Università degli studi di Urbino

Dipartimento di Scienze Pure e Applicate
Corso di Laurea in Informatica Applicata

Relazione progetto Programmazione a oggetti

di Gambini Edoardo matricola : 314901
Fraternali Matteo matricola : 316637
Anno di corso: terzo

Anno accademico 2023/2024 - Sessione Autunnale

1 Analisi

Il nostro progetto si propone di sviluppare un software per simulare la gestione di un albergo. L'applicazione è progettata per gestire vari aspetti operativi dell'hotel, suddivisi in sei figure manageriali, ciascuna con compiti specifici. Ogni figura manageriale può essere gestita dal manager principale dell'hotel che tramite un interfaccia grafica potrà decidere agevolmente a quale manager vuole attingere.

Figure Manageriali e le Loro Funzionalità

1. **ActivityManager**: Questo gestore si occupa di tutte le attività offerte dall'albergo, come escursioni ed eventi. È responsabile per la creazione e la gestione delle attività, inclusa la modifica e la rimozione di quelle esistenti. Inoltre, fornisce una panoramica delle attività disponibili e si occupa della memorizzazione di queste informazioni in un file per garantire la loro persistenza.
2. **BillingManager**: Il gestore dei conti è incaricato di tutto ciò che riguarda la fatturazione e le spese. Crea e gestisce le fatture per i clienti, registra le spese aziendali e si occupa dell'elaborazione dei pagamenti. Monitora anche i pagamenti in sospeso e produce report finanziari annuali. Tutti questi dati sono salvati su file per garantire la loro conservazione e accessibilità futura.
3. **CustomerManager**: Questa figura gestisce tutte le informazioni relative ai clienti dell'albergo. Si occupa di registrare nuovi clienti, visualizzare e aggiornare i loro dettagli, e rimuovere quelli che non sono più attivi. Inoltre, gestisce i reclami dei clienti e assicura che tutte le informazioni sui clienti siano conservate in file per garantirne la persistenza.
4. **ReservationManager**: Il gestore delle prenotazioni e delle stanze si occupa di creare e gestire le prenotazioni delle stanze per i clienti. Fornisce anche dettagli sulle prenotazioni esistenti e gestisce la cancellazione e la modifica di prenotazioni. Tiene traccia delle stanze disponibili e gestisce l’aggiunta di nuove stanze, incluse le operazioni di caricamento e salvataggio delle informazioni relative alle stanze in file.
5. **EmployeeManager**: Questa figura è responsabile della gestione dei dipendenti dell'albergo. Aggiunge nuovi dipendenti, visualizza e aggiorna le loro informazioni e si occupa della loro rimozione dal sistema. Monitora anche la presenza quotidiana dei dipendenti e conserva tutte le informazioni sui dipendenti e sui record di presenza in file per garantirne la persistente accessibilità.
6. **ServiceManager**: Il gestore dei servizi aggiuntivi si occupa di tutti i servizi offerti dall'albergo, come spa e ristorazione. Gestisce l’aggiunta, la modifica e la rimozione dei servizi, e fornisce dettagli sui servizi disponibili. Tutti i dati sui servizi sono memorizzati in file per garantire la loro persistenza.

1.1 Requisiti

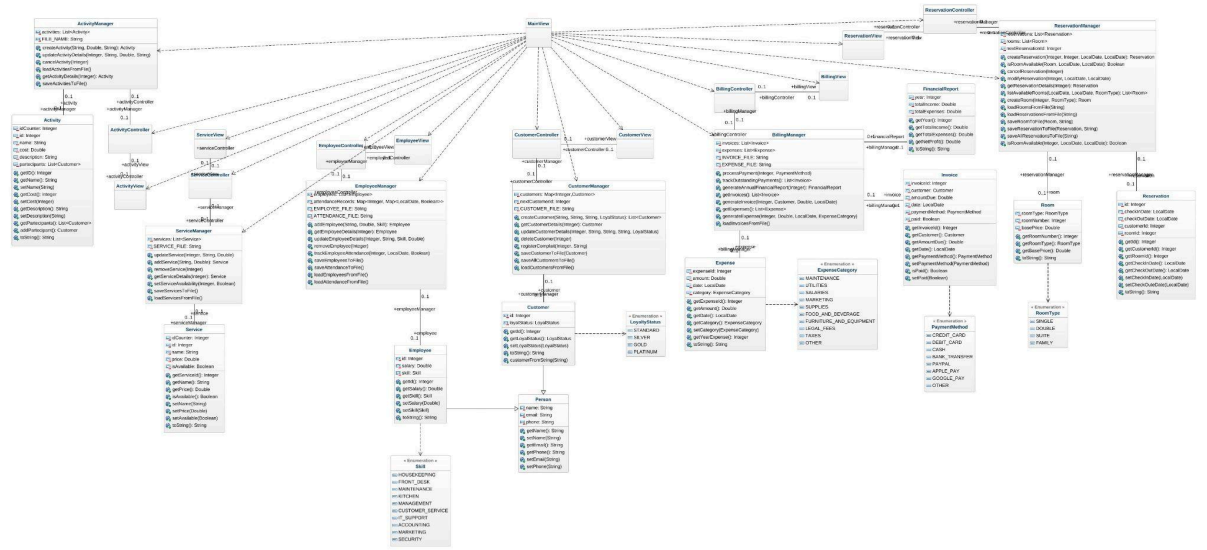
L'applicazione per la gestione dell'albergo deve rispettare i seguenti requisiti principali:

1. **Gestione delle Attività**
    - **Creazione di Attività:** Il manager deve poter creare nuove attività, specificando nome, costo e descrizione.
    - **Aggiornamento dei Dettagli di un'Attività:** Il manager deve avere la possibilità di modificare i dettagli di attività esistenti.
    - **Cancellazione di un'Attività:** Il manager deve poter rimuovere attività non più necessarie dal sistema.
    - **Visualizzazione dei dettagli di una Attività:** Il manager deve poter visualizzare i dettagli dell'attività da lui richiesta
    - **Salvataggio su File:** Tutte le attività devono essere salvate su file per garantire la persistenza dei dati.
2. **Gestione dei Conti**
    - **Generazione di Fatture:** Il manager deve poter generare fatture per i clienti
    - **Generazione di Spese:** Il manager deve poter registrare le spese aziendali.
    - **Elaborazione dei Pagamenti:** Il manager deve poter gestire i pagamenti delle fatture, aggiornando lo stato a "pagato".
    - **Tracciamento dei Pagamenti in Sospeso:** Il manager deve poter monitorare le fatture non ancora saldate.
    - **Generazione di Report Finanziari Annuali:** Il manager deve poter generare report finanziari annuali che riepiloghino entrate e spese.
    - **Salvataggio delle Fatture e delle Spese:** Tutte le fatture e spese devono essere salvate su file per garantire la persistenza dei dati.
3. **Gestione dei Clienti**
    - **Creazione di Clienti:** Il manager deve poter registrare nuovi clienti con nome, email, telefono e stato di fedeltà.
    - **Visualizzazione dei Dettagli di un Cliente:** Il manager deve poter consultare le informazioni dei clienti.
    - **Aggiornamento dei Dettagli di un Cliente:** Il manager deve poter aggiornare le informazioni dei clienti esistenti.
    - **Cancellazione di un Cliente:** Il manager deve poter rimuovere clienti dal sistema.
    - **Registrazione di Reclami:** Il manager deve poter registrare e gestire i reclami dei clienti.
    - **Salvataggio di Clienti su File:** Le informazioni sui clienti devono essere salvate su file per garantire la persistenza dei dati.

4. **Gestione delle Prenotazioni e delle Stanze**
    - **Creazione di Prenotazioni:** Il manager deve poter effettuare prenotazioni per i clienti, specificando le date di check-in e check-out.
    - **Visualizzazione dei Dettagli di una Prenotazione:** Il manager deve poter consultare le informazioni di prenotazioni esistenti.
    - **Cancellazione di una Prenotazione:** Il manager deve poter annullare prenotazioni non più necessarie.
    - **Modifica di una Prenotazione:** Il manager deve poter aggiornare le date e i dettagli delle prenotazioni esistenti.
    - **Verifica della disponibilità di un determinato tipo di stanza:** Il manager deve poter verificare quali stanze sono disponibili in un determinato range temporale, specificando il tipo di stanza che si vuole cercare
    - **Creazione di Stanze:** Il manager deve poter aggiungere nuove stanze al sistema.
    - **Caricamento delle Stanze da File:** Le informazioni delle stanze devono poter essere importate da un file esterno.
    - **Salvataggio delle Stanze su File:** Le informazioni sulle stanze devono essere salvate su file per garantire la persistenza dei dati.
5. **Gestione dei Dipendenti**
    - **Aggiunta di Dipendenti:** Il manager deve poter aggiungere nuovi dipendenti, specificando nome, stipendio e skill.
    - **Visualizzazione dei Dettagli di un Dipendente:** Il manager deve poter consultare le informazioni sui dipendenti.
    - **Aggiornamento dei Dettagli di un Dipendente:** Il manager deve poter modificare le informazioni dei dipendenti esistenti.
    - **Rimozione di un Dipendente:** Il manager deve poter rimuovere dipendenti dal sistema.
    - **Tracciamento della Presenza dei Dipendenti:** Il manager deve poter registrare le presenze quotidiane dei dipendenti.
    - **Salvataggio dei Dipendenti e dei Record di Presenza:** Le informazioni sui dipendenti e sui record di presenza devono essere salvate su file per garantire la persistenza dei dati.
6. **Gestione dei Servizi Aggiuntivi**
    - **Aggiunta di un Servizio:** Il manager deve poter inserire nuovi servizi aggiuntivi offerti dall'hotel.
    - **Aggiornamento di un Servizio:** Il manager deve poter modificare le informazioni dei servizi esistenti.
    - **Rimozione di un Servizio:** Il manager deve poter eliminare servizi non più disponibili.
    - **Visualizzazione dei Dettagli di un Servizio:** Il manager deve poter consultare le informazioni sui servizi offerti.
    - **Salvataggio dei Servizi su File:** Le informazioni sui servizi devono essere salvate su file per garantire la persistenza dei dati.

1.2 Modello del dominio

Il modello del dominio, come descritto nei passaggi precedenti, è stato implementato dai membri del gruppo seguendo il diagramma UML fornito qui sotto. Questo schema UML rappresenta la struttura e le relazioni tra le varie entità del sistema di gestione dell'hotel, e serve come guida per l'implementazione delle funzionalità richieste.



Il diagramma UML rappresenta un'architettura basata sul pattern Model-View-Controller (MVC), in cui la MainView funge da interfaccia principale per l'avvio dell'applicazione. All'avvio, la MainView presenta una finestra con sei pulsanti, ciascuno associato a una specifica area funzionale gestita da un diverso controller (come ActivityController, BillingController, ecc.).

Quando l'utente seleziona un pulsante, la MainView nasconde la finestra principale e istanzia la vista corrispondente (ad esempio, ActivityView, BillingView, ecc.) insieme al controller associato. Ogni controller, a sua volta, interagisce con il rispettivo manager (come ActivityManager, BillingManager, ecc.), che rappresenta il modello per la gestione dei dati e della logica di business.

La MainView, pur essendo una vista, inizialmente svolge anche un ruolo di controller, dirigendo il flusso dell'applicazione verso le diverse componenti a seconda delle scelte dell'utente.

Il link di visualizzazione associato all’UML descritto precedentemente è il seguente:

https://app.genmymodel.com/api/projects/_Sa5X4E36Ee-f95DqthbXog/diagrams/_Sa5-8k36Ee-f95DqthbXog/svg

2.1 Architettura (Design)

L'architettura del gestionale è stata progettata con l'obiettivo di garantire una chiara separazione delle diverse componenti del sistema e facilitare la gestione delle interazioni tra di esse. Per raggiungere questi obiettivi, abbiamo scelto di adottare il pattern architetturale **Model-View-Controller (MVC)**.

Il pattern MVC suddivide l'applicazione in tre componenti principali: **Model**, **View**, e **Controller**. Questa suddivisione consente una gestione modulare dell'applicazione, permettendo di isolare le responsabilità di ciascun componente:

- **Model:** Gestisce i dati e la logica di business, indipendentemente dall'interfaccia utente.
- **View:** Si occupa della presentazione dei dati e dell'interfaccia grafica, rendendo possibile l'interazione con l'utente.
- **Controller:** Agisce come intermediario tra la View e il Model, interpretando gli input dell'utente e aggiornando sia il modello che la vista.

Questa separazione non solo facilita la manutenzione del codice, ma rende anche l'applicazione più scalabile e testabile, poiché ogni componente può essere sviluppato e aggiornato in modo indipendente, senza influenzare direttamente gli altri.

Model

Il **Model** rappresenta la componente del sistema che gestisce i dati e la logica di business dell'applicazione. È responsabile di mantenere lo stato dei dati, applicare le regole di business e gestire la logica per la manipolazione dei dati stessi. Il Model è indipendente dalla vista e dall'interfaccia utente, il che significa che può essere riutilizzato e testato separatamente.

Funzionalità principali:

- **Gestione dei Dati:** Il Model gestisce i dati dell'applicazione, che possono provenire da diverse fonti come database, API esterne o file locali. È il responsabile della creazione, lettura, aggiornamento e cancellazione (operazioni CRUD) dei dati.
- **Logica di Business:** Implementa la logica di business che definisce come i dati devono essere processati o manipolati. Questo include le regole e le condizioni che governano le operazioni sui dati.
- **Validazione:** Esegue la validazione dei dati per assicurarsi che essi siano corretti e completi prima di salvarli o utilizzarli ulteriormente.
- **Comunicazione con il Controller:** Il Model fornisce metodi che possono essere invocati dal Controller per recuperare o modificare i dati. Può anche

notificare il Controller (o direttamente la View, in alcuni casi) di eventuali cambiamenti nello stato dei dati.

View

La **View** è responsabile della presentazione dell'interfaccia utente. Mostra i dati gestiti dal Model all'utente e fornisce i mezzi per l'interazione con il sistema. La View è la componente che si occupa di come i dati vengono visualizzati, ma non contiene logica di business.

Funzionalità principali:

- **Visualizzazione dei Dati:** La View prende i dati dal Model (tramite il Controller) e li rende visibili all'utente. Questo può includere la visualizzazione di informazioni testuali, grafici, tabelle o altri tipi di output visivo.
- **Interfaccia Utente:** Gestisce gli elementi dell'interfaccia utente (UI) come pulsanti, moduli, menu, e finestre. Rende possibile l'interazione dell'utente con il sistema.
- **Aggiornamento Dinamico:** La View deve essere in grado di aggiornarsi in risposta ai cambiamenti nel Model. Ad esempio, se i dati nel Model cambiano, la View dovrebbe riflettere tali modifiche automaticamente.
- **Comunicazione con il Controller:** Sebbene la View sia responsabile della visualizzazione, è il Controller che interpreta gli input dell'utente. La View quindi passa gli eventi dell'interfaccia utente (come clic, inserimenti di dati, etc.) al Controller per l'elaborazione.

Controller

Il **Controller** agisce come intermediario tra il Model e la View. È responsabile di interpretare gli input dell'utente provenienti dalla View e di tradurli in azioni che manipolano il Model o aggiornano la View. In sostanza, il Controller coordina l'interazione tra le altre due componenti.

Funzionalità principali:

- **Gestione degli Input dell'Utente:** Il Controller riceve gli input dall'utente attraverso la View (come comandi, clic, o inserimenti di dati) e determina come questi debbano essere gestiti.
- **Invocazione delle Operazioni del Model:** In base agli input dell'utente, il Controller invoca le operazioni appropriate sul Model. Ad esempio, se un utente inserisce nuove informazioni, il Controller potrebbe chiedere al Model di aggiornare i dati.
- **Aggiornamento della View:** Dopo aver eseguito operazioni sul Model, il Controller può aggiornare la View per riflettere i cambiamenti nei dati o nel

sistema. Questo potrebbe includere l'aggiornamento di visualizzazioni, il refresh delle informazioni, o la navigazione verso nuove schermate.
- **Controllo del Flusso dell'Applicazione:** Il Controller può gestire il flusso complessivo dell'applicazione, determinando quale vista mostrare all'utente in risposta a specifiche azioni o condizioni.

3.1 Fase di testing: approccio adottato

Durante lo sviluppo del nostro programma gestionale, abbiamo adottato un approccio pragmatico alla fase di testing. Invece di implementare un sistema di testing automatizzato, come potrebbe essere realizzato con strumenti come JUnit o altri framework di test, abbiamo optato per un metodo più diretto: l'esecuzione manuale dell'applicazione per verificarne il funzionamento.

Esecuzione dell'Applicazione per il Testing

Il nostro approccio si è basato sull'idea che eseguendo direttamente l'applicazione, avremmo potuto osservare immediatamente come si comportava rispetto alle specifiche e alle aspettative che avevamo stabilito durante la fase di progettazione. L'applicazione è stata quindi avviata e testata in un ambiente il più vicino possibile a quello reale, simulando il comportamento degli utenti finali.

Questa metodologia ci ha permesso di individuare in maniera intuitiva e immediata le problematiche legate alla logica di business, all'interfaccia utente, e all'interazione tra i diversi componenti del sistema. Ogni volta che veniva riscontrato un problema “che fosse un errore nel flusso dell'applicazione, un bug nell'interfaccia grafica, o una mancata corrispondenza dei dati” intervenivamo direttamente nel codice per risolverlo. Successivamente, l'applicazione veniva rieseguita per verificare se la modifica avesse risolto il problema senza introdurre nuove anomalie.

Uno dei principali vantaggi di questo approccio è stato l'immediatezza con cui abbiamo potuto identificare e correggere i problemi. La possibilità di osservare l'applicazione in funzione ci ha fornito un feedback diretto sul suo comportamento, permettendoci di fare correzioni rapide e iterare velocemente durante il processo di sviluppo.

Per avviare il nostro gestionale, abbiamo creato una classe Main che funge da punto di ingresso dell'applicazione. All'interno di questa classe, abbiamo usato il metodo SwingUtilities.invokeLater per garantire che l'interfaccia utente venga avviata in modo sicuro sul thread dedicato alla gestione della GUI. Il metodo invokeLater esegue un blocco di codice che crea un'istanza della MainView, la finestra principale del nostro gestionale. Questa finestra viene mostrata immediatamente all'avvio dell'applicazione, fornendo all'utente un'interfaccia attraverso la quale può interagire

con tutte le funzionalità del sistema. La MainView, come descritto nei passi precedenti, è collegata a tutte le altre classi principali del modello MVC, facilitando l'interazione tra la vista, i controller e i modelli. Questo design assicura che l'interfaccia utente sia immediatamente accessibile e che il flusso dell'applicazione possa partire senza ritardi, rendendo l'esperienza utente fluida e coerente.

3.2 Metodologia del lavoro

Per la realizzazione di questo programma c'è stata una suddivisione del lavoro ben definita, abbiamo stabilito a monte i compiti che ogni membro del gruppo sarebbe dovuto andare a sviluppare.

3.2.1 Gambini Edoardo

Classi manager create e relative view e controller:

BillingManager

La classe BillingManager è il nucleo centrale del sistema di gestione della fatturazione, e si occupa delle principali funzionalità come la generazione di fatture e spese, l'elaborazione dei pagamenti e la creazione di rapporti finanziari annuali.

Attributi:

- invoices: Una lista di oggetti Invoice che rappresenta le fatture create.
- expenses: Una lista di oggetti Expense che rappresenta le spese registrate.
- INVOICE_FILE e EXPENSE_FILE: Costanti che rappresentano i nomi dei file in cui vengono salvati rispettivamente le fatture e le spese.

- Costruttore:
  - Il costruttore inizializza le liste invoices e expenses e carica i dati dai file di testo (invoices.txt e expenses.txt) tramite i metodi loadInvoicesFromFile() e loadExpensesFromFile(). Questo permette di mantenere la persistenza dei dati tra le sessioni del programma.
- Metodi Principali:
  - generateInvoice(int invoiceId, Customer customer, double amountDue, LocalDate date): Crea una nuova fattura e la aggiunge alla lista invoices, salvandola poi nel file.
  - getInvoices(), getExpenses(): Restituiscono rispettivamente le liste di fatture e spese.

        - generateExpense(int expenseId, double amount, LocalDate date, ExpenseCategory category): Crea una nuova spesa e la aggiunge alla lista expenses, salvandola poi nel file.
        - processPayment(int invoiceId, PaymentMethod method): Processa il pagamento di una fattura specificata, aggiornando il suo stato a "pagata" e salvando la modifica nel file.
        - trackOutstandingPayments(): Restituisce una lista di fatture non ancora pagate.
        - generateAnnualFinancialReport(int year): Genera un rapporto finanziario annuale, calcolando il totale delle entrate e delle uscite per l'anno specificato.
- Metodi Privati:
    - loadInvoicesFromFile(), loadExpensesFromFile(): Caricano le fatture e le spese dai file di testo.
    - saveInvoicesToFile(), saveExpensesToFile(): Salvano le fatture e le spese nei rispettivi file di testo.

Questa classe gestisce l'intero ciclo di vita delle fatture e delle spese, assicurandosi che i dati siano sempre aggiornati e persistenti.

BillingController

La classe BillingController funge da intermediario tra la logica di business gestita dalla BillingManager e l'interfaccia utente gestita dalla BillingView. Si occupa di ascoltare gli eventi dall'interfaccia utente e di tradurli in azioni appropriate.

- Attributi:
    - billingManager: Riferimento alla classe BillingManager per gestire la logica di business.
    - billingView: Riferimento alla classe BillingView per gestire l'interfaccia utente.
- Costruttore:
    - Il costruttore associa gli ascoltatori di eventi (listeners) ai vari pulsanti dell'interfaccia utente, collegando le azioni dell'utente alle operazioni di business. Inoltre, visualizza tutte le fatture e le spese già esistenti al momento dell'avvio dell'applicazione.
- Listener Interni:
    - GenerateInvoiceListener, GenerateExpenseListener: Questi listener si occupano della creazione di nuove fatture e spese rispettivamente, utilizzando i dati inseriti dall'utente nell'interfaccia.
    - ProcessPaymentListener: Gestisce l'elaborazione dei pagamenti per le fatture.
    - TrackPaymentsListener: Si occupa di tracciare e visualizzare le fatture non pagate.

    - GenerateReportListener: Gestisce la generazione e la visualizzazione di un rapporto finanziario annuale.
- Metodo Privato:
    - displayAllInvoicesAndExpenses(): Visualizza tutte le fatture e le spese nell'interfaccia utente.

Il BillingController coordina tutte le operazioni, garantendo che le azioni eseguite dall'utente siano correttamente tradotte in operazioni sul modello di business.

BillingView

La classe BillingView gestisce l'interfaccia grafica dell'applicazione, permettendo all'utente di interagire con il sistema di gestione della fatturazione.

- Componenti dell'Interfaccia:
    - Campi di testo (JTextField) per l'inserimento dei dati relativi alle fatture, spese, pagamenti e rapporti finanziari.
    - Pulsanti (JButton) per generare fatture, spese, processare pagamenti, tracciare pagamenti in sospeso e generare rapporti.
    - Aree di testo (JTextArea) per visualizzare i dettagli delle fatture, delle spese, dei pagamenti in sospeso e dei rapporti finanziari.
- Costruttore:
    - Configura la finestra principale dell'applicazione (JFrame), crea i vari pannelli (ognuno con il proprio colore e titolo) e dispone i componenti in un layout a griglia (GridLayout). Alla fine, rende la finestra visibile.
- Metodi Getter:
    - Forniscono accesso ai dati inseriti dall'utente nei vari campi di testo, necessari per la creazione di fatture, spese, ecc.
- Metodi di Aggiornamento:
    - updateInvoiceAndExpenseDetails(String details), updateOutstandingPayments(List<String> outstandingPayments), updateFinancialReport(String report): Aggiornano il contenuto delle aree di testo per visualizzare i dettagli delle operazioni.
- Metodo di Visualizzazione dei Messaggi:
    - showMessage(String message): Mostra un messaggio pop-up all'utente, ad esempio per confermare il successo di un'operazione o segnalare un errore.

Questa classe fornisce l'interfaccia attraverso la quale l'utente può interagire con l'applicazione, mentre gli altri componenti gestiscono la logica di business e l'interazione con i dati.

CustomerManager

La classe CustomerManager gestisce tutte le operazioni relative ai clienti all'interno del sistema, come la creazione, l'aggiornamento, l'eliminazione e la gestione dei reclami dei clienti. Questa classe si occupa anche della persistenza dei dati, salvando e caricando le informazioni sui clienti da un file di testo.

Attributi:

- customers: Una mappa che associa gli ID dei clienti (Integer) agli oggetti Customer. Questa struttura permette un accesso rapido ai dati dei clienti.
- nextCustomerId: Un contatore incrementale che fornisce un ID univoco a ogni nuovo cliente.
- CUSTOMER_FILE: Una costante che rappresenta il nome del file di testo in cui vengono salvate le informazioni sui clienti.

Costruttore:

- Il costruttore carica i dati dei clienti dal file di testo CUSTOMER_FILE all'avvio della classe, permettendo di mantenere la persistenza dei dati tra le sessioni.

Metodi Principali:

- createCustomer(String name, String email, String phone, LoyaltyStatus loyaltyStatus): Crea un nuovo cliente e lo aggiunge alla mappa customers. Salva i dettagli del nuovo cliente nel file di testo.
- getCustomerDetails(int customerId): Restituisce i dettagli di un cliente dato il suo ID.
- updateCustomerDetails(int customerId, String name, String email, String phone, LoyaltyStatus loyaltyStatus): Aggiorna i dettagli di un cliente esistente e salva tutte le informazioni aggiornate nel file di testo.
- deleteCustomer(int customerId): Rimuove un cliente dalla mappa customers e aggiorna il file di testo per riflettere la modifica.
- registerComplaint(int customerId, String complaintDetails): Registra un reclamo per un cliente specifico

Metodi Privati:

- saveCustomerToFile(Customer customer): Salva i dettagli di un singolo cliente nel file di testo.
- saveAllCustomersToFile(): Salva tutti i clienti nel file di testo, sovrascrivendo i dati esistenti.
- loadCustomersFromFile(): Carica i dettagli dei clienti dal file di testo CUSTOMER_FILE e li aggiunge alla mappa customers.

CustomerController

La classe CustomerController funge da intermediario tra la logica di business gestita dalla classe CustomerManager e l'interfaccia utente gestita da CustomerView. Si occupa di gestire gli eventi generati dall'utente e di tradurli in azioni appropriate sul modello di business.

Attributi:

- customerManager: Riferimento alla classe CustomerManager, che gestisce la logica di business relativa ai clienti.
- customerView: Riferimento alla classe CustomerView, che gestisce l'interfaccia utente.

Costruttore:

- Il costruttore associa vari listener di eventi ai componenti dell'interfaccia utente di CustomerView, come pulsanti e campi di testo, per gestire le azioni dell'utente.

Listener Interni:

- CreateCustomerListener: Ascolta l'evento di creazione di un nuovo cliente. Raccoglie i dati inseriti dall'utente e invia una richiesta di creazione al CustomerManager.
- GetCustomerDetailsListener: Ascolta la richiesta di ottenere i dettagli di un cliente. Cerca i dettagli del cliente specificato e li visualizza nell'interfaccia.
- UpdateCustomerDetailsListener: Gestisce l'aggiornamento dei dettagli di un cliente. Prende i nuovi dati inseriti dall'utente e aggiorna le informazioni del cliente.
- DeleteCustomerListener: Gestisce l'eliminazione di un cliente. Rimuove il cliente dal sistema e aggiorna l'interfaccia.
- RegisterComplaintListener: Ascolta e gestisce la registrazione di un reclamo per un cliente. Invia i dettagli del reclamo al CustomerManager.

Questa classe garantisce che le azioni dell'utente siano correttamente eseguite nel modello di business, mantenendo l'interfaccia utente aggiornata.

CustomerView

La classe CustomerView gestisce l'interfaccia grafica dell'applicazione, permettendo al manager di interagire con il sistema di gestione dei clienti. Si occupa di raccogliere i dati inseriti dall'utente e di visualizzare le informazioni restituite dal sistema.

Componenti dell'Interfaccia:

- Campi di testo (JTextField): Utilizzati per inserire i dettagli dei clienti, come nome, email, telefono e ID del cliente.
- ComboBox (JComboBox): Permette di selezionare il LoyaltyStatus del cliente.
- Pulsanti (JButton): Usati per eseguire operazioni come creare un nuovo cliente, ottenere i dettagli di un cliente, aggiornare le informazioni e registrare un reclamo.
- Area di testo (JTextArea): Visualizza i dettagli dei clienti e le risposte alle azioni eseguite dall'utente.

Costruttore:

- Configura la finestra principale dell'applicazione, organizza i vari pannelli che contengono i componenti dell'interfaccia e imposta il layout generale. Ogni pannello è associato a un colore e un titolo specifico per migliorare la chiarezza visiva. Infine, rende visibile la finestra dell'applicazione.

Metodi Getter:

- Forniscono accesso ai dati inseriti dall'utente nei campi di testo e nelle combo box, necessari per creare o aggiornare le informazioni sui clienti.

Metodi di Aggiornamento:

- setCustomerDetails(String details): Aggiorna l'area di testo con i dettagli del cliente o le risposte del sistema.

ReservationManager

La classe ReservationManager è il nucleo del sistema di gestione delle prenotazioni di un hotel. Si occupa delle principali funzionalità come la creazione, modifica e cancellazione di prenotazioni, la gestione delle camere, e la persistenza dei dati relativi alle prenotazioni e alle camere.

Attributi:

- reservations: Una lista di oggetti Reservation che rappresenta le prenotazioni registrate.
- rooms: Una lista di oggetti Room che rappresenta le camere disponibili nell'hotel.
- nextReservationId: Un intero che tiene traccia dell'ID della prossima prenotazione da assegnare, garantendo che ogni prenotazione abbia un ID univoco.

Costruttore:

- Il costruttore inizializza le liste reservations e rooms, e carica i dati relativi alle camere e alle prenotazioni dai file di testo tramite i metodi loadRoomsFromFile() e loadReservationsFromFile(). Questo garantisce la persistenza dei dati tra le sessioni del programma.

Metodi Principali:

- createReservation(int customerId, int roomId, LocalDate checkInDate, LocalDate checkOutDate): Crea una nuova prenotazione se la camera è disponibile per le date richieste, assegnando un ID univoco. Aggiunge la prenotazione alla lista reservations e la salva nel file.
- getReservationDetails(int reservationId): Restituisce i dettagli di una prenotazione specifica cercandola per ID. Se l'ID non esiste, lancia un'eccezione.
- cancelReservation(int reservationId): Cancella una prenotazione specificata per ID e aggiorna il file per riflettere la modifica.
- modifyReservation(int reservationId, LocalDate newCheckInDate, LocalDate newCheckOutDate): Modifica le date di check-in e check-out di una prenotazione esistente e salva le modifiche nel file.
- listAvailableRooms(LocalDate startDate, LocalDate endDate, RoomType roomType): Restituisce una lista di camere disponibili per un determinato intervallo di date e tipo di camera.
- createRoom(int roomNumber, RoomType roomType): Crea una nuova camera e la aggiunge alla lista rooms, salvandola nel file.

Metodi Privati:

- loadRoomsFromFile(String filename): Carica le informazioni delle camere da un file di testo. Ogni riga del file rappresenta una camera e contiene il numero della camera e il tipo.
- loadReservationsFromFile(String filename): Carica le prenotazioni da un file di testo. Ogni riga del file rappresenta una prenotazione e contiene l'ID della prenotazione, l'ID del cliente, l'ID della camera, la data di check-in e la data di check-out.
- saveRoomToFile(Room room, String filename): Salva una camera specifica in un file di testo, aggiungendola al file esistente.
- saveReservationToFile(Reservation reservation, String filename): Salva una prenotazione specifica in un file di testo, aggiungendola al file esistente.
- saveAllReservationsToFile(String filename): Salva tutte le prenotazioni nel file di testo, sovrascrivendo il file esistente con i dati aggiornati.
- isRoomAvailable(int roomNumber, LocalDate startDate, LocalDate endDate): Verifica se una camera è disponibile per un determinato intervallo di date, confrontando le date con le prenotazioni esistenti.

ReservationController

La classe ReservationController funge da intermediario tra il modello (ReservationManager) e la vista (ReservationView). Coordina le azioni dell'utente con le funzionalità del sistema di gestione delle prenotazioni.

Attributi:

- reservationManager: Un'istanza di ReservationManager che gestisce la logica delle prenotazioni.
- reservationView: Un'istanza di ReservationView che gestisce l'interfaccia utente.

Costruttore:

- Il costruttore collega il ReservationManager e la ReservationView, e registra i listener per gestire le azioni dell'utente (es. creare, modificare, cancellare prenotazioni).

Classi Interne Principali:

- CreateRoomListener: Gestisce l'azione di creazione di una camera. Raccoglie i dati dalla vista e li passa al modello per creare la camera, aggiornando poi la vista con il risultato.
- CreateReservationListener: Gestisce l'azione di creazione di una prenotazione. Recupera i dettagli della prenotazione dalla vista e la crea tramite il modello, mostrando poi i dettagli della prenotazione creata nella vista.
- GetReservationDetailsListener: Gestisce la richiesta di visualizzare i dettagli di una prenotazione. Recupera l'ID della prenotazione dalla vista, ottiene i dettagli dal modello e li mostra nella vista.
- CancelReservationListener: Gestisce la cancellazione di una prenotazione. Prende l'ID della prenotazione dalla vista e lo passa al modello per cancellarla, aggiornando poi la vista con un messaggio di conferma.
- ModifyReservationListener: Gestisce la modifica di una prenotazione. Recupera i nuovi dettagli della prenotazione dalla vista e li passa al modello per modificarla, aggiornando la vista con il risultato.
- ListAvailableRoomsListener: Gestisce la richiesta di elencare le camere disponibili. Recupera le date e il tipo di camera dalla vista, ottiene le camere disponibili dal modello e le mostra nella vista.

ReservationView

La classe ReservationView è responsabile della gestione dell'interfaccia utente, utilizzando componenti Swing per permettere agli utenti di interagire con il sistema di prenotazione.

Componenti Principali:

- Campi di testo e caselle combinate (JTextField, JComboBox): Utilizzati per raccogliere i dati necessari per creare, modificare e visualizzare prenotazioni e camere.
- Pulsanti (JButton): Attivano le diverse azioni (es. creare una prenotazione, cancellare una prenotazione, etc.).
- Aree di testo (JTextArea): Visualizzano i risultati delle azioni (es. dettagli delle prenotazioni, camere disponibili, etc.).

Costruttore:

- Il costruttore crea e organizza l'interfaccia utente, suddividendo le funzionalità in pannelli separati per una migliore usabilità. Ogni pannello è personalizzato con un titolo e un colore di sfondo distintivo per migliorare l'aspetto visivo e la navigabilità dell'applicazione.

Metodi Principali:

- Getter: Metodi per ottenere i valori inseriti dall'utente nei campi di testo e nelle caselle combinate. Esempi includono getRoomNumber(), getCustomerId(), getCheckInDate(), etc.
- Setter: Metodi per aggiornare le aree di testo con i risultati delle operazioni, come setRoomCreationStatus(), setReservationDetails(), etc.
- Metodi per aggiungere listener: Metodi come addCreateRoomListener(), addCreateReservationListener(), etc., che consentono al controller di collegare le azioni dell'utente con le rispettive operazioni nel modello.

3.2.2 Fraternali Matteo

Classi manager create e relative view e controller:

ActivityManager

La classe ActivityManager è il cuore della gestione delle attività all'interno dell'hotel. Si occupa delle operazioni principali, come la creazione, l'aggiornamento, l'eliminazione e il recupero dei dettagli delle attività, oltre a gestire la persistenza dei dati tramite un file di testo.

Attributi:

- activities: Una lista di oggetti Activity che rappresenta tutte le attività disponibili nell'hotel.
- FILE_NAME: Una costante che rappresenta il nome del file di testo (activities.txt) in cui sono salvate le informazioni sulle attività. Questo file viene utilizzato per la persistenza dei dati tra le sessioni.

Costruttore: Il costruttore della classe ActivityManager inizializza la lista activities e carica i dati delle attività dal file di testo tramite il metodo privato loadActivitiesFromFile(). Questo permette di mantenere le informazioni sulle attività salvate tra una sessione e l'altra dell'applicazione.

Metodi Principali:

- createActivity(String name, double cost, String description): Crea una nuova attività con i parametri specificati (nome, costo, descrizione) e la aggiunge alla lista activities. La nuova attività viene poi salvata nel file per garantire la persistenza dei dati.
- getActivityDetails(int activityId): Restituisce i dettagli dell'attività corrispondente all'ID fornito. Se l'attività non viene trovata, restituisce null.
- updateActivityDetails(int activityId, String name, double cost, String description): Aggiorna i dettagli di un'attività esistente identificata dall'ID. Dopo l'aggiornamento, i nuovi dettagli vengono salvati nel file.
- cancelActivity(int activityId): Elimina l'attività corrispondente all'ID fornito dalla lista activities e aggiorna il file per riflettere questa eliminazione.

Metodi Privati:

- loadActivitiesFromFile(): Carica le attività dal file di testo activities.txt e le aggiunge alla lista activities. Gestisce eventuali eccezioni che potrebbero verificarsi durante la lettura del file.

- saveActivitiesToFile(): Salva tutte le attività attualmente nella lista activities nel file di testo activities.txt. Questo metodo assicura che ogni modifica fatta alle attività (creazione, aggiornamento, eliminazione) venga salvata nel file.

ActivityController

La classe ActivityController funge da intermediario tra la logica di gestione delle attività (gestita dalla classe ActivityManager) e l'interfaccia utente (gestita dalla classe ActivityView). Si occupa di ascoltare gli eventi generati dall'interfaccia utente e di tradurli in azioni sul modello di business.

Attributi:

- activityManager: Un riferimento alla classe ActivityManager, utilizzato per eseguire le operazioni di gestione delle attività.
- activityView: Un riferimento alla classe ActivityView, utilizzato per gestire l'interfaccia utente.

Costruttore: Il costruttore associa diversi listener agli eventi dei pulsanti dell'interfaccia utente. Questi listener sono responsabili dell'interazione con l'utente e della traduzione delle azioni utente in operazioni di gestione delle attività.

Listener Interni:

- CreateActivityListener: Ascolta l'evento di creazione di una nuova attività. Quando viene premuto il pulsante di creazione, questo listener recupera i dati inseriti dall'utente, crea una nuova attività tramite ActivityManager, e aggiorna l'interfaccia utente con i dettagli dell'attività appena creata.
- GetActivityDetailsListener: Ascolta l'evento di richiesta dei dettagli di un'attività. Recupera l'ID dell'attività inserito dall'utente, cerca i dettagli corrispondenti tramite ActivityManager, e li visualizza nell'interfaccia utente.
- UpdateActivityListener: Ascolta l'evento di aggiornamento dei dettagli di un'attività. Recupera i nuovi dati inseriti dall'utente, aggiorna l'attività tramite ActivityManager, e notifica l'utente dell'aggiornamento avvenuto.
- DeleteActivityListener: Ascolta l'evento di eliminazione di un'attività. Recupera l'ID dell'attività da eliminare, la rimuove tramite ActivityManager, e aggiorna l'interfaccia utente per riflettere l'eliminazione.

ActivityView

La classe ActivityView gestisce l'interfaccia grafica dell'applicazione, permettendo all'utente di interagire con il sistema di gestione delle attività dell'hotel. Questa classe utilizza la libreria Swing di Java per creare un'interfaccia grafica semplice e intuitiva.

Componenti dell'Interfaccia:

- Campi di testo (JTextField): Utilizzati per l'inserimento dei dati relativi alle attività, come il nome, il costo e l'ID dell'attività.
- Aree di testo (JTextArea): Utilizzate per visualizzare i dettagli delle attività, inclusa la descrizione e i risultati delle operazioni come la creazione o l'aggiornamento delle attività.
- Pulsanti (JButton): Pulsanti per eseguire le varie operazioni come la creazione, l'aggiornamento, la cancellazione e la visualizzazione dei dettagli delle attività.

Costruttore: Il costruttore configura la finestra principale dell'applicazione (JFrame), crea i pannelli per ogni operazione (ognuno con un colore e un titolo specifico), e dispone i componenti utilizzando un layout a griglia (GridLayout). Alla fine, la finestra viene resa visibile.

Metodi Getter:

- getName(), getCost(), getDescription(): Recuperano i dati inseriti dall'utente nei campi di testo per la creazione di una nuova attività.
- getActivityId(), getUpdateActivityId(), getDeleteActivityId(): Recuperano l'ID dell'attività inserito dall'utente per le operazioni di recupero, aggiornamento o cancellazione di un'attività.

Metodi di Aggiornamento:

- setActivityDetails(String details): Aggiorna il contenuto dell'area di testo che visualizza i dettagli delle attività, dopo operazioni come la creazione, l'aggiornamento o la cancellazione.

Metodo di Visualizzazione dei Messaggi:

- showMessage(String message): Mostra un messaggio pop-up all'utente, ad esempio per confermare il successo di un'operazione o segnalare un errore.

EmployeeManager

La classe EmployeeManager è il fulcro della gestione del personale nell'hotel, occupandosi della creazione, modifica e monitoraggio dei dipendenti e della loro presenza.

Attributi:

- employees: Una lista di oggetti Employee che rappresenta tutti i dipendenti attualmente registrati.

- attendanceRecords: Una mappa che associa ogni dipendente (identificato tramite il loro ID) con i loro record di presenza, specificando per ogni data se il dipendente era presente o meno.
- EMPLOYEE_FILE e ATTENDANCE_FILE: Costanti che definiscono i nomi dei file utilizzati per salvare rispettivamente i dati dei dipendenti e i record di presenza.

Costruttore: Il costruttore inizializza la lista employees e la mappa attendanceRecords, e carica i dati da file di testo (employees.txt e attendance.txt) tramite i metodi privati loadEmployeesFromFile() e loadAttendanceFromFile(). Questo garantisce che i dati siano persistenti tra le diverse sessioni del programma.

Metodi Principali:

- addEmployee(String name, double salary, Skill skill): Crea un nuovo dipendente con un ID univoco, lo aggiunge alla lista employees e salva l'aggiornamento nel file.
- getEmployeeDetails(int employeeId): Restituisce i dettagli di un dipendente specifico, ricercandolo tramite l'ID.
- updateEmployeeDetails(int employeeId, String name, Skill skill, double salary): Aggiorna i dati di un dipendente esistente e salva le modifiche nel file.
- removeEmployee(int employeeId): Rimuove un dipendente dalla lista e cancella i suoi record di presenza, aggiornando i file.
- trackEmployeeAttendance(int employeeId, LocalDate date, boolean isPresent): Registra la presenza di un dipendente in una data specifica e salva il record. Fornisce un feedback sulla presenza o assenza del dipendente tramite un messaggio.

Metodi Privati:

- saveEmployeesToFile(): Salva tutti i dipendenti attualmente registrati nel file employees.txt.
- saveAttendanceToFile(): Salva tutti i record di presenza nel file attendance.txt.
- loadEmployeesFromFile(): Carica i dati dei dipendenti dal file employees.txt all'inizializzazione della classe.
- loadAttendanceFromFile(): Carica i record di presenza dal file attendance.txt all'inizializzazione della classe.

Questa classe gestisce l'intero ciclo di vita dei dipendenti, dall'aggiunta alla rimozione, assicurando che i dati siano sempre aggiornati e correttamente registrati nei file di testo.

EmployeeController

La classe EmployeeController agisce come intermediario tra la logica di business gestita da EmployeeManager e l'interfaccia utente gestita da EmployeeView. Si occupa di ascoltare gli eventi generati dall'interfaccia utente e tradurli in azioni sul modello di business.

Attributi:

- employeeManager: Riferimento alla classe EmployeeManager, che gestisce la logica di business legata ai dipendenti.
- employeeView: Riferimento alla classe EmployeeView, che gestisce l'interfaccia utente.

Costruttore: Il costruttore associa diversi ascoltatori di eventi (listeners) ai pulsanti dell'interfaccia utente, garantendo che le azioni dell'utente siano correttamente tradotte in operazioni sul modello di business.

Listener Interni:

- AddEmployeeListener: Gestisce l'aggiunta di un nuovo dipendente, utilizzando i dati inseriti dall'utente e visualizzando un messaggio di conferma o di errore.
- GetEmployeeDetailsListener: Recupera e visualizza i dettagli di un dipendente specifico quando richiesto dall'utente.
- UpdateEmployeeDetailsListener: Aggiorna i dati di un dipendente esistente e fornisce un feedback all'utente.
- RemoveEmployeeListener: Rimuove un dipendente specificato dall'utente, gestendo anche la cancellazione dei relativi record di presenza.
- TrackAttendanceListener: Registra la presenza o l'assenza di un dipendente in una data specifica, fornendo un feedback immediato.

Il EmployeeController coordina tutte le operazioni tra l'interfaccia utente e la logica di business, assicurando che le azioni dell'utente siano correttamente implementate.

EmployeeView

La classe EmployeeView è responsabile dell'interfaccia grafica dell'applicazione, fornendo un'interfaccia attraverso la quale gli utenti possono interagire con il sistema di gestione del personale.

Attributi:

- nameField, salaryField, employeeIdField, ecc.: Campi di testo e altri componenti dell'interfaccia utente che raccolgono l'input dell'utente per operazioni come l'aggiunta, la modifica o la rimozione di dipendenti.
- addEmployeeButton, getEmployeeDetailsButton, updateEmployeeDetailsButton, ecc.: Pulsanti che attivano le diverse azioni come aggiungere, ottenere dettagli, aggiornare o rimuovere un dipendente.
- employeeDetailsArea: Un'area di testo utilizzata per visualizzare i dettagli di un dipendente specifico.

Costruttore: Il costruttore configura l'interfaccia utente, organizzando i vari componenti in pannelli distinti, ciascuno con uno scopo specifico (aggiungere dipendenti, ottenere dettagli, aggiornare, rimuovere e tracciare la presenza). Imposta anche i colori e l'aspetto generale dell'interfaccia.

Metodi Principali:

- Getter per ottenere l'input dell'utente dai campi di testo e altri componenti.
- addAddEmployeeListener(ActionListener listener), addGetEmployeeDetailsListener(ActionListener listener), ecc.: Metodi per associare i listener ai rispettivi pulsanti dell'interfaccia.

ServiceManager

La classe ServiceManager è il cuore del sistema di gestione dei servizi offerti da un hotel. Si occupa di gestire l'elenco dei servizi, permettendo di aggiungerli, modificarli, rimuoverli, e visualizzarne i dettagli.

Attributi:

- services: Una lista di oggetti Service che rappresenta tutti i servizi disponibili in hotel.
- SERVICE_FILE: Costante che rappresenta il nome del file in cui vengono salvati i dati relativi ai servizi.

Costruttore:

- Il costruttore inizializza la lista services e carica i dati dal file di testo services.txt tramite il metodo loadServicesFromFile(). Questo permette di mantenere la persistenza dei dati dei servizi tra le sessioni dell'applicazione.

Metodi Principali:

- addService(String name, double price): Crea un nuovo servizio con il nome e il prezzo specificati, lo aggiunge alla lista services, e salva la lista aggiornata nel file.
- updateService(int serviceId, String name, double price): Modifica il nome e il prezzo di un servizio esistente identificato da serviceId, e salva le modifiche nel file.
- removeService(int serviceId): Rimuove un servizio dalla lista basato sull'ID del servizio (serviceId), aggiornando anche il file.
- getServiceDetails(int serviceId): Restituisce i dettagli di un servizio specifico, identificato dal suo ID.
- setServiceAvailability(int serviceId, boolean isAvailable): Imposta la disponibilità di un servizio specifico (attivo/inattivo) e salva la modifica nel file.

Metodi Privati:

- saveServicesToFile(): Salva lo stato attuale della lista services nel file services.txt, garantendo la persistenza delle modifiche.
- loadServicesFromFile(): Carica i dati dei servizi dal file services.txt, ricostruendo la lista services.

ServiceController

La classe ServiceController funge da intermediario tra la logica di business gestita dalla ServiceManager e l'interfaccia utente gestita dalla ServiceView. Gestisce gli eventi dell'interfaccia utente e li traduce in operazioni sul modello di business.

Attributi:

- serviceManager: Riferimento alla classe ServiceManager per gestire le operazioni sui servizi.
- serviceView: Riferimento alla classe ServiceView per gestire l'interfaccia utente.

Costruttore:

- Il costruttore collega la vista e il manager, registrando gli ascoltatori di eventi per i pulsanti nell'interfaccia utente. Gli eventi come l'aggiunta, la modifica, la rimozione e la visualizzazione dei dettagli di un servizio sono gestiti da listener specifici.

Listener Interni:

- AddServiceListener: Gestisce l'aggiunta di un nuovo servizio, prendendo i dati dall'interfaccia utente e passandoli a ServiceManager per creare il servizio.
- UpdateServiceListener: Gestisce l'aggiornamento dei dettagli di un servizio esistente.
- RemoveServiceListener: Gestisce la rimozione di un servizio.
- GetServiceDetailsListener: Recupera e visualizza i dettagli di un servizio specifico.

ServiceView

La classe ServiceView gestisce l'interfaccia grafica dell'applicazione, permettendo all'utente di interagire con il sistema di gestione dei servizi.

Componenti dell'Interfaccia:

- Campi di testo (JTextField): Utilizzati per inserire dati relativi ai servizi, come nome, prezzo e ID del servizio.
- Pulsanti (JButton): Permettono di eseguire azioni come aggiungere, aggiornare, rimuovere un servizio o visualizzarne i dettagli.
- Area di testo (JTextArea): Visualizza i dettagli di un servizio specifico.

Costruttore:

- Il costruttore configura la finestra principale dell'applicazione, creando pannelli distinti per le diverse operazioni (aggiunta, aggiornamento, rimozione, visualizzazione dei dettagli) e assegnando loro colori distinti per migliorare l'usabilità. Ogni pannello contiene etichette e campi di testo per l'inserimento dei dati, oltre ai pulsanti per eseguire le operazioni.

Metodi Getter:

- getServiceName(), getServicePrice(): Restituiscono il nome e il prezzo di un servizio inseriti dall'utente.
- getUpdateServiceId(), getUpdateName(), getUpdatePrice(): Restituiscono l'ID del servizio, il nuovo nome e il nuovo prezzo per un aggiornamento.
- getRemoveServiceId(): Restituisce l'ID del servizio da rimuovere.
- getServiceId(): Restituisce l'ID del servizio di cui si vogliono visualizzare i dettagli.

Metodi di Aggiornamento:

- setServiceDetails(String details): Aggiorna l'area di testo con i dettagli di un servizio specifico.

3.2.3 Classi necessarie per il funzionamento dei manager

Activity

La classe Activity rappresenta un'attività offerta dall'hotel, come ad esempio una lezione di yoga, un'escursione o un corso di cucina. Questa classe contiene vari attributi e metodi per gestire le informazioni e i partecipanti all'attività.

- **Attributi:**
  - idCounter: Un contatore statico utilizzato per assegnare automaticamente un ID univoco a ogni attività creata.
  - id: Un identificatore univoco per l'attività.
  - name: Il nome dell'attività.
  - cost: Il costo dell'attività.
  - description: Una descrizione dell'attività.
  - participants: Una lista di oggetti Customer che rappresentano i clienti che partecipano all'attività.
- **Metodi:**
  - Activity(String name, double cost, String description): Costruttore che inizializza l'attività con un nome, un costo e una descrizione, assegnando un ID univoco.
  - getId(), getName(), getCost(), getDescription(), getParticipants(): Metodi getter per accedere agli attributi della classe.
  - setName(String name), setCost(double cost), setDescription(String description): Metodi setter per modificare gli attributi dell'attività.
  - addParticipant(Customer customer): Aggiunge un cliente alla lista dei partecipanti.
  - toString(): Restituisce una rappresentazione in formato stringa dell'attività, inclusi i dettagli e la lista dei partecipanti.

Customer

La classe Customer rappresenta un cliente dell'hotel. Questa classe eredita dalla classe Person e aggiunge attributi specifici del cliente.

- **Attributi:**
    - id: Un identificatore univoco per il cliente.
    - loyaltyStatus: Lo stato di fedeltà del cliente, rappresentato da un'enumerazione LoyaltyStatus.
- **Metodi:**
    - **Costruttori:**
        - Customer(String name, String email, String phone, LoyaltyStatus loyaltyStatus): Inizializza un cliente con nome, email, telefono e stato di fedeltà.
        - Customer(int id, String name, String email, String phone, LoyaltyStatus loyaltyStatus): Inizializza un cliente con ID, nome, email, telefono e stato di fedeltà.
        - Customer(int id, String name, String email, String phone): Inizializza un cliente con ID, nome, email e telefono, senza specificare lo stato di fedeltà.
        - Customer(String name): Inizializza un cliente solo con il nome.
    - getId(), getLoyaltyStatus(): Metodi getter per ottenere l'ID e lo stato di fedeltà del cliente.
    - setLoyaltyStatus(LoyaltyStatus status): Metodo setter per impostare lo stato di fedeltà del cliente.
    - toString(): Restituisce una stringa che rappresenta il cliente con i suoi attributi.
    - fromString(String str): Metodo statico che crea un oggetto Customer da una stringa formattata.

Employee

La classe Employee rappresenta un dipendente dell'hotel e, come Customer, eredita dalla classe Person.

- **Attributi:**
    - id: Un identificatore univoco per il dipendente.
    - salary: Il salario del dipendente.
    - skill: La competenza o specializzazione del dipendente, rappresentata da un oggetto Skill.
- **Metodi:**
    - **Costruttore:**
        - Employee(int id, String name, double salary, Skill skill): Inizializza un dipendente con ID, nome, salario e competenza.
    - getId(), getSalary(), getSkill(): Metodi getter per accedere agli attributi del dipendente.
    - setSalary(double salary), setSkill(Skill skill): Metodi setter per modificare il salario e la competenza del dipendente.

    - toString(): Restituisce una stringa che rappresenta il dipendente con i suoi attributi.

Expense

La classe Expense rappresenta una spesa sostenuta dall'hotel. Questa classe gestisce i dettagli relativi a una particolare spesa.

- **Attributi:**
    - expenseId: Un identificatore univoco per la spesa.
    - amount: L'importo della spesa.
    - date: La data in cui è stata effettuata la spesa.
    - category: La categoria della spesa, rappresentata da un oggetto ExpenseCategory.
- **Metodi:**
    - **Costruttore:**
        - Expense(int expenseId, double amount, LocalDate date, ExpenseCategory category): Inizializza una spesa con ID, importo, data e categoria.
    - getExpenseId(), getAmount(), getDate(), getCategory(): Metodi getter per ottenere i dettagli della spesa.
    - setCategory(ExpenseCategory category): Metodo setter per modificare la categoria della spesa.
    - getYearExpense(): Restituisce l'anno in cui la spesa è stata sostenuta.
    - toString(): Restituisce una stringa che rappresenta la spesa con i suoi attributi.

ExpenseCategory

L'enumerazione ExpenseCategory definisce le diverse categorie di spesa che un hotel può sostenere. Queste categorie aiutano a classificare e organizzare le spese per una migliore gestione finanziaria.

- **Valori dell'enumerazione:**
    - MAINTENANCE: Rappresenta le spese per la manutenzione.
    - UTILITIES: Include le spese per le utenze come acqua, elettricità, ecc.
    - SALARIES: Spese relative agli stipendi del personale.
    - MARKETING: Spese per attività di marketing e promozione.
    - SUPPLIES: Costi per forniture generali.
    - FOOD_AND_BEVERAGE: Spese per cibo e bevande.
    - FURNITURE_AND_EQUIPMENT: Costi per mobili e attrezzature.
    - LEGAL_FEES: Spese legali.
    - TAXES: Tasse e imposte.
    - OTHER: Altre spese non categorizzate.

- **Attributo:**
    - displayName: Una stringa che contiene il nome visualizzato per ogni categoria, utile per una rappresentazione leggibile.
- **Metodi:**
    - getDisplayName(): Restituisce il nome visualizzato della categoria.
    - toString(): Restituisce il nome dell'enumerazione in formato maiuscolo.

FinancialReport

La classe FinancialReport rappresenta un rapporto finanziario annuale dell'hotel, che include il reddito totale, le spese totali e il profitto netto per l'anno specificato.

- **Attributi:**
    - year: L'anno a cui si riferisce il rapporto finanziario.
    - totalIncome: Il reddito totale dell'hotel per l'anno.
    - totalExpenses: Le spese totali sostenute dall'hotel nell'anno.

- **Metodi:**
    - **Costruttore:**
        - FinancialReport(int year, double totalIncome, double totalExpenses): Inizializza il rapporto finanziario con l'anno, il reddito totale e le spese totali.
    - getYear(): Restituisce l'anno del rapporto.
    - getTotalIncome(): Restituisce il reddito totale per l'anno.
    - getTotalExpenses(): Restituisce le spese totali per l'anno.
    - getNetProfit(): Calcola e restituisce il profitto netto, ovvero la differenza tra reddito totale e spese totali.
    - toString(): Restituisce una rappresentazione in formato stringa del rapporto finanziario, con dettagli su reddito, spese e profitto netto.

Invoice

La classe Invoice rappresenta una fattura emessa dall'hotel a un cliente per un importo dovuto. Include informazioni sul cliente, l'importo, la data e lo stato del pagamento.

- **Attributi:**
    - invoiceId: Un identificatore univoco per la fattura.
    - customer: L'oggetto Customer associato a questa fattura.
    - amountDue: L'importo dovuto dal cliente.
    - date: La data di emissione della fattura.
    - paid: Un booleano che indica se la fattura è stata pagata.

    - paymentMethod: Il metodo di pagamento utilizzato, rappresentato dall'enumerazione PaymentMethod.
- **Metodi:**
    - **Costruttore:**
        - Invoice(int invoiceId, Customer customer, double amountDue, LocalDate date): Inizializza una fattura con un ID, cliente, importo dovuto e data. Per default, la fattura non è pagata e non ha un metodo di pagamento associato.
    - getInvoiceId(): Restituisce l'ID della fattura.
    - getCustomer(): Restituisce il cliente associato alla fattura.
    - getAmountDue(): Restituisce l'importo dovuto.
    - getDate(): Restituisce la data di emissione della fattura.
    - isPaid(): Restituisce true se la fattura è stata pagata, false altrimenti.
    - setPaid(boolean paid): Imposta lo stato di pagamento della fattura.
    - getPaymentMethod(): Restituisce il metodo di pagamento utilizzato.
    - setPaymentMethod(PaymentMethod paymentMethod): Imposta il metodo di pagamento per la fattura.

LoyaltyStatus

L'enumerazione LoyaltyStatus rappresenta i diversi livelli di fedeltà di un cliente. Questi livelli possono essere utilizzati per applicare benefici o sconti ai clienti fedeli.

- **Valori dell'enumerazione:**
    - STANDARD: Livello base per tutti i clienti.
    - SILVER: Livello di fedeltà intermedio.
    - GOLD: Livello di fedeltà avanzato.
    - PLATINUM: Livello di fedeltà più alto.
- **Attributo:**
    - displayName: Una stringa che contiene il nome visualizzato per ogni livello di fedeltà.
- **Metodi:**
    - getDisplayName(): Restituisce il nome visualizzato del livello di fedeltà.
    - toString(): Restituisce il nome dell'enumerazione in formato maiuscolo.

PaymentMethod

L'enumerazione PaymentMethod definisce i vari metodi di pagamento che un cliente può utilizzare per saldare una fattura.

- **Valori dell'enumerazione:**
    - CREDIT_CARD: Carta di credito.
    - DEBIT_CARD: Carta di debito.
    - CASH: Contanti.
    - BANK_TRANSFER: Bonifico bancario.

    - PAYPAL: Pagamento tramite PayPal.
    - APPLE_PAY: Pagamento tramite Apple Pay.
    - GOOGLE_PAY: Pagamento tramite Google Pay.
    - OTHER: Altro metodo di pagamento.
- **Attributo:**
    - displayName: Una stringa che contiene il nome visualizzato per ogni metodo di pagamento.
- **Metodi:**
    - getDisplayName(): Restituisce il nome visualizzato del metodo di pagamento.
    - toString(): Restituisce il nome visualizzato.

Person

La classe Person è una classe base (o superclasse) che rappresenta una persona generica all'interno del sistema. Altre classi, come Customer ed Employee, ereditano da Person.

- **Attributi:**
    - name: Il nome della persona.
    - email: L'indirizzo email della persona.
    - phone: Il numero di telefono della persona.
- **Metodi:**
    - **Costruttori:**
        - Person(String name, String email, String phone): Inizializza una persona con nome, email e telefono.
        - Person(String name): Inizializza una persona solo con il nome.
    - getName(), getEmail(), getPhone(): Metodi getter per accedere agli attributi della persona.
    - setName(String name), setEmail(String email), setPhone(String phone): Metodi setter per modificare gli attributi della persona.

Reservation

La classe Reservation rappresenta una prenotazione effettuata da un cliente per una specifica camera in un determinato periodo.

- **Attributi:**
    - id: Identificatore univoco della prenotazione.
    - customerId: Identificatore del cliente che ha effettuato la prenotazione.
    - roomId: Identificatore della camera prenotata.
    - checkInDate: Data di arrivo (check-in) del cliente.
    - checkOutDate: Data di partenza (check-out) del cliente.

- **Metodi:**
  - **Costruttore:**
    - Reservation(int id, int customerId, int roomId, LocalDate checkInDate, LocalDate checkOutDate): Inizializza una prenotazione con i dettagli specificati.
  - getId(): Restituisce l'ID della prenotazione.
  - getCustomerId(): Restituisce l'ID del cliente.
  - getRoomId(): Restituisce l'ID della camera.
  - getCheckInDate(): Restituisce la data di check-in.
  - setCheckInDate(LocalDate checkInDate): Imposta la data di check-in.
  - getCheckOutDate(): Restituisce la data di check-out.
  - setCheckOutDate(LocalDate checkOutDate): Imposta la data di check-out.
  - toString(): Restituisce una rappresentazione in formato stringa della prenotazione, con dettagli su ID, cliente, camera e date di soggiorno.

Room

La classe Room rappresenta una camera dell'hotel. Ogni camera è associata a un tipo di camera specifico (RoomType) e ha un prezzo base.

- **Attributi:**
  - roomNumber: Numero della camera.
  - roomType: Tipo della camera, rappresentato dall'enumerazione RoomType.
  - basePrice: Prezzo base della camera, derivato dal tipo di camera.
- **Metodi:**
  - **Costruttore:**
    - Room(int roomNumber, RoomType roomType): Inizializza una camera con un numero e un tipo specificato.
  - getRoomNumber(): Restituisce il numero della camera.
  - getRoomType(): Restituisce il tipo di camera.
  - getBasePrice(): Restituisce il prezzo base della camera.
  - toString(): Restituisce una rappresentazione in formato stringa della camera, con dettagli su numero e tipo di camera.

RoomType

L'enumerazione RoomType definisce i diversi tipi di camere disponibili nell'hotel, ciascuno con un nome, un prezzo base e una descrizione.

- **Valori dell'enumerazione:**
  - SINGLE: Una camera per una persona con un letto singolo.
  - DOUBLE: Una camera per due persone con un letto matrimoniale.

    - SUITE: Una camera lussuosa con area soggiorno separata e servizi extra.
    - FAMILY: Una camera con più letti, adatta a famiglie o gruppi.
- **Attributi:**
    - name: Nome del tipo di camera.
    - basePrice: Prezzo base della camera.
    - description: Descrizione del tipo di camera.
- **Metodi:**
    - getName(): Restituisce il nome del tipo di camera.
    - getBasePrice(): Restituisce il prezzo base del tipo di camera.
    - getDescription(): Restituisce la descrizione del tipo di camera.
    - toString(): Restituisce una rappresentazione in formato stringa del tipo di camera, includendo il nome, la descrizione e il prezzo base.

Service

La classe Service rappresenta un servizio offerto dall'hotel, come ad esempio il servizio in camera, il Wi-Fi, o la lavanderia. Ogni servizio ha un nome, un prezzo, ed è disponibile o meno per i clienti.

- **Attributi:**
    - id: Identificatore univoco del servizio (generato automaticamente).
    - name: Nome del servizio.
    - price: Prezzo del servizio.
    - isAvailable: Indica se il servizio è attualmente disponibile.
- **Metodi:**
    - **Costruttore:**
        - Service(String name, double price, boolean isAvailable): Inizializza un servizio con un nome, un prezzo e una disponibilità.
    - getServiceId(): Restituisce l'ID del servizio.
    - getName(): Restituisce il nome del servizio.
    - getPrice(): Restituisce il prezzo del servizio.
    - isAvailable(): Restituisce true se il servizio è disponibile, false altrimenti.
    - setName(String name): Imposta il nome del servizio.
    - setPrice(double price): Imposta il prezzo del servizio.
    - setAvailable(boolean isAvailable): Imposta la disponibilità del servizio.
    - toString(): Restituisce una rappresentazione in formato stringa del servizio, con dettagli su ID, nome, prezzo e disponibilità.

Skill

L'enumerazione Skill rappresenta le diverse competenze che il personale dell'hotel può possedere. Queste competenze sono utilizzate per assegnare i compiti appropriati al personale.

- **Valori dell'enumerazione:**
  - HOUSEKEEPING: Competenze nelle pulizie e nella gestione delle camere.
  - FRONT_DESK: Competenze nelle operazioni di reception e gestione clienti.
  - MAINTENANCE: Competenze nella manutenzione generale dell'hotel.
  - KITCHEN: Competenze nella cucina e gestione dei servizi alimentari.
  - MANAGEMENT: Competenze nella gestione e amministrazione dell'hotel.
  - CUSTOMER_SERVICE: Competenze nel servizio clienti.
  - IT_SUPPORT: Competenze nel supporto IT e gestione dei sistemi informatici.
  - ACCOUNTING: Competenze nella contabilità e gestione finanziaria.
  - MARKETING: Competenze nel marketing e promozione dell'hotel.
  - SECURITY: Competenze nella sicurezza e gestione delle emergenze.
- **Attributo:**
  - description: Una stringa che contiene la descrizione di ogni competenza.
- **Metodi:**
  - getDescription(): Restituisce la descrizione della competenza.

MainView

La classe MainView è responsabile dell'interfaccia grafica principale dell'applicazione di gestione alberghiera. Questa interfaccia fornisce agli utenti un modo per accedere a diverse funzionalità del sistema, come la gestione delle attività, della fatturazione, dei clienti, dei dipendenti, delle prenotazioni e dei servizi.

Attributi:

- activityButton, billingButton, customerButton, employeeButton, reservationButton, serviceButton:
  - Sono i pulsanti che rappresentano le diverse sezioni o funzionalità dell'applicazione. Ogni pulsante, quando cliccato, apre una nuova finestra specifica per la gestione di una particolare area (attività, fatturazione, ecc.).

Costruttore:

- Il costruttore MainView() configura l'interfaccia utente principale:
    - **Titolo e dimensioni**: La finestra viene impostata con il titolo "Hotel Management System" e una dimensione di 400x300 pixel.
    - **Layout**: Utilizza un layout a griglia (GridLayout) con 3 righe e 2 colonne, per disporre i pulsanti in modo ordinato.
    - **Creazione e aggiunta dei pulsanti**: Crea i sei pulsanti principali e li aggiunge alla finestra.
    - **Gestione degli eventi**: Ogni pulsante ha un ActionListener che definisce cosa accade quando l'utente clicca su uno di essi (apre la finestra corrispondente e chiude quella attuale).

Metodi Principali:

- openActivityManager(), openBillingManager(), openCustomerManager(), openEmployeeManager(), openReservationManager(), openServiceManager():
    - Questi metodi vengono chiamati quando l'utente clicca su uno dei pulsanti. Ogni metodo chiude la finestra corrente (dispose()) e apre la finestra specifica per la gestione della relativa area (ad esempio, openActivityManager() apre la finestra per gestire le attività).
    - Ogni metodo crea un nuovo controller, collegando la logica di gestione (modello) con la nuova interfaccia specifica.

4 Conclusioni

In conclusione, lo sviluppo del nostro gestionale per hotel ha seguito un processo meticoloso di progettazione e implementazione, fondato su una suddivisione del lavoro accurata e ben organizzata tra i membri del team. Ciascun componente del gruppo ha apportato il proprio contributo in modo significativo, lavorando su diverse parti del programma per creare un sistema completo e funzionale.

Il risultato è un’applicazione che offre un'ampia gamma di funzionalità per la gestione efficace di un hotel, coprendo aree fondamentali come la gestione dei servizi, la fatturazione, e l'interazione con i clienti. Abbiamo scelto di adottare l'architettura Model-View-Controller (MVC), che ha permesso una netta separazione tra la logica di business, la presentazione dell’interfaccia utente e la gestione degli eventi, migliorando così la manutenibilità e l'espandibilità del software.

Durante lo sviluppo, abbiamo posto particolare attenzione alla progettazione modulare delle classi, garantendo che ogni componente del sistema fosse autonomo ma allo stesso tempo ben integrato nel contesto generale. Questo approccio ha

facilitato l'implementazione delle diverse funzionalità e ha assicurato che il sistema rimanesse flessibile e adattabile a future espansioni.

Inoltre, abbiamo investito tempo nella gestione della persistenza dei dati, assicurandoci che tutte le informazioni rilevanti, come i servizi offerti e le fatture emesse, fossero salvate e recuperate in modo affidabile. La creazione di interfacce utente intuitive e ben strutturate ha poi garantito un'esperienza d'uso semplice e diretta, permettendo al manager di gestire le attività quotidiane dell'hotel con facilità.