Laboratorio di Programmazione
Edizione 1 - Turni A, B, C

ESAME del 15 Giugno 2015

Avvertenze

- 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.
- Si consiglia CALDAMENTE l'utilizzo dello script "checker.sh" per compilare ed effettuare una prima valutazione del proprio elaborato.
- Ricordarsi, quando si programma: *Repetita NON iuvant* o DRY (*Don't Repeat Yourself*).

1 Tema d'esame

Lo scopo dell'esercizio è realizzare un'applicazione che gioca al gioco dell'oca. Il gioco dell'oca si gioca su un tabellone sul quale è disegnato un percorso composto da un certo numero di caselle contrassegnate con numeri o altri simboli. I giocatori iniziano con un segnalino nella casella di partenza e, a turno, procedono lungo il percorso di un numero di caselle ottenuto attraverso il lancio di un dado. Alcune caselle di arrivo hanno un effetto speciale (tornare alla partenza, ripetere il movimento appena effettuato, ecc.). In questa versione del gioco, i giocatori sono tutti dotati di un'energia iniziale pari a 100 e, ogni volta che si spostano, usano una quantità di energia pari all'esito del lancio del dado. Inoltre su ogni casella può esserci al piú un giocatore e in caso di collisione uno dei due torna alla partenza. Lo scopo del gioco è raggiungere la casella finale o rimanere unico giocatore.

Classi Le classi da realizzare sono le seguenti (dettagli nelle sezioni successive):

- `Casella`: classe astratta, rappresenta una casella generica
- `CasellaStandard`: sottoclasse di `Casella`, rappresenta caselle senza effetti speciali.
- `CasellaRaddoppia`: sottoclasse di `Casella`, rappresenta caselle con l'effetto speciale di far ripetere il movimento appena effettuato.
- `CasellaTornaAllaPartenza`: sottoclasse di `Casella`, rappresenta caselle con l'effetto speciale di far andare il giocatore alla casella di partenza.
- `Tabellone`: è formato da caselle, rappresenta un tabellone.
- `Giocatore`: rappresenta un giocatore.
- `Partita`: rappresenta una partita su un tabellone con i giocatori.

2 Specifica delle classi

Le classi (**pubbliche**!) dovranno esporre almeno i metodi e costruttori **pubblici** specificati, più eventuali altri metodi e costruttori se ritenuti opportuni.
Gli attributi (campi) delle classi devono essere *privati*.
Se si usano tipi generici, si suggerisce di utilizzarne le versioni opportunamente istanziate (es. `ArrayList<String>` invece di `ArrayList`).
Ogni classe deve avere il metodo `toString` che rappresenti lo stato delle istanze.

2.1 class Casella

Rappresenta una generica casella. Può avere un giocatore (il giocatore che ha la pedina sulla casella stessa). Deve disporre dei seguenti metodi pubblici:

- `int destinazioneGiocatore(int pos, int lancio)`
  Metodo astratto, restituisce la nuova posizione a cui andare. Ogni tipo di casella lo implementerà opportunamente.
- `void partenzaGiocatore()`
  Toglie dalla casella il giocatore di turno presente.

- `void arrivoGiocatore(Giocatore nuovo)`
  Accoglie il nuovo giocatore destinato lí. Se sulla casella c'è già un giocatore, questo verrà sostituito dal nuovo giocatore solo se quest'ultimo ha energia piú alta dell'altro (cioè a parità di energia, rimarrà il vecchio giocatore).
- `Giocatore getGiocatore()`
  Restituisce il giocatore presente sulla casella, `null` se non c'è nessuno.

2.2 class CasellaStandard

Sottoclasse di `Casella`, rappresenta caselle standard, cioè senza effetti speciali. Deve implementare il metodo pubblico:

- `int destinazioneGiocatore(int pos, int lancio)`.
  La casella è standard, quindi non ci sono effetti speciali di cui tenere conto. Restituisce la nuova posizione a cui andare, data dalla posizione corrente sommata all'esito del lancio.

2.3 class CasellaRaddoppia

Sottoclasse di `Casella`, rappresenta caselle con l'effetto speciale che il giocatore che vi arriva con un lancio di dado, avanza di un ulteriore numero di caselle pari al lancio appena fatto. Deve implementare il metodo pubblico:

- `int destinazioneGiocatore(int pos, int lancio)`.
  Restituisce la nuova posizione a cui andare, tenendo conto dell'effetto speciale 'raddoppia'.

2.4 class CasellaTornaAllaPartenza

Sottoclasse di `Casella`, rappresenta caselle con l'effetto speciale che il giocatore che vi arriva con un lancio di dado, deve tornare alla prima casella del tabellone.

- `int destinazioneGiocatore(int pos, int lancio)`.
  Restituisce la nuova posizione a cui andare, tenendo conto dell'effetto speciale 'torna alla partena'.

2.5 class Tabellone

Rappresenta un tabellone di gioco, cioè una sequenza di Caselle. Deve disporre dei seguenti costruttori e metodi pubblici oltre ad eventuali altri metodi che riterrete opportuni.

- `Tabellone(int dim)`.
  Costruttore che accetta un int per il numero di caselle. Solleva un'eccezione se dim è minore di 50 o maggiore di 100. Predispone un tabellone di caselle tutte standard tranne quelle a un quarto, a metà e a tre quarti che sono CasellaTornaAllaPartenza e quelle dalla nona ogni nove che sono CasellaRaddoppia; in ogni caso però la prima e l'ultima devono essere Standard.
- `void spostaGiocatore(Casella casella, int lancio)`.
  Sposta il giocatore in casella `casella` dalla sua posizione corrente alla casella di destinazione determinata in base al lancio del dado e al tipo di casella (Standard, Raddoppia, TornaAllaPartenza) in cui il giocatore arriva con il lancio. Se la destinazione è fuori dal tabellone o se il giocatore non ha piú energia, il giocatore non si sposta.
- `boolean vittoria()`.
  Verifica la condizione di fine partita: se un giocatore è arrivato al traguardo (ultima casella).
- `Casella getCasella(int i)`.
  Restituisce la casella di posto i nel tabellone.
- `int getIndexCasella(Casella c)`.
  Restituisce la posizione nel tabellone della casella c.
- `Casella getCasella(Giocatore g)`.
  Restituisce la casella su cui c'è il giocatore g.

2.6 class Giocatore

Rappresenta un giocatore con nome, livello di energia e stato. L'energia iniziale è fissata a 100 per tutti i giocatori. Un giocatore è in uno di due possibili stati: con energia (operativo) oppure senza energia (inoperativo). La classe deve implementare Comparable<Giocatore > e il confronto è sul livello di energia. Deve inoltre disporre dei seguenti costruttori e metodi pubblici oltre ad eventuali altri metodi che riterrete opportuni. NON deve invece disporre di un metodo getEnergia().

- `Giocatore(String nome)`
  Costruttore che accetta una stringa per il nome. Solleva un'eccezione se viene passata una stringa nulla.
- `void usaEnergia(int n)`

Diminuisce l'energia del giocatore di una quantità n pari all'esito del lancio del dado. Se l'energia residua è minore o uguale a zero, il giocatore diventa inoperativo.
- `boolean haEnergia()`
  Restituisce `true` se il giocatore ha energia maggiore di zero.
- `String toString()`
  Restituisce la descrizione dell'istanza di giocatore (nome e energia).

2.7 class Partita

Rappresenta una partita, descritta da un tabellone di gioco e dai giocatori che vi partecipano. Si consiglia di utilizzare un array per i giocatori. Deve disporre dei seguenti costruttori e metodi pubblici oltre ad eventuali altri metodi che riterrete opportuni.
- `Partita(int numGiocatori, int numCaselle)`.
  Costruttore che accetta il numero di giocatori per la partita e il numero di caselle del tabellone. Solleva un'eccezione se il numero di giocatori è minore di 2 o maggiore di 8.
- `void setGiocatori(String fileName)`. Inizializza l'array dei giocatori leggendone i nomi dal file fileName.
- `void gioca()`.
  Gestisce una partita, facendo muovere ciascun giocatore al suo turno, se ha energia, finché un giocatore arriva al traguardo, oppure rimane un solo giocatore operativo. Quando un giocatore deve partire, non è associato a nessuna casella e occorre quindi assegnarlo alla prima casella del tabellone e poi farlo avanzare da lí. Alla fine stampa l'eventuale vincitore, specificando se è arrivato al traguardo o è l'unico con energia. Es.:
  `Bruno (energia 13) sei arrivato al traguardo, hai vinto!`
  oppure `Ada (energia 18) sei rimasto solo tu, hai vinto!`
- `String stampaSituazione()`.
  Restituisce la posizione del traguardo e, per ogni giocatore, la sua descrizione e la casella in cui si trova (-1 se non si trova in nessuna casella). Ad esempio:
  ```
  Ada (energia 56) -- casella 12
  Bruno (energia 30) -- casella 58
  Carla (energia 56) -- casella -1
  Traguardo 58
  ```
- `boolean fine()`. Restituisce `true` se è rimasto al piú un giocatore con energia o se un giocatore ha raggiunto il traguardo.
- `int lancioDado()`.
  Metodo statico che restituisce l'esito di un lancio di un dado (un numero casuale tra 1 e 6).

3 Raccomandazioni

La codifica delle classi dovrebbe essere svolta in modo incrementale, ed accompagnata da frequenti sessioni di compilazione e da test (parziali).
Affinché l'elaborato sia valutato, è richiesto che sia le classi sviluppate che i test risultino *compilabili*. A tal fine i metodi/costruttori che non saranno sviluppati dovranno comuque avere una implementazione fittizia come la seguente :

```
public void nomeDelMetodo () {
   throw new UnsupportedOperationException();
}
```

Si suggerisce quindi di dotare da subito le classi di tutti i metodi richiesti, implementandoli in modo fittizio, e poi di sostituire man mano le implementazioni fittizie con implementazioni che rispettino le specifiche.

4 Consegna

Si ricorda che le classi devono essere tutte *public* e che vanno consegnati tutti i file *.java* prodotti.
NON vanno consegnati i *.class*.
NON vanno consegnati i file relativi al meccanismo di autovalutazione (*Test_*.java, AbstractTest.java, *.sh*).
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`.
