Corso di Programmazione
Anno Accademico 2015–2016
Prof. Giovanni Pighizzini

Note ed esercizi aggiuntivi

Gli esercizi proposti sono utili per rivedere gli esempi riportati, che sono stati sviluppati e discussi in dettaglio a lezione.

8. Ancora tipi generici, array

Esempio. Leggere una sequenza di interi terminata da `0` (che non fa parte della sequenza). Visualizzare poi la sequenza letta.

```
import prog.io.ConsoleInputManager;
import prog.io.ConsoleOutputManager;
import prog.utili.Sequenza;

class ElencoInteri {
  public static void main(String[] a) {
    ConsoleInputManager in = new ConsoleInputManager();
    ConsoleOutputManager out = new ConsoleOutputManager();

    Sequenza<Integer> seq = new Sequenza<Integer>();

    //legge i numeri e li memorizza nella sequenza
    int numero = in.readInt("Inserisci il prossimo numero (0 per terminare) ");
    while (numero != 0) {
      seq.add(numero);  // qui viene usato l'autoboxing
      numero = in.readInt("Inserisci il prossimo numero (0 per terminare) ");
    }

    //visualizzazione della sequnza
    out.println("Numeri inseriti: ");
    for (Integer n: seq)
      out.println(n);
  }
}
```

Note
Nell'invocazione `seq.add(numero)` il parametro atteso dal metodo `add` è un riferimento di tipo `Integer`, mentre l'argomento indicato è di tipo `int`. Ciò è possibile grazie al meccanismo dell'autoboxing. In sostanza, al momento della chiamata viene costruito un oggetto di tipo `Integer` che rappresenta il valore contenuto nella variabile `numero`. Al metodo `add` viene fornito come argomento il riferimento a tale oggetto. L'uso di questo meccanismo viene deciso dal compilatore durante

l'analisi dei tipi (type checking) sulla base del tipo dell'argomento fornito al metodo nella chiamata (cioè `int`) e del parametro atteso dal metodo `add` (cioè `Integer`). In sostanza, il compilatore traduce l'invocazione come fosse `seq.add(new Integer(numero))`.

Esercizio 8.1
Fate riferimento all'esempio precedente.

- Provate a dichiarare la variabile numero di tipo `Integer` anziché di tipo `int`. Il comportamento del programma cambia? Individuate tutti i punti del programma in cui, dopo questa modifica, vengono effettuate operazioni di unboxing o di autoboxing. Dal punto di vista dell'efficienza, ritenete sia meglio dichiarare numero di tipo `int` o di tipo `Integer`? Dal punto di vista della chiarezza del codice, ci sono differenze significative?

- Cosa succede se dichiarate la variabile `n` utilizzata nel ciclo *for-each* di tipo `int` anziché di tipo `Integer`? Spiegate le ragioni di questo comportamento.

- Cosa succede se sostituite la riga

  ```
  Sequenza<Integer> seq = new Sequenza<Integer>();
  ```

  con la riga seguente?

  ```
  Sequenza<int> seq = new Sequenza<int>();
  ```

  Spiegate le ragioni di questo comportamento.

- Modificate la fase di visualizzazione in modo che in caso di sequenza vuota fornisca un apposito messaggio (la classe `Sequenza` fornisce un metodo `isEmpty` per controllare se una sequenza sia vuota, trovate i dettagli nella documentazione).

- Modificate l'applicazione in modo che visualizzi la sequenza anche in ordine crescente. Servitevi della classe `SequenzaOrdinata`.

Esercizio 8.2
Scrivete un'applicazione che legga un intero `n` e produca un elenco alfabetico dei tutti i numeri ordinali da `1` a `n` e un elenco alfabetico di tutti i numeri cardinali da `1` a `n`. (Servitevi delle classi `Intero` e della classe `SequenzaOrdinata`). Ecco l'output che deve essere prodotto, se il numero fornito è `5`:

```
Sequenza alfabetica numeri ordinali:
primo
quarto
quinto
secondo
terzo

Sequenza alfabetica numeri cardinali:
cinque
due
quattro
tre
uno
```

Esercizio 8.3
Scrivete un'applicazione che legga un intero `n` e produca un elenco alfabetico dei numeri cardinali da `1` a `n`, indicando la posizione di ciascuno di essi, come numero ordinale. Ad esempio, se viene inserito `5`, il risultato prodotto dovrà essere:

```
Sequenza alfabetica numeri cardinali:
primo numero in ordine alfabetico: cinque
secondo numero in ordine alfabetico: due
terzo numero in ordine alfabetico: quattro
quarto numero in ordine alfabetico: tre
quinto numero in ordine alfabetico: uno
```

Esercizio 8.4
Scrivete un'applicazione che legga un numero `n` e un numero `max` e generi `n` numeri a caso compresi tra `1` e `max`. L'applicazione dovrà poi elencare i numeri letti in ordine crescente. (Servitevi delle classi `Integer`, `java.util.Random` e `SequenzaOrdinata`.)

Esercizio 8.5
Modificate l'applicazione precedente, in modo che fornisca anche un elenco in ordine alfabetico dei numeri generati (utilizzate anche la classe `Intero`).

Esempio. Lettura di una sequenza di stringhe terminata dalla stringa vuota (che non fa parte della sequenza) e riscrittura sul monitor della sequenza stessa.

```
import prog.io.ConsoleInputManager;
import prog.io.ConsoleOutputManager;

class SequenzaStringhe {

  public static void main(String[] a) {
    ConsoleInputManager in = new ConsoleInputManager();
    ConsoleOutputManager out = new ConsoleOutputManager();

    final int MAX = 10;  //capacita' dell'array
    String[] tab = new String[MAX];

    //fase di lettura
    int pos = 0;  //prima posizione libera
    String x = in.readLine("Stringa? ");
    while (!x.equals("")) {
      tab[pos++] = x;
      x = in.readLine("Stringa? ");
    }

    //fase di scrittura
    for (int i = 0; i < pos; i++)
      out.println(tab[i]);
  }
}
```

Note

- Si osservi l'uso dell'operatore di *incremento postifisso* per accedere alla posizione `pos` dell'array, cui viene assegnata la stringa letta, e per predisporre, allo stesso tempo, `pos` sulla posizione successiva.
- Se l'utente inserisce più di `10` stringhe (costante `MAX`) l'esecuzione dell'applicazione viene interrotta dal tentativo di accedere a una posizione dell'array non esistente. Provate a far eseguire l'applicazione per vedere il messaggio di errore che viene visualizzato.

Nella versione seguente la lettura delle stringhe viene terminata nel caso l'array sia completamente riempito.

```
import prog.io.ConsoleInputManager;
import prog.io.ConsoleOutputManager;

class SequenzaStringhe {

  public static void main(String[] a) {
    ConsoleInputManager in = new ConsoleInputManager();
    ConsoleOutputManager out = new ConsoleOutputManager();

    final int MAX = 10;  //capacita' dell'array
    String[] tab = new String[MAX];

    //fase di lettura
    int pos = 0;  //prima posizione libera
    String x = in.readLine("Stringa? ");
    while (!x.equals("") && pos < MAX) {
      tab[pos++] = x;
      if (pos < MAX)
        x = in.readLine("Stringa? ");
    }

    //fase di scrittura
    for (int i = 0; i < pos; i++)
      out.println(tab[i]);
  }
}
```

Note
La capacità dell'oggetto array è stabilita al momento della creazione (invocazione del costruttore `new String[...]`). Ciò limita l'utilità di questo codice: il ciclo di lettura termina comunque quando si raggiunge la capacità. Si potrebbe leggere preliminarmente il numero di stringhe da inserire, prima di creare l'oggetto array, in modo da dimensionarlo opportunamente. Tuttavia questo richiede all'utente di sapere o contare a priori le stringhe da inserire, soluzione del tutto insoddisfacente. Una soluzione migliore è proposta nell'esercizio successivo.

Esercizio 8.6
Fate riferimento all'esempio precedente. Quando l'array risulta pieno, è possibile simularne l'ampliamento come segue.

- Si crea un nuovo oggetto array (raggiungibile tramite un riferimento ausiliario) di capacità maggiore, ad esempio doppia.

- Si copiano tutti gli elementi presenti nell'array riferito da `tab` nelle corrispondenti posizioni del nuovo array.
- Si copia in `tab` il riferimento al nuovo array.

A questo punto si può proseguire l'esecuzione leggendo una nuova stringa.

Queste operazioni possono essere codificate all'interno del cicli `while` in modo che siano effettuate ogni volta che si raggiunge la capacità dell'array che si sta utilizzando, cioè ogni volta che `pos` risulti uguale a `tab.length`. Effettuate queste modifiche sul codice precedente.
Nota. Classi come `Vector` *o* `ArrayList` *delle librerie standard di Java utilizza proprio un meccanismo di questo tipo per simulare array la cui capacità può variare dinamicamente.*