# Metodologie di Programmazione 2007 – 2008
# SECONDO APPELLO: 4 LUGLIO 2008

**NOME COGNOME** _____

**MATRICOLA** _____

**ISTRUZIONI**

- 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:

| 1 | 2 | 3 | 4 | TOT |
|---|---|---|---|---|
| | | | | |

# Java

Considerate la seguente gerarchia di classi:

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

class A implements M {
    public M m() { return this; }
}

class B extends A {
    public void k() { }
}

class C extends A implements N {
    public void n() {}
    public void p() {}
}
```

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

**MOTIVATE LE RISPOSTE: RISPOSTE NON MOTIVATE NON SARANNO CONSIDERATE**.

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

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

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

# Datatypes

Completate la seguente definizione della classe `BiArrayList` che rappresenta una tabella bidimensionale dinamica, formata da un numero fisso di righe ed un numero variabile di colonne. Definite i campi per realizzare la rappresentazione interna della classe nei modi che ritenete più opportuni, ed implementate i metodi in accordo alla vostra scelta, rispettando la specifica data.

```
/**
 * Una matrice bi-dimensionale di oggetti di tipo T con un numero fissato
 * di righe ed un numero variabile di colonne (tutte le colonne hanno lo
 * stesso numero di posizioni).
 */
class BiArrayList<T>
{
    /**
     * @pre : n > 0
     * @post: crea una BiArrayList vuota, con n righe e 0 colonne
     */
    public BiArrayList(int n)

    /**
     * @pre: TRUE, @result = numero di righe di this
     */
    public int rows()

    /**
     * @pre: TRUE, @result = numero di colonne di this
     */
    public int cols()

    /**
     * @pre: c != null  && c.length == rows()
     * @post: aggiunge c come ultima colonna di this.
     *   Lancia l'eccezione se la precondizione e` violata
     */
    public void insert (T[] c) throws IllegalArgumentException

    /**
     * @pre:  x != null & 0 <= i < rows() && 0 <= j < cols()
     * @post: modifica la posizione (i,j) della matrice settandola
     *        ad x. Eccezione se la precondizione e` violata
     */
    public void set(int i, int j, T x) throws IllegalArgumentException

    /**
     * @pre:  0 <= i < rows() && 0 <= j < cols()
     * @result = il valore memorizzato in posizione (i,j)
     * @post: Eccezione se la precondizione e` violata
     */
    public T get(int i, int j)  throws IllegalArgumentException

    /**
     * @result = un iteratore che restituisce gli elementi di this
     *    enumerandoli in ordine di riga e di colonna
     */
    public Iterator<T> iterator()
}
```

# Progetto

Sia data la seguente classe per rappresentare un buffer di $n$ posizioni.

```
class Buffer {
    private char[] contents;

    public Buffer(int n)
    {
      if (n <= 0) throw new ArrayOutOfBoundsException();
      contents = new char[n];
    }

    public void set(int i, char c) throws ArrayIndexOutOfBoundsException
    {
      contents[i] = c;
    }

    public char get(int i) throws ArrayIndexOutOfBoundsException
    {
      return contents[i];
    }
}
```

Definite una sottoclasse `UndoBuffer` di `Buffer` che fornisca un metodo `undo()` che permetta di annullare l'effetto delle operazioni di `set` sulle posizioni del buffer. Il metodo `undo()` si comporta come l'operazione di *annulla* disponibile in un qualunque editor, ovvero: annulla l'effetto dell'ultima operazione di `set()` che non sia stata già annullata; se non ci sono `set()` da annullare, `undo()` non ha alcun effetto.

Esempio:

```
// OPERAZIONE           // STATO DEL BUFFER (_ = carattere nullo)
b = new UndoBuffer(4);  // _::_::_::_
b.set(1,'a');           // _::a::_::_
b.set(1,'b');           // _::b::_::_
b.undo();               // _::a::_::_
b.set(3,'c');           // _::a::_::c
b.undo();               // _::a::_::_
b.undo();               // _::_::_::_
b.undo();               // _::_::_::_
```

# Esercitazioni

Vogliamo definire una gerarchia di classi e interfacce per rappresentare e valutare espressioni booleane con una struttura definita dalla seguente sintassi.

$$B ::= \texttt{TRUE} \mid \texttt{FALSE} \mid \texttt{IF}\ B\ \texttt{THEN}\ B\ \texttt{ELSE}\ B$$

Sia data la seguente interfaccia:

```
interface BoolExp
{
    /**
     * @result = il risultato della valutazione di this
     * @post: nochange
     */
     boolean double eval()

    /**
     * @result = la stringa che rappresenta this
     * @post: nochange
     */
     String toString()}
}
```

1. Realizzate:
   - una classe `Const` che implementa `BoolExp` e rappresenta una costante di tipo booleano
   - una classe `Cond` che implementa `BoolExp` e rappresenta una expressione condizionale della forma *if-then-else*.

   In entrambe le classi, la specifica dei metodi `eval()` e `toString()` è quella definita nell'interfaccia `BoolExp`.
2. Descrivete la sequenza di istruzioni che costruiscono la rappresentazione della espressione seguente, dove `a` e `b` sono (la rappresentazione di) arbitrarie espressioni booleane.

   ```
   if a then (if b then FALSE else TRUE) else FALSE
   ```