Metodologie di Programmazione 2003 – 2004

SECONDO APPELLO: 9 Febbraio 2004

Nome: _____ Matricola: _____

Istruzioni

- Scrivete il vostro nome su tutti i fogli.
- Scrivete le soluzioni nello spazio riservato a ciascun esercizio.
- Giustificate le risposte: le risposte senza giustificazione non saranno considerate.
- Tempo a disposizione 2 ore e 30.
- No libri, appunti o altro.

LASCIATE IN BIANCO:

A1	
A2	
A3	
B1	
B2	
B3	
Totale	

Nome: _____ Matricola: _____

Nome: _____ Matricola: _____

Esercizio A1 Considerate la seguente gerarchia di classi:

```
interface M { A m(); }
interface N { void n(); }
interface K { void k(); }

class A implements M, K {
   public A m() { return this; }
   public void k() {}
}
class B extends A {
   public void k() {}
}
class C implements N {
   public void n() {}
}
```

Quale è il risultato della compilazione e della (eventuale, nel caso la compilazione non dia errori) esecuzione dei seguenti frammenti? **Motivate le risposte**

1. `M a = new B(); K b = (K)(a.m());`

2. `M a = new B(); B b = (B)(a.m());`

3. `M a = new B(); B b = ((B)a).m();`

Nome: _____ Matricola: _____

Nome: _____ Matricola: _____

Esercizio A2 Considerate le seguenti classi.

```
class A {
  void show() { p((B)this); }
  void p(A x)
  { System.out.println("A"); }
}
class B extends A {
  void p(A x)
  { System.out.println("B"); }
}
```

1. Quale è il risultato della compilazione e della eventuale esecuzione delle istruzioni: `A a = new A(); a.show();`? **Motivate la risposta**

2. Quale è il risultato della compilazione e della eventuale esecuzione delle istruzioni: `A a = new B(); a.show();` ? **Motivate la risposta**

Nome: _____ Matricola: _____

Nome: _____ Matricola: _____

Esercizio A3 Data la seguente definizione:

```
interface T {  void m();  }
```

definite il corpo del seguente metodo:

```
int iterateAndApply(List alist) {
  // scorre alist invocando il metodo m() su tutti
  // gli elementi che hanno tipo T









}
```

Nome: _____ Matricola: _____

Nome: _____ Matricola: _____

Parte B In questo set di esercizi dovete costruire un sistema di classi per gestire una partita di scacchi.

La CASELLE e la SCACCHIERA La classe `Casella` definisce le posizioni sulla scacchiera. Ogni casella è caratterizzata da due coordinate (valori interi compresi tra 0 e 7), che ne individuano la posizione sulla scacchiera.

La classe `Scacchiera` gestisce una matrice (8×8) su cui sono disposti i pezzi, fornendo i seguenti metodi:

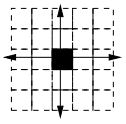
- `Pezzo get(Casella c)`: restituisce il pezzo che si trova alle coordinate corrispondenti alla casella `c`, se un tale pezzo esiste, altrimenti altrimenti `null`.
- `Pezzo mossa(Giocatore g, Casella from, Casella to) throws IllegalMove,NothingToMove:`

  Se la casella `from` non contiene alcun pezzo lancia l'eccezione `NothingToMove`. Se la mossa non è legale lancia una eccezione `IllegalMove`. Altrimenti muove il pezzo da `from` a `to` e restituisce il pezzo contenuto nella casella `to` *prima* della mossa. Il fatto che la mossa sia legale dipende dal tipo del pezzo che è oggetto della mossa e da due ulteriori condizioni: (1) il pezzo deve appartenere al giocatore, (2) la casella `to` non deve essere occupata da un altro pezzo dello stesso giocatore.
- `void demo(MoveGenerator mg)`: crea due giocatori associati alla schacchiera, uno di colore bianco, l'altro di colore nero. Esegue un loop in cui ad ogni passo `mg` viene richiesto di generare una nuova mossa; la mossa generata viene fatta eseguire alternativamente ai giocatori bianco e nero (iniziando dal bianco). Il loop termina non appena uno dei due giocatori mangia il `Re` dell'avversario. Se la mossa generata ad un passo non è legale il giocatore che ha tentato la mossa ripete il suo turno.

I PEZZI Tutti i pezzi hanno un colore, occupano una casella, ed hanno i seguenti metodi:

- `boolean appartiene(Giocatore p)`: `true` se il colore del pezzo e del giocatore coincidono
- `void muovi(Scacchiera s, Casella to) throws IllegalMove`: controlla che sia legale muovere il pezzo dalla casella corrente alla posizione `to`: se si, setta la casella corrente al valore `to`, altrimenti lancia una eccezione. Quando il metodo è invocato, la posizione `to` non contiene un pezzo dello stesso colore del pezzo corrente.

Il controllo che la mossa sia legale dipende dal tipo effettivo del pezzo. Ad esempio: nella classe `Torre`, una mossa è legale se, data la casella corrente, indicata dal quadrato nero, la casella `to` è raggiungibile lungo una delle quattro direzioni indicate nella figura a lato, e tutte le caselle intermedie tra la corrente e la casella `to` (eslusa) sono libere (non contengono alcun pezzo).



I GIOCATORI Ogni giocatore ha un colore, gioca su una scacchiera e mantiene la lista (inizialmente vuota) dei pezzi vinti nel corso di ciascuna partita. Oltre ad opportuni costruttori, la classe `Giocatore` fornisce il metodo:

- `Pezzo muovi(Casella from, Casella to)`: muove il pezzo da `from` a `to`. Se la mossa è legale e la posizione `to` è occupata da un pezzo dell'avversario, quest'ultimo viene *mangiato* e acquisito dal giocatore. Restituisce il pezzo mangiato o null, se nessun pezzo viene mangiato dalla mossa.

In tutte le classi, utilizzate i qualificatori `private` per i tutti campi, fornendo metodi *getter* e *setter* se necessario. Per i colori, utilizzate il tipo `Color` definito nel package `java.awt` ed in particolare le costanti `Color.white` e `Color.black`. Infine, assumete date le seguenti definizioni:

```
class IllegalMove extends Exception {}
class NothingToMove extends Exception {}

interface Mossa {
    public Casella from();
    public Casella to();
}
interface MoveGenerator {
    public Mossa nuovaMossa();
}
```

Nome: _____ Matricola: _____

Nome: _____ Matricola: _____

Esercizio B1 Definite le classi `Casella` e `Scacchiera`

Nome: _____ Matricola: _____

Nome: _____ Matricola: _____

Esercizio B2 Definite le classi `Pezzo` e `Torre`.

Nome: _____ Matricola: _____

Nome: _____ Matricola: _____

Esercizio B3 Definite la classe `Giocatore`

Nome: _____ Matricola: _____