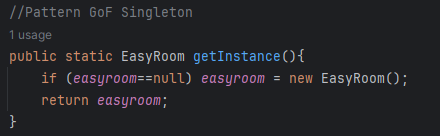
# Elaborazione – Iterazione 3 (Pattern GoF)

L’obiettivo dell’Iterazione 3 è implementare lo scenario principale di successo e tutte le estensioni individuate nel caso d’uso UC3: “Modifica/annullamento di una prenotazione” nella fase di ideazione. Un altro obiettivo è quello di implementare alcuni pattern GoF per rendere più robusta l’applicazione.

### Pattern GoF Singleton

Lo scopo del pattern GoF Singleton è quello di garantire che una classe abbia una sola istanza e fornire un punto di accesso globale a quella istanza. Per fare ciò, viene implementato un metodo statico chiamato “getInstance()” che ritorna l’unica istanza, se questa non dovesse esistere allora verrebbe creata e restituita.



Affinché possa funzionare, nella classe EasyRoom è stato definito un attributo “private static EasyRoom easyroom”. Nel main dell’applicazione viene invocato “getInstance()”:



### Pattern GoF Facade

Lo scopo del pattern GoF Facade è quello di fornire un’interfaccia unificate per semplificare l’uso di un sistema complesso nascondendo la complessità interna e fornendo una serie di punti di ingresso semplificati. All’interno del sistema, nella classe “Main” viene applicato il pattern Facade, essa contiene i metodi statici “applicazioneAdmin()” e “applicazioneUtente()”. In questo modo si ha una distinzione anche tra i ruoli degli attori che accedono al sistema.

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

### Pattern GoF Factory Method

Il pattern Factory Method appartiene ai design pattern, esso si concentra sulla creazione di oggetti, fornendo un’interfaccia per la creazione, ma delegando alle sottoclassi (che implementano materialmente la creazione) la responsabilità. Tra i motivi per cui viene utilizzato il pattern Factory Method rientrano la flessibilità, in quanto è sufficiente creare una nuova implementazione della factory senza modificare il codice che utilizza il Factory Method; separazione della responsabilità, in quanto l’utente non deve preoccuparsi di conoscere materialmente come devono essere creati gli oggetti, ma delega una sottoclasse utilizzando un metodo conosciuto.

Immagine che contiene testo, schermata, linea, Carattere

Descrizione generata automaticamente

Nel sistema è stata realizzata una interfaccia chiamata “GiocatoreFactory” che contiene la definizione di un metodo “createGiocatore()”, che è materialmente implementato all’interno della classe “DefaultGiocatoreFactory”. Per poter creare un’istanza di Giocatore, è necessario creare una “DefaultGiocatoreFactory” ed invocare il metodo “createGiocatore()”, passandogli i parametri corretti.

L’utilizzo di tale pattern ha portato ad una modifica di “SD inserisciUtente()” in cui non si fa più riferimento alla classe “Giocatore” per la creazione dell’istanza, ma a “DefaultGiocatoreFactory”

Immagine che contiene testo, schermata, numero, Carattere

Descrizione generata automaticamente

### Pattern GoF Observer

Il pattern Observer appartiene ai pattern comportamentali, esso permette di mantenere una lista di osservatori che saranno notificati quando si verificano dei cambiamenti nello stato del soggetto.

Immagine che contiene testo, schermata, linea, Carattere

Descrizione generata automaticamente

Il pattern Observer è stato implementato direttamente nella classe EasyRoom (che funge da Subject), inoltre 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 viene inserita una prenotazione. Gli osservatori ricevono l'ID della prenotazione e l'elenco dei codici fiscali dei partecipanti.

L’utilizzo di tale pattern ha portato ad una modifica di “SD: inserisciPrenotazione”

Immagine che contiene testo, schermata, diagramma, Parallelo

Descrizione generata automaticamente

### Pattern GoF Strategy

Il pattern Strategy permette di definire una serie di algoritmi , incapsularli e renderli intercambiabili. In questo caso, il pattern è stato utilizzato per ottenere delle strategie per il calcolo del costo della prenotazioni di una stanza.

Immagine che contiene testo, schermata, linea, diagramma

Descrizione generata automaticamente

L’applicazione di questo pattern ha portato ad una modifica di “SD: calcoloPrezzo” che per comodità è stato rinominato in “SD: prospettoPrezzo”

Immagine che contiene testo, schermata, diagramma, linea

Descrizione generata automaticamente