GENERICS

CORSO DI GAME PROGRAMMING
1° ANNO

Docente **Davide Caio**

ACCADEMIA ITALIANA VIDEOGIOCHI

IL PROBLEMA

Voglio creare un nuovo tipo di collezione all'interno del mio software. Un array circolare.

Un array circolare è un array nel quale l'ultimo elemento dell'array è seguito dal primo elemento dell'array stesso.

Come faccio a costruire una classe del genere, sapendo che un array è un insieme di elementi di qualsiasi tipo? Ma ancor più specificatamente, come faccio a costruire questa classe senza dover ogni volta che lavoro con essa, effettuare il cast al corretto tipo della classe (quindi senza usare la reflection, no GetType ()).

GENERICS

In C# i **generics** introducono il concetto di **parametri di tipo**, che rendono possibile la progettazione di classi e metodi che rinviano la specifica di uno o più tipi fino a quando la classe o il metodo non viene dichiarato e ne viene creata un'istanza dal codice client.

Quali sono quindi i 3 tipi di parametri che abbiamo visto fino ad ora?

In questo modo l'array circolare sarebbe costruito con un unico generics **T** che ne definirà a run time il tipo degli elementi che lo comporranno.

```
public class CircularArray<T> {

//Descrizione della classe

}
```

GENERICS

Il parametro di tipo **T** viene usato in diverse posizioni in cui un tipo concreto viene normalmente usato. In particolare, viene usato nei seguenti modi:

- Come **parametro di un metodo** all'interno della classe in cui viene definito.
- Come **tipo restituito** da una proprietà o da un metodo all'interno della classe in cui viene definito.
- Come **tipo di un campo** della funzione stessa

Il parametro generico ha effetto all'interno della classe, e delle classi annidate.

ESEMPIO GENERICS

```
public class TestGeneric<T> {

        private T myData; //usato come parametro di un campo della classe


        public void SetMyData (T newData) {  //usato come parametro di un metodo della classe
                myData = newData;
        }

        public T GetMyData () { //usato come tipo di ritorno di un metodo o proprietà
                return myData;
        }

        private class TesNestedClass () {

                private T nestedData;  //usato come campo di una classe innestata nella classe che
        contiene il      generic.
        }

}
```

ESERCIZIO 1

Creare la classe CircularArray che rappresenta un array circolare.

Sostanzialmente deve avere un array di dati del tipo definito dall'utente e deve trattare questo array come se fosse un array circolare dove cioè all'ultimo elemento segue il primo.

Deve avere un costruttore in cui gli viene passata la dimensione di questo array, un metodo per settare un elemento dell'array passandogli un indice e un metodo che ritorna l'elemento in una determinata posizione passata come parametro.

TIPS: Pensare bene a cosa vuol dire "*all'ultimo elemento segue il primo*"

GENERICS MULTIPLI

Ovviamente è possibile utilizzare più generics in una singola classe.

Es: la classe Dictionary è formata da due generics, che vengono definiti come TKey e TValue.

In una classe con N generics, ognuno di essi può essere utilizzato per definire le stesse cose che possono essere definite con un solo generics.

ESEMPIO DI DEFINITIZIONE

public class MyDictionary <**TKey, TValue**> {

}

GENERICS VINCOLI

Abbiamo visto come i generics sono un sinonimo per dire che in T rappresenta un qualsiasi tipo di C# (sia nativo che definito dall'utente).

PROBLEMA

Ma se voglio dire qualcosa come, ok voglio creare la mia classe con un tipo generico ma questo tipo non può essere un qualsiasi tipo nel mio programma ma solamente un sottoinsieme di tipi.

SOLUZIONE

La clausola **Where** ti permette di specificare che tipi di tipi sono ammessi durante la creazione della classe con quel generic.

GENERICS VINCOLI

Vincolo	Descrizione
`where T : struct`	L'argomento di tipo deve essere un tipo di valore non nullable. Per informazioni sui tipi di valore Nullable, vedere tipi di valore Nullable. Poiché tutti i tipi di valore hanno un costruttore senza parametri accessibile, il vincolo `struct` implica il vincolo `new()` e non può essere combinato con il vincolo `new()`. Non è inoltre possibile combinare il vincolo di `struct` con il vincolo di `unmanaged`.
`where T : class`	L'argomento tipo deve essere un tipo riferimento. Questo vincolo si applica anche a qualsiasi tipo di classe, interfaccia, delegato o matrice.
`where T : notnull`	L'argomento di tipo deve essere un tipo non nullable. L'argomento può essere un tipo di riferimento non nullable in C# 8,0 o versione successiva oppure un tipo di valore non nullable. Questo vincolo si applica anche a qualsiasi tipo di classe, interfaccia, delegato o matrice.
`where T : unmanaged`	L'argomento di tipo deve essere un tipo non gestitoche non ammette i valori null. Il vincolo `unmanaged` implica il vincolo `struct` e non può essere combinato con i vincoli `struct` o `new()`.
`where T : new()`	L'argomento tipo deve avere un costruttore pubblico senza parametri. Quando il vincolo `new()` viene usato con altri vincoli, deve essere specificato per ultimo. Non è possibile combinare il vincolo `new()` con i vincoli `struct` e `unmanaged`.
`where T :` *<nome della classe base >*	L'argomento tipo deve corrispondere alla classe di base specificata o derivare da essa.
`where T :` *<nome dell'interfaccia >*	L'argomento tipo deve corrispondere all'interfaccia specificata o implementare tale interfaccia. È possibile specificare più vincoli di interfaccia. L'interfaccia vincolante può anche essere generica.
`where T : U`	L'argomento tipo fornito per T deve corrispondere all'argomento fornito per U o derivare da esso.

GENERICS LETTURE CONSIGLIATE

Come ulteriore lettura ecco tutti i link di tutti i costrutti che possono essere definiti come generici in C#:

- Classi Generiche
- Interfacce Generiche
- Metodi Generici

SPACE SHOOTER - GENERICS

In quale parte del nostro engine component based abbiamo un enorme bisogno di utilizzare i Generics?

Implementiamo questa parte.

ESERCIZIO 2

Implementare la classe MyList come la classe List di System.Collection.Genercs.

ESERCIZIO 3

Creare una classe generica Box che può essere inizializzata con un tipo qualsiasi IComparable e la cui funzione è immagazzinare un valore di quel tipo. Sovrascrivere il metodo ToString() di modo che stampi il tipo e il valore del dato immagazzinato nell'istanza nel formato {class full name: value}

ESERCIZIO 4

Creare un metodo che riceve come argomento una collezione che implementa il ciclo foreach e un elemento del tipo della lista. Il metodo ritorna il numero degli elementi della lista che sono maggiori del valore passato come parametro.