# Esercitazione di Fine Settimana – Week 2

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Nome | Michael |
|  |  | Cognome | Locci |
|  |  | 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#

Per ritornare più valori da un metodo abbiamo 3 possibilità:

* *Classe/struttura*: in questo caso il tipo di ritorno del metodo è un’istanza della classe con le sue proprietà (es. Il metodo Add della classe ComplexNumber che restituisce un oggetto complex number. Oppure un altra possibilità è applicare il design pattern “Pattern Request- Respose” dove di creano le classi Request e Respose ed in ognuna si implementano la proprietà che vogliamo far restituire al metodo ovvero le caratteristiche che deve avere il valore di ritorno del metodo (es. PersonRequest e PersonRespose dove quest’ultima è la classe che contiene le caratteristiche della risposta che desidero come le proprietà successo, messagio, eccezione,...)
* *Tupla*: questa tipologia prevede di restituire una tupla come valore di ritorno e prevede che nella tupla ci siamo tutti i valori da ritornare:

public (bool, string, Exception) GetData(PersonRequest requestdata), inoltre quando la si usa nel main deve essere spacchettata ovvero:

(bool success, string message, Exception ex) = Person.GetData(PersoneRequest pr, “ “)

* *Parametri out*: in questo caso abbiamo un tipo di ritorno e gli altri tipi di ritorno sono posti nella firma del metodo e devono essere popolati all’interno del metodo prima di fare il return: public bool GetData(PersonRequest pr, out string message, out Exception ex)

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

Le tipologie di casting sono 2:

* Conversioni *implicite* dove non è richiesta la sintassi perchè la conversione avviene sempre con successo in quanto non avviene una perdita di dati. Ad esempio la conversione di un int in un long.
* Conversioni *esplicite* dove invece è richiesta un’espressione di cast, in quanto c’è la possibilità di perdere le informazioni oppure che la conversione non vada a buon fine. Ad esempio la conversione di un double in un int => double x = 1234.9;

int a;

a = (int) x;

dove si perdono informazioni in quanto il range di valori tipo DOUBEL è maggiore del range di valori del tipi INT.

1. Quali sono gli utilizzi della keyword static?

La parola static può essere usata nella definizione di una classe, così facendo questa non può essere istanziata ovvero non è possible utilizzare la parola chiave new per crearne un’istanza, inoltre una classe statica può contenere solo metodi statici mentre una classe non statica può avere sia metodi statici che non. Per accedere ai membri statici si utilizza il nome della classe stessa.

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

Un evento è un membro che permette alla classe di inviare notifiche verso l’esterno e mantiene una lista di Subscriber che ricevono la notifica quando un evento viene pubblicato per cui si parla di due entità: Publisher che inoltra l’evento a tutti i Subscriber e quest’ultimo che riceve la notifica e per farlo si deve iscrivere all’evento attraverso il metodo Subscribe che implementa “c.Evento += f” ed analogamente si può disiscrivere con UnSubscribe che implementa “c.Evento -= f”. L’evento si dichiara con la parola chiave event e viene invocato con un normale metodo, la cui firma deve essere uguale al delegato che prevede due parametri: sender l’oggeto che scatena l’evento ed args ovvero un’istanza della classe che deriva da EventArgs. Quindi per fare il subscribe, ovvero per ricevere la notifica, è sufficiente aggiungere all’evento un delegate con il comando c.MyEvent += f; analogamente per l’unsubscribe.

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

L’interfaccia IEnumerable<T> serve per esporre un enumeratore, tramite il metodo GetEnumerator(), che supporta un’iterazione su una collezione generica. Per fare questo si deve implementare l’interfaccia ovvero : public class NameClass<T> : IEnumerator<T>

La quale implementa i metodi:

* public IEnumerator<T> GetEnumerator(){ }
* Ienumerator Ienumerator.GetEnumerator(){ }

**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