File ad accesso diretto funzioni Hash Parte3

Prof. Francesco Accarino
IIS Altiero Spinelli Sesto San Giovanni

Sviluppo delle altre funzionalità

La prima funzione che ci verrebbe in mente di implementare ovviamente sarebbe la funzione visualizza. A questo proposito se proviamo a immaginare cosa dovrebbe fare questa funzione.
Essa dovrebbe scorre tutto l'archivio e restituirci tutti i record Pieni saltando quelli vuoti. Quindi sostanzialmente dovrebbe restituirci la lista di tutti i record pieni
In C# come in java esiste l'oggetto List così definito:

Una lista è un oggetto che contiene variabili in un ordine specifico. Il tipo di variabile che la lista può memorizzare è definita utilizzando la sintassi generica. Ecco un esempio di definizione di un elenco denominato numeri interi che contiene numeri:

```
List <int> numeri = new List <int> ();
```

Quindi nel nostro caso la definizione sarebbe:

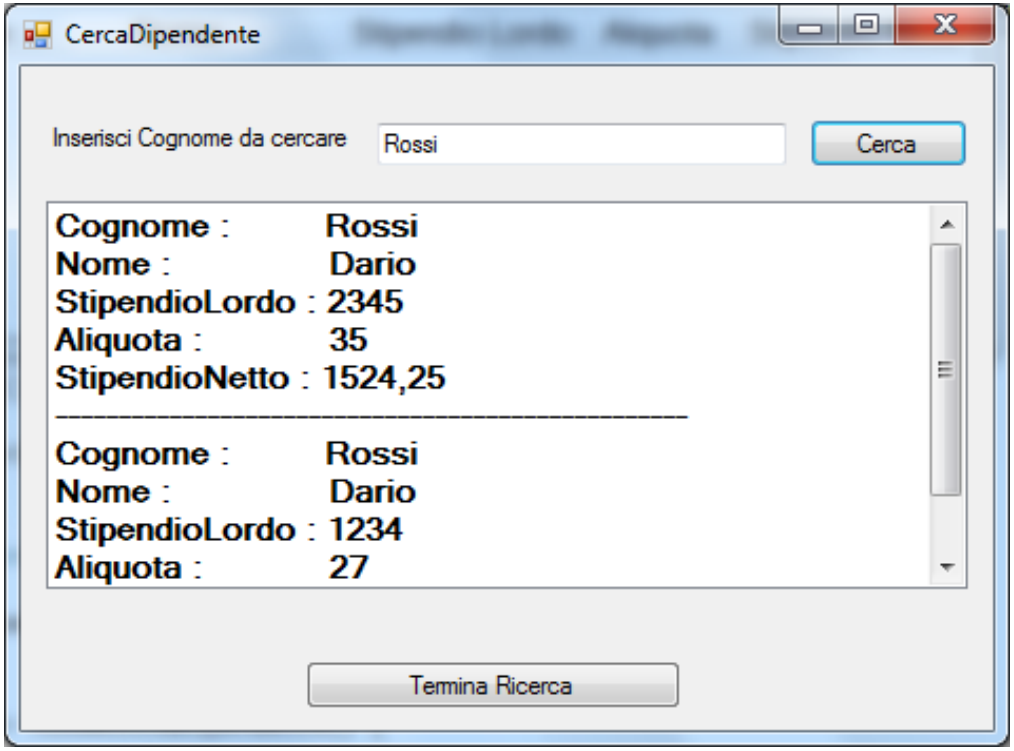
```
List<Dipendente> elenco = new List<Dipendente>();
```

Oggetto List Metodi più importanti

	Name	Description
	Add	Adds an object to the end of the List<T>.
	AddRange	Adds the elements of the specified collection to the end of the List<T>.
	Clear	Removes all elements from the List<T>.
	Insert	Inserts an element into the List<T> at the specified index.
	Remove	Removes the first occurrence of a specific object from the List<T>.
	RemoveAll	Removes all the elements that match the conditions defined by the specified predicate.
	Sort()	Sorts the elements in the entire List<T> using the default comparer.

Sviluppo delle altre funzionalità

Se poi pensiamo alla funzione cerca ipotizzando per esempio di voler implementare una interfaccia come la seguente in cui l'utente inserisce semplicemente un cognome, è evidente che anche in questo caso avremmo ancora una volta bisogno di una funzione che ci restituisca una lista di tutti i dipendenti con quel cognome.



Sviluppo delle altre funzionalità

Quindi possiamo immaginare di creare un metodo interno alla classe Dipendenti che ciameremo Cerca da utilizzare sia per la funzione di visualizzazione che di ricerca e riceverà come parametri la stringa cognome e un booleano cerca. Le due funzioni le possiamo quindi implementare come segue:

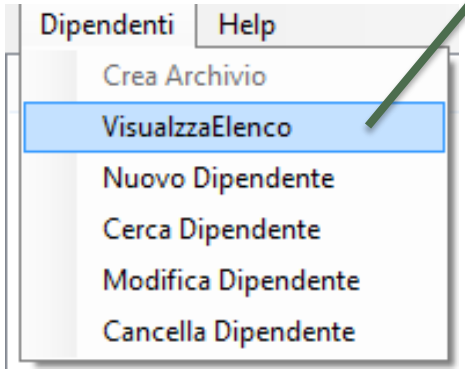
```
public List<Dipendente> CercaDipendente(string cognomeDaCercare)
{
    return cerca(cognomeDaCercare, true);
}

public List<Dipendente> visualizzaDipendenti() {

    return cerca(null, false);
}
```

Sviluppo delle altre funzionalità la funzione cerca

```
private List<Dipendente> cerca(string cognomeDaCercare, bool cerca)
   {
       long letti = 0,pos;
       List<Dipendente> elenco = new List<Dipendente>();
       fs = new FileStream(archivio , FileMode.Open,FileAccess.Read);
       br = new BinaryReader(fs);

       while (letti < numRecInseriti)
       {

           while (fs.ReadByte() == Convert.ToByte('-'))
               fs.Seek(54, SeekOrigin.Current);

           fs.Seek(-1, SeekOrigin.Current);
            pos=fs.Position;
           Dipendente letto = new Dipendente(br.ReadString().Trim(), br.ReadString().Trim(), br.ReadDouble(),pos);

           br.ReadInt32();
           br.ReadDouble();
           letti++;
           if (cerca)
           {
               if (letto.getCognome() == cognomeDaCercare)
                   elenco.Add(letto);
           }
           else
               elenco.Add(letto);
       }
       fs.Close();
       br.Close();
       return elenco;
   }
```

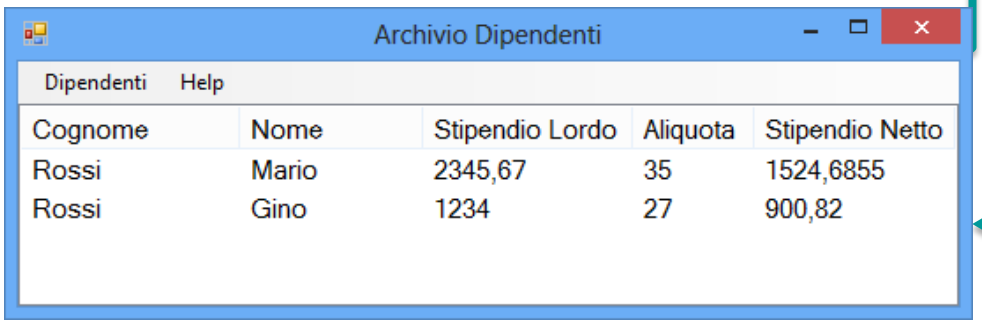
Notare che nella costruzione di un dipendente viene anche salvata la posizione questo è utile perché ci permetterà di accedere direttamente ad esso quando faremo le operazioni di modifica e cancellazione selezionando il record dalla listview

Visualizzazione




```
public void visualizzaElencoToolStripMenuItem_Click(object sender, EventArgs e)
{
    ListViewItem riga; //oggetto della listview per creare le righedella listview
    List<Dipendente> elenco; //arraylist di oggetti di tipo dipendente

    if (mioArchivio.getNumRec() == 0)//se l'archivio è vuoto messaggio
        MessageBox.Show("L'archivio è vuoto");
    else
    {//altrimenti chiamo la funzione visualizzaDipendenti che mi restituisce la lista
        elenco = mioArchivio.visualizzaDipendenti();
        resettaListaView();//funzione con la quale resettiamo la listview

        foreach (Dipendente trovato in elenco)
        { //ciclo iterativo ad oggetti. in pratica per ogni oggetto di tipo dipendente
          //contenuto nell'arraylist elenco si assegna il suo indirizzo a trovato e con esso
          // si aggiorna la listview
            riga = new ListViewItem(trovato.getCognome());
            riepilogo.Items.Add(riga);
            riga.SubItems.Add(trovato.getNome());
            riga.SubItems.Add(Convert.ToString(trovato.getStipendio()));
            riga.SubItems.Add(Convert.ToString(trovato.getAliquota()));
            riga.SubItems.Add(Convert.ToString(trovato.getStipendioNetto()));
            riga.SubItems.Add(Convert.ToString(trovato.getPosizione()));
        }
    }
}
```

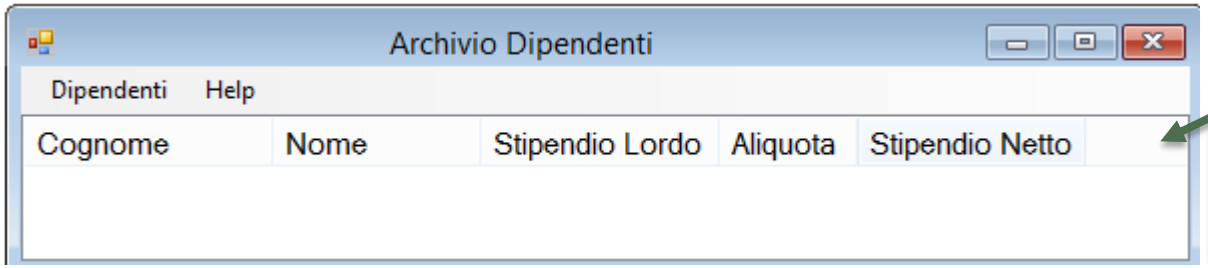
Notarre che nella listview aggiungiamo il campo posizione ma esso è nascosto. Questo si ottiene impostando la dimensione a 0.



Diamo un'occhiata alla funzione resettaListaView

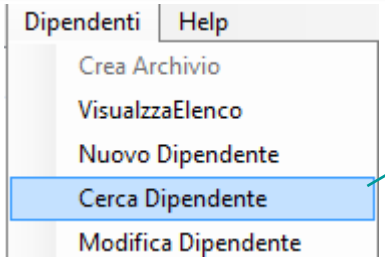
```
private void resettaListaView() {
    riepilogo.Clear();
    riepilogo.Columns.Add("Cognome");
    riepilogo.Columns.Add("Nome");
    riepilogo.Columns.Add("Stipendio Lordo");
    riepilogo.Columns.Add("Aliquota");
    riepilogo.Columns.Add("Stipendio Netto");
    riepilogo.Columns.Add("");
    riepilogo.Columns[0].Width = 124;
    riepilogo.Columns[1].Width = 104;
    riepilogo.Columns[2].Width = 119;
    riepilogo.Columns[3].Width = 69;
    riepilogo.Columns[4].Width = 114;
    riepilogo.Columns[5].Width = 0;
    riepilogo.View = View.Details;
}
```

Se guardiamo il codice notiamo che alla listview è stato aggiunta una colonna nascosta nella quale sarà memorizzata la posizione di ciascun record contenuto nell'archivio. Questo sotterfugio ci semplifica di molto le operazioni di cancellazione e modifica perché come abbiamo già visto nelle precedenti esperienze l'utente selezionerà un' intera riga e quindi grazie al valore del campo nascosto contenuto in essa possiamo trovare automaticamente la posizione del record nell'archivio.



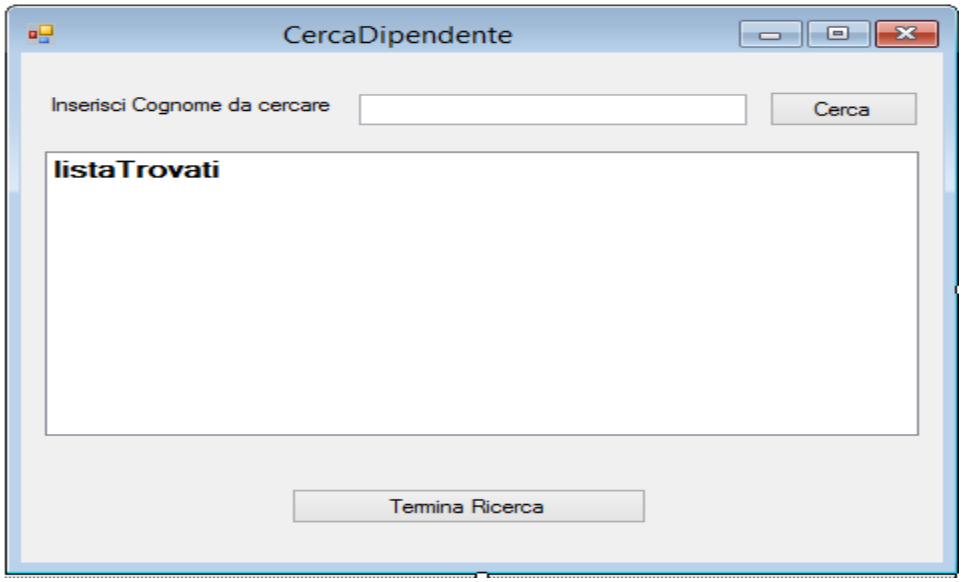
Campo nascosto

Cerca dipendente



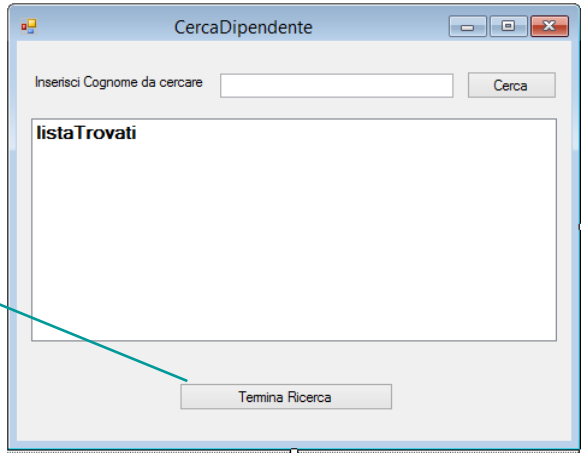

```
private void cercaDipendenteToolStripMenuItem_Click(object sender, EventArgs e)
{
    CercaDipendente cd = new CercaDipendente(mioArchivio);
    cd.Show();
}
```

Come prima cosa ci costruiamo un form di nome Cerca Dipendente come quello mostrato in figura. La lista di visualizzazione dei trovati è una ListBox di nome lista. Anche per questa classe modificheremo il costruttore per passargli l'oggetto Dipendenti




```
public partial class CercaDipendente : Form
{
    Dipendenti dp;
    public CercaDipendente(Dipendenti d)
    {
        dp = d;
        InitializeComponent();

    }
```

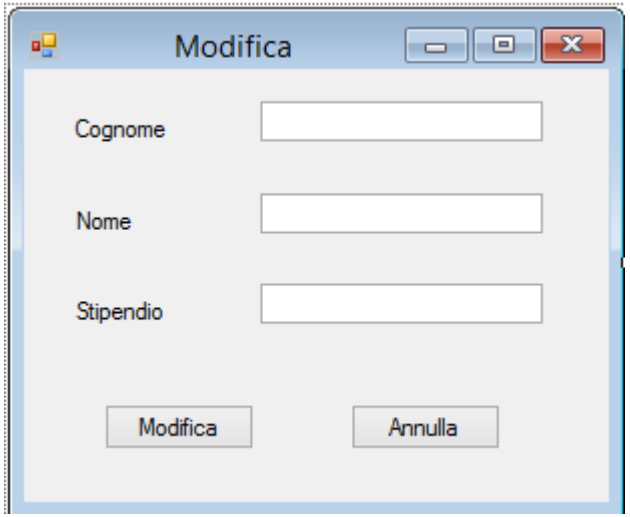
Cerca dipendente




```
private void cercaBtn_Click(object sender, EventArgs e)
{
    List<Dipendente> trovati;

    trovati = dp.CercaDipendente(cognome.Text);
    if (trovati.Count != 0)
    {
        listaTrovati.Items.Clear();
        foreach (Dipendente trovato in trovati)
        {
            listaTrovati.Items.Add("Cognome : ".PadRight(18,' ') + trovato.getCognome());
            listaTrovati.Items.Add("Nome : ".PadRight(21, ' ') + trovato.getNome());
            listaTrovati.Items.Add("StipendioLordo : " + trovato.getStipendio());
            listaTrovati.Items.Add("Aliquota : ".PadRight(22, ' ') + trovato.getAliquota());
            listaTrovati.Items.Add("StipendioNetto : "+trovato.getStipendioNetto());
            listaTrovati.Items.Add(new String('-', 50));
        }

    }
    else MessageBox.Show("Cognome inesistente");
}
```

Modifica dipendente

Per prima cosa ci creiamo un nuovo Form di nome Modifica come quello qui afianco:



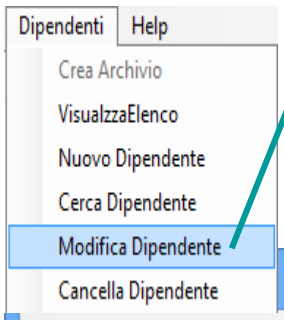
E poi modifichiamo il codice del costruttore come segue:

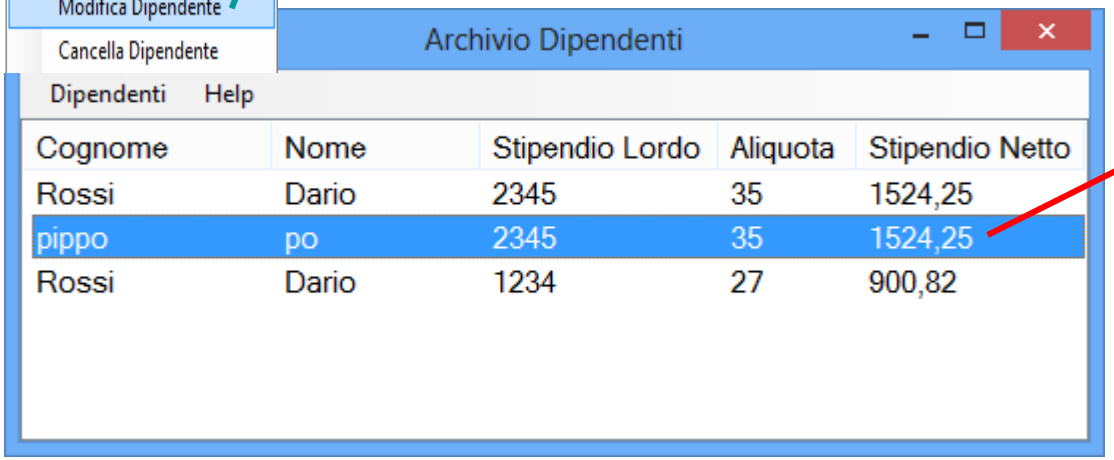
```
public partial class Modifica : Form
{
     ListView list;
      public Modifica(ListView lista) {
          list = lista;
           InitializeComponent();

    this.cognome.Text = lista.SelectedItems[0].Text.Trim();
    this.nome.Text = lista.SelectedItems[0].SubItems[1].Text.Trim();
    this.stipendio.Text = lista.SelectedItems[0].SubItems[2].Text;


    }
```

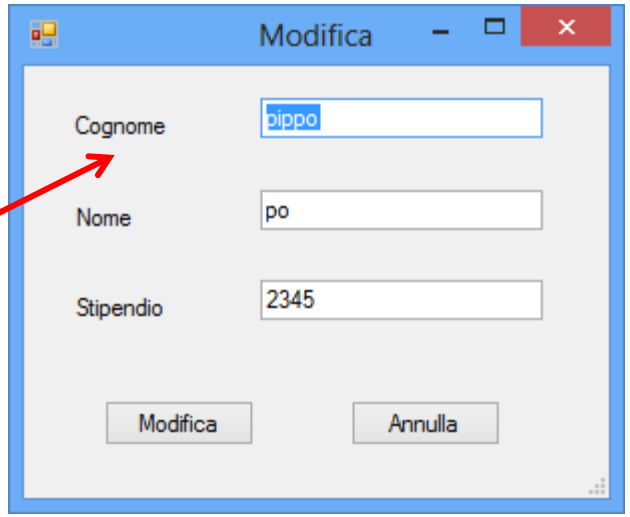
In pratica il costruttore riceve la ListView in modo da poter visualizzare le informazioni del dipendente da modificare:

Modifica dipendente

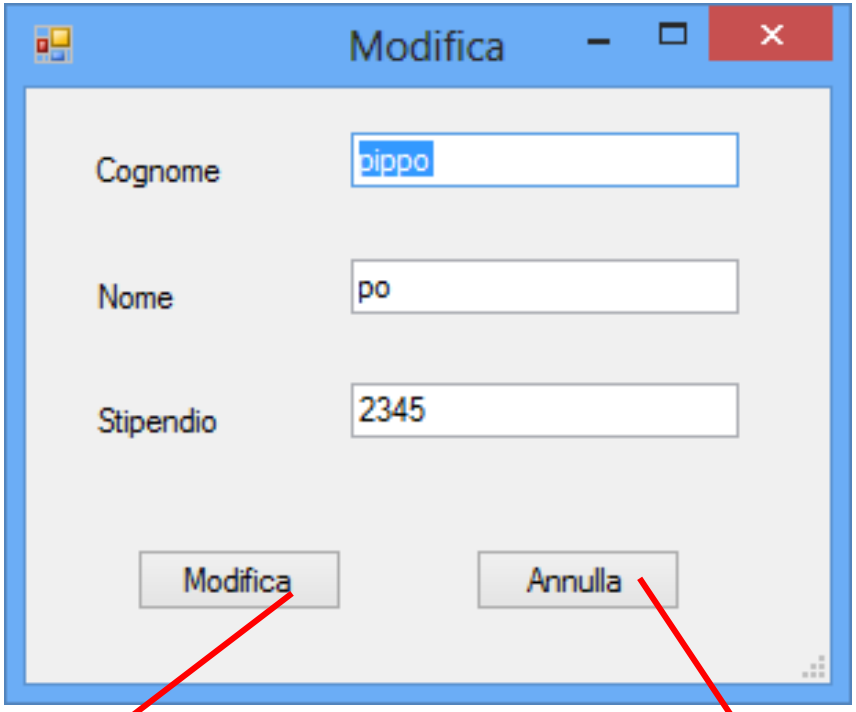
```
private void modificaDipendenteToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (riepilogo.SelectedItems.Count == 0) MessageBox.Show("devi selezionare un dipendente da modificare");
    else {
        Modifica frm = new Modifica(riepilogo);
        if (frm.ShowDialog() == DialogResult.OK) {
            string pos=riepilogo.SelectedItems[0].SubItems[5].Text;
            Dipendente modificato = new Dipendente(frm.cognome.Text, frm.nome.Text,
                                                   Convert.ToDouble(frm.stipendio.Text),Convert.ToInt64(pos));
            mioArchivio.modifica(modificato);

            visualizzaElencoToolStripMenuItem_Click(this, null);
        }

    }
}
```







Modifica dipendente




```
private void modificaBtn_Click(object sender, EventArgs e)
{

    this.Close();

}
```

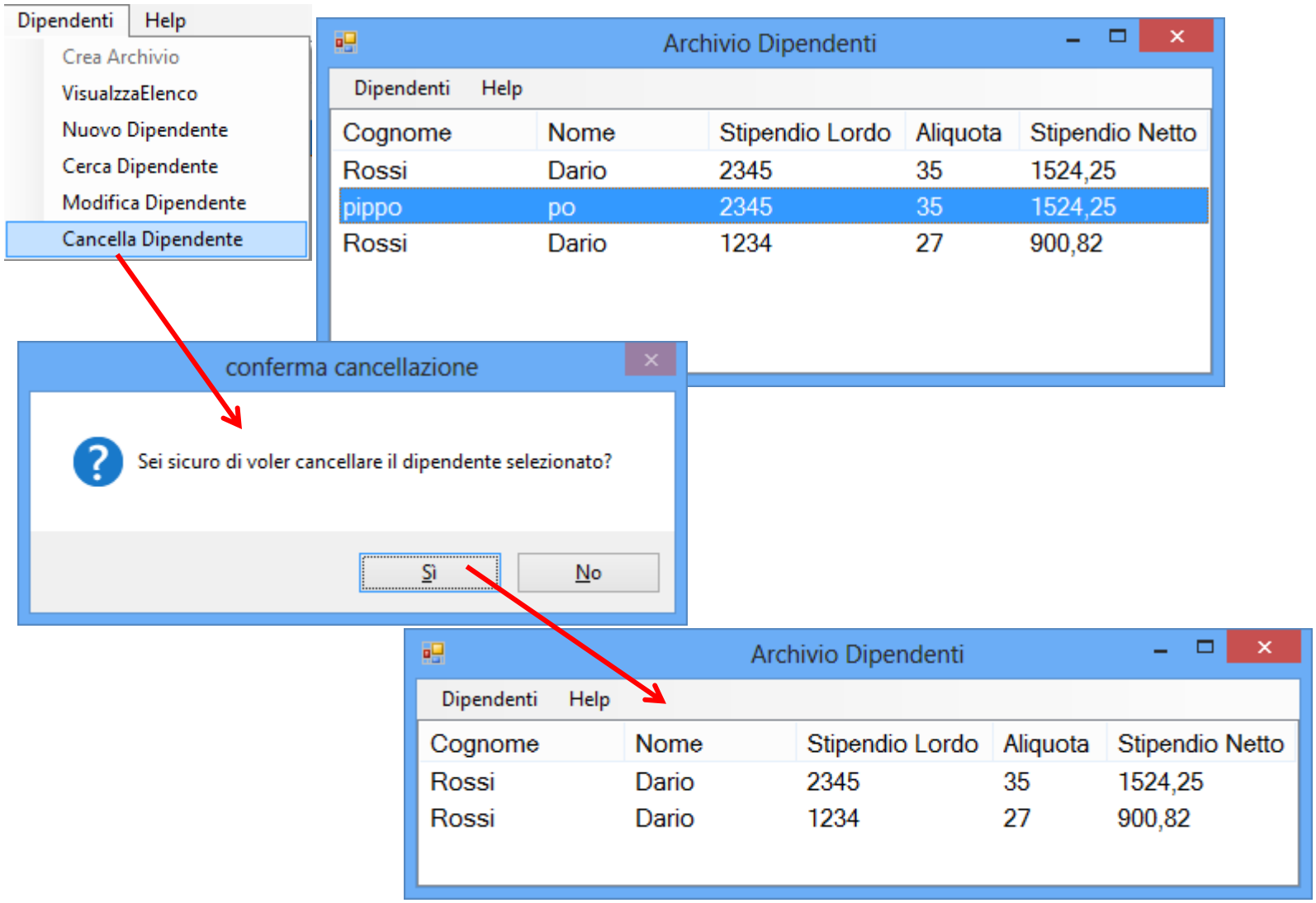
```
private void annullaBtn_Click(object sender, EventArgs e)
{
    this.Close();
}
```

Modifica dipendente Metodo Modifica Dipendente di Dipendenti

```
public void modifica(Dipendente d){

    fs = new FileStream(archivio, FileMode.Open, FileAccess.Write);
    bw = new BinaryWriter(fs);
    fs.Seek(d.getPosizione(),SeekOrigin.Begin);
    bw.Write(d.getCognome().PadRight(19, ' '));
    bw.Write(d.getNome().PadRight(14, ' '));
    bw.Write(d.getStipendio());
    bw.Write(d.getAliquota());
    bw.Write(d.getStipendioNetto());
    fs.Close();
    bw.Close();
}
```

Grazie al sotterfugio del campo nascosto relativo alla posizione, il dipendente da modificare contiene anche la sua posizione nell'archivio e la modifica diventa molto semplice

Cancella dipendente



Cancella dipendente evento click sulla voce di menu

```
private void cancellaDipendenteToolStripMenuItem_Click(object sender, EventArgs e)
{

    if (riepilogo.SelectedItems.Count == 0) MessageBox.Show("devi selezionare un dipendente da Cancellare");
    else
    {
        if (MessageBox.Show("Sei sicuro di voler cancellare il dipendente selezionato?", "conferma cancellazione",
             MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
        {
            mioArchivio.cancella(Convert.ToInt64(riepilogo.SelectedItems[0].SubItems[5].Text));

            visualizzaElencoToolStripMenuItem_Click(this, null);
        }

    }
}
```

Come si vede dal codice questa volta vado a prendere solo la posizione del record ada cancellare contenuta nel campo nascosto e la passo al metodo cancella di Dipendenti

Cancella dipendente

```
public void cancella(long pos) {

    fs = new FileStream(archivio, FileMode.Open, FileAccess.Write);
    fs.Seek(pos, SeekOrigin.Begin);
    fs.WriteByte(Convert.ToByte('-'));
    fs.Close();
    numRecInseriti--;

}
```

Il metodo cancella è davvero semplce perché ricevendo la posizione del record basta andare in modo diretto sul record e cancellarlo logicamente cioè inserire un trattino nel primo carttere