Molti di voi vedranno questa lingua come una creazione Microsoft che non ha nulla a che fare con il Software Libero. Tuttavia, mentre è vero che la sua creazione dipendeva direttamente dalla società di Redmond ed è il linguaggio principale per sviluppare con la piattaforma .NET, il grande Miguel de Icaza è stato in grado di creare un'alternativa gratuita compatibile con questa nuova tecnologia.

Il nome di questo progetto è Mono e questo sarà lo strumento che useremo in tutta una serie di voci in cui vi darò le nozioni di base per utilizzare C.

Preparazione e installazione

Per installare Mono dovremo installare il pacchetto con lo stesso nome sul nostro sistema, con il nostro gestore di pacchetti preferito. Oltre a Mono, useremo MonoDevelop come ambiente di sviluppo integrato, per semplificare il processo di compilazione e prendersi cura dei diversi compilatori per C- a seconda della versione. Il pacchetto MonoDevelop avrà lo stesso nome.

Se si utilizza Ubuntu si può fare direttamente: sudo apt-get installare monosviluppo e le sue dipendenze verranno installate.

Salve, mondo!

Iniziato:

- •Apriamo MonoDevelop
- •Fare clic su "Avvia nuova soluzione"
- •Abbiamo scelto "Console Project" e indicato il titolo: "HelloWorld"
- •Continuiamo, lasciando vuote le opzioni di packaging e così via che proponi e abbiamo già davanti a noi il codice di Hello World! presso C Sharp.

Puoi vedere il codice qui sotto:

```
using System;

namespace holaMundo
{
	class MainClass
	{
		public static void Main (string[] args)
		{
			Console.WriteLine ("¡Hola Mundo!");
		}
	}
}
```

Spiegazione

Andiamo in parti.

- •utilizzando il sistema; Usiamo lo spazio dei nomi System. Utilizzando spazi dei nomi al di fuori del nostro possiamo fare uso di funzioni, ad esempio, del sistema.
- •spazio dei nomi holamundo Definiamo il nostro spazio dei nomi, helloworld.

- mainClass (classe) Definizione della classe, la chiamiamo MainClass.
- public static void Main(string[] args) Definizione del metodo Main, come parametri passiamo gli argomenti con cui viene chiamato il programma (anche se non li useremo).
- Console.WriteLine("Hello World!"); Nella classe Console è presente un metodo WriteLine che viene passato una stringa e scrive la stringa "Hello World!" nella console.

Nel corso di questa serie di ingressi andremo un po 'più in profondità in Mono e C.

Seguiamo il corso di C- con Mono scavando più a fondo nel linguaggio. Se la scorsa settimana abbiamo parlato di come ottenere tutto e funzionante il tipico Hello World! questa settimana vedremo come trattare con condizionali, iteratori e operatori.

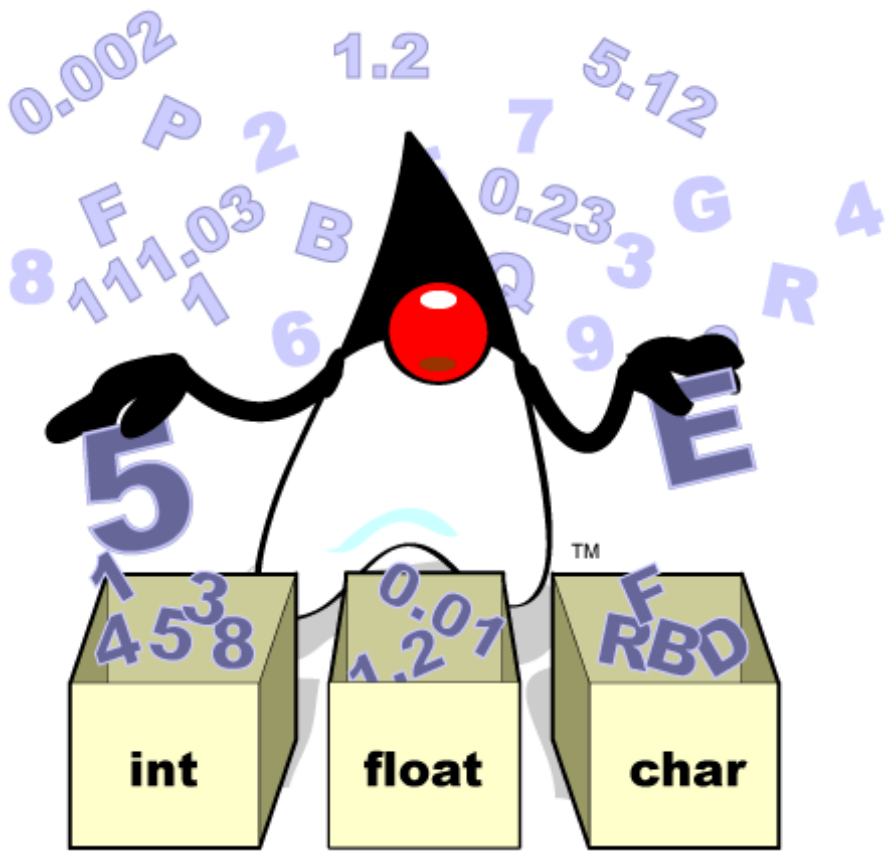
L'idea di questo corso non è quello di parlare di ciò che un'iterazione è, o anche di approfondire gli aspetti più dettagliati dell'uso; ma dare alcuni concetti brevi ma importanti, in modo che si può approfondire voi stessi.

Introduzione

Proprio come in tutti i linguaggi in C- ci sono tipi di dati. Questi sono molto simili a Java e come nota voglio solo commentare che:

- Il carattere predefinito, char, è UNICODE. Cioè, non occupa 8 bit, ma 16, e può gestire tutti i tipi di simboli (accenti, eees, ecc.) per impostazione predefinita.
- Se si proviene da C e si sta cercando di utilizzare numeri interi: 0 e diverso da 0 (false o true), il tipo bool è lo strumento per farlo.
- Oltre agli interi tipi di dati C, è sempre possibile archiviare riferimenti a oggetti, vettori e così via o anche tipi di dati **di base nelle variabili.**

Per passare riferimenti alle funzioni di tipo di dati di base usiamo le parole chiave out e ref (ora vedremo l'uso di entrambi). Si **tratta di una differenza rilevante con Java, in**cui le variabili dei tipi di dati di base vengono sempre passate per valore.



Oggetto dualità/tipo di dati

Come nel linguaggio Java questo metodo, i tipi di dati di base corrispondono effettivamente alle classi. Ad esempio, il tipo di dati float è in realtà un'abbreviazione per la creazione di un'istanza della classe System.Single.

Diamo un'occhiata a un esempio:

```
float a;
System.Single b = new System.Single();
```

Stiamo dichiarando, due variabili per memorizzare valori a virgola mobile a precisione singola delle forme. La variabile in cui viene dichiarata in stile C e la variabile b rappresenta in realtà un'istanza della classe System.Single.

Se all'interno di using abbiamo dichiarato lo spazio dei nomi System,può essere fatto direttamente: Single b s new Single().

Utilizzo del parametro out

Diamo un'occhiata direttamente a un esempio di utilizzo del parametro out. Inizializzeremo due variabili su 0, chiameremo una funzione in cui varieremo il suo valore. Dopo il mapping, riletto il valore delle variabili e verificare che siano state modificate.

Diamo un'occhiata al codice:

```
public static void Main (string[] args)
{
```

```
        int a = 0;
        int b = 0;

        Console.WriteLine("Valor de: a = {0}, b = {1}
        antes de llamar a la función", a, b);

        probarOut(out a,out b);

        Console.WriteLine("Valor de: a = {0}, b = {1}
        tras llamar a la función", a, b);
}

public static void probarOut(out int a, out int b)
{
        a = 100;
        b = 200;
}
```

All'interno della funzione testOut non abbiamo provato a leggere le variabili perché non sarebbe possibile. Se vogliamo farlo, simile a quello che faremmo con & in C, dovremmo usare ref.

Il parametro out viene utilizzato in alcune funzioni di conversione dei dati, ad esempio TryParse.

Utilizzo del parametro ref

Per finire questo articolo, vedremo l'uso della parola ref riservato. Faremo una funzione che scambia il valore di due numeri interi. In precedenza li inizializzeremo, li stamperemo per schermo e dopo aver controllato la chiamata alla funzione, ristampamo per schermo per verificare che il loro valore sia cambiato.

Diamo un'occhiata al codice:

```
public static void Main (string[] args)
{
        int a = 1;
        int b = 2;

        Console.WriteLine("Valor de: a = {0}, b = {1}
        antes de llamar a la función", a, b);

        intercambiar(ref a, ref b);
```

```
        Console.WriteLine("Valor de: a = {0}, b = {1}
        tras llamar a la función", a, b);
}

public static void intercambiar(ref int a, ref int b)
{
        int tmp = b;
        b = a;
        a = tmp;
}
```

Come si vedrà, i meccanismi di riferimento in C- sono molto più naturali che in Java. Se si desidera passare un riferimento di un tipo di dati di base in Java, è necessario utilizzare meccanismi quali le chiamate: AtomicReference.

Si sa che quando si tratta di oggetti che non sono tipi di dati di base, nessun problema.

Nell'ultima puntata, la scorsa settimana, abbiamo parlato di tipi di dati e di come fare riferimenti alle variabili,tuttavia, non ne abbiamo ancora parlato.

Variabili

Le variabili rappresentano un determinato spazio di memoria che ci riserviamo per archiviare un determinato valore. Questo valore può essere di qualsiasi tipo di cui abbiamo parlato l'altro giorno.

La tua dichiarazione è sempre la stessa: tipo di dati seguito dal nome che ti diamo. Diamo un'occhiata a un esempio di utilizzo delle variabili:

```
public static void Main (string[] args)
{
        string nombre;
        string apellidos;

        Console.Write("¿Cuál es tu nombre? ");
        nombre = Console.ReadLine();
        Console.Write("¿Cuáles son tus apellidos? ");
        apellidos = Console.ReadLine();

        Console.WriteLine("Usuario: {0}, {1}", apellidos, nombre);
}
```

Come si può vedere, vengono utilizzate due variabili, che vengono quindi stampate per schermo nell'ordine formale per visualizzare il nome e il cognome: "Cognome1 Apelllido2, Nome".

Quando si dichiara la variabile possiamo sempre assegnarle un valore:

```
string nombre = "Javier";
```

Costante

C'è un concetto molto simile che è quello delle costanti. A differenza dei precedenti, il relativo valore viene inizializzato e non può essere modificato in fase di esecuzione.

In linguaggi come C questo tipo di necessità viene in genere soddisfatto gestendo le direttive per il preprocessore,in C, c'è uno strumento espressamente per questo lavoro. Perché dovremmo voler gestire le costanti? Dipende molto dalle esigenze, ma gli esempi tipici sono sempre di numeri matematici ben noti, come il numero Pi o e:

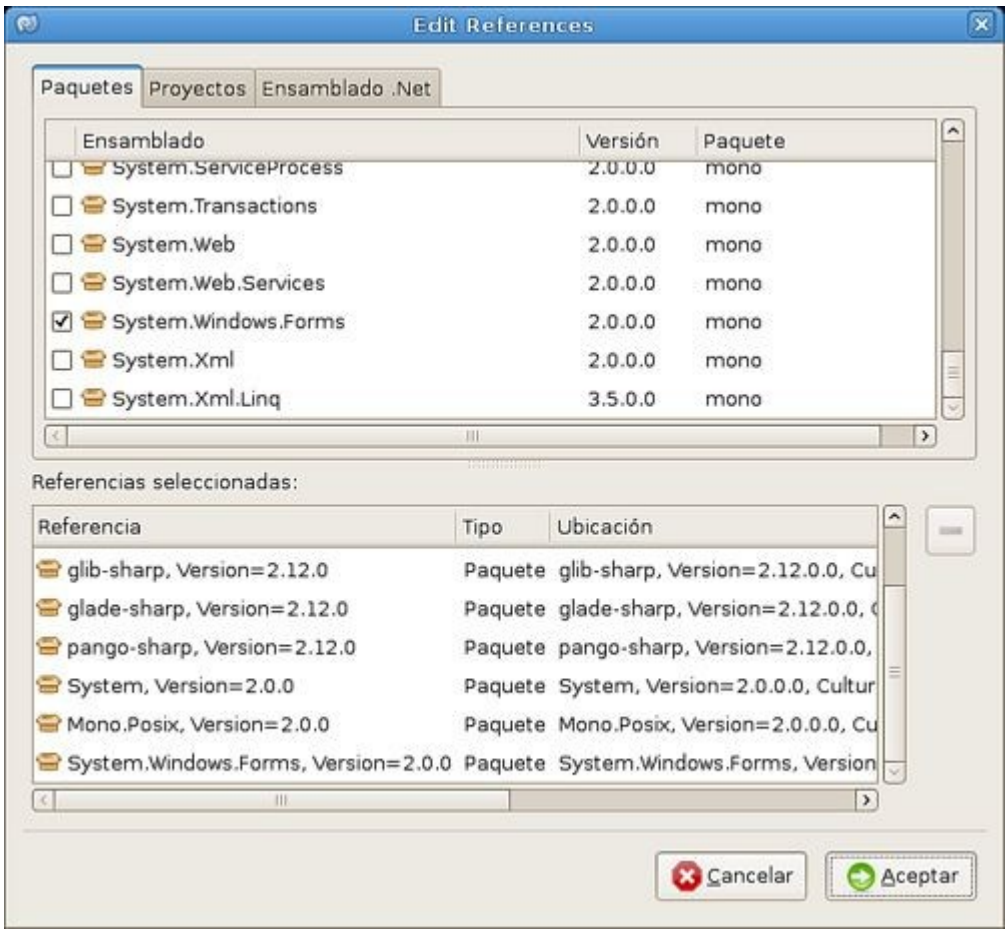
```
const decimal e=2.718281828459045235 4M;
```

Se si tenta di assegnarle un nuovo valore, il compilatore fornirà un messaggio di errore. Usiamo il suffisso M per memorizzare il numero come decimale e non come un numero a virgola (in una serie di concetti informatici parleremo di questo argomento).

Corso C- con Mono - Assemblaggio

Pubblicato: 15 Novembre 2010/Sotto: News/By: F. Javier Carazo Gil

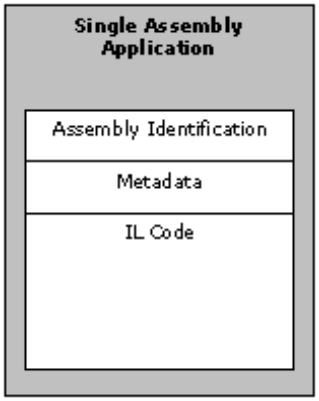
.netCAssembleaNamespaceScimmia



In tutte le tecnologie di questo tipo, esistono meccanismi per fornire all'utente metodi che consentono di incorporare funzionalità di file esterni. Questo, insieme all'orientamento degli oggetti, crea un problema: come organizzare e fare riferimento a tutte le classi (parleremo di classi più profondamente), in modo semplice e ciò non si trasquili in caso di incoerenze.

Assembly

Per risolvere questo problema, struttura in mono (e in .NET) assembly. Questi possono essere sia eseguibili (con .exe estensione), così come la libreria (con .dll estensione). Entrambe le estensioni sono state un retaggio delle tecnologie Microsoft fin dai tempi di MS-DOS. Gli assembly sono file in cui si trovano una serie di classi, compilate nel linguaggio intermedio.



Struttura di base di un assieme

Gli assembly di base vengono installati nel sistema durante l'installazione di Mono. Il sistema che gestisce l'organizzazione ed evitando problemi di controllo delle versioni (l'architettura dell'assembly stessa è orientata a evitare problemi come l'inferno di DLL) è la cosiddetta GAC, La Global Assemblie Cache.

Quando un progetto Mono fa riferimento a un assieme, esegue la ricerca in tre parti:

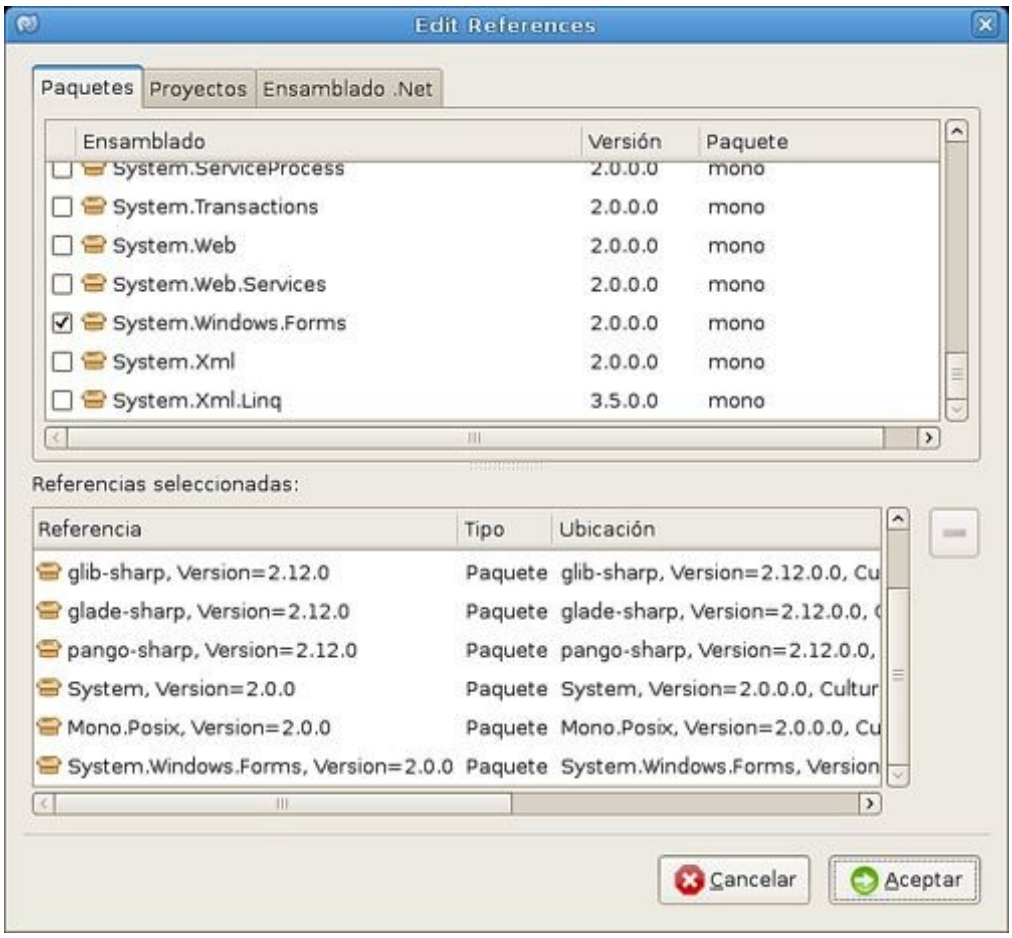
1. In primo luogo, nel percorso dell'assieme (nello stesso percorso)
2. Secondo nel percorso definito dalla variabile di ambiente MONO_PATH
3. Infine, chiedere alla Global Assembly Cache l'assemblea se non è stata trovata dai metodi di cui sopra

Come fare riferimento agli assembly dal nostro progetto con MonoDevelop?

Dal momento che sappiamo, anche sopra, come sono organizzate le biblioteche e le classi e in Mono, la prossima cosa è mettere questo che abbiamo appena spiegato in pratica.

Quando creiamo un progetto in MonoDevelop, gli assiemi di base vengono aggiunti automaticamente per quello che stiamo cercando: progetto console, con GTK...

Tuttavia, se abbiamo bisogno di includere più assembly dovremo farlo manualmente. Dove? Sul lato sinistro del nostro schermo, abbiamo la struttura della soluzione. Al suo interno, dopo il nome, all'interno dell'albero, c'è un livello che dice: **Riferimenti**, si fa clic destro su di esso e scegliere Modifica **riferimenti...**.



Apriamo un dialogo come quello che abbiamo appena mostrato. Ci vengono presentate tre ciglia:

- Pacchetti: tutti gli assembly gestiti dalla Global Assembly Cache
- Progetti: assembly di altri progetti
- Assembly .NET: qualsiasi assembly in dll o exe che abbiamo, possiamo unirlo e iniziare a lavorare con esso

Nella prossima puntata parleremo degli spazi dei nomi e di come chiamare le funzionalità e di come sono strutturati nel codice tramite gli **spazi dei nomi.**

Corso C- con Mono - Spazio dei nomi e regioni
Pubblicato: 19 Novembre 2010/Sotto: Tutorial /Da: F. Javier Carazo Gil
CCorsoNamespaceScimmia

Curso C# con Mono – Espacio de nombres y regiones

L'altro giorno abbiamo parlato di come il codice eseguibile è stato organizzato all'interno di assembly, che abbiamo potuto fare riferimento all'interno del nostro progetto immediatamente con Mono Develop. Supponiamo che abbiamo appena fatto riferimento al connettore MySQL per Mono/.NET e si desidera utilizzare la sua funzionalità nel codice. Come si organizzano assembly, classi e funzioni nel codice? Lo strumento di base per questo lavoro sono gli spazi dei nomi.

Namespaces

Gli spazi dei nomi o gli spazi dei nomisono il modo per organizzare all'interno di codice, classi e funzioni per la funzionalità. Sono direttamente correlati agli assembly, perché gli assembly definiscono se stessi, uno spazio dei nomi e a loro volta possono contenere un numero di essi.

Ogni volta che creiamo un progetto con Mono Develop, viene creato automaticamente uno spazio dei nomi per il progetto, da cui le sezioni si bloccano gerarchicamente man mano che il progetto cresce.

Per vedere quali funzioni, classi o quali altri spazi dei nomi, è possibile eseguire direttamente le operazioni seguenti. Digitare il nome dello spazio dei nomi (ad esempio System) e subito dopo, un punto, automaticamente l'IDE ci mostrerà un elenco di possibilità.

Qui di seguito potete vedere il menu a discesa. Perdonami per le trasparenze, ma gli effetti grafici di Compiz quando si fanno schermate con drop-down, giocare questi passaggi cattivi.

Regioni

Un altro modo per organizzare il codice, anche se non formalmente, ma per lo sviluppatore è attraverso le aree. Alcuni mesi fa ho scritto una voce su di esso, quindi lascio il riferimento direttamente.
Corso C- con Mono - Dichiarazione di classi e strutture
Pubblicato: 4 Dicembre 2010/Sotto: Tutorial /Da: F. Javier Carazo Gil
CClassiCorsoStruttureScimmia

Curso C# con Mono – Declaración de clases y estructuras

A questo punto, abbiamo potuto parlare così profondamente e per così tanto tempo di classi, oggetti, metodi, attributi... che minerebbe il significato di questo corso. Immagino che tu abbia nozioni di base di questo intero paradigma di orientamento agli oggetti, e vedremo come implementare le classi in C- con Mono.

Potremmo dire in modo molto grossolano che:

Le classi sono l'unità di base della strutturatura in un programma C #
Le strutture sono un'ereditarietà di linguaggi meno recenti, ad esempio C, in cui creiamo un gruppo comune per varie variabili, senza attribuire loro possibilità di esecuzione (metodi)
Le strutture possono avere metodi di base per la lettura o la modifica delle variabili.

Come creare una classe?

Dico come creare e non come dichiarare una classe, perché il corso è basato su MonoDevelop e dopo aver creato una classe con l'interfaccia, vedremo come viene dichiarato nel codice.

Declaración de nueva clase con MonoDevelop

All'interno dello spazio dei nomi, a sinistra dell'interfaccia, facciamo: clic destro, "Aggiungi", "Nuovo file", "Classe vuota" e indicare il nome. Verrà creato uno scheletro di classe di base simile al seguente:

```
using System;

namespace holaMundo
{
	public class persona
	{
		public persona ()
		{
		}
	}
}
```

Ora diamo un'occhiata alla classe un po 'più farcito. Come potete vedere, sono all'interno dello stesso spazio dei nomi in cui abbiamo fatto gli esempi di cui sopra, "helloWorld". Sono inclusi: attributi, metodi per la gestione di attributi, costruttore e un metodo di esempio.

```
using System;

namespace holaMundo
{
	public class persona
	{
		// atributos
		string nombre;
		int edad;

		// propiedades: métodos para manejar los atributos
		public string Nombre {
			get { return nombre; }
			set { nombre = value; }
		}

		public int Edad {
			get { return edad; }
			set { edad = value; }
		}

		// constructor
		public persona (string _nombre, int _edad)
		{
			// usamos los métodos para manejar atributos en lugar de this.nombre
			Nombre = _nombre;
			Edad = _edad;
		}

		// método para imprimir datos de la instancia
		public void print()
		{
			System.Console.WriteLine("Nombre: " + nombre + " - Edad: " +
edad.ToString());
		}
	}
}
```

Anche se la sintassi sarà abbastanza familiare se avete lavorato con linguaggi come Java o C , c'è qualche curiosità che vorrei commentare:

Il getter e il setter, chiamati anche proprietà, sono metodi che interagiscono con gli attributi, senza utilizzare quelli tipici e fastidiosi: getVariable() e setVariable(value). Si integrano molto bene nel codice e quando si effettuano chiamate vecchio stile, qualcosa o qualcosa, Age,saranno chiamati automaticamente.

MonoDevelop crea automaticamente i casi come pubblici, inmodo che sia possibile accedervi dall'esterno dell'assembly.

Gli attributi di classe possono anche avere diversi livelli di accesso. Per impostazione predefinita, se non indicato è privato. Diamo un'occhiata alle differenze tra di loro:

public: accessibile da altre classi. Non consigliato, migliore privato e proprietà per accedere dall'esterno.
protected: rende l'attributo accessibile direttamente da altre classi derivate da quella in cui è dichiarato, ma non dal resto.
private: opzione predefinita. La variabile sarà accessibile solo dalla propria classe.
internal: accessibile solo dall'assieme stesso.
Come ho detto, ci sono molte altre opzioni e molto altro da commentare in questa sezione, ma se si conosce l'orientamento agli oggetti, con questo si avrà abbastanza.

Tuttavia, tramite commenti, risponderò a tutte le domande che possono sorgere :).

Strutture

Prima di terminare, faremo una breve introduzione a questo tipo di meccanismo, che usiamo per raggruppare diverse variabili intorno a un'unione comune, senza fornire alcun tipo di metodo per la loro gestione (questo li distingue direttamente dalle classi), ma a differenza dei linguaggi più vecchi, consentendo loro di avere variabili private e proprietà per la loro gestione.

Vediamo come vengono dichiarati e come vengono chiamati i loro membri:

```
public struct direccion{
	string calle;
	int numero;
	char letra;

	public string Calle{
		get { return calle; }
		set { calle = value; }
	}

	public int Numero{
		get { return numero; }
		set { numero = value; }
	}

	public char Letra{
```

```
                get { return letra; }
                set { letra = value; }
        }
}
```

Faremo chiamate alla proprietà, che possiamo cambiare se siamo interessati al codice di cui sopra con funzioni più complesse, chiamate a BB.DD., ecc. in breve, con quello di cui abbiamo bisogno.

```
direccion personal;
personal.Calle = "Avenida sin nombre";
personal.Numero = 6;
personal.Letra = 'C'
```

Corso C- con Mono - Dichiarazione di classi e strutture
Pubblicato: 4 Dicembre 2010/Sotto: Tutorial /Da: F. Javier Carazo Gil
CClassiCorsoStruttureScimmia

Curso C# con Mono – Declaración de clases y estructuras

A questo punto, abbiamo potuto parlare così profondamente e per così tanto tempo di classi, oggetti, metodi, attributi... che minerebbe il significato di questo corso. Immagino che tu abbia nozioni di base di questo intero paradigma di orientamento agli oggetti, e vedremo come implementare le classi in C- con Mono.

Potremmo dire in modo molto grossolano che:

Le classi sono l'unità di base della strutturatura in un programma C #
Le strutture sono un'ereditarietà di linguaggi meno recenti, ad esempio C, in cui creiamo un gruppo comune per varie variabili, senza attribuire loro possibilità di esecuzione (metodi)
Le strutture possono avere metodi di base per la lettura o la modifica delle variabili.

Come creare una classe?

Dico come creare e non come dichiarare una classe, perché il corso è basato su MonoDevelop e dopo aver creato una classe con l'interfaccia, vedremo come viene dichiarato nel codice.

Declaración de nueva clase con MonoDevelop

All'interno dello spazio dei nomi, a sinistra dell'interfaccia, facciamo: clic destro, "Aggiungi", "Nuovo file", "Classe vuota" e indicare il nome. Verrà creato uno scheletro di classe di base simile al seguente:

```
using System;

namespace holaMundo
{
        public class persona
        {
                public persona ()
                {
                }
        }
```

}
Ora diamo un'occhiata alla classe un po 'più farcito. Come potete vedere, sono all'interno dello stesso spazio dei nomi in cui abbiamo fatto gli esempi di cui sopra, "helloWorld". Sono inclusi: attributi, metodi per la gestione di attributi, costruttore e un metodo di esempio.

```
using System;

namespace holaMundo
{
	public class persona
	{
		// atributos
		string nombre;
		int edad;

		// propiedades: métodos para manejar los atributos
		public string Nombre {
			get { return nombre; }
			set { nombre = value; }
		}

		public int Edad {
			get { return edad; }
			set { edad = value; }
		}

		// constructor
		public persona (string _nombre, int _edad)
		{
			// usamos los métodos para manejar atributos en lugar de this.nombre
			Nombre = _nombre;
			Edad = _edad;
		}

		// método para imprimir datos de la instancia
		public void print()
		{
			System.Console.WriteLine("Nombre: " + nombre + " - Edad: " +
edad.ToString());
		}
	}
}
```

Anche se la sintassi sarà abbastanza familiare se avete lavorato con linguaggi come Java o C , c'è qualche curiosità che vorrei commentare:

Il getter e il setter, chiamati anche proprietà, sono metodi che interagiscono con gli attributi, senza utilizzare quelli tipici e fastidiosi: getVariable() e setVariable(value). Si integrano molto bene nel codice e quando si effettuano chiamate vecchio stile, qualcosa o qualcosa, Age,saranno chiamati automaticamente.

MonoDevelop crea automaticamente i casi come pubblici, inmodo che sia possibile accedervi dall'esterno dell'assembly.

Gli attributi di classe possono anche avere diversi livelli di accesso. Per impostazione predefinita, se non indicato è privato. Diamo un'occhiata alle differenze tra di loro:

public: accessibile da altre classi. Non consigliato, migliore privato e proprietà per accedere dall'esterno.
protected: rende l'attributo accessibile direttamente da altre classi derivate da quella in cui è dichiarato, ma non dal resto.
private: opzione predefinita. La variabile sarà accessibile solo dalla propria classe.
internal: accessibile solo dall'assieme stesso.
Come ho detto, ci sono molte altre opzioni e molto altro da commentare in questa sezione, ma se si conosce l'orientamento agli oggetti, con questo si avrà abbastanza.

Tuttavia, tramite commenti, risponderò a tutte le domande che possono sorgere :).

Strutture

Prima di terminare, faremo una breve introduzione a questo tipo di meccanismo, che usiamo per raggruppare diverse variabili intorno a un'unione comune, senza fornire alcun tipo di metodo per la loro gestione (questo li distingue direttamente dalle classi), ma a differenza dei linguaggi più vecchi, consentendo loro di avere variabili private e proprietà per la loro gestione.

Vediamo come vengono dichiarati e come vengono chiamati i loro membri:

```
public struct direccion{
	string calle;
	int numero;
	char letra;

	public string Calle{
		get { return calle; }
		set { calle = value; }
	}

	public int Numero{
		get { return numero; }
		set { numero = value; }
	}

	public char Letra{
		get { return letra; }
		set { letra = value; }
	}
}
```

Faremo chiamate alla proprietà, che possiamo cambiare se siamo interessati al codice di cui sopra con funzioni più complesse, chiamate a BB.DD., ecc. in breve, con quello di cui abbiamo bisogno.

```
direccion personal;
personal.Calle = "Avenida sin nombre";
personal.Numero = 6;
```

personal.Letra = 'C'
Corso C- con Mono - Conversione di tipi e informazioni culturali
Pubblicato: 11 Dicembre 2010/Sotto: Tutorial /Da: F. Javier Carazo Gil
.netCc taglienteConversioneCulturaleIcazaScimmiaParseTipiTostringa

Quando si lavora con diversi tipi di dati, è sempre consigliabile conoscere i meccanismi per la conversione di un tipo di dati in altri. Se a questo aggiungiamo, possiamo fare applicazioni in cui lavoriamo inserendo i dati dopo il formalismo spagnolo (virgole per separare i decimali, punti per separare le migliaia) o viceversa (si può vedere una mappa di dove ogni separatore decimale viene utilizzato qui).

Si tratta di ciò che viene chiamato informazioni culturali, che viene inserito all'interno dell'assembly: System.Globalization, e che a parte in questo argomento è conveniente conoscere per altre attività.

Casting

È il modo più semplice per eseguire una conversione dei dati. Viene ereditato direttamente da C e può essere implicito (se non specificato) o esplicito (quando lo facciamo). Ha grandi limitazioni perché fondamentalmente permette solo di non convertire, ma di mimetizzarsi, per una particolare istruzione il tipo di dati.

Diamo un'occhiata a un esempio:

```
int i = 10;
float f = 0;
f = i;  // conversión implícita
f = 0.5F;
i = (int)f;  // conversión explícita. Existe pérdida de información, los decimales
```

ToString Metodi

Come in molti altri linguaggi di programmazione, a questo livello di astrazione, esiste un metodo ToString,definito in ogni classe, che converte qualsiasi oggetto in una stringa. Se abbiamo un numero intero che ha un valore pari a 1 al suo interno, restituirà una stringa "1"; accadrà anche con un decimale, ma è qui che entra in gioco le informazioni culturali in gioco. Se abbiamo la cultura predefinita come lo spagnolo, mostrerà virgole e punti in base al nostro modo di farlo, se definiamo un'altra cultura predefinita, il comportamento cambierà di conseguenza. Possiamo anche cambiare le impostazioni cultura predefinite per un particolare punto in modo tempestivo.

Diamo un'occhiata a esempi di uso avanzato di ToString utilizzando gli identificatori, per visualizzare lo stesso numero per schermo con decimali, senza di essi, in formato valuta, con percentuale...

```
decimal valor = 16325.62m;
string especificador;

// Sin especificador
Console.WriteLine("Sin el especificador tenemos el valor: " + valor.ToString());
// Especificador numérico estándar
especificador = "G";
```

```
Console.WriteLine("Con el especificador: " + especificador + " tenemos el valor: " +
valor.ToString(especificador));
// Muestra: 16325,62
especificador = "C";
Console.WriteLine("Con el especificador: " + especificador + " tenemos el valor: " +
valor.ToString(especificador));
// Muestra: 16.325,62€
especificador = "E04";
Console.WriteLine("Con el especificador: " + especificador + " tenemos el valor: " +
valor.ToString(especificador));
// Muestra: 1,6326E+004
especificador = "F";
Console.WriteLine("Con el especificador: " + especificador + " tenemos el valor: " +
valor.ToString(especificador));
// Muestra: 16325,62
especificador = "N";
Console.WriteLine("Con el especificador: " + especificador + " tenemos el valor: " +
valor.ToString(especificador));
// Muestra: 16.325,62
especificador = "P";
Console.WriteLine("Con el especificador: " + especificador + " tenemos el valor: " +
valor.ToString(especificador));
// Muestra: 1.632.562,00%
```

Esaminiamo un esempio di utilizzo delle informazioni sulle impostazioni cultura (CultureInfo):

```
System.Globalization.CultureInfo culture  = new System.Globalization.CultureInfo("EN-us");
Console.WriteLine("Cambiando a cultura de inglés de Estados Unidos tenemos: " +
valor.ToString(culture.NumberFormat));
// Muestra: 16325.62
```

Esistono oggetti, che pur avendo un tale metodo, la stringa che restituiscono non contiene praticamente nulla delle informazioni dell'oggetto. Ad esempio, se abbiamo un DataTable e richiamare il ToString metodo, l'output è probabilmente solo che si tratta di un'istanza del DataTable tipo di oggetto.

Metodi Parse

Si potrebbe dire che le funzioni Parse sonole funzioni inverse per il ToString metodi. A differenza del primo, da una stringa, sono in grado di leggerne il contenuto e tradurlo in un tipo di dati specifico, ad esempio da stringa a numero intero.

In C c'erano alcune funzioni che hanno fatto lo stesso lavoro: atoi, atol.... Oltre ad essere veramente utile nelle interfacce grafiche e web, ricordate che i dati che gli utenti immettono di solito sono in formato stringa, dalla costituzione stessa di tali interfacce.

Sono anche molto utili quando si gestiscono tipi di dati che non è possibile eseguire il cast direttamente, poiché la soluzione più pratica consiste nel creare un ToString seguito da un Parse.

Diamo un'occhiata agli esempi:

```
string valor = "1";
int a = int.Parse(a);
Console.WriteLine("El valor de a es: " a.ToString());
```

Come potete immaginare, se vogliamo passare i dati decimali con virgole e punti in questo modo, dobbiamo anche utilizzare le informazioni culturali a questo punto.
Corso C- con Mono - Trattamento eccezioni
Pubblicato: 23 Dicembre 2010/Sotto: Tutorial /Di: F. Javier Carazo Gil
.netCc taglienteCorsoEccezioneScimmiascimmia sviluppare

OBJ

Una caratteristica comune di tali linguaggi di alto livello è la presenza di gestione delle eccezioni.

Che cos'è un'eccezione?

Un'eccezione è una situazione anomala che si verifica durante l'esecuzione del programma. Il gestore di eccezioni è una struttura di controllo che consente di eseguire un codice in base a tale situazione anomala in modo controllato.

Che tipo di azioni vengono eseguite di solito?

Da un lato abbiamo azioni di tipo terminazione. Invece di vedere un messaggio di errore del sistema operativo, il programma stesso lo rilascia informando ciò che finirà la sua esecuzione.

D'altra parte, abbiamo gestori che modificano il comportamento del programma a un certo punto. Per tali azioni, potremmo dire che le eccezioni rappresentano più una forma di controllo del flusso che una risorsa di emergenza.

Quali altri meccanismi esistono per questo stesso problema?

Esempi tipici di eccezioni sono:

Quando cerchiamo di dividere tra zero
Quando si tenta di chiamare un metodo di un'istanza che è null
Quando cerchiamo di convertire una stringa in numero e la stringa non ha un formato numerico
E così abbiamo potuto seguire un lungo eccetera di possibili eccezioni. Come si vedrà tutte queste eccezioni, potrebbero essere controllati come viene fatto in lingue che non dispongono di questo meccanismo: da condizionali ogni volta che c'è un'operazione che potrebbe non riuscire. Perché è meglio usare le eccezioni?

Vantaggi delle eccezioni

Perché complicarci la vita con un nuovo concetto se possiamo usare direttamente i controlli?

Separare il codice principale dal programma, dal trattamento degli errori
Propagare l'errore nello stack di esecuzione,
Raggruppare e classificare gli errori
Come intercettare e trattare un'eccezione con Mono e C?

Per intercettare un'eccezione in C- usiamo la seguente struttura (diamo un'occhiata con un esempio):

```
int a = 1;
int b = 0;

try{
```

```
        int c = a / b; // zona protegida
}catch(DivideByZeroException exc){
        Console.Write("Ha intentado dividir desde cero"); // código a ejecutar
}
```

Come potete vedere, stiamo parlando direttamente dell'eccezione di divisione zero. Se si desidera gestire qualsiasi tipo di eccezione, è necessario utilizzare solo T:System.Exception , la classe di eccezione da cui ereditano tutti gli altri, Exception. Possiamo anche combinarli in modo da poter gestire determinati tipi di eccezioni e qualsiasi altra eccezione che non abbiamo controllato.

Esaminiamo un caso di controllo per un particolare tipo di eccezione (il formato errato) e il controllo per qualsiasi altro caso:

```
string cadena = "2a";

try{
        int valor = int.Parse(cadena);
}catch(FormatException exc){
        Console.Write("Ha ocurrido una excepción por el formato del fichero. Detalles:\n" +
exc.Message);
}catch(Exception exc){
        Console.Write("Ha ocurrido otro tipo de error. Detalles:\n" + exc.Message);
}
```

Corso C- con Scimmia - Ereditarietà
Pubblicato: 4 Gennaio 2011/Sotto: Tutorial /Di: F. Javier Carazo Gil
CClassiCorsoEreditàScimmiaProgrammazione

Continuando con caratteristiche tipiche dei linguaggi orientati agli oggetti di alto livello, siarriva all'ereditarietà.

Che cos'è l'eredità?

Nella programmazione orientata agli oggetti, l'ereditarietà è il meccanismo di base per la creazione di nuove classi (figlie o derivate), in base alle classi precedenti (che chiameremo padre, base o superclasse). La relazione è simile a quella che si verifica in molte situazioni della vita quotidiana, ad esempio nel campo della biologia nelle relazioni padre-figlio.

Grazie a questo meccanismo, possiamo fare affidamento sulle classi precedenti, per creare nuove classi sul lavoro già fatto, in modo da estendere la funzionalità della classe padre a un nuovo campo, senza dover riscrivere tutto il contenuto di esso.

Esempi

Ci sono un sacco di casi come questo. Ad esempio, l'eccezione della classe padre fornisce metodi e attributi generali per tutte le possibili eccezioni. Ciò comporta eccezioni specifiche, ad esempio la divisione per zero, che oltre ai metodi e agli attributi esistenti, aggiungono funzionalità aggiuntive, ma riutilizzano tutte le funzionalità della classe padre.

Un altro esempio classico sono i controlli in un'interfaccia grafica. La classe del controllo è l'elemento padre di un intero intervallo di classi figlio. La classe del controllo presenta ad esempio

la situazione (tutti i controlli hanno situazione), ma ogni classe figlio fornisce nuove funzionalità (ad esempio, i propri eventi).

Implementazione

Esamineremo l'implementazione in C-C con un esempio specifico. Dal momento che i veicoli elettrici sono così di moda, stiamo andando a implementare una serie di classi, con patrimonio, che riflettono la seguente realtà:

Classe padre: motore, avrà le caratteristiche di tutti i tipi di motori, gestiremo la potenza (integer in kW) e la coppia (decimale in Nm)
Classe figlio: elettrica, conterrà informazioni esclusive di motori elettrici, intensità (integer in amplificatori), potenziale (intero in volt)
Classe figlio: combustione interna, gestirà le informazioni su questo tipo di motore. Spostamento (integer in cc.), numero di cilindri (integer) ed emissioni di CO2 (decimale in gr/km)

```
public class Motor{
	int potencia;
	decimal par;

	public int Potencia {
		get { return this.potencia; }
		set { potencia = value;	}
	}

	public decimal Par {
		get { return this.par; }
		set { par = value; }
	}
}

public class Electrico : Motor{
	int intensidad;
	int potencial;

	public int Intensidad {
		get { return this.intensidad; }
		set { intensidad = value; }
	}

	public int Potencial {
		get { return this.potencial; }
		set { potencial = value; }
	}
}

public class CombustionInterna : Motor{
	int cilindrada;
	int cilindros;

	public int Cilindrada {
		get { return this.cilindrada; }
		set { cilindrada = value; }
```

```
        }

        public int Cilindros {
                get { return this.cilindros; }
                set { cilindros = value; }
        }
}
```

Come si può vedere l'implementazione è come qualsiasi altra classe, ma l'aggiunta dei due punti e il nome della classe padre quando si dichiara alla classe figlio. Poi possiamo effettuare chiamate del tipo:

```
Electrico a = new Electrico();
a.Potencia = 100;
```

Classi astratte

Se includiamo la parola riservata astratta quando si dichiara la classe padre: classe astratta Motor; stiamo dicendo Monkey che quella classe non sarà in grado di essere creata un'istanza. Vale a dire, è possibile creare classi derivate e crearne un'istanza, ma non creare mai oggetti della classe padre (in questo caso, è possibile creare Electric o CombustionInterna oggetti, ma non è stato possibile creare un'istanza Motor).

Metodi virtuali

Se dichiariamo un metodo nella classe padre, nella classe figlio non possiamo dichiararlo nuovamente a meno che non sia del tipo virtuale nella classe padre. A tale scopo, nella dichiarazione del metodo nell'elemento padre si userebbe la seguente struttura:

```
public virtual void mostrarDatos()
```

E nell'elemento figlio, dichiarando il metodo sarebbe fare uso della parola chiave override:

```
public override void mostrarDatos()
```

Corso C- con Mono - Interfacce
Pubblicato: 24 Gennaio 2011/Sotto: Tutorial /Di: F. Javier Carazo Gil
CClassiCorsoInterfacceScimmiaProgrammazione

Se l'ultima volta che abbiamo parlato di C, stavamo commentando l'eredità,oggi parleremo delle interfacce.

Interfacce nei linguaggi di programmazione orientati agli oggetti

Conoscete tutti le interfacce utente grafiche. Sono il meccanismo software che ci permette di interagire più o meno intuitivamente con le funzionalità fornite dal programma. Vale a dire, è un meccanismo per comunicare diversi livelli, nascondendo i dettagli dell'implementazione.

Questo è ciò che le interfacce sono in linguaggi di programmazione orientata agli oggetti, ad esempio C- o Java. Si tratta di un set di metodi, delegati o eventi, che non hanno alcuna implementazione. Lo sviluppo della funzionalità viene eseguito nella classe che implementa l'interfaccia . È un concetto simile ai prototipi di funzione in linguaggi come C.

affermazione

Seguendo l'esempio dei motori, illustrato nella consegna precedente dedicata all'ereditarietà,verrà creata un'interfaccia IMotor, a cui si farà riferimento dall'implementazione della classe. Ho incluso un altro metodo per rendere le possibilità più chiare.

```
interface IMotor
{
	// declaración de propiedades
	int Potencia
	{
		get;
		set;
	}
	decimal Par
	{
		get;
		set;
	}

	// declaración de métodos
	void printInfo();
	// declaración de eventos
}
public class Motor : IMotor
{
	int potencia;
	decimal par;
	public int Potencia {
		get { return this.potencia; }
		set { potencia = value;		}
	}
	public decimal Par {
		get { return this.par; }
		set { par = value; }
	}
	public void printInfo()
	{
		Console.WriteLine("La potencia es " + Potencia.ToString() + "
	   y el par: " + Par.ToString());
	}
}
```

Come si può vedere, la dichiarazione di interfacce è molto simile a quella di una classe, inclusa la parola chiave Interface. La chiamata all'interfaccia dalla classe che la implementa viene effettuata come se si tratta di un'ereditarietà.

Le interfacce supportano anche l'ereditarietà da sole. L'unico dettaglio da ricordare è che in caso di conflitto sulla duplicazione dei nomi (un'interfaccia eredita da altre interfacce in cui è presente un metodo con lo stesso nome in entrambi), deve essere fatto riferimento esplicito a cui viene chiamato uno:

```
InterfaceA.method()
InterfaceB.method()
```

A cosa sono?

Più di quello per cui sono, ho intenzione di commentare ciò che li uso per nei miei sviluppi. Anche se non ha molto senso per le classi di piccole dimensioni farlo, quando gli oggetti raccolgono una certa dimensione, le interfacce rendono il controllo degli errori molto più semplice. Ad esempio, se abbiamo dimenticato di definire un metodo o se sono presenti conflitti di tipi di dati, il compilatore ci invia una notifica. Ricordare quanto sia conveniente e immediato dichiarare direttamente le interfacce, anziché le classi, con tutto lo sviluppo coinvolto.

Dopo tutto, è un altro modo per creare un maggiore grado di astrazione e per essere in grado di sfruttare meglio le possibilità offerte dall'orientamento degli oggetti.