Ing39 – Égalité d'objets et Collections

Pierre Courtieu Serge Rosmorduc Virginia Aponte

Dans ce cours

- le problème à résoudre
- rôle de `equals()` et `hashCode()` dans les collections
- quand et comment les rédéfinir ?

Une classe `Heure` pour illustrer le problème


```
public class Heure {
    private final int heure;
    private final int minutes;
    public Heure(int h, int m) {
         this.heure = h; this.minutes = m;
    }
    // getters ...
    @Override
     public String toString() {
          return "[heure=" + heure
               + ", minutes=" + minutes + "]";
     }
}
```

...et 2 objets Collection<Heure>

```
List<Heure> liste = new ArrayList<Heure>();  // une liste
Set<Heure> set = new HashSet<Heure>();       // un ensemble

liste.add(new Heure(13,43)); // ajoutons 13h43

boolean b = liste.contains(new Heure(13,43)); // contient 13h43?
System.out.println("liste contient 13H43 ?  "+b);

// ajoutons 2 fois 13h43 dans set
set.add(new Heure(13,43));
set.add(new Heure(13,43));     // doublon

System.out.println("set="+set.toString()); // afficher set

// Affichages
liste contient 13H43? false
set=[[heure=13, minutes=43],[heure=13, minutes=43]]
```

- ⋆ liste ⇒ 13h43 a été ajouté, mais contains ne le trouve pas
- ⋆ set ⇒ 13h43 apparaît deux fois

Les collections en interne

- ⋆ possèdent en interne des structures variées de stockage :
  - ⋆ tableaux, tables de hachage, arbres de recherche
- ⋆ accès/comparaison éléments via 2 méthodes héritées d'`Object` :
  - ⋆ `boolean equals(Object o)` : compare `this` et o par « égalité »
  - ⋆ ces méthodes se trouvent par défaut dans tout objet
  - ⋆ `int hashCode()` : entier permettant de calculer l'indice où insérer/chercher `this` dans table de hash interne.
- ⋆ Exemple : `liste.contains(new Heure(13,43))`
  - ⇒ parcours `liste` à la recherche de `new Heure(13,43)`
  - ⇒ pour chaque e de `liste`
  - ⇒ invoque `e.equals(new Heure(13,43))`

`equals` et `hashCode` héritées d'`Object`

Implantations de **`equals`** et **`hashCode`** dans **`Object`**

-  basées sur adresses et NON SUR CONTENUS des objets
  - **`e1.equals(e2)`** compare adresses de e1 et e2
  - **`e.hashCode()`** retourne entier calculé à partir de l'adresse de e
  - par défaut dans tout objet !
- Exemple
  - **`liste.contains(new Heure(13,43))`**

  ⇒ **`e.equals((new Heure(13,43)))`** ?

  ⇒ `e` et `(new Heure(13,43))` sont des instances différentes

  ⇒ leurs adresses sont différentes

  ⇒ différents selon `equals` !

Résumé du problème

- ⋆ 2 instances **e1**, **e2** de même contenu ⇒ ont des adresses différentes
- ⋆ si on ne re-définit pas **equals** et **hashCode** :
  - ⇒ différentes selon **equals**
  - ⇒ indices différents où les chercher/insérer selon **hashCode**
  - ⇒ fonctionnement de collections incohérent avec notion d'égalité d'objets dictée par « égalité des contenus »

```
Heure h1 = new Heure(13,43);
Heure h2 = new Heure(13,43);

System.out.println("h1 equals h2? "+h1.equals(h2));
System.out.println("hashCode h1="+h1.hashCode());
System.out.println("hashCode h2="+h2.hashCode());

// affichages
h1 equals h2? false
hashCode h1=1586600255
hashCode h2=932583850
```

Problème avec tables de hachage en stockage interne

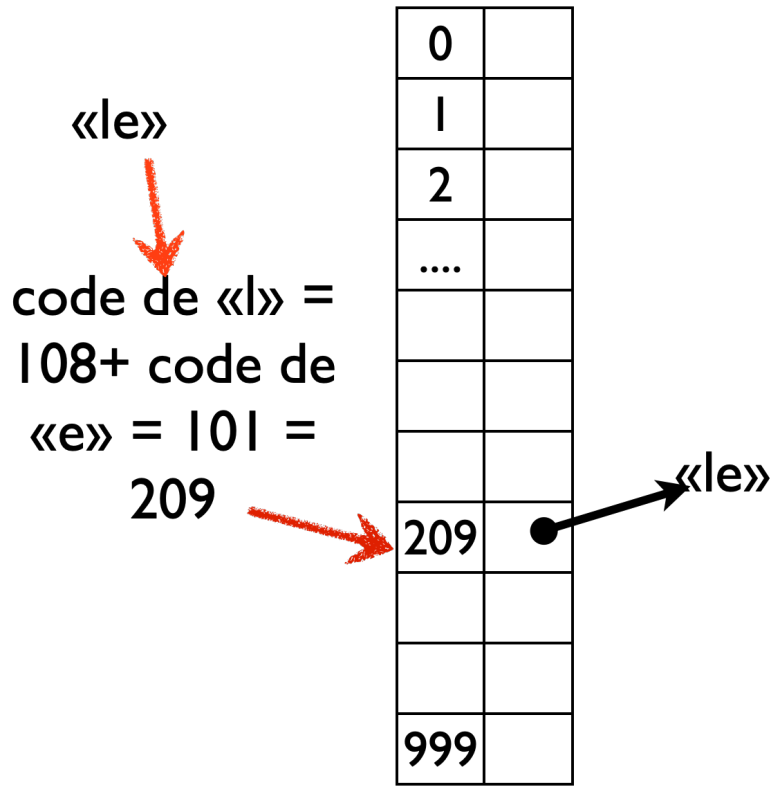
« Les tables à adressage dispersé » (ou de *hachage*)

- ⋆ tableau de taille N + *fonction de hachage* utilisée pour calculer l'indice où placer/chercher un élément.
- ⋆ Exemple de fonction de hachage sur String : additionne les codes Unicode de ses caractères. *hachage*(`"le"`) $= 209$
- ⋆ si e est un élément à stocker/chercher dans `tabHash` on fait :
  - ⋆ appliquer fonction de hachage sur e $\Rightarrow n =$ *hachage*(e)
  - ⋆ l'indice où placer/chercher e dans `tabHash` est $i = n$ *modulo* N
- ⋆ problème : deux chaînes avec mêmes caractères, "el" et "le, ont le même hachage et donc mêmes indices.
- ⋆ solution : chaque case i pointe vers une liste de "collisions" contenant les éléments à cet indice.
- ⋆ il faudra chercher/placer e dans cette liste.

Rappel : structure de stockage interne pour `HashSet`, `HashMap`, ….

Hachage simplifié

On veut implanter un ensemble de Strings dans un tableau de *1000* cases

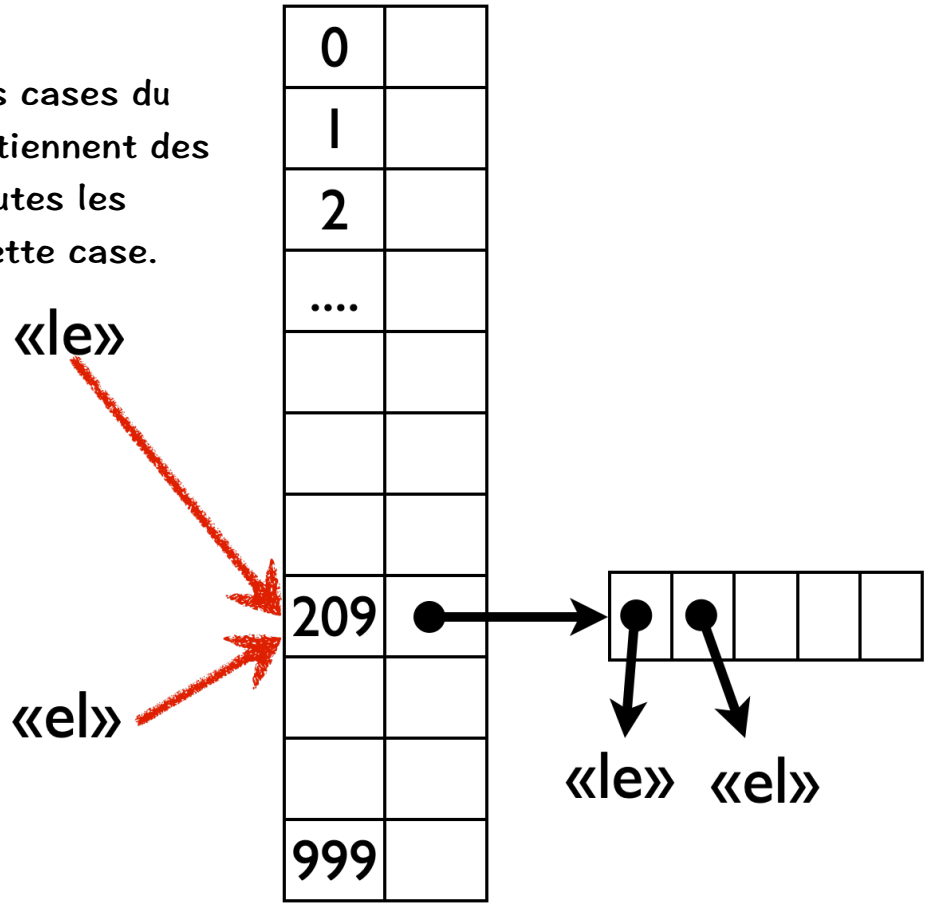


Ex de fonction de hachage: somme des codes Unicode des ses caractères

Hachage avec collisions

Listes de Collisions

Solution: les cases du tableau contiennent des listes de toutes les valeurs à cette case.



Exemple pour `hashcode()`

Rappel : dans un Set on s'attend à ne pas avoir des doublons. Du coup, l'opération `monset.add(o)`, commence par chercher o dans monset, et l'ajoute seulement s'il n'y est pas déjà.

```
HashSet<Heure> ps= new HashSet<Heure>();
ps.add(p1); // ajout 2 objets de même contenu
ps.add(p2);
```

Comportement de l'appel `ps.add(p1)` (premier ajout) $\Rightarrow$

- $\star$ calcul de p_1.`haschode()` $\Rightarrow$ un indice n_1 de la table de Hash interne ;
- $\star$ par défaut, ce calcul se fait à partir de l'adresse de p_1 ;
- $\star$ dans la liste de collision à cet indice, on cherche p_1 (avec `equals`)
- $\star$ si non trouvé, $p1$ est ajouté dans la liste de collision à cet indice.

Comportement du 2ème ajout


```
HashSet<Heure> ps= new HashSet<Heure>();
ps.add(p1); // ajout 2 objets de même contenu
ps.add(p2);
```

`ps.add(p2)` $\Rightarrow$ cherche si p_2 est dans ps avant de l'ajouter :

- $\star$ calcul de p_2`.haschode()` $\Rightarrow$ un indice n_2 différent de n_1 car calculé à partir de l'adresse de p_2 qui est différente de celle de p_2
- $\star$ recherche de $p2$ dans la liste à l'indice n_2 alors que p_1 est à l'indice n_1.
- $\star$ on ne le trouve pas ...
- $\star$ $\Rightarrow$ les 2 objets de même contenu seront ajoutés dans le Set !

Rédéfinir equals et hashCode

- où ?
  - dans la classes **`E`** d'éléments à ranger dans collection (seulement si E non standard)
- quand ?
  - si l'identité d'un objet donnée par son contenu, et si des multiples instances de même contenu peuvent exister
    - ex : classes **`AdressePostale`**, **`NumeroTelephone`**, etc.
    - on parle de *classes valeurs*
    - contra exemple : classe **`Compte`** (instance unique, possiblement partagée)
  - si **`E`** standard (**`String`**, **`Integer`**, etc), déjà rédéfinies !
- comment ?
  - redéfinir **`equals`** et **`hashCode`** toujours en même temps
  - prendre en compte les mêmes variables pour les deux
  - respecter les contrats de leur documentation Oracle
  - la plupart des IDE proposent de générer leurs définitions : il suffit d'indiquer les champs à prendre en compte.

Contraintes pour rédéfinir `equals`

Pour bien fonctionner **`equals`** doit satisfaire les propriétés suivantes :

- *réflexive* : `o1.equals(o1)` doit renvoyer true ;
- *symétrique* `o1.equals(o2)` et `o2.equals(o1)` $\Rightarrow$ même résultat
- *transitive* : si `o1.equals(o2)` et `o2.equals(o3)` $\Rightarrow$ `o1.equals(o3)`
- cohérente avec **`hashCode`** :
  - si 2 objets égaux par **`equals`**, alors ils renvoient le même **`hashCode`** !
  -  attention : la reciproque n'est pas vrai !

il faut s'assurer que **`equals`** vérifie ces propriétés ;
il faut redéfinir **`hashCode`** à chaque fois que l'on rédéfinit **`equals`** !
et il faut considérer les mêmes variables dans les deux !!

Comment redéfinir `equals` ?

En général :

- ⋆ on teste si même adresse mémoire ⇒ ils sont trivialement égaux,
- ⋆ sinon, on teste les cas trivialement faux en renvoyant `false` :
  - ⋆ si le pointeur vers l'autre objet est `null`,
  - ⋆ si l'autre objet n'est pas de même type que this
- ⋆ sinon, on teste l'égalité des variables pertinentes pour la comparaison (via `==` si primitives, via `equals` sinon)

Comment redéfinir `hashCode()` ?

Doit être fait en même temps et en cohérence avec la re définition de **`equals`**. Pas de recette unique.

En général :

- on utilise les mêmes variables qui ont été comparées par **`equals`**, de manière à garantir la cohérence avec celle-ci ;
- on calcule une valeur de type **`int`** pour chaque champ ;
- on les combine en les additionnant et les multipliant par un facteur constant, généralement un nombre premier ;

Exemple de rédéfinition (fait par Eclipse !)


```
@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + heure;
    result = prime * result + minutes;
    return result;
}
@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Heure other = (Heure) obj;
    if (heure != other.heure)
        return false;
    if (minutes != other.minutes)
        return false;
    return true;
}
```