# Esercitazione di Fine Settimana – Week 2

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Nome | Roberta |
|  |  | Cognome | Beretta |
|  |  | Data | 17/09/2021 |

Leggete attentamente ogni domanda e argomentare quanto più possibile fornendo anche degli esempi.

1. Descrivere le modalità per ritornare più valori da un metodo in C#

Abbiamo visto tre modi per far ritornare ad una funzione più valori: keyword out, t-uple, restituire una classe.

Nel primo caso possiamo inserire nella firma del metodo un paramentro marcato con la keyword out che sarà un parametro calcolato nel metodo e restituito in uscita.

Un esempio di questo utilizzo è con la funzione TryParse.

Quando chiediamo ad un utente di inserire un valore intero ad esempio, possiamo utilizzare int.TryParse che ci restituisce un booleano che risulta essere vero se e solo se l’inserimento è corretto. In tal caso avremo in uscita il numero intero che è stato inserito tramite il parametro marcato con out, altrimenti tale parametro conterrà un valore di default.

Possiamo anche decidere che il metodo restituisca una t-upla ovvero una sequenza ordinata di parametri scelti da chi scrive il metodo.

Supponiamo di voler costruire un metodo che restituisca nome di una persona, data di nascita e numero di fratelli. Avrò quindi la seguente terna in uscita (string nome, DateTime dataNascita, int numeroFratelli).

Infine possiamo creare una classe che abbia come proprietà e/o campi i dati che ci interessano e restituirla in output al metodo.

Riprendendo l’esempio del punto precedente, potremmo costruire la classe Persona con le proprietà Nome, DataNascita e NumeroFratelli. Una volta acquisiti i dati nel metodo posso restituirla in output.

1. Descrivere le due tipologie di casting tra tipi in C#

Ci sono due tipologie di casting, quello implicito e quello esplicito.

Il casting implicito è quello che avviene in automatico perchè non si ha perdita di dati e dunque non c’è bisogno di dichiararlo. Ad esempio se voglio convertire un int in un long non ho bisogno di dichiararlo esplicitamente perchè il formato long è piu “capiente” del formato int.

Se volessi invece fare il viceversa dovrei dichiararlo esplicitamente, è questo il caso appunto del casting esplicito.

La sintassi in questo caso è la seguente:

double x = 100.5;

int n =(int)x;

Abbiamo visto che il casting lo si può definire anche per le classi e non solo per i tipi di base.

E’ sufficiente implementare nella classe prescelta l’overload del casting implicito o esplicito che sia.

1. Quali sono gli utilizzi della keyword static?

Abbiamo visto la keyword static applicata alle classi e/0 ai membri di una classe.

Una classe è statica quando non può essere istanziata, ovvero non è possibile creare oggetti di quella data classe.

Se una classe è statica tutti i suoi membri devono essere statici e, dal momento che non possiamo istanziarla, tutti i membri sono riferiti al concetto cenerale di classe.

Anche una classe non statica può contenere membri statici.

In questo caso le istanze di classi non statiche non possono vedere i membri statici che sono visti solo dal tipo classe.

1. Descrivere le modalità di implementazione di un evento in C#

Un evento è un membro di una classe che serve a mandare notifiche all’eserno della classe e, nello specifico, ad altre classi che si sono “iscritte” a quell’evento.

Per implementare un evento all’interno di una classe bisogna prima di tutto inserirlo appunto tra i membri della classe e marcarlo con la keyword event e specificando nella firma dell’evento il delegato che si è scelto per l’evento.

Un delegato è un tipo di dato che rappresenta una famiglia di funzioni aventi la stessa firma.

1. A cosa serve l’interfaccia IEnumerable<T>? Come si implementa in una nostra classe?

L’interfaccia IEnumerable<T> è una interfaccia che eredita da IEnumerable.

Quest ultima permette di ciclare sulle classi che la implementano, la prima invece si riferisce al generic <T>.

Abbiamo visto che quanto un oggetto implementa l’interfaccia IEnumerable o la corrispettiva Generics è possibile utilizzare su di esso LinQ, ovvero un framework che permette di utilizzare query sui dati dell’oggetto.

Quando vogliamo marcare una classe come IEnumerable<T> è sufficilente mettere dopo il nome della classe : IEnumerable<T>, dove T ad esempio può essere un’altra interfaccia.

**Esercitazione Pratica**

* Realizzare una classe Warehouse per gestire un Magazzino Merci, con le seguenti proprietà:
  + *Id Magazzino (GUID)*
  + *Indirizzo*
  + *Importo Totale Merci in giacenza*
  + *Data Ultima Operazione*
  + *Lista delle Merci in giacenza*
* Realizzare **l’overload degli operatori + e –** in modo che sia possibile aggiungere e rimuovere Merci dalla lista (l’overload dovrà anche occuparsi di aggiornare l’Importo Totale e la Data di Ultima Operazione)
* Realizzare un metodo StockList() che stampi i dati del Magazzino, inclusa la lista delle Merci in giacenza
* Realizzare una gerarchia di classi per rappresentare le Merci (Good). Tutte le classi avranno le proprietà
  + *Codice Merce*
  + *Descrizione*
  + *Prezzo*
  + *Data di Ricevimento*
  + *Quantità in Giacenza*
  + Realizzare le classi che rappresentano:
    - ElectronicGood, con la proprietà aggiuntiva *Produttore*
    - PerishableGood, con le proprietà aggiuntive *Data di Scadenza* e *Modalità di Conservazione* (enum con i valori FREEZER, FRIDGE e SHELF)
    - SpiritDrinkGood, con le proprietà aggiuntive *Tipo* (enum con i valori WHISKY, WODKA, GRAPPA, GIN e OTHER) e *Gradazione Alcoolica*
  + Tutte le classi saranno dotate di costruttore che accetti tutti i parametri necessari per popolare le proprietà
  + Tutte le classi dovranno implementare la propria versione del metodo ToString() e visualizzare tutti i dati
  + ***OPZIONALE***: realizzare una procedura di caricamento dati da un file della lista delle Merci in giacenza (l’implementazione di eventi per notificare le fasi di caricamento dati costituisce un bonus)
  + ***OPZIONALE 2***: utilizzare una (o più) eccezione custom per gestire tutte le tipologie di errori che dovessero verificarsi durante l’utilizzo della classe Warehouse e delle classi Good
* Realizzare una Console app che
  + Crei un nuovo Magazzino
  + Permetta di ricevere diverse tipologie di Merci (gestire l’input dall'utente)
  + Stampi i dati del Magazzino e le Merci in giacenza