Astrazioni sul controllo: iteratori

Collezionare oggetti

- Problema: **raggruppare** un insieme di oggetti insieme e accedere ad essi secondo regole particolari
  - Per esempio: coda, pila, liste circolari, mappe, …
  - Spesso l'utilizzo degli array non è sufficiente

- Soluzione 1: Realizzare una propria classe che fornisce i metodi di accesso opportuni
- Soluzione 2: Utilizzare classi già pronte fornite da Java, scegliendo quella più opportuna ai propri bisogni

- Il **Java Collections Framework (JCF)** fornisce un insieme molto ampio di classi (concrete) in grado di collezionare oggetti fornendo (estese dalle proprie classi) relative a Pile, Code, Insiemi, Liste, Mappe, Insiemi ordinati ecc …

(JCF) Java Collections Framework

- Collection:
  - List
    - ArrayList
    - LinkedList
    - Vector
  - Set
    - SortedSet
    - HashSet
    - LinkedHashSet
- Altre interfacce disponibili
  - Queue, Dequeue, Stack, Map, SortedMap …

Da Thinking in Java

- Collection
  - Group of objects, known as its elements
  - Some collections allow duplicate elements and others do not
  - Some are ordered and others unordered
    - boolean add(Object e)
    - void clear()
    - boolean contains(Object o)
    - Iterator iterator()
    - boolean remove(Object o) int size()
- List
  - An ordered collection (also known as a sequence)
  - the user of this interface has precise control over where in the list each element is inserted
    - E get(int index)
    - E set(int index, E element)
- Set
  - A collection that contains no duplicate elements
- SortedSet
  - A collection that contains sorted elements

Astrazione

- Astrarre: ignorare dettagli non essenziali per lo scopo che si vuole raggiungere
- L'informatica è basata interamente sul concetto di astrazione:
  - Il calcolatore è un insieme di interruttori
    - Gli interruttori hanno due posizioni: aperto ("0") o chiuso ("1")
  - I programmi descrivono come "assegnare" i valori
  - Astrazioni nella programmazione in-the-small:
    - Concetto di variabile e di tipo
    - Istruzioni di controllo (while, if, ...)

Nuove iterazioni

- Definendo un nuovo tipo come **collezione** di oggetti si vorrebbe disporre anche di un'operazione che consenta iterazioni
- Se la collezione è implementa List, è facile:
  - Organizzazione lineare degli elementi
- Con altre collezioni?
  - Il principio dell' Information Hiding non consente (giustamente!) l'accesso diretto all'organizzazione interna di un contenitore!

Iterare su collezioni

- Il for generalizzato non basta
  - Fornisce una sola politica di scansione
  - Come fare a prevedere in anticipo tutti gli usi?
- Soluzione inefficiente: avere un metodo che restituisce un array con tutti gli elementi
- Soluzione inefficiente: definire size e get
  - for (int i = 0; i < set.size(); i++)
    System.out.println(set.get(i));
  - Su molte strutture dati l'accesso casuale all'i-esimo elemento può essere inefficiente!

da Thinking in Java

- There's not much you can do with the Java Iterator except:
  - Ask a collection to hand you an Iterator using a method called iterator()
  - This Iterator will be ready to return the first element in the sequence on your first call to its next() method
  - Get the next object in the sequence with next()
  - See if there are any more objects in the sequence with hasNext()
  - Remove the last element returned by the iterator with remove()

Soluzione generale

- Introdurre oggetti detti iteratori in grado di “spostarsi” sugli elementi dei contenitori
  - L’iteratore rappresenta **un'astrazione** del concetto di “puntatore a un elemento del contenitore” e permette di scorrere il contenitore senza bisogno di conoscere l'effettivo tipo degli elementi
  - Esempio con IntSet: oggetto “iteratore su IntSet” con metodi: next(), per restituire il prossimo elemento della collezione, e hasNext() per verificare se siamo sull’ultimo elemento

Interface Iterator

- In java.util

```
public interface Iterator<E> {
public boolean hasNext();
public E next() throws NoSuchElementException;
public void remove(); // OPZIONALE
}
```

Interfaccia Iterable (1)

- Per standardizzare l'uso degli iteratori, in Java esiste anche l'interfaccia Iterable (in java.lang)
- Le classi che implementano l'interfaccia Iterable sono tipicamente collezioni che forniscono un metodo di nome standard iterator() per ottenere un oggetto iteratore, con cui scandirne gli elementi

Method Summary	
Iterator<T>	**iterator()** Returns an iterator over a set of elements of type T.

Method Detail

iterator

Iterator<T> **iterator()**

Returns an iterator over a set of elements of type T.

Returns:

an Iterator.

Interfaccia Iterable (2)

- E’ necessario implementare Iterable se si vuole costruire una collezione da usare con un for generalizzato
  - Il prezzo della semplificazione lo paga l’implementatore, che deve realizzare il metodo iterator()
  - …in maniera efficiente, o nessuno lo userà!

Iteratore per IntSet: Esempio

```
public static boolean stampa(IntSet set){
   for (Iterator<Integer> itr = set.elements(); itr.hasNext();)
      System.out.println(itr.next());
}


public static boolean insiemePositivo(IntSet set){
   for (Iterator<Integer> itr = set.elements(); itr.hasNext();)
      if (itr.next()<0) return false;
   return true;
}
```

Iterare su collezioni qualunque

- È un'astrazione sul controllo molto potente…
- …"the true power of the Iterator: the ability to separate the operation of traversing a sequence from the underlying structure of that sequence"

```
public class Iterations {
   public static void printAll(Iterator<E> itr){
      while(itr.hasNext())
         System.out.println(itr.next());
         // NB: conv. automatica a String
   }

   public static boolean cerca(Iterator<E> itr, <E> o){
      while (itr.hasNext())
         if (o.equals(itr.next())) return true;
         return false;
   }
}
```

Implementazione con classe interna

```
public class Poly implements Iterable{
   private int[] trms; // coefficienti dei termini
   private int deg; // il grado del polinomio

   public Iterator<Integer> terms() {
      return new PolyGen(this); }

   // classe interna
   private class PolyGen implements Iterator<Integer> {
     // tutte le altre operazioni della classe
   }
}
```

La classe per l'iteratore

```
import java.util.Iterator;
import java.util.NoSuchElementException;

// restituisce il grado (dei termini non nulli)

class PolyGen implements Iterator<Integer> {
     private Poly p;  // il Poly da iterare
     private int n;   // primo elemento non "consumato"

     PolyGen(Poly lui) { p=lui; n = 0; }

     public boolean hasNext(){
          while(n <= p.deg) {
               if(p.trms[n] != 0) return true;
               else n++;
          }
          return false;
     }

     public Integer next() throws NoSuchElementException {
          if(hasNext()) return n++;
          else throw new NoSuchElementException("Poly.terms");
     }
}
```

Iteratori "stand alone"

- Un iteratore potrebbe **non** far parte di una classe, ma essere una procedura statica “stand alone”
- Per esempio, un iteratore che genera tutti i numeri di Fibonacci
  - …o un generatore di numeri primi
- Per questo motivo, serve definire la classe interna come static: un metodo static di una classe non può accedere a un costruttore di una classe interna non static

Esempio

```
public class Nums {
   public static Iterator<Integer> allFibos( ) {
      return new FibosGen( );
   }

   // classe interna
   private static class FibosGen implements Iterator<Integer> {
      private int prev1, prev2; // i due ultimi generati
      private int nextFib; // prossimo Fibonacci

      FibosGen() {prev2=1; prev1=0;}

      public boolean hasNext() {return true;}
      public Integer next() {
        nextFib = prev1+prev2;
        prev2=prev1; prev1=nextFib;
        return nextFib ;
      }
   }
}
```

Uso dell'iteratore per Fibonacci

```
public static void printFibos(int m) {
   Iterator<Integer> g = Nums.allFibos();
   while (g.hasNext()) { // sempre true…
      int p = g.next();
      if (p > m) return;
      System.out.println("Il prox Fibonacci e` " + p);
   }
}
```

Osservazioni

- “An iterator is an object whose job is to move through a sequence of objects and select each object in that sequence without the client programmer knowing or caring about the underlying structure of that sequence; in addition, an iterator is usually what’s called a “light-weight” object… one that’s cheap to create”
- Una collezione può avere più tipi di iteratori (e quindi più metodi per generarli)

- Esempio: una collezione potrebbe avere:

```
public Iterator<T> smallToBig()
public Iterator<T> bigToSmall()
public Iterator<T> oneEveryTwo()
```

Collezioni mutabili e metodo remove

- Remove è un'operazione "opzionale": se non è implementata, quando invocata viene lanciata UnsupportedOperationException
- In effetti, raramente è utile modificare una collezione durante un' iterazione: più spesso si modifica alla fine
  - Esempio: trovo elemento e lo elimino, poi iteratore non più usato

Collezioni mutabili e metodo remove

- Semantica di remove() assicurata dal “contratto” dell’interfaccia:
  - …elimina dalla collezione l’ultimo elemento restituito da next(), sempre che:
    - Venga chiamata una sola volta per ogni chiamata di next()
    - La collezione non venga modificata in qualsiasi altro modo (diverso dalla remove) durante l’iterazione
  - …altrimenti
    - Viene lanciata un’eccezione IllegalStateException se il metodo next() non è mai chiamato o se remove() già invocata dopo ultima chiamata di next()
    - Semantica della remove() **non specificata** se la collezione modificata in altro modo (diverso dalla remove) durante l’iterazione