Esame Laboratorio di Programmazione - Edizione 1

25 febbraio 2014

Avvertenza: Nello svolgimento dell'elaborato è possibile usare qualunque classe delle librerie standard di Java. Non è invece ammesso l'uso delle classi del package `prog` allegato al libro di testo del Prof. Pighizzini e impiegato nella prima parte del corso.

L'obiettivo è scrivere una applicazione che permette di costruire delle torri usando dei blocchi. I blocchi sono numerati e possono essere di due tipi: blocchi *colorati*, che oltre al numero hanno un colore, e blocchi *jolly*.

Negli esempi, i blocchi di una torre sono elencati partendo dalla base (blocco più a sinistra) fino alla cima (blocco più a destra). Un esempio di torre è:

```
(42,blu) (92,blu) (11,JOLLY) (19,JOLLY) (61,verde) (12,verde)
```

La torre contiene 6 blocchi, di cui 4 colorati e 2 jolly. Il prossimo blocco inserito sarà posto sopra al blocco `(12,verde)`.

Le classi da implementare sono descritte sotto. Si noti che:

- Le classi e i metodi definiti nella specifica devono essere **pubblici**, i campi delle classi devono essere **privati**.
- Le classi, oltre ai metodi specificati, possono definire altri metodi che si ritengono utili (es., metodo `toString`, metodi set&get per accedere o modificare i campi, ecc.).
- L'intestazione di alcuni metodi è incompleta (vanno aggiunte le definizioni dei tipi).
- Le operazioni di input e output (stampa) vanno fatte esclusivamente nelle classi che definiscono il metodo `main`.
- Quando si usano le classi parametriche, ricordarsi di istanziarle (ad es., non scrivere `ArrayList`, ma `ArrayList<E>`, con `E` opportunamente istanziato).

Classe astratta `Blocco`

La classe astratta `Blocco` rappresenta un blocco. Ogni blocco ha un numero (intero positivo).

La classe possiede un costruttore che costruisce un blocco di cui viene specificato il numero. Il costruttore solleva una eccezione se il numero specificato non è positivo (si può usare una delle eccezioni definite nelle API, ad esempio `IllegalArgumentException`).

Classe `BloccoColorato`

Sottoclasse concreta di `Blocco` che rappresenta un blocco colorato. Un blocco colorato, oltre al numero, ha un colore (una stringa).

La classe possiede un costruttore che costruisce un blocco colorato di cui vengono specificati numero e colore. Il costruttore solleva una eccezione se il numero non è positivo oppure se il colore contiene più di 10 caratteri.

Classe `BloccoJolly`

Sottoclasse concreta di `Blocco` che rappresenta un blocco jolly.

La classe possiede un costruttore che costruisce un blocco jolly di cui viene specificato il numero. Il costruttore solleva una eccezione se il numero non è positivo.

Classe `Torre`

La classe `Torre` rappresenta una torre di blocchi. Ogni torre ha un nome e contiene i blocchi da cui è formata; non vanno posti limiti sul numero di blocchi che la torre può contenere.

Possiede un costruttore che costruisce una torre vuota di cui viene specificato il nome.

La classe definisce i metodi:

- `addBlocco(Blocco b)`

  Pone in cima alla torre il blocco `b`.

- `numColorati()`

  Restituisce il numero di blocchi colorati presenti nella torre.

- `numJolly()`

  Restituisce il numero di blocchi jolly presenti nella torre.

Esercizio 1

**** Se non si svolge questo esercizio non verranno valutati gli altri ****

Scrivere una classe `Esercizio1` in cui il metodo `main` esegue le istruzioni nella Figura 1 (alcune istruzioni sono da completare). Dopo aver inserito tutti i blocchi, la torre costruita deve contenere 14 blocchi, di cui 10 colorati e 4 jolly.

Costruzioni di torri con vincoli

Nei prossimi esercizi, nel costruire una torre vanno rispettati i seguenti **vincoli**:

1. un blocco colorato può stare sopra a un blocco colorato dello stesso colore oppure a un blocco jolly;
2. un blocco jolly può stare sopra a un qualunque altro blocco;
3. una torre non può contenere due blocchi con lo stesso numero.

Un esempio di torre che rispetta i vincoli è:

```
(42,blu) (92,blu) (11,JOLLY) (19,JOLLY) (61,verde) (12,verde)
```

Sulla torre (ossia, sopra al blocco `(12,verde)`) è ora possibile porre un blocco colorato di colore verde oppure un jolly; il numero del nuovo blocco deve essere diverso dai numeri dei blocchi presenti nella torre.

La specifica delle classi va arricchita nel modo descritto sotto.

```
Torre torre = ... // *** creare una nuova torre di nome alfa
torre.addBlocco( new BloccoColorato( 1, "verde") );
torre.addBlocco( new BloccoColorato( 5, "verde") );
torre.addBlocco( new BloccoColorato( 4, "rosso") );
torre.addBlocco( new BloccoJolly(10) );
torre.addBlocco( new BloccoJolly(20) );
torre.addBlocco( new BloccoJolly(1) );
torre.addBlocco( new BloccoColorato( 5, "rosso") );
torre.addBlocco( new BloccoColorato( 14, "rosso") );
torre.addBlocco( new BloccoColorato( 25, "rosso") );
torre.addBlocco( new BloccoJolly(30) );
torre.addBlocco( new BloccoColorato( 10, "azzurro") );
torre.addBlocco( new BloccoColorato( 52, "blu") );
torre.addBlocco( new BloccoColorato( 48, "grigio") );
torre.addBlocco( new BloccoColorato( 48, "blu") );
... // ***  Stampare tutta la torre
System.out.println("NUM. BLOCCHI COLORATI: " + torre.numColorati() );
System.out.println("NUM. BLOCCHI JOLLY: " + torre.numJolly() );
```

Figura 1: Codice del metodo `main` (da completare) per gli esercizi 1 e 2

(i) La classe astratta `Blocco` deve definire il metodo

- `abstract puoStareSopraA(Blocco b)`
  Metodo astratto che restituisce `true` se questo blocco può stare sopra al blocco `b` in base ai vincoli 1 e 2, `false` altrimenti.

(ii) Ciascuna delle sottoclassi `BloccoColorato` e `BloccoJolly` di `Blocco` deve definire il metodo `puoStareSopraA` concreto che implementa il metodo `puoStareSopraA` astratto della superclasse.

(iii) La classe `Torre` deve definire il metodo

- `containsBlocco(Blocco b)`
  Restituisce `true` se la torre contiene un blocco il cui numero è uguale al numero del blocco `b`, `false` altrimenti.

(iv) Il metodo `addBlocco` della classe `Torre` va modificato come segue (si consiglia di lasciare fra commenti il metodo `addBlocco` che si è già scritto):

- `addBlocco(Blocco b)`
  Pone in cima alla torre il blocco `b` se questo è possibile, cioè se sono rispettati i vincoli 1, 2 e 3.

Per verificare i vincoli, usare i metodi definiti sopra.

Se il metodo `addBlocco` è stato implementato correttamente, eseguendo ora la classe `Esercizio1` la torre costruita deve essere:

```
(1,verde) (5,verde) (10,JOLLY) (20,JOLLY) (14,rosso) (25,rosso) (30,JOLLY) (52,blu) (48,blu)
```

Esercizio 2

Scrivere una classe `Esercizio2` in cui il metodo `main` esegue le istruzioni nella Figura 1 (come nell'Esercizio 1). Successivamente, devono essere stampati tutti i blocchi presenti nella torre in ordine decrescente rispetto al numero (quindi, il primo blocco da stampare è quello avente il numero più alto). In alternativa, se non si riesce a ordinare i blocchi in ordine decrescente, ordinarli in ordine crescente.

Suggerimento. Occorre per prima cosa inserire i blocchi presenti nella torre in una opportuna struttura dati (la torre non va modificata!). Per ordinare i blocchi, evitare di implementare un algoritmo di ordinamento; le API dispongono di metodi che permettono di ordinare oggetti.

Esercizio 3

Scrivere una applicazione `Esercizio3` che permette di gestire più torri.

Il programma legge da standard input una sequenza di linee, ciascuna corrispondente a una operazione; quando una linea è letta, va compiuta l'operazione corrispondente.

Ogni linea in input può avere una dei seguenti formati:

```
T,nome_torre
+,numero_blocco,colore_blocco
+,numero_blocco
S
```

Il primo elemento della linea specifica l'operazione da compiere.

- Le linee che iniziano con `T` richiedono la costruzione di una nuova torre avente il nome specificato.
- Le linee che iniziano con `+` richiedono l'inserimento di un nuovo blocco in cima all'ultima torre creata con l'operazione `T`. L'inserimento è effettuato solo se è possibile farlo (è stata creata almeno una torre; valgono i vincoli 1, 2 e 3). In particolare, la linea

  `+,numero_blocco,colore_blocco`

  richiede l'inserimento di un blocco colorato avente numero e colore specificati. La linea

  `+,numero_blocco`

  richiede l'inserimento di un blocco jolly avente il numero specificato.

  Le eccezioni eventualmente sollevate dai costruttori delle classi `BloccoColorato` e `BloccoJolly` vanno catturate, in modo da evitare che l'applicazione termini.
- Le linee che iniziano con `S` richiedono la stampa di tutte le torri costruite.

Al termine dell'esecuzione, devono essere stampati tutti i blocchi presenti in tutte le torri in ordine decrescente rispetto al numero (come specificato nell'Esercizio 2).

Si assume che le linee di input abbiano sempre il formato specificato; non vanno fatte assunzioni sul numero di linee di input e neppure sul numero di torri che possono essere definite.

Esempio

Un esempio di linee di input è nella Figura 2. Con tale input, quando tutti i blocchi sono stati inseriti, le torri costruite sono:

```
TORRE alfa
(12,JOLLY) (16,verde) (11,JOLLY)
BLOCCHI COLORATI: 1 JOLLY: 2
TORRE beta
(35,rosso) (30,JOLLY) (38,JOLLY) (31,blu)
BLOCCHI COLORATI: 2 JOLLY: 2
TORRE gamma
(25,blu) (27,blu) (26,JOLLY) (28,JOLLY) (22,verde)
BLOCCHI COLORATI: 3 JOLLY: 2
```

Istruzioni per la consegna

Si ricorda che le classi devono essere tutte *public* e che vanno consegnati tutti i file `.java` prodotti; NON vanno consegnati i file `.class`.

Per la consegna, eseguite l'upload dei SINGOLI file sorgente (NON un file archivio!) dalla pagina web http://upload.di.unimi.it, autenticandosi per la sessione relativa al proprio turno (in caso di dubbi chiedere ai docenti).

ATTENZIONE!!! NON VERRANNO VALUTATI GLI ELABORATI CON ERRORI DI COMPILAZIONE.

Per ritirarsi occorre caricare un file (anche vuoto) di nome `ritirato.txt` e segnalare il ritiro al proprio docente.

```
+,12
T,alfa
+,12
+,16,verde
+,13,rosso
+,11
+13,blu
S
T,beta
+,35,rosso
+,35
+,30
+,38
+,31,blu
S
T,gamma
+,25,blu
+,-5,blu
+,27,blu
+,22,verde
+,25
+,26
+,28
+,22,verdeolivastro
+,22,verde
S
```

Figura 2: Esempio di linee di input per l'Esercizio 3