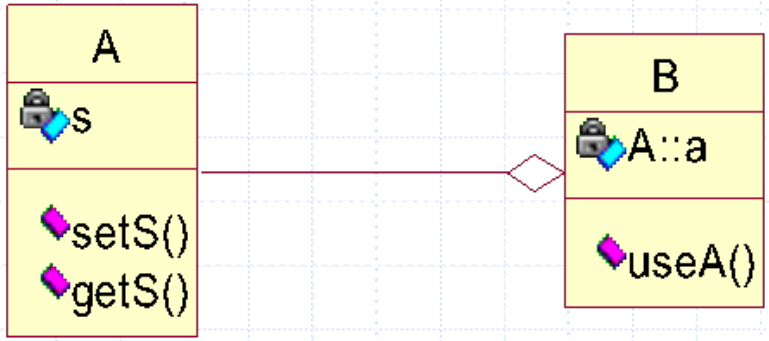
UML: Aggregazione

```
class A {
    int s;
    public void setS(int){…};
    public int getS() {…};
}
class B {A ob;
    public void useA() {…};
}
```

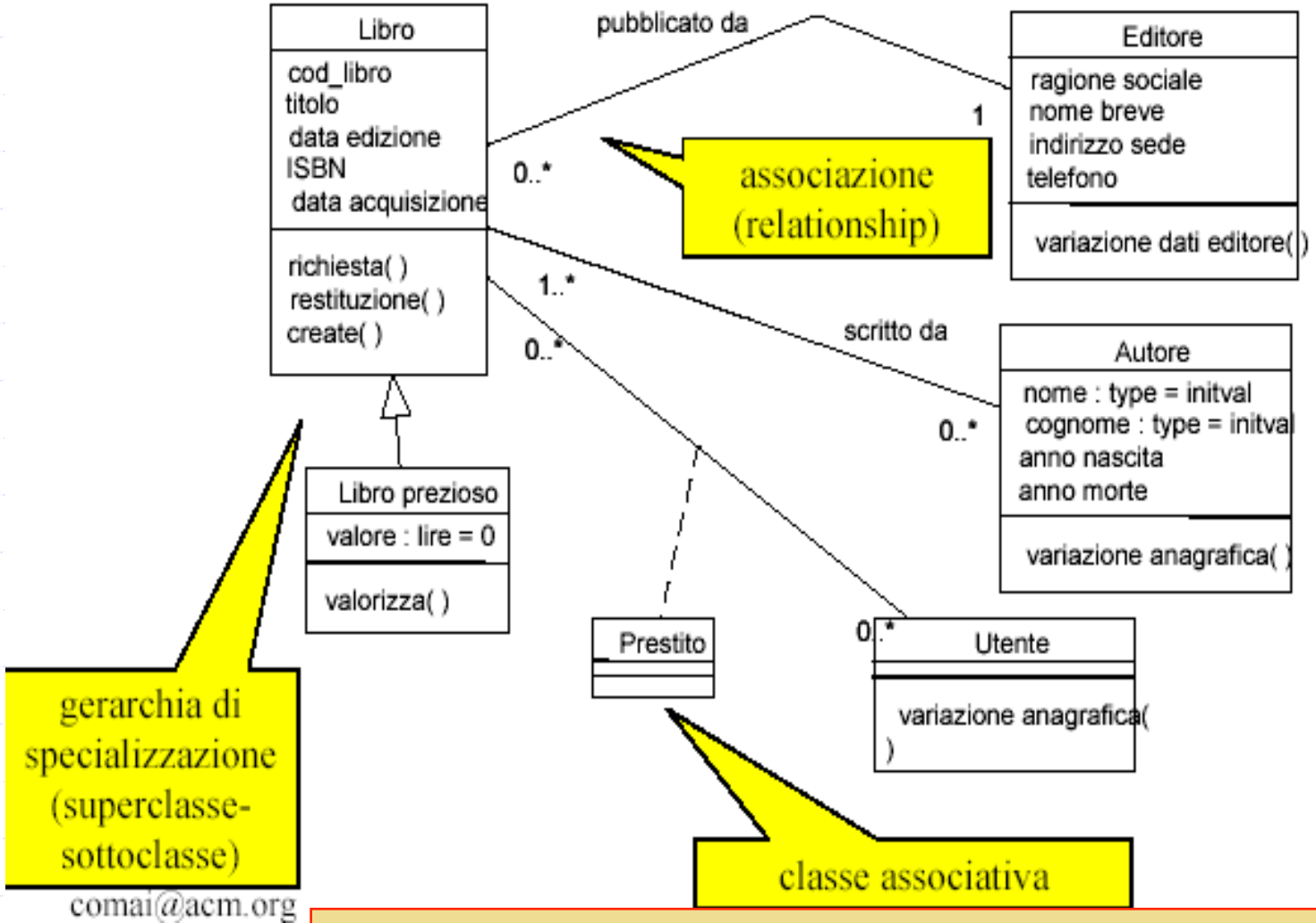


Aggregation - Composition

Use *aggregation (has-a)* when the lifecycle of the partecipating elements is different (one can exist without the other).

Use *composition (part-of)* when the *container* cannot be conceived without the *contained*.

UML – Class Diagram



comai@acm.org

Disegno ripreso da: Adriano Comai
http://www.analisi-disegno.com/a_comai/corsi/

Java

Generics

Uso di Generics nelle API di Java

```
// Removes 4-letter words from c. Elements must be strings
   static void expurgate(Collection c) {
      for (Iterator i = c.iterator(); i.hasNext(); )
        if (((String) i.next()).length() == 4)
          i.remove();
   }
```

Problemi?

Uso di Generics nelle API di Java

```
// Removes 4-letter words from c. Elements must be strings
   static void expurgate(Collection c) {
      for (Iterator i = c.iterator(); i.hasNext(); )
        if (((String) i.next()).length() == 4)
          i.remove();
   }
```

Here is the same example modified to use generics:

```
   // Removes the 4-letter words from c
   static void expurgate(Collection<String> c) {
      for (Iterator<String> i = c.iterator(); i.hasNext(); )
        if (i.next().length() == 4)
          i.remove();
   }
```

A partire da Java 5 molte classi sono state riscritte usando i generics

Vantaggi

No cast

Fail quick!

Uso semplice delle Collections

```
Collection insieme<Point>=new LinkedList<Point>t();

Point a=new Point(1,2); insieme.add(a);
Point b=new Point(3,4); insieme.add(b);

for (Point k:insieme) {
   System.out.println(k);
}
```

Sezione: Costruttori

Costruttori

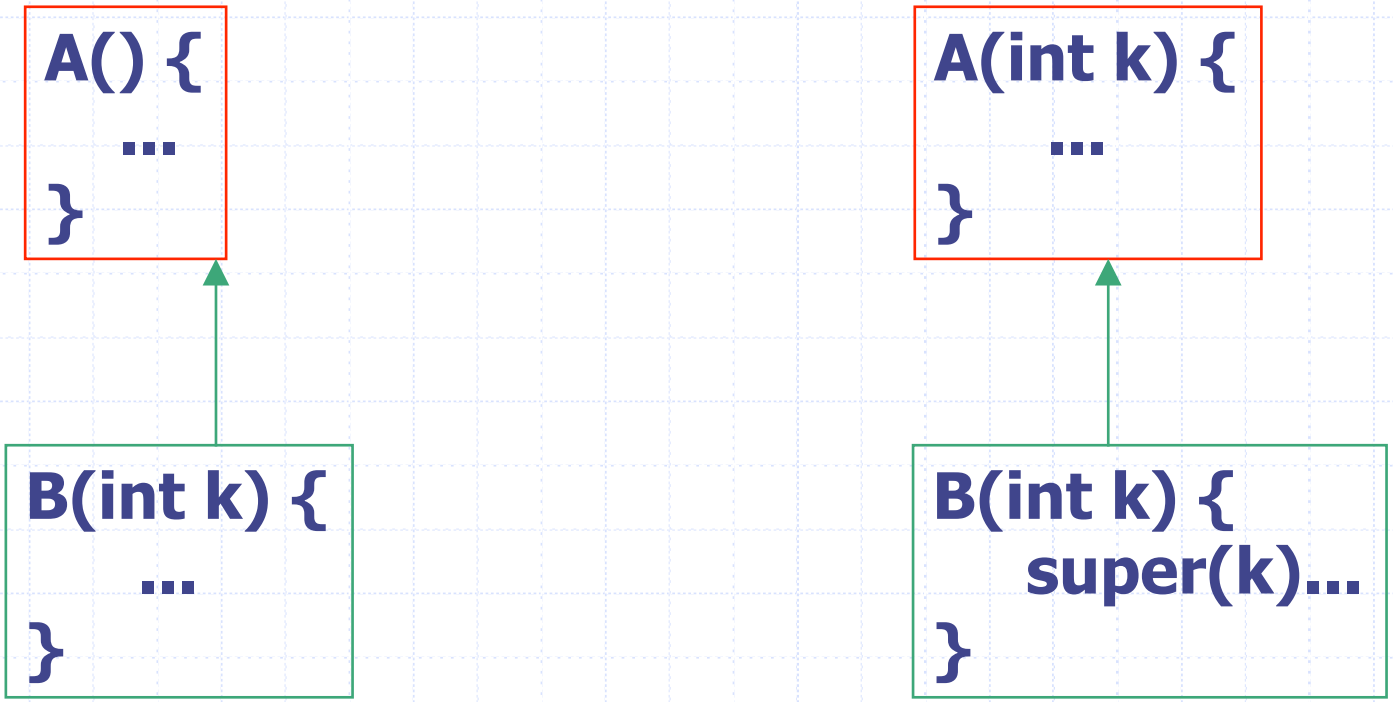
Definizione dei costruttori

Se per una classe A non scrivo nessun costruttore, il sistema automaticamente crea il costruttore A();

Se invece definisco almeno un costruttore non void, ad es. A(int s), il sistema non crea il costruttore A();

Definizione dei costruttori

Se B è figlia di A, il costruttore di B come prima cosa invoca A(), a meno che la prima istruzione non sia una super.



Invocazione dei costruttori

```
public class A {
  public A() {
    System.out.println("Creo A");
  }
}
public class B extends A {
  public B() {
    System.out.println("Creo B");
  }
  public B(int k) {
    System.out.println("Creo B_int”);
  }
}
```

Output:
Creo A
Creo B_int

```
public static void main(String [] a) {
    B b=new B(1);
 }
```

Invocazione dei costruttori

```
public class A {
  public A(int k) {
    System.out.println("Creo A");
  }
}
public class B extends A {
  public B() {
    System.out.println("Creo B");
  }
  public B(int k) {
    System.out.println("Creo B_int”);
  }
}
```

Output:
ERRORE !

Perchè ?

```
public static void main(String [] a) {
    B b=new B(1);
  }
```

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)

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!!!

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"

esempio di hashCode()?

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

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

ma

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

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

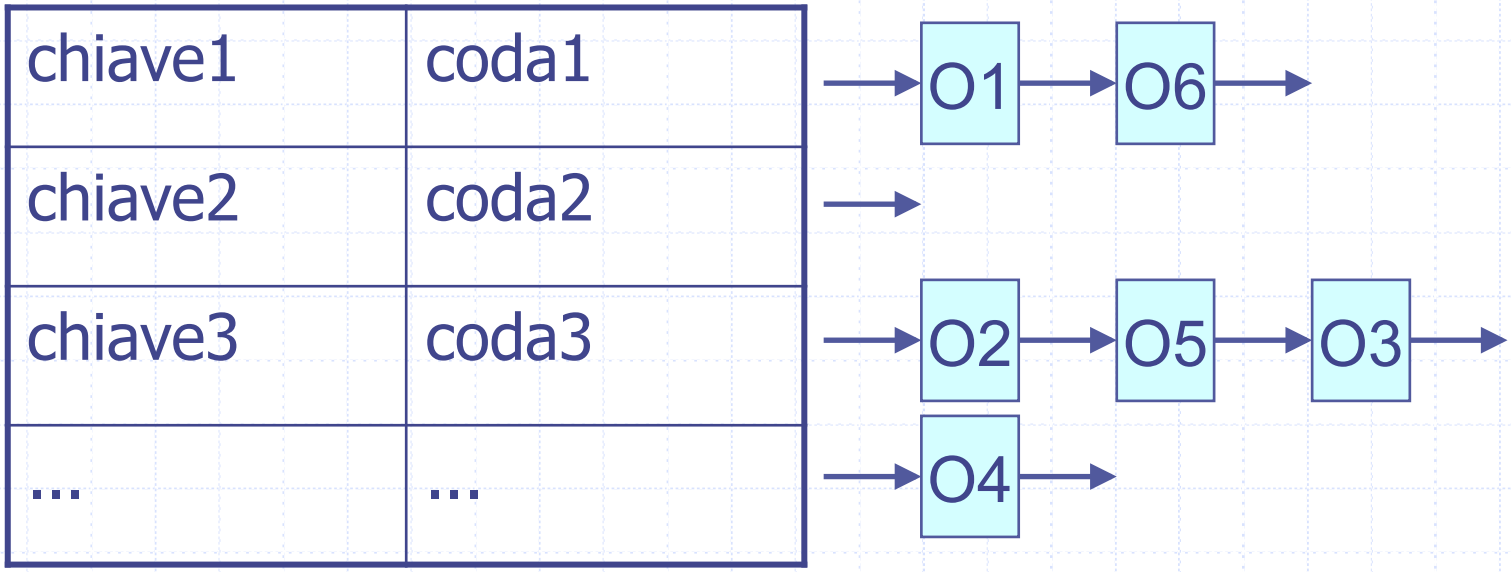
Hashcode DIVERSI => Oggetti diversi

NON VALGONO I VICEVERSA!!!

A che serve hashCode?

Tabelle associative

Dato un oggetto O1 è possibile calcolarne la chiave C1



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?