Universita` di Roma, “La Sapienza” Facolta` di Ingegneria

Corso di

Laboratorio di Programmazione

Anno Accademico 2004/05

Corso di Laurea in Ingegneria Informatica

Prof. Giuseppe De Giacomo (A-L) e Prof. Paolo Liberatore (M-Z)

Realizzazione di una classe che implementa dell’interfaccia Set del Collections Framework

L’interfaccia Set

```
public interface Set extends Collection {

    // Basic Operations
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(Object element);    // Optional
    boolean remove(Object element); // Optional
    Iterator iterator();

    // Bulk Operations
    boolean containsAll(Collection c);
    boolean addAll(Collection c);    // Optional
    boolean removeAll(Collection c); // Optional
    boolean retainAll(Collection c); // Optional
    void clear();                    // Optional

    // Array Operations
    Object[] toArray();
    Object[] toArray(Object[] a);
}
```

Struttura base di una classe

```
public class XXX  {

  // campi dati (rappresentazione degli oggetti XXX)

  // costruttori

  // funzioni proprie della classe

  // funzioni speciali ereditate da Object

  // funzioni non pubbliche ausiliarie

}
```

Realizzazione dell'interfaccia Set

```
import java.util.*;

public class InsiemeLista implements Set {

  // campi dati

  // costruttori

  // funzioni proprie della classe
  // (realizzazione delle funzioni di Set)

  // basic operations
  public int size() { ... }

  public boolean isEmpty() { ... }

  public boolean contains(Object e) { ... }

  public boolean add(Object e) { ... }

  public boolean remove(Object e) { ... }

  public Iterator iterator() { ... }

  . . .
```

Realizzazione dell'interfaccia Set (cont.)

```
  . . .
  // bulk Operations
  public boolean containsAll(Collection c) { ... }

  public boolean addAll(Collection c){ // opzionale - non supportata
    throw new UnsupportedOperationException("addlAll() non e' supportata");
  }
  public boolean removeAll(Collection c) { // opzionale - non supportata
    throw new UnsupportedOperationException("removeAll() non e' supportata");
  }
  public boolean retainAll(Collection c) { // opzionale - non supportata
    throw new UnsupportedOperationException("retainAll() non e' supportata");
  }
  public void clear() { // opzionale - non supportata
    throw new UnsupportedOperationException("clear() non e'  supportata");
  }

  // array operations
  public Object[] toArray() { ... }

  public Object[] toArray(Object[] a) { ... }

  // funzioni speciali ereditate da Object

  // funzioni ausiliarie
}
```

Scelta della rappresentazione degli oggetti InsiemeLista

```
import java.util.*;

class Nodo {
    Object info;
    Nodo next;
}


public class InsiemeLista implements Set {

  // campi dati
  protected Nodo inizio;
  protected int cardinalita;


  // costruttori

  // funzioni proprie della classe
  // (realizzazione delle funzioni di Set)
  . . .
```

Costruttore per InsiemeLista

```
import java.util.*;

class Nodo {
    Object info;
    Nodo next;
}

public class InsiemeLista implements Set {

  // campi dati
  protected Nodo inizio;
  protected int cardinalita;


  // costruttori
  public InsiemeLista() {
    inizio = null;
    cardinalita = 0;
  }

  // funzioni proprie della classe
  // (realizzazione delle funzioni di Set)
  . . .
```

Scelta del trattamento delle funzioni speciali

```
public class InsiemeLista implements Set, Cloneable {
                                          //^^^^^^^^^
  // campi dati
  . . .
  // costruttori
  . . .
  // funzioni proprie della classe
  // (realizzazione delle funzioni di Set)
  . . .

  // funzioni speciali ereditate da Object
  public boolean equals(Object o) {
    // realizzare uguaglianza profonda
    // (verifica sugli elementi della lista che rappresenta l'insieme)
  }

  public Object clone() {
    // realizzare copia profonda
    // (copiare la lista che rappresenta l'insieme)
  }

  public String toString() { ... }

  // funzioni ausiliarie
}
```

Realizzazione delle funzioni: equals()

```
. . .
// campi dati
protected Nodo inizio;
protected int cardinalita;
. . .

// funzioni speciali ereditate da Object
public boolean equals(Object o) {
  if (o != null && getClass().equals(o.getClass())) {
    InsiemeLista ins = (InsiemeLista)o;
    if (cardinalita != ins.cardinalita) return false;
      // ins non ha la cardinalita' giusta
    else {
      // verifica che gli elementi nella lista siano gli stessi
      Nodo l = inizio;
      while (l != null) {
        if (!appartiene(l.info,ins.inizio)) //appartiene(): funz. ausiliaria
          return false;
        l = l.next;
      }
      return true;
    }
  }
  else return false;
}
. . .
```

Realizzazione delle funzioni: clone()

```
. . .
// campi dati
protected Nodo inizio;
protected int cardinalita;
. . .

// funzioni speciali ereditate da Object
. . .
 public Object clone() {
   try {
     InsiemeLista ins = (InsiemeLista) super.clone();
     // chiamata a clone() di Object che esegue la copia campo a campo;
     // questa copia e' sufficiente per i campi cardinalita e elemClass
     // ma non per il campo inizio del quale va fatta una copia profonda
     ins.inizio = copia(inizio); //copia() - funz. ausiliaria
     return ins;
   } catch(CloneNotSupportedException e) {
     // non puo' accadere perche' implementiamo l'interfaccia cloneable,
     // ma va comunque gestita
     throw new InternalError(e.toString());
   }
 }
. . .
```

Realizzazione delle funzioni: toString()

```
. . .
// campi dati
protected Nodo inizio;
protected int cardinalita;
. . .
// funzioni speciali ereditate da Object
. . .
public String toString() {
  String s = "{ ";
  Nodo l = inizio;
  while (l != null) {
    s = s + l.info + " ";
    l = l.next;
  }
  s = s + "}";
  return s;
}
. . .
```

Realizzazione delle funzioni: size(), isEmpty(), contains()

```
. . .
// campi dati
protected Nodo inizio;
protected int cardinalita;
. . .
// funzioni proprie della classe
// (realizzazione delle funzioni di Set)

// basic operations

public int size() {
  return cardinalita;
}

public boolean isEmpty() {
  return inizio == null;
}

public boolean contains(Object e) {
  return appartiene(e,inizio);
}
. . .
```

Realizzazione delle funzioni: add(), remove()

```
protected Nodo inizio;
protected int cardinalita;
. . .
// basic operations
public boolean add(Object e) {
  if (appartiene(e,inizio)) return false;
  else {
    Nodo l = new Nodo();
    l.info = e;
    l.next = inizio;
    inizio = l;
    cardinalita = cardinalita + 1;
    return true;
  }
}

public boolean remove(Object e) {
  if (!appartiene(e,inizio)) return false;
  else {
    inizio = cancella(e,inizio);
    cardinalita = cardinalita - 1;
    return true;
  }
}
. . .
```

Realizzazione delle funzioni: iterator()

```
public class InsiemeLista implements Set, Cloneable {

// campi dati
protected Nodo inizio;
protected int cardinalita;
. . .
// funzioni proprie della classe
// (realizzazione delle funzioni di Set)// basic operations
. . .
public Iterator iterator() {
  return new IteratorInsiemeLista(this);  //nota!
}
. . .
```

L'interfaccia Iterator

```
public interface Iterator {
  boolean hasNext();
  Object next();
  void remove(); // Optional
}
```

Esempio di uso dell'interfaccia Iterator

```
class Cliente DiSet {
  . . .
  public static void StampaElementi(Set s) {
    Iterator it = s.iterator();
    while(it.hasNext()) {
      Object o = it.next();
      System.out.println(o);
    }
  }
  . . .
}
```

La classe IteratorInsiemeLista

```
// Quanto segue deve stare nello stesso package di InsiemeLista

import java.util.*;

class IteratorInsiemeLista implements Iterator {  //nota non e' pubblico!
  private Nodo rif;

  public IteratorInsiemeLista(InsiemeLista ins) {
    rif = ins.inizio; //nota inizio e' accessibile perche'
                      //InsiemeLista e' nello stesso package!!!
  }

  // Realizzazione funzioni di Itarator
  public boolean hasNext() {
    return rif != null;
  }

  public Object next() {
    Object e = rif.info;
    rif = rif.next;
    return e;
  }

  public void remove() {
    throw new UnsupportedOperationException("remove() non e' supportata");
  }
}
```

Le funzioni toArray()

```
. . .
// array operations
public Object[] toArray() {
  Object[] a = new Object[size()];
  int i = 0;
  Iterator it = iterator();
  while (it.hasNext()) {
    a[i] = it.next();
    i++;
  }
  return a;
}

public Object[] toArray(Object[] a) {
  // se possibile usa riempi l'array a e restituiscilo
  // se a e' troppo piccolo, crea un nuovo array
  // i cui elementi hanno il tipo degli elementi dell'array
  // (riferito da) a
}
. . .
```

Le funzioni toArray()

```
java.lang.reflect.Array;
. . .
public Object[] toArray(Object[] a) {
  if (a.length < size()) {
    //prendi il tipo degli elementi di a
    //e costruisci un array di elementi di quel tipo
    Class elemClass = a.getClass().getComponentType();
    a = (Object[])Array.newInstance(elemClass,size());
  }
  //riempi l'array con gli elementi della collezione
  int i = 0;
  Iterator it = iterator();
  while (it.hasNext()) {
    a[i] = it.next();
    i++;
  }
  //se l'array e' piu' grande poni a null gli elem. rimanenti
  for (; i < a.length; i++)
    a[i] = null;
  return a;
}
. . .
```

Collezioni omogenee

L'implementazione InsiemeLista di Set, cosi` come tutte le implementazioni predefinite del Collections Framwork, realizzano collezioni **disomogenee**, cioe` i cui elementi possono essere di tipi diversi (es. In prima posizione possiamo avere una String, in seconda un Integer, ecc).

Come possiamo realizzare collezioni **omogenee**, cioe` i cui elementi siano tutti dello stesso tipo?

Imporre che una collezione sia omogenea in modo statico (cioe` a tempo di compilazione) non si puo` in Java!

Ma si puo` imporlo in modo dinamico (cioe` a tempo di esecuzione):

- dobbiamo memorizzare nella classe il tipo degli elementi,
- dobbiamo verificare durante l'inserimento che gli oggetti che inseriamo siano del tipo giusto.

Insieme di elementi omogenei

- Costruiamo una classe **`InsiemeListaOmogeneo`** che implementa **`Set`**, ma realizza **insiemi omogenei**.
- Riusiamo l'implementazione **`InsiemeLista`**, realizzando **`InsimeListaOmogeneo`** come una sua **classe derivata**, dove faremo overriding dei metodi che vanno modificati

InsiemeListaOmogeneo

```
public class InsiemeListaOmogeneo extends InsiemeLista {

  // campi dati
  protected Class elemClass;

  // costruttori
  public InsiemeListaOmogeneo(Class cl) {
    super();
    elemClass = cl;
  }

  public InsiemeListaOmogeneo() {
    this(Object.class);
  }

  ...

}
```

InsiemeListaOmogeneo

```
...

// overriding delle basic operations
public boolean contains(Object e) {
  if (!elemClass.isInstance(e)) return false;
  else return super.contains(e);
}

public boolean add(Object e) {
  if (!elemClass.isInstance(e)) return false;
  else return super.add(e);
}

public boolean remove(Object e) {
  if (!elemClass.isInstance(e)) return false;
  else return super.remove(e);
}

// overriding delle bulk operations: non serve

// overriding delle array operations: non serve

...
```

InsiemeListaOmogeneo

```
...

// overriding delle funzioni speciali ereditate da Object
public boolean equals(Object o) {
  if (super.equals(o)) {
    InsiemeListaOmogeneo ins = (InsiemeListaOmogeneo)o;
    if (elemClass.equals(ins.elemClass)) return true;
    else return false;
  }
  else return false;
}

public Object clone() {
  InsiemeListaOmogeneo ins = (InsiemeListaOmogeneo)super.clone();
  return ins;  // Nota: clone() di Object si occupa gia' della copia
               //(superficiale) del campo elemClass.

  //oppure semplicemente: return super.clone();
}

// overriding di toString() non serve

...
```

Il codice

- InsiemeLista.java
- IteratorInsiemeLista.java
- InsiemeListaOmogenea.java
- Main.java