A.A. 2020/2021

Remedi Tommaso, Matricola N. 300535

- **SPECIFICA DEL SOFTWARE:**
  Il progetto consiste nella simulazione di una giornata di lavoro in un'officina di auto.
  All'interno di questa officina c'è un meccanico che a inizio giornata prende in input da un file di testo tutte le auto con le relative condizioni dei vari pezzi(motore,gomme,sospensioni,fari,freni e batteria) su cui dovrà lavorare quel giorno e le mette in una lista. Per ogni macchina prima controlla lo stato di ogni singolo pezzo, che se risulterà sotto la soglia della sufficienza andrà riparato o eventualmente sostituito. Per decidere se un pezzo verrà sostituito o andrà riparato viene sommato alle condizioni di quel pezzo un modificatore fortuna di [-3;+3], se grazie all'aggiunta di questo modificatore il pezzo supera la soglia della sufficienza allora esso verrà riparato, con una spesa dimezzata per il cliente che verrà aggiunta all'entrate dell'officina. In caso questo modificatore non bastasse il cliente dovrà pagare il prezzo pieno, l'officina aggiunge alle spese del fornitore il prezzo della riparazione scontato del 20% ed aggiunge alle entrate il prezzo pieno della riparazione.
  Quando il meccanico ha controllato tutte le auto termina il programma mostrando a terminale il guadagno giornaliero.
  Viene anche implementata una funzione di log che tiene traccia in un file di testo di tutti i lavori eseguiti sulle auto durante questa giornata.
  Ho deciso di implementare due pattern, uno creazionale(singleton) sulla classe Officina ed uno comportamentale(state pattern) per il meccanico.

- **STUDIO DEL PROBLEMA:**
  - **PUNTI CRITICI:**
    Durante la realizzazione del progetto si sono presentate diverse difficoltà, tra le quali:
    1: L'utilizzo di un file esterno di testo per il caricamento di tutte le auto su cui il meccanico dovrà lavorare.
    2: La creazione di un file di testo per i log, in cui il meccanico andrà a scrivere tutte le operazioni che esegue sulle varie auto.
    3: L'adattamento dei design sulle classi del progetto.
    4: La creazione di una fattura in formato XML.
    5: Scorrere i vari pezzi di un'automobile cambiando anche i nomi dei lavori eseguiti senza utilizzare una serie di if.

  - **FRONTEGGIAMENTO:**
    1-Il caricamento di tutte le auto avviene dentro la classe Meccanico, il meccanico andrà a prendere dal file di input del programma, chiamato Input .txt, tutte le informazioni per tutte le automobili. Ogni riga del file di input rappresenta un auto con gli attributi che dovrà caricare separati da ;. Per ogni macchina il meccanico andrà a prelevare i suoi attributi in questo ordine: la marca, il nome del modello, lo stato del motore, lo stato delle gomme, lo stato delle sospensioni, lo stato dei fari, lo stato dei freni e lo stato della batteria. Come ultima cosa andrà a leggere il conto del cliente, che di default parte da 0, ma in caso il cliente abbia già dei debiti con l'officina si può scegliere da quanto farlo partire. Ho pensato di gestirlo come un file CSV ed ho usato le funzioni di libreria per caricare correttamente tutte le auto con le relative condizioni dei pezzi dal file di input ad un lista di tipo Automobile.

```
public void CaricaDaFile(List<Automobile> automobili) {

    Console.WriteLine("Inserire il percorso del file di input compreso il backslash finale: ");
    //Variabile necessaria per l'acquisizione degli input da file, indica il percorso del file di input
    //Incollare il percorso del file Input.txt tra le "", esempio: "D:\Progetto PMO\Officina Riparazioni\"
    string filePath = Console.ReadLine();
    try
    {
        var config = new CsvConfiguration(CultureInfo.InvariantCulture)
        {
            HasHeaderRecord = false,
        };
        using (var reader = new StreamReader(filePath + "Input.txt"))
        using (var csv = new CsvReader(reader, config))
        {
            csv.Context.RegisterClassMap<AutomobileMap>();
            while (csv.Read())
            {
                automobili.Add(csv.GetRecord<Automobile>());
                Console.WriteLine("AUTO CORRETTAMENTE AGGIUNTA\n");

            }
        }
    }catch (FileNotFoundException)
    {
        Console.WriteLine("File di input non trovato, impossibile avviare programma");
    }
}
```

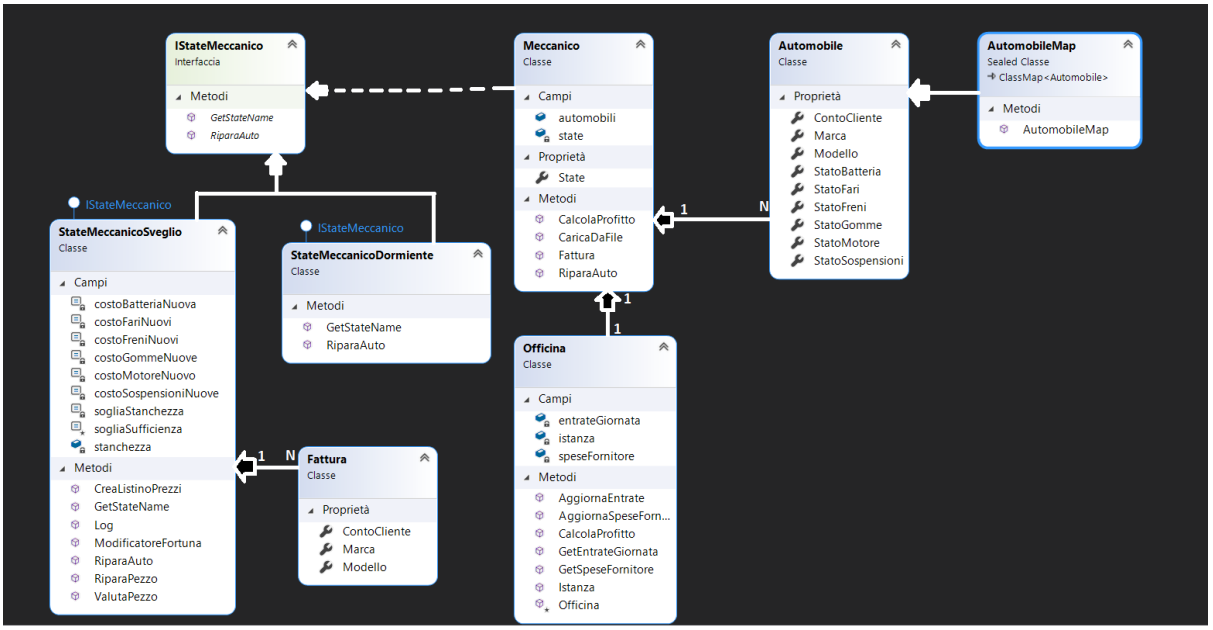
2-La creazione dei log del programma è stata implementata grazie appunto alla funzione Log presente dentro la classe Meccanico. Ogni volta che una macchina avrà un componente in pessime condizioni, il meccanico andrà ad eseguire la riparazione o la sostituzione di esso, per poi andare a scrivere nel file di testo denominato Log.txt che intervento ha eseguito e su quale auto.. Quindi a fine esecuzione il file dei log conterrà la lista di tutti i lavori che sono stati eseguiti su tutte le macchine.
3-L'implementazione dei due design pattern è stata eseguita modificando opportunamente la classe Officina per ottenere il Singleton, mentre lo state pattern è stato ottenuto attraverso la creazione di un'interfaccia e di due state del meccanico. I due stati del meccanico implementano l'interfaccia IMeccanico mentre la classe Meccanico utilizza questa interfaccia.
4-Per l'implementazione della fattura per ogni auto in formato XML ho creato innanzitutto una classe di tipo fattura, contenente tre attributi, marca, modello e contoCliente. Dopodiché creo una lista di tipo fattura che andrò a popolare utilizzando i valori presenti nella lista automobili a fine lavorazione del meccanico.Poi vado a creare un file chiamato Fatture.xml ed infine grazie alla librerie System.Xml.Schema e System.Xml.Serialization ho serializzato la lista in un file xml andando a creare le fatture per le varie automobili.

```
2 riferimenti
public void Fattura(List<Automobile> automobili) {
    List<Fattura> fatture = new List<Fattura>();
    for (int i = 0; i < automobili.Count; i++)
    {
        Fattura fattura = new Fattura();
        fattura.Marca = automobili[i].Marca;
        fattura.Modello = automobili[i].Modello;
        fattura.ContoCliente = (int)automobili[i].ContoCliente;
        fatture.Add(fattura);
    }


    Stream stream = File.OpenWrite("Fatture.xml");
    XmlSerializer xmlSer = new XmlSerializer(typeof(List<Fattura>));
    xmlSer.Serialize(stream, fatture);
    stream.Close();
}
```

5-Sono riuscito ad ovviare a questo problema facendo diverse ricerche online. Innanzitutto sono andato a creare una classe AutomobileMap che estende Classmap<Automobile> per avere un mapping degli attributi della classe Automobile su degli indici che vanno da 0 a 8.
Poi con un foreach ciclo per ogni proprietà in auto e se è di tipo stringa come marca o modello lo faccio semplicemente mostrare a terminale, se è di tipo intero lo casto a intero e lo valuto ed in caso lo riparo. Quando sono andato a mappare gli attributi ho fatto il conto di tipo double per far sì che non venisse contato come un pezzo.

```
foreach (PropertyInfo prop in auto.GetType().GetProperties())
{
    var type = Nullable.GetUnderlyingType(prop.PropertyType) ?? prop.PropertyType;
    if (type == typeof(String))
    {
        Console.WriteLine(prop.GetValue(auto, null).ToString());
    }

    if (type == typeof(int))
    {
        if (ValutaPezzo(int.Parse(prop.GetValue(auto, null).ToString()), componenti[i]) == false)
        {
            RiparaPezzo(auto, int.Parse(prop.GetValue(auto, null).ToString()), costiPezzi[i], componenti[i], Officina.Istanza());
        }
        i++;
    }
```

- **SCELTE ARCHITETTURALI:**
    - **DIAGRAMMA DELLE CLASSI:**



- **DESCRIZIONE DELL'ARCHITETTURA:**

  L' architettura del programma è incentrata su classi che si concentrano sulle rispettive funzionalità. La classe Officina presenta metodi per occuparsi della parte finanziaria, esponendo i metodi calcola profitto e i due metodi per aggiornare le spese sia dei clienti che del fornitore. Inoltre la classe Officina presenta il metodo Istanza e l'attributo di tipo Officina istanza La classe automobile presenta 9 proprietà, che in ordine stanno a rappresentare la marca della macchina, il modello, lo stato del motore, lo stato delle gomme. lo stato delle sospensioni, lo stato dei fari, lo stato dei freni, lo stato della batteria e da quanto parte il conto del cliente possessore di quella auto.
  La classe Meccanico invece si occupa di popolare la lista di auto prese dal file di input e di lavorare su ognuna di esse, trascrivendo tutti i lavori che fa nel file dei log. La classe Meccanico utilizza l'interfaccia IMeccanico per gestire i due stati che può avere(dormiente o lavorativo), a loro volta le classi MeccanicoDormiente e MeccanicoLavorativo implementano l'interfaccia IMeccanico, ed hanno entrambi un metodo chiamato RiparaAuto(Automobile). In base alla classe su cui verrà richiamato esso avrà un comportamento diverso, nel caso del MeccanicoLavorativo andrà effettivamente a continuare il suo lavoro di valutazione e riparazione dei pezzi, invece nel caso del

MeccanicoDormiente non andrà avanti con il lavoro ma smetterà di lavorare e si riposerà per un po', scrivendo a terminale che si è addormentato, fatto ciò sarà pronto per lavorare di nuovo. Il meccanico per stancarsi ha un attributo che viene incrementato di uno ogni volta che i lavori su una macchina sono finiti, quando questa stanchezza raggiunge la soglia di stanchezza il meccanico diventa dormiente e va a riposare, riportando la stanchezza a 0 rendendolo di nuovo pronto per lavorare.

- **DESCRIZIONE DEI PATTERN UTILIZZATI:**

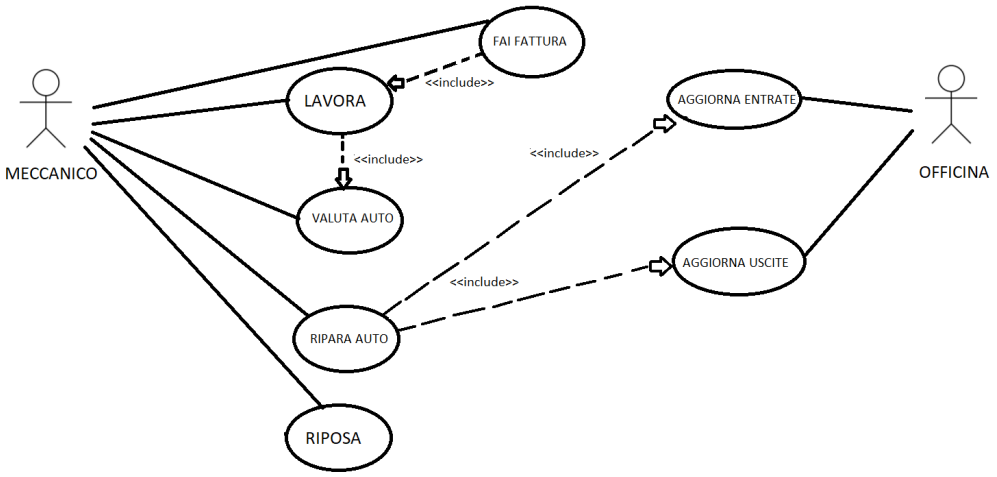
  All'interno del mio progetto ho deciso di utilizzare due pattern di due tipi diversi, uno creazionale ed uno comportamentale.

  SINGLETON: Questo pattern, nel mio caso applicato alla classe Officina, ha la particolarità di assicurarci che esista una sola istanza della classe che lo implementa, e nel caso in cui questa istanza non esiste la crea. Dentro la classe Officina per l'appunto esiste un metodo chiamato Istanza() che viene richiamato ogni volta che si vuole fare una qualsiasi operazione con l'oggetto unico Officina, come per esempio richiamare un suo metodo.

  STATE PATTERN: ho deciso di applicare questo pattern per rappresentare gli stati che il meccanico può cambiare a tempo d'esecuzione(rispettivamente lavorativo o dormiente). La classe MeccanicoSveglio e MeccanicoDormiente, che sono gli stati del mio Meccanico implementano l'interfaccia IMeccanico, che viene utilizzata dalla classe Meccanico. Il meccanico nello stato lavorativo valuta e ripara le automobili mentre nello stato dormiente va a dormire e si riposa(stampando a terminale che sta riposando). Ad inizio esecuzione il meccanico è nello stato di sveglio e non è per niente stanco(stanchezza = 0). Il cambiamento di stato avviene così: Ogni qualvolta che finisce di lavorare ad una macchina la stanchezza viene incrementata di uno e viene controllato se ha raggiunto la costante sogliaStanchezza, nel caso l'abbia raggiunta si procede a returnare uno stato di tipo MeccanicoDormiente e si porta la stanchezza a 0, far riposare il meccanico e poi

returnare un nuovo stato di MeccanicoSveglio. Così finchè non finiscono le auto a cui deve lavorare.

- **DOCUMENTAZIONE SULL'UTILIZZO:**
  L'intero progetto viene fornito all'interno di una cartella compressa contenente il progetto stesso, si noti bene che il file di log non sarà presente finché il programma non viene eseguito per la prima volta. Viene già fornito un file di input chiamato Input.txt per risparmiare tempo al professore, il programma, ad inizio esecuzione, chiederà soltanto di incollare il percorso di questo file, Esempio di stringa da incollare a terminale quando viene richiesta: D:\Progetto PMO\Officina Riparazioni\
- **USE CASES CON RELATIVO SCHEMA UML:**



Nel diagramma si nota come il meccanico(attore) possa eseguire l'azione di riposare o l'azione di lavorare, che porta necessariamente a fargli valutare le automobili, e se l'automobile è in pessime condizioni lui esegue l'azione di ripararla. A fine riparazione in base a quanto è costato il tutto l'officina (attore) andrà ad eseguire l'azione di aggiornare sia le entrate che le spese della giornata. Il meccanico può anche eseguire l'azione del fare la fattura per tutti i clienti.