1. Il Value Type è un tipo di valore che contiene direttamente il *valore* di una variabile (allocata nello stack), a differenza invece del Reference Type che contiene il *riferimento* a un oggetto (che è "conservato" nell'heap). Esempi di value type sono tutti i tipi primitivi (come int, bool, double…), mentre esempi di reference type sono le stringhe (seppure reference type particolari) così come tutte le classi predefinite di C# o definite dall'utente stesso.
2. Un delegate è un particolare tipo definibile dal programmatore che, invece si fare riferimento a un'istanza, fa riferimento a un metodo. La classe definita dal delegate andrà ad estendere la classe Delegate o MulticastDelegate. Per definirlo è necessario usare la parola chiave *delegate* seguita dal tipo di ritorno, nome del delegate e tipo e nomi dei parametri di input (il tutto preceduto ovviamente da eventuali modificatori di visibilità). Ad esempio, questo è un modo per definire un delegate: `public delegate string Presentazione(string nome);`. A questo punto si potrà usare *Presentazione* per istanziare dei riferimenti a metodi che rispetteranno la firma del delegate definito (nel concetto di "firma" rientra il tipo di ritorno, numero e tipo di parametri di input). Se, per esempio si definisse all'interno della classe Program un metodo come il seguente

```
1 reference
private static string Saluto(string nome)
{
    return $"Ciao, mi chiamo {nome}";
}
```

   nel main si potrà fare quanto segue:

   a. Creare un delegato del metodo Saluto `Presentazione x = Saluto;`
   b. Usare il delegato per stampare il messaggio restituito dal metodo stesso

```
Console.WriteLine(x("Ludovica"));
```

   L'output sarà il seguente: `Ciao, mi chiamo Ludovica`.

3. Il Factory pattern è un design pattern che, come può suggerire la parola *Factory*, serve per *creare* delle classi (non a caso rientra nella categoria dei pattern cosiddetti "Creazionali"). In particolare, fornisce uno scheletro per la realizzazione di programmi in cui si debbano istanziare oggetti di specifiche classi sulla base di un certo parametro richiesto. Per fare un esempio, si supponga di avere un programma per la gestione di una banca in cui si voglia creare un certo tipo di Conto corrente sulla base dell'età del cliente che apre un conto. Si potrà creare un'interfaccia che definisce le funzionalità di ContoCorrente (es. una funzionalità StampaDettagli) e delle classi (come ContoYoung, ContoFamily, ContoBig) che implementano quest'ultima. A questo punto si creerà una classe ContoFactory che istanzia un particolare Conto a seconda di un input (nel nostro caso un int che rappresenta l'età del cliente) proveniente dalla classe che ha richiesto il "servizio di creazione" della classe Contofactory.
4. La fase di "assert" è quella fase del Test Driven Development (TDD) in cui si vuole testare il risultato prodotto a seguito dell'esecuzione di un particolare blocco o riga di codice. Per far questo, si utilizza la classe Assert della libreria XUnit la quale fornisce una serie di metodi attraverso cui possiamo testare se certe condizioni si sono verificate durante il test. Tra i tanti si ricordano: Equal(double, double) che confronta tra di loro due numeri di tipo double e controlla

se questi sono uguali o meno; False(bool) che prende in input un booleano e verifica se questo ha valore di verità *false* e True(bool) che è il complementare di False(bool).