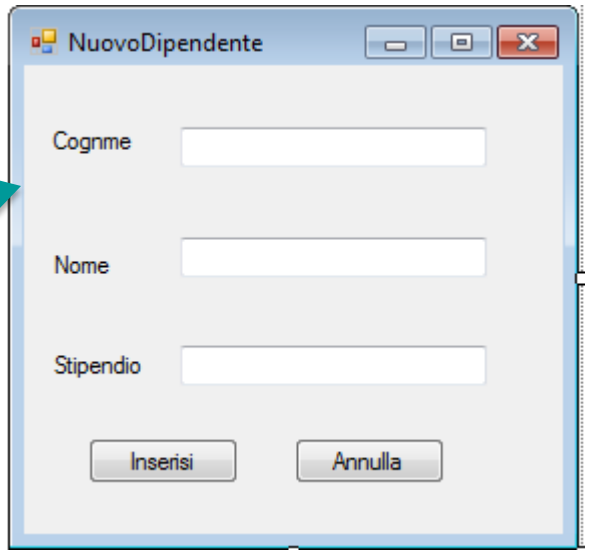
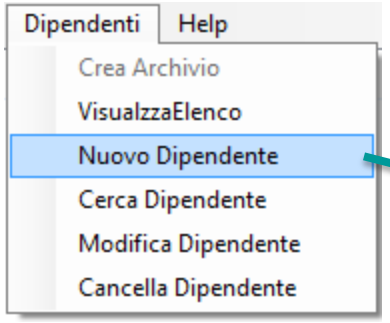
File ad accesso diretto funzioni Hash Parte2

Prof. Francesco Accarino
IIS Altiero Spinelli Sesto San Giovanni

Risposta Al click sulla voce di menu

Ovviamente alla risposta al click sulla voce di menu nuovoDipendente dovrò visualizzare un form simile a questo Dopodiché nel gestore dell'evento click della voce di menù potrò scrivere il codice seguente per visualizzarlo





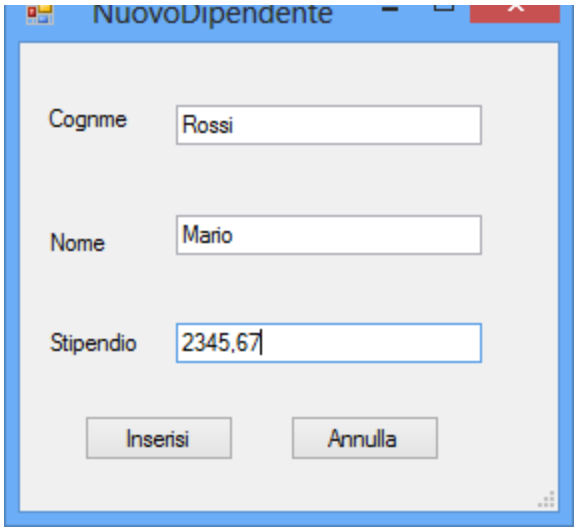

```
private void nuovoToolStripMenuItem_Click(object sender, EventArgs e)
{
    NuovoDipendente nd;
    if (mioArchivio.getNumRec() < mioArchivio.getNumDip())
        nd = new NuovoDipendente(mioArchivio);
     else MessageBox.Show("L'archivio è Pieno!!", "errore",
                          MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

Al costruttore di NuovoDipendente viene passato il reference a Dipendenti mioArchivio per potre usare il metodo insericiDipendente

Risposta Al click sulla voce di menu

Come prima cosa andiamo a modificare il costruttore della classe Nuovo dipendente aggiungendo il parametro per il reference all'archivio

```
public partial class NuovoDipendente : Form
{
    Dipendenti archivio;
    public NuovoDipendente(Dipendenti d)
    {
        archivio = d;
        InitializeComponent();

    }
```

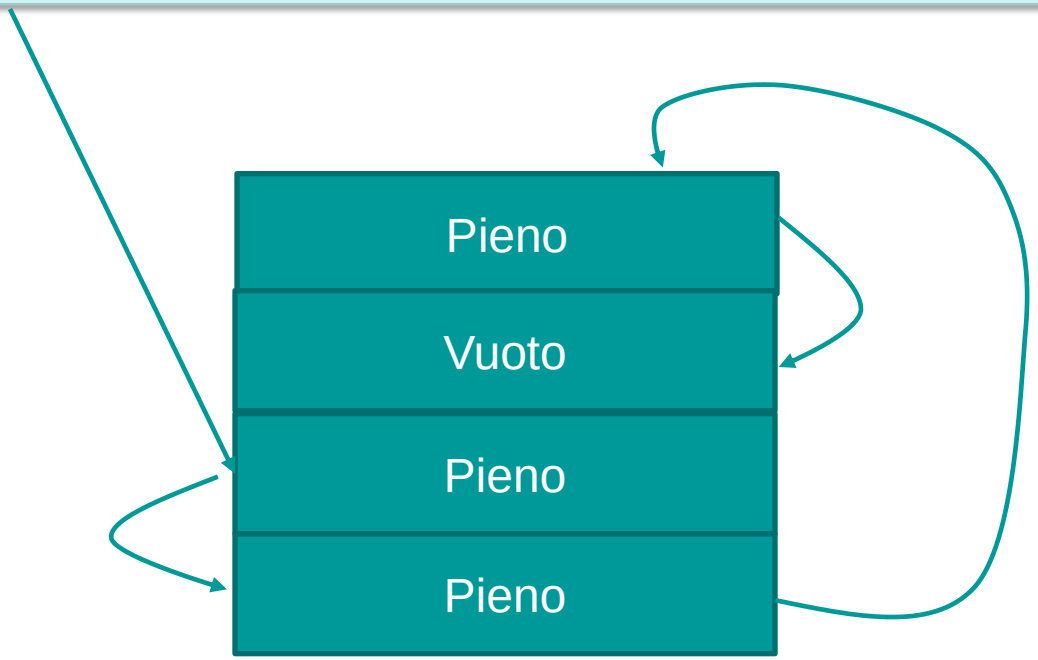


E poi una volta che l'utente ha compilato i campi e clicca sul bottone inserisci, dopo aver controllato che tutti i campi sono stati compilati andremo a calcolarci la posizione relativa nell'archivio corrispondente a questo record mediante la funzione di hash che trasforma il cognome in un numero e poi mediante la funzione di divisione modulo. Segue il codice:

```
private void inserisciBtn_Click(object sender, EventArgs e)
{
    CharEnumerator car;//oggetto per enumerare  singoli caratteri di una stringa

    long hashCode,posizioneRelativa;//hashCode valorenumerico ottenuto come somma dei valori ascii dei singoli
                                    //caratteri del cognome da inserire
                                    //posizione relativa resto intero della divisione tra hashCode e numero di record

    if (cognome.Text == "" || nome.Text == "" || stipendio.Text == "")
    {
        MessageBox.Show("Devi Compilare tutti i Campi!", "errore", MessageBoxButtons.OK, MessageBoxIcon.Error);
        this.Close();
    }//se non sono stati inseriti i dati lo segnalo e esco

    else{//inizio operazioni di inserimnto

    //creo un nuovo dipendete con i valori inseriti
    Dipendente dipendente =new Dipendente(cognome.Text,nome.Text,Convert.ToDouble(stipendio.Text));

    //calcolo l'indice relativo del record nel file mediante la funzione di hash
    hashCode = 0;
    car = dipendente.getCognome().GetEnumerator();

    while (car.MoveNext() == true)
    {
      hashCode = hashCode + car.Current; //somma dei valori ascii dei caratteri del cognome
    }
    hashCode = hashCode * dipendente.getCognome().Length;//moltiplico la somma dei singoli caratteri per la lunghezza del cognome
    posizioneRelativa = (hashCode % archivio.getNumDip()); //calcolo il resto intero della divisione
    posizioneRelativa = posizioneRelativa * 55; //moltiplico indice record trovato per la lunghezza del record
    archivio.inserisci(posizioneRelativa, dipendente);//chiamo il metodo di archivio per inserire il nuovo dipendente

    }

    this.Close();
}
```

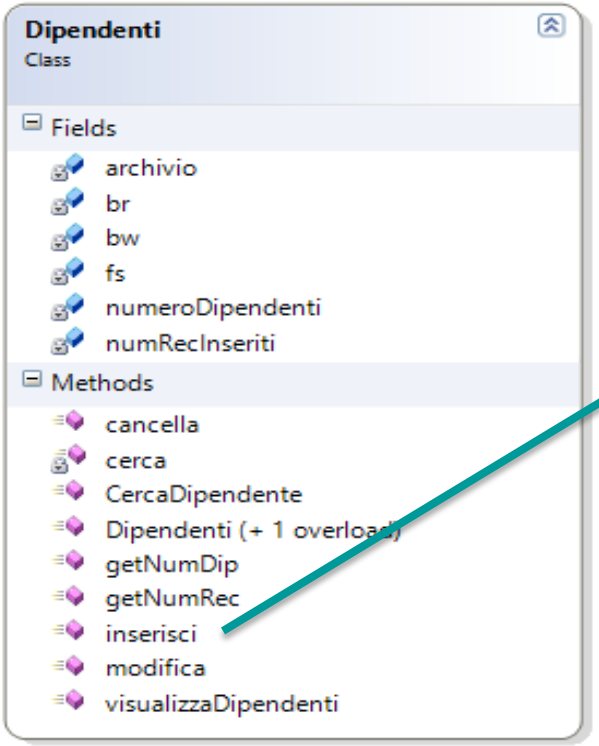
Vediamo ora come implementare il codice del metodo inserisci della Classe dipendenti, ricordando che un record è vuoto se inizia per - .

- Apro il file e mi posiziono nella posizione relativa passata e se questa è già occupata utilizzo la scansione lineare per cercarne una vuota segnalando le collisioni



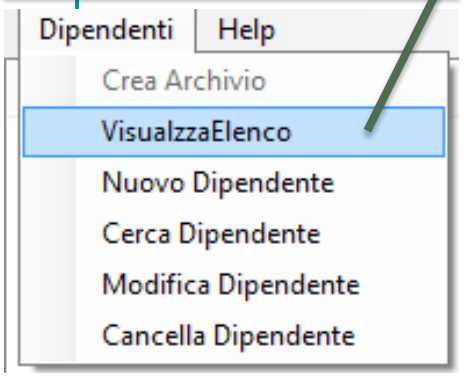
- Una vota trovato la posizione libera inserisco nel'archivio i dati utilizzando i metodi di Dipendente che mi è stato passato

Codice della funzione Inserisci

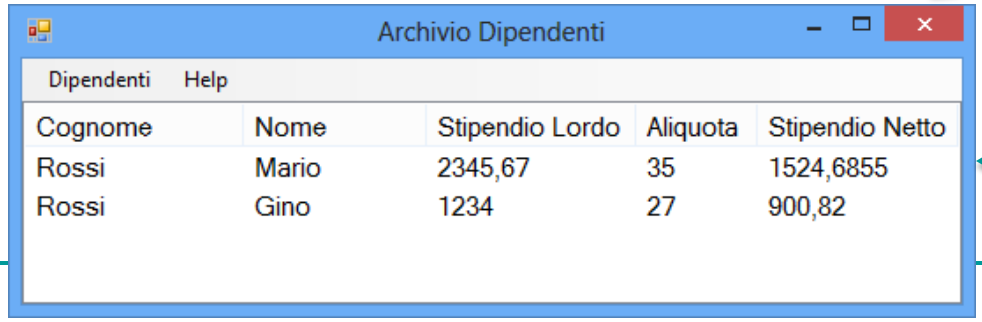



```
public void inserisci(long p, Dipendente dip)
   {

   //apro un Filestream sul file e mi posiziono sulla posizione corispondente al record
          fs = new FileStream(archivio, FileMode.Open);
          fs.Seek(p, SeekOrigin.Begin);
          //mentre trovo una collisione mi sposto sul prossimo e
          //se raggiungo la fine ricomincio dall'inizio
          while (fs.ReadByte()!=Convert.ToByte('-'))
          {
             MessageBox.Show("collisione cerco nuova posizione");
              if (fs.Position<(numeroDipendenti*55)- 54)
                  fs.Seek(54, SeekOrigin.Current);
              else
                  fs.Position = 0;
          }
          //trovata la posizione finalmente inserisco il record
          numRecInseriti++;
          fs.Seek(-1, SeekOrigin.Current);
          BinaryWriter bw = new BinaryWriter(fs);
          bw.Write(dip.getCognome().PadRight(19,' '));
          bw.Write(dip.getNome().PadRight(14,' '));
          bw.Write(dip.getStipendio());
          bw.Write(dip.getAliquota());
          bw.Write(dip.getStipendioNetto());
          bw.Close();
          fs.Close();
   }
```

Visualizzazione dell'archivio



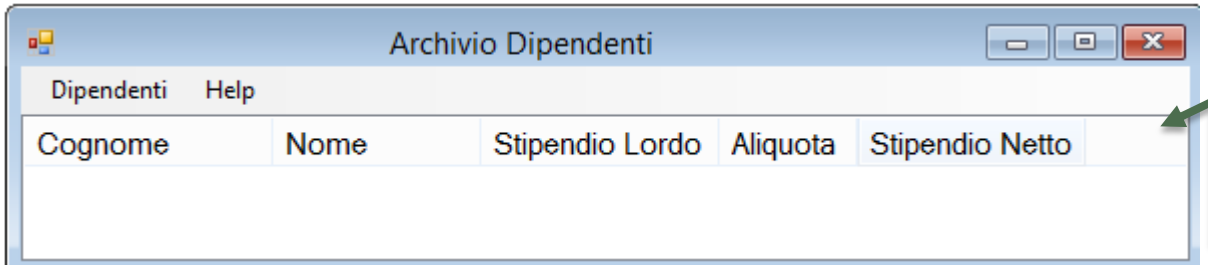

```
public void visualizzaElencoToolStripMenuItem_Click(object sender, EventArgs e)
{
    ListViewItem riga; //oggetto della listview per creare le righedella listview
    List<Dipendente> elenco; //arraylist di oggetti di tipo dipendente

    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()));
        }
    }

}
```



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