Java : exercice 6.5

Utilisation de la classe HashMap

Préparation

Créer une classe `Point2`

- possédant deux attributs privés `int x` et `y`
- un constructeur (avec deux arguments) permettant d'initialiser ces deux attributs
- une méthode `affiche()` qui affiche à l'écran ces deux attributs ainsi que le hashCode de l'objet
- des méthodes get pour chacun des deux attributs

Créer une classe `TestHashMap` avec une méthode main qui

- crée trois instances `p1`, `p2` et `p3` de `Point2`

```
Point2 p1 = new Point2(1,3);
Point2 p2 = new Point2(2,2);
Point2 p3 = new Point2(4,5);
```

- déclarer une collection `hashMapPoints` de type `<Point2, Integer>` et y range ces instances de `Point2`
  - clé : l'instance de `Point2`
  - valeur : un entier quelconque

```
hashMapPoints.put(p1, 10);
```

À la fin de la classe `TestHashMap`, déclarer une fonction `affiche()` permettant d'afficher (grâce à un itérateur) le contenu d'une collection de type `HashMap <Point2, Integer>` :

```
Iterator<Point2> iter = ens.keySet().iterator();
while(iter.hasNext()){
     Point2 p = iter.next();
     System.out.println( "key: " + p.hashCode() + " value: " +
ens.get(p) );
}
```

Lancer cette fonction `affiche()` à la fin de la méthode main.

Exécuter `TestHashMap`.

Ajout d'instances de `Point2` ; quelques méthodes supplémentaires

Ajouter dans la collection `hashMapPoints`, le couple clé-valeur `(p2, 20)`. Exécuter le `TestHashMap`. Qu'en conclure ?

Afficher le résultat de l'instruction `hashMapPoints.get(p1)` qui permet de retourner la valeur associée à la clé `p1`. Afficher le résultat de l'instruction `hashMapPoints.containsKey(p1)` qui permet de savoir si `p1` appartient bien à la collection. Afficher le résultat de l'instruction `hashMapPoints.containValue(10)` qui permet de savoir si « 10 » appartient bien à la collection.

On définit une nouvelle instance de `Point2`

```
Point2 p4 = new Point2(1,3);
```

Afficher le hashCode de `p4`.

Tester si `p4` appartient à la collection. La réponse est non, et c'est normal, car par défaut la méthode `equals()` sur laquelle se base ce test vérifie les hashcodes, qui sont ici différents.

Redéfinition de la méthode `equals()`

On peut avoir envie de redéfinir la méthode `equals()` de la classe `Point2` de la façon suivante :

```
public boolean equals(Object pp) {
     Point2 p = (Point2)pp;
     return ((this.x == p.x) && (this.y == p.y));
}
```

Deux instances de `point2` sont égales si elles ont les mêmes coordonnées.

Lancer TestHashMap après avoir redéfini la fonction `equals()` de la classe `Point2` : `p4` n'appartient toujours pas à la collection alors même que ses coordonnées sont les mêmes que celles de `p1` ! La raison est ici qu'en redéfinissant `equals()`, nous brisons le contrat qui lie `hashcode()` et `equals()` : deux instances sont égales, mais est leurs hashCodes sont différents.

Il faut donc redéfinir également la méthode `hashcode()` de `Point2`, par exemple de la façon suivante

```
public int hashCode() {
     return x+y;
}
```

La redéfinition d'un « bon » hashCode pour éviter les collisions – une collision se produit quand deux objets différents au sens de la méthode `équals()` ont des hashCodes identiques (ce qui n'est pas un problème en soi, la recherche dans la HashMap devant juste se faire en deux étapes, mais peut dégrader les performances de l'application) – n'est pas simple et sort du cadre de ce cours.