Domande:

1. Descrivere le differenze tra Reference Type e Value Type

Il value type contiene direttamente il valore del dato sullo stack, inoltre se faccio una copia della variabile vuol dire che sto copiando il contenuto della variabile stessa e se faccio una modifica questa è relativa all’istanza corrente. Il value type contiene sempre un valore e non è ammesso il null. Fra i value type troviamo I tipi primitivi cone int, bool, double ed anche gli enum.

I reference type invece contengono un riferimento ad un oggetto nell’heap, per cui se faccio una copia di un reference type allora sto facendo una copia del riferimento (indirizzo) nella mia nuova variabile. Inoltre se eseguo una modifica su due reference allora sto modificando l’oggetto a cui puntano. A differenza del value type, il reference type che non punta a nessuna istanza ha valore null. Inoltre tutte le classi sono reference type.

1. Cosè un delegate?

Il delegate in C# è l’equivalente di un puntatore in C++ e viene dichiarato con la parola chiave: delegate.

Ad esempio con la dicitura: **delegate void MyDelegate(double d)** il compilatore crea una classe di nome MyDelegate che deriva da System.Delegate.

Il delegate viene istanziato passandogli nel costruttore il nome del metodo di cui si vuole creare il delegate, ad esempio:

**MyDelegate del = new MyDelegate(MyMethod)** ed in metodo deve avere gli stessi parametri del delegate creato per cui : **void MyMethod(double d) { }**  quindi l’istanza può essere invocata con **del(20.5)** che esegue il metodo (MyMethod) con parametro un double.

1. Definire come costruire il design pattern Factory e qual’è la sua utilità

Il design pattern Factory è un tipo di pattern creazionale. Il pattern viene costruito creando un’interfaccia per il tipo generale che si vuole creare (ad esempio IVeicolo), successivamente si creano le classi concrete che implementano l’interfaccia creata che si occuperanno di gestire il singolo oggetto (ad esempio la classe macchina che implementa IVeicolo conterrà delle proprietà relative alla macchina). Tutta questa parte resta invisible al client in quando nel pattern abbiamo un’altra classe che si occupa della creazione degli oggetti chiamata ItemFactory ( nel nostro caso VeicoloFactory), questa riceve la richiesta dal client e si occuperà di creare l’oggetto richiesto in base ad un parametro (ad esempio il numero di route del veicolo oppure in base ai cavalli del motore, dove le classi precedenti non saranno più “machina”, “bici” ma saranno “supercar”, “citycar” …). La classe Factory avrà un metodo (creazioneVeicolo) che si occuperà della creazione dell’oggetto (veicolo) in base al valore del parametro scelto (route). Nel client troviamo il main, che una volta appreso il parametron (route) richiederà la creazione dell’oggetto (veicolo), creando un’istanza della classe Factory ed invocando su di essa il metodo per la creazione dell’oggetto.

Per cui l’utilità del pattern è quella di demandare alla classe factory il compito della creazione degli oggetti, rendendo invisibile al client tutta la parte delle classi degli oggetti (veicoli).

1. Dare una panoramica degli elementi principali nel Framwork .NET. Cosa è successo con l’introduzione di .NET 5?

Nel Framework .NET troviamo 3 blocchi:

1. L’infrastruttura comune dove troviamo i compilatori (CLR), i linguaggi (C#, F#) ed I component runtime. Si occupa di convertire il codice in IL e memorizzato in un assembly (DLL e EXE)
2. La libreria standard che specifca un insieme di API
3. Il blocco .NET in cui troviamo:
   1. .NET Framework rilasciata nel 2001 ed applicabile solo a Windows
   2. .NET Core rilasciata nel 2015, è cross-platform ovvero la possibilità di operare su più piattaforme (win, linux macos), è open-source ed è utilizzata per tutti i nuovi sviluppi.
   3. XAMARIAN rilascita nel 2014, open-source ed applicabile per lo sviluppo mobile.

Con l’introduzione di .NET 5 si è unificato tutte le piattaforme .NET, tranne quella mobile e sono stati aggiunti dei tools come Visual Studio, Visual Studio Code.