Appello di Settembre 2007 e Gennaio 2008	**Monopoli**
	Laurea Triennale in Comunicazione Digitale Laboratorio di Programmazione

1 Descrizione

Il progetto consiste nel realizzare un programma per simulare (con notevoli restrizioni) il gioco da tavolo Monopoli[1]. In particolare data una configurazione iniziale per il campo di gioco (situazione case/alberghi, posizione dei lotti, etc ...), lo stato corrente (denaro e posizione) dei giocatori ed una sequenza di lanci di dadi che condurranno i giocatori in un nuovo stato si dovrà simulare l'avanzamento dei giocatori sul campo di gioco.

2 Struttura del Campo di Gioco

Il campo di gioco è quello classico del Monopoli, riportato comunque in Fig. 1. Dal nostro punto di vista si parla di un anello composto da 40 caselle.

Le 40 caselle rappresentano i vari lotti di terreno, le quattro ferrovie ed una serie di caselle con caratteristiche particolari (le solite caselle a cui siamo abituati ad eccezione delle caselle imprevisti e probabilità). Sui lotti di terreno è possibile edificare.

La *configurazione* del campo di gioco descrive la posizione dei vari lotti (che quindi non è fissa). Le 40 caselle sono distribuite come segue:

- 24 lotti di terreno edificabile divisi in gruppi da 3 lotti non necessariamente adiacenti identificati da uno stesso colore;
  - i lotti dello stesso gruppo possono essere edificati (una casa/hotel per turno) se appartengono tutti e tre allo stesso proprietario;
  - edificare una casa costa 100€ (bisogna costruirne una per ogni lotto del gruppo ma non più di tre in totale su ogni lotto),
  - costruire un albergo costa 1000€ (uno per ogni lotto del gruppo e rimpiazzando le 3 case già costruite su ogni lotto);

[1] `http://it.wikipedia.org/wiki/Monopoli_(gioco)`

Figura 1: **Campo di Gioco del Monopoli**

– 4 ferrovie non edificabili;

– 1 prigione e 1 vai in prigione;

– 1 azienda del gas, 1 azienda dell'acqua, 1 azienda elettrica e 1 azienda telefonica;

– 2 tasse di lusso;

– 1 via e 3 terreni comunali.

Il campo di gioco è rappresentato da una singola stringa composta da 40 triple: «tipo di allotment, nome allotment, colore allotment».

- il «tipo di allotment» è identificato da un carattere: `C` per le compagnie, `R` per le stazioni, `L` per un appezzamento senza costruzioni, `1`, `2`, `3` e `H` rispettivamente per appezzamenti con 1, 2, 3 case o un hotel, `T` casella delle tasse, `G` il via, `␣` per il terreno comunale, `J` la prigione e `>` per la casella che manda in prigione.

- il «nome allotment» è una stringa che può contenere caratteri alfabetici e spazi è terminata da un $.

- il «colore allotment» è presente solo per gli appezzamenti edificabili e rappresenta il colore dell'appezzamento, i possibili colori sono: *yellow* (`y`), *cyan* (`c`), *purple* (`p`), *red* (`r`), *violet* (`v`), *orange* (`o`), *brown* (`b`), e *green* (`g`) identificati nel file dalla lettera tra parentesi[2]; **dovranno** essere realizzati tramite un enum.

[2]Si sottolinea inoltre che i codici che identificano tipi e colori sono scelti in modo da poter verificare in modo univoco se per un dato allotment siano presenti due oppure tre campi.

Ovviamente, l'ordine con cui si incontrano le triple stabilisce la posizione delle caselle sul campo di gioco a partire dalla casella posta a nord-est del campo da gioco e muovendosi in senso orario a partire da questa casella. Non necessariamente è la casella di via e soprattutto non ne gode delle proprietà. Ad inizio partita, i giocatori partiranno **SEMPRE** dalla casella di via.

I seguente è un esempio di input per il campo di gioco (per semplicità l'input va a capo dopo ogni terna, nel file non sarà così).

```
GVia$
1Vicolo Corto$p
1Vicolo Stretto$p
1Vicolo Buio$p
TTassa Patrimoniale$
RStazione Nord$
HBastioni Gran Sasso$c
CCompagnia Elettrica$
HViale Monterosa$c
HViale Vesuvio$c
JPrigione$
LVia Accademia$o
CSocietà Telefonica$
LCorso Ateneo$o
LPiazza Università$o
RStazione Ovest$
LVia Verdi$b
␣Mercato Rionale$
LCorso Raffaello$b
LPiazza Dante$b
␣Parcheggio Gratuito$
3Via Marco Polo$r
CSocietà del Gas$
3Corso Magellano$r
3Largo Colombo$r
RStazione Sud$
LViale Costantino$y
LViale Traiano$y
CSocietà Acqua Potabile$
LPiazza Giulio Cesare$y
>Vai in Prigione!$
HVia Roma$g
HCorso Impero$g
␣ASL 42$
HLargo Augusto$g
RStazione Est$
LViale dei Giardini$v
LNaviglio Ricco$v
TTassa di Lusso$
LParco della Vittoria$v
```

Nota. Non ci sono garanzie che l'input contenga il giusto quantitativo di caselle, nel-

la giusta proporzione e con i giusti raggruppamenti, sarà cura del vostro programma verificarlo.

3 Regole del Gioco

I quattro giocatori, una volta inizializzato il campo da gioco e loro stessi si muoveranno seguendo i lanci dei dadi predefiniti. Al termine di ogni singola mossa si dovrà calcolare il pedaggio dovuto al proprietario o all'erario, secondo le seguenti regole:

- il pedaggio (per i giocatori non proprietari del terreno) sul terreno è gratuito quando non edificato, 50€ se c'è una casa, 100€ con due case, 200€ con 3 case e 500€ con un albergo;
- il pedaggio (per i non proprietari) sul terreno della ferrovia è gratuito se non appartengono tutte e 4 allo stesso proprietario, altrimenti sono 100€;
- il pedaggio per le varie aziende è sempre di 50€;
- sostando sulla casella marcata come tassa di lusso si devono pagare 1000€;
- sui terreni comunali non c'è pedaggio, mentre passando dal via si ricevono 100€.

Sostando sulla casella etichettata come ≪vai in prigione≫, il giocatore finisce nella casella prigione senza passare dal via e sta fermo un turno (cioè il lancio di dati corrispondente a quel turno viene scartato). Sostando sulla casella prigione senza provenire dalla casella ≪vai in prigione≫ non ha nessun effetto particolare.

I lotti sono assegnati ad inizio partita (letti da file) e NON se ne possono acquistare durante la partita, mentre è possibile aggiungere case o sostituirle con un hotel.

4 Classi da Realizzare

È **obbligatorio** realizzare in Java il programma descritto nelle sezioni precedenti utilizzando le seguenti classi:

Allotment

`Allotment` è una classe astratta usata per descrivere l'elemento generico che compone il campo di gioco.

Attributi

- `String name`: variabile di istanza che contiene il nome del lotto, ad es., `"Via Roma"` o `"Tassa di Lusso"`.

Notare la peculiare presenza di un attributo nella classe astratta, vale sempre la regola che lo stato degli oggetti deve essere nascosto ma in questo caso deve poter essere usato dalle classi concrete che estenderanno la classe astratta.

Metodi

- **`public abstract int`** `toll()`:
  calcola il pedaggio per il giocatore che sta sostando sulla casella tenendo conto di eventuali sovrapprezzi e proprietà; se il giocatore non ha abbastanza soldi andrà in rosso.

- **`public abstract int`** `owner()`:
  restituisce un riferimento al giocatore proprietario di questo appezzamento; 0 se non è di nessuno o è dello stato o il concetto non è applicabile.

Dalla classe `Allotment` si dovranno derivare le seguenti classi organizzandole in una gerarchia di ereditarietà conveniente: `Company` (per le varie aziende), `RailwayStation` (per le ferrovie), `Tax` (per le tasse comunali di lusso), `FreeLot` (generico appezzamento), `Lot` (generico appezzamento edificabile), `LotWithAnHotel`, `LotWithAHouse`, `LotWithTwoHouses`, `LotWithThreeHouses`, `Go` (il via), `Jail` (per la prigione), `GoToJail` (la casella che ti manda in prigione).

Tenete presente che gli appezzamenti edificabili hanno anche il concetto di colore che li lega tre a tre, mentre le posizione della prigione è strettamente correlata a quella dell'elemento che "manda in prigione".

Board

Descrive il campo di gioco in termini della sua configurazione corrente.

Attributi

- `board`: variabile di istanza che memorizza elementi nel campo di gioco (cioè le istanze di `Allotment`).

- `String filename`: variabile di istanza contenente il nome del file da cui si legge la configurazione. Implementare anche i corrispondenti setter and getter: `String getFilename()` e **`void`** `setFilename(String)`.

Metodi e Costruttori

- **`public`** `Board(String filename)`:
  costruttore che legge la configurazione del campo di gioco, `filename` è il nome del file su disco contenente una serie di campi da gioco, uno per riga, l'ultima riga letta diventerà la configurazione corrente del campo di gioco.

- **`public`** `String toString()`:
  metodo che crea e ritorna una stringa rappresentante la configurazione corrente del campo di gioco, sfruttando le convenzioni descritte precedentemente;

- **`public void`** `setBoard(String)`:
  metodo che permette di inizializzare il campo di gioco alla configurazione passata come parametro, la configurazione è codificata nella stringa secondo le stesse convenzioni descritte precedentemente;

- **public** `String getBoard()`:
  metodo che ritorna la configurazione corrente codificata secondo le convenzioni descritte precedentemente;
- **public void** `write()`:
  metodo che appende la configurazione corrente della scacchiera al file corrispondente;
- **public void** `reload()`:
  metodo che rilegge dal file indicato dall'attributo `filename` la configurazione della scacchiera;
- **public** `Object clone()`:
  metodo che crea una copia dell'oggetto e la restituisce;
- **public void** `backup(String)`:
  metodo che effettua una copia di backup della configurazione del campo di gioco, salvandola in un file il cui nome è specificato nella stringa passata come argomento, il file verrà creato rimpiazzandone uno eventualmente già esistente;
- **public** `Allotment getAllotment(int i)`:
  restituisce l'appezzamento alla posizione specificata dal parametro nell'campo di gioco. Solleverà un'eccezione quando la posizione richiesta non esiste;
- **public void** `setAllotment(int i, Allotment a)`:
  cambia l'allotment in posizione i-esima con quello passato come parametro;

Player

Classe rappresentante un generico giocatore.

Attributi

- **int** `player`: variabile di istanza che identifica il giocatore e può assumere valori da 1 a 4.
- **int** `position`: variabile di istanza che conserva la posizione del giocatore all'interno del campo da gioco.
- **int**`[] tosses`: variabile di istanza che contiene i lanci di dadi per le mosse successive.
- **int** `sum`: variabile di istanza che contiene la somma di denaro a disposizione; inizialmente è inizializzata a 3000€.
- `String filename`: variabile di istanza contenente il nome del file da cui si leggono i possedimenti e i lanci del giocatore; il nome è strettamente legato al nome del file del campo di gioco, ad esempio, se il campo di gioco è memorizzato in `"test.txt"`, i file dei 4 giocatori saranno rispettivamente `"test1.txt"`, `"test2.txt"`, `"test3.txt"`, e `"test4.txt"`. Implementare anche i corrispondenti setter and getter: `String getFilename()` e **void** `setFilename(String)`.

Il file di ogni giocatore conterrà i possedimenti ed i lanci dei dadi codificati su due righe come segue:

- la prima riga riguarda i dadi, ed e codificata in: `m` $\mathtt{n}_1\mathtt{n}_2\cdots\mathtt{n}_m$ con $\mathtt{m}\in\mathbb{N}$ e $\mathtt{n}_i\in\{1,2,\cdots,6\}\ \forall i\in\{1,2,\cdots,m\}$ (non ci sono spazi tra gli $\mathtt{n}_i$, vi è invece uno spazio tra `m` e $\mathtt{n}_1$);
- la seconda riga riguarda i possedimenti che sono espressi in termini della loro posizione all'interno del campo di gioco; la codifica è: `m` $\mathtt{n}_1\mathtt{n}_2\cdots\mathtt{n}_m$ con $\mathtt{m}\in\{1,2,\cdots,32\}$ e $\mathtt{n}_i\in\{1,2,\cdots,40\}\ \forall i\in\{1,2,\cdots,m\}$ (c'è uno spazio tra ogni coppia di valori numerici successivi);

Nota. Un giocatore non può possedere il via, la prigione, la casella che manda in prigione, gli appezzamenti comunali e le caselle esattoriali, il tentare di attribuirglieli pone in una condizione di errore.

Metodi e Costruttori

- **`public`** `Player(String filename)`:
  il costruttore riceverà LO STESSO nome di file che riceve il costruttore di `Board` e calcolerà automaticamente il numero del giocatore e quindi il nome del file da usare; questo costruttore non può essere invocato più di quattro volte;
- **`public int`** `getSum()` e **`public void`** `setSum(`**`int`**`)`:
  getter e setter per l'attributo `sum`;
- **`public int`** `nextDiceToss()`:
  questo metodo ritorna il risultato del lancio del dado per il turno corrente; ovviamente il giocatore tiene traccia internamente del turno.
- **`public void`** `move(`**`int`** `toss)`:
  questo metodo sposta la pedina del giocatore di `toss` caselle (ricordiamo che il campo di gioco è un anello);
- **`public`** `String toString()`:
  questo metodo stamperà **esattamente** la stringa:

  ```
  The Player «id» is on lot «position»; it has «money» Euro.
  ```

  dove ovviamente `«id»`, `«position»` e `«money»` devono essere rimpiazzati da identità, posizione e soldi del giocatore.

Game

La classe `Game` permette di gestire una partita a Monopoli.

Attributi.

- `Board board`: rappresenta il campo di gioco;

- `Player[] players`: i giocatori in gara;
- `String filename`: il nome del file contenente la configurazione del campo di gioco;
- **`int`** `moves`: contiene il numero di lanci di dado che compongono la partita.

Costruttore e Metodi.

- **`public`** `Game(String fn)`:
  il costruttore riceve esclusivamente il nome del file cui può attingere tutte le informazioni;
- **`public void`** `makeBoard()`:
  costruisce il campo di gioco;
- **`public void`** `makePlayers()`:
  costruisce i giocatori, leggendo le relative informazioni dai file corrispondenti.
- **`public void`** `move(`**`int`** `player)`:
  muove il giocatore indicato da `player`, calcolando il pedaggio che deve pagare e pagandolo a chi di dovere (erario o altro giocatore) analogamente riscuote le proprie competenze se ne ha.
- **`public void`** `buildAHouse(`**`int`** `player, `**`int`**`[] where)`:
  fa costruire al giocatore, indicato da `player`, una nuova casa su ognuno dei tre appezzamenti specificati da `where`. Il metodo deve verificare che il giocatore sia proprietario di tutti gli appezzamenti, che questi siano parte dello stesso lotto e che effettivamente ci sia ancora spazio per costruire una casa (cioè che ci siano meno di tre case sugli appezzamenti in questione), inoltre calcola e riscuote la spesa (se il giocatore non ha abbastanza soldi si solleva un eccezione `NoMoneyException`).
- **`public void`** `buildAHotel(`**`int`** `player, `**`int`**`[] where)`:
  fa costruire al giocatore, indicato da `player`, un hotel su ognuno dei tre appezzamenti specificati da `where`. Il metodo deve verificare che il giocatore sia proprietario di tutti gli appezzamenti, che questi siano parte dello stesso lotto e che effettivamente sia possibile costruire un albergo (cioè ci siano già tre case sugli appezzamenti in questione), inoltre calcola e riscuote la spesa (se il giocatore non ha abbastanza soldi si solleva un eccezione `NoMoneyException`).

Eccezioni.

Definire in modo opportuno le eccezioni che si ritengono utili e che verranno sollevate quando si verificano condizioni di errore, sia quelle descritte precedentemente che quelle omesse (per lo più le eccezioni sono state omesse nella specifica dei metodi).

Tutte le altre eccezioni previste dall'uso di metodi Java devono filtrare ed essere gestite nel metodo `main()` anche se non espressamente indicato dalle segnature dei

metodi introdotti in questo documento.

A parte quanto espressamente richiesto, è lasciata piena libertà sull'implementazione delle singole classi e sull'eventuale introduzione di classi aggiuntive, a patto di seguire correttamente le regole del paradigma di programmazione orientata agli oggetti ed i principi di buona programmazione. Si suggerisce di porre particolare attenzione alla scelta dei modificatori relativi a variabili d'istanza e metodi, alla creazione di metodi di accesso alle variabili di istanza quando questi siano ritenuti necessari, alla definizione di classi e metodi astratti e di metodi o di variabili statici dove questi risultino opportuni, nonché alla dichiarazione e alla gestione delle eccezioni che possono venire lanciate dai vari metodi e alle relazioni di ereditarietà tra le varie classi. Si ricorda altresì di rispettare alla lettera il formato descritto per interpretare i contenuti dei file.

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

È invece espressamente richiesto di non utilizzare package non standard di Java (si possono quindi utilizzare `java.util`, `java.io` e così via) e di rispettare l'interfaccia fornita.

5 Modalità di Consegna

Il progetto deve essere svolto a gruppi di al massimo tre persone che intendono sostenere l'intero esame di Fondamenti di Architetture e Programmazione - Laboratorio di Programmazione nell'appello di **Settembre 2007** e **Gennaio 2008**, e deve essere consegnato entro mezzanotte di **domenica 23 settembre 2007**, tramite il sito di sottoposizione pubblicato all'indirizzo:

`http://homes.dsi.unimi.it/~malchiod/LP/sottoposizione`

Per sottomettere un progetto è necessario che tutti i componenti del gruppo si registrino; successivamente un componente potrà creare un gruppo a cui affiliare i rimanenti componenti. La sottoposizione è fatta a nome del gruppo e vale per tutti i suoi componenti. Va sottolineato che delle sottoposizioni successive cancellano quelle fatte in precedenza.

Dovranno essere consegnati tutti i sorgenti Java che permettano al programma di essere eseguito correttamente, compressi in un archivio di tipo ZIP **che estragga i file nella directory in cui si trova l'archivio stesso** (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.

È inoltre richiesto di consegnare, entro lunedì **24 settembre 2007**, una copia cartacea della stampa del codice sorgente prodotto in portineria del DSI indicando chiaramente nome, cognome e numero di matricola di tutti i componenti del gruppo, nonché il turno e il docente di riferimento, specificando l'appello in cui ogni singolo componente del gruppo sosterrà la discussione.

Le sottoposizioni che non seguono queste specifiche ed i programmi che non compilano correttamente non verranno valutati.

È richiesto di indicare chiaramente all'inizio di tutti i documenti consegnati nome, cognome e matricola dei vari componenti del gruppo.

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 documentazione presentata;
- assenza di errori nel programma.

Walter Cazzola
Dipartimento di Informatica e Comunicazione
Via Comelico 39/41 20135 Milano
Stanza S220 — Tel. +39.010.353.6637
e-mail: `cazzola@dico.unimi.it`

Dario Malchiodi
Dipartimento di Scienze dell'Informazione
Via Comelico 39/41 20135 Milano
Stanza S238 — Tel. +39.02.503.16338
e-mail: `malchiodi@dsi.unimi.it`