# Esercitazione di Fine Settimana – Week 2

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Nome | Laura |
|  |  | Cognome | Martines |
|  |  | 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#

Ci sono tre diverse modalità per ottenere più valori di ritorno da un metodo in C# (sebbene di default esso ne possa restituire solo uno). Il primo metodo è la parola chiave out, che messa davanti al nome di un parametro in ingresso, fa si che esso venga restituito (modificato) in uscita. É poi possibile far restituire al metodo una struttura (o una classe) che contenga tutte le variabili che si vuole ottenere in uscita. Il terzo metodo è quello delle tuple, che sono un elenco ordinato di variabili che viene inserito tra parentesi tonde e restituito dal emtodo stesso. L’utente può poi destrutturarlo per ottenere i valori richiesti singolarmente; inoltre se uno dei valori della tupla non interessa (e quindi non si vuole recuperarlo) è possibile inserire al suo posto nell’elenco un underscore.

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

Il casting, ossia la conversione tra diversi tipi di dato, in C# può essere fatto in diversi modi, ossia con il cast, ossia con una operazione (la cui sintassi è quella di scrivere tra parentesi tonde il nome del tipo di arrivo vicino al nome della variabile di cui si vuole fare il casting. Esso è un operatore che cerca di fare la conversione, e se non ci riesce genera una eccezione. Il casting può essere implicito, ossia che si esegue senza implementare la sintassi sopra descritta, nel caso in cui la conversione non comporti perdita di informazioni (come ad esempio nel passaggio da intero a double), mentre è esplicito nel caso in cui la sua esecuzione comporti effettivamente una perdita di informazioni. In questo secondo caso, è necessario implementare la sintasi.

Il cast può essere poi fatto attraverso l’operatore AS, che svolge una conversione più sicura in quanto cerca di fare la conversione, e se non ci riesce ritorna un valore di tipo null.

In aggiunta, esiste l’operatore IS, che è un operatore di guardia, in quanto ci restituisce un booleano che indica se è possibile eseguire il cast o meno, permettendoci quindi di eseguire l’operazione solamente nei casi in cui avrà successo (ed evitando il sollevamento di eccezioni).

1. Quali sono gli utilizzi della keyword static?

La keyword static si usa per dichiarare classi o membri di una classe statica. Una classe statica è una classe che non può essere istanziata, e che può contenere solo membri statici. A tali membri si accede usando il nome della classe stessa (poiché non è possibile definirne una istanza): molte classi che contengono solamente metodi spesso sono statiche.

I membri statici contenuti in una classe non statica invece, sono membri il cui valore è valido per tutte le istanze della classe stessa (non è specifico per ogni singola istanza). Tali membri, nella memoria, si trovano in un’area separata da quella per le proprietà non statiche, proprio perché comuni a tutta la classe.

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

Un evento è un membro di una classe che ha il compito di mandare messaggi all’esterno. Esso segue un pattern chiamato publisher-subscriber. Il publisher è quello che genera l’evento, quindi la classe che manda messaggi all’esterno. L’evento è dichiarato al suo interno utilizzando la keyword “event”, ed è seguito dal delegate che lo definisce ( e che definisce la firma che poi dovranno avere gli EventHandler) e dal nome. I delegate che definiscono gli eventi possono essere dichiarati dall’utente, ma ne esistono anche di predefiniti in C#, che si chiamano EventHandler. Essi prendono sempre in ingresso due parametri, che sono l’oggetto sender, che rappresenta la classe che ha generato l’evento, e una struttura dati detta EventArgs che contiene delle informazioni sull’evento stesso. Essa può essere vuota (nei delegate che non prendono parametri in ingresso) oppure può essere popolata. In quest’ultimo caso, non si può usare la classe base EventArgs ma l’utente deve generarne una apposita da popolare.

Una volta che il publisher invoca l’evento (chiamandolo come un normale metodo), se tale evento ha almeno un sottoscrittore, viene generata la notifica. Cosa accade una volta invocato l’evento è deciso dal subscriber, che quando si iscrive all’evento (usando un overload dell’operatore +=) associa all’evento il metodo che si richiama in caso di invocazione dell’evento stesso. Il subscriber può poi cancellare l’iscrizione all’evento usando l’operatore -=.

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

IEnumerable è un’interfaccia che rende la classe che la implementa enumerabile, ossia che da la possibilità di elencare in modo ordinato tutti gli elementi che una sua istanza contiene (è l’interfaccia da implementare se vogliamo inserire in un foreach una istanzanza di tale classe, ad esempio). La classe che la implementa deve avere una collection privata di elementi generici, e per implementarla deve contenere due metodi di tipo GetEnumerator, uno generico (ereditato da object) e uno specifico per il tipo usato. Per implementare il primo normalmente non si fa altro che richiamare il secondo. Il secondo invece, che espone un enumeratore tipizzato, contiene un foreach che gira su tutti gli oggetti contenuti nella collezione privata della lista, e ogni volta ne restituisce uno. Tale foreach fa uso della keyword yield, che permette di “mantenere in memoria” quale elemento della collezione privata è stato stampato per ultimo, in modo tale che il foreach non ricominci sempre dal primo elemento (in quanto yield è affiancato da un return, che quindi comporta l’uscita dal ciclo foreach).

Una volta implementata questa interfaccia, è anche possibile eseguire delle query con linq sulle istanze della classe stessa

**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