Il framework .NET consiste di cinque componenti fondamentali:

Common Language Runtime (CLR). Esso fornisce

le funzionalità fondamentali per l’esecuzione di un’applicazione.

si occupa inoltre dell’interfacciamento con il sistema operativo.

Lo strato immediatamente al di sopra del CLR, è costituito dalla Base Class Library (o .NET Framework

Class Library) di .NET, cioè un insieme di classi fondamentali, utili e necessarie a tutte le applicazioni ed a

tutti gli sviluppatori.

contiene i tipi primitivi, le classi per l’Input/Output, per il

trattamento delle stringhe, per la connettività, o ancora per creare collezioni di oggetti.

Al di sopra della BCL,

vengono quindi fornite le classi per l’accesso alle basi di dati e per la manipolazione dei dati XML,

Lo strato più alto del framework è costituito da quelle funzionalità che offrono un’interfacciamento con

l’utente finale (Windows Forms,web forms,web services).

CLR:

Il CLR si occupa dell’istanziazione degli oggetti, esegue dei controlli di sicurezza, ne segue tutto il ciclo di

vita, ed al termine di esso esegue anche operazioni di pulizia e liberazione delle risorse.

In .NET ogni programma scritto in un linguaggio supportato dal framework viene tradotto in un linguaggio intermedio comune, detto CIL (Common Intermediate Language) dal compilatore adatto per il linguaggio scelto.

Il risultato della compilazione non dipende dalla macchina su cui verra’ eseguito il programma.

Il CLR però non esegue direttamente dei moduli, esso lavora con delle entità che sono chiamate assembly.

Un assembly può essere non solo un’eseguibile, ma anche una libreria DLL contenente una collezione di

tipi utilizzabili eventuamente in altre applicazioni.

Il CIL viene trasformato in codice nativo dal compilatore JIT e puo’ essere eseguito sulla macchina a patto che sia presente il CLR corrispondente.

Per garantire l’interoperabilita’ con gli altri linguaggi C# deve poter essere utilizzabile anche con gli altri linguaggi.

Per permettere questo Common Type

System (CTS), suddivisi in particolare, in due grandi categorie, tipi riferimento e tipi valore.

Ma ogni tipo ha come classe base il tipo object. (tutto e’ un oggetto).

Ci sono delle regole da seguire che se non rispettate non rendono i linguaggi compatibili tra loro.

Il CLR si occupa della gestione della memoria in maniera automatica, per mezzo del meccanismo di

garbage collection.

Il LINGUAGGIO C#:

i programmi possono naturalmente essere contenuti su più file di testo, e C# usa

l’estensione .cs per questi.

Ogni file C# può contenere uno o più namespace.

CiaoMondo.exe prodotto dalla compilazione e puo’ essere eseguito sulla macchina in cui e’ installato il .NET runtime.

La libreria di classi del framework .NET contiene, fra le altre, la classe System.Console.

Serve per gli input e gli output.

Il metodo main e’ il metodo principale dell’applicazione.

Al momento della compilazione, il compilatore ricerca per default il metodo Main dell’applicazione, e per

default esso se ne aspetta uno ed uno solo.

E’ però possibile scrivere più metodi Main in diverse classi, cosa molto utile ad esempio per la creazione di

un test per ogni classe che implementiamo.

Dobbiamo specificare quale main vogliamo sia visto come punto di ingresso del programma.

COMMENTI: C# supporta inoltre la generazione automatica di documentazione mediante i commenti.

Namespace: non è necessaria cioè corrispondenza fra il nome dei namespace e il nome delle directory in cui sono

contenuti. Per mezzo dei namespace ad esempio possiamo raggruppare classi che si riferiscono o che forniscono

funzionalità correlate.

BASI del linguaggio.

Tipi riferimento: I tipi riferimento sono i tipi le cui istanze vengono create nell’area di memoria heap. Il linguaggio C#

fornisce due tipi di riferimento primitivi, il tipo object ed il tipo string.

Il tipo **object** è il padre di tutti i tipi, primitivi e non. Tutti i tipi del framework .NET derivano dalla

classe System.Object.

metodi ereditati direttamente dalla classe System.Object, ad esempio i metodi ToString() o Equals().

E’ possibile spostare una variabile dallo stack allo heap e viceversa.

Attenzione!!! Una stringa viene sempre creata sullo heap e non puo’ essere modificata… se si fa un’assegnazione ad un tipo string viene creato un nuovo oggetto sullo heap di tipo string.

ARRAY:anche gli array sono tipi riferimento

Conversioni di tipi: con is si verifica che un tipo sia effettivamente compatibile in fase di esecuzione (restituisce true o false).. mentre con as viene usato per la conversione ad un altro tipo ma se il tipo non e’ convertibile viene restituito null (e non viene sollevata un’eccezione di conversione non possibile… paragonabile al dynamic\_cast utile per fare downcasting). Se si utilizza l’operatore di casting esplicito controllare la possibilita’ di fare casting con is.

PROGRAMMAZIONE AD OGGETTI:

una classe sealed non puo’ essere ereditata dalle classi figlie

System.String è una classe sealed, per evitare che si possa creare ereditare da essa una diversa

implementazione delle stringhe.

NB: tutte le classi derivano da object (anche i tipi primitivi come int double float ecc)

Tipi valore e tipi riferimento: i tipi valore sono i tipi primitivi mentre i riferimenti sono gli oggetti.

Passaggio dei parametri: in C# il passaggio dei parametri avviene per valore!!

Vale a dire che si crea una copia dell’oggetto passato al metodo.

Per i tipi valore viene creata una nuova variabile parametro formale mentre per i riferimenti viene fatta una copia del riferimento.

Se modifico una variabile valore all’interno di un metodo il valore della variabile esterna non cambia.. per cambiarla devo anteporre alla variabile la keyword ref.

Per i tipi riferimento si crea una copia del riferimento ed alla modifica si modifica anche la variabile esterna. Nel caso in cui cambio il riferimento con un nuovo oggetto all’interno del metodo la variabile riferimento all’esterno non cambia.

Se voglio cambiare il riferimento della variabile esterna basta mettere anche in questo caso la keyword ref.

ATTENZIONE: se un metodo vuole un object (quindi riferimento) e passo un int (quindi valore) viene fatto un cast implicito da int a object creando una nuova istanza di un object (che rappresenta l’intero) il nuovo oggetto e’ un riferimento all’oggetto che rappresenta la copia dell’int e la variabile esterna non viene modificata.

UPCASTING E DOWNCASTING

Up sempre concesso implicitamente perche’ c’e’ la certezza che un oggetto sia convertibile nella sua classe base.

Down deve essere esplicito con l’operatore o utilizzando as (meglio utilizzare as perche’ se l’oggetto non e’ convertibile non restituisce un’eccezione InvalidCastException ma ritorna null).

Se si vuole verificare se l’oggetto e’ convertibile utilizzare prima is.

La classe System.Object

la keyword object non è altro che un alias per System.Object,

dunque perfettamente equivalente.

public virtual string ToString() Restituisce una stringa che rappresenta

l’oggetto.

public virtual int GetHashCode() Restituisce un intero, utilizzabile come valore di

hash, ad per la ricerca dell’oggetto in un elenco

di oggetti.

public virtual bool Equals(object o) Effettua un test di uguaglianza con un’altra

istanza della classe.

public static bool Equals(object a,object b) Effettua un test di uguaglianza fra due istanze

della classe.

public static bool ReferenceEquals(object a,object b) Effettua un test di uguaglianza per verificare se

due riferimenti si riferiscono alla stessa istanza

della classe.

public Type GetType() Restituisce un oggetto derivato da System.Type

che rappresenta il tipo dell’istanza.

protected object MemberwiseClone() Effettua una copia dei dati contenuti

nell’oggetto, creando un’altra istanza.

protected virtual void Finalize() Distruttore dell’istanza.

Il metodo ToString

Studente studente=new Studente(1234,”pinco”, “pallino”);

Console.WriteLine(studente);

}

}

}

La chiamata Console.WriteLine(studente) invoca al suo interno il metodo studente.ToString(), e in questo

caso stamperà una stringa del tipo:

TestObject.Studente

Mentre magari ci saremmo aspettati una stringa che contenga matricola, nome e cognome dello studente.

Per far ciò, come abbiamo imparato nel precedente capitolo, scriviamo nella classe Studente un override del

metodo ToString():

public override string ToString()

{

return "Studente "+matricola+" - "+cognome+" "+nome;

}

I metodi Equals e ReferenceEquals

I metodi Equals effettuano il confronto di due istanze, e sono fondamentali in quanto vengono richiamati

in diverse altre classi per testare l’uguaglianza di due oggetti, ad esempio nelle collezioni,

, il metodo di istanza bool Equals(object o) viene utilizzato per verificare se esse

contengono o meno una certa istanza (la stessa istanza).

Se voglio verificare se hanno i campi uguali allora devo fare l’override del metodo.

(riferirmi alla classe base se voglio riottenere il comportamento precedente o utilizzare referenceEquals)

ReferenceEquals usato per verificare i riferimenti.

Il metodo GetHashCode

In particolare la Base Class Library fornisce una classe Hashtable, che utilizza il metodo GetHashCode per ricavare il codice hash, ed il metodo Equals per confrontare gli oggetti da memorizzare o memorizzati.

Attenzione: visto che getHashTable e Equals sono usati dalla classe HashTable e se facciamo l’override de equals viene segnalato un warning perche’ bisogna fare anche l’override del metodo gethashtable.

Il metodo GetType

Il metodo GetType() restituisce un'istanza della classe System.Type, che può essere utilizzata per ottenere una grande varietà di informazioni a run-time sul tipo corrente di un oggetto, ad esempio il namespace, il nome completo, i metodi di una classe, quindi è il punto di accesso alla cosiddetta tecnologia di Reflection, la quale consente di esplorare il contenuto di un tipo qualunque.

Clonare un oggetto

La classe System.Object fornisce il metodo protected MemberwiseClone() che consente di ottenere una copia dell'oggetto sul quale viene invocato.

Il metodo restituisce una copia campo a campo dell’oggetto passato (i tipi per i tipi valore verranno fatte delle copie mentre per i tipi riferimenti verranno copiati solamente i riferimenti (non vengono fatte delle copie profonde)).

Nei casi in cui servisse invece una copia profonda dell'oggetto, è necessario che la classe implementi l'interfaccia ICloneable, che espone l'unico metodo object Clone(), in cui possiamo realizzare sia una deep copy che una shallow copy

Distruzione di un oggetto

Molti tipi però utilizzano delle risorse unmanaged, ad esempio aprono dei file, delle connessioni di rete, o verso un database, ed è necessario prevedere un meccanismo che liberi anche tali risorse.

Quando il Garbage Collector stabilisce che un oggetto può essere rimosso dalla memoria, esso cerca ed eventualmente invoca il metodo Finalize ereditato dalla classe System.Object.

Il compilatore non permette di fare l’override del metodo direttamente ma definiendo un distruttore che libera le risorse in realta’ si sta facendo override del metodo (serve a liberare le risorse che non sono gestite dal garbage collector ad esempio una connessione ad un db).

In ogni caso meglio liberare le risorse in modo esplicito utilizzando il metodo Finalize con il pattern Dispose.

Il pattern Dispose() non virtuale… permette di definire le funzionalità da fornire e da implementare in una classe per ottenere in maniera deterministica ed esplicita la distruzione delle sue istanze e la liberazione delle relative risorse occupate.

L’implementazione di dispose() viene fatta nella classe che utilizza un metodo di utilita’ dispose(bool)

La classe System.String

In C# il tipo System.String è un tipo riferimento, derivato da System.Object, ma nonostante ciò non è possibile costruire una stringa utilizzando l'operatore new, e quindi la stringa è trattata come un tipo primitivo, cioè è sufficiente una semplice assegnazione di un literal ad una variabile di tipo string.

Per ottenere una stringa modificabile consultare la classe StringBuilder: nota: se si supera la lunghezza dell’array di caratteri del builder viene creato un altro oggetto che rappresenta la stringa con lunghezza opportuna. Attenzione se si utilizza il metodo ToString() che restituisce un rif quindi il rif non si riferisce piu’ all’oggetto contenuto nel builder.

COLLEZIONI

Utilizzare la classe System.Collections

Eccezioni

Un'eccezione in C# è un oggetto, derivato in modo diretto o indiretto dalla classe System.Exception. Quando si verifica una eccezione, viene creato un tale oggetto contenente, tra le altre, informazioni sull'errore che si è verificato. Eccezioni più specifiche possono inoltre derivare da classi particolari, ad esempio eccezioni che riguardano errori di Input/Output sono in genere derivate dalla classe IOException, da cui a loro volta derivano fra le altre FileNotFoundException o FileLoadException che riguardano ancora più specificatamente eccezioni nella gestione di file.

Eventi

I messaggi sono il meccanismo principale utilizzato dalle applicazioni Windows.

Per generare un evento dobbiamo prima definire una classe che contenga le informazioni correlate ad esso. Il framework .NET fornisce una classe base da cui derivare i nostri eventi personalizzati, la classe System.EventArgs.

Consumare un evento Per consumare un evento bisogna prevedere un metodo che venga richiamato al verificarsi dell’evento, e registrare tale metodo come gestore dell’evento

Applicazioni Windows Forms

In questo capitolo vedremo come avviene la creazione di un’applicazione Windows (o in generale a finestre) in C#, mediante l’uso delle classi messe a disposizioni dalla Base Class Library, ed in particolare nel namespace System.Windows.Forms.

In genere un’applicazione interattiva a finestre, ha almeno una finestra che si presenta all’utente all’avvio del programma. Tale finestra è creata in C# mediante una classe ereditata dalla classe System.Windows.Forms.Form

In genere lo startup dell’applicazione in C# avviene per mezzo del metodo statico Run della classe Application che prende come unico parametro in ingresso un’istanza di una classe derivata da form, quindi in genere un’istanza della form principale della nostra applicazione

3La classe Form

Una Form del framework .NET è quella che in genere siamo abituati a chiamare finestra (window)

Windows.Forms e’ il namespace

Non e’ sufficiente creare una form per visualizzarla… invocare quindi il metodo show().

Il metodo hide() nasconde (non distrugge) la finestra.

Invocare il metodo Application.Run per la visualizzazione dellatype form principale del programma, mentre per le form che saranno generate dal programma in un secondo tempo, sarà sufficiente invocarne il metodo Show. (altrimenti il programma termina immediatamente).

La proprieta’ text da il nome alla finestra.

Se non passo al metodo run la form alla chiusura della finestra l’applicazione restera’ aperta.

MessageBox

utilizzata per mostrare la classica finestra di messaggio all’utente, ed eventualmente con l’aggiunta di uno o più pulsanti per ottenere una conferma o far effettuare una scelta durante l’esecuzione dell’applicazione.

Una MessageBox può contenere più pulsanti, fino ad un massimo di tre.

Finestre modali:

L’unica differenza, rispetto a quanto visto finora, per mostrare una form in maniera modale, è di invocare il metodo ShowDialog invece che il già visto Show. Alla chiusura di una finestra modale viene restituito DialogResult (come per message box).

Tale valore deve dunque essere impostato mediante la proprietà Form.DialogResult, quindi deve essere resa invisibile la form mediante il metodo Hide, in modo che il chiamante può leggere il valore impostato, e sarà il metodo chiamante ad effettuare la chiusura della form.

Il framework .NET fornisce delle classi per utilizzare le comuni dialog delle applicazioni Windows System.Windows.Forms e derivate

10.3.4 Proprietà e metodi delle form

NB: Form deriva da Control quindi eredita tutti i comandi che possono avere i componenti che vengono posizionati al’interno della form.

f.Icon=new Icon("hello.ico");

f.Size=new Size(300,400);

La dimensione dell’area client, cioè dell’area della form escludendo la barra del titolo ed i bordi è invece rappresentata dalla proprietà ClientSize.

La posizione sullo schermo è rappresentata dalla proprietà Location di classe Point, mentre la posizione rispetto al desktop, che tiene quindi conto ad esempio di barre degli strumenti o della barra delle applicazioni, è rappresentata dalla proprietà DesktopLocation:

f.MinimumSize=new Size(200,150);

f.MaximumSize=new Size(800,600);

f.StartPosition=FormStartPosition.CenterScreen;//centra la form sullo schermo f.StartPosition=FormStartPosition.CenterParent;//centra rispetto alla form madre

Il colore di sfondo della form viene impostato agendo sulla proprietà BackColor:

Con l’uso della proprietà Opacity possiamo invece dare un effetto di trasparenza.

10.4Aggiungere i controlli

in generale un controllo deriva dalla classe System.Windows.Forms.Control.

La classe Form mantiene in una collezione i controlli in essa contenuti, la proprietà Controls permette di ricavare tale collezione.

f.Controls.Add(txt); // per aggiungere un controllo alla form

f.Controls.Remove(txt); // per rimuovere il controllo dalla form

bt.Parent=f; //imposto la proprieta’ parent del pulsante

è equivalente all’utilizzo del metodo f.Controls.Add(bt), mentre assegnando il valore null alla proprietà Parent, si rimuove il controllo dalla collezione Controls della form Parent.

TextBox[] caselle=new TextBox[4]; f.Controls.AddRange(caselle); per inserire un array di controlli in un colpo solo.

10.4.1 Proprietà dei controlli

La classe principale da cui derivano tutti i controlli è la classe Control

Le proprietà della classe Control, insieme ai suoi metodi, sono estese da eventuali altri membri specializzati delle classi derivate da Control.

10.4.2 Controlli di testo

TextBox ….. Select oppure SelectAll per selezionare tutto il testo.

txt.PasswordChar=’\*’;

LinkLabel, cioè una label il cui testo può però contenere uno o più collegamenti ipertestuali

10.4.3 Controlli di comando

Button

La funzionalità tipica di un pulsante è naturalmente quella di eseguire una certa azione quando l’utente clicca

Per ottenere una notifica di tale evento bisogna associare al pulsante un gestore adeguato:

private void bt\_Click(object sender, EventArgs e) {

Button button=(Button)sender;

MessageBox.Show("Hai cliccato "+button.Text);

}

Nel costruttore della classe button personalizzato:

bt.Click+=new EventHandler(bt\_Click);

Installa un nuovo gestore per l’evento Click, di nome bt\_Click e che rispetta la firma definita dal delegato EventHandler.

Il primo parametro sender, rappresenta l’oggetto che ha generato l’evento, in questo caso il pulsante bt. Essendo tale parametro di tipo object, per accedere alle proprietà di bt, è necessario un cast esplicito:

IMPORTANTE:

L’evento Click non è un’esclusiva della classe Button, ma è ereditato dalla classe Control, dunque ogni altro controllo può ricevere una notifica di tale evento, e la gestione è analoga a quanto visto per un Button.

NotifyIcon

notify.ContextMenu=new ContextMenu( new MenuItem[]{new MenuItem("Ripristina"),new MenuItem("Esci")} );

10.4.4 Controlli di selezione

I controlli di selezione permettono di specificare dei valori o effettuare delle scelte fra più valori

CheckBox e RadioButton

Il controllo Listbox permette di visualizzare una lista di elementi, selezionabili dall’utente con un click del mouse.

ListView

Il controllo ListView permette di rappresentare una lista di elementi, caratterizzati da un testo e da una eventuale icona

Una ListView viene popolata con elementi rappresentati da oggetti ListViewItem, contenuti nella collezione Items. Ad esempio per creare un nuovo elemento possiamo scrivere: ListViewItem lvi=new ListViewItem("Elemento 1");

l’elemento creato viene quindi aggiunto agli Items della ListView: lv.Items.Add(lvi); mentre per rimuoverlo basta utilizzare il metodo Remove: lv.Items.Remove(lvi);

ThreeNodeView

consultare il documento (crea un albero in cui e’ possibile vedere le sottocartelle in una directory)

5Menù

La classe Menu, del namespaceSystem.Windows.Forms, costituisce la classe base di ogni tipo di menù

Dalla classe Menu derivano la classe MainMenu, che rappresenta il menù classico associabile ad una form, mentre ContextMenu permette di realizzare i menù contestuali associati ai singoli controlli.

Ogni menù è costituito da una serie di oggetti MenuItem, la cui classe deriva anch’essa da Menu.

10.5.1 Il menù di un’applicazione

La classe MainMenu rappresenta il contenitore per la struttura dei menù di una form. Come detto un menù è costituito da un array di oggetti MenuItem

Per associare ad una form un menù, basta impostare la sua proprietà Menu: MainMenu menuPrincipale=new MainMenu(menuItems); miaForm.Menu= menuPrincipale;

MenuItem

MenuItem mnuFileNuovo=new MenuItem();

Per la creazione di menu annidati

MenuItem mnuFile=new MainMenu(“&File”,new MenuItem[]{mnuFileNuovo,mnuFileApri});

//per aggiungere delle azioni

mnuFileApri.Click+= new System.EventHandler(this. mnuFileApri\_Click);

e naturalmente bisogna implementare il relativo metodo di gestione: private void mnuFileApri\_Click(object sender, System.EventArgs e) { OpenFileDialog fd = new OpenFileDialog(); fd.ShowDialog(); }

l’Entity Framework per interazione con il db

System.Data.Entity.DbContext

Stato dell’entita’

**System.Data.EntityState** che dichiara i seguenti valori -

* **Aggiunto:** l'entità è contrassegnata come aggiunta.
* **Eliminato:** l'entità è contrassegnata come eliminata.
* **Modificato:** l'entità è stata modificata.
* **Invariato:** l'entità non è stata modificata.
* **Distaccato:** l'entità non viene rilevata.

Attenzione… la connessione viene chiusa quando viene chiamato dispose

Domande Gianluca: