Appello TAP del 18/2/2022

Scrivere nome, cognome e matricola sul foglio protocollo e sul foglio con le risposte al quiz. Avete a disposizione due ore e mezza.

Esercizio 1 (9 punti)

Si consideri il seguente frammento incompleto di codice per la gestione degli appuntamenti di uno studio medico

L'interfaccia `IAppointment` rappresenta la richiesta di appuntamento fatta da un paziente e incapsula il nome del paziente, le sue proposte di possibili appuntamenti e la data effettivamente stabilita dallo studio (`null` finché non viene fissata).

La classe `MedicalScheduler` rappresenta l'agenda dello studio medico: in `FreeSlots` ci sono possibili slot in cui assegnare appuntamenti (tutti fra loro distinti) e in `Appointments` sono memorizzati gli appuntamenti fissati fino a quel momento.

```
public interface IAppointment {
    public string PatientName { get; }
    public IEnumerable<DateTime> ProposedTimes { get; }
    public DateTime? SelectedAppointmentTime { set; }
}

public class MedicalScheduler {
    public Dictionary<DateTime, string> Appointments { get; }
        = new Dictionary<DateTime, string>();
    public List<DateTime> FreeSlots { get; }= new List<DateTime>();
    ....
    public IEnumerable<Tuple<string, bool>>
        Schedule(IEnumerable<IAppointment> requests) {...}
```

Assunzioni che potete dare per scontate e non dovete controllare :

- tutti gli orari usati sono ore esatte, ovvero hanno le componenti minuti e secondi uguali a zero;
- gli orari in `FreeSlots` e in `Appointments` sono tutti diversi fra loro.

Dovete implementare il metodo `Schedule` che prende una sequenza di richieste di appuntamento e restituisce la sequenza di coppie "paziente"+"successo della prenotazione". Il metodo dovrà elaborare ciascuna richiesta nel suo argomento e cercare uno slot libero che coincida con uno degli orari proposti dal paziente. Se ne trova uno dovrà

- eliminare lo slot da quelli disponibili
- inserire l'appuntamento che assegna quello slot al paziente in `Appointments`
- assegnare lo slot alla proprietà `SelectedAppointmentTime` del paziente
- restituire come risultato la coppia paziente e `true`

Se nessuno degli slot liberi coincide con almeno uno delle proposte del paziente, `Schedule` dovrà solo restituire come risultato la coppia paziente e `false`.

Esercizio 2 (7 punti)

Implementare, usando NUnit, i seguenti test relativi a `Schedule`, dell'esercizio 1.

1. Input della chiamata sotto test: `source` contiene i dati di un solo paziente, "giorgio" che propone un unico orario possibile, il 3/5/2022 alle 11

   Stato dell'oggetto su cui fare la chiamata: nessun appuntamento fissato e nessuno slot libero

   Output atteso: una sequenza con un unico elemento, la coppia "giorgio" e `false`.

2. Input della chiamata sotto test: `source` contiene i dati di un solo paziente, "ada" che propone due orari possibili

   - il 2/7/2022 alle 18
   - il 16/4/2022 alle 10

   Stato dell'oggetto su cui fare la chiamata: nessun appuntamento fissato e gli slot liberi sono:

   - il 2/8/2022 alle 8
   - il 16/4/2022 alle 11
   - il 16/4/2022 alle 10
   - il 3/5/2022 alle 10

   Output atteso: la proprietà `SelectedAppointmentTime` di "ada" è stata aggiornata al 16/4/2022 alle 10.

3. Input della chiamata sotto test: `source` contiene i dati di un solo paziente, "ugo" che propone due orari possibili

   - il 12/3/2022 alle 15
   - il 13/3/2022 alle 15

   Stato dell'oggetto su cui fare la chiamata:

   - gli slot liberi sono:
     - il 20/8/2022 alle 18
     - il 6/6/2022 alle 16
     - il 13/3/2022 alle 15
     - il 3/3/2022 alle 15
   - gli appuntamenti già assegnati sono
     - 13/8/2022 alle 10 → franco
     - 6/6/2022 alle 15 → luca
     - 6/6/2022 alle 17 → paola

   Output atteso:

   - il risultato della chiamata è una sequenza con un unico elemento, la coppia "ugo" e `true`
   - la proprietà `SelectedAppointmentTime` di "ugo" è stata aggiornata al 13/3/2022 alle 15
   - gli slot ancora liberi sono
     - il 20/8/2022 alle 18
     - il 6/6/2022 alle 16
     - il 3/3/2022 alle 15
   - gli appuntamenti già assegnati sono
     - 13/8/2022 alle 10 → franco
     - 6/6/2022 alle 15 → luca
     - 6/6/2022 alle 17 → paola
     - 13/3/2022 alle 15 → ugo

Esercizio 3 (9 punti)

Per ciascuna delle seguenti affermazioni, indicate se è vera o falsa

1. In un test del metodo statico `M()` della classe `C`:

```
public static IEnumerable<int>M(){
    for (int i = 0; i < 10; i++) yield return i;
    throw new InvalidOperationException();
}
```

   Le seguenti asserzioni sono *successful*

Vero	Falso	
☐	☐	`Assert.That(C.M(), Is.Not.Empty);`
☐	☐	`Assert.That(C.M(), Throws.TypeOf<InvalidOperationException>());`
☐	☐	`Assert.That(()=>C.M(), Throws.TypeOf<InvalidOperationException>());`
☐	☐	`Assert.That(()=>C.M().Any(), Throws.TypeOf<InvalidOperationException>());`
☐	☐	`Assert.That(()=>C.M().ToArray(), Throws.TypeOf<InvalidOperationException>());`
☐	☐	`Assert.That(C.M().ToArray(), Throws.TypeOf<InvalidOperationException>());`

2. Se le interfacce `I1` e `I2` definiscono lo stesso metodo `M()` fornendo diverse implementazioni di default

Vero	Falso	
☐	☐	nessuna classe può implementare simultaneamente `I1` e `I2`
☐	☐	nella definizione di una classe che implementa simultaneamente `I1` e `I2` bisogna optare per una delle due implementazioni
☐	☐	una classe che implementa simultaneamente `I1` e `I2` può implementare in modo esplicito le due varianti di `M()`

3. Dato

```
IEnumerable<string?>  A = ....;
var a1 = A.Skip(3).FirstOrDefault();
var a2 = A.ElementAtOrDefault(3);
var b = A.Take(3);
var a3 = A.FirstOrDefault();
```

Vero	Falso	
☐	☐	qualunque sia il valore di `A` si ha `a1==a2`
☐	☐	si ha `a1==a2` solo se `A` ha almeno quattro elementi
☐	☐	se non ci sono accessi concorrenti ad `A`, `a1==a3` qualsiasi siano gli elementi di `A`

4. Nell'ambito dell'Entity Framework Core, la scelta del tipo di base di dati da usare per rendere permanenti le entità può essere effettuato

Vero	Falso	
☐	☐	nel costruttore del contesto
☐	☐	nel metodo `OnModelCreating`
☐	☐	nel metodo `SaveChanges`

5. Dato `IQueryable<int> X`

Vero	Falso	
☐	☐	in `X.Any()&&32==X.First()` la chiamata a `First` può sollevare eccezioni anche se quella a `Any()` restituisce `true`
☐	☐	enumerazioni diverse di `X` possono produrre risultati differenti
☐	☐	in `foreach(var i in X){...}` `var b=X.Any()` a `b` viene sempre assegnato `false` perché `X` è stato visitato fino alla fine e non ci sono ulteriori elementi

6. In Git, il comando `rebase`

Vero	Falso	
☐	☐	può modificare la storia passata del repository
☐	☐	causa errore in esecuzione se si è precedentemente fatto `push` del repo sul server
☐	☐	causa errore in esecuzione se un diverso utente ha già fatto `pull` dei `commit` coinvolti

7. Considerate queste due varianti (i puntini indicano parti non rilevanti ai fini dell'esercizio) dal punto di vista della DI (Dependency Injection)

```
class D { ... }
class C  {

    C() { ... }
    void M() {
        ... var x = new D(); ...}
}
```

1

```
class D:ID { }
class C {
    private ID _myD;
    C(ID d) { _myD = d; ....}
    void M() {
        ... var x = _myD; ... }
}
```

2

Vero	Falso	
☐	☐	la variante 1 rispetta i principi della DI
☐	☐	la variante 2 rispetta i principi della DI
☐	☐	non ci può essere differenza fra il comportamento delle varianti 1 e 2
☐	☐	non si può applicare la DI alla variante 1 senza alterare il comportamento di `M`
☐	☐	per applicare la DI alla variante 1 il costruttore di `C` dovrebbe prendere come parametro una factory per `ID`
☐	☐	quando si aggiungono parametri al costruttore per applicare la DI è opportuno mantenere anche il costruttore originale, per evitare di rompere i client