Esame di Programmazione 2

14 febbraio 2023

Note per la cosegna

- I file contenenti il codice vanno caricati alla pagina `https://upload.mat.unimi.it/` nella sessione dedicata all'esame. Vanno caricati solamente i file `.java`, non i file `.class`. In caso vengano consegnate più copie dello stesso file, verà corretta l'ultima versione.
- Prima di caricare un file su upload, verificare che sia compilabile; cancellare o includere in commenti le parti che provocano errori in compilazione. **I file che danno errori in compilazione non verranno valutati**. Può essere utile usare opzione `-Xlint` di `javac`, che controlla l'uso corretto dei tipi generici (`javac -Xlint MioFile.java`).
- **Non** vanno definiti package.

Torri

Lo scopo è definire delle classi che permettano di costruire delle torri formati da blocchi colorati. I blocchi possono essere numerati oppure blocchi Jolly, il cui colore è predefinito (colore di default). Una torre deve verificare i seguenti **vincoli**:

(V1) Un blocco numerato A può stare sopra a un blocco numerato B se e solo se:
(i) A e B hanno lo stesso colore, oppure
(ii) A e B hanno colori diversi e il numero di A è maggiore del numero di B.

(V2) Un blocco Jolly può trovarsi in qualunque posizione. Quindi, un Jolly può avere sopra e sotto di sè un blocco di qualunque tipo.

(V3) La torre non può contenere due blocchi numerati uguali, ossia due blocchi numerati aventi stesso colore e stesso numero.

Nel costruire una torre, un blocco può essere aggiunto solamente in cima alla torre stessa, purché vengano rispettati i vincoli (V1)-(V3). Esempi di torre che verificano i vincoli sono rappresentati nelle figure 2 e 3. Notare che il blocco in cima alla torre `alfa` in Fig. 2 è un blocco numerato di colore `rosso` e numero 4; a tale torre può essere aggiunto (sopra il blocco menzionato) (1) un blocco Jolly, OPPURE (2) un blocco numerato rosso (qualunque sia il suo numero), OPPURE (3) un blocco numerato avente colore diverso dal rosso e numero maggiore di 4; nei casi (2) e (3) il nuovo blocco numerato non deve essere uguale a un blocco già presente nella torre.

Le classi da realizzare sono quelle definite sotto e le eccezioni `ColoreException` e `NumeroException`; le eccezioni non vengono usate negli esercizi 1 e 2, quindi possono essere implementate dopo aver svolto tali esercizi. Le classi da definire devono essere pubbliche, inoltre:

- i costruttori e i metodi descritti nella specifica devono essere pubblici;
- i campi di una classe devono essere privati;
- nella specifica le segnature sono a volte scritte in modo incompleto (mancano alcuni tipi);
- è possibile aggiungere a una classe ulteriori metodi (pubblici o privati) non menzionati nella specifica;

– le classi elencate non devono compiere operazioni di stampa (`System.out.println`, ecc.); le uniche classi che possono farlo sono le classi che implementano una applicazione (classi con metodo `main`);
– è ammesso che i messaggi stampati dal proprio programma abbiano un formato diverso da quelli riportati nel testo, purché le informazioni siano equivalenti e chiaramente identificabili.

Si racconmanda di dare ai nuovi metodi dei nomi significativi, che esplicitino il loro significato, eventualmente scrivere commenti chiarificatori.

Classe `Blocco`

Classe astratta che descrive un blocco colorato. La classe definisce un colore di default, che va impostato durante l'esecuzione (usando l'apposito metodo descritto sotto) e non può successivamente essere modificato. La classe definisce i seguenti costruttori:

– Un costruttore che costruisce un blocco di cui si specifica il colore (una stringa).
– Un costruttore privo di argomenti che costruisce un blocco avente come colore il colore di default. Se il colore di default non è stato impostato, va sollevata l'eccezione `ColoreException`.

La classe deve definire i seguenti metodi pubblici:

- `setColoreDefault(colore)`
  Metodo statico che imposta il colore di default. Più precisamente, il colore di default è posto uguale al colore specificato dal parametro. Se il colore di default è già stato impostato, va sollevata l'eccezione `ColoreException`.
- `puoStareSopraA(Blocco b)`
  Metodo astratto che restituisce `true` se questo blocco può stare sopra al blocco `b` in base ai vincoli (V1) e (V2), `false` altrimenti.

Classe `BloccoNumerato`

Sottoclasse concreta della classe astratta Blocco che descrive un blocco numerato; il numero deve essere un intero positivo. La classe definisce i seguenti costruttori:

– Un costruttore che costruisce un blocco di cui si specifica il colore e il numero. Se il numero è minore o uguale a zero, va sollevata l'eccezione `NumeroException`.
– Un costruttore che costruisce un blocco di cui si specifica il numero; al blocco è assegnato il colore di default. Se il colore di default non è stato impostato, va sollevata l'eccezione `ColoreException`. Se il numero è minore o uguale a zero, va sollevata l'eccezione `NumeroException`.

La classe definisce il seguente metodo pubblico:

- `puoStareSopraA(BloccoNumerato b)`
  Restituisce `true` se e solo se questo blocco può stare sopra al blocco numerato `b` in base al vincolo (V1), `false` altrimenti.

Classe `BloccoJolly`

Sottoclasse concreta della classe astratta `Blocco` che descrive un blocco Jolly. La classe possiede un unico costruttore privo di argomenti che costruisce un blocco Jolly, a cui è assegnato il colore di default. Se il colore di default non è stato impostato, va sollevata l'eccezione `ColoreException`.

Classe `Torre`

Classe concreta che descrive una torre. Una torre è caratterizzata da un nome (stringa) e dai blocchi che la compongono. Per rappresentare i blocchi, definire un campo privato di tipo `List<T>` (con `T` opportunamente istanziato). Può essere utile definire un campo privato `top` che contiene un riferimento al blocco in cima alla torre (ricordarsi di aggiornare il valore di `top` dopo l'inserimento di un blocco). La classe possiede i seguenti costruttori:

- Un costruttore che costruisce una torre vuota di cui si specifica il nome.
- Un costruttore che costruisce una torre di cui si specifica il nome e che contiene un unico blocco specificato come argomento (quindi il costruttore ha complessivamente due argomenti).

La classe definisce i seguenti metodi pubblici.

- `add(Blocco b)`
  Aggiunge il blocco `b` in cima a questa torre, se questo è possibile, ossia se ciò è compatibile con i vincoli (V1)-(V3). Se non è possibile aggiungere `b`, il metodo non compie alcuna operazione.
- `totaleBlocchi()`
  Restituisce il numero totale di blocchi in questa torre.
- `totaleBlocchiNumerati()`
  Restituisce il numero totale di blocchi numerati in questa torre.
- `totaleBlocchiJolly()`
  Restituisce il numero totale di blocchi Jolly in questa torre.

Esercizio 1

Completare il file `Esercizio1.java` (vedi Fig. 1). Il comando `java Esercizio1` deve stampare le linee in Fig. 2.

Esercizio 2

Aggiungere alla classe `BloccoNumerato` i seguenti metodi statici e pubblici:

- `sort( List<BloccoNumerato> listBNum )`
  Ordina la lista `listBNum` (lista di blocchi numerati) in ordine crescente rispetto al colore; i blocchi con lo stesso colore vanno ordinati in ordine crescente rispetto al numero. Tale ordinamento va definito come l'*ordinamento naturale (natural order)* fra i blocchi numerati.
- `sort( List<BloccoNumerato> listBNum, cmp )`
  Ordina la lista `listBNum` (lista di blocchi numerati) in base all'ordine definito dal comparatore `cmp`. Notare che `cmp` deve avere tipo `Comparator<T>`, dove `T` va opportunamente specificato.

```
// Inserire sotto le linee di commento //####  le istruzioni corrispondenti
// Il resto non va modificato
public class  Esercizio1 {
    public static void main(String[] args)  {
//###   creare la torre vuota torreAlfa di nome alfa
//###   impostare come colore di default il colore oro
torreAlfa.add( new BloccoNumerato("verde",5) );
torreAlfa.add( new BloccoNumerato("verde",4) );
torreAlfa.add( new BloccoNumerato("rosso",3) ); // non inserito (vincolo sul numero)
torreAlfa.add( new BloccoNumerato("rosso",5) );
torreAlfa.add( new BloccoJolly() );
torreAlfa.add( new BloccoNumerato("nero",1) );
torreAlfa.add( new BloccoNumerato("rosso",3) );
torreAlfa.add( new BloccoNumerato("rosso",5) ); // non inserito (gia' presente)
torreAlfa.add( new BloccoNumerato("nero",3) );  // non inserito (vincolo sul numero)
torreAlfa.add( new BloccoNumerato("nero",4) );
torreAlfa.add( new BloccoNumerato("oro",3) );  // non inserito (vincolo sul numero)
torreAlfa.add( new BloccoNumerato("nero",3) );
torreAlfa.add( new BloccoNumerato("nero",1) );  // non inserito (gia' presente)
torreAlfa.add( new BloccoJolly() );
torreAlfa.add( new BloccoJolly() );
torreAlfa.add( new BloccoNumerato("verde",5) ); // non inserito (gia' presente)
torreAlfa.add( new BloccoNumerato("verde",7) );
torreAlfa.add( new BloccoNumerato("rosso",10) );
torreAlfa.add( new BloccoNumerato("rosso",4) );
System.out.println(torreAlfa);
System.out.println("Totale blocchi: " + torreAlfa.totaleBlocchi() );
System.out.println("Totale blocchi numerati: " + torreAlfa.totaleBlocchiNumerati() );
System.out.println("Totale blocchi Jolly: " + torreAlfa.totaleBlocchiJolly() );
    }// end main
}//end class
```

Figura 1. File `Esercizio1.java`

```
===  TORRE alfa ====
(rosso,4)
(rosso,10)
(verde,7)
(oro,JOLLY)
(oro,JOLLY)
(nero,3)
(nero,4)
(rosso,3)
(nero,1)
(oro,JOLLY)
(rosso,5)
(verde,4)
(verde,5)
--------------------
Totale blocchi: 13
Totale blocchi numerati: 10
Totale blocchi Jolly: 3
```

Figura 2. Linee stampate dalla applicazione `Esercizio1`; il blocco in cima alla torre è `(rosso,4)`.

Aggiungere alla classe `Torre` il metodo pubblico:

- `getBlocchiNumerati()`
  Restituisce la lista dei blocchi numerati contenenti in questa torre (l'ordine dei blocchi è irrilevante).

Definire la applicazione `Esercizio2` (classe pubblica con metodo `main`) che compie le seguenti operazioni:

1. Costruisce la torre `torreAlfa` come nell'Esercizio 1.
2. Stampa l'elenco dei blocchi numerati contenuti nella torre `torreAlfa` ordinati rispetto all'ordine naturale
3. Stampa l'elenco dei blocchi numerati contenuti nella torre `torreAlfa` in ordine decrescente rispetto al numero; a parità di numero, i blocchi vanno ordinati in ordine crescente rispetto al colore

Il comando `java Esercizio2` deve stampare:

```
==== BLOCCHI NUMERATI ORDINATI (ORDINE NATURALE) ====
(nero,1) (nero,3) (nero,4) (rosso,3) (rosso,4) (rosso,5) (rosso,10) (verde,4) (verde,5) (verde,7)
==== BLOCCHI NUMERATI ORDINATI IN ORDINE DECRESCENTE PER NUMERO ====
(rosso,10) (verde,7) (rosso,5) (verde,5) (nero,4) (rosso,4) (verde,4) (nero,3) (rosso,3) (nero,1)
```

Applicazione `Test`

Definire una applicazione `Test` che legge da standard input delle linee, ciascuna delle quali descrive una operazione da compiere. Il formato delle linee di input è il seguente.

(1) Linea della forma

`+,nomeTorre,colore,numero`

Aggiunge alla torre di nome `nomeTorre` un blocco avente il colore e il numero specificato, se questo è possibile (ossia, se non vengono violati i vincoli (V1)-(V3)). Se non esiste alcuna torre avente nome `nomeTorre`, viene creata una nuova torre di nome `nomeTorre` contenente il blocco specificato.
Vanno intercettate le eccezioni `ColoreException` e `NumeroException`; quando una di tali eccezioni è sollevata, va stampato un opportuno messaggio e la linea di input va ignorata.

(2) Linea della forma

`+,nomeTorre`

Come in (1), con la differenza che il blocco da aggiungere è un blocco Jolly. Notare che in questo caso può essere solamente sollevata l'eccezione `ColoreException`.

(3) Linea della forma

`S,colore`

Imposta il colore specificato come colore di default. Se viene sollevata l'eccezione `ColoreException`, va stampato un opportuno messaggio e la linea di input va ignorata.

(4) Linea della forma

`P`

Vanno stampate in ordine alfabetico tutte le torri costruite.

Si assume che le linee in input abbiano il formato richiesto.

Un esempio di esecuzione è mostrato in Fig. 3; sulla colonna di sinistra sono riportate le linee di input, in quella di destra le linee stampate dalla applicazione.

```
                          ColoreException: Il colore di default non e' stato impostato
+,beta,blu,20              ColoreException: Il colore di default non puo' essere modificato
+,beta,blu,15              NumeroException: -10: numero negativo
+,beta,verde,8             ########  TORRI ########
+,beta,verde,25            ====  TORRE alfa ====
+,beta                     (oro,JOLLY)
S,oro                     (oro,JOLLY)
+,beta                     (blu,70)
+,beta                     (rosso,10)
+,beta,blu,20              (rosso,1)
+,beta,giallo,20           --------------------
+,alfa,rosso,1             ====  TORRE beta ====
S,argento                 (verde,21)
+,alfa,rosso,10            (giallo,20)
+,alfa,verde,-10           (oro,JOLLY)
+,alfa,blu,70              (oro,JOLLY)
+,beta,verde,20            (verde,25)
+,beta,verde,21            (blu,15)
+,tau,rosso,30             (blu,20)
+,tau,verde,40             --------------------
+,tau                      ====  TORRE gamma ====
+,alfa                     (oro,JOLLY)
+,gamma,blu,20             (verde,30)
+,alfa                     (blu,20)
+,gamma,verde,30           --------------------
+,gamma                    ====  TORRE tau ====
+,gamma,blu,20             (rosso,31)
+,tau,rosso,30             (oro,JOLLY)
+,tau,rosso,31             (verde,40)
P                         (rosso,30)
                          --------------------
```

Figura 3. Input e output per la applicazione `Test`.