1506
UNIVERSITÀ
DEGLI STUDI
DI URBINO
CARLO BO

CORSO DI LAUREA IN
**INFORMATICA APPLICATA**
SCUOLA DI
**SCIENZE TECNOLOGIE E FILOSOFIA DELL'INFORMAZIONE**

# RELAZIONE PROGETTO

## PROGRAMMAZIONE E MODELLIZZAZIONE AD OGGETTI

Sessione Riservata 2021/2022

Studente: Martina Breccia
Nr. Matricola: 292514
Email: m.breccia3@campus.uniurb.it

# 1. Specifica del progetto

Sviluppo di un software in grado di gestire le prenotazioni per l'ingresso allo stadio.

Il programma permette di scegliere tra tre diversi stadi, e quattro diversi settori che differiscono dagli altri per il costo.

Una volta selezionati questi dati, verranno aggiunti ad un carrello. L'utente sarà poi in grado di rimuovere i biglietti che non desidera più acquistare.

Verrà calcolata:

- la quantità degli articoli presenti nel carrello
- il totale
- il prezzo medio dei biglietti;

una volta confermato l'ordine, l'utente dovrà inserire i propri dati (nome, cognome, indirizzo, città, telefono) per la spedizione ed infine dare la conferma definitiva dell'acquisto.

Il programma salverà la ricevuta in un file di testo.

## 2. Studio del problema

I punti critici del programma sono:

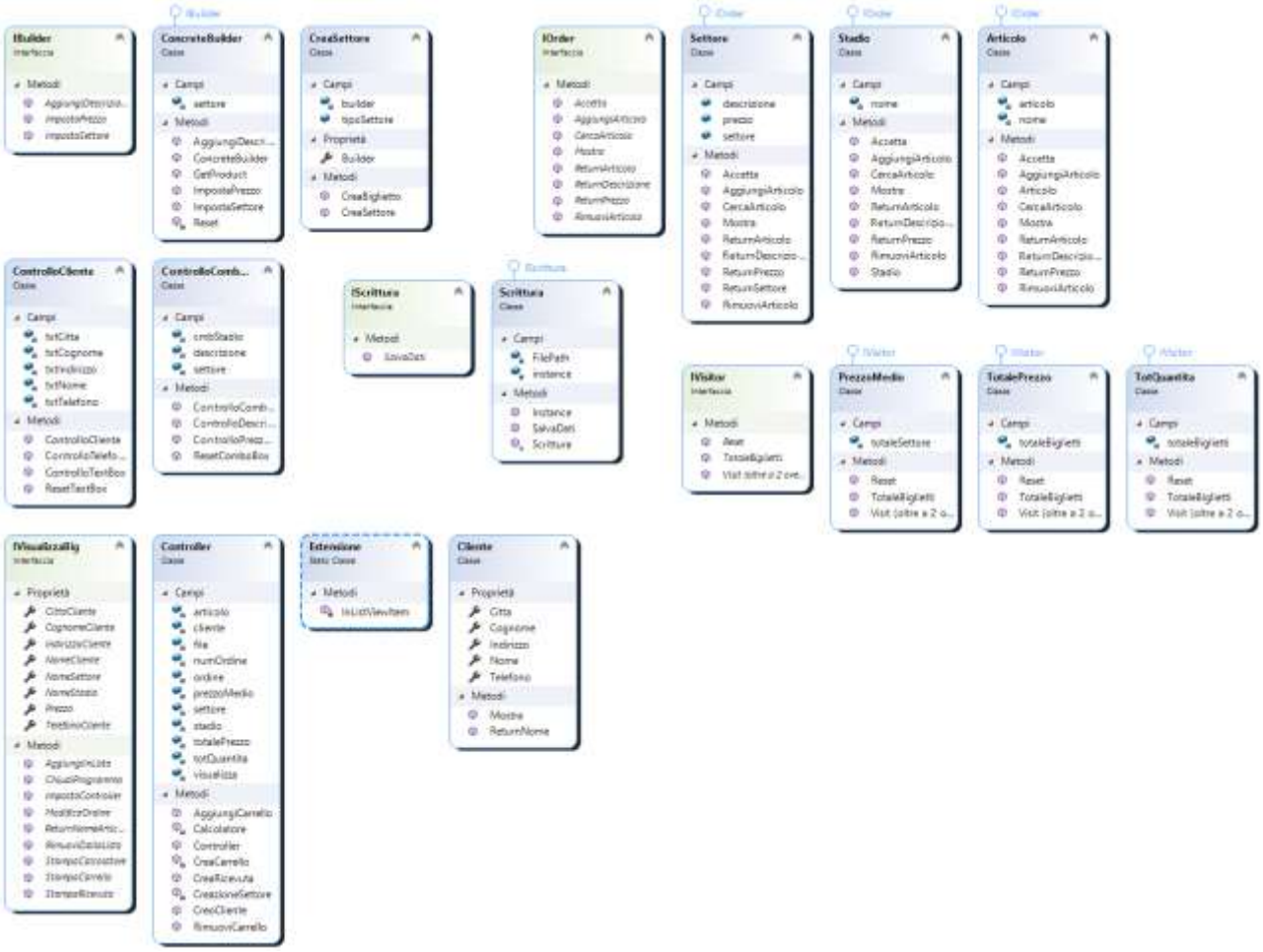
1. L'acquisizione del biglietto, e quindi dello stadio e del settore
2. La struttura per gestire il carrello
3. La costruzione dell'oggetto 'CreaSettore'
4. I vari calcoli effettuati
5. Acquisizione dei dati relativi al cliente
6. Scrittura su file di testo

Soluzione:

1. Per l'acquisizione del biglietto, si è scelto di utilizzare due combobox, nelle quali l'utente può scegliere lo stadio e il settore.
2. Per la struttura del carrello ho scelto di utilizzare il Composite Pattern, dove gli oggetti composti sono i biglietti (chiamati articoli all'interno del programma), mentre le foglie sono rappresentate dal Settore e dallo Stadio.
3. Per costruire l'oggetto 'CreaSettore' ho utilizzato il Builder Pattern. Ho deciso di utilizzare questo pattern perché questo oggetto è costruito con vari elementi.
4. I vari calcoli effettuati dal programma verranno gestiti attraverso un altro pattern: Visitor Pattern. Infatti durante le aggiunte o le rimozioni dei vari biglietti all'interno del carrello, questi calcoli (totale quantità, prezzo totale, prezzo medio dei biglietti) verranno aggiornati costantemente.
5. L'acquisizione dei dati verrà fatta attraverso delle textbox dove l'utente potrà inserire tutti i suoi dati. Una volta confermati, si avrà un messaggio con la pre-stampa della ricevuta, la quale, se da modificare, ci farà tornare al carrello, altrimenti verrà confermato l'ordine.
6. Una volta confermato l'ordine, la ricevuta con i dettagli dell'acquisto verrà stampato su un file di testo. Siccome per scrivere dati all'interno di un file si avrà bisogno di una sola istanza, verrà utilizzato il Singleton Pattern, il quale appunto ci permette di avere una sola istanza e quindi un unico punto di accesso in essa.

# 3. Scelte architetturali

Diagramma delle classi:



Per realizzare questo programma ho utilizzato quattro pattern:

- Builder
- Composite
- Visitor
- Singleton

Ho utilizzato il **Builder Pattern** per creare la classe *'Settore'*. Questo pattern è in grado di separare la costruzione di un oggetto complesso dalla sua rappresentazione, in modo che il processo di costruzione stesso possa creare rappresentazioni diverse.

Così facendo si avrà un meccanismo per la creazione di oggetti complessi indipendente dalle varie parti che compongono l'oggetto e da come vengono assemblate.

Per utilizzare questo pattern avrò la classe ConcreteBuilder che costruisce e assembla le parti del settore implementando l'interfaccia IBuilder, definisce e tiene traccia della rappresentazione che crea.
Utilizzo anche la classe CreaSettore che mi costruisce il settore utilizzando l'interfaccia IBuilder, ed infine ho la classe Articolo che rappresenta l'oggetto complesso in costruzione.

Ho utilizzato il **Composite Pattern** per la gestione della struttura dell'ordine e dei vari articoli contenuti al suo interno.
Questo è un pattern strutturale, e mi permette di comporre oggetti in strutture ad albero, nella quale i nodi sono degli oggetti composti, che nel mio caso vengono rappresentati dai vari articoli che contengono le foglie rappresentate dallo Stadio e il Settore.

Ho utilizzato il **Visitor Pattern** per calcolare la quantità, il prezzo totale dell'ordine e il prezzo medio dei biglietti.
Questo pattern rappresenta un'operazione da eseguire sugli elementi di una struttura e mi permette di definire una nuova operazione senza modificare le classi degli elementi su cui opera.
La mia struttura è l'ordine del cliente, che viene gestito come detto in precedenza dal pattern composite, mentre gli algoritmi che sono applicati su di essa mi calcoleranno: il prezzo totale dell'ordine, la quantità dei biglietti acquistati, e il prezzo medio dei biglietti acquistati.

Infine ho utilizzato il **Singleton Pattern**, per costruire la classe Scrittura, la quale mi permetterà di salvare la ricevuta dell'ordine d'acquisto in un file di testo. Ho utilizzato questo pattern in modo tale da essere sicura che la classe abbia solamente un'istanza e fornisca un punto di accesso globale ad essa.

# 4. Documentazione sull'utilizzo

Il programma, come detto in precedenza, mi dà la possibilità di salvare la ricevuta dell'ordine d'acquisto del cliente all'interno di un file di testo. Per fare in modo che il programma scriva all'interno di questo file, occorrerà passargli il percorso del file di testo, all'interno della classe "Program.cs".

## 5. Use Cases con relativo schema UML

| |
|---|
| **Use Case**: Aggiungi Biglietto |
| **ID:** UC1 |
| **Attore:** User |
| **Presupposti:**<br>- Selezionare lo stadio<br>- Selezionare il settore |
| **Corso base degli eventi:**<br>- Cliccare sul bottone “Aggiungi al Carrello”<br>- Creazione oggetti Stadio e Settore<br>- Creazione dell’articolo e aggiunta dell’articolo all’ordine<br>- Aggiunta dell’ordine nella ListView Carrello |
| **Post Condizione:**<br>- Aggiunta del biglietto nell’ordine finale<br>- Visualizzazione del biglietto nella ListView |
| **Percorsi Alternativi:**<br>- Se lo stadio e/o il settore non sono stati selezionati apparirà un messaggio d’errore |

| |
|---|
| **Use Case:** Rimuovi Biglietto |
| **ID:** UC2 |
| **Attore:** User |
| **Presupposti:**<br>- Deve essere selezionato almeno un biglietto |
| **Corso base degli eventi:**<br>- Selezionare un biglietto<br>- Cliccare sul bottone “Rimuovi Biglietto Selezionato”<br>- Rimozione del biglietto dall’ordine<br>- Rimozione del biglietto dalla ListView |
| **Post Condizione:**<br>- Il cliente potrà aggiungere altri biglietti o confermare l’ordine |
| **Percorsi Alternativi:**<br>- Se nessun biglietto viene selezionato, il bottone non farà nulla |

| |
|---|
| **Use Case:** Conferma Acquisto |
| **ID:** UC3 |
| **Attore:** User |
| **Presupposti:**<br>- All'interno del carrello deve esserci almeno un biglietto |
| **Corso base degli eventi:**<br>- Cliccare sul bottone "Conferma Acquisto"<br>- Compilare i Dati Personali<br>- Creazione dell'oggetto Cliente<br>- Stampa sulla ricevuta<br>- Scegliere se Confermare l'ordine o modificarlo |
| **Post Condizione:**<br>- Nel caso in cui l'ordine viene confermato, la ricevuta viene salvata all'interno del file di testo "FileP.Txt" e verrà visualizzato un messaggio di avvenuto acquisto<br>- Nel caso in cui il cliente voglia modificare l'ordine, verrà rimandato al carrello |
| **Percorsi Alternativi:**<br>- Se nel carrello non vi è presente nessun biglietto, si avrà un messaggio di errore |