DELEGATES + LAMBDA

CORSO DI GAME PROGRAMMING
1° ANNO

Docente **Davide Caio**

ACCADEMIA ITALIANA VIDEOGIOCHI

SIGNATURE DI UN METODO

Dato un metodo, si definisce **SIGNATURE** o **FIRMA** del metodo, la definizione delle seguenti informazioni:

- Nome del metodo;
- Tipo di ritorno;
- Tipi delle variabili di input;
- Ordinamento delle variabili di input;

Una signature identifica univocamente il tipo di metodo.
Due metodi con nomi diversi hanno SIGNATURE COMPATIBILI se gli ultimi 3 punti descritti in precedenza sono uguali (tipo di ritorno, tipo e ordinamento delle variabili di ingresso)

POLIMORFISMO

Il polimorfismo permette al codice che scriviamo di essere “**dinamico**”, ovvero di assumere comportamenti diversi in base al contesto di esecuzione, in maniera non definita a compile time ma a Runtime.

Abbiamo visto come un modo per implementare il polimorfismo nella programmazione ad oggetti è l’uso dell’ereditarietà e delle interfacce.

Due classi che derivano dalla stessa interfaccia, definiscono le stesse funzionalità con implementazione diverse.

Overridando metodi di un’interfaccia (o in generale di una classe base) poniamo quindi le fondamenta per l’ utilizzo del polimorfismo all’ interno del nostro codice.

POLIMORFISMO e EREDITARIETA’

Il “problema” di utilizzare esclusivamente l’ ereditarietà per implementare il polimorfismo, è che per definire un nuovo comportamento polimorfo dobbiamo **ogni volta definire una nuova classe**.

Detto in altri termini classi diverse hanno comportamenti diversi (tramite ereditarietà e override dei metodi) ma *DUE OGGETTI ESATTAMENTE DELLA STESSA CLASSE hanno esattamente lo stesso comportamento*.

Quello che vorremmo è fare in modo che I metodi di una classe abbiamo un comportamento polimorfo senza dover creare classi derivate da quest’ ultima.

DELEGATES

Quello dei DELEGATES è un potente strumento che permette, tra le altre cose, di implementare il comportamento polimorfo dei metodi di una classe, in maniera del tutto dinamica.

Un Delegato è sostanzialmente un OGGETTO che si riferisce ad una (o più) funzione(i).

Mentre per quel che riguarda i dati, un tipo di riferimento è una variabile che fa riferimento a un dato in memoria, **un delegates è un oggetto che fa riferimento a una funzione/metodo**.

In altre parole, è come se fosse un riferimento a del codice (che anch'esso è ovviamente caricato in memoria).

DELEGATES - UTILIZZI

I delegate sono strumenti potentissimi e permettono, tra le altre cose, di :

- Realizzare metodi polimorfici
- Passare un riferimento a una funzione in un metodo
- Eseguire metodi anonimi (vedremo cosa sono)
- Realizzare codice basato su eventi (ne vedremo una implementazione in Aiv.Fast2D.Component)

DELEGATES - COME SI USA

Per poter utilizzare un delegate esso va:

- **DICHIARATO**: in questo contesto dichiariamo il nome del delegato e la *firma* delle funzioni a cui esso può fare riferimento.

- **ISTANZIATO**: creiamo un oggetto della classe delegate utilizzando il costruttore, nel quale passiamo la funzione a cui dovrà riferirsi
- **INVOCATO**: invocando l'istanza del delegato non facciamo altro che eseguire le funzioni ad esse collegate (quelle a cui fa riferimento).

DELEGATES - DICHIARAZIONE

Un delegate si definisce in questo modo:

<MODIFICATORE> delegate <TIPO_RITORNO><NOME_DELEGATO>(<PARAMETRI>)

Esempio

public delegate int MyDelegate (int x, int y, string s);

In questo modo abbiamo creato un nuovo tipo nel nostro programma, chiamato MyDelegate, che rappresenta un oggetto in grado di fare riferimento a un qualsiasi metodo che abbia come tipo di ritorno interi, e accetta come parametri in input due interi e una stringa.

DELEGATES - ISTANZIARE

Quando istanziamo un delegato, stiamo creando un oggetto della classe System.delegate. Questo oggetto, come già detto, è in grado di avere dei riferimenti a delle funzioni.
Come si istanza?

```
public delegate void PrintStringDelegate(string s); //DICHIARAZIONE DELEGATE

1 reference
public static void StampaStringa(string s) { //METODO CHE CORRISPONDE ALLA FIRMA DEL DELGATE DICHIARATO
    Console.WriteLine(s);
}

0 references
static void Main(string[] args) {

    PrintStringDelegate myDelegate = new PrintStringDelegate(StampaStringa); //CREAZIONE DI UN OGGETTO DELEGATE
                                                                              //DA QUESTO MOMENTO IN POI, LA VARIABILE
                                                                              //myDelegate RIFERISCE AL METODO STAMPA STRINGA

}
```

DELEGATES - INVOCAZIONE

Sempre facendo riferimento al codice della slide precedente, come si invoca un delegate (cioè si esegue le funzioni a cui fanno riferimento)?

```
myDelegate?.Invoke("Ciaone"); //COSì INVOCO (solo se myDelegate non è null, grazie all'operatore ?) TUTTE LE FUNZIONI
                              //ASSOCIATE AL MIO OGGETTO myDelegate, PASSANDOGLI COME PARAMETRO LA STRINGA "Ciaone"
```

Ad un oggetto delegate posso aggiungere più funzioni di riferimento, attraverso l'operatore +=. Quando invocherò il delegate, saranno eseguite tutti i metodi a cui farà riferimento.

```
    myDelegate += StampaStringaReverse;
    myDelegate?.Invoke("Ciaone"); //A QUESTO PUNTO BEN DUE METODI SARANNO INVOCATI. STAMPA STRINGA CHE STAMPERA' A CONSOLE "Ciaone"
                                  //E STAMPA STRINGA REVERSE CHE STAMPERA' "enoaiC"
}



1 reference
public static void StampaStringaReverse (string s) {
    char[] array = s.ToCharArray();
    Array.Reverse(array);
    Console.WriteLine(array);

}
```

DELEGATES - Classe System.Delegate

Quando istanziamo un delegato, stiamo creando un oggetto della classe System.Delegate.

Questa classe ovviamente è SEALED ovvero non è possibile derivarla in alcun modo.

Leggetevi la documentazione:

https://docs.microsoft.com/it-it/dotnet/csharp/programming-guide/delegates/

Riferimento a un metodo di istanza

Nell'esempio abbiamo visto come aggiungere una reference al delegato un metodo statico.
Ovviamente un qualsiasi metodo può essere referenziato da un delegato, anche un metodo istanza di un oggetto.

```
class Program {

    public delegate void PrintStringDelegate(string s); //DICHIARAZIONE DELEGATE

    1 reference
    public static void StampaStringa(string s) { //METODO CHE CORRISPONDE ALLA FIRMA DEL DELEGATE DICHIARATO
        Console.WriteLine(s);
    }

    0 references
    static void Main(string[] args) {
        Dummy classObj = new Dummy();
        PrintStringDelegate myDelegate = new PrintStringDelegate(StampaStringa);
        myDelegate += classObj.DummyMethod;
    }
}

2 references
class Dummy {
    1 reference
    public void DummyMethod (string s) { //METODO CHE CORRISPONDE ALLA FIRMA DEL DELEGATE
        //faccio cose
    }
}
```

Passaggio di delegates come parametri

Ovviamente possiamo passare un oggetto delegate anche come parametro di un metodo.
Definiamo un delegate e una classe con un metodo che accetta come parametro quel delegate

```
public delegate float OperationDelegate(float f1, float f2); //DICHIARAZIONE DELEGATE

3 references
class OperationClass {
    float op1;
    float op2;

    1 reference
    public OperationClass(float op1, float op2) {
        this.op1 = op1;
        this.op2 = op2;
    }

    2 references
    public float DoOperation(OperationDelegate OD) {
        return OD.Invoke(op1, op2);
    }

}
```

Passaggio di delegates come parametri

Cosa stampa a console questo main? (Con riferimento a classe e delegate della slide precedente?)

```
class Program {

    1 reference
    public static float Sum(float x, float y) {
        return x + y;
    }

    1 reference
    public static float Product (float x, float y) {
        return x * y;
    }

    0 references
    static void Main () {
        OperationDelegate OPDelegate = new OperationDelegate(Sum);
        OperationClass myClass = new OperationClass(3f, 7f);
        Console.WriteLine(myClass.DoOperation(OPDelegate));
        OPDelegate = new OperationDelegate(Product);
        Console.WriteLine(myClass.DoOperation(OPDelegate));
    }

}
```

Definizione metodo “OnTheFly”

Una cosa molto, molto scomoda è che, dobbiamo per forza definire da qualche parte il metodo da assegnare al delegate.

Se ad esempio questo metodo deve essere assegnato solo a quel delegate e non sarà utilizzato mai più in nessun’altra parte del mio software, io devo comunque dichiarare un metodo in una classe (che poi sia statico, o istanziato non fa differenza).

Se si potesse definire “OnTheFly” il codice che verrà eseguito a seguito all’invocazione del delegato?

Due possibilità:

- Funzioni Anonime
- Lamda Expressions

Funzioni Anonime

Una funzione anonima è una funzione creata “al volo” e passata a un delegato.

Come si definisce una funzione anonima?

```
delegate (parametri) {
      //corpo della funzione
}
```

Attenzione che la keyword delegate in questo caso NON VA CONFUSA CON QUELLA CHE UTILIZZIAMO PER DEFINIRE UN DELEGATO. In questo contesto ha un significato differente.
In questo modo POSSIAMO PASSARE DEL CODICE COME PARAMETRO DI UN METODO

Funzioni Anonime date in pasto a un delegate

```
public delegate float OperationDelegate(float f1, float f2); //DICHIARAZIONE DELEGATE

0 references
class Program {
    0 references
    static void Main() {
        float a = 3;
        float b = 7;


        OperationDelegate OPDelegate = new OperationDelegate(delegate (float x1, float x2) {
            return x1 * x2;
        });
        Console.WriteLine(OPDelegate.Invoke(a, b));

        OPDelegate = new OperationDelegate(delegate (float x1, float x2) {
            return x1 + x2;
        });

        Console.WriteLine(OPDelegate.Invoke(a, b));
    }

}
```

Lambda Expressions

Le Funzioni Anonime(keyword delegate) attualmente non sono molto utilizzate, poiché hanno una notazione molto “pesante” soprattutto quando dobbiamo passare una funzione anonima come parametro ad un’altra funzione.

Attualmente la modalità più utilizzata per definire codice “OnTheFly” sono le LAMBDA EXPRESSIONS.

Hanno esattamente la stessa funzionalità e potenza espressiva delle Funzioni Anonime, ma si scrivono più velocemente.

Lambda Expression: (parametri) => {corpo della funzione};

equivalente a:

Funzione anonima: delegate(parametri){corpo della funzione};

Lambda Expressions - esempio

Rivediamo lo stesso programma della slide 17, ma scritto con le lambda:

```
0 references
class Program {
    0 references
    static void Main() {
        float a = 3;
        float b = 7;


        OperationDelegate OPDelegate = new OperationDelegate((float x1, float x2) => {
            return x1 * x2;
        });
        Console.WriteLine(OPDelegate.Invoke(a, b));

        OPDelegate = new OperationDelegate((float x1, float x2) => {
            return x1 + x2;
        });

        Console.WriteLine(OPDelegate.Invoke(a, b));

        Console.ReadLine();
    }

}
```

Delegates Anonimi: Func ed Action

Dover definire sempre il delegato prima di istanziarlo (cioè creare un oggetto di quel delegato) è molto scomodo. Richiede scrivere molto codice.

C# offre due tipi di delegati Anonimi, cioè due modi per creare un oggetto di tipo delegato, che può avere dei riferimenti a delle funzioni, dichiarando la firma delle funzioni direttamente nel delegato.

Func: delegato anonimo che prevede un tipo di ritorno:

Func<ParametroIngresso1,ParametroIngresso2,.....,TipoParamRitorno>

Action: delegato anonimo che NON prevede tipo di ritorno

Action<TipoIngresso1,TipoIngresso2,...>

Func : esempio

```
0 references
class Program {

    public delegate float OperationDelegate(float f1, float f2); //DICHIARAZIONE ESPLICITAMENTE

    2 references
    public static float Sum(float f1, float f2) {
        return f1 * f2;
    }

    0 references
    static void Main() {
        OperationDelegate noramlDelegateObj = new OperationDelegate(Sum);

        Func<float, float, float> anonymousDelegate = new Func<float, float, float>(Sum);
    }

}
```

Action: esempio

```
0 references
class Program {

    public delegate void PrintString(string s); //DICHIARAZIONE ESPLICITAMENTE

    2 references
    public static void PrintConsole(string s) {
        Console.WriteLine(s);
    }

    0 references
    static void Main() {
        PrintString noramlDelegateObj = new PrintString(PrintConsole);

        Action<string> anonymousDelegate = new Action<string>(PrintConsole);
    }

}
```