COGNOME: **NOME:** **MATRICOLA:**

Programmazione II, 27 settembre 2011 – Laurea in INFORMATICA

Esercizio 1[16 punti]

Una *linked list doppia* è una raccolta di oggetti, o *nodi*, ognuno dei quali contiene un riferimento ad un elemento, un riferimento al nodo precedente e un riferimento al nodo successivo. Realizzare una classe `DList` che implementa una linked list doppia i cui nodi sono oggetti della classe `Nodo` definita come segue:

```
public class Nodo {
    protected String element;
    protected Nodo next, prev;

    /**costruttore che crea un nodo con campi assegnati */

   public Nodo(String e, Nodo p, Nodo n) {
        element = e;
        prev = p;
        next = n;
    }

   /**restituisce l'elemento nel nodo corrente */

    public String getElement() {
        return element;
    }

   /**restituisce il nodo precedente al nodo corrente */

    public Nodo getPrev() {
        return prev;
    }

   /**restituisce il nodo successivo al nodo corrente */

    public Nodo getNext() {
        return next;
    }

    public void setElement(String newElem){element = newElem;}
    public void setPrev(Nodo newPrev){prev = newPrev;}
    public void setNext(Nodo newNext){next = newNext;}
}
```

La classe `DList` dovrà contenere

- due variabili di istanza protected, `head` e `trail`, di tipo `Nodo`, che rappresentano l'inizio e a fine della lista;
- una variabile di istanza protected, `size`, di tipo `int` che contiene il numero di nodi presenti nella lista (inizialmente il suo valore è zero per indicare che la lista è vuota);
- un costruttore senza parametri che crea una lista vuota; la lista vuota è rappresentata da una lista in cui head punta a un nodo il cui campo `next` contiene `trail` e gli altri due campi `null`, mentre `trail` punta a un nodo il cui campo `prev` punta a `head` e gli altri due campi contengono `null`.

Nota che in un oggetto `DList` il primo elemento della lista è quello immediatamente dopo `head`, mentre l'ultimo elemento è quello immediatamente prima di `trail`.
La classe dovrà implementare la seguente interfaccia:

```
public interface DListInt {

    /** Restituisce il numero di elementi nella lista */
    public int size();

    /** Restituisce un valore che dice se la lista e' vuota */
    public boolean isEmpty();

    /** Restituisce il primo nodo della lista e lancia
     * un'eccezione del tipo specificato se la lista e' vuota */
    public Nodo getFirst() throws IllegalStateException;

    /** Restituisce l'ultimo nodo della lista e lancia
     * un'eccezione del tipo specificato se la lista e' vuota */
    public Nodo getLast() throws IllegalStateException;

   /** Restituisce il nodo che precede il nodo v e lancia
     * un'eccezione del tipo specificato se v e' l'header */
    public Nodo getPrev(Nodo v) throws IllegalArgumentException;

   /** Restituisce il nodo che segue il nodo v e lancia
     * un'eccezione del tipo specificato se v e' il trailer */
    public Nodo getNext(Nodo v) throws IllegalArgumentException;

    /** Inserisce il nodo v all'inizio della lista e incrementa
     * il numero di elementi della lista */
    public void addFirst(Nodo v);

    /** Inserisce il nodo v alla fine della lista e incrementa
     * di uno il numero di elementi della lista */
    public void addLast(Nodo v);

    /** Rimuove il nodo v dalla lista e decrementa
     * di uno il numero di elementi della lista */
    public void remove(Nodo v);
}
```

In caso di lista vuota, i metodi `getFirst` e `getLast` devono lanciare un'eccezione di tipo `IllegalStateException`, una sottoclasse di `java.lang.RuntimeException`. Si usi il costruttore

```
public IllegalStateException(String s)
```

per specificare un opportuno messaggio di errore.
Inoltre i metodi `getPrev` e `getNext` devono lanciare un'eccezione di tipo `IllegalArgumentException`, una sottoclasse di `java.lang.RuntimeException`, quando l'argomento `v` è `head` nel metodo `getPrev` e `trail` nel metodo `getNext`. Si usi il costruttore

```
public IllegalArgumentException(String s)
```

per specificare un opportuno messaggio di errore.

Esercizio 2[16 punti] La seguente classe astratta definisce oggetti che possono essere 'disegnati' su schermo usando caratteri e il carattere usato viene memorizzato in un attributo `c`:

```
abstract class DisegnabileConCarattere {
    protected char c = '*';

   public void impostaCarattere(char c) {
       this.c = c;
    }

   protected char carattere() {
        return this.c;
    }

    abstract String disegna();
}
```

(a) Definire una classe astratta `Segmento` che estende `DisegnabileConCarattere` e inoltre contiene:

- una variabile protected `lunghezza` di tipo `int` (lunghezza del segmento),
- un metodo `String toString()` che restituisce la stringa ottenuta invocando il metodo `disegna`.

(b) Definire due classi `SegmentoOrizzontale` e `SegmentoVerticale` che estendono `Segmento` definendo il metodo astratto ereditato attraverso la gerarchia. Ciascuna delle due classi contiene un costruttore con un parametro intero usato per inizializzare la variabile `lunghezza`.

(c) Scrivere infine un programma `UsaSegmento` che crea un segmento orizzontale lungo 6 e un segmento verticale lungo 4 e li stampa.