Appello TAP del 11/9/2023

Scrivere nome, cognome e matricola sul foglio protocollo. Avete a disposizione due ore e mezza.

Esercizio 1 (8 punti)

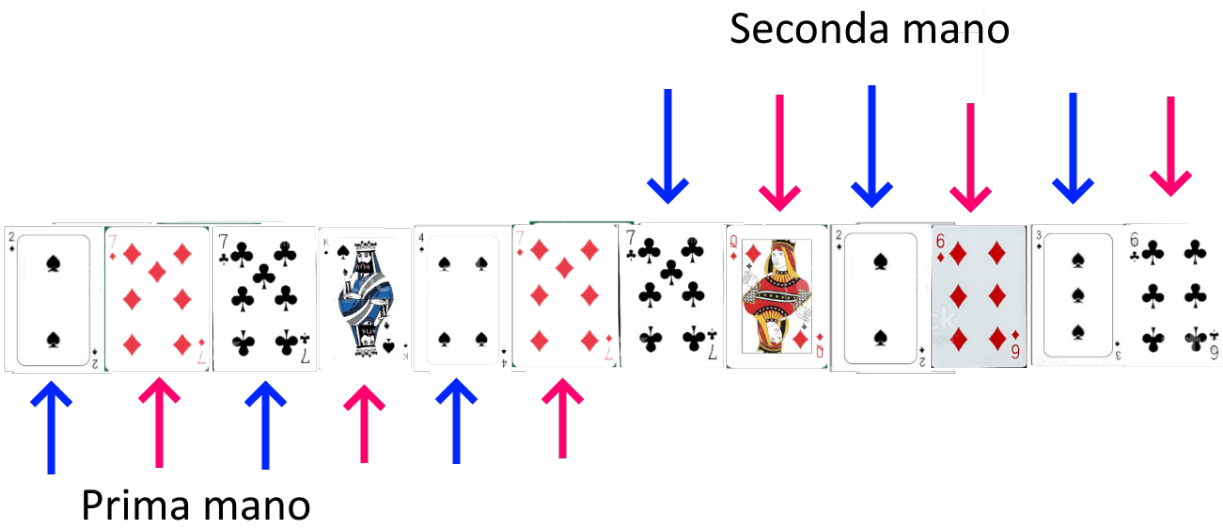
Vogliamo fornire gli strumenti per implementare un elementare gioco di carte in cui i due giocatori si alternano a pescare da un mazzo, potenzialmente infinito (quindi in cui le carte si possono ripetere), fino ad avere una mano di tre carte. Vince chi nella sua mano ha la carta più alta. In caso di pareggio, vince chi ha iniziato a pescare per primo.

Completare la definizione degli operatori relazionali nella seguente interfaccia in modo che una carta sia più piccola di un'altra se il suo valore è inferiore (dove l'asso è il più piccolo possibile) oppure se ha lo stesso valore ma il suo seme è più *debole* (dove cuori/hearts > quadri/diamonds > fiori/clubs > picche/spades).

```
public enum Cards {Ace, Two, Three, Four, Five, Six, Seven, Jack, Queen, King}
public enum Suits {Spades, Clubs, Diamonds, Hearts}
public interface IPlayingCard {
  Cards Value { get; }
  Suits Suit { get; }
  static bool operator <=(IPlayingCard first, IPlayingCard second) {/*...*/}
  static bool operator >=(IPlayingCard first, IPlayingCard second) {/*...*/}
}
```

Scrivere l'extension-method `FirstWins` che, data una sequenza (potenzialmente infinita) di carte da gioco (`IPlayingCard`), `s`, produce una sequenza di booleani corrispondenti alla vittoria del primo giocatore.

Ad esempio, sul mazzo che contiene, nell'ordine, un 2 di picche, un 7 di quadri, un 7 di fiori, un re di picche, un 4 di picche, un 7 di denari, un 7 di fiori, una donna di quadri, un 2 di picche, un 6 di denari, un 3 di picche e un 6 di fiori, come in figura, dovrà restituire la sequenza `false`, `false`.



Infatti il primo giocatore (frecce blu) nella prima mano pesca un 2 di picche, un 7 di fiori e un 4 di picche, la sua carta più alta è il 7 di fiori. Il secondo giocatore (frecce rosa) pesca due 7 di quadri e un re di picche, che è la sua carta più alta. Il re di picche è più alto del 7 di fiori e quindi vince il secondo giocatore. Analogamente nella seconda mano la carta più alta del primo giocatore è un 7 di fiori, che è più bassa della donna di quadri che ha in mano il secondo giocatore.

Il metodo dovrà sollevare `ArgumentException` se il mazzo è finito e non contiene abbastanza carte per completare l'ultima mano. Ad esempio, se contiene solo 5 carte (non si riesce a completare la prima mano) o ne contiene 19 (non si riesce a completare la 4 mano).

Esercizio 2 (9 punti)

Implementare, usando NUnit ed eventualmente Moq, i seguenti test relativi all'esercizio 1.

1. Scrivere un test parametrico, con due parametri di tipo array, rispettivamente di `Cards` e di `Suits` che verifica che se i due giocatori pescano le stesse carte vince sempre il primo.

   Se i parametri hanno lunghezza diversa o non multipla di 3 il test dovrà risultare inconclusive.

   Altrimenti dovrà usare come argomento per la chiamata sotto test il mazzo in cui ogni carta è ripetuta due volte ed è costruita usando i dati nei parametri. Quindi la prima e la seconda carta avranno il valore del primo elemento del primo parametro e il seme del primo elemento del secondo parametro. La terza e quarta carta avranno il valore del secondo elemento del primo parametro e il seme del secondo elemento del secondo parametro e così via.

2. Scrivere un test per verificare che il metodo invocato su un mazzo di infinite carte generate casualmente produca un risultato contenente almeno 1000 elementi.

   Hint In C# esiste la classe `Random` con metodi `Next()`, `Next(int MinValue, int MaxValue)`, e `Next(int MaxValue)`.

3. Scrivere un test per verificare che il metodo sollevi eccezione di tipo `ArgumentException` su un mazzo contenente 7 carte.

Esercizio 3 (9 punti)

1. Si considerino le seguenti tre varianti di gestione parziale di una eccezione, dove `MyException` e tutte le sue eventuali estensioni abbiano i costruttori convenzionali per le eccezioni

Variante 1

```
catch (MyException e) {
  M();
  throw;
}
```

Variante 2

```
catch (MyException e)
  when (F(e))   {}
```

dove

```
bool F(MyException e){
    M();
    return false;
 }
```

Variante 3

```
catch (MyException e)  {
  M();
  throw Activator.CreateInstance(
    e.GetType(), e)
        as Exception;
}
```

Vero	Falso	
☐	☐	tutte le varianti si concludono con una eccezione non gestita
☐	☐	tutte le varianti si concludono con una eccezione non gestita dello stesso tipo di quella catturata
☐	☐	nella variante 1 l'eccezione `e` risulta ancora non gestita all'uscita dal blocco `catch`
☐	☐	nella variante 2 l'eccezione `e` risulta ancora non gestita all'uscita dal blocco `catch`
☐	☐	nella variante 3 l'eccezione `e` risulta ancora non gestita all'uscita dal blocco `catch`

2. Introdurre un extension method per la classe `C`

Vero	Falso	
☐	☐	causa errore statico se `C` contiene già un metodo con la stessa segnatura non dichiarato `virtual`
☐	☐	richiede l'uso della parola chiave `override` se `C` contiene già un metodo con la stessa segnatura
☐	☐	rende impossibile dichiarare un metodo con la stessa segnatura in una sottoclasse di `C` se non dichiarato `virtual`
☐	☐	può andare in overriding di un metodo omonimo di `C` o di una sua sottoclasse

3. Si consideri il seguente codice che usa l'Entity Framework

```
[Index(nameof(CustomerName),nameof(RegistrationDate),IsUnique = true)]
public class Customer {
    public int CustomerId { get; set; }
    public string CustomerName { get; set; }
    [Required]
    public DateTime? RegistrationDate { get; set; }
    [Required]
    public float Balance { get; set; }
    public DeliveryAddress DeliveryAddress { get; set; }
    public int AgentId { get; set; }
}
public class DeliveryAddress {
    public string Street { get; set; }
    public int DoorNumber { get; set; }
}
public class Agent {
    public int AgentId { get; set; } /*...*/
}
public class MyContext : DbContext {
    public DbSet<Customer> Customers { get; set; }
    protected override void OnConfiguring(DbContextOptionsBuilder o) {
        o.UseSqlServer(@"Data Source=.;Initial Catalog=MyDb;
            Integrated Security=True");
         base.OnConfiguring(o);
    }
    protected override void OnModelCreating(ModelBuilder modelBuilder) {
        modelBuilder.Entity<Customer>().OwnsOne(u => u.DeliveryAddress);
        base.OnModelCreating(modelBuilder);
    }
}
```

Vero	Falso	
☐	☐	Nel DB associato a un `MyContext` esiste una tabella corrispondente alla classe `Customer`
☐	☐	Nel DB associato a un `MyContext` esiste una tabella corrispondente alla classe `DeliveryAddress`
☐	☐	Nel DB associato a un `MyContext` esiste una tabella corrispondente alla classe `Agent`
☐	☐	Rimuovere l'attributo `Required` dalla definizione della property `RegistrationDate` cambia lo schema del DB associato a un `MyContext`
☐	☐	Rimuovere l'attributo `Required` dalla definizione della property `Balance` cambia lo schema del DB associato a un `MyContext`
☐	☐	Nel DB associato a un `MyContext` la tabella corrispondente alla classe `Customer` ha più di 6 colonne
☐	☐	Nel DB associato a un `MyContext` la tabella corrispondente alla classe `Customer` ha una chiave esterna verso la tabella corrispondente alla classe `Agent`
☐	☐	Nel DB associato a un `MyContext` la tabella corrispondente alla classe `Customer` ha una chiave esterna verso la tabella corrispondente alla classe `DeliveryAddress`
☐	☐	Nel DB associato a un `MyContext` la tabella corrispondente alla classe `Customer` ha colonne nullabili