

Note ed esercizi aggiuntivi

6. Ancora uso dei cicli, tipi generici

Esercizio 6.1

Scrivete un'applicazione che legga un numero intero n e comunichi il risultato della somma di frazioni $1/1 + 1/2 + \dots + 1/n$. Nel caso venga inserito un numero negativo, l'applicazione deve chiedere all'utente di ripetere l'inserimento.

Esercizio 6.2

Scrivete un'applicazione che legga un numero n e produca un elenco di tutte moltiplicazioni dei numeri da 1 a n . Si consiglia di utilizzare due cicli for innestati. Se ad esempio viene fornito il valore 4, l'output prodotto dovrà essere:

```
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
4 * 1 = 4
4 * 2 = 8
4 * 3 = 12
4 * 4 = 16
```

Esercizio 6.3

Ripetete l'esercizio precedente, producendo una tabella delle moltiplicazioni. In una prima versione potete produrre una tabella come la seguente, senza preoccuparvi degli allineamenti:

```
1 2 3 4
1 1 2 3 4
2 2 4 6 8
3 3 6 9 12
4 4 8 12 16
```

Migliorate poi la visualizzazione, facendo in modo che i numeri di ciascuna colonna risultino allineati a destra, sulla base dello spazio richiesto dal numero più grande (che sarà quello sull'ultima riga):

```
1 2 3 4
1 1 2 3 4
2 2 4 6 8
3 3 6 9 12
4 4 8 12 16
```

Per ottenere l'allineamento possono essere utili le indicazioni date per l'esercizio 5.4. In alternativa, la classe `ConsoleOutputManager` e la classe `PrintStream` delle librerie standard (a cui appartiene l'oggetto che rappresenta il monitor riferito da `System.out`) offrono un metodo `printf` che permette una visualizzazione "formattata" secondo lo stile della funzione `printf` del linguaggio C. Consultate la documentazione per conoscere i dettagli.

Esempio.

/ Esercizio:*

Scrivete un'applicazione che legga un elenco di importi e ne visualizzi la media.

L'applicazione deve poi visualizzare:

- un elenco di tutti gli importi che superano la media
- un elenco di tutti gli importi che non superano la media

Per semplicità l'elenco non può essere vuoto.

**/*

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

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

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

        //legge l'elenco, ne calcola la somma, e memorizza gli importi
        Importo somma = new Importo(0);

        do {
            out.println("Inserire un importo:");
            int euro = in.readInt(" - euro? ");
            int cent = in.readInt(" - cent? ");
            Importo imp = new Importo(euro, cent);

            somma = somma.piu(imp); // aggiunge l'importo letto alla somma

            seq.add(imp); // memorizza l'importo letto nella sequenza
        } while (in.readSiNo("Ci sono altri importi (s/n)? "));

        Importo media = somma.diviso(seq.size());

        out.println("Media degli importi " + media);
    }
}
```

```
        out.println("Importi superiori alla media: ");
        for (Importo i: seq)
            if (i.isMaggiore(media))
                out.println(i);
        out.println();

        out.println("Importi inferiori alla media: ");
        for (Importo i: seq)
            if (i.isMinore(media))
                out.println(i);
        out.println();

    }
}
```

Esercizio 6.4

Fate riferimento all'esempio precedente.

- Riscrivete il codice precedente in modo che sia possibile inserire anche un elenco vuoto.
- Dopo avere visualizzato i due elenchi richiesti ai punti a e b, visualizzate gli stessi elenchi, ma in ordine crescente (la classe `SequenzaOrdinata` fornisce sia un costruttore, sia un metodo, utili a questo scopo; potete usare uno dei due a vostra scelta).

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 sequenza
        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 6.5

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 6.6

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 6.7

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 6.8

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 6.9

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