Appelli di gennaio, febbraio e aprile 2005

Carte

Laurea triennale in Comunicazione Digitale
Laboratorio di Informatica Generale

1 – Descrizione

Il progetto consiste nel realizzare un programma per gestire l'estrazione di carte da gioco da un mazzo, con conseguente valutazione del loro valore in una serie di giochi.

2 – Le classi da realizzare

E' richiesto di realizzare in JAVA il programma descritto nella sezione precedente utilizzando le seguenti classi:

- `Seme`, che descrive uno tra i semi di Cuori, Denari, Fiori e Picche. La classe dovrà contenere:
  - la definizione delle costanti intere `CUORI`, `DENARI`, `PICCHE` e `FIORI`, inizializzate a valori numerici opportuni;
  - la variabile d'istanza `s`, di tipo `int`, che in ogni oggetto della classe dovrà contenere una delle costanti definite al punto precedente;
  - un costruttore `Seme(int)`, che inizializza un oggetto della classe quando l'argomento passato corrisponde a una delle costanti sopra definite, altrimenti lancia un'eccezione `InvalidCardException` (la cui corrispondente classe dovrà essere definita in modo opportuno).
  - i metodi di accesso alla variabile d'istanza, `void set(int)` e `int get()`. Tali metodi devono lanciare l'eccezione `InvalidCardException` in tutti i casi in cui venga fatto riferimento a un valore intero diverso da quelli corrispondenti alle costanti indicate nel punto precedente;
  - il metodo `String toString()`, che ritorna una delle stringhe "cuori", "denari", "fiori" o "picche", a seconda del valore contenuto nella variabile di istanza;
  - l'implementazione dell'interfaccia `Comparable`, nel modo giudicato più opportuno per la realizzazione globale del progetto.
- `Valore`, che indica un generico valore di una carta. La classe dovrà contenere
  - la definizione delle costanti intere `ASSO`, `DUE`, `TRE`, `QUATTRO`, `CINQUE`, `SEI`, `SETTE`, `OTTO`, `NOVE`, `DIECI`, `FANTE`, `DONNA` e `RE`, inizializzate a valori numerici opportuni;

- la variabile d'istanza `v`, di tipo `int`, che in ogni oggetto della classe dovrà contenere una delle costanti definite al punto precedente;
- un costruttore `Valore(int)`, che inizializza un oggetto della classe quando l'argomento passato corrisponde a una delle costanti sopra definite, altrimenti lancia un'eccezione `InvalidCardException`;
- i metodi di accesso alla variabile d'istanza, `void set(int)` e `int get()`. Tali metodi devono lanciare l'eccezione `InvalidCardException` in tutti i casi in cui venga fatto riferimento a un valore intero diverso da quelli corrispondenti alle costanti indicate nel punto precedente;
- il metodo `String toString()`, che ritorna una delle stringhe "asso", "due", "tre", "quattro", "cinque", "sei", "sette", "otto", "nove", "dieci", "fante", "donna" o "re", a seconda del valore contenuto nella variabile di istanza;
- l'implementazione dell'interfaccia `Comparable`, nel modo giudicato più opportuno per la realizzazione globale del progetto.

- `Carta`, che indica una generica carta da gioco. La classe dovrà contenere:
  - La variabile d'istanza `Seme s`, che conterrà il seme della carta;
  - la variabile d'istanza `Valore v`, che conterrà il valore della carta;
  - il costruttore `Carta(Seme, Valore)`, che istanzierà un oggetto della classe sulla base di un seme e di un valore specificati come argomento;
  - il costruttore `Carta(int, int)`, che istanzierà un oggetto della classe sulla base di due valori interi da interpretare rispettivamente come il seme e il valore della carta, lanciando l'eccezione `InvalidCardException` nel caso i valori degli argomenti risultino differenti da quelli che indicano una delle costanti definite nelle classi `Seme` e `Valore`;
  - i metodi di accesso alle variabili d'istanza, `void setSeme(Seme)`, `void setValore(Valore)`, `int getSeme()` e `int getValore()`. Tali metodi devono lanciare l'eccezione `InvalidCardException` in tutti i casi in cui venga fatto riferimento a valori interi diversi da quelli che indicano una delle costanti definite nelle classi `Seme` e `Valore`;
  - il metodo `String toString()`, che ritorna la descrizione testuale dell'oggetto, ottenuta combinando opportunamente i valori ritornati dai metodi `toString()` nelle classi `Seme` e `Valore`;
  - l'implementazione dell'interfaccia `Comparable`, nel modo giudicato più opportuno per la realizzazione globale del progetto.
- `Mazzo`, che indica un generico mazzo di carte. La classe dovrà contenere:
  - La definizione della costante intera `MAXCARTE`, inizializzata a un valore numerico opportuno, che indica il numero di carte contenute nel mazzo;
  - la variabile d'istanza `Carta c[]`, contenente il riferimento a un array di oggetti della classe `Carta`;

- la variabile d'istanza `int prossimaCarta`, contenente la posizione dell'array in cui si trova la prossima carta che sarà pescata dal mazzo;
- il costruttore `Mazzo()`, che istanzia un oggetto della classe, inizializzando la variabile `prossimaCarta` a un valore opportuno, creando i singoli oggetti della classe `Carta` che corrispondono a ogni possibile carta del mazzo e inserendoli nell'array `c`. Si noti che non è richiesto che gli oggetti della classe Carta siano inseriti nell'array in un ordine particolare. E' però obbligatorio inserire nell'array uno e un solo oggetto per ogni possibile combinazione di seme e valore;
- il metodo `String toString()`, che ritorna una descrizione testuale dell'intero mazzo, costituita, per ogni posizione dell'array `c`, da una riga contenente la descrizione del corrispondente oggetto fornita dal metodo `toString()` della classe `Carta`;
- il metodo `void mescola()`, che mescola le carte contenute nel mazzo (si faccia riferimento alla Sezione 3 per un possibile algoritmo su cui basare l'implementazione di questo metodo); tale metodo deve anche reimpostare in modo opportuno il valore della variabile d'istanza `prossimaCarta`;
- il metodo `Carta pesca()`, che "pesca" una carta, ritornando l'oggetto della classe `Carta` contenuto, nell'array `c`, nella posizione indicata dalla variabile d'istanza `prossimaCarta`. Il metodo, che si occupa anche di incrementare `prossimaCarta` di un'unità, dovrà lanciare l'eccezione `NoMoreCardsException` (la cui corrispondente classe dovrà essere definita in modo opportuno) quando si tenta di pescare una carta oltre la fine del mazzo;
- il metodo `void distribuisci(Mano, Mano)`, che accetta come argomento due riferimenti ad oggetti della classe `Mano` (vedi oltre), di cui ridefinisce i contenuti pescando un numero sufficiente di carte dal mazzo, che verranno distribuite nel modo seguente: la prima al primo oggetto, la seconda al secondo oggetto, la terza al primo oggetto e così via. Anche questo metodo dovrà lanciare l'eccezione `NoMoreCardsException` quando si tenta di pescare carte oltre la fine del mazzo.

- `Mano`, che indica la mano, consistente di cinque carte pescate da un mazzo, di un generico gioco. La classe dovrà contenere:
  - La definizione della costante intera `NUMCARTE`, inizializzata al valore numerico 5;
  - la variabile d'istanza `Carta c[]`, contenente il riferimento a un array di oggetti della classe `Carta`;
  - il costruttore `Mano(Mazzo)`, che inizializza un oggetto della classe, ottenendo i singoli oggetti della classe `Carta` tramite chiamate al metodo `pesca()` dell'oggetto passato come argomento e inserendoli nell'array `c`;
  - i metodi `Carta getCarta(int)` e `void setCarta(int, Carta)`, che ritornano e impostano, rispettivamente, l'oggetto della classe `Carta` contenuto nell'array `c` in una specifica posizione;

- il metodo `String toString()`, che ritorna una descrizione testuale della mano, costituita, per ogni posizione dell'array `c`, da una riga contenente la descrizione del corrispondente oggetto fornita dal metodo `toString()` della classe `Carta`;
- il metodo `void ordina()`, che ordina gli oggetti contenuti nell'array `c` sulla base dell'ordinamento definito implicitamente dall'implementazione della classe `Carta`;
- il metodo astratto `int valore()`, che ritorna un intero rappresentante il valore numerico di una mano in un particolare gioco (vedi la definizione delle classi ottenute estendendo `Mano`).

- `ManoPiuAlta` (ottenuta estendendo `Mano`), che rappresenta la mano di un gioco in cui vince chi ha la carta con il valore più alto. Tale classe deve contenere:
  - l'implementazione del metodo `valore` ereditato da `Mano`, che deve ritornare l'intero corrispondente al valore della carta della mano che ha il valore più alto;
  - il costruttore `ManoPiuAlta(Mazzo)`, da implementare tramite chiamata all'analogo costruttore della classe `Mano`.
- `ManoPiuBassa` (ottenuta estendendo `Mano`), che rappresenta la mano di un gioco in cui vince chi ha la carta con il valore più basso. Tale classe deve contenere:
  - l'implementazione del metodo `valore` ereditato da `Mano`, che deve ritornare l'intero corrispondente al valore della carta della mano che ha il valore più basso;
  - il costruttore `ManoPiuBassa(Mazzo)`, da implementare tramite chiamata all'analogo costruttore della classe `Mano`.
- `ManoUguali` (ottenuta estendendo `Mano`), che rappresenta la mano di un gioco in cui vince chi ha il maggior numero di carte con uguale valore. Tale classe deve contenere:
  - l'implementazione del metodo `valore` ereditato da `Mano`, che deve ritornare il maggior numero di carte nella mano aventi uguale valore, partendo dalle coppie (cioè 0 se tutte le carte sono diverse, 2 se si ha almeno una coppia, nessun tris e nessun poker, 3 se si ha un tris e nessun poker e 4 se si ha un poker);
  - il costruttore `ManoUguali(Mazzo)`, da implementare tramite chiamata all'analogo costruttore della classe `Mano`.
- `ManoBriscola` (ottenuta estendendo `Mano`), che rappresenta la mano di un gioco in cui vince chi ha la carta con il valore più alto, fissato un determinato seme (la briscola, appunto). Tale classe deve contenere:
  - la variabile di istanza `Seme briscola`, che indica qual è il seme corrispondente alla briscola;
  - l'implementazione del metodo `valore` ereditato da `Mano`, che deve ritornare l'intero corrispondente al valore della carta della mano che ha il valore più alto, tra le carte il cui seme è lo stesso di quello contenuto nella variabile d'istanza `briscola` e 0 se la mano non contiene carte avente lo stesso seme della briscola;

- il costruttore `ManoBriscola(Mazzo, Seme)`, da implementare tramite chiamata al costruttore della classe `Mano`, unitamente all'impostazione della variabile d'istanza `briscola` al valore specificato tramite il secondo argomento.

- `InvalidCardException` e `NoMoreCardsException`, da definire in modo opportuno, che rappresentano le eccezioni da lanciare quando si verificano le condizioni di errore descritte nei punti precedenti.
- `Torneo`, che rappresenta la classe da lanciare per eseguire il progetto. Questa classe contiene solamente il metodo `main`, in cui viene chiesto all'utente di scegliere tra
  - uno dei giochi relativi alle classi `ManoPiuAlta`, `ManoPiuBassa`, `ManoUguali` e `ManoBriscola`,
  - un numero di giocate
  - una modalità di distribuzione delle carte dal mazzo (tra quelle corrispondenti, rispettivamente, al costruttore delle classi derivate da `Mano` e all'invocazione del metodo distribuisci nella classe `Mazzo`).

  Fatte queste scelte, il metodo deve simulare il gioco selezionato, per il numero di giocate richieste, tra due giocatori, tenendo conto del numero di vittorie per ognuno di loro. Al termine deve essere riportato il numero totale di vittorie per ogni giocatore e determinare quale dei due ha vinto il maggior numero di volte, oppure l'eventuale situazione di pareggio. Il metodo deve anche prevedere la gestione di eventuali eccezioni che possono venire lanciate durante l'esecuzione della classe.

A parte quanto espressamente richiesto, è lasciata piena libertà sull'implementazione delle singole classi e sull'eventuale introduzione di altre classi, a patto di seguire le regole del paradigma ad oggetti ed i principi di buona programmazione. Si suggerisce di porre particolare attenzione alla scelta dei modificatori relativi a variabili d'istanza e metodi, nonchè alla dichiarazione delle eccezioni che possono venire lanciate dai vari metodi.

Non è richiesto l'utilizzo di particolari modalità grafiche di visualizzazione: è sufficiente una qualunque modalità di visualizzazione basata sull'uso dei caratteri.

E' invece **espressamente richiesto** di non utilizzare package non standard di Java (si possono quindi utilizzare `java.util`, `java.io` e così via), con l'unica eccezione package `prog.io` incluso nel libro di testo per gestire l'input da tastiera e l'output a video.

3 – Mescolare il mazzo di carte

Per mescolare il mazzo di carte è possibile ripetere per un certo numero di volte la seguente operazione elementare: scelte due carte a caso all'interno del mazzo, scambiare le loro posizioni. Il seguente algoritmo descrive questa procedura in modo più dettagliato

- inizializzare a un valore opportuno le costanti `MINSCAMBI` e `MAXSCAMBI`, che indicheranno rispettivamente il numero minimo e massimo di scambi tra carte;
- estrarre un numero pseudocasuale compreso tra `MINSCAMBI` e `MAXSCAMBI` e assegnarlo alla variabile `numScambi`;
- eseguire il seguente codice per un numero di volte pari al valore contenuto in `numScambi`;
  - estrarre due posizioni a caso nell'array di carte e assegnarle rispettivamente alle variabili `x` e `y`;
  - scambiare tra di loro gli oggetti dell'array di carte contenuti alle posizioni `x` e `y`

Per quanto riguarda le estrazioni a caso, il metodo `Math.random()` ritorna un valore di tipo double scelto in modo pseudocasuale nell'intervallo tra 0 e 1. Pertanto, fissato un generico valore positivo `a`, l'espressione `Math.random()*a` ritornerà un valore **di tipo double** scelto in modo pseudocasuale tra 0 e `a`.

E' possibile utilizzare altri algoritmi per mescolare il mazzo di carte, a patto che

- ad ogni esecuzione dell'algoritmo si ottenga una differente configurazione del mazzo mescolato;
- il mazzo mescolato (così come quello originale) contenga uno e un solo oggetto della classe `Carta` per ogni possibile combinazione di seme e valore.

4 – Modalità di consegna

Il progetto può essere svolto al massimo da tre persone che intendono sostenere l'intero esame di Informatica Generale e Laboratorio negli appelli di Gennario, Febbraio o Aprile 2005, e deve essere consegnato **entro mezzanotte di mercoledì 9 febbraio 2005**, utilizzando il sito di sottoposizione delle esercitazioni (all'indirizzo `http://infogen.dsi.unimi.it`). Per poter effettuare la sottoposizione è necessario autenticarsi utilizzando un nome di login e una password. Nella pagina principale del sito stesso è spiegato come ottenere questi dati. Nel caso il progetto venga svolto da più di una persona, dovrà essere fatta in ogni caso **una sola sottoposizione**,

indicando chiaramente in un commento all'inizio dei sorgenti consegnati nome, cognome e matricola dei vari componenti del gruppo

Dovranno essere consegnati tutti i **sorgenti** Java che permettano al programma di essere compilato ed eseguito correttamente

- compressi in un archivio di tipo ZIP che estragga i file nella directory in cui si trova l'archivio stesso, **oppure**
- contenuti in un unico file in cui tutte le classi **non devono essere dichiarate di tipo public**;

altri tipi di sottoposizioni verranno automaticamente rifiutate dal sito.

All'archivio dovrà anche essere accluso un breve documento **in formato txt o rtf** in cui

- verrà descritto il modo in cui interfacciarsi con il programma
- saranno illustrate le principali scelte implementative e le strategie utilizzate per svolgere il progetto

Il sistema rifiuterà automaticamente le sottoposizioni i cui sorgenti contengano errori rilevati in fase di compilazione.

E' inoltre richiesto di consegnare, **entro venerdì 11 febbraio 2005**, una copia cartacea della stampa del codice sorgente prodotto in portineria del DSI o nella casella di posta del docente, indicando chiaramente nome, cognome e numero di matricola di tutti i componenti del gruppo, nonchè il turno e il docente di riferimento.

6 – Valutazione

Durante la prova orale con i singoli studenti saranno discusse le modalità implementative adottate e la padronanza di alcuni dei concetti necessari per preparare il progetto e/o spiegati a lezione. La valutazione del progetto sarà fatta in base alla

- conformità dell'implementazione scelta per risolvere il problema con il paradigma di programmazione a oggetti;
- conformità del codice presentato alle regole di buona programmazione;
- adeguatezza del manuale utente presentato a descrivere il modo in cui un utente può utilizzare il programma;
- assenza di errori nel programma;
- usabilità del programma;

Dario Malchiodi
Dipartimento di Scienze dell'Informazione
Via Comelico 39/41 20135 Milano
Stanza T304 – Tel. +39 02 503 16338
eMail malchiodi@dsi.unimi.it

Walter Cazzola
Dipartimento di Informatica e Comunicazione
Via Comelico 39/41 20135 Milano
Stanza S233 – Tel. +39 0103536637
eMail cazzola@disi.unige.it