# Test di valutazione – Modulo 2

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Nome | Beatrice |
|  |  | Cognome | Testa |
|  |  | Data | 21/10/2019 |

Leggete attentamente ogni domanda e argomentare quanto più possibile fornendo anche degli esempi.  
ATTENZIONE: Le domande a risposta multipla possono contenere più risposte corrette.

1. *Cosa si intende quando si utilizza il tipo “var” durante la dichiarazione delle variabili?*

la variabile non ha un tipo specifico quindi può assumere qualunque tipo

il motore di .NET determina il tipo a runtime time

il compilatore C# determina il tipo a compile time

la variabile ha un tipo implicito che potrebbe essere anche esplicitamente dichiarato a codice

1. *A cosa servono il Costruttore e il Distruttore di una classe?*

A creare e distruggere la classe a cui sono applicate

A gestire le operazioni di inizializzazione e pulizia delle risorse di una classe

A permettere ad una classe di gestire in maniera efficiente il Garbage Collector

A inizializzare una classe senza che la stesse venga terminata dal Garbage Collector

1. *Quali sono i pilastri sulla quale si basa la OOP? Dare una definizione di ciascuno di essi descrivendo per quale motivo sono fondamentali per questo tipo di approccio allo sviluppo*

I pilastri della OOP sono l’ereditarietà, l’incapsulamento ed il polimorfismo. L'incapsulamento indica che un gruppo di proprietà, metodi e altri membri correlati vengono considerati come una singola unità o un singolo oggetto. L'ereditarietà indica la capacità di creare nuove classi sulla base di una classe esistente. Il polimorfismo indica la capacità di usare più classi in modo intercambiabile, anche se in ognuna di esse le stesse proprietà o gli stessi metodi sono implementati in modi diversi.

1. *Qual è la differenza tra overloading e overriding? Fornire un esempio di una classe per cui si applica un override di un metodo e un overload dello stesso metodo*

L'overload di un metodo consente a più metodi della stessa classe di avere lo stesso nome, purché abbiano firme univoche; è un tipo di polimorfismo. Significa creare piu di un metodo con lo stesso nome ma con firma diversa. Per overriding invece si intende avere due metodi con lo stesso nome e stessa firma. Una viene implementata nella classe base, l'altra nella classe derivata. Vuol dire sovrascrivere il metodo della classe base. Si ottiene utilizzando le parole chiavi virtual(nella implementazione della classe base) e override (nella implementazione della classe derivata)

class Prog {

public int Addition(int num1, int num2) {

return (num1 + num2);

}

public int Addition (int num1, int num2, int num3) {

return (num1 + num2 + num3);

}

public string Addition (string value1, string value2) {

return (value1 + " " + value2);

}

public string Addition (string value1, string value2) {

return (value1 + " " + value2);

}

public virtual int Add(int num1, int num2) {

return (num1 + num2);

}

static void Main(string[] args) {

Prog pro = new Prog();

Console.WriteLine("two int parameter :" + pro. Addition (3, 2));

Console.WriteLine("three int parameter :" + pro. Addition (3, 2, 8));

Console.WriteLine(" string parameter :" + pro. Addition ("hello", "world"));

DerivedCl derivedc= new DerivedCl();

derivedc.Add(4,5);

}

}

Public class DerivedCl:Prog{

public override int Add(int num1, int num2) {

Console.WriteLine("Addition from derived class with int parameter :" );

return (num1 + num2);

}

}

1. *Cosa è il Garbage Collector e a cosa serve nel mondo .NET?*

Garbage Collection recupera automaticamente la memoria occupata dagli oggetti inutilizzati.

Il Garbage Collector funge da gestore di memoria automatico; esso offre diversi vantaggi e consente di: sviluppare l'applicazione senza dover liberare manualmente la memoria per gli oggetti creati; allocare gli oggetti nell'heap gestito in maniera efficiente; recuperare gli oggetti inutilizzati, ne cancella la memoria e tiene la memoria a disposizione per le future allocazioni e garantire protezione per la memoria assicurando che un oggetto non possa usare il contenuto di un altro oggetto.

1. *Qual è la differenza tra Values Type e Reference Type?*

Le Values Type contengono direttamente i propri dati, mentre le Reference Type archiviano i riferimenti ai propri dati, noti come oggetti. Con i reference type, due variabili possono fare riferimento allo stesso oggetto e di conseguenza le operazioni su una delle due variabili possono influire sull'oggetto a cui fa riferimento l'altra. Con i values type invece, ogni variabile ha una propria copia dei dati e non è possibile che le operazioni su una variabile influiscano sull'altra (tranne nel caso delle variabili di parametro ref e out).

1. *Cos’è e a cosa serve LINQ? Dare una definzione dello stesso, e descrivere per quale motivo è diventato fondamentale per aumentare la produttività di uno sviluppatore .NET*

LINQ (Language-Integrated Query) è il nome di un set di tecnologie basate sull'integrazione delle funzionalità di query direttamente nel linguaggio C#. In genere, le query sui dati vengono espresse come stringhe semplici senza il controllo dei tipi in fase di compilazione o il supporto di IntelliSense. Con LINQ, una query è un costrutto del linguaggio di prima classe, come le classi, i metodi e gli eventi. È possibile scrivere query su insiemi di oggetti fortemente tipizzati usando le parole chiave del linguaggio e gli operatori comuni. Per uno sviluppatore che scrive query, la parte integrata nel linguaggio più visibile di LINQ è l'espressione di query. Le espressioni di query vengono scritte con una sintassi di query dichiarativa. Tramite la sintassi di query è possibile eseguire operazioni di filtro, ordinamento e raggruppamento sulle origini dati usando una quantità minima di codice. Vengono usati gli stessi modelli di espressioni di query di base per eseguire una query e trasformare i dati in database SQL, set di dati ADO .NET, documenti e flussi XML e raccolte .NET.

1. *Quali sono le differenze tra una classe, una classe astratta e un’interfaccia? Elencare quante più caratteristiche possibili di ciascuno di questi costrutti*

Le classi vengono dichiarate usando la parola chiave class. Una classe può ereditare l'implementazione solo da una classe di base singola. Una classe può implementare più di un'interfaccia.

Una classe astratta è una classe che non può essere istanziata (cioè non è possibile creare oggetti di questa classe). Quindi una classe astratta deve essere derivata. Una classe astratta può contenere membri astratti ma anche membri non astratti. La classe derivata dalla classe astratta, deve implementare i membri astratti facendone l'override. In alternativa potrebbe anche non implementarli tutti, ma in questo caso deve essere a sua volta astratta.

Una interfaccia è un insieme di 'firme' di proprietà, metodi, delegati e eventi. Per firme si intende che le proprietà, i metodi e gli eventi sono solo definiti come nome, parametri di input e di output, ma non ne è specificato il funzionamento. Per essere utilizzabile, una interfaccia deve essere 'implementata' da una classe, che ne specifica quindi il funzionamento nel dettaglio. Una interfaccia non può essere istanziata e non contiene codice che definisce i suoi membri, semplicemente definisce solo i membri. È la classe che implementa l'interfaccia che definisce i membri. Una classe può implementare più interfacce, così come più classi possono implementare la stessa interfaccia. Per convenzione le interfacce hanno nomi che iniziano con 'I'. Non è possibile istanziare una interfaccia ma si può creare una variabile che punta a ad una istanza di una classe derivata dall'interfaccia.

1. Come si definisce una “classe parametrica”? Fornire un esempio della stessa, fornendo sia il codice implementativo che il suo utilizzo pratico

Una classe parametrica è una classe che scorpora le funzioi del contesto. Utilizza i generics; ha una entità in input. Una classe può essere definita come dipendente da parametri che sono tipi invece che valori di un tipo, che compaiono fra parentesi angolari <> subito dopo il nome della classe. Tali parametri possono poi essere usati all'interno della classe come se fossero dei tipi ordinari. Nella definizione del costruttore non si scrivono i parametri-tipi fra parentesi angolari. Public abstract class ManagerBase<TEnity> where TEntity:EntitaMonitorabilr

1. *Esercitazione pratica. Si chiede di creare un app console in .NET Framework che, in fase di esecuzione, permetta di inserire e visualizzare un catalogo di automobili e biciclette (entrambe le entità avranno le medesime funzioni).* 
   * *Quando l’applicazione viene avviata permette di selezione se si vuole lavorare sul catalogo delle biciclette (selezione da menu “A”) o quello delle automobili (selezione da menu “B”).*
   * *Dovrà essere possibile inserire una bicicletta (o un’automobile) e salvarla su un “database” realizzato con un file di json*
   * *Dovrà essere possibile visualizzare il contenuto intero del database dell’oggetto di riferimento*
   * *Dovrà essere possibile visualizzare solo gli elementi che corrispondono al criterio di ricerca sul campo “Modello”.*
   * *La bicicletta è definita dal campo “Modello”, “Marca”, “NumeroTelaio” e “IsElettrica”;*
   * *L’automobile è definita da “Modello”, “Marca”, “NumeroCavalli”, “IsDiesel” e “DataImmatricolazione”.*
   * *Ogni entità è caratterizzata da un campo “Id” numerico che deve essere univoco nella stessa tipologia di oggetti*
   * *Dovrà essere presente una funzione che permette di eseguire la creazione dei mezzi di locomozione di una famiglia (N biciclette e 1 automobile), passando le quantità e i valori come parametri alla funzione CreaMezziDellaFamiglia(int numeroDiBiciclette, string string marcaBiciclette, string[] modelliBiciclette, string marcaAutomobile, string modelloAutomobile)*

[Tot: \_\_/10]