## Elaborazione – Iterazione 3

## 3.1 Pattern GoF Strategy

È stato attuato uno Strategy Pattern per gestire i calcoli dei costi di prenotazione. A tal proposito si registra la seguente modifica allo scenario di successo “SSD UC2 Scenario di Successo”.   
Immagine che contiene testo, schermata, linea, Parallelo

Descrizione generata automaticamente

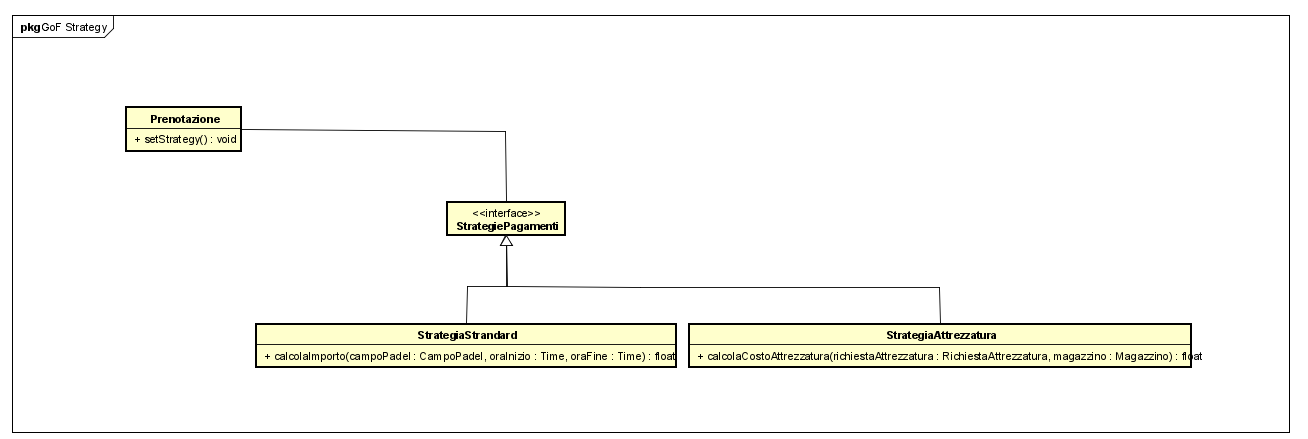
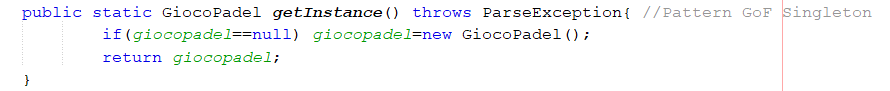
Si nota l’eliminazione dell’“5. SD InserimentoAttrezzatura”. Adesso, nell’atto dell’Inserimento della nuova prenotazione se l’attributo “attrezzaturaRichiesta” è settato a true, viene chiesto il numeroRacchette e il numeroPalline. A questo punto si attuano due Strategie differenti a seconda se è richiesta o meno l’attrezzatura per il calcolo del costo di prenotazione.   
Si osservano le seguenti modifiche all’ “4. SD InserisciNuovaPrenotazione”, che diventa:

Immagine che contiene testo, diagramma, linea, Parallelo

Descrizione generata automaticamente

3.2 Pattern GoF Singleton

Il pattern GoF Singleton assicura che una classe abbia una sola istanza e fornisce un punto di accesso globale a questa istanza. Nel nostro codice, la classe GiocoPadel ha un metodo statico getInstance() che controlla se l'istanza della classe è già stata creata o meno. Se l'istanza esiste già, il metodo restituisce l'istanza esistente; altrimenti, crea una nuova istanza della classe GiocoPadel e la restituisce. Inoltre, la classe GiocoPadel ha un campo statico giocopadel per mantenere l'istanza singleton.



## 3.3 Pattern GoF Facade

Il pattern Facade è un pattern strutturale che fornisce un'interfaccia unificata semplificata per un sottosistema complesso, rendendolo più facile da usare. Il Facade nasconde i dettagli complessi del sottosistema e semplifica l'interazione con esso, fornendo un'interfaccia più semplice e intuitiva per gli utenti. Nel nostro programma, la classe App funge da Facade. Essa fornisce un'interfaccia semplificata e di alto livello per interagire con il sistema GiocoPadel. Esso contiene i metodi statici: “AmministratoreMenu” e “PadeleurMenu”.  
  
3.4 Pattern GoF Observer   
Il pattern Observer è stato utilizzato per notificare gli osservatori (in questo caso i partecipanti di una giocata di padel) quando una nuova prenotazione viene confermata. La classe GiocoPadel funge da Subject e l'interfaccia Observer contiene un unico metodo update(…) che verrà chiamato dal Subject quando viene confermata una nuova prenotazione.

Immagine che contiene testo, schermata, linea, Rettangolo

Descrizione generata automaticamente

Nella classe GiocoPadel, sono stati aggiunti i seguenti metodi per gestire gli osservatori:

* addObserver(…): Aggiunge un nuovo osservatore alla lista degli osservatori interessati al Subject.
* removeObserver(…): Rimuove un nuovo osservatore alla lista degli osservatori interessati al Subject.
* notifyObservers(…): Notifica tutti gli osservatori nella lista quando una nuova prenotazione viene confermata. Gli osservatori ricevono l'ID della prenotazione e l'elenco di e-mail dei partecipanti.

Nel nostro caso specifico non abbiamo bisogno di creare le classi “ConcreteObserver” e “ConcreteSubject” poiché abbiamo implementato il pattern Observer direttamente all'interno della classe GiocoPadel.   
  
Viene apportata una modifica al diagramma “5. SD ConfermaPrenotazione”:  
Immagine che contiene testo, schermata, linea, diagramma

Descrizione generata automaticamente

In conclusione, utilizzando questo pattern GoF possiamo rendere la nostra applicazione migliore restituendo a tutti i partecipanti l’ID della prenotazione e i partecipanti ad una partita di padel.   
  
3.5 Pattern GoF Factory Method  
Abbiamo implementato il Factory Method per l’inserimento di un nuovo Padeleur da parte dell’amministratore.   
  
Abbiamo definito un'interfaccia PadeleurFactory che dichiara il metodo createPadeleur, e abbiamo poi implementato questa interfaccia nella classe DefaultPadeleurFactory.  
  
Il motivo principale che ci ha spinti a pensare al fatto per cui l'amministratore dovrebbe utilizzare il Factory Method pattern è per incapsulare la creazione degli oggetti e nascondere i dettagli specifici di come gli oggetti vengono creati. Ciò può portare ai seguenti vantaggi:

* Separazione delle responsabilità: L'amministratore che utilizza il Factory Method non ha bisogno di preoccuparsi di come esattamente vengono creati gli oggetti. Tutto ciò di cui hanno bisogno è l'interfaccia PadeleurFactory e possono ottenere nuove istanze di Padeleur chiamando il metodo createPadeleur su questa fabbrica.
* Facilità di cambiamenti futuri: Se in futuro l’amministratore desidera cambiare la logica di creazione degli oggetti Padeleur, è sufficiente creare una nuova implementazione della PadeleurFactory senza dover modificare il codice che utilizza il Factory Method. Ad esempio, potrebbe avere esigenze diverse per la creazione di Padeleur a seconda del contesto o di nuove specifiche aziendali.
* Inversione del controllo: Con il Factory Method, il controllo della creazione degli oggetti è spostato dalla classe client a una classe dedicata (DefaultPadeleurFactory). Questo può semplificare la gestione delle dipendenze e migliorare l'organizzazione del codice.

Immagine che contiene testo, schermata, linea, diagramma

Descrizione generata automaticamente