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

Note ed esercizi aggiuntivi

Gli esercizi proposti sono utili per rivedere gli esempi riportati, alcuni dei quali sviluppati e discussi in dettaglio a lezione.

4. Cicli `do-while` e `while`, metodi statici

Esempio. Lettura di una sequenza di importi e calcolo della somma.

```
import prog.io.*;
import prog.utili.Importo;

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

    Importo somma = new Importo(0);

    //lettura e calcolo
    do {
      //leggi un importo
      out.println("Inserire il prezzo:");
      int euro = in.readInt(" --- euro? ");
      int cent = in.readInt(" --- centensimi? ");
      Importo prezzo = new Importo(euro, cent);

      //aggiungilo alla somma
      somma = somma.piu(prezzo);

    } while (in.readSiNo("Vuoi inserire altri prezzi? (s/n) "));

    //comunicazione risultato
    out.println("La somma dei prezzi inseriti e' " + somma);

  }
}
```

Note

- Una dichiarazione di variabile è un'informazione *destinata al compilatore* per stabilire il significato degli identificatori, utile per le questioni relative ai tipi, e per decidere come

sarà organizzata durante l'esecuzione la memoria del metodo, in questo caso `main`. Poiché le variabili `euro`, `cent` e `prezzo` sono utili solo all'interno del corpo del ciclo, la loro dichiarazione è collocata in quel punto. Ciò *non significa assolutamente* che ad ogni iterazione vi siano nuove variabili `euro`, `cent` e `prezzo`: esiste una sola variabile `euro`, una sola variabile `cent` e una sola variabile `prezzo`, come stabilito dal compilatore analizzando il testo del programma, *indipendentemente* dal numero di iterazioni del ciclo durante l'esecuzione.

- La costruzione degli oggetti avviene invece in esecuzione. Pertanto viene creato un *nuovo* oggetto di tipo `Importo` ad ogni esecuzione dell'istruzione `new` all'interno del ciclo. Il riferimento all'oggetto è assegnato alla variabile `prezzo`. Il riferimento precedentemente contenuto in `prezzo` viene dunque perso. Analogamente, ad ogni invocazione del metodo `piu` all'interno del ciclo, viene costruito un nuovo oggetto di tipo `Importo`, il cui riferimento viene memorizzato nella variabile `somma`, al posto del riferimento alla somma precedente.

Esercizio 4.1
Fate riferimento all'esempio precedente.

- Modificate il codice in modo che alla fine comunichi la somma anche in lettere, nel formato utilizzato per gli assegni (ad esempio `12,05` va indicato come `dodici/05`).
- Con il codice organizzato in questo modo, viene letto sempre almeno un prezzo. Modificate il codice in modo che sia possibile terminare l'esecuzione senza avere inserito alcun prezzo.
- Modificate il codice in modo che ad ogni iterazione non venga chiesto all'utente se desidera inserire altri importi. L'utente segnalerà la fine della sequenza di prezzi inserendo `0` come valore di euro e centensimi.

Esempio. Lettura di una sequenza di interi e calcolo della somma.

```
import prog.io.*;

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

    int somma = 0;

    //lettura e calcolo
    int numero = in.readInt("Inserisci un numero ");

    while (numero != 0) {
      somma = somma + numero;
      numero = in.readInt("Inserisci un numero ");
    }

    //comunicazione risultato
    out.println("La somma vale " +  somma);

  }
}
```

Esercizio 4.2
Scrivete un'applicazione che calcoli il prodotto di una sequenza di numeri interi letta da tastiera. Supponete che l'inserimento di `0` indichi la fine della sequenza.

Esercizio 4.3
Scrivete un'applicazione che calcoli la media di una sequenza di numeri interi. Per la divisione si utilizza l'operatore `/`, su cui vi è overloading (divisione intera, divisione con la virgola). Quando fate eseguire l'applicazione riuscite a ottenere i decimali? Capiremo ciò che succede approfondendo lo studio dei tipi primitivi e delle conversioni implicite ed esplicite tra essi.

Esempio. Lettura di una sequenza di prezzi e calcolo del minimo.

```
import prog.io.*;
import prog.utili.Importo;

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

    Importo minimo = null;

    //lettura e calcolo
    do {
      //leggi un importo
      out.println("Inserire il prezzo:");
      int euro = in.readInt(" --- euro? ");
      int cent = in.readInt(" --- centensimi? ");
      Importo prezzo = new Importo(euro, cent);

      //calcola il nuovo minimo
      if (minimo == null || prezzo.isMinore(minimo))
        minimo = prezzo;

    } while (in.readSiNo("Vuoi inserire altri prezzi? (s/n) "));

    //comunicazione risultato
    out.println("Il piu' piccolo prezzo inserito e' " + minimo);

  }
}
```

Note

- Il metodo `isMinore` della classe `Importo`, se riceve un riferimento `null`, non è in grado di svolgere il proprio compito (che senso avrebbe confrontare un importo con un importo che non c'è?). Scrivete del codice di prova per vedere cosa succede se si fornisce un riferimento `null` come argomento a `isMinore`. Si possono fare le stesse considerazioni se al posto di `isMinore` si utilizza `compareTo`.
- Attenzione alla condizione della selezione, che si basa sulla *lazy evaluation* dell'operatore `||`. Cosa succede se si scambiano i due operandi di `||`?

Esercizio 4.4
Fate riferimento all'esempio precedente.

- Modificate il codice in modo che oltre al prezzo minimo determini anche quello massimo e la differenza tra i due.
- Modificate il codice precedente in modo che determini il più piccolo tra tutti i prezzi letti superiori a 100 euro. Potete utilizzare lo schema dell'esempio, aggiungendo un'ulteriore condizione per la selezione (attenzione all'uso degli operatori di tipo `boolean`!) e servendovi di un riferimento a un oggetto costruito inizialmente che rappresenti l'importo di 100 euro.

  Prestate attenzione al fatto che l'utente potrebbe inserire esclusivamente prezzi che non superano 100 euro. Cosa succede in tal caso al termine dell'esecuzione del ciclo? Cosa conterrà la variabile `minimo`? Fate in modo che in questo caso il programma, anziché comunicare il prezzo minore, comunichi che non sono stati inseriti prezzi superiori a 100 euro. *(È del tutto inutile introdurre ulteriori variabili per controllare questa situazione.)*
- Modificate la soluzione per il punto precedente, sostituendo un prezzo inserito inizialmente dall'utente al posto della soglia fissata di 100 euro.
- Scrivete un metodo `main` che legga una sequenza di prezzi e determini quello *più vicino* a 100 euro. Se ad esempio i prezzi letti sono `98,05`, `102,00`, `103,01`, il risultato sarà `98,05`, se invece sono `98,05`, `101,00`, `103,01`, il risultato sarà `101,00`. Nel caso vi siano due prezzi più vicini, come ad esempio nella sequenza `98,00`, `102,00`, `103,01`, il risultato potrà essere uno qualunque dei due.
- Modificate la soluzione per il punto precedente, sostituendo un prezzo inserito inizialmente dall'utente al posto del prezzo fissato di 100 euro.

Esempio. Lettura di una sequenza di stringhe e calcolo della stringa lessicograficamente minima. L'inserimento da parte dell'utente della stringa vuota indica la fine della sequenza (la stringa vuota non fa parte della sequenza stessa).

```
import prog.io.*;

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

    String minima = null;

    //lettura delle stringhe e calcolo della minima
    String s = in.readLine("Prossima stringa? ");
    while (!s.equals("")) {
      if (minima == null || s.compareTo(minima) < 0)
        minima = s;

      s = in.readLine("Prossima stringa? ");
    }

    //comunicazione risultato
    if (minima == null)
      out.println("Non e' stata inserita alcuna stringa!");
    else
```

```
      out.println("La stringa minima in ordine lessicografico e' " + minima);

  }
}
```

Note

- Osservate l'uso del ciclo con il controllo in testa. Se l'utente inserisce immediatamente la stringa vuota, il corpo del ciclo non viene eseguito (la sequenza esaminata è vuota).
- Fate attenzione alla condizione del ciclo: cosa succede se viene sostituita dalla condizione `s != ""`? Perché? *È fondamentale comprendere chiaramente la differenza tra le due condizioni.*

Esercizio 4.5
Ispirandovi all'esempio precedente, scrivete un'applicazione che legga una sequenza di stringhe e comunichi:

- il numero di stringhe lette;
- la somma e la media delle lunghezze delle stringhe lette;
- la stringa lessicograficamente minima;
- la stringa lessicograficamente massima.

L'inserimento della stringa vuota (che non fa parte della sequenza) indica la fine della sequenza stessa.

Esempio. Minimo di una sequenza di interi inserita dall'utente. L'inserimento di 0 indica la fine della sequenza (0 non fa parte della sequenza stessa).

```
import prog.io.*;

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

    boolean primoNumero = true;
    int minimo = 123456789;  //un valore qualunque per far tacere il compilatore

    //lettura e calcolo
    int numero = in.readInt("Inserisci un numero ");

    while (numero != 0) {
      if (primoNumero || numero < minimo) {
        minimo = numero;
        primoNumero = false;
      }
      numero = in.readInt("Inserisci un numero ");
    }

    //comunicazione risultato
```

```
    if (primoNumero)
      out.println("Non e' stato inserito alcun numero");
    else
      out.println("Il piu' piccolo numero inserito e' " + minimo);

  }
}
```

Note

- Poiché la variabile minimo è del *tipo primitivo* `int`, non è possibile assegnarle, come nel caso della sequenze di prezzi, il riferimento `null` per indicare che non è ancora stato letto nulla. A tale scopo viene utilizzata invece la variabile `primoNumero` di tipo `boolean`, che vale `true` se deve ancora essere letto o elaborato il primo numero della sequenza.
- L'inizializzazione `minimo = 123456789` (o una inizializzazione della variabile `minimo` a un *qualunque valore*) è necessaria per ragioni tecniche. Il compilatore effettua un'analisi del codice per individuare situazioni in cui si utilizzano valori contenuti in variabili che *potrebbero* non essere state inizializzate. Senza tale inizializzazione, il compilatore segnalerà che nel confronto `numero < minimo` e nell'istruzione di scrittura finale si sta utilizzando il valore di una variabile che *potrebbe* non essere stata inizializzata in precedenza (provate a togliere l'inizializzazione per vedere i messaggi prodotti). In realtà, conoscendo la logica con cui è stato concepito il codice, siamo certi che ciò non può avvenire e che il valore `123456789` indicato non verrà mai utilizzato: ogni volta che si usa la variabile `minimo` essa contiene un valore letto e assegnato in precedenza. Dunque l'inzializzazione introdotta ha come unico scopo quello di evitare questo tipo di messaggi dal compilatore (si ricordi che il compilatore non può conoscere la logica con cui sono scritti i programmi).
  EVITATE di introdurre inizializzazioni inutili nel codice, che non fanno altro che complicare la lettura. Se siete costretti a farlo, come in questo esempio, inserite sempre un breve commento.

Esercizio 4.6
Fate riferimento all'esempio precedente.

- Modificate l'applicazione in modo che determini anche il più grande numero inserito.
- Modificate l'applicazione in modo che determini il minimo numero pari inserito. Attenzione al significato della variabile di tipo `boolean`, che in questo caso dovrà ricordare se deve essere letto e elaborato il primo numero pari. È opportuno cambiare il nome di tale variabile, ad esempio in `primoPari`.
- Modificate l'applicazione ottenuta al punto precedente in modo che determini, oltre al minimo numero pari inserito, anche il minimo numero dispari.

Esercizio 4.7
Scrivete un'applicazione che legga una sequenza di numeri e comunichi:

- il più piccolo numero pari,
- il più piccolo numero dispari,
- il più grande numero pari,
- il più grande numero dispari,
- la media di tutti i numeri inseriti.

Esempio. Lettura di una sequenza di prezzi in lire e calcolo della somma dei prezzi equivalenti in euro. L'inserimento di `0` indica la fine della sequenza.

```
import prog.io.*;
import prog.utili.Importo;

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

    Importo sommaEuro = new Importo(0);

    //lettura ed elaborazione della sequenza
    int prezzoLire = in.readInt("Inserire un prezzo in lire (0 per terminare) ");
    while (prezzoLire != 0) {
      //calcola prezzo in euro
      Importo prezzoEuro = Importo.fromLire(prezzoLire);
      out.println("Euro : " + prezzoEuro);

      sommaEuro = sommaEuro.piu(prezzoEuro);

      prezzoLire = in.readInt("Inserire un prezzo in lire (0 per terminare) ");
    }

    //comunicazione risultato
    out.println("Somma prezzi in Euro = " + sommaEuro);
  }
}
```

Note
Prestate attenzione all'invocazione del metodo `fromLire`. *Tale metodo è statico.* Ciò significa che esso è un servizio offerto *direttamente* dalla classe `Importo`. Pertanto viene richiamato rivolgendosi direttamente alla classe, con l'invocazione

```
Importo.fromLire(...)
```

Sebbene il compilatore permetta di richiamare un metodo statico utilizzando il riferimento a un oggetto della classe, al posto del nome della classe stessa, tale scrittura *deve essere assolutamente evitata* in quanto l'oggetto non è in alcun modo coinvolto nell'esecuzione del metodo che, invece, è un compito della classe.

Esercizio 4.8
Modificate il codice dell'esempio precedente in modo che comunichi anche la somma dei prezzi in lire inseriti.

Esercizio 4.9
Scrivete un'applicazione che legga una sequenza di prezzi espressi in lire. Per ogni prezzo l'applicazione deve comunicare il corrispondente in euro. Supponete che l'inserimento di `0` indichi la fine della sequenza. L'applicazione deve poi comunicare:

- la somma dei prezzi in lire inseriti,
- la somma dei prezzi corrispondenti in euro e il suo corrispondente in lire.

A causa delle regole di arrotondamento nella trasformazione da lire in euro, le somme in lire ottenute ai due punti precedenti potrebbero differire. L'applicazione deve indicare se globalmente dall'arrotondamento si è ottenuto un guadagno o una perdita, fornendone l'ammontare in lire. Seguono due esempi di come dovrà essere l'esecuzione:

```
Inserire un prezzo (0 per terminare) 1500
Euro : 0,77
Inserire un prezzo (0 per terminare) 1500
Euro : 0,77
Inserire un prezzo (0 per terminare) 0
Somma prezzi in Lire = 3000
Somma prezzi in Euro = 1,54 - Corrispondente a Lire 2982
Perdita: Lire 18
```

```
Inserire un prezzo (0 per terminare) 2000
Euro : 1,03
Inserire un prezzo (0 per terminare) 4000
Euro : 2,07
Inserire un prezzo (0 per terminare) 0
Somma prezzi in Lire = 6000
Somma prezzi in Euro = 3,10 - Corrispondente a Lire 6002
Guadagno: Lire 2
```

Esercizio 4.10
Modificate l'applicazione costruita per l'esercizio 4.9 in modo che venga visualizzato anche il valore in euro corrispondente al guadagno o alla perdita in lire. Introducete poi ulteriori modifiche in modo che tutti i valori visualizzati vengano forniti anche in lettere, oltre che in cifre.

Esercizio 4.11
Scrivete un'applicazione che legga due numeri interi, ne calcoli la somma e la visualizzi incolonnata, nel formato presentato nei seguenti esempi di esecuzione:[1]

```
Primo addendo? 54
Secondo addendo? 60
 54 +
 60 =
-----
114
```

```
Primo addendo? 1234567890
Secondo addendo? -987654321
1234567890 +
-987654321 =
------------
 246913569
```

Per la visualizzazione utilizzate gli usuali metodi `print` e `println`. L'incolonnamento può essere ottenuto calcolando il numero di cifre da cui è composto ciascun numero (cioè la lunghezza della stringa che rappresenta il numero), e facendo precedere il numero, se necessario, da spazi. La classe `String` offre un metodo *statico* di nome `valueOf` che riceve come argomento un `int` e restituisce il riferimento a un oggetto contenente la stringa corrispondente. Si può tenere conto del fatto che, in valore assoluto, il più grande intero rappresentabile è compreso tra due miliardi e tre miliardi.

[1] Rivedete la soluzione presentata nella lezione 10, in cui si ipotizzava che i due numeri letti non fossero negativi. Svolgete l'esercizio rimuovendo tale ipotesi.

Esercizio 4.12
Modificate l'applicazione scritta per l'esercizio 4.11, in modo che le cifre vengano separate tre a tre da un punto, per evidenziare migliaia, milioni, ecc, come nel seguente esempio:

```
Primo addendo? 1234567890
Secondo addendo? -987654321
1.234.567.890 +
 -987.654.321 =
---------------
  246.913.569
```

Esercizio 4.13
Riscrivete l'applicazione dell'esercizio 4.11 in modo che sia in grado di trattare interi di grandezza arbitraria. A tale scopo, al posto del tipo primitivo `int`, rappresentate gli interi come oggetti della classe `java.math.BigInteger`, i cui oggetti rappresentano numeri interi di grandezza arbitraria. Consultate la documentazione per i dettagli.