Fondamenti di Java

hashCode

equals e hashCode

Programmers should take note that

any class that overrides the Object.equals method must also override the Object.hashCode method

in order to satisfy the general contract for the Object.hashCode method.

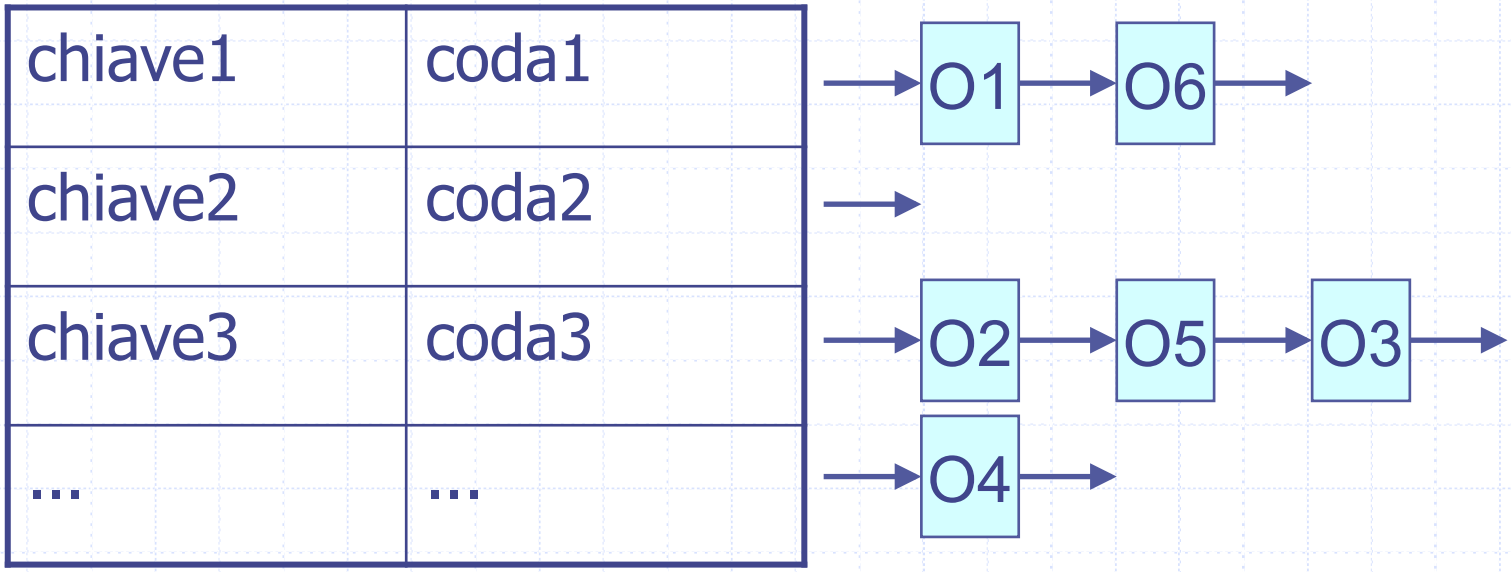
In particular, c1.equals(c2) implies that c1.hashCode()==c2.hashCode()

(the vice versa need not be true)

A che serve hashCode?

Tabelle associative

Dato un oggetto O1 è possibile calcolarne la chiave C1



esempio di hashCode()?

"ABBA" => 65+66+66+65 = 262

"ABBB" => 65+66+66+66 = 263

ma

"ABAB" => 65+66+65+66 = 262

equals e hashCode

if (c1.hashCode()!=c2.hashCode())

c1 e c2 diversi

if (c1.hashCode()==c2.hashCode())

per sapere se c1 è uguale a c2

devo usare la equals

E' un meccanismo di "fail quick"

equals e hashCode

c1.equals(c2) => c1.hashCode()==c2.hashCode()

Uguaglianza degli oggetti implica hashCode uguali

Diversità di hashCode implica non uguaglianza degli oggetti

c1.hashCode()!=c2.hashCode() => ! c1.equals(c2)

Non valgono i viceversa!!!

ATTENZIONE!

As much as is reasonably practical, the hashCode method defined by class Object does return distinct integers for distinct objects.

(This is typically implemented by converting the internal address of the object into an integer, but this implementation technique is not required by the Java programming language.)

Nel dubbio...

```
public int hashCode() {
      int hash = 0;
      return hash;
}
```

Inefficiente, ma corretto!

Per approfondimenti:

http://eclipsesource.com/blogs/2012/09/04/the-3-things-you-should-know-about-hashcode/

Regoletta...

Oggetti UGUALI => Hashcode UGUALI

Hashcode DIVERSI => Oggetti diversi

NON VALGONO I VICEVERSA!!!

Esercizio

Definire la classe "Automobile" con variabili di istanza Marca (es. VW), Tipo (Es, Golf), Colore (es. Bianco), Cilindrata (es. 1600), Targa, Proprietario.

Identificare diversi scenari di uso che abbiano differenti scenari di "equals":

- es. uno scenario in cui una Tipo e una Golf siano considerate "uguali", ed uno in cui due Golf di colore diverso sono considerate "uguali"

- implementare la equals per i diversi scenari.

Esercizio

Definire una hashCode corretta. Aggiungere tre diverse istanze di automobili "uguali" a un set, e controllare la dimensione del set ottenuto. Vi torna il valore ottenuto per la dimensione del set?

Definire una hashCode non corretta. Aggiungere tre diverse istanze di automobili "uguali" a un set, e controllare la dimensione del set ottenuto. Vi torna il valore ottenuto per la dimensione del set? Potete spiegare quel che osservate?

Fondamenti di Java

Collection: object ordering

Object ordering with Comparable

A List l1 may be sorted as follows:

```
Collections.sort(l1);
```

If the list consists of String elements, it will be sorted into lexicographic (alphabetical) order.

If it consists of Date elements, it will be sorted into chronological order.

How does Java know how to do this?

String and Date both implement the Comparable interface. The Comparable interfaces provides a *natural ordering* for a class, which allows objects of that class to be sorted automatically.

Object ordering

Ci sono due modi per ordinare oggetti:

The Comparable interface provides automatic *natural order* on classes that implement it.

The Comparator interface gives the programmer complete control over object ordering. These are *not* core collection interfaces, but underlying infrastructure.

Comparable Interface

int **compareTo**(**Object** o)

- Compares this object with the specified object for order.
- Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

- Definisce l'"ordinamento naturale" per la classe implementante.

Comparable

```
public class Car implements Comparable{

    public int maximumSpeed=0;
    public String name;
    Car(int v,String name) {
      maximumSpeed=v;
      this.name=name;
    }
    public int compareTo(Object o){
        if (!(o instanceof Car)) {
            System.out.println("Tentativo di comparare
                    mele e pere!");
            System.exit(1);
        }
        if (maximumSpeed<((Car)o).maximumSpeed) return -1;
        else return (1);
    }
}
```

Comparable

```
class TestCar{
    List<Car> macchine=null;
    public static void main(String[] args) {
        new TestCar();
    }
    TestCar(){
        macchine=new LinkedList<Car>();
        Car a=new Car(100,"cinquecento");
        macchine.add(a);
        Car b=new Car(250,"porsche carrera");
        macchine.add(b);
        Car c=new Car(180,"renault Megane");
        macchine.add(c);
        printMacchine();
        Collections.sort(macchine);
        printMacchine();

    }
```

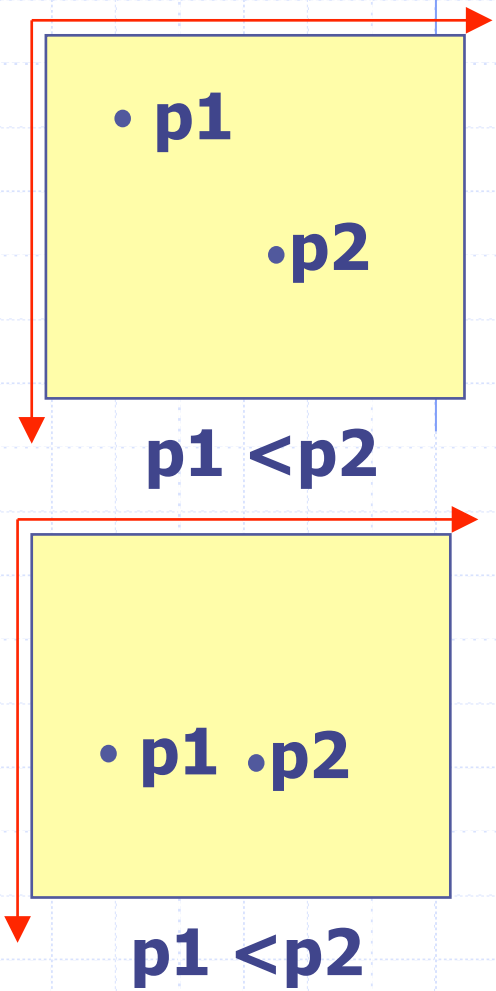
Comparable

```
void printMacchine(){ // con Iterator
        Iterator<Car> i=macchine.iterator();
        while (i.hasNext()) {
            System.out.println((i.next()).name);
        }
    }
}
```

```
void printMacchine(){ // senza Iterator
        for (i:macchine)
           System.out.println(i.name);
        }
    }
}
```

Comparable Interface

```
Class Point implements Comparable {
    int x; int y;
    ....
    int compareTo(Object p) {
        … check if Point…
        // ordino sulle y
        retval=y-((Point)p).y;
        // a parità di y ordino sulle x
        if (retval==0) retval=x-((Point)p).x;
        return retval;
}
```



Comparator Interface

int **compare(T o1, T o2)**

Compares its two arguments for order.

```
class NamedPointComparatorByXY
        implements Comparator {
   int compare (NamedPoint p1, NamedPoint p2) {
      // ordino sulle y
      retval=p1.y-p2.y;
      // a partità di y ordino sulle x
      if (retval==0) retval=p1.x-p2.x;
      return retval;
}
```

Comparator Interface

```
class NamedPointComparatorByName
                              implements Comparator {
   int compare (NamedPoint p1, NamedPoint p2) {
       //usa l'ordine lessicografico delle stringhe
       return (p1.getName().compareTo(p2.getName()));
  }
}
... In un metodo di un altra classe:
     // sia c una Collection di NamedPoints
     Comparator cmp1= new NamedPointComparatorByName();
     Comparator cmp2= new NamedPointComparatorByXY();
     List x = new ArrayList(c);
     Collections.sort(x,cmp1)
```

```
import java.util.*;
class EmpComparator implements Comparator {
   public int compare(Object o1, Object o2) {
      EmployeeRecord r1 = (EmployeeRecord) o1;
      EmployeeRecord r2 = (EmployeeRecord) o2;
      return r2.hireDate().compareTo(r1.hireDate());
} }
class EmpSort {
     EmpSort() {
        Collection employees = ... ; // Employee Database
        List emp = new ArrayList(employees);
        Collections.sort(emp, new EmpComparator());
        System.out.println(emp);
     }
     public static void main(String args[ ]) {new EmpSort();}
}
```

Esercizio

Scrivere una classe Impiegato con campi

- Stipendio
- Data di assunzione (usare la classe Data del primo esercizio fatto in classe!)
- Età
- Cognome

Scrivere comparatore naturale basato sul nome

Scrivere comparatori basati sugli altri campi

Creare una lista di impiegati, e ordinarla secondo I vari criteri.