# Esercitazione di Fine Settimana – Week 2

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Nome | Mauro |
|  |  | Cognome | Abozzi |
|  |  | Data | 17/9/2021 |

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#
2. Descrivere le due tipologie di casting tra tipi in C#
3. Quali sono gli utilizzi della keyword static?
4. Descrivere le modalità di implementazione di un evento in C#
5. A cosa serve l’interfaccia IEnumerable<T>? Come si implementa in una nostra classe?

**1) Le modalità sono 3: return di una classe con al suo interno tutti i valori necessari, tramite tuple da C# 7 in poi (esempio di tupla: (double, int) tupla = (5.0, 2);) , utilizzando la parola chiave out (utilizzato ad esempio nel TryParse, int.TryParse(myString, out int i) restituisce un booleano e l’intero i che è il risultato della conversione).**

**2) I tipi di casting sono implicito ed esplicito. Il casting implicito non richiede di specificare il tipo a cui l’oggetto verrà convertito, si usa quando si è sicuri che non c’è perdita di informazione, ad esempio da int a double (int a = 1; double b = a;). Nel casting esplicito si specifica tra parentesi il tipo a cui deve essere convertito l’oggetto, si usa quando c’è perdita di informazione (ad esempio da double a int in cui si perderebbe la parte decimale double a = 4.5; int b = (int) a;).**

**3) La keyword static può essere utilizzata su una classe per dichiarare che tutti i suoi membri sono statici, questa classe non può essere istanziata (non si può usare il new, si accede ai membri tramite il nome della classe). Una classe non statica può avere membri statici che verranno richiamati con la sintassi NomeDellaClasse.Membro.**

**4) L’evento viene dichiarato nel Publisher tramite la parola chiave event che è un (public event EventHandler Evento; //dove EventHandler è un delegate), qui viene anche controllato se ci sono subscriber all’evento, il publisher quindi notificherà ai subscriber. I subscriber si sottoscriveranno all’evento tramite la sintassi subscriber.Evento += subscriber\_evento; Nel subscriber si andrà quindi ad implementare ciò che succederà allo scatenarsi dell’evento (ad esempio un messaggio tramite Console).**

**5) L’interfaccia IEnumerable<T> è utilizzata da oggetti che contengono elementi uno dopo l’altro. Il tipo generico T permette di scrivere classi indipendenti dal tipo. Viene implementata con la sintassi: public class MyClass<T> : IEnumerable<T> (si può aggiungere where T:MyClass2 se T vogliamo che sia un particolare tipo). La classe MyClass deve implementare i metodi public IEnumerator GetEnumerator() e public IEnumerator<T> GetEnumerator() che supportano l’iterazione sulla collection.**

**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