Programmazione ad Oggetti 2011 - 2012
II Esercitazione
Consegna 18 Novembre 2011

Esercizio 1: Documenti

Progettate un package `documenti` per l'impaginazione di documenti. Supponiamo che i documenti possano essere solo di due tipi – *article* e *draft* – ciascuno con un suo specifico stile di impaginazione, o *layout*. Il package include due gerarchie di classi, che realizzano le pagine ed i documenti, ed una classe applicazione.

Completate e realizzate tutte le definizioni così come richiesto nella descrizione che segue.

Pagine. Completate la seguente definizione della classe astratta `Page`, che rappresenta una pagina generica. Potete aggiungere nuovi campi e/o metodi, se necessario ad implementare le funzionalità richieste.

```
abstract class Page
{
     protected int np           // numero della pagina
     private List<String> text; // la lista dei paragrafi del testo

     /** inizializza i campi ai valori dei parametri orrispondenti */
     public Page(int np, List<String> text){ /* completare */ }

     /**
       restituisce una stringa ottenuta concatenando le stringhe del campo
       text ed aggiungendo una linea finale con il numero di pagina.
      */
     String layout() { /* completare */ }
}
```

Realizzate due sottoclassi concrete di `Page`, `ArticlePage` e `DraftPage` caratterizzate da due stili di layout. Le due classi sovrascrivono il metodo `layout()`: in `ArticlePage`, il metodo aggiunge un frontespizio (una linea che precede il testo) formato come segue: nelle pagine dispari, contiene i nomi degli autori, nelle pagine pari contiene il titolo del documento. Nella classe `DraftPage`, il metodo aggiunge il frontespizio `Draft - non diffondere`.

Documenti. La gerarchia delle classi documento ha una struttura parallela a quella delle pagine. Completate la definizione della classe astratta `Documento` che rappresenta un generico documento. Potete aggiungere nuovi campi e/o metodi, se necessario.

```
abstract class Document
{
     private List<String> authors;  // autori
     private String title;          // titolo
     private List<Page> pages;      // lista delle pagine

     /** crea una pagina con numero n e testo text */
     abstract Page createPage(int n, String text);

     /**
      costruisce un documento con authors come autori, title come titolo
      titolo ed una lista di pagine che corrisponde alla lista dei testi ts
     */
     public Document(List<String> authors, String title, List<String> ts);

     /**
      restituisce la stringa ottenuta concatenando il layout della lista di pagine
     */
     String print() { /* completare */ }
 }
```

Realizzate due classi concrete `Article` e `Draft` che estendono `Document`. Le due classi implementano il metodo `createPage()` creando, rispettivamente, la `ArticlePage` e la `DraftPage` associate alla stringa che ricevono come parametro.

Classe applicazione. Realizzate una classe applicazione `Test` per testare la vostra implementaizone.

Esercizio 2: Calcolatrice

Progettate una package `calcolatrice` per simulare il funzionamento di una semplice calcolatrice elettronica, descritto qui di seguito. Una calcolatrice per mette di eseguire semplici operazioni su valori interi ed è dotata delle seguenti strutture dati:

- un'area memoria `RAM` in cui memorizza i valori;
- un'area di memoria `PROG` in cui memorizza la sequenza di istruzioni che formano i programmi;
- quattro registri: `A` = accumula i risultati; `IP` = contiene l'indice dell'istruzione corrente; `IR` = contiene l'istruzione corrente; `S` = stato della macchina (stop o meno).

L'*instruction set* di una CE comprende le istruzioni nella tabella seguente.

Formato Simbolico	**Significato**
LOAD ind	A ← RAM[ind]
LOADC val	A ← val
STORE ind	RAM[ind] ← A
ADD ind	A ← A + RAM[ind]
MUL ind	A ← A * RAM[ind]
JUMP ind	IP ← ind
INCR	A ← A + 1
DECR	A ← A - 1
ALT	ferma l'esecuzione

Il ciclo di esecuzione è descritto dallo pseudo-codice seguente

```
procedure interpreta (start:integer)
begin
  IP ← start; S ← false
  while (not S)
    IR ← PROG[IP]
    if IR = (LOAD ind) then A ← RAM[ind]
    elsif IR = (STORE ind) then RAM[ind] ← A
    elsif ...
    ...
    elsif IR = ALT then S ← true
    endif
    if IR != (JUMP ind) then IP ← IP+1
end
```

Il package include le classi descritte nel seguito, di cui dovete implementare le funzionalità.

Calcolatrice. La classe `CE` è la classe che realizza la calcolatrice elettronica con le caratteristiche descritte in precedenza. In particolare, la classe mette definisce due metodi:

- `void exec(int start)`: simula la funzione `interpreta` definita in precedenza;
- `void trace(int start)`: simula il comportamento della funzione `interpreta`, come il metodo `exec()` ed inoltre stampa ogni istruzione dopo averla eseguita.

Istruzioni. Per realizzare le istruzioni, utilizziamo una interfaccia `instruction` che definisce il comportamento generico di tutte le istruzioni, ed una serie di classi che implementano l'interfaccia realizzando le specifiche istruzioni. L'interfaccia `instruction` fornisce due metodi:

- `boolean esegui(CE c)`: esegue l'istruzione corrente sulle strutture dati di `c`;
- `String toString()` restituisce una rappresentazione dell'istruzione, che include il nome ed i parametri.

Definite l'interfaccia `instruction` e tutte le classi che la implementano per realizzare le varie istruzioni concrete nell'instruction set della calcolatrice.

Test. Definite una classe applicazione per testare la vostra implementazione creando una `CE` e facendogli eseguire la rappresentazione del seguente programma: `(LOADC 0)::INCR::INCR::ALT`.

Esercizio 3: Iteratori

Sono date le seguenti definizioni, già viste a lezione.

```
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:

```
/**
 * restituisce l'ampiezza dell'intervallo delle misure considerate,
 * ovvero la differenza tra le misure degli elementi massimo e
 * minimo aggiunti all'insieme
 */
public double size()

/**
 * restituisce un iteratore che permette di enumerare in sequenza gli
 * elementi la cui misura e' minore del valore d
 */
public Iterator<Measurable> lessThan(double d)
```