# Esercitazione di Fine Settimana – Week 2

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Nome | Carlotta |
|  |  | Cognome | Colla |
|  |  | Data | 02/04/2021 |

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 in C# esistono vari metodi:  
   - Ritornare un oggetto di una classe che ha vari campi/proprietà in cui i salvare vari dati  
   - Utilizzare la parola out per ritornare un valore con i parametri presi in input  
   - Ritornare una tupla, ovvero una colleziona di oggetti di un tipo ben specificato, ad esempio se si vuole ritornare due stringhe e un intero il metodo sarà dichiarato con public (string, string, int) NomeFunzione()  
   - Utilizzando il passaggio di parametri per riferimento, in questo modo il loro valore verrà modificato nel corpo della funzione
2. Descrivere le due tipologie di casting tra tipi in C#  
   Le due tipoligie di casting sono:  
   - Casting implicito: ovvero quando si passa da un tipo restringente a un tipo più generico, come ad esempio da int a long. In questi casi si ha la certezza di non avere perdita di memoria e che il cast non generi eccezioni  
   - Casting espicito: in questo caso può esserci perdita di memoria e se il cast non va a buon fine verrà lancita l’eccezione InvalidCastException. Con il cast espicito si usano gli operatori di cast come le parentesi tonde: sono poco sicure perchè se il cast non va a buon fine si genera l’eccezione, “IS”: per sapere prima se un tipo può essere covertito in un altro, si mette in un if (x is Tipo), se risponde true allora può essere convertito, “AS”: con as se il cast non va a buon fine allora assegna null senza lanciare eccezioni, “TryParse”: è un metodo sicuro, ritorna un booleano per sapere se la conversione è andata a buon fine o no, se il booleano è true allora assegna a una variabile out il valore della conversione  
   - Esiste anche il casting tra tipi definiti da utente in cui è necessario scrivere le funzioni di cast implicito o espicito.
3. Quali sono gli utilizzi della keywork static?  
   La keyword static può essere utilizzata con:  
   - Campi: un campo statico di una classe è un campo uguale per tutte le istanze di quella classe, viene assegnato utilizzando il nome della classe e non quello dell’istanza  
   - Metodo: Un metodo statico è uguale per tutte le istanze, questo vuol dire che dovrà prendere in input l’istanza della classe su cui fare i calcoli  
   - Classe: una classe statica non può essere istanziata con la parola new, i suoi campi e metodi possono essere utilizzati usando direttamente il nome della classe.  
   I dati statici vengono salvati in una parte di memoria differente.
4. Descrivere le differenze tra le interfacce utilizzabili per le collection  
   Le collection implementano varie interfacce, tra cui:  
   - IEnumerable: è la più importante, permette di utilizzare il foreach per scorrere tutti gli elementi della lista  
   - ICollection: introduce i metodi di Add, Remove, Count ..  
   - IDictionary: viene utilizzato dalle collection che hanno coppie di elementi chiave-valore come HashTable e Dictionary  
   - IEqualityComparer: serve per stabilire quando due elementi della collection sono uguali, mentre ICompare per stabilire quale tra due elementi è il maggiore per ordinarli  
   - IList: viene implementata dalle collection che hanno gli indici degli elementi e introduce RemoveAt, InsertiAt ..
5. Cos’è un Generic?  
   Un generic è un tipo di dato che viene utilizzato quando si scrivono classi o metodi indipendenti dal tipo. Per convezione si utilizza <T> per definie un generic.  
   I suoi grandi vantaggia sono la possibilità di scrivere meno codice (al posto di implementare la stessa classe per ogni tipo di dato se ne implementa solo una con un generic che potrà essere utilizzata da qualsiasi tipo). Un genric può assumere tutti i valori tranne null.  
   In un classe o interfaccia con un generic è possibile definire alcuni vincoli sul tipo che T può assumere utilizzando la keyword “where”.

**Esercitazione Pratica**

* Realizzare una classe Account per gestire un conto bancario, con le seguenti proprietà:
  + *Numero di Conto*
  + *Nome della Banca*
  + *Saldo*
  + *Data Ultima Operazione*
  + *Lista di Movimenti*
* Realizzare l’overload degli operatori + e – in modo che sia possibile aggiungere movimenti attivi e passivi alla lista dei movimenti (l’overload dovrà anche occuparsi di aggiornare il Saldo e la Data di Ultima Operazione)
* Realizzare un metodo Statement() che stampi i dati del conto, inclusa la lista dei movimenti
* Realizzare una gerarchia di classi per rappresentare movimenti bancari (Movement). Tutte le classi avranno le proprietà
  + *Importo*
  + *Data del Movimento*
  + Realizzare le classi che rappresentano:
    - CashMovement, con la proprietà aggiuntiva *Esecutore*
    - TransfertMovement, con le proprietà aggiuntive *Banca d’Origine* e *Banca Destinazione*
    - CreditCardMovement, con le proprietà aggiuntive *Tipo* (enum con i valori AMEX, VISA, MASTERCARD, OTHER) e *Numero di Carta*
  + 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
* Realizzare una Console app che
  + Crei un nuovo Account
  + Permetta di inserire diversi tipi di Movimenti (input dall'utente)
  + Stampi i dati del conto e i movimenti