Prova di Laboratorio di Programmazione — Edizione 2

18 luglio 2014

ATTENZIONE: Non è possibile usare le classi del package `prog.io` del libro di testo

Lo scopo è realizzare un programma che calcoli la lunghezza di percorsi che attraversano delle città indicate dal nome e dalle loro coordinate (x, y).

Oltre ai metodi richiesti in ciascuna classe, è opportuno implementare altri metodi che si ritengono utili per svolgere l'esercizio nel modo più corretto ed elegante, per esempio il metodo `toString` e i metodi di accesso ai campi.

Il codice esterno ad una classe non deve mai accedere direttamente ai campi della classe.

Le operazioni di stampa vanno effettuate esclusivamente nelle classi che definiscono il metodo `main`.

Esercizio 1

Si implementino le seguenti classi.

Classe `City`.
Classe che descrive una città caratterizzata da: nome, coordinata x, coordinata y. Si assuma che le coordinate siano interi compresi fra 0 e 100 e che città caratterizzate da coordinate diverse debbano avere nomi diversi.

Classe `Tour`.
Classe le cui istanze rappresentano un percorso, ovvero una sequenza di città. Un percorso è valido se e solo se ogni città compare nella sequenza una sola volta.

La classe `Tour` definisce il metodo

- `double lunghezza()`
  Il metodo restituisce la lunghezza di un percorso valido, calcolata come la distanza totale percorsa per attraversare le città nell'ordine indicato dalla sequenza e quindi ritornare alla città di partenza. Per calcolare la distanza fra la città A di coordinate (x_a, y_a) e la città B di coordinate (x_b, y_b) si usi la formula della distanza Euclidea

  $$\|(x_a, y_a) - (x_b, y_b)\| = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$$

  Quindi, la lunghezza del percorso valido A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ A, rappresentato dalla sequenza di città (A, B, C) è calcolata come

  $$\|(x_a, y_a) - (x_b, y_b)\| + \|(x_b, y_b) - (x_c, y_c)\| + \|(x_c, y_c) - (x_a, y_a)\|$$

Per il calcolo della radice quadrata potete usare il metodo

```
public static double Math.sqrt(double a)
```

Scrivere un programma `Esercizio1` che legge da **standard input** una sequenza di linee con i seguenti formati:

```
A,nome,coordX,coordY
C
```

Le linee che iniziano con `A` richiedono la creazione e memorizzazione di una città con nome e coordinate specificate. Le città vengono aggiunte ad un percorso nella stessa sequenza in cui vengono create. Le linee che iniziano con `C` richiedono di calcolare e stampare il percorso corrente e la sua lunghezza nel caso in cui il percorso sia valido.

Non è necessario fare controlli sulla correttezza dell'input. Non vanno fatte assunzioni sul numero di linee in input. L'input va letto da **standard input**, possibilmente utilizzando la redirezione fornita da shell (si veda l'esempio sotto).

Esempio

Il file `in1.txt` contiene le linee seguenti

```
A,Aldburg,0,60
A,Fornost,17,65
A,Nogrod,35,25
A,Umbar,43,17
A,Pelargir,60,55
A,Arceto,15,30
A,Edoras,100,80
C
```

e il comando (di redirezione da in1.txt) `java Esercizio1 < in1.txt` deve stampare

```
Tour [Aldburg, Fornost, Nogrod, Umbar, Pelargir, Arceto, Edoras] valido.
Lunghezza = 366.60
```

in quanto 366.60 è la lunghezza totale del percorso

$$Aldburg \to Fornost \to Nogrod \to Umbar \to Pelargir \to Arceto \to Edoras \to Aldburg$$

Il file `in1bis.txt` contiene le linee seguenti

```
A,Aldburg,0,60
A,Fornost,17,65
A,Nogrod,35,25
A,Umbar,43,17
A,Arceto,15,30
A,Fornost,17,65
A,Edoras,100,80
C
```

e il comando `java Esercizio1 < in1bis.txt` deve stampare

```
Tour [Aldburg, Fornost, Nogrod, Umbar, Arceto, Fornost, Edoras] non valido.
```

in quanto il percorso passa due volte per Fornost.

Esercizio 2

Si aggiunga alla classe `Tour` il seguente metodo:

- `Tour shortTour(City startCity)`
  Il metodo calcola un nuovo percorso utilizzando tutte le città del percorso che esegue il metodo. Il nuovo percorso inizia dalla città `startCity` fornita come argomento (`startCity` deve appartenere al percorso che esegue il metodo). La seconda città del nuovo percorso è la più vicina a `startCity`. Ogni città successiva nel nuovo percorso è la più vicina, fra quella non ancora utilizzate, a quella precedente (si veda l'esempio qui sotto).

Scrivere un programma `Esercizio2` che estende il programma precedente `Esercizio1` in modo che legga da **standard input** una sequenza di linee con i seguenti formati:

```
A,nome,coordX,coordY
C
N,nome
```

I comandi `A` e `C` sono gli stessi dell'Esercizio 1. Il comando `N` richiede l'esecuzione del metodo `shortTour` utilizzando come `startCity`, ovvero città di partenza del nuovo tour, la città `nome`.

Non è necessario fare controlli sulla correttezza dell'input. Non vanno fatte assunzioni sul numero di linee in input. L'input va letto da **standard input**, possibilmente utilizzando la redirezione fornita da shell (si veda l'esempio sotto).

Esempio

Il file `in2.txt` contiene le linee seguenti

```
A,Aldburg,0,60
A,Fornost,17,65
A,Nogrod,35,25
A,Umbar,43,17
A,Pelargir,60,55
A,Arceto,15,30
A,Edoras,100,80
C
N,Nogrod
```

e il comando `java Esercizio2 < in2.txt` deve stampare

```
Tour [Aldburg, Fornost, Nogrod, Umbar, Pelargir, Arceto, Edoras] valido.
Lunghezza = 366.60
Tour: [Nogrod, Umbar, Arceto, Aldburg, Fornost, Pelargir, Edoras]
Lunghezza = 269.91
```

In quanto Umbar è la città più vicina a Nogrod, Arceto è la più vicina a Umbar senza considerare Nogrod, Aldburg è la più vicina a Arceto senza considerare Nogrod e Umbar, e così via.

Istruzioni per la consegna

Consegnare soltanto i file seguenti (**non** i file `.class`):

```
City.java
Tour.java
Esercizio1.java
Esercizio2.java
```

come un unico file compresso creato col comando

```
> zip provalab City.java Tour.java Esercizio1.java Esercizio2.java
```

Eseguire l'upload del file all'indirizzo `upload.di.unimi.it`.

Non consegnare file che non compilano.