a très souvent besoin lorsqu'on programme de savoir si deux objets sont égaux. Par exemple, lorsqu'on veut savoir si une personne `p` donnée apparaît dans une liste de personne, il faut parcourir la liste jusqu'à trouver un élément de la liste qui **est égal à p**. Il faut donc avoir une méthode qui permet de tester l'égalité entre 2 objets de la classe. En java, c'est la méthode `equals` qui joue ce rôle. Elle est définie dans `Object` et est donc applicable sur n'importe quel objet. Dans Object, `a.equals(b)` `ssi a et b sont la même adresse`.

exercice 1 ⊛

Télécharger depuis Moodle la classe `ExerciceEgaliteEtSet.java` , récupérez dedans le code ci dessous, exécutez le puis répondez aux questions. :

```
Adresse a1, a2, a3, a4, a5;
a1 = new Adresse(34, "bd Magenta", "Paris", "France");
a2 = new Adresse(34, "bd Magenta", "Cahors", "France");
a3 = new Adresse(12, "rue boule", "Marseille", "France");
a4 = new Adresse(34, "bd Magenta", "Paris", "USA");
a5 = new Adresse(34, "bd Magenta", "Paris", "France");

System.out.println("a1 est il egal a a2 ? :" + a1.equals(a2));
System.out.println("a1 est il egal a a3 ? :" + a1.equals(a3));
System.out.println("a1 est il egal a a4 ? :" + a1.equals(a4));
System.out.println("a1 est il egal a a5 ? :" + a1.equals(a5));
System.out.println("a1 est il egal a a1 ? :" + a1.equals(a1));
```

1. Quelles sont les variables parmi `a1` ... `a5` qui sont considérées comme égales par Java (pour lesquelles `equals` répond true) ?
2. La méthode `equals` de `Object` vous semble-t-elle adaptée pour les objets de type `Adresse` ? (Répondre oui ou non)
3. Quelles sont les variables parmi `a1` ... `a5` que vous aimeriez considérer comme égales (pour lesquelles vous aimeriez que `equals` réponde true) ?
4. Trouvez une solution pour que ce soit le cas et implémenter là.
5. Utilisez `source/generate hashcode and equals` de Eclipse pour définir `equals` et comparez votre code au code engendré.

Aide-mémoire

Engendrer equals et hashcode avec Eclipse

Dans la suite de ce cours, il est absolument obligatoire de toujours utiliser Eclipse pour engendrer equals.
Lorsqu'on utilise cette fonctionnalité d'Eclipse, le `equals` engendré répondra true ssi tous les attributs **que vous avez cliqué dans la fenêtre** ont la même valeur. De plus, vous constaterez qu'Eclipse a géré (sans doute) beaucoup plus de cas que vous : tout ceux où l'un des attributs vaut `null`. Par ailleurs, nous y reviendrons, Eclipse engendre aussi la fonction `hashcode` qui, elle aussi, est fondamentale.

exercice 2 ⊛

La classe `Personne` caractérise les personnes comme ayant un identifiant unique, un nom, un prénom et une adresse.

1. Selon vous, quelles spécifications de `equals` sont réalistes ?

(a) `a.equals(b)` si et seulement si tous les attributs ont la même valeur.

(b) `a.equals(b)` si et seulement si a et b ont le même identifiant.

(c) `a.equals(b)` si et seulement si a et b ont le même nom.

(d) `a.equals(b)` si et seulement si a et b ont le même nom, le même prénom et la même adresse.

2. Utilisez Eclipse pour engendrer `equals()` de façon à ce que `a.equals(b)` si et seulement si a et b ont le même identifiant.
3. vérifier (dans un main) qu'avec votre définition, aucune des variables déclarées comme suit ne sont `equals()` (disponible dans la classe `ExerciceEgaliteEtSET`).

```
Personne p1,p2,p3,p4;
p1 = new Personne("Dupont", "antoine", a1);
p2 = new Personne("Durand", "marie", a2);
p3 = new Personne("Dupond", "leo", a1);
p4 = new Personne("Dupont", "antoine", a1);
```

exercice 3 ⊛

`public boolean contains(Object o)` est une méthode prédéfinie sur les `ArrayList` et permet de vérifier que `o` est un élément de la liste.

1. En gardant la définition de `equals` dans `Adresse`, exécutez dans un main les instructions suivantes :

```
ArrayList<Adresse> la = new ArrayList<>();
        la.add(a1);
        la.add(a2);
        la.add(a3);
        la.add(a4);
        System.out.println("contains" + la.contains(a5));
```

2. Réexecutez ce code après avoir mis en commentaire la définition de `equals` . Expliquez pourquoi `true` s'affiche dans le premier cas et `false` dans le second.

2 La structure de données Set

Aide-mémoire

la structure de données Set

Vous connaissez déjà 2 structures de données permettant de regrouper des éléments de même type : les tableaux et les arrayListes. On appelle ces structures des collections. `Set` est une autre structure de collection.

Définition

Un ensemble (Set) est une collection qui mémorise chaque élément au plus une fois.

Il n'y a pas d'ordre entre les éléments.

Quand utiliser un Set ?

On utilise donc un set lorsque les 2 conditions suivantes sont vérifiées :

1. Nous n'avons pas envie d'avoir de répétitions dans une collection. C'est par exemple le cas lorsqu'on veut gérer des mots clés, ou encore lorsqu'une agence de location de véhicule veut gérer ses voitures. En revanche, pour gérer un historique de commande dans un éditeur de texte, un set n'a aucun intérêt : on peut évidement faire plusieurs opérations "save" par exemple.
   Techniquement, dire qu'il n'y a pas de répétition dans un Set signifie qu'**il n'y a jamais 2 élements `a1` et `a2` dans le set tels que `a1.equals(a2)`**. C'est le set qui gère lui même l'absence de répétition dans le code des opérations d'ajout, et non l'utilisateur.
2. Et l'ordre dans lequel sont rangés les éléments dans la collection n'a pas d'importance. Cela peut convenir pour les voitures de l'agence ou les mots clés à condition que nous n'ayons pas envie d'ordonner les éléments pour optimiser les recherches. Cela ne convient pas pour l'historique car nous avons besoin de savoir qu'une commande a été faite avant une autre.

Quelques méthodes de Set

Comme sur les `ArrayList` nous avons entre autre `add`, `remove` `contains` et la boucle `foreach`
Contrairement aux `ArrayList` nous n'avons aucune des méthodes permettant d'accéder aux éléments par leur indice dans la liste : `get(i)` `add(i,e)`, `remove(i)` ne sont pas définies sur les set. Il est donc impossible de parcourir un set au moyen d'une boucle `for (int i=0; i<l.size;i++)....`

Set est une interface

On ne peut donc créer d'objet instance de Set. Il faut alors utiliser l'une de classes Java implémentant cette interface par exemple `HashSet`.

exercice 1 ⊛

On reprend les classes `Personne` et `Adresse` avec leur définitions de `equals` ainsi que les déclarations de variables.

Recopiez dans votre main les lignes suivantes (disponibles dans `ExerciceEgaliteEtSET`) et exécutez. Combien y a t il d'éléments dans `carnet` et `carnet2` ? Expliquez pourquoi.

```
Adresse a1, a2, a3, a4, a5;
a1 = new Adresse(34, "bd_Magenta", "Paris", "France");
a2 = new Adresse(34, "bd_Magenta", "Cahors", "France");
a3 = new Adresse(12, "rue_boule", "Marseille", "France");
a4 = new Adresse(34, "bd_Magenta", "Paris", "USA");
a5 = new Adresse(34, "bd_Magenta", "Paris", "France");

Personne p1 = new Personne("Dupont", "antoine", a1);
Personne p2 = new Personne("Durand", "marie", a2);
Personne p3 = new Personne("Dupond", "leo", a1);
Personne p4 = new Personne("Dupont", "antoine", a1);
```

```
Set<Adresse> carnet = new HashSet<>();
System.out.println(carnet.add(a1));
System.out.println(carnet.add(a2));
System.out.println(carnet.add(a3));
System.out.println(carnet.add(a4));
System.out.println(carnet.add(a5));

System.out.println(carnet);

Set<Personne> carnet2 = new HashSet<>();
System.out.println(carnet2.add(p1));
System.out.println(carnet2.add(p2));
System.out.println(carnet2.add(p3));
System.out.println(carnet2.add(p4));

System.out.println(carnet2);
```

exercice 2 ⊛

Mettre en commentaire la définition de `hashcode` dans `Adresse` et réexécuter. Combien y a t il d'éléments dans `carnet` ?
5 et non plus 4. `a5` bien que `equals` à `a1`- qui est déjà dans le carnet- est ajouté...
En vous aidant de l'aide mémoire qui suit, expliquez le phénomène.

Aide-mémoire

`hashcode` et son rapport avec `equals`

La spécification Java de hashcode (cf javadoc de Object) dit explicitement qu'il faut toujours que les définitions de ces 2 méthodes respectent le contrat suivant :

si 2 objets sont equals alors ils doivent avoir le même hashcode.

Certaines classes Java (notament toutes les classes dont le nom contient `hash`), se fondent sur ce contrat pour optimiser les tests d'égalité. Pour tester l'égalité de 2 objets, ils comparent d'abord leur hashcode. Si les hashcodes sont égaux, ils appellent equals. Mais si les hascodes sont différents, conformément au contrat (c'est la contraposée, rappelez vous !), ils déduisent que les objets ne sont pas equals.
Il faut donc lorsqu'on redéfinit `equals` redéfinir aussi `hashcode` en le codant de façon à respecter le contrat. La façon la plus facile de faire est d'utiliser Eclipse.

exercice 3 ⊛

Modifiez la définition de `equals` et de `hashcode` (avec Eclipse) pour qu'à l'issue de l'exécution des lignes de code de l'exercice 1, `carnet` ne contienne que 2 éléments : `a1` et `a4`.