Metodologie di Programmazione 2008 – 2009
I APPELLO: 30 GENNAIO 2009

Nome: _____ Matricola: _____

Istruzioni

- Scrivete il vostro nome sul primo foglio.
- Scrivete le soluzioni nello spazio riservato a ciascun esercizio.
- Gli esercizi 1 e 2 sono obbligatori.
- Gli esercizi 3 e 4 sono ispirati dalle esercitazioni e sono
  - obbligatori per chi non ha sostenuto i quiz ovvero non ha ottenuto la sufficienza,
  - facoltativi per chi volesse migliorare il risultato dei quiz.

  La consegna di questi esercizi annulla comunque il risultato conseguito nei quiz.
- Il voto è il risultato della media pesata tra il punteggio dei primi due esercizi (70%), ed il punteggio degli esercizi 3 e 4 o dei quiz (30%)
- Due turni di consegna: dopo 1,5 ore per i primi due esercizi; dopo 2,5 per tutti gli esercizi. Chi non consegna entro il primo turno perde il punteggio dei quiz e viene valutato sugli esercizi 3 e 4.
- No libri, appunti o altro.

LASCIATE IN BIANCO:

1	2	3	4	TOT

Esercizio 1

Vogliamo definire la rappresentazione di una semplice unità di elaborazione. L'architettura dell'elaboratore include un'area di memoria MEM per i dati, un insieme REG di registri di supporto alle operazioni ed un'area codice PROG dove memorizzare i programmi. L'instruction set dell'elaboratore include le seguenti operazioni elementari.

ISTRUZIONE	**EFFETTO**
LOADC i, v	REG[i] ← v
LOAD i, j	REG[i] ← MEM[j]
STORE i, j	MEM[i] ← REG[j]
STOREC i, v	MEM[i] ← v
ADD i, j	REG[i] ← REG[i] + REG[j]
SUB i, j	REG[i] ← REG[i] - REG[j]
MUL i, j	REG[i] ← REG[i] * REG[j]
DIV i, j	REG[i] ← REG[i] / REG[j]
INC i	REG[i] ← REG[i] + 1
DEC i	REG[i] ← REG[i] - 1

Assumete date le seguenti due definizioni:

```
interface Instruction<T> {
    T unit();
    void exec() throws RuntimeException;
}
class WrongOperandException extends Exception {}
class WrongInstructionException extends Exception {}
class RuntimeException extends Exception {}
```

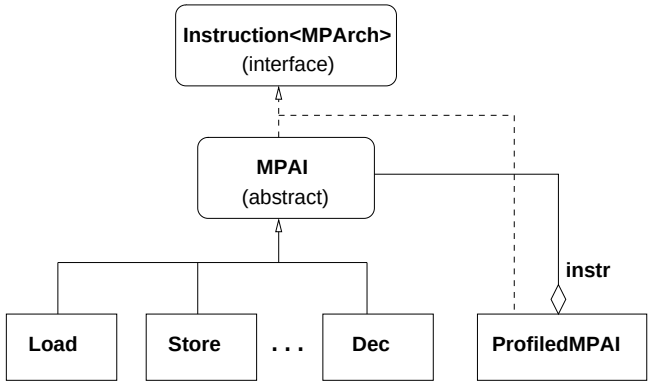
Un oggetto di tipo `Instruction<T>` rappresenta una operazione elementare, il cui effetto si ottiene invocando il metodo `exec()`. Ogni istruzione è associata ad una unità di elaborazione, cui si accede invocando il metodo `unit()`.

Definite una classe `MPArch` che realizzi l'architettura descritta sopra, attenendovi alle seguenti indicazioni:

- *Rappresentazione*
  - l'area di memoria ed i registri contengono valori `double`
  - l'area codice memorizza il programma come un lista di oggetti di tipo `Instruction<MPArch>`.
- *Costruttore*: costruisce una unità di elaborazione dimensionando l'area di memoria e l'insieme di registri come specificato dai parametri:
  ```
  public MPArch(int memsize, int regsize)
  ```
- *Istruzioni elementari*: per ciascuna istruzione la classe definisce un metodo che ne realizza l'effetto. Il metodo lancia `WronglOperandException` nel caso in cui gli argomenti che corrispondono a indirizzi nell'area di memoria e/o a indici di registro siano fuori dei rispettivi range. Implementate solamente i metodi `loadc()` e `store()` a titolo di esempio, attenendovi alle firme riportate qui di seguito:
  ```
  public void loadc(int i, double v) throws WrongOperandException
  public void store(int i, int j) throws WrongOperandException
  ```
- *Loader*: carica la lista di istruzioni nell'area codice, controllando che ciascuna istruzione sia associata all'unità `this`. In caso contrario, lancia l'eccezione.
  ```
  public void load(List<Instruction<MPArch>> prog)
                                        throws WrongInstructionException
  ```
- *Interprete*: esegue in sequenza ciascuna delle istruzioni dell'area codice. Se una delle istruzioni lancia `RuntimeExeption`, il metodo termina istantanamente senza eccezioni stampando un messaggio di errore.
  ```
  public void run()
  ```

Esercizio 2

Il seguente diagramma UML descrive una possibile implementazione delle istruzioni per l'architettura `MPArch` delineata nell'esercizio precedente.



Implementate le classi `MPAI`, `Store` e `ProfiledMPAI`, definendo campi, metodi e costruttori in modo da realizzare le funzionalità descritte qui di seguito.

La classe `MPAI`, astratta, definisce il formato generico di una istruzione, stabilendo l'associazione tra l'istruzione e l'unità di elaborazione per conto della quale sarà eseguita. Implementa i metodi dell'interfaccia `Instruction<MPArch>`, possibilmente definendoli `abstract` e definisce un ulteriore metodo pubblico `String profile()` che restituisce la rappresentazione testuale dell'istruzione.

Ciascuna delle sottoclassi `Load`, `Store`, ..., `Dec` è concreta e rappresenta una istruzione specifica. In ciascuna classe, il metodo `exec()` esegue l'istruzione, invocando il metodo corrispondente sull'unità di elaborazione associata.

La classe `ProfiledMPAI`, anch'essa concreta, realizza una versione "profiled" delle istruzioni: in tali istruzioni, l'invocazione del metodo `exec()` causa la stampa della rappresentazione testuale dell'istruzione stessa, e successivamente la sua esecuzione.

Esercizio 3

Siano date le seguenti definizioni.

```
public interface Measurable
{
   double measure();
}

public class DataStat
{
   public void add(Measurable x)
   {
     if (x == null) return;
     sum = sum + x.measure();
     if (count == 0 || max.measure() < x.measure())
        max = x;
     count++;
   }

   public Measurable max() { return max; }

   public double average() { return (count == 0) ? 0 : sum/count; }

   private double sum;
   private Measurable max;
   private int count;
}
```

Definite una sottoclasse `MoreDataStat` di `DataStat` che definisce i due metodi seguenti:

```
/**
 * @pre true
 * @post @nochange
 * @result = l'ampiezza dell'intervallo delle misure considerate,
 *  ovvero la differenza tra le misure degli elementi con misura
 *  massima e minima.
 */
public double size()

/**
 * @pre d >= 0
 * @post @nochange
 * @result = un iteratore che permette di ottenere in sequenza gli
 *  elementi la cui misura e' minore del valore d
 */
public Iterator<Measurable> lessThan(double d)
```

Esercizio 4

Considerate la seguente gerarchia di tipi:

```
interface M { M m(); }
interface N { }

class A implements M {
   public M m() { return new B(); }
}
class B extends A {
   public M m() { return new A(); }
}
class C extends A implements N {
   public M m() { return this; }
}
```

Indicate il tipo statico ed il tipo dinamico per ciascuna (sotto) espressione nei seguenti frammenti di codice.

Determinate il risultato della compilazione e, nel caso la compilazione non dia errori, dell'esecuzione.

1. `N x = new C(); M y = x.m();`

2. `M x = new A(); B y = (B)x.m();`

3. `M x = new B(); B y = (B)x.m();`