Università degli studi di Napoli Parthenope

Progetto di ingegneria del software e interazione uomo-macchina

Biglietteria automatica per cinema

System Design Document

Alessandro di Stefano 0124002276
Daniele Biagio De Luca 0124002504

Sommario

1 - Introduzione

Scopo del sistema

Il sistema in sviluppo è progettato per gestire una Biglietteria automatica di un Cinema, garantendo un'esperienza utente fluida e un'efficace amministrazione delle risorse. Il Cinema, con diverse sale e disponibilità di posti, permette agli utenti di acquistare biglietti interi o ridotti, con prezzi maggiorati nei fine settimana.

Gli amministratori possono aggiornare la programmazione dei film, gestire i prezzi dei biglietti e generare report periodici sulle affluenze e sui ricavi. Gli utenti possono acquistare biglietti pagando in contanti, carta di credito o bancomat, ricevendo un biglietto con dettagli sul film, sala, orario, nome dell'acquirente e costo, con colori diversi per biglietti interi e ridotti. Inoltre, è possibile annullare l'ultima operazione entro 10 minuti.

Il sistema mira a garantire un'efficiente gestione del Cinema e una positiva esperienza utente, combinando funzionalità amministrative e di vendita automatica di biglietti in modo intuitivo e accessibile.

Obiettivi di progettazione

Durante la scoperta dei requisiti, abbiamo specificato i seguenti requisiti non funzionali per il sistema di gestione della Biglietteria automatica per Cinema:

- **Conformità sulle linee guida**: Il sistema software dev'essere sviluppato con Java.
- **Garanzia di modifica dati**: Il sistema deve garantire la visualizzazione, l'aggiornamento, l'inserimento e la rimozione di dati.
- **Modularità del codice**: Il codice del sistema deve essere scritto in modo modulare per facilitare aggiornamenti e manutenzioni future.
- **Garanzia di prestazioni**: Il sistema deve rispondere rapidamente alle operazioni di acquisto e gestione.
- **Scalabilità**: Il sistema deve essere scalabile per gestire l'aumento del numero di utenti e delle transazioni senza degrado delle prestazioni.

Riferimenti:

Si faccia riferimento al RAD del sistema, allegato insieme a questo documento per una maggiore compressione del sistema creato.

Attenzione:

"Per mantenere la focalizzazione sugli aspetti architetturali e di design del core del sistema, abbiamo deciso di omettere le classi relative all'interfaccia grafica, implementate utilizzando JavaFX, dalla presente relazione. La complessità e la vastità delle classi di interfaccia grafica avrebbero richiesto la rappresentazione di oltre 90 diagrammi UML aggiuntivi, rendendo la

documentazione eccessivamente dettagliata e meno accessibile per lo scopo di questo documento, che mira a evidenziare principalmente la logica di business e i pattern di design utilizzati."

Panoramica:

Di seguito viene riportata la versione decomposta e raffinata del diagramma ad oggetti precedentemente visto nel RAD. In particolar modo si precisa che i pattern utilizzati sono: *COMMAND*, *FACTORY METHOD*, *STRATEGY*, *OBSERVER*.

1. **Command:** adottato per gestire le richieste degli utenti e degli amministratori in modo controllato, permettendo operazioni come l'aggiornamento dei film e l'acquisto dei biglietti.

2. **Factory Method:** impiegato per creare diverse tipologie di biglietti, il che rende il sistema flessibile nella gestione di varie caratteristiche dei biglietti come il prezzo o il colore.

3. **Observer:** usato per monitorare cambiamenti significativi (es. vendite di biglietti) e aggiornare automaticamente i sistemi di reportistica, garantendo che le informazioni siano sempre aggiornate.

4. **Strategy:** applicato in due aree distinte, il pagamento e la determinazione dei prezzi, per fornire varietà nelle modalità di pagamento e flessibilità nella gestione dei prezzi.

Pattern Observer

Nel sistema il pattern Observer è usato per gestire e rispondere agli aggiornamenti relativi alle operazioni sui biglietti del cinema. La gestione dei rimborsi dei biglietti è un esempio significativo di come questo modello sia utilizzato per monitorare e reagire a cambiamenti specifici all'interno del registro dei biglietti.

Applicazione del Pattern Observer nel Sistema di Gestione del Cinema:

1. **Observable (AbstractRegistroBiglietti):**

Questa classe astratta funge da base per gli oggetti che possono essere osservati.
Mantiene una lista di osservatori (IReport) e include metodi per aggiungerli (addObserver), rimuoverli (removeObserver), e notificarli (notifyObservers).
Ogni volta che avviene un cambiamento significativo nel registro dei biglietti, come l'aggiunta o l'annullamento di un biglietto, AbstractRegistroBiglietti invoca notifyObservers per avvisare tutti gli osservatori registrati del cambiamento.

2. **Concrete Observable (RegistroBiglietti):**

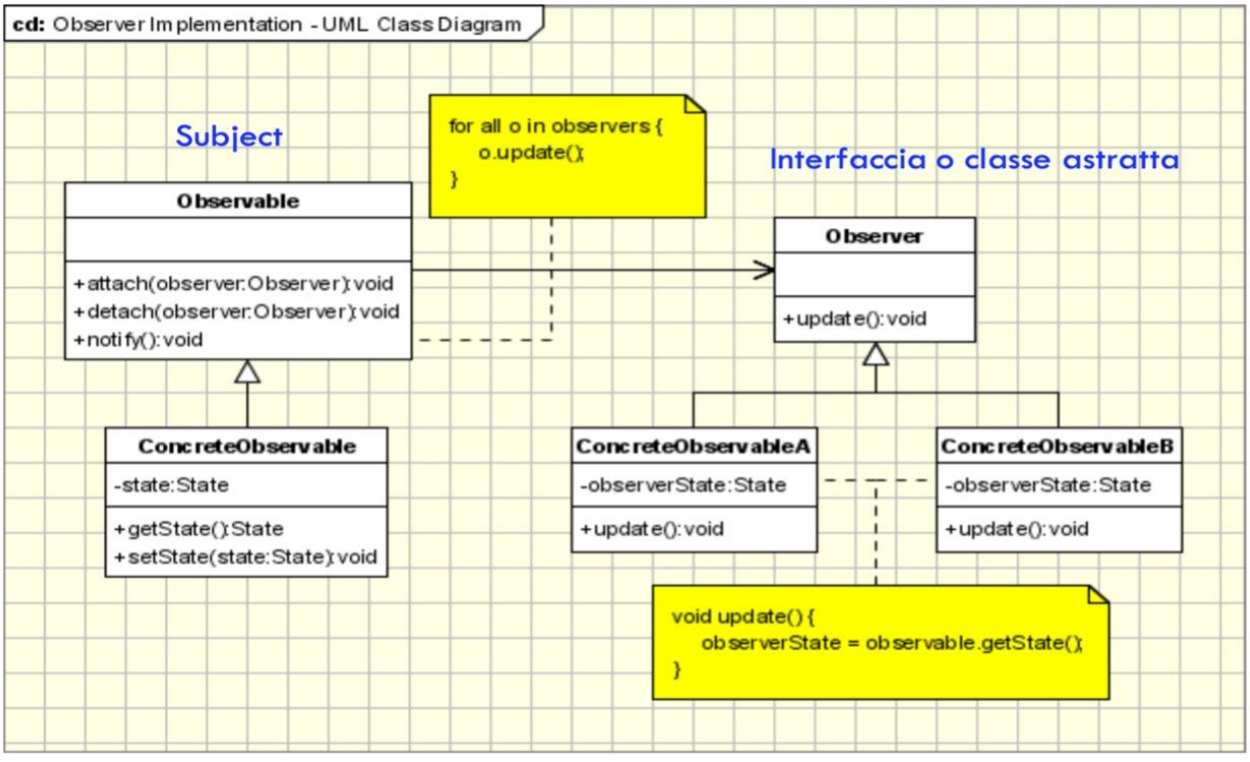
Estende AbstractRegistroBiglietti, implementando la logica specifica per la gestione del

registro dei biglietti del cinema. Questa classe gestisce gli stati concreti che interessano gli observer, come le operazioni di aggiunta e annullamento dei biglietti. Anche se non esplicitamente denominato come Concrete Observable nel diagramma, funge da tale implementando ed estendendo le funzionalità dell'Observable astratto.

3.Observer (IReport):
L'interfaccia IReport definisce i metodi che ogni osservatore deve implementare, in particolare update() e generate(). Il metodo update() viene chiamato dall'Observable quando si verifica un cambiamento, permettendo all'Observer di aggiornare il proprio stato interno o ricalcolare i dati necessari.

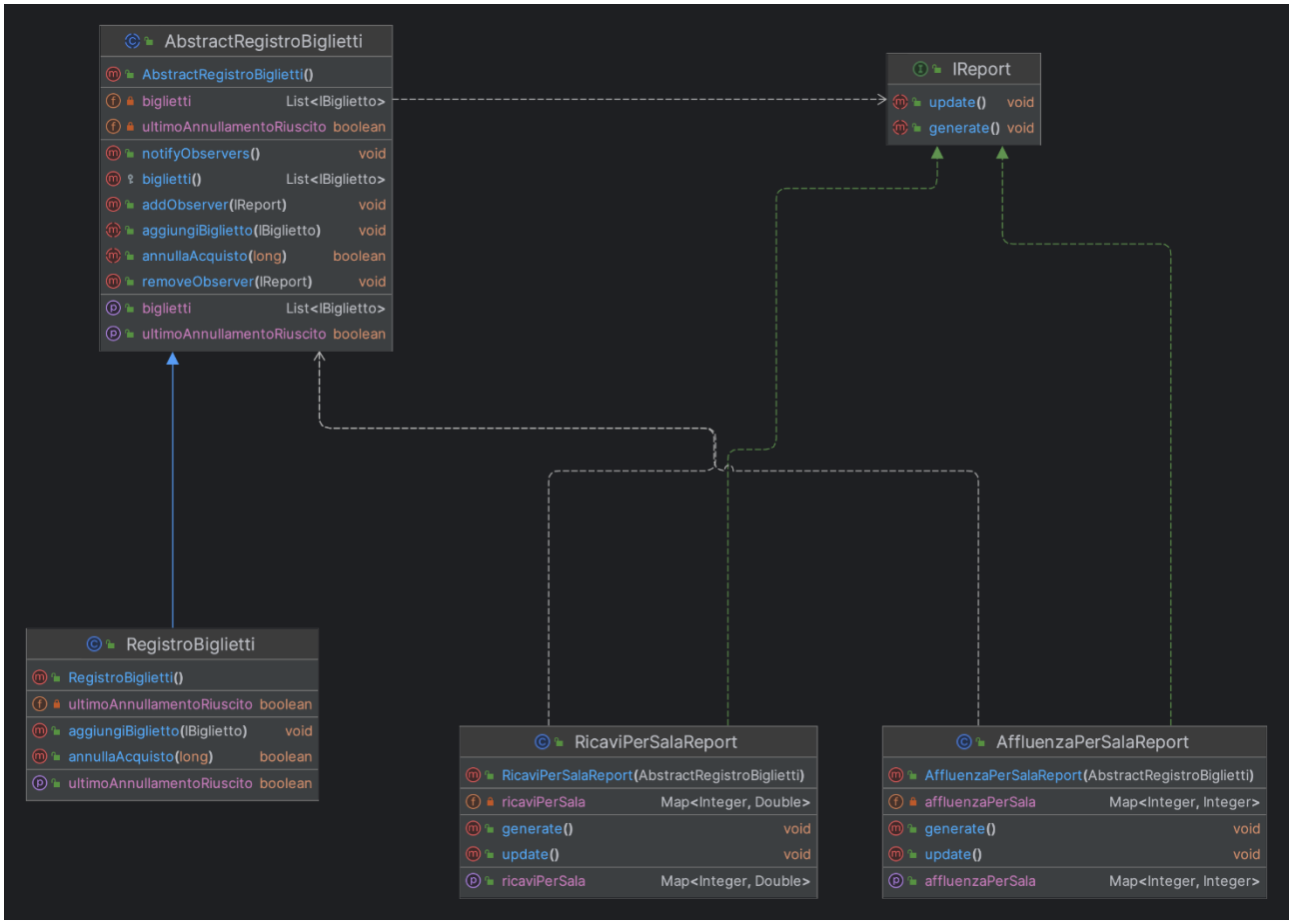
4.Concrete Observers (RicaviPerSalaReport e AffluenzaPerSalaReport):
Queste classi concrete implementano l'interfaccia IReport e definiscono come rispondere ai cambiamenti notificati dall'Observable. RicaviPerSalaReport calcola e tiene traccia dei ricavi per sala, mentre AffluenzaPerSalaReport monitora l'affluenza di pubblico per sala. Entrambe le classi registrano se stesse presso l'Observable utilizzando addObserver e implementano il loro metodo update() per rispondere in modo specifico ai cambiamenti osservati (ad esempio, aggiornando i loro dati interni o ricalcolando i report).

Struttura del Pattern Observer:



Struttura del pattern Observer

IMPLEMENTAZIONE:



Funzionamento del Pattern
Quando si verifica un evento rilevante, come l'annullamento o l'acquisto di un biglietto, RegistroBiglietti chiama notifyObservers(). Questo metodo, a sua volta, itera attraverso tutti gli Observer registrati (come RicaviPerSalaReport e AffluenzaPerSalaReport) e chiama il loro metodo update(), che attiva la logica specifica di aggiornamento dei report.
Il pattern Observer qui implementato consente di mantenere un basso accoppiamento tra la gestione dei dati dei biglietti e la generazione dei report, facilitando modifiche ed estensioni future del sistema senza perturbare gli altri componenti.

Pattern Factory Method

Il pattern Factory Method è utilizzato nel sistema cinema per creare diverse tipologie di biglietti, come i biglietti interi e i biglietti ridotti. Questo pattern consente di definire un'interfaccia per creare un oggetto, ma lascia che le sottoclassi decidano quale classe istanziare, promuovendo così la flessibilità e la riusabilità del codice.

Applicazione del Pattern Factory Method nel Sistema di Gestione del Cinema:

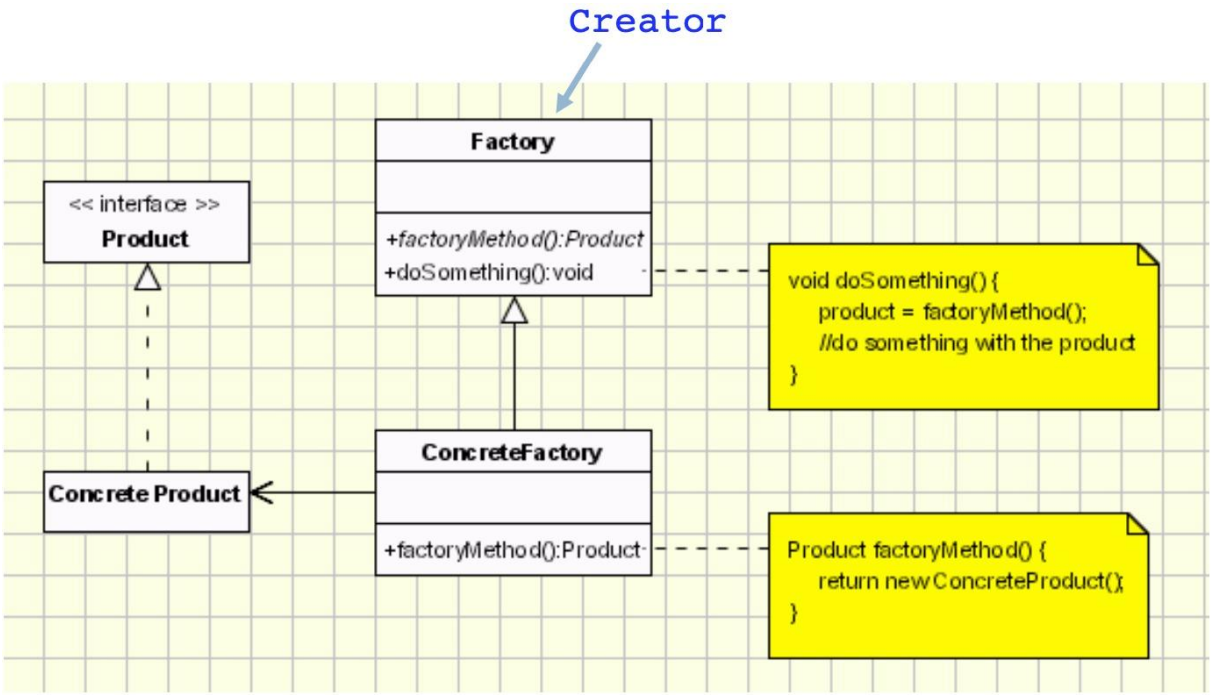
1. **Product (IBiglietto):**
Questa interfaccia rappresenta il tipo di oggetti che il metodo factory crea. Nel sistema, IBiglietto definisce le operazioni comuni che ogni biglietto, come validità, costo e dettagli dello spettacolo, deve supportare.

2. **Concrete Product (BigliettoIntero, BigliettoRidotto):**
Queste classi implementano l'interfaccia IBiglietto e rappresentano i tipi specifici di biglietti che possono essere creati. BigliettoIntero e BigliettoRidotto differiscono per alcune proprietà, come il costo, che può essere più basso per i biglietti ridotti a seconda di alcuni criteri come l'età.

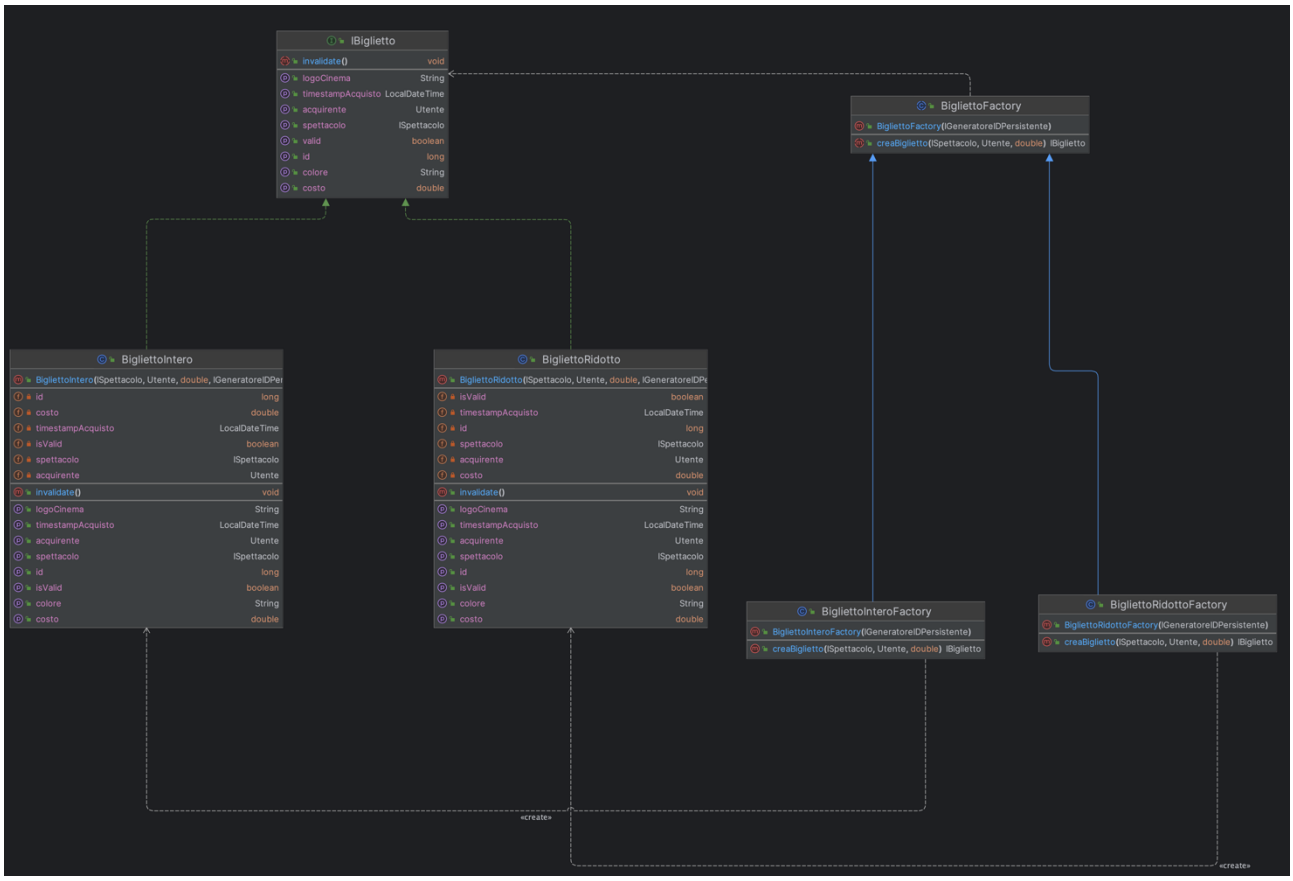
3. **Factory (BigliettoFactory):**
Una classe astratta che dichiara il metodo factory creaBiglietto(...), responsabile per l'istanziazione degli oggetti IBiglietto. Non crea direttamente gli oggetti, ma fornisce il framework entro cui le sue sottoclassi operano.

4. **Concrete Factory (BigliettoInteroFactory, BigliettoRidottoFactory):**
Queste sottoclassi di BigliettoFactory implementano il metodo factory specificando come creare i diversi tipi di biglietti. BigliettoInteroFactory crea un BigliettoIntero, mentre BigliettoRidottoFactory crea un BigliettoRidotto. Ogni factory concreta conosce il tipo di prodotto concreto da creare e come inizializzarlo basandosi sui parametri forniti (come lo spettacolo, l'utente e il prezzo).

Struttura del Pattern Factory Method:



IMPLEMENTAZIONE:



Pattern Command

Il pattern Command è stato implementato nel progetto cinema per gestire le operazioni sia in modalità amministratore che in modalità utente, facilitando la creazione di comandi e procedure complesse attraverso interazioni semplici e dirette.

Applicazione del Pattern Command nel Sistema di Gestione del Cinema:

1. **Client:** Responsabile della creazione e configurazione degli oggetti Command. Nel contesto del cinema, questo ruolo è spesso assunto dalla parte dell'applicazione che interagisce direttamente con l'utente, che può essere un'interfaccia grafica o un controller.

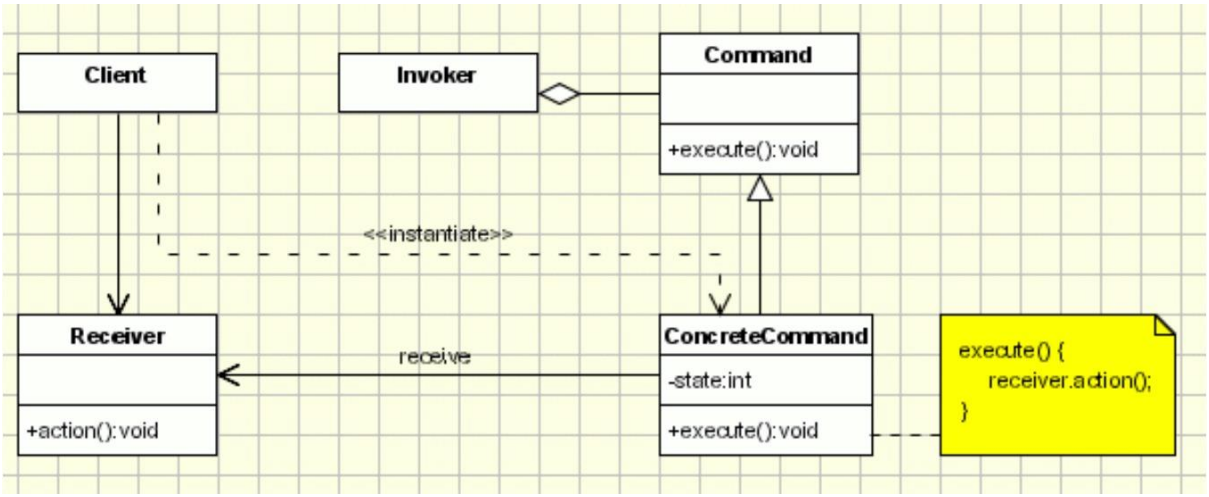
2. **Invoker:** Mantiene un riferimento a un oggetto Command e chiama il suo metodo execute(). Nell'implementazione del sistema del cinema, l'Amministratore e l'Utente possono funzionare come Invokers, richiedendo l'esecuzione di specifici comandi.

3. **Command** (ICommand): Un'interfaccia che dichiara il metodo execute(), il quale definisce l'azione da eseguire.

4. **ConcreteCommand**: Classi che implementano l'interfaccia Command e definiscono l'azione concreta da eseguire. Questi possono includere comandi come AggiungiFilmCommand, GeneraReportRicaviCommand, ModificaSpettacoloCommand, e RimuoviSalaCommand.

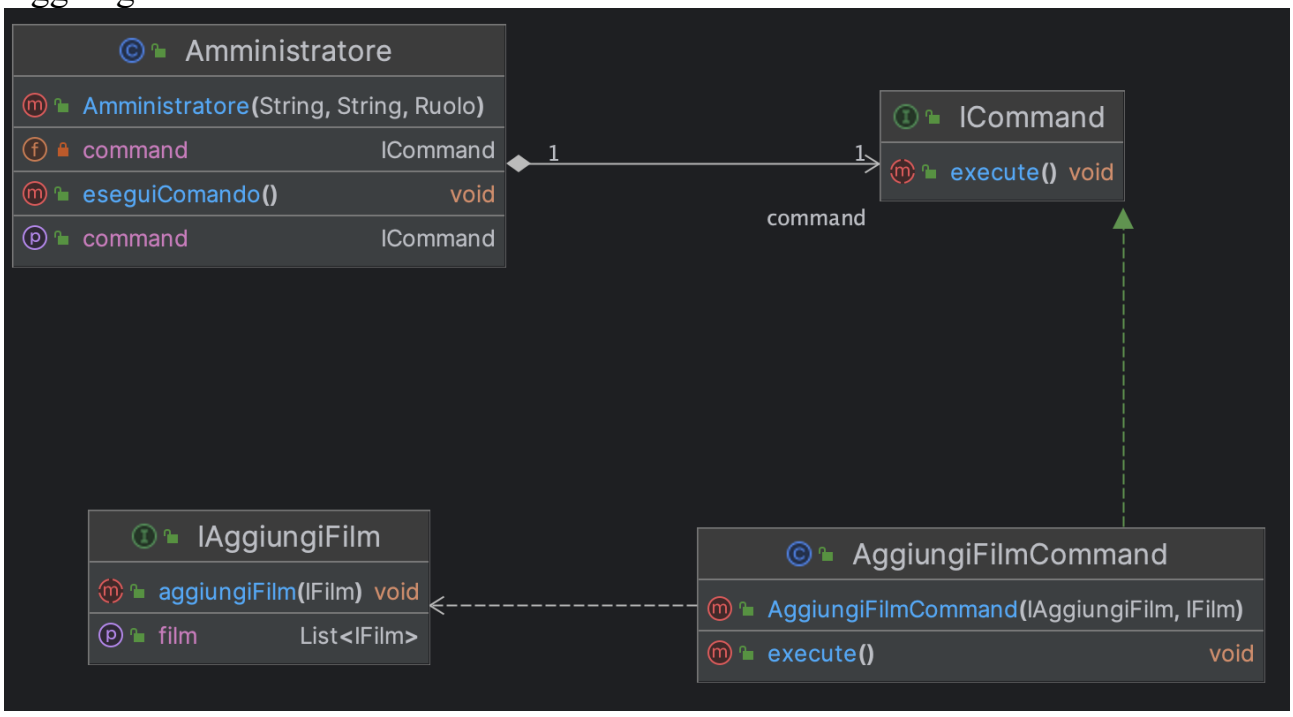
5. **Receiver:** Gli oggetti sui quali vengono eseguite le azioni concrete. Questi oggetti possiedono la logica effettiva necessaria per eseguire le azioni richieste dai comandi.

Struttura del Pattern Command:

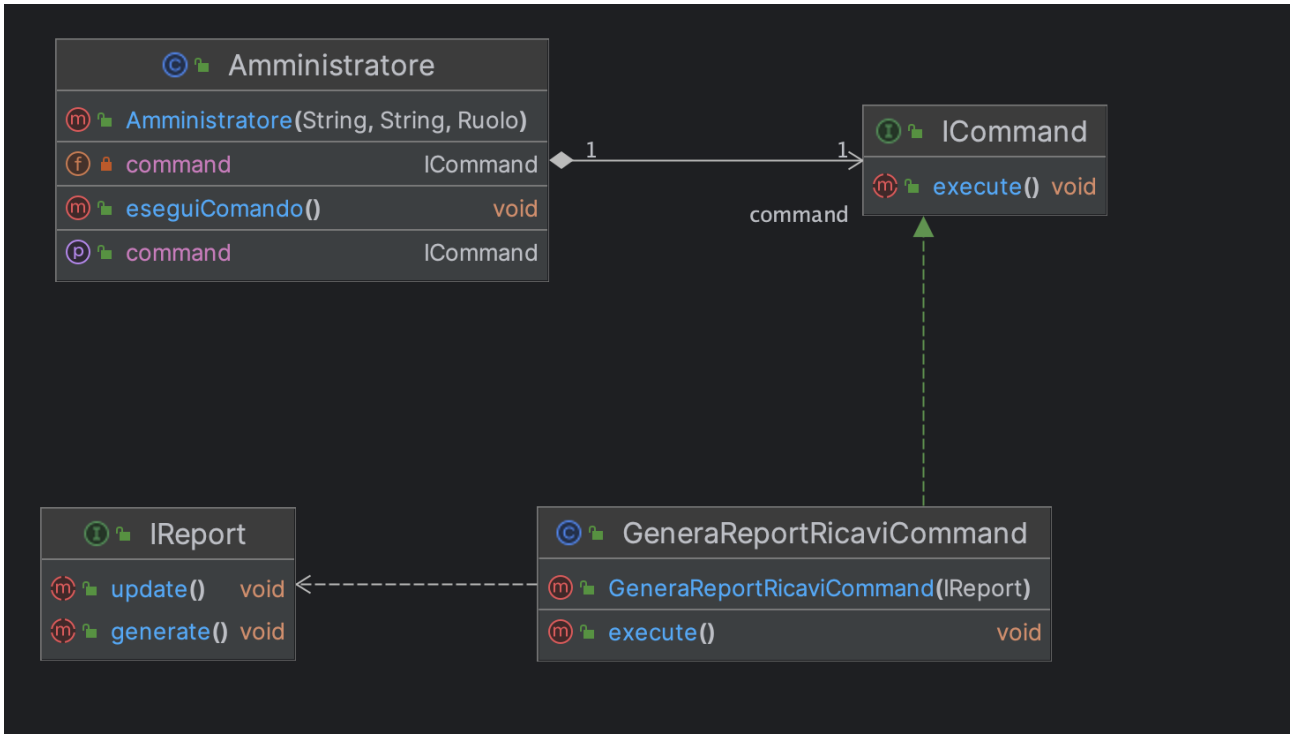


IMPLEMENTAZIONE Command Admin

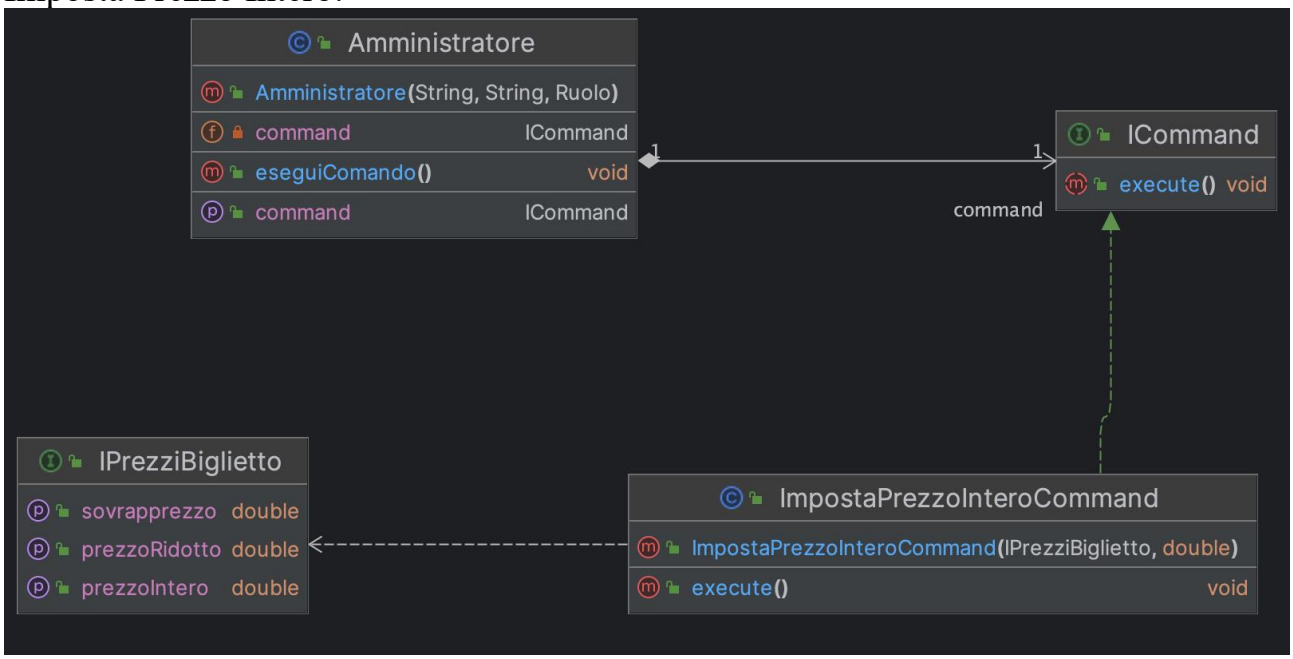
Aggiungi film:



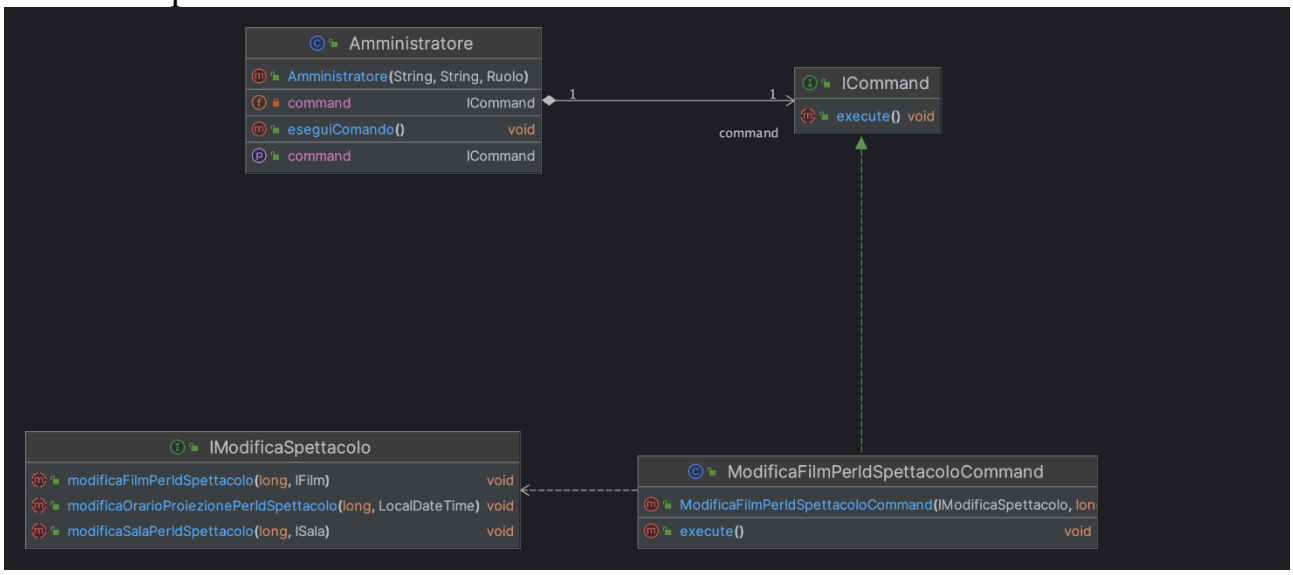
Genera Ricavi:



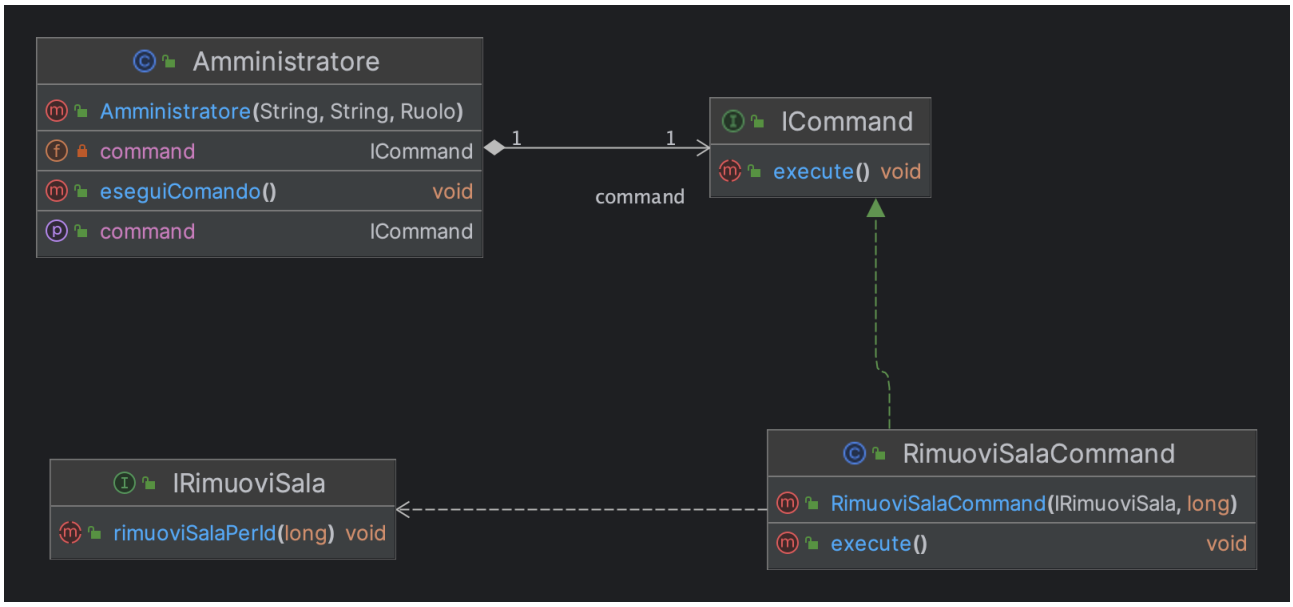
Imposta Prezzo Intero:



Modifica Spettacolo:

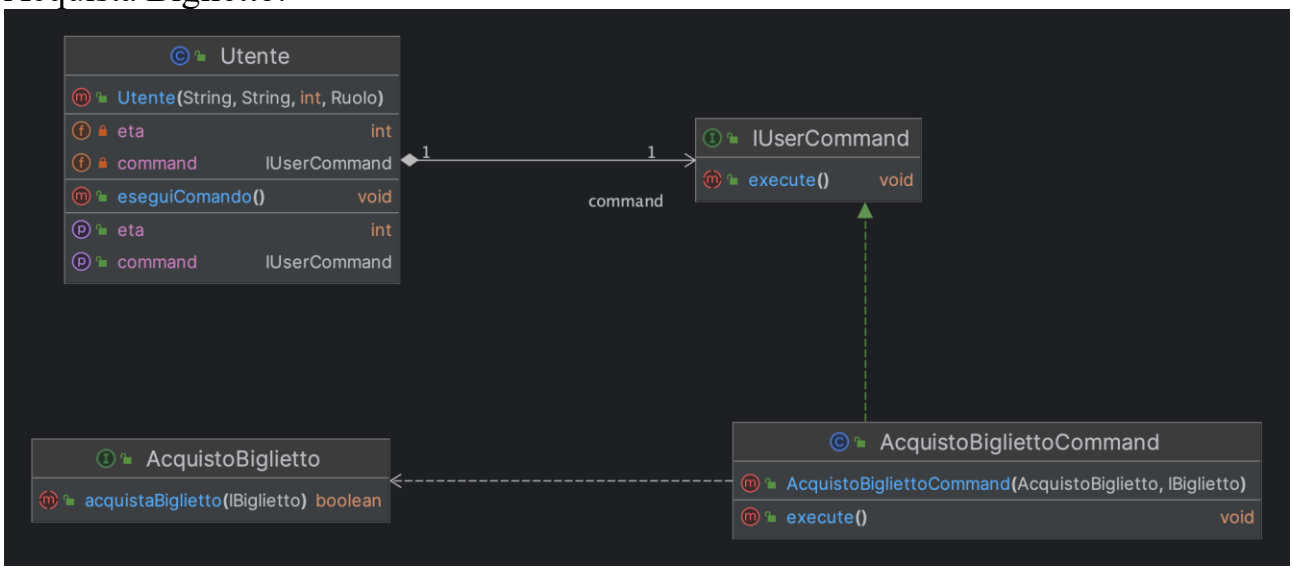


Rimuovi Sala:

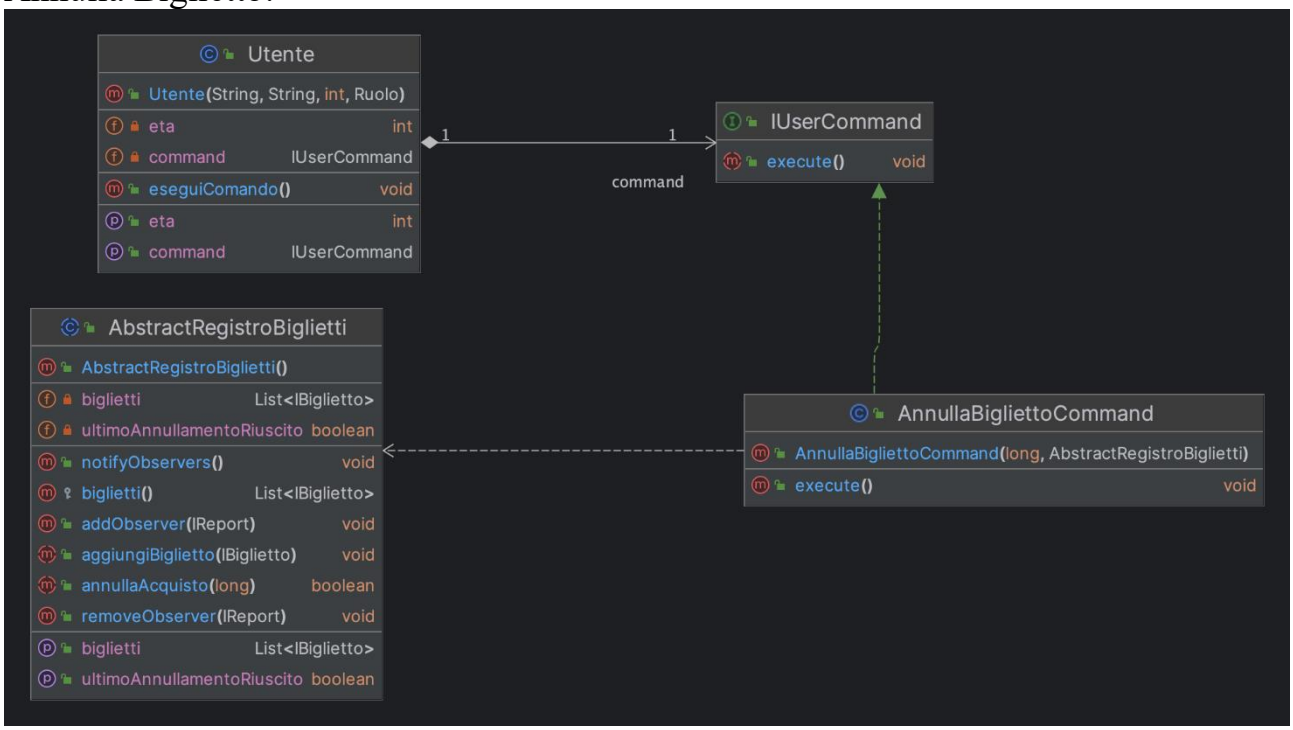


IMPLEMENTAZIONE Command Utente

Acquista Biglietto:



Annulla Biglietto:



PATTERN STRATEGY

*Il pattern Strategy è stato implementato due volte per gestire due aspetti cruciali: i **metodi di pagamento** e la **definizione dei prezzi dei biglietti**.*

Pattern Strategy (Strategie di prezzo)

Il pattern Strategy per la gestione dei prezzi dei biglietti nel sistema di cinema permette di adattare e modificare dinamicamente la strategia di prezzatura in base a variabili specifiche, come il giorno della settimana o eventi speciali.

Applicazione del Pattern Strategy nel Sistema di Gestione del Cinema:

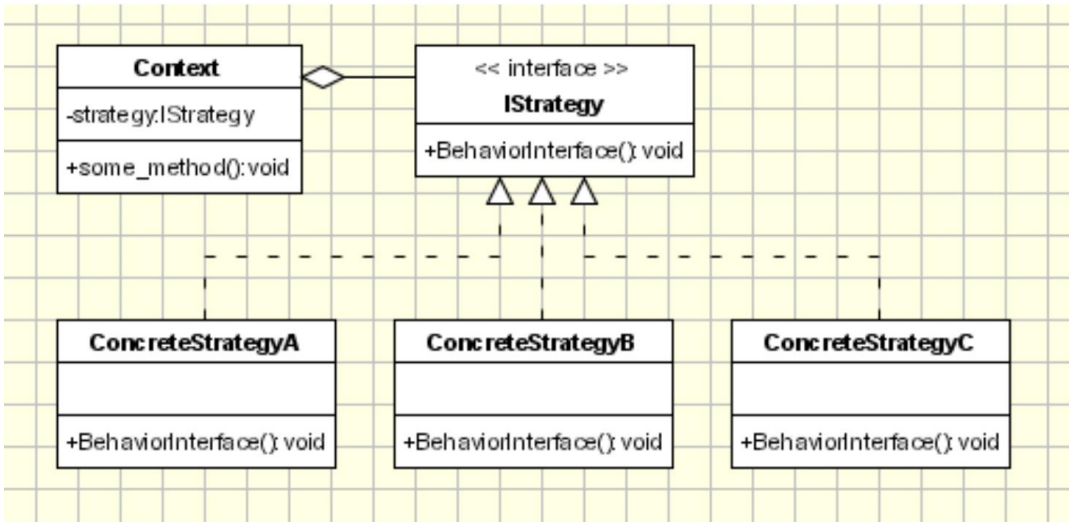
1. **Context (Context):** Questa classe mantiene un riferimento alla strategia di prezzo corrente (**IPrezzoStrategy**). Agisce come un mediatore tra il client e le strategie concrete, delegando l'esecuzione delle operazioni di calcolo del prezzo alla strategia attualmente associata.

2. **Strategy Interface (IPrezzoStrategy):** Un'interfaccia che definisce un metodo comune execute() che tutte le strategie concrete devono implementare. Questo metodo è responsabile per calcolare il prezzo finale del biglietto in base alla strategia applicata.

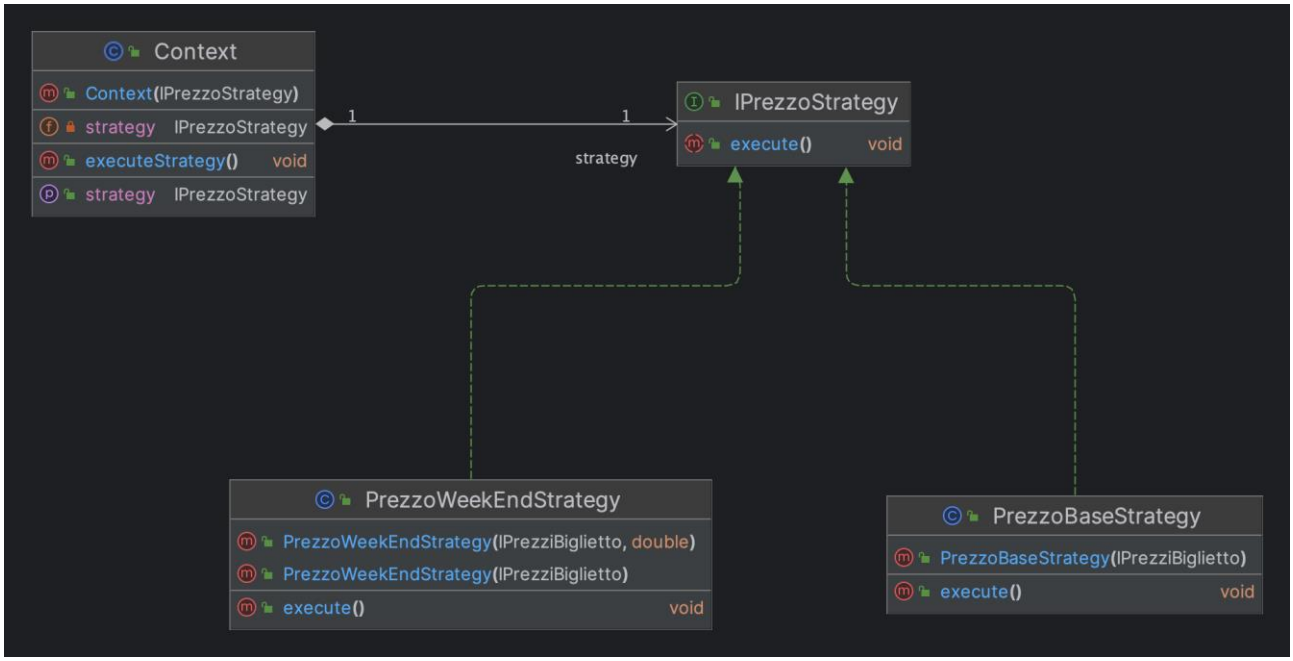
3. **Concrete Strategies:**

- **PrezzoBaseStrategy:** Applica una tariffazione standard dei biglietti, utilizzata durante i giorni feriali o in condizioni normali.
- **PrezzoWeekEndStrategy:** Aumenta il prezzo dei biglietti durante il fine settimana, riflettendo la maggiore domanda o offerte speciali legate a eventi o premiere.

Struttura del Pattern Strategy:



IMPLEMENTAZIONE:



Pattern Strategy (Strategie di Pagamento)

Il pattern Strategy è stato implementato per gestire i diversi metodi di pagamento.

Applicazione del Pattern Strategy nel Sistema di Gestione del Cinema:

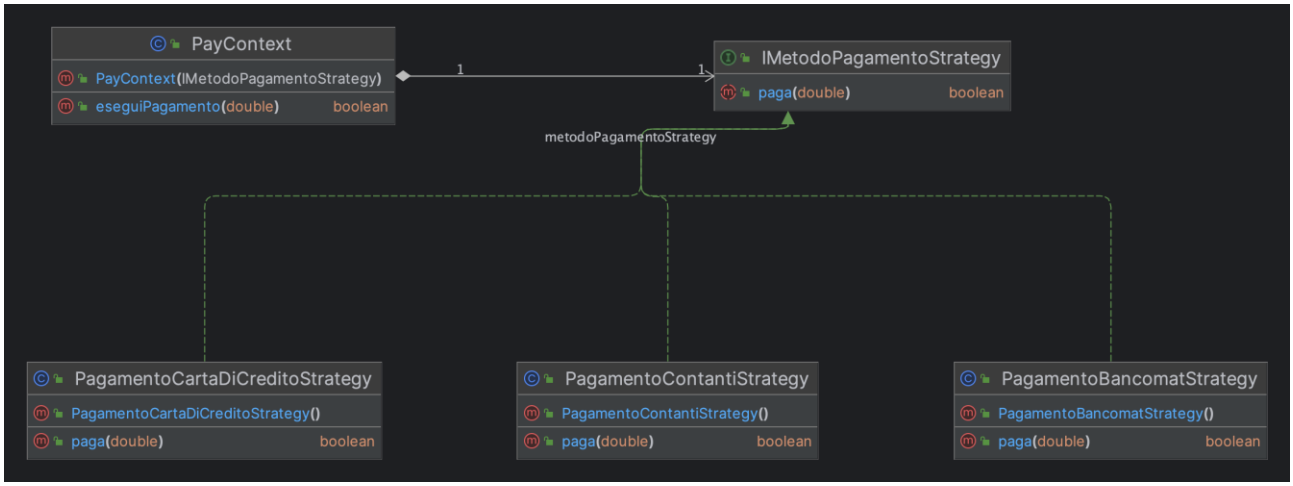
1. **Context (PayContext):** Agisce come il contesto che mantiene un riferimento alla strategia di pagamento attuale **(IMetodoPagamentoStrategy)**. Facilita l'esecuzione del pagamento delegando l'azione alla strategia scelta.

2. **Strategy Interface (IMetodoPagamentoStrategy)**: Questa interfaccia definisce un metodo paga(double) che tutte le strategie di pagamento devono implementare. Il metodo è responsabile per elaborare il pagamento e confermare il risultato dell'operazione.

3. **Concrete Strategies**:

- **PagamentoCartaDiCreditoStrategy:** Gestisce i pagamenti effettuati tramite carta di credito, implementando verifiche di sicurezza e autorizzazione specifiche per questo tipo di transazioni.
- **PagamentoBancomatStrategy:** Adatta per i pagamenti tramite bancomat, questa strategia potrebbe interfacciarsi con sistemi bancari per verificare la disponibilità di fondi e completare la transazione.
- **PagamentoContantiStrategy:** Utilizzata per i pagamenti in contanti, questa strategia gestisce la ricezione fisica del denaro e la registrazione del pagamento.

IMPLEMENTAZIONE:



2 – Sistema corrente

Il seguente sistema è un progetto Greenfield ispirato ad un progetto già esistente chiamato **Veezi**.
Veezi è un software che mette a disposizione diversi strumenti per gestire una biglietteria automatica di un cinema. In questa sezione, mettiamo a confronto entrambi i sistemi, con lo scopo di capire meglio quali caratteristiche e funzionalità esistenti possono soddisfare al meglio le esigenze di un cinema, sia per gli utenti che per gli amministratori.

Il confronto verrà articolato sui seguenti punti chiave:

Finalità e scopo

Sistema proposto

È progettato per la gestione di una biglietteria automatica in un cinema. Consente sia la gestione amministrativa che l'acquisto dei biglietti da parte degli utenti.
Specificamente focalizzato su un singolo cinema con n sale e prevede operazioni come la vendita di biglietti interi e ridotti, con prezzi variabili nel weekend.

Veezi

È progettato per cinema indipendenti. Consente una configurazione iniziale attraverso una prova gratuita.
Focalizzato sulla gestione completa di un cinema, dal caricamento dei film alla gestione dei biglietti e della disposizione dei posti.

Funzionalità amministrative

Sistema proposto

Consente l’aggiornamento dei film proiettati, la modifica dei prezzi dei biglietti, e la generazione di report sulle affluenze e visualizzazione dei ricavi.

Veezi

Consente l’inserimento di dettagli generali sul cinema (nome, indirizzo, numero di telefono), l’aggiunta di dettagli sugli spettacoli (numero totale di posti, posti standard, riservati, per disabili), il caricamento di film dal database principale di Veezi, l’importazione e creazione di nuovi tipi di biglietti.

Funzionalità Utente

Sistema proposto

Consente l’acquisto di biglietti (interi o ridotti) con pagamento in contanti, carta di credito o bancomat, emissione del biglietto con dettagli (film, sala, ora di proiezione, nome, cognome, costo) e l’annullamento dell'ultima operazione entro 10 minuti.

Veezi

Non pervenute. Si deduce sia possibile acquistare biglietti.

Reportistica e Analisi

Sistema proposto

Genera report periodici sulle affluenze nelle sale e sui ricavi, permettendo un'analisi dettagliata delle performance del cinema.

Veezi

Genera report in base a un periodo scelto dall’amministratore; consente l’analisi dei ricavi giornaliero e di aggiungere note.

3- Sistema proposto

Stile architetturale

Lo stile architetturale scelto è il **Model-View-Controller** ed è stato applicato come segue:

Model

Il modello gestisce i dati relativi ai film, sale, spettacoli e biglietti. Si occupa di:

- **Aggiunta ed eliminazione di film** con classi e metodi per aggiungere e rimuovere film dal database.
- **Gestione delle sale** con classi e metodi per aggiungere e rimuovere sale.
- **Gestione degli spettacoli**: con classi e metodi per programmare, annullare e modificare spettacoli.
- **Impostazione dei prezzi dei biglietti** con classi e metodi per impostare i prezzi.
- **Acquisto e annullamento di biglietti** con classi e metodi per acquistare e annullare biglietti.
- **Logica di business**: Implementa le regole relative alla gestione delle risorse del cinema.

View

La vista è realizzata utilizzando JavaFX e include le seguenti componenti:

- **Interfaccia amministrativa**: Scene e controlli JavaFX per gestire film, sale, spettacoli, prezzi dei biglietti e ricavi.
- **Interfaccia utente**: Scene e controlli JavaFX per visualizzare film disponibili, spettacoli, acquistare e annullare biglietti.

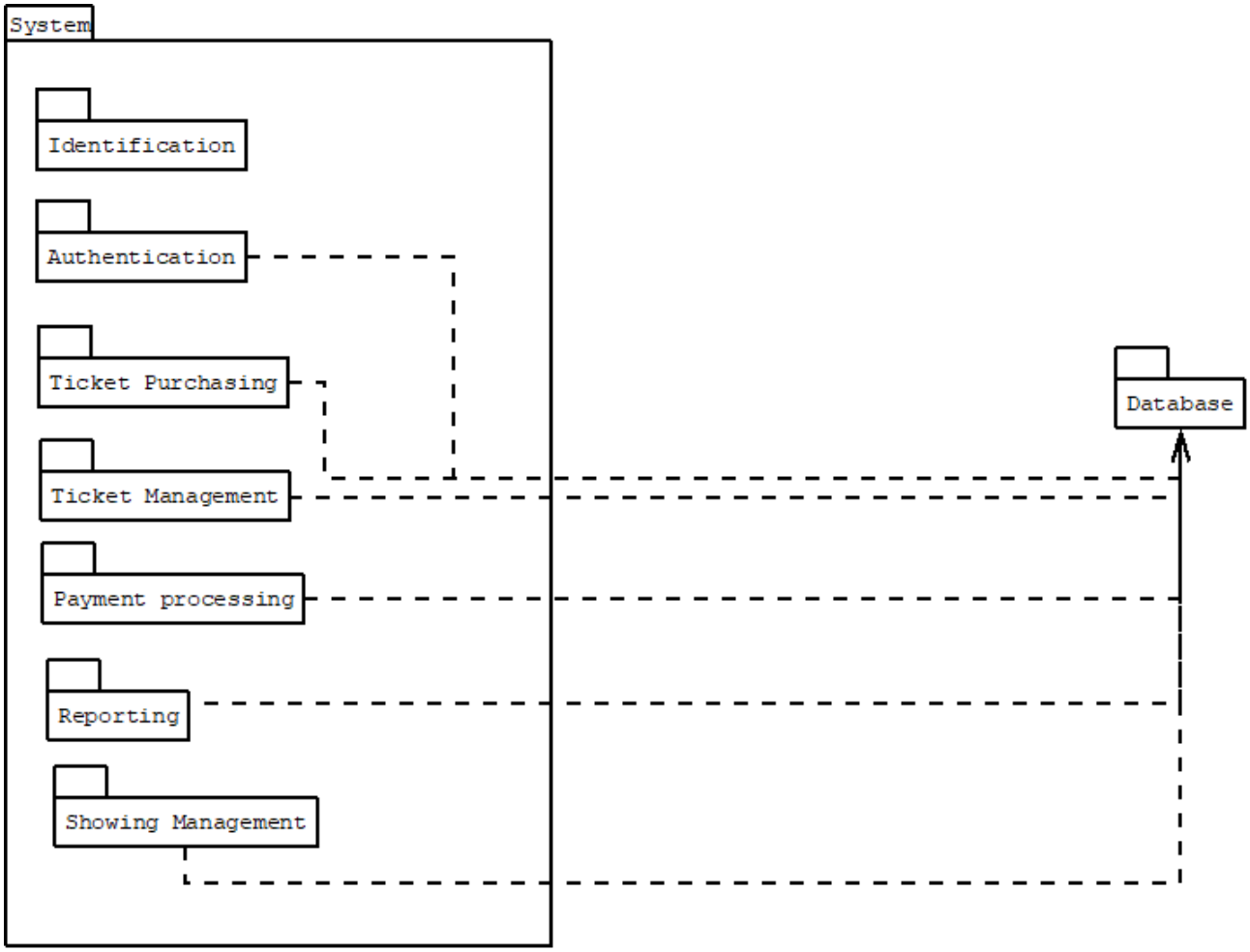
Controller

Il controller gestisce le interazioni tra la vista e il modello. Riceve l'input dall'utente, elabora le richieste e aggiorna il modello di conseguenza. Poi aggiorna la vista per riflettere i cambiamenti.

Perché abbiamo utilizzato questo approccio?

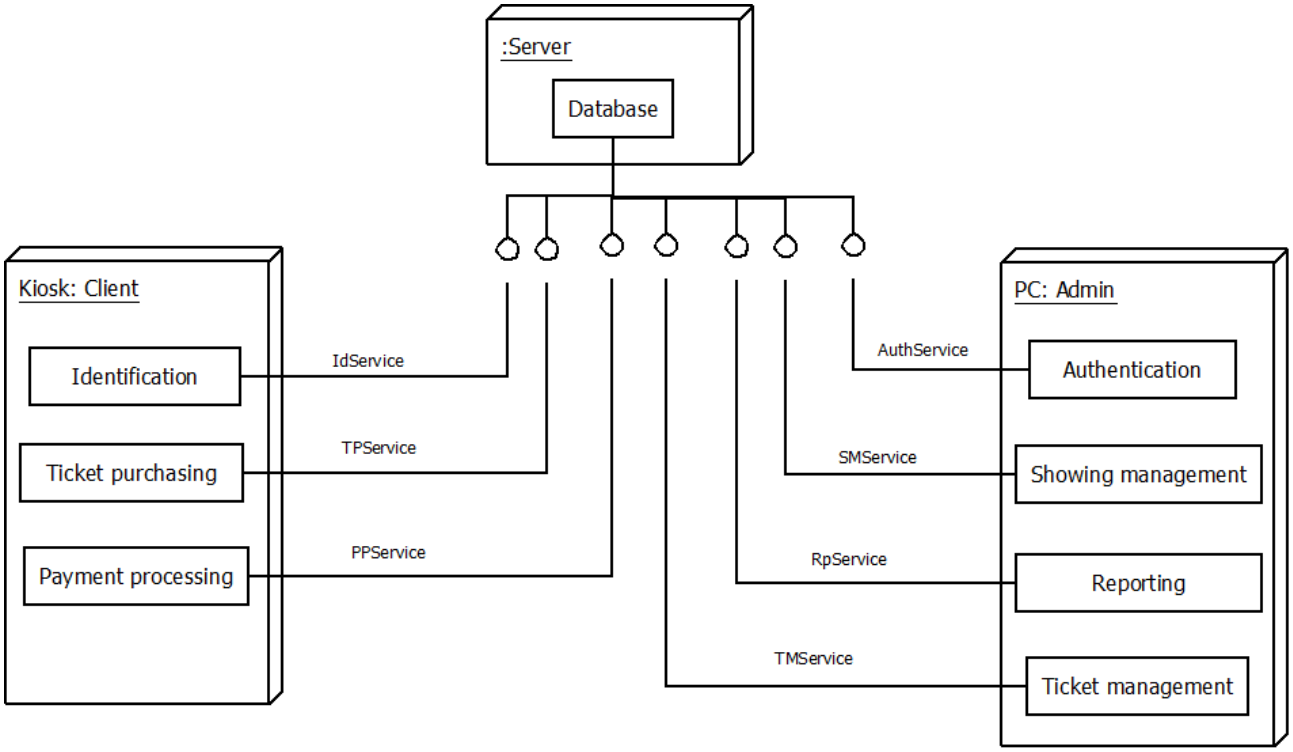
Questo approccio garantisce una gestione chiara e organizzata delle diverse funzionalità dell'applicazione di biglietteria automatica del cinema, facilitando la manutenzione e l'estensibilità del codice. La separazione dei compiti tra modello, vista e controller rende l'applicazione più robusta e modulare.

Decomposizione sottosistema



Identification: area del sistema in cui viene gestita l'identificazione del cliente prima dell'acquisto del biglietto.
Authentication: area del sistema che gestisce l'autenticazione dell'amministratore.
Ticket Purchasing: area del sistema che consente al cliente di acquistare biglietti.
Ticket Management: area del sistema che gestisce la modifica dei prezzi dei biglietti da parte dell'amministratore.
Payment Processing: gestisce il pagamento dei biglietti attraverso diverse modalità.
Reporting: area del sistema esclusiva all'amministratore. Gestisce le operazioni riguardanti l'elaborazione del report delle affluenze e degli incassi.
Showing Management: area del sistema che consente all'amministratore di gestire gli spettacoli e gli elementi relativi ad essi (film, sale).

Hardware/Software mapping



Gestione dei dati persistenti

Per il seguente sistema, si è optato l'utilizzo di un database relazionale per gestire i dati riguardanti gli spettacoli, i biglietti venduti, le transazioni effettuate ecc… Inoltre, le operazioni che si possono effettuare con un database relazionale si rivelano particolarmente utili in diverse occasioni: per esempio nel caso della generazione del report.

Controllo accessi e sicurezza

Oggetti/Attori	Amministratore	Cliente
Identification		identifyClient()
Authentication	login()	
Ticket Purchasing		buyTicket(), cancelPurchase()
Ticket Management	modifyReducedFare(), modifyFullPrice(), modifyWeekendSurcharge()	
Payment processing		processPayment()
Reporting	generateReport()	
Showing Management	addFilm(), removeFilm(), addTheater(), removeTheater(), addShowing(), deleteShowing(), modifyShowing().	

4 – Condizioni limite

- Gestione di un alto numero di richieste contemporanee, specialmente durante i periodi di punta (ad esempio, fine settimana o prime visioni).
- Ottimizzazione delle risorse per evitare rallentamenti o crash del sistema.
- Gestione accurata della disponibilità dei posti in tempo reale per evitare overbooking.
- Aggiornamento istantaneo della disponibilità dei posti quando vengono acquistati o cancellati i biglietti.
- Funzionamento offline in caso di interruzione della connessione internet, con sincronizzazione successiva quando la connessione è ripristinata.
- Ridondanza e backup dei dati per prevenire la perdita di informazioni.
- Facilità d'uso per utenti di tutte le età e competenze tecniche.
- Resistenza agli input errati o incompleti da parte degli utenti.
- Gestione sicura delle transazioni finanziarie, supportando vari metodi di pagamento (carte di credito, debito, portafogli elettronici, ecc.).
- Conformità alle normative sulla sicurezza dei dati (es. GDPR, PCI-DSS).
- Supporto per utenti con disabilità, come comandi vocali o interfaccia compatibile con screen reader.
- Disponibilità di lingue multiple per servire una clientela diversificata.
- Capacità di rilevare e gestire errori hardware o software, come malfunzionamenti del touch screen, problemi di stampa dei biglietti, ecc.
- Meccanismi di recupero e ripristino per minimizzare i tempi di inattività.
- Possibilità di aggiornare il software senza interrompere il servizio.
- Manutenzione programmata con impatto minimo sugli utenti.
- Capacità di espandere il sistema per gestire più schermi o cinema in una rete.
- Integrazione con altri sistemi di gestione, come quelli per la prenotazione online.
- Funzionalità per raccogliere feedback degli utenti e rispondere alle domande comuni.
- Accesso rapido a supporto tecnico in caso di problemi.

5 - Glossario

- **JavaFX**: Una libreria Java per la creazione di interfacce utente ricche e moderne, utilizzata per lo sviluppo dell'interfaccia dell'applicazione di biglietteria automatica.

- **Pattern Command**: Un pattern di progettazione che consente di incapsulare una richiesta come un oggetto, permettendo di parametrizzare i client con code di richieste, log delle richieste e supporto per operazioni annullabili.

- **Pattern Factory Method**: Un pattern di progettazione creazionale che fornisce un'interfaccia per creare oggetti in una superclasse, consentendo alle sottoclassi di modificare il tipo di oggetti che saranno creati.

- **Pattern Strategy**: Un pattern di progettazione comportamentale che consente di definire una famiglia di algoritmi, incapsularli e renderli intercambiabili, permettendo agli algoritmi di variare indipendentemente dal client che li utilizza.

- **Pattern Observer**: Un pattern di progettazione comportamentale che definisce una dipendenza uno-a-molti tra gli oggetti, in modo tale che quando un oggetto cambia stato, tutti i suoi dipendenti vengono notificati e aggiornati automaticamente.

- **Client**: L'utente finale che interagisce con il sistema di biglietteria automatica per acquistare i biglietti del cinema.

- **Identification**: Il processo di riconoscimento dell'utente all'interno del sistema, utilizzando credenziali come Nome, Cognome ed Età.

- **Ticket Purchasing**: Il processo di selezione e acquisto dei biglietti da parte del cliente per le proiezioni cinematografiche tramite l'applicazione di biglietteria automatica.

- **Payment Processing**: La gestione delle transazioni finanziarie per l'acquisto dei biglietti, inclusi vari metodi di pagamento come carte di credito, contanti e Bancomat.

- **Authentication**: Il processo di verifica delle credenziali dell'amministratore per garantire l'accesso sicuro al sistema e alle funzionalità riservate.

- **Showing Management**: Area del sistema che consente all’amministratore di gestire gli spettacoli e gli elementi relativi ad essi (film, sale, spettacoli).

- **Reporting**: La generazione di report sulle vendite dei biglietti, sull'affluenza e altre metriche rilevanti per la gestione del cinema.

- **Ticket Management**: La gestione completa dei biglietti, inclusi il prezzo e il monitoraggio dei biglietti venduti.