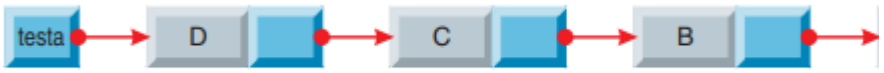
Esercitazione N°11 Gestione di una pila in memoria dinamica

In questa esercitazione vogliamo realizzare una applicazione che implementi il funzionamento di una pila In memoria dinamica. Questo significa che non avremo una struttura predefinita come nell'esempio dei vettori ma una struttura basata su un elemento nodo che saranno collegati dinamicamente nella memoria man mano che saranno inseriti nuovi dati:



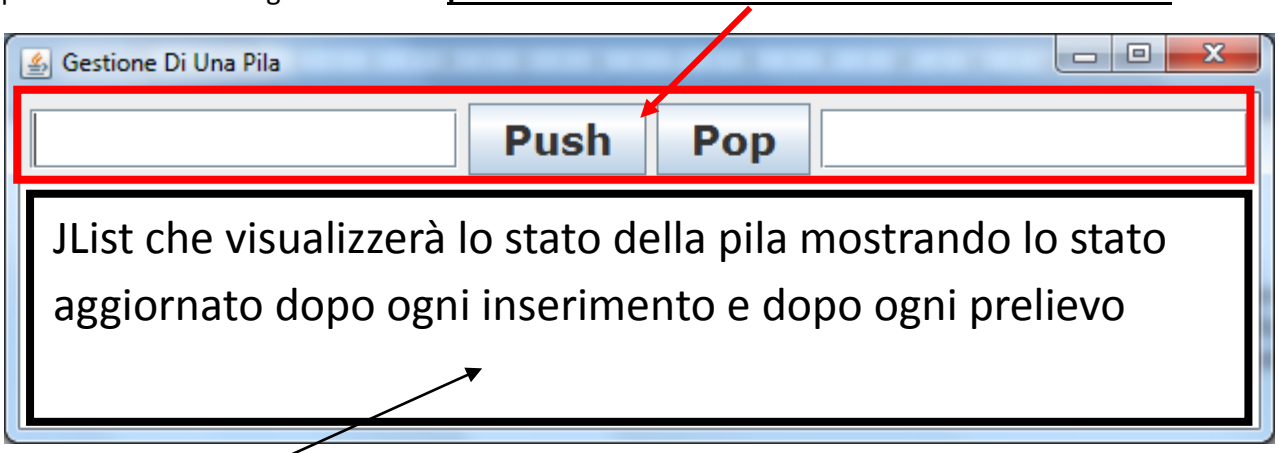
Ovviamente testa all'inizio punterà a null;
La struttura di un nodo la implementeremo con una classe così strutturata:

```
public class Nodo {
      String dato;
      Nodo ind;

      public Nodo(String s){

            dato=s;
            ind=null;
      }
}
```

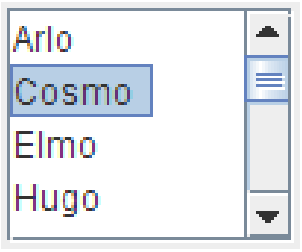
Mentre la classe pila sarà quella di seguito dove il costruttore si limita ad impostare testa a null e i due metodi push e pop modificano testa in base al nuovo inserimento o al prelievo dell'elemento :

```
package pilaDinamica;
import javax.swing.*;
public class Pila {
   Nodo testa;
    public Pila(){
     testa=null;
        }

 public void push(String s){
    Nodo nuovo;
    nuovo= new Nodo(s);
    if(testa==null)
        testa=nuovo;
    else{
    nuovo.ind=testa;
    testa=nuovo;
    }
}

public String pop(){
    String l="";
    if(testa==null)
        JOptionPane.showMessageDialog(null, "La pila è vuota");
    else{
        l=testa.dato;
        testa=testa.ind;

        }
    return l;
}
}
```

La costruzione della finestra sarà fatta utilizzando come LayoutManager il border Layout e posizionando nella regione nord un **pannello che contiene le due JTextField e i due JButton**



e nalla Regione Centro un JList



Una JList è un componente in grado di visualizzare un elenco di oggetti. All'atto della sua dichiarazione viene specificato il tipo di oggetto che visualizzerà come elemento di lista. Se il numero di elementi contenuti supera la sua dimensione mostra automaticamente una barra di scorrimento. La dichiarazione che noi andremo ad effettuare nel codice della finestra Sarà:

```
private JList<String> lstPila;
```

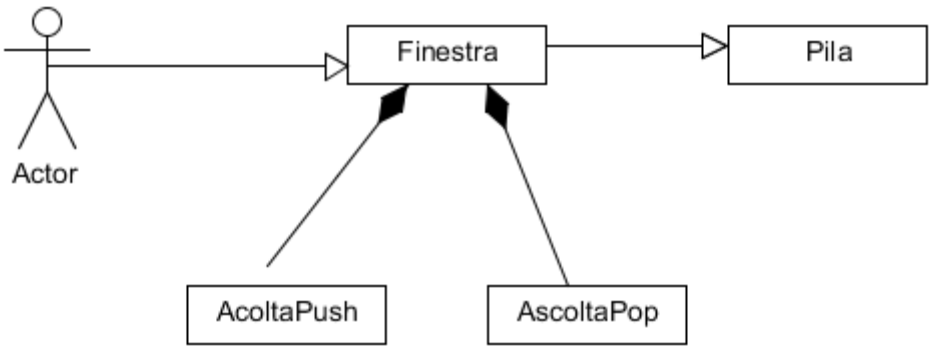
in pratica tra **< > viene** dichiarato il tipo di oggetto e lstPila è la nostra variabile di tipo JList.
Il costruttore della JList è il seguente:

list = new JList(dati); dove data è il vettore di oggetti del tipo dichiarato

ed in ogni istante si può aggiornare il vettore di oggetti in essa contenuto invocando il metodo:

```
lstPila.setListData(dati);
```

Lo schema delle Classi della nostra applicazione dovrebbe essere il seguente:



In pratica abbiamo la classe finestra, che costituisce l'interfaccia grafica con cui interagisce l'utente, che contiene le due sottoclassi per la gestione degli eventi sui due bottoni push e pop, e che usa la classe Pila che abbiamo già visto.

Quindi nel codice della Finestra la dichiarazione delle variabili sarà:

```
import java.awt.BorderLayout;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;
 public class Finestra extends JFrame{
    private JButton btnPush;
    private JButton btnPop;
    private JTextField txfIn;
    private JTextField txfOut;
    private JPanel pnlUsaPila;
    private JList<String> lstPila;
    private Pila pila;
    private DefaultListModel<String> listModel = null;
```

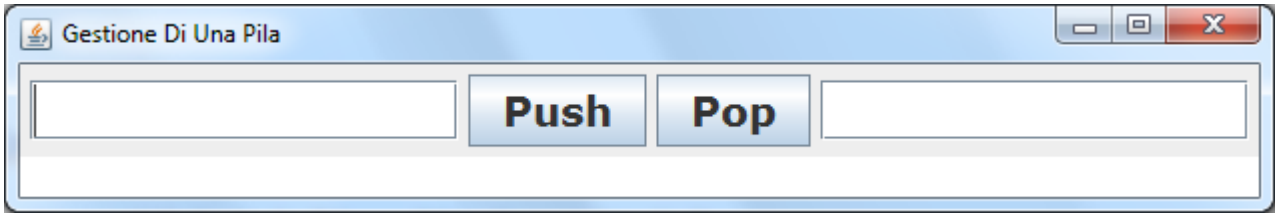
E il costruttore:

```
 public Finestra(String titolo)
 {
     pila = new Pila();
     inizializza();
     this.setVisible(true);
 }
```

La funzione Inizializza:

```
 private void inizializza()
 {
     this.setDefaultCloseOperation(EXIT_ON_CLOSE);
     this.getContentPane().setLayout(new BorderLayout());
     pnlUsaPila= new JPanel();
     btnPush = new JButton("Push");
     btnPush.setFont(new Font("Verdana",Font.BOLD,20));
     btnPush.addActionListener(new btnPush_Click() );
     btnPop = new JButton("Pop");
     btnPop.setFont(new Font("Verdana",Font.BOLD,20));
     btnPop.addActionListener(new btnPop_Click() );
     txfIn= new JTextField(10);
     txfIn.setFont(new Font("Verdana",Font.BOLD,20));
     txfOut= new JTextField(10);
     txfOut.setFont(new Font("Verdana",Font.BOLD,20));
     pnlUsaPila.add(txfIn);
     pnlUsaPila.add(btnPush);
     pnlUsaPila.add(btnPop);
     pnlUsaPila.add(txfOut);

     listModel = new DefaultListModel<String>();
     lstPila = new JList<String>(listModel);

     lstPila.setFont(new Font("Verdana",Font.BOLD,20));
     this.add(pnlUsaPila, BorderLayout.NORTH);
     this.add(lstPila, BorderLayout.CENTER);
     this.pack();
 }
```

Dove l'istanza della JList viene fatta mediante il costruttore che riceve anche ilListModel che utilizzeremo per modificare il contenuto da visualizzare ad ogni inserimento. Essendo il ListModel all'inizio vuoto la finestra si presenta all'apertura in questo modo:



Ed in seguito a inserimenti automaticamente si allarga e si restringe,grazie al codice delle due classi ascoltatrici

```
/*
 * Classe che realizza l'ascoltatore dell'evento click sul bottone Push
 * implementando il metodo actionPerformed
 */
private class btnPush_Click implements ActionListener{

    public void actionPerformed(ActionEvent e){
        pila.push(txfIn.getText());
        Nodo n=pila.testa;
        listModel.clear();
        while(n!=null){
            listModel.addElement(n.dato);
            n=n.ind;

        }
        ridimensiona();
    }
}

 /*
     * Classe che realizza l'ascoltatore dell'evento click sul bottone Pop
     * implementando il metodo actionPerformed
     */
    private class btnPop_Click implements ActionListener{

        public void actionPerformed(ActionEvent e){

            txfOut.setText(pila.pop());
            Nodo n=pila.testa;
            listModel.clear();
            while(n!=null){
                listModel.addElement(n.dato);
                n=n.ind;

            }
            ridimensiona();
        }
    }
```

Ed in fondo alla classe Finestra aggiungiamo il metodo che la ridimensiona invocando il metodo thi.pack();

```
public void ridimensiona(){
    this.pack();
}
```