# Test di fine settimana – Week2

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Nome | Isabella |
|  |  | Cognome | Centofanti |
|  |  | Data | 28/05/2021 |

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 definizione di classe statica*

*Una classe statica è una classe che presenta al suo interno solo proprietà e metodi statici. Essa non può essere istanziata. Si può usare ad esempio quando nell’ereditarietà, hai una classe padre che non ti serve istanziare perché è una classe troppo generica.*

*Static class Persona*

*{*

*Public static string Nome {get;set;}*

*Public static string Cognome {get;set;}*

*Public static int Eta {get;set;}*

*Public static string ToString()*

*{*

*Return $”Nome {Nome}, cognomen {Cognome}, eta {Eta}”;*

*}*

1. *Quali tipologie di ereditarietà sono consentite in C# e come si definisce? Fornire un esempio*

*L’ereditarietà è un rapporto gerarchico tra classi diverse. Di fatto, una classe diventa dipendente da un’altra, che gli passa proprietà, campi e metodi. Per indicare il rapporto tra due classi, nella classe che eredita metodi e proprietà aggiungiamo “:”.*

*Class Quadrato : Forma*

*SI definisce una classe Padre, che è quella da cui dipenderà un’altra(o più) classe, e la classe Figlia.*

*In c# abbiamo 3 tipi di ereditarietà:*

1. *Interazione semplice, dove una classe dipende strettamente da un’altra (esempio classe Forma <- classe Quadrato*
2. *Gerarchica, dove a una stessa classe padre corrispondono più classi figlie*

*FORMA*

*Quadrato Triangolo Rettangolo*

1. *multilivello, è un’ereditarietà gerarchica ma su più livelli, quindi le classi figlie avranno a loro volta delle classi che dipenderanno da loro*

*FORMA*

*Quadrato Triangolo Trapezio Rettangolo*

*TriangoloRettangolo TriangoloIsoscele TrapezioIsoscele TrapezioRettangolo*

1. *Elencare le principali caratteristiche della classe System.Object.*

*In c# tutto deriva dalla classe System.Object. Essa è la classe Padre per eccellenza di tutte le classi.*

*Vi sono dei metodi che sono propri della classe Object che noi richiamiamo quando creiamo altri oggetti.*

*ToString, converte un oggetto in una stringa*

*Equals, comparazione tra gli oggetti della classe*

*Finalize, si chiama in fase di cancellazione da parte del Garbage collector*

*GetType, recupera il tipo dell’oggetto*

1. *Descrivere le due fasi di gestione delle eccezioni.*

Le eccezioni ci aiutano a gestire gli errori che si possono verificare nel corso del nostro programma, per evitare che esso si interrompa. Sono una classe: la madre di tutte le eccezioni è Exception che è l’eccezione generica, che a sua volta ha molte sottoclassi.

Per gestire le eccezioni, facciamo uso del blocco TRY-CATCH.

Nel Try scrivo la parte di codice che potrebbe dare un errore,

Nel Catch gestisco le eccezioni che potrei trovare: poiché le eccezioni sono tante, di solito scrivo prima delle eccezioni specifiche che penso di poter generare e come ultima l’eccezione generica. Ad ognuna mi faccio stampare il tipo di errore riscontrato.

Vi è anche il finally, che il blocco finale: non è obbligatorio scriverlo, ma può essere utile perché viene letto sia che il programma venga eseguito sia che intercorra in un’eccezione. Mi fa comodo magari se devo chiudere uno Stream aperto, così da evitare di sprecare risorse/permettere a qualcun altro di usarle

Try

{

Using(StreamWriter writer = File.CreateText(path))

{

writer.WriteAsync(“speriamo funzioni”);

}

}

Catch (ArgumentException ex)

{

Console.Writeline(ex.Message);

}

Catch (Exception e)

{

Console.Writeline(e.Message);

}

Finally

{

writer.Close();

}

*Esercizio Pratico*

Creare una Console Application che gestisca i Task dell’utente.

Per Task viene inteso un oggetto che ha una descrizione, una data di scadenza e un livello di importanza (Basso, Medio, Alto).

L’utente può:

* Vedere i Task inseriti
* Aggiungere un nuovo Task
* Eliminare un Task
* Filtrare i Task per importanza

Requisiti Tecnici:

- Recuperare i task da file

- Salvare i Task in un file

- Utilizzare adeguatamente il concetto di classe

- Dividere le funzionalità in relative funzioni e procedure

- Commentare le scelte algoritmiche

- Mettere una nomenclatura conforme

- Verificare le date di scadenza dei task: devono essere posteriori o uguali rispetto alla data di inserimento

- Controllare l’input utente

Opzionale: Utilizzare Enum

Mettere il codice dell’esercizio in un Repository di GitHub.