Laboratorio di Programmazione
Edizione Diurna - Turni A, B, C

Appello dell'11 luglio 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.

Tema d'esame

Lo scopo è realizzare un'applicazione per giocare a Nim. Si tratta di un gioco in cui due giocatori prelevano a turno biglie da un mucchio. In ciascun turno, il giocatore sceglie quante biglie prendere: deve prenderne almeno una, ma non più della metà del mucchio; quindi, se il numero di biglie è n, il giocatore può prelevare un numero di biglie k tale che $1 \leq k \leq n/2$. Quando $n = 1$, il giocatore di turno non può effettuare alcuna mossa e il gioco è vinto dall'avversario. Un esempio di partita è mostrato al termine dell'Esercizio 1.
Le classi da realizzare sono le seguenti (dettagli nelle sezioni successive):

- `NimPlayer`: un giocatore; ha un nome.
- `NimExpertPlayer`, `NimNonExpertPlayer`: sottoclassi di `NimPlayer`, rappresentano giocatori gestiti dal computer, ognuno con una sua strategia di gioco.
- `NimHumanPlayer`: sottoclasse di `NimPlayer`, rappresenta un giocatore umano che interagisce con il programma per giocare.
- `NimMarbles`: un mucchio di biglie, caratterizzato dal numero di biglie attualmente rimaste
- `NimGame`: una partita di Nim, caratterizzata da un mucchio di biglie e due giocatori.
- `Esercizio1`, `Esercizio2`, `Esercizio3`: classi che contengono il metodo `main`.

Le classi dovranno esporre almeno i metodi specificati nelle sezioni seguenti. Eventuali metodi di servizio possono essere aggiunti a piacimento. Ogni classe (eccetto quelle con metodo `main`) deve avere il metodo `toString()` che rappresenti lo stato delle istanze e i costruttori adeguati per gli attributi che vengono dichiarati. Dato che gli attributi devono essere tutti privati, creare opportunamente, e solo dove necessario, i metodi di accesso (`set` e `get`). Si suggerisce, anche dove non segnalato, di utilizzare, se esistenti e se applicabili, le classi parametriche (es. *ArrayList<E>* invece di *ArrayList*). Alcuni controlli di coerenza vengono suggeriti nel testo, potrebbero essercene altri a discrezione. Si consiglia di posporre l'implementazione dei controlli di coerenza, come ultima operazione, dopo aver realizzato un sistema funzionante.

Esercizio 1

Implementare le seguenti classi che permettono di gestire una partita a Nim tra due giocatori non esperti (computer contro computer).

abstract class NimPlayer

Classe astratta, rappresenta un generico giocatore, caratterizzato da un nome. La classe deve disporre dei seguenti metodi pubblici, oltre ad eventuali altri metodi che riterrete opportuni:

- `public NimPlayer(String nome)`: un costruttore che accetta in ingresso la stringa che rappresenterà il nome del giocatore, deve lanciare un'eccezione nel caso il parametro sia nullo.
- `int move(NimMarbles m)`: metodo astratto, deve restituire il numero di biglie che il giocatore vuole prelevare dal mucchio di biglie.
- `public String toString()`: descrive un giocatore indicandone il nome.

class NimNonExpertPlayer (extends NimPlayer)

Rappresenta un giocatore non esperto: la sua strategia di gioco è sottrarre dal mucchio un numero casuale di biglie, purché sia una quantità ammessa (compresa, quindi, fra 1 e $n/2$, dove n è il numero di biglie presenti nel mucchio).
La classe deve disporre dei seguenti metodi pubblici, oltre ad eventuali altri metodi che riterrete opportuni.

- `public NimNonExpertPlayer(String nome)`: un costruttore che accetta in ingresso la stringa che rappresenterà il nome del giocatore, deve lanciare un'eccezione nel caso il parametro sia nullo.
- `int move(NimMarbles m)`: deve restituire il numero di biglie che il giocatore preleverà dal mucchio: un numero casuale tra 1 e $n/2$, dove n è il numero di biglie presenti nel mucchio. Per generare numeri casuali, si consiglia di usare la classe `Random` del package `util`.
- `public String toString()`: descrive un giocatore indicandone il nome e il tipo.

class NimMarbles

Rappresenta un mucchio di biglie, è caratterizzata dal numero di biglie presenti; sono fissati inoltre il minimo (MIN) e il massimo (MAX) numero di biglie che può avere all'inizio di ogni partita. Ponete MIN = 10 e MAX = 100.
La classe deve disporre dei seguenti metodi pubblici, oltre ad eventuali altri metodi che riterrete opportuni.

- `public NimMarbles()`: un costruttore che crea un mucchio di n biglie, dove n è un numero casuale compreso tra MIN e MAX.
- `void takeMarbles(int m)`: toglie m biglie dal mucchio; solleva un'eccezione se il numero m è minore di 1 o maggiore di $n/2$, dove n è il numero di biglie presenti nel mucchio.
- `public String toString()`: non dimenticare!
  Facoltativo: implementare `toString` in modo che descriva un mucchio di biglie indicandone il numero e disegnandolo con asterischi in righe da 10. Ad esempio:

```
n. biglie: 23
**********
**********
***
```

class NimGame

Classe che descrive una partita di Nim, caratterizzata da un mucchio di biglie, da due giocatori e un turno che indica quale sarà il giocatore a fare la prossima mossa.
La classe deve disporre dei seguenti metodi pubblici, oltre ad eventuali altri metodi che riterrete opportuni.

- `public NimGame()`: un costruttore che crea una nuova partita. La classe crea un mucchio di biglie e, generando un numero intero casuale, zero o uno (utilizzare la classe `Random`), stabilisce il primo turno, cioè a quale giocatore tocca la prima mossa.
- `boolean isGameOver()`: verifica se la partita è finita, cioè se nel mucchio è rimasta una sola biglia.
- `int executeMove()`: fa eseguire al giocatore di turno la prossima mossa, avendo controllato che la partita non sia finita; ricordarsi di aggiornare il mucchio di biglie e il turno del prossimo giocatore.
- `void setPlayer1(NimPlayer p)` e `void setPlayer2(Nimplayer p)`: imposta i due giocatori della partita (l'ordine non è rilevante, il giocatore iniziale è scelto casualmente).
- `public String toString()`: non dimenticare!

class Esercizio1

La classe `Esercizio1` contiene il metodo `main`, oltre ad eventuali altri metodi che riterrete opportuni, e deve gestire una partita computer contro computer tra due giocatori non esperti (`NimNonExpertPlayer`); assegnare a ciascun giocatore un nome a piacere. Il programma dovrà visualizzare, di mossa in mossa, il mucchio di biglie, il giocatore di turno, e la mossa fatta. A fine partita dovrà indicare il vincitore.
Esempio di esecuzione:

```
java Esercizio1
n. biglie: 12
**********
**
tocca a Mary - non esperto
tolte 1 biglie
```

```
n. biglie: 11
**********
*
tocca a John - non esperto
tolte 3 biglie

n. biglie: 8
********
tocca a Mary - non esperto
tolte 4 biglie

n. biglie: 4
****
tocca a John - non esperto
tolte 1 biglie

n. biglie: 3
***
tocca a Mary - non esperto
tolte 1 biglie

n. biglie: 2
**
tocca a John - non esperto
tolte 1 biglie

John - non esperto: hai vinto!
```

Esercizio 2

Implementare la classe `Esercizio2` che ha lo stesso comportamento di `Esercizio1` e in più memorizza tutte le mosse effettuate durante il gioco. Al termine della partita, devono essere stampate tutte le mosse effettuate.
Per gestire la storia della partita, va modificata opportunamente la classe `NimGame`. La classe deve disporre dei metodi:

- `public addMove(int k)`
  Aggiunge alle mosse memorizzate la mossa "preleva k biglie". Si noti che non è necessario indicare chi effettua la mossa, in quanto questo può essere dedotto sapendo quale giocatore ha iniziato il gioco. Si assume che la mossa corrisponda a una mossa effettivamente svolta durante il gioco (quindi non occorre fare alcun controllo su k).
- `public String getHistory()`
  Restituisce una stringa che descrive la partita
  Esempio di possibile stringa:
  ```
  giocatori: John - non esperto, Mary - non esperto
  n biglie: 12
  Mossa 1: Mary toglie 1 biglie
  n. biglie: 12
  Mossa 2: John toglie 3 biglie
  ....
  Vince John
  ```

Esercizio 3

Implementare ora una delle classi `NimHumanPlayer` e `NimExpertPlayer` (o entrambe), che descrivono nuovi tipi di giocatore. Successivamente, implementare la classe `Esercizio3` che gestisce un gioco in cui possono prendere parte i nuovi tipi di giocatori.

class NimHumanPlayer (extends NimPlayer)

Rappresenta un giocatore umano, deve quindi essere in grado di interagire con l'utente (ad esempio, deve chiedere all'utente quante biglie vuole togliere).
La classe deve disporre dei seguenti metodi pubblici, oltre ad eventuali altri metodi che riterrete opportuni.

- `public NimHumanPlayer(String nome)`: un costruttore che accetta in ingresso la stringa che rappresenterà il nome del giocatore, deve lanciare un'eccezione nel caso il parametro sia nullo.
- `int move(NimMarbles m)`: deve chiedere all'utente il numero di biglie da prelevare dal mucchio; solleva un'eccezione se il numero fornito dall'utente (da tastiera) non è compreso tra 1 e $n/2$, dove n è il numero di biglie presenti nel mucchio.
- `public String toString()`: descrive un giocatore indicandone il nome e il tipo.

class NimExpertPlayer (extends NimPlayer)

Rappresenta un giocatore esperto: la sua strategia di gioco è sottrarre un numero di biglie affinché il numero di quelle rimanenti sia uguale a una potenza di due, meno uno: 3, 7, 15, 31, ... Questa è sempre una mossa possibile tranne quando il numero delle biglie presenti è uguale a una potenza di due meno uno: in questo caso, il computer preleverà dal mucchio un numero casuale di biglie, purché sia un numero ammesso (compreso, quindi, fra 1 e $n/2$, dove n è il numero di biglie presenti nel mucchio).
La classe deve disporre dei seguenti metodi pubblici, oltre ad eventuali altri metodi che riterrete opportuni.

- `public NimExpertPlayer(String nome)`: un costruttore che accetta in ingresso la stringa che rappresenterà il nome del giocatore, deve lanciare un'eccezione nel caso il parametro sia nullo.
- `int move(NimMarbles m)`: deve restituire il numero di biglie che il giocatore preleverà dal mucchio, cioè un numero compreso tra 1 e $n/2$, dove n rappresenta il numero di biglie presenti nel mucchio, determinato secondo la strategia del giocatore esperto (vedi sopra).
  Suggerimento: Si consiglia di implementare un metodo privato `int intLog2(int n)` che calcoli la parte intera del logaritmo in base 2 di un numero n, ricordando che

  $$\log_2(x) = \log_{10}(x)/\log_{10}(2)$$

  e che un *cast* a `int` di un `double` d implicitamente restituisce la parte intera di d. Per il logaritmo in base 10, utilizzare il metodo `Math.log10(double x)`. Per calcolare il numero m di biglie da togliere si può utilizzare la seguente formula: $m = n - (2^k - 1)$, dove n è il numero di biglie nel mucchio e $k = \lfloor \log_2(n) \rfloor$ (i.e. la parte intera di $\log_2(n)$)
- `public String toString()`: descrive un giocatore indicandone il nome e il tipo

class Esercizio3

La classe `Esercizio3` contiene il metodo `main`, oltre ad eventuali altri metodi che riterrete opportuni. La classe deve gestire partite in cui i giocatori possono essere utenti o computer. Di seguito si specifica il comportamento di `Esercizio3` nel caso si siano implementate entrambe le classi `NimHumanPlayer` e `NimExpertPlayer`; se si è implementata solo una classe, la specifica va adattata nel modo opportuno.
La classe classe `Esercizio3` deve essere "lanciata" specificando di seguito su linea di comando un nome (il nome del giocatore) se si vuole lanciare una partita contro un utente o "computer" o niente se si vuole lanciare una partita computer contro computer.
Rispetto alla classe `Esercizio1`, cambia solo che devono essere determinati i tipi dei due giocatori. Per quanto riguarda il o i giocatori gestiti dal computer, stabilire generando un numero intero casuale, zero o uno, se il computer giocherà in modo esperto o non esperto.

Consegna

Si ricorda che le classi devono essere tutte *public* e che vanno consegnati tutti i file *.java* prodotti. NON vanno consegnati i *.class* Per la consegna, eseguite l'upload dei SINGOLI file sorgente (NON un file archivio!) dalla pagina web: http://upload.di.unimi.it

ATTENZIONE!!! NON VERRANNO VALUTATI GLI ELABORATI CON ERRORI DI COMPILAZIONE. UN SINGOLO ERRORE DI COMPILAZIONE INVALIDA **TUTTO** L'ELABORATO.

- **Per ritirarsi**:
  Fare l'upload di un file vuoto di nome `ritirato.txt`.