# Modulo 1

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Nome | Carlotta |
|  |  | Cognome | Colla |
|  |  | Data | 26/03/2021 |

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. *Dare una panoramica degli elementi principali nel Framework .Net. Cosa è successo con .Net 5?*.Net è cross-platform e open source ed è diviso in Framework e Core. Framework permette di sviluppare applicazioni solo per window mentre core per tutti i sistemi operativi (grazie alla bash). Con .Net è possibile sviluppare applicazioni web e app per desktop, per sviluppare applicazioni mobile si utilizza Xamarin. In .Net 5 non c’è più la suddivisioni tra Core e Framework, tutte le applicazioni create sono cross-platform. È possibile sviluppare applicazioni desktop, mobile con xamarin, web, cloud, gaming con unity e applicazioni con l’intelligenza artificiale.
2. *Definire i principi su cui si basa il linguaggio C#*C# è un linguaggio object oriented e si basa sui 3 principi di incapsulamento, ereditarietà e polimorfismo.  
   L’incapsulamento serve per stabilire la visibilità tra classi attraverso i modificatori di accessibilità che possono essere public, private, protected, internal e internal protected. In c# le classi possono derivare da altre classi, ovvero ereditare i campi, le proprietà e i metodi (se non sono privati). L’ereditarietà multipla non è permessa: una classe può derivare da una sola classe base. Grazie al polimorfismo una classe base può avere dei metodi marcati virtual, ovvero che hanno un comportamento base e tramite l’override la classe derivata può modificare quel comportamento oppure dei metodi abstract, in quel caso la classe derivata deve implementare quella funzionalità. Dato che l’ereditarietà multipla non è concessa si utilizzano le interfacce, in questo caso una classe può implementare più interfacce.
3. *Dare una spiegazione di Garbage Collector.*

Il garbage collector serve per libeare la memoria quando è piena. La parte di memoria che libera è l’heap, esso è diviso in 3 parti chiamate generation 0, 1, 2. I reference type vengono inizialmente salvati nella generation 0 (che è la più piccola), una volta piena viene richiamato il garbage collector che compie le segueti azioni:  
1- Segna tutte le variabili come garbage  
2- Segna quali variabili sono ancora in uso  
3- Elimina le variabili che non servono più  
4- Compatta la memoria  
Una volta che una generation è piena, i dati segnati come validi vengono spostati nella generation successiva.

1. *Descrivere le differenze tra Reference Type e Value Type*

I Value type contengono direttamente il dato, vengono salvati sullo stack (tranne quando vengono dichiarati all’interno di una classe), non possono essere nulli e non hanno bisogno di essere inizializzati. Alcuni esempi di value type sono i tipi primitivi come int, bool, e struct e enum.  
Mentre i reference type contengono un indirizzo alla cella di memoria dove viene salvato il dato, l’indirizzo viene salvato sullo stack mentre il dato sull’heap. I reference type possono essere nulli e hanno bisogno di essere istanziati, esempi di questa categoria sono le stringhe (anche se non hanno bisogno di essere inizializzate con new) e le classi.

1. *Cosa è un delegate?*

Un delegate è come un puntatore ad una funzione. Prima di tutto viene creata una funzione, es. Public int Sum(int x, int y), poi si dichiara il delegate con la stessa firma più il tipo di ritorno, es. Delegate int Somma(int a, int b), questo delegate viene utilizzato come classe per creare un oggetto che prende in input la funzione, Somma prova = new Somma(Sum); infine si richiama la funzione con prova(3,4);

1. *Scrivere una query LINQ : Date le gare disponibili, ricavare il tempo medio, massimo e minimo per disciplina.*

Var tempi = gare.GroupBy( g => g.disciplina)  
.Select(tempo => new {  
Disciplina = tempo.Key,  
TempoMedio = tempo.Avarage( d => d.tempoImpiegato),

TempoMax = tempo.Max(d => d.tempoImpiegato),  
TempoMin = tempo.Min(d => d.tempoImpiegato)  
});

1. *Definire cosa è il design Pattern Factory*Il design pattern Factory viene utilizzato quando si conosce solo un crtiterio per la creazione degli oggeti, non la loro gerarchia. In questo pattern è presente un’interfaccia per creare degli oggetti senza sapere come verranno creati. Il client è a conoscenza solo dell’interfaccia, non delle classi concrete. Il cliente passa un parametro alla factory e in base a quello crea l’oggeto corrispondente e chiama i suoi metodi.

*Mettere la prova pratica e teorica su Github.*