Esercizi per il Corso di Metodologie di Programmazione

Capitolo 5

Università degli Studi di Roma "La Sapienza"

Esercizio 1

Dato un numero intero inserito in input dall'utente, stampare a video il suo segno, effettuando un controllo apposito nel caso in cui esso valga zero.

Esercizio 2

Dato un numero in virgola mobile a doppia precisione (*double*) inserito in input dall'utente, effettuare due tipologie di controlli:

1. Se il suo valore assoluto è maggiore di 1, il numero sarà categorizzato come *small*. Se invece il valore assoluto è maggiore di 1000000, esso sarà categorizzato come *large*
2. Effettuare i controlli dell'esercizio precedente, stampando a video il suo segno. Nel caso di segno positivo o negativo, la stampa conterrà anche l'informazione precedente sulla grandezza del numero.

Consiglio: Il calcolo del valore assoluto è fornito dal metodo *abs* all'interno della classe *Math*.
Esempio:

- "It's a *small* positive number."
- "It's a *large* negative number."

Esercizio 3

Dato un intero *long* letto da input, controllare se esso sia negativo e, nel caso, convertirlo al segno opposto. Successivamente, tramite una serie di controlli, stampare il numero di cifre in esso contenute, senza conversioni di alcun tipo.

Esercizio 4

Data una stringa letta da input, controllare l'uguaglianza tra la prima e l'ultima lettera e stampare a video l'opportuno messaggio, gestendo tutti i possibili esiti di questo controllo.

Esercizio 5

Data una stringa letta da input, controllare se la sua prima e seconda metà, al netto del carattere centrale, siano uguali, stampando a video l'opportuno messaggio.
Consiglio: per trovare la metà di una parola si può utilizzare il metodo *length* della classe *String*.
Esempio:
Enter a word (ENTER to terminate): ***pianopforte***
The first and last half of your word differ
first = piano last = forte

Enter a word (ENTER to terminate): ***pianoppiano***
The first and last half of your word are the same
first = piano last = piano

Esercizio 6

Dati tre numeri a virgola mobile con precisione doppia letti da input, verificare:

- il caso in cui tutti e tre siano uguali
- il caso in cui tutti e tre siano diversi
- la non appartenenza ai due casi precedenti

Per ogni casistica, stampare l'opportuno messaggio a video.

Esercizio 7

Data una sequenza di tre numeri a virgola mobile con precisione doppia letti da input, verificare:

- il caso in cui la sequenza fornita sia crescente
- il caso in cui la sequenza fornita sia decrescente
- la non appartenenza ai due casi precedenti

Per ogni casistica, stampare l'opportuno messaggio a video.

Esercizio 8

Estendere l'esercizio precedente dando la possibilità di decidere se la sequenza debba essere crescente o decrescente in senso stretto o lato. Lo schema di realizzazione dell'esercizio è il seguente:

1. L'utente sceglie se applicare un controllo di tipo stretto inserendo la lettera *S* o *s* o di tipo lato tramite *L* o *l*.
2. In base all'opzione scelta, applicare i controlli dell'esercizio precedente adeguandoli opportunamente e stampando a video gli esiti

Esempio:
Enter S for strict ordering, L for lenient (S or L):
S
Enter three numbers:
3 4 5
increasing

Enter S for strict ordering, L for lenient (S or L):
l
Enter three numbers:
4 3 3
decreasing

Esercizio 9

Dati tre numeri interi letti da input, verificare se essi siano in ordine (crescente o decrescente) o meno e stampare a video l'opportuno messaggio.

Esercizio 10

Dati quattro numeri interi letti da input, verificare se essi siano uguali a coppie, stampando l'opportuno messaggio.

Esercizio 11

Dato un numero letto da input rappresentante il valore corrente di una bussola, in gradi, dire se essa stia puntando a *Nord-Est, Est, Sud-Est, Sud, Sud-Ovest, Ovest, Nord-Ovest, Nord.*
Consiglio: ai fini del risultato è necessario definire delle costanti che associno un punto cardinale al suo corrispettivo valore in gradi. Assumendo ovviamente che i gradi aumentino in senso orario:

- Il Nord comincia a 0 gradi
- Il Nord-Est si trova tra gli 1 ed i 22.5 gradi
- L'Est si trova tra i 22.6 ed i 67.5 gradi
- Il Sud-Ovest si trova tra i 67.6 ed i 90 gradi

E così via, aumentando di 45 gradi alla volta.
Esempio:
Please the direction the compass is pointing (in degrees from North (0..360): ***90***
90.0 Degrees is East

Please the direction the compass is pointing (in degrees from North (0..360): ***67.4***
67.4 Degrees is North East

Esercizio 12

Creare una classe *SimpleTime* che rappresenti l'orario, strutturata nel seguente modo:
Attributi, con visibilità *public*:

- *hour*: di tipo intero, rappresenta l'ora
- *minute*: di tipo intero, rappresenta i minuti
- *time*: di tipo *String*, rappresenta l'orario in formato testuale

Costruttore: accetta due parametri in ingresso che rappresentano l'ora e i minuti. Esso:

- Assegnerà agli attributi il valore dei corrispettivi parametri
- Valorizzerà *time* come *hour:minute*

Metodi:

- *compareTo*: accetta come parametro un oggetto di tipo *SimpleTime*, confrontando i due orari per decretare chi tra i due precede l'altro. Il metodo ritornerà −1 se l'orario del chiamante precede l'orario contenuto nell'oggetto parametro, 0 se sono uguali, 1 altrimenti.

La classe *SimpleTimeDemo* istanzierà due oggetti di tipo *SimpleTime* valorizzando ora e minuti tramite l'utente. Successivamente, uno dei due oggetti chiamerà il metodo *compareTo* utilizzando l'altro come parametro. In base al valore ritornato dalla funzione, il programma stamperà a video un opportuno messaggio decretando l'ordine cronologico dei due orari.
Esempio:
To compare two times in military format (24 hr), first enter the hour of the first time: 15
Enter the minutes of the first time: 00
Now enter the hour of the second time: 16
Enter the minutes of the second time: 35
15:00 comes first.

Esercizio 13

Creare la classe *Grade*, la quale rappresenta e gestisce un voto scolastico in lettere.
Attributi, con visibilità *private*:

- *letterGrade*: di tipo *String*, rappresenta la votazione in lettere, che può essere *A, B, C, D, F* con eventuali modificatori + e -

Costruttore: accetta un unico parametro che rappresenta il voto in lettere e lo assegna al relativo attributo.
Metodi:

- *getLetterGrade*: ritorna il valore dell'attributo *letterGrade*
- *getNumericGrade*: converte e restituisce il corrispettivo numerico del voto. La funzione ritornerà −1 per indicare un errore nella conversione. Nel caso invece in cui tutto sia corretto, la votazione numerica avverrà secondo il seguente schema:
  - I modificatori + e - valgono rispettivamente 0.3 e −0.3

  - *A* vale 4
  - *B* vale 3
  - *C* vale 2
  - *D* vale 1
  - *F* vale 0. Se questo voto era seguito da dei modificatori (ovvero *F+* o *F-*), la funzione ritornerà −1
  - Qualsiasi altra lettera porterà ad un valore di ritorno pari a −1
  - Il voto numerico, al netto dell'aggiunta di un eventuale modificatore, non potrà essere maggiore di 4

Creare una classe *GradeTester* che verifichi il corretto funzionamento del programma.
Consiglio: l'estrazione della lettera e del modificatore può essere realizzata tramite i metodi *charAt* o *substring*, appartenenti alla classe *String*.
Esempio:
A: 4.0
Expected: 4
A+: 4.0
Expected: 4
A-: 3.7
Expected: 3.7
C: 2.0
Expected: 2
C+: 2.3
Expected: 2.3
C-: 1.7
Expected: 1.7
F: 0.0
Expected: 0

Esercizio 14

Sulla traccia dell'esercizio precedente, creare una classe *Grade* che rappresenti e gestisca un voto scolastico, questa volta di tipo numerico.
Attributi, con visibilità *private*:

- *numericGrade*: di tipo intero, rappresenta un voto tra 0 e 4, decimali inclusi

Costruttore: accetta un unico parametro, il voto numerico, ed assegna al corrispettivo attributo il suo valore.
Metodi:

- *getLetterGrade* converte e ritorna al chiamante il voto letterale, ottenuto da quello numerico secondo il seguente schema:
  - *A*: da 3.5 in su

- *B*: da 2.5 in su
- *C*: da 1.5 in su
- *D*: da 0.5 in su
- *F*: tra 0 e 0.4

La gestione dei decimali decreterà la scelta del modificatore introdotta nell'esercizio precedente. Infatti:

- Il segno *+* sarà associato a tutti quei voti il cui decimale si trova all'interno dell'intervallo $[0.15, 0.5)$
- Il segno sarà associato a tutti quei voti il cui decimale si trova all'interno dell'intervallo $[0.5, 0.85)$

Verificare il corretto funzionamento di *Grade* tramite la creazione di una classe chiamata *GradeTester*.

Esempio:
4.0: A
Expected: A
4.2: A+
Expected: A+
2.85: B
Expected: B
2.84: B-
Expected: B-
0.1: F
Expected: F
0.49: F
Expected: F
0.51: D-
Expected: D-

Esercizio 15

Creare la classe *TaxReturn* che gestisca il calcolo delle tasse di un singolo cittadino in base allo scaglione di reddito generato.
Attributi, con visibilità *private*:

- *income*: di tipo *double*, rappresenta il reddito del cittadino
- Tutte le costanti relative allo scaglione e alla relativa percentuale di tassazione da applicare (vedi sotto)

Costruttore: accetta un parametro *double* che rappresenta il reddito, il cui valore verrà assegnato al corrispondente attributo.
Metodi:

- *getTax*: applica la tassazione sul reddito in base allo scaglione di cui fa parte, ritornando al chiamante l'ammontare di tasse da pagare. Le fasce sono definite nel seguente modo:
  - *Prima fascia*: redditi superiori a 500000, con una tassazione del 6%
  - *Seconda fascia*: redditi superiori a 250000, con una tassazione del 5%
  - *Terza fascia*: redditi superiori a 100000, con una tassazione del 4%
  - *Quarta fascia*: redditi superiori a 75000, con una tassazione del 3%
  - *Quinta fascia*: redditi superiori a 50000, con una tassazione del 2%
  - *Sesta fascia*: ogni altro reddito inferiore alla quinta fascia, con una tassazione del 1%

Creare una classe chiamata *TaxCalculatorTester* che verifichi il corretto funzionamento della classe *TaxReturn*.

Esempio:
Income of 600000.0, Tax: 26250.0
Expected: 26250.0
Income of 1000.0, Tax: 10.0
Expected: 10.0
Income of 117000.0, Tax: 2430.0
Expected: 2430.0
Income of 60000.0, Tax: 700.0
Expected: 700.0
Income of 85000.0, Tax: 1300.0
Expected: 1300.0
Income of 110000.0, Tax: 2150.0
Expected: 2150.0
Income of 260000.0, Tax: 8250.0
Expected: 8250.0
Income of 510000.0, Tax: 20850.0
Expected: 20850.0

Esercizio 16

Creare la classe *Card* che gestisca la rappresentazione di una carta appartenente ad un mazzo di carte da poker.
Attributi, *private*:

- *type*: di tipo *String*, rappresenta il valore della carta
- *suit*: di tipo *String*, rappresenta il seme della carta

A seguire, viene fornita la notazione dei possibili valori di una carta:

- Da 1 a 10: corrispettivo numerico
- *J*: Jack
- *Q*: Regina

- *K*: Re
- *A*: Asso

E dei semi possibili:

- *S*: Spade
- *H*: Cuori
- *D*: Diamanti
- *C*: picche

Costruttore: accetta un parametro che contiene la notazione della carta (es: *10S*). Il costruttore dovrà determinare il valore ed il seme: il primo comprenderà tutti i caratteri della stringa eccetto l'ultimo, che diventerà invece il valore del seme. Se la lunghezza della stringa passata sarà minore di 2, valore e seme saranno valorizzati con un *?*.

Metodi:

- *getDescription*: ritorna la descrizione letterale della carta. Per quanto riguarda i valori, i numeri saranno convertiti nella loro forma letterale, mentre le figure seguiranno la descrizione fornita in precedenza. Lo stesso varrà per i semi. In caso di valore o seme non valido, il metodo ritornerà la parola *Unknown*.

Esempio:
4S: Four of Spades
Expected: Four of Spades
CQ: Queen of Clubs
Expected: Queen of Clubs
AD: Ace of Diamonds
Expected: Ace of Diamonds
10S:Ten of Spades
Expected: Ten of Spades
CA: Unknown
Expected: Unknown
: Unknown
Expected: Unknown
?: Unknown
Expected: Unknown

Esercizio 17

Dati tre numeri inseriti da input da un utente, stampare il massimo.

Esercizio 18

Creare la classe *StringSet* la quale, date tre stringhe, le ordini alfanumericamente.
Attributi, *private*:

- *smallest*: di tipo *String*, rappresenta la stringa più piccola
- *largest*: di tipo *String*, rappresenta la stringa più grande
- *middle*: di tipo *String*, rappresenta la stringa rimanente

Costruttore: accetta come parametri tre stringhe e ne decreta l'ordinamento, assegnandole al corrispettivo parametro.
Metodi:

- *getSmallest*: ritorna la stringa più piccola tra le tre
- *getLargest*: ritorna la stringa più grande tra le tre
- *getMiddle*: ritorna la stringa rimanente

Creare una classe chiamata *StringSetTester* la quale, date tre parole lette da input, stampi in sequenza la parola più piccola, quella media ed infine la più grande.
Esempio:
StringSet->"Tom", "Doe", "Harry"
Doe
Expected: Doe
Harry
Expected: Harry
Tom
Expected: Tom

StringSet->"Harry", "Doe", "Tom"
Doe
Expected: Doe
Harry
Expected: Harry
Tom
Expected: Tom
Consiglio: la classe *String* offre il metodo *compareTo* per effettuare il confronto alfanumerico tra due stringhe.

Esercizio 19

Creare la classe *Comparer* la quale, dati due numeri a virgola mobile, controlla se la loro differenza è significativa rispetto ad una tolleranza fornita da input, pari a $10^{-1+digits}$
Attributi, di tipo *private*:

- *digits*, di tipo intero, indica il numero di cifre decimali utili al calcolo del valore di soglia

Costruttore: riceve come parametro un intero che indica indica il numero di cifre decimali della soglia, il cui valore verrà assegnato al parametro *digits*.

Metodi:

- *areSame*: riceve come parametri due numeri a virgola mobile per i quali controllerà che la loro differenza sia al di sotto della soglia calcolata tramite *digits*. Se la differenza non è

significativa, la funzione ritornerà *True*, indicando che i due valori sono uguali. In caso contrario, verrà ritornato *False*.

Creare una classe chiamata *ComparerTester* che verifichi la corretta implementazione della classe precedente.

Esempio:
Enter two floating-point numbers: ***2.473 2.476***
They are the same up to two decimal places.

Enter two floating-point numbers: ***2.46 2.41***
They are different.

Enter two floating-point numbers: ***2.46 2.45***
They are the same up to two decimal places.

Esercizio 20

Creare un programma che legga una parola letta da input. Se la sua lunghezza è pari a uno, controllare se il carattere inserito appartenga o meno alle vocali o alle consonanti, stampando a video un opportuno messaggio.