

SMARTHAIR

-Fase di Ideazione

Documento di Visione

INTRODUZIONE

Si vuole realizzare un'applicazione per la gestione di una parrucchieria che permette di gestire le prenotazioni in modo efficiente. Il sistema permetterà di ottimizzare la gestione delle prenotazioni suggerendo servizi consoni allo specifico cliente in presenza delle prenotazioni passate. Inoltre, offre una gestione delle pulizie, dello staff e dei pagamenti.

OPPORTUNITÀ DI BUSINESS

Gestire una parrucchieria usando sistemi cartacei porta spesso alla generazione di errori, che possono compromettere la qualità del servizio. SmartHair è stato ideato per eliminare questi problemi, creando un sistema digitale e automatizzato che semplifica ogni aspetto, dalle prenotazioni alla gestione dell'intero palinsesto, vedi pulizie, staff, pagamenti e quant'altro.

PARTI INTERESSATE

Gli utenti principali di SmartHair sono:

- **Amministratore:** gestisce il servizio desiderato dall'utente in funzione delle competenze specifiche del dipendente e si occupa della gestione della pulizia del locale.
- **Cliente:** crea il proprio profilo, si prenota al servizio desiderato, visualizza e modifica le eventuali prenotazioni ed infine, può ricaricare la propria carta.
- **Sistema:** si occupa di detrarre o rimborsare il cliente nella propria tessera qualora esso abbia effettuato o abbia eliminato una prenotazione. Si occupa anche di generare eventuali consigli di prenotazioni qualora il cliente abbia prenotazioni passate.

GENERICITA' SUL PRODOTTO

SmartHair sarà accessibile sia dal personale sia dai clienti attraverso un'interfaccia semplice. Una volta effettuato l'accesso al proprio profilo, i clienti potranno visualizzare direttamente le prenotazioni con i dettagli relativi al servizio, alla data e all'orario.

Modello dei casi d'uso

REQUISITI

SmartHair è un sistema software per la gestione di una parrucchieria. Tale applicazione si occupa di:

- Gestire la creazione delle prenotazioni (inserimento, rimozione e visualizzazione).
- Gestire la registrazione del cliente.
- Gestire le pulizie del locale
- Gestire i servizi di ogni dipendente
- Gestire i pagamenti
- Suggerire i servizi al cliente in funzione delle prenotazioni precedenti

OBIETTIVI E CASI D'USO

Analizzando i requisiti sopra riportati, sono stati individuati gli attori principali e i loro obiettivi. Da queste informazioni sono stati ricavati i seguenti casi d'uso principali:

ATTORE	OBIETTIVO	CASO D'USO
Cliente	Creare il proprio profilo.	UC1: Registrazione utente (CRUD)
Cliente	Prenota, elimina e visualizza una prenotazione	UC2: Gestione Prenotazione
Cliente	Aggiungi credito alla carta.	UC3: Ricarica tessera
Amministratore	Assegna al singolo dipendente il turno di pulizia.	UC4: Gestione delle pulizie
Sistema	Detrazione o rimborso dell'importo della prenotazione della tessera cliente	UC5: Gestione pagamento

Sistema	Generare un consiglio sul servizio futuro che il cliente potrà prenotare in funzione delle loro vecchie prenotazioni	UC6: Generazione consiglio
Amministratore	Assegna il servizio al dipendente	UC7: Assegnazione servizio

Modello dei casi d'uso

Tra i casi d'uso individuati precedentemente, si è scelto di fornire una descrizione dettagliata per i seguenti casi d'uso:

- UC2: Gestione prenotazione
- UC4: Gestione delle pulizie

Gli altri casi d'uso individuati vengono descritti in forma breve.

UC2: Gestione prenotazione

Nome caso d'uso	UC2: Gestione prenotazione
Portata	Applicazione SmartHair
Livello	Obiettivo utente
Attore primario	Cliente
Parti interessate e interessi	Cliente: vuole aggiungere, visualizzare o eliminare una prenotazione.
Pre-Condizioni	Il cliente è registrato e autenticato nel sistema.
Garanzia di successo	Il cliente effettua/visualizza una o più prenotazioni con successo.
Scenario principale di successo	<ol style="list-style-type: none">1. Il cliente accede al proprio profilo.2. Sceglie l'azione desiderata (aggiungi, visualizza o elimina prenotazione).3. Il sistema aggiunge/visualizza/cancella i dati necessari.4. Il sistema verifica la validità dei dati.5. Il cliente conferma l'operazione.6. Il sistema aggiorna le prenotazioni disponibili.
Estensioni	*a. i dati inseriti sono incompleti o errati: il sistema notifica l'errore e richiede la correzione.
Requisiti speciali	
Elenco delle varianti tecnologie e dei dati	
Frequenza di ripetizione	Ogni volta che un cliente desidera aggiungere/modificare/eliminare una prenotazione

UC4: GESTIONE DELLE PULIZIE

Nome caso d'uso	UC4: Gestione delle pulizie
Portata	Applicazione SmartHair
Livello	Obiettivo amministratore/dipendente
Attore primario	Amministratore
Parti interessate e interessi	Amministratore: vuole assegnare al singolo dipendente il turno di pulizia settimanale
Pre-Condizioni	L'amministratore è registrato e autenticato nel sistema e il dipendente non ha assegnato nessun turno di pulizia.
Garanzia di successo	L'amministratore effettua un'assegnazione con successo al dipendente desiderato.
Scenario principale di successo	<ol style="list-style-type: none"> 1. L'amministratore accede al sistema. 2. Sceglie l'opzione “Assegna turno pulizia”. 3. L'amministratore sceglie il dipendente che deve effettuare il servizio di pulizia. 4. SmartHair ritorna l'elenco dei giorni in cui manca un dipendente che faccia le pulizie del locale. 5. L'amministratore seleziona il giorno desiderato. 6. Il sistema verifica il tutto e conferma l'assegnazione. 7. Il sistema notifica all'amministratore i dettagli.
Estensioni	*a. l'amministratore non completa l'assegnazione: il sistema annulla l'operazione dopo un tempo limite.
Requisiti speciali	
Elenco delle varianti tecnologie e dati	
Frequenza di ripetizione	Ogni volta che l'amministratore desidera assegnare i turni di pulizia.

UC1: Registrazione utente

Il cliente registra il proprio profilo. Il sistema verifica la validità del profilo del cliente controllando per esempio se quell'email è già presente.

UC3: Ricarica tessera

Il cliente ricarica la tessera di cui dovrà far uso all'atto della prenotazione.

UC5: Gestione pagamento

Il sistema dovrà occuparsi di detrarre il credito dalla tessera in caso di prenotazione e di rimborsare andando a ricaricare la tessera qualora la prenotazione venga annullata.

UC6: Generazione consiglio

Il sistema genera un consiglio in funzione dell'ultima prenotazione da parte del cliente, suggerendo il servizio prossimo da effettuare, qualora il cliente abbia delle vecchie prenotazioni.

UC7: Assegnazione servizio

L'amministratore assegna il servizio(taglio/colore/piega) al suddetto dipendente in maniera tale che quel dipendente si occuperà solo di quel servizio.

Regole di business

Per il corretto utilizzo del sistema devono essere rispettate le seguenti regole di dominio:

ID	Regola	Modificabilità	Sorgente
R1	Per prenotare un servizio, il paziente deve essere registrato nel sistema.	Bassa	Politica interna del salone
R2	Il cliente può prenotare solo orari disponibili.	Bassa	Politica interna del salone
R3	Il cliente non può prenotarsi ad un servizio se ha un credito minore o uguale a 0.	Media	Politica interna del salone
R4	Un cliente non può effettuare più di una prenotazione per la stessa fascia oraria.	Nulla	Politica interna del salone
R5	Si può effettuare una eliminazione di una prenotazione entro e non oltre le 2 ore da essa.	Bassa	Politica interna del salone
R6	Un dipendente non può effettuare più di un servizio nella stessa fascia oraria	Nulla	Politica interna del salone
R7	Un cliente può prenotarsi entro e non oltre un'ora dall'inizio del servizio.	Media	Politica interna del salone

Specifiche Supplementari

Usabilità

- L'interfaccia deve essere intuitiva e accessibile anche per utenti non esperti.
- L'interazione con il sistema deve essere semplice e immediata, riducendo al minimo la necessità di formazione per il personale del salone e gli amministratori.

Affidabilità e Performance

- Il software deve essere affidabile e garantire la conservazione dei dati anche in caso di guasti hardware o interruzioni di corrente.
- Il sistema deve garantire un'elevata disponibilità, con tempi di risposta rapidi per la gestione delle prenotazioni.

Vincoli Hardware e Software

- Il software deve essere compatibile con qualsiasi sistema operativo purché sia installata la Java Virtual Machine (JVM).

Glossario

- **Sistema:** Software dedicato alla gestione e all'ottimizzazione delle attività di un salone di parrucchieri, che semplifica l'amministrazione.
- **Amministratore di Sistema:** Gestore del salone che si occupa della programmazione dei servizi.
- **Cliente:** Utente registrato su SmartHair che può prenotare un servizio, visualizzare le prenotazioni, eliminare le prenotazioni.
- **Personale del salone:** Professionista qualificato che si occupa di eseguire il servizio indicato dal cliente.
- **Servizio:** servizio richiesto da un utente tenuto da un professionista del personale(taglio/pieghe/colore).
- **Prenotazione:** Procedura attraverso la quale un cliente sceglie il servizio e la data.
- **Salone:** Struttura fisica in cui vengono eseguiti i servizi.
- **Tessera:** componente che serve per mantenere un conto da parte dell'utente.

- Elaborazione – Iterazione 1

Introduzione

Conclusa la fase di ideazione, si passa a quella di elaborazione. Scopo delle iterazioni seguenti sarà quello di: raffinare la visione, implementare in maniera iterativa il nucleo dell'architettura del software, risolvere le problematiche relative ai rischi maggiori, identificare la maggior parte dei requisiti e la portata, fornire delle stime più realistiche del piano di lavoro e delle risorse complessive. Durante questa prima iterazione i requisiti scelti su cui concentrarsi sono i seguenti:

- Implementare lo scenario principale di successo dei casi d'uso
 - UC1: Registrazione Utente
 - UC2: Gestione Prenotazione
 - UC3: Ricarica tessera
- Implementare i casi d'uso di Start Up necessari per gestire le esigenze di inizializzazione per questa iterazione.

Modello dei casi d'uso

UC1: Registrazione utente

Nome caso d'uso	UC1: Registrazione utente
Portata	Applicazione SmartHair
Livello	Obiettivo utente
Attore primario	Cliente
Parti interessate e interessi	<p>Cliente: vuole registrarsi e quindi creare un suo profile utente.</p> <p>Tessera: viene generata associata al profile creato dall'utente ed inizializzata a 0.</p>
Pre-Condizioni	L'applicazione SmartHair è attiva e funzionante
Garanzia di successo	Il cliente effettua la registrazione con successo. Viene generata la tessera in modo automatico.
Scenario principale di successo	<ol style="list-style-type: none"> 1. Il cliente accede all'applicazione. 2. Inserisce i suoi dati personali. 3. Il sistema verifica la validità dei dati. 4. Il cliente genera la tessera associata al cliente. 5. Il sistema aggiorna le informazioni sugli utenti registrati. 6. Il cliente accede all'app eseguendo il login tramite e-mail e password.
Estensioni	*a. i dati inseriti sono incompleti o errati: il sistema notifica l'errore e richiede la correzione.
Requisiti speciali	
Elenco delle varianti tecnologie e dei dati	
Frequenza di ripetizione	Ogni volta che un cliente desidera creare un nuovo profilo.

UC2: Gestione prenotazione

Nome caso d'uso	UC2: Gestione prenotazione
Portata	Applicazione SmartHair

Livello	Obiettivo utente
Attore primario	Cliente
Parti interessate e interessi	Cliente: vuole aggiungere, visualizzare o eliminare una prenotazione. Tessera: viene controllato che il credito sia adeguato per la prenotazione.
Pre-Condizioni	Il cliente è registrato, autenticato nel Sistema e possiede un credito superiore al credito necessario per prenotarsi al servizio desiderato.
Garanzia di successo	Il cliente effettua/visualizza una o più prenotazioni con successo.
Scenario principale di successo	<ol style="list-style-type: none"> 1. Il cliente accede al proprio profilo. 2. Sceglie l'azione desiderata (aggiungi, visualizza o elimina prenotazione). 3. Il Sistema controlla che il conto nella tessera sia adeguato. 4. Il sistema aggiunge/visualizza/cancella i dati necessari. 5. Il sistema verifica la validità dei dati. 6. Il cliente conferma l'operazione. 7. Il Sistema genera una nuova prenotazione 8. Il sistema aggiorna le prenotazioni disponibili.
Estensioni	*a. i dati inseriti sono incompleti o errati: il sistema notifica l'errore e richiede la correzione.
Requisiti speciali	
Elenco delle varianti tecnologie e dei dati	
Frequenza di ripetizione	Ogni volta che un cliente desidera aggiungere/modificare/eliminare una prenotazione

UC3: Ricarica tessera

Nome caso d'uso	UC3: Ricarica tessera
Portata	Applicazione SmartHair
Livello	Obiettivo utente
Attore primario	Cliente
Parti interessate e interessi	Cliente: vuole ricaricare la propria tessera per poter effettuare una prenotazione. Tessera: deve essere ricaricato il suo conto
Pre-Condizioni	Il cliente è registrato e autenticato nel sistema.

Garanzia di successo	Il cliente effettua la ricarica con successo.
Scenario principale di successo	<ol style="list-style-type: none"> 1. Il cliente accede al proprio profilo. 2. Sceglie l'azione di ricarica tessera. 3. Il sistema verifica la validità dei dati. 4. Il cliente conferma l'operazione. 5. Il sistema aggiorna il nuovo credito presente nella tessera dell'utente.
Estensioni	*a. i dati inseriti sono incompleti o errati: il sistema notifica l'errore e richiede la correzione.
Requisiti speciali	
Elenco delle varianti tecnologie e dei dati	
Frequenza di ripetizione	Ogni volta che un cliente desidera aggiornare il suo credito.

Analisi orientata agli oggetti

L'analisi orientata agli oggetti si basa sulla creazione di una descrizione del dominio da un punto di vista ad oggetti. Vengono utilizzati diversi strumenti per fornire tale descrizione: Modello di Dominio, SSD (Sequence System Diagram) e Contratti delle operazioni.

Modello di Dominio

Il modello di dominio di SmartHair rappresenta i concetti chiave, gli attributi e le relazioni fondamentali che compongono il sistema. Dopo un'analisi del contesto e dei casi d'uso, sono state identificate le seguenti classi concettuali:

- **SmartHair:** Rappresenta l'intero sistema software per la gestione della parrucchiaria
- **Amministratore di sistema:** Gestore del salone che si occupa della programmazione dei servizi.
- **Cliente:** Utente registrato su SmartHair che può prenotare un servizio, visualizzare le prenotazioni, eliminare le prenotazioni.
- **Parrucchiere:** Professionista qualificato che si occupa di eseguire il servizio indicato dal cliente.
- **Tessera:** componente che serve per mantenere un conto da parte dell'utente.
- **Prenotazione:** Procedura attraverso la quale un cliente sceglie il servizio e la data.

pkg ModelloDiDominio

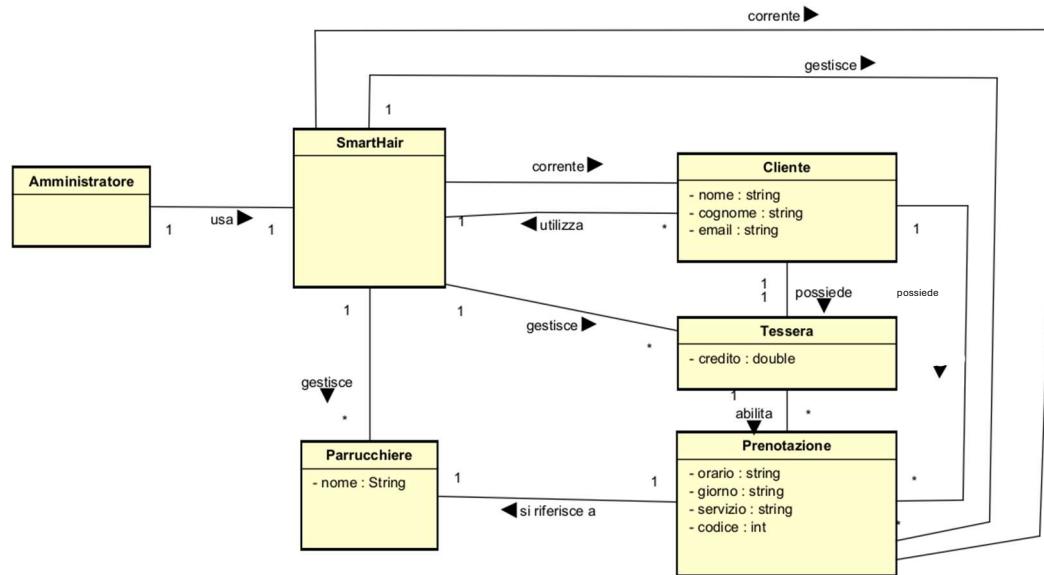
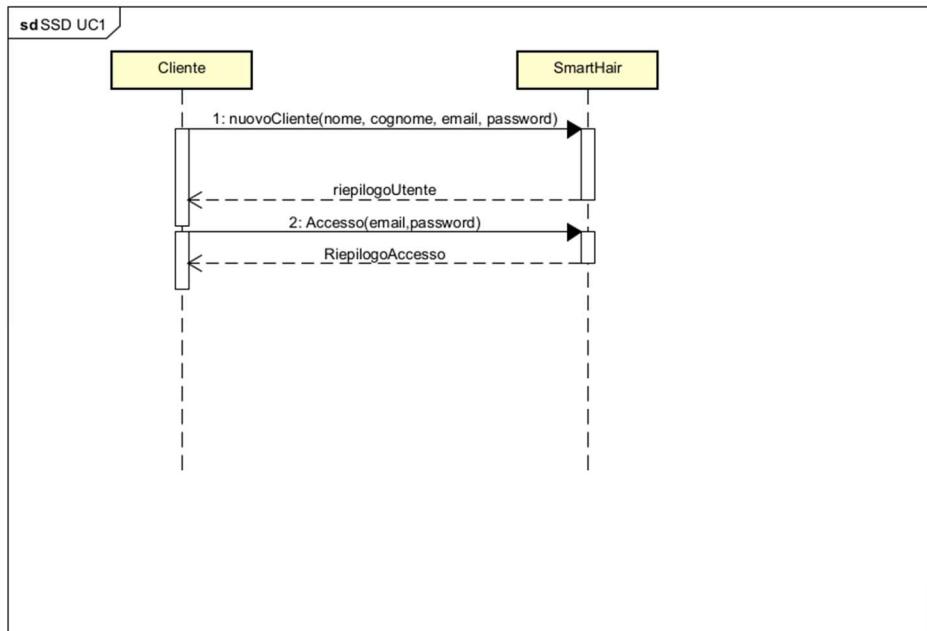


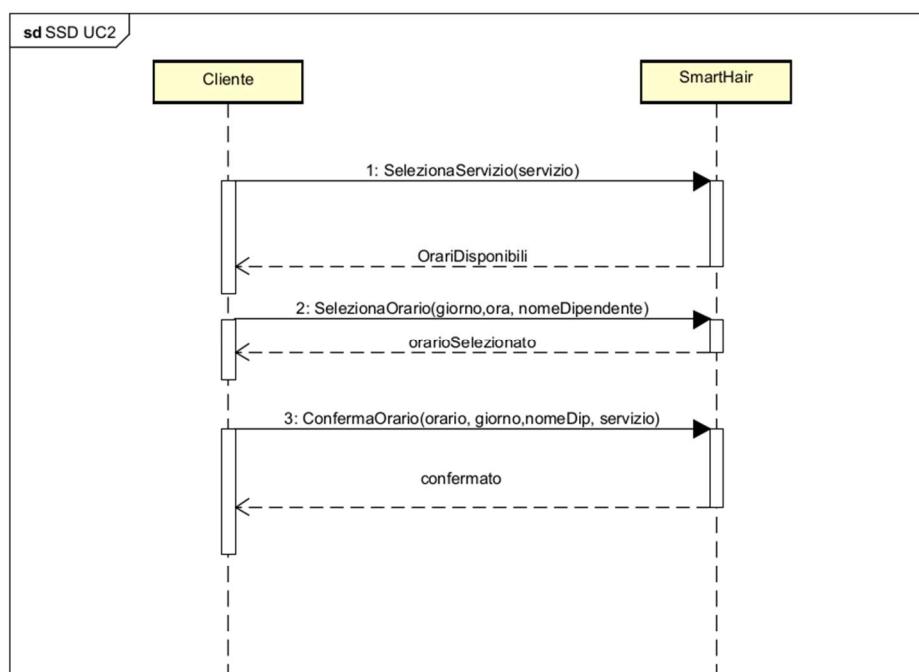
Diagramma di sequenza di sistema

Procedendo con l'analisi Orientata agli Oggetti, il passo successivo è la creazione del Diagramma di Sequenza di Sistema (SSD) al fine di illustrare il corso degli eventi di input e di output per lo scenario principale di successo dei casi d'uso scelti (UC1, UC2 e UC3), quindi avremo:

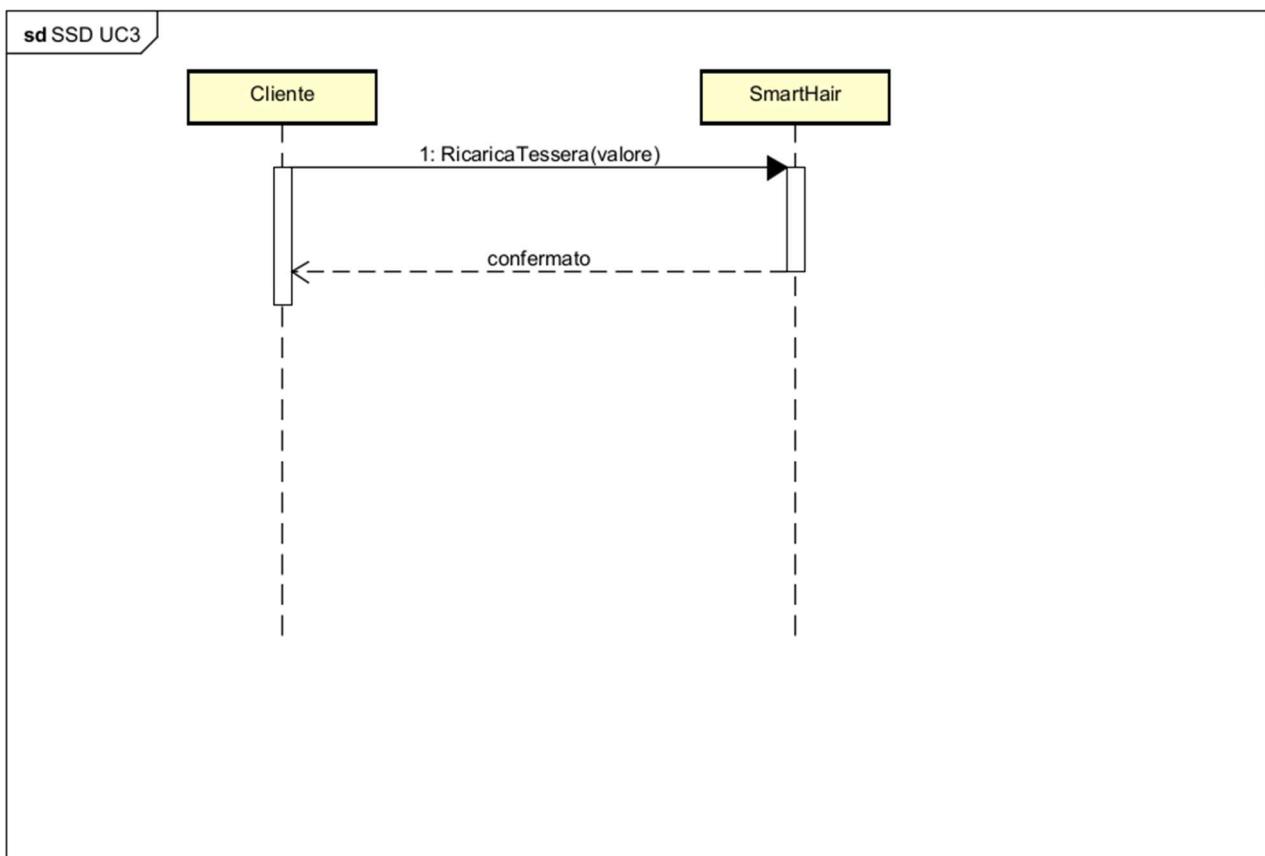
SSD- Caso d'uso UC1: Registrazione utente



SSD- Caso d'uso UC2: Gestione prenotazione



SSD- Caso d'uso UC3: Ricarica tessera



Contratti delle operazioni

Vengono ora descritte attraverso i Contratti le principali operazioni di sistema che si occupano di gestire gli eventi di sistema individuati negli SSD.

Contratti Operazioni UC1

Contratto CO1: nuovoCliente

Operazione	nuovoCliente(nome: String, cognome: String, email: String, password: String)
Riferimenti	Caso d'uso UC1: Registrazione utente
Pre-Condizioni	<ul style="list-style-type: none">L'applicazione SmartHair deve essere avviata.Non deve esistere già un cliente con la stessa e-mail.
Post-Condizioni	<ul style="list-style-type: none">È stata creata un'istanza di Cliente.Il cliente è stato inizializzato con le informazioni inserite

	<ul style="list-style-type: none"> • Il cliente è stato associato a SmartHair tramite l'associazione “corrente”.
--	---

Contratto CO2: Accesso

Operazione	Accesso(email: String, password: String)
Riferimenti	Caso d'uso UC1: Registrazione utente
Pre-Condizioni	<ul style="list-style-type: none"> • L'applicazione SmartHair deve essere avviata.
Post-Condizioni	<ul style="list-style-type: none"> • Il cliente ha effettuato l'accesso con successo

Contratti Operazioni UC2

Contratto CO1: SelezionaServizio

Operazione	SelezionaServizio(servizio: String)
Riferimenti	Caso d'uso UC2: Gestione Prenotazione
Pre-Condizioni	<ul style="list-style-type: none"> • L'applicazione SmartHair deve essere avviata. • L'utente deve essere loggato.
Post-Condizioni	<ul style="list-style-type: none"> • È stato selezionato un servizio per il quale prenotare un orario disponibile.

Contratto CO2: SelezionaOrario

Operazione	SelezionaOrario(Giorno: String, ora: String, nomeDipendente:String)
Riferimenti	Caso d'uso UC2: Gestione Prenotazione

Pre-Condizioni	<ul style="list-style-type: none"> • L'applicazione SmartHair deve essere avviata. • L'utente deve essere loggato. • È stato selezionato un servizio per il quale prenotare un orario disponibile. • Il credito è adeguato. • C'è almeno un orario disponibile per il servizio.
Post-Condizioni	<ul style="list-style-type: none"> • È stato selezionato un orario disponibile per il servizio desiderato.

Contratto CO3: ConfermaOrario

Operazione	ConfermaOrario(orario: String, giorno: String, nomeDip: String, servizio: String)
Riferimenti	Caso d'uso UC2: Gestione Prenotazione
Pre-Condizioni	<ul style="list-style-type: none"> • L'applicazione SmartHair deve essere avviata. • L'utente deve essere loggato. • È stato selezionato un servizio per il quale prenotare un orario disponibile. • Il credito è adeguato.
Post-Condizioni	<ul style="list-style-type: none"> • È stata creata un'istanza di prenotazione. • La prenotazione è stata associata a SmartHair tramite l'associazione "corrente".

Contratti Operazioni UC3

Contratto CO1: RicaricaTessera

Operazione	RicaricaTessera (valore: int)
Riferimenti	Caso d'uso UC3: Ricarica Tessera
Pre-Condizioni	<ul style="list-style-type: none"> • L'applicazione SmartHair deve essere avviata.

	<ul style="list-style-type: none"> L'utente deve essere loggato e deve avere una tessera associata.
Post-Condizioni	<ul style="list-style-type: none"> È stata effettuata la ricarica del valore fornito

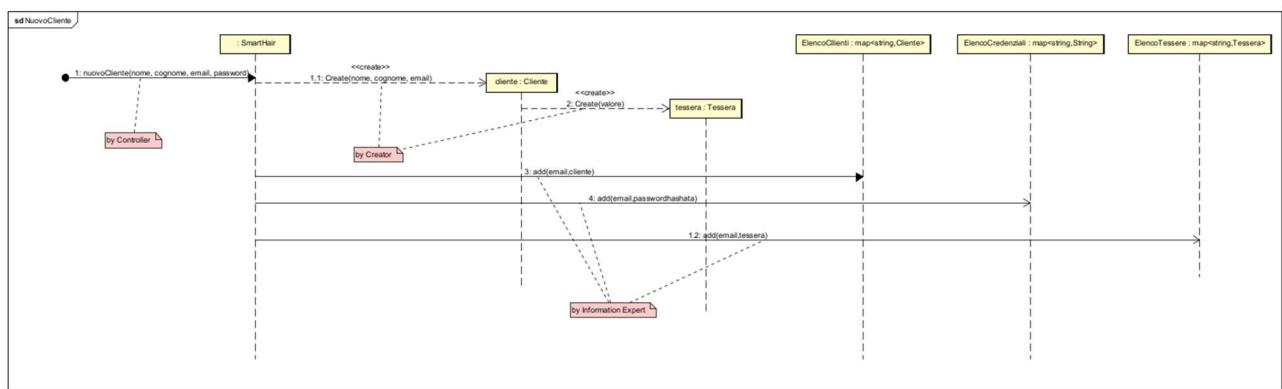
Progettazione

La progettazione orientata agli oggetti è la disciplina interessata alla definizione degli oggetti software, delle loro responsabilità e a come questi collaborano per soddisfare i requisiti individuati nei passi precedenti. L'elaborato principale di questa fase che è stato preso in considerazione è il Modello di Progetto, ovvero l'insieme dei diagrammi che descrivono la progettazione logica sia da un punto di vista dinamico (Diagrammi di Interazione) che da un punto di vista statico (Diagramma delle Classi). Seguono dunque i diagrammi di Interazione più significativi e il diagramma delle Classi.

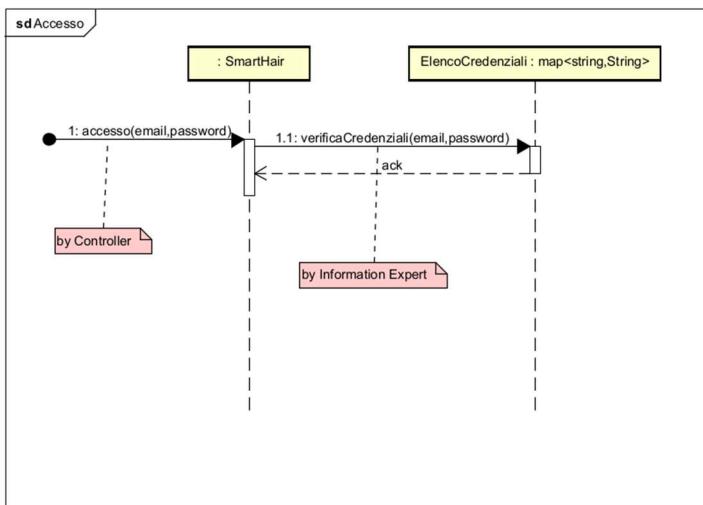
Diagrammi di Sequenza

UC1

• nuovoCliente

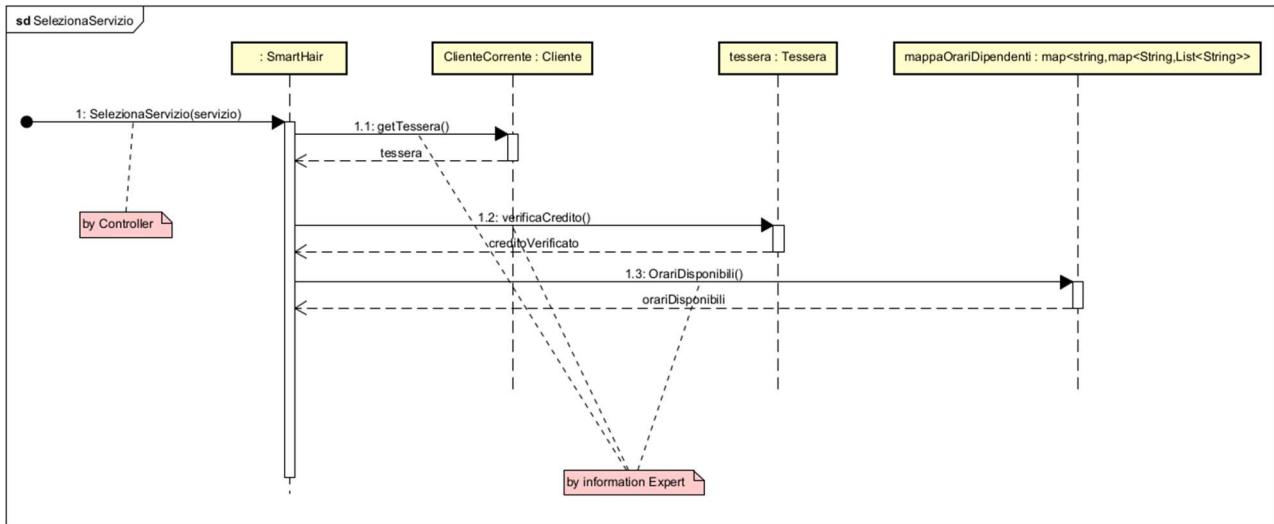


• Accesso

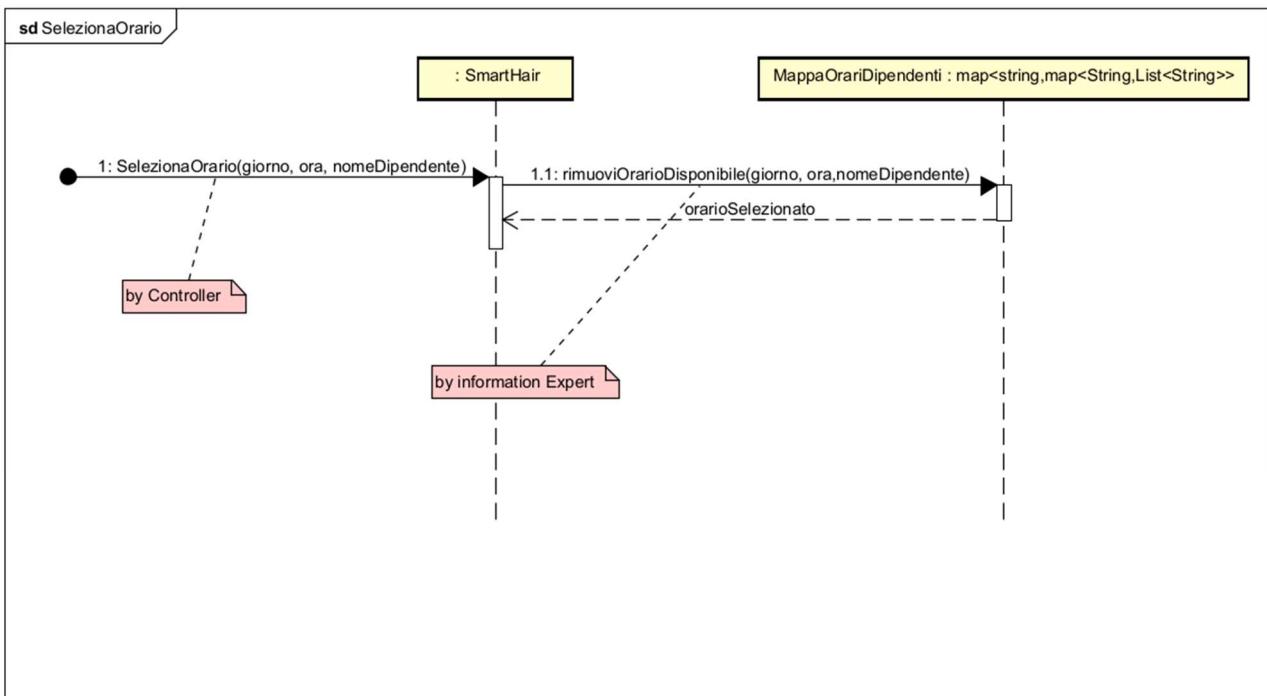


UC2

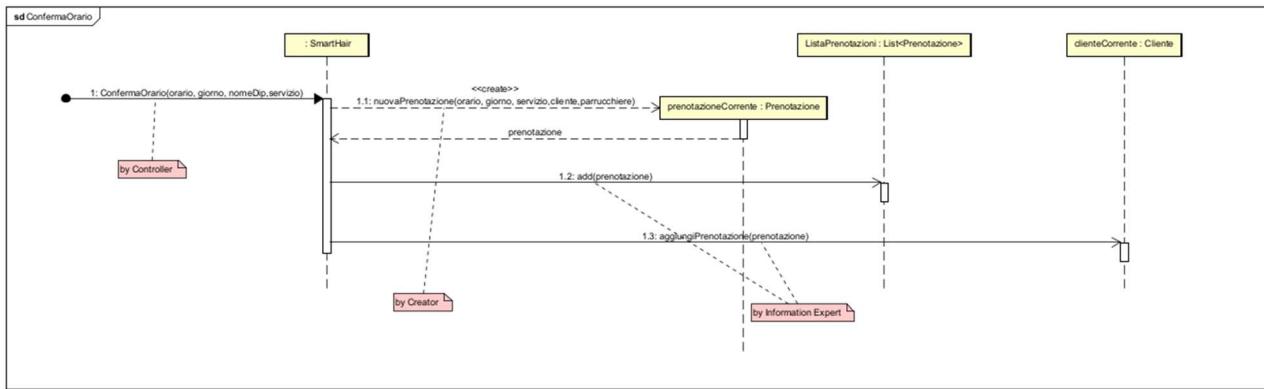
• SelezioneServizio



• SelezioneOrario

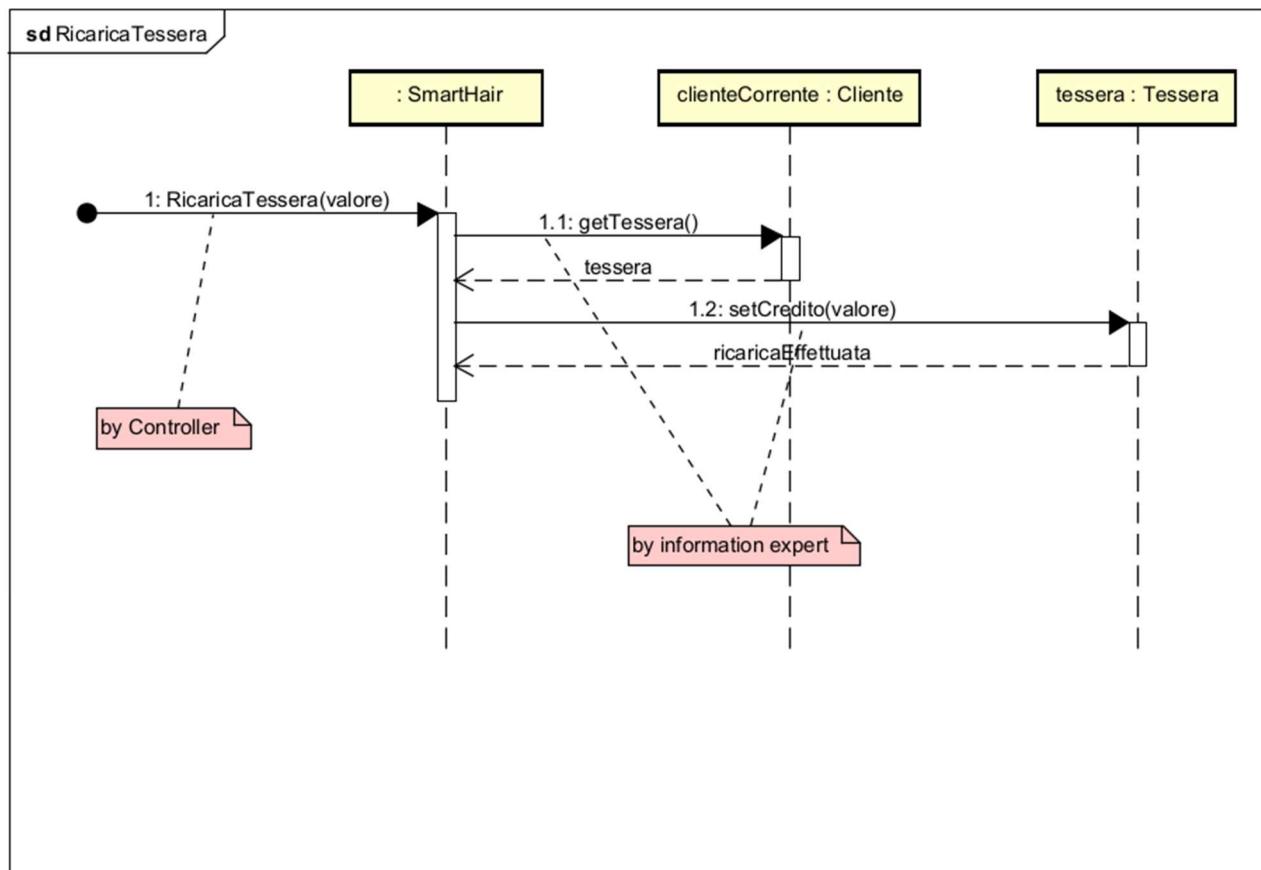


• ConfermaOrario



UC3

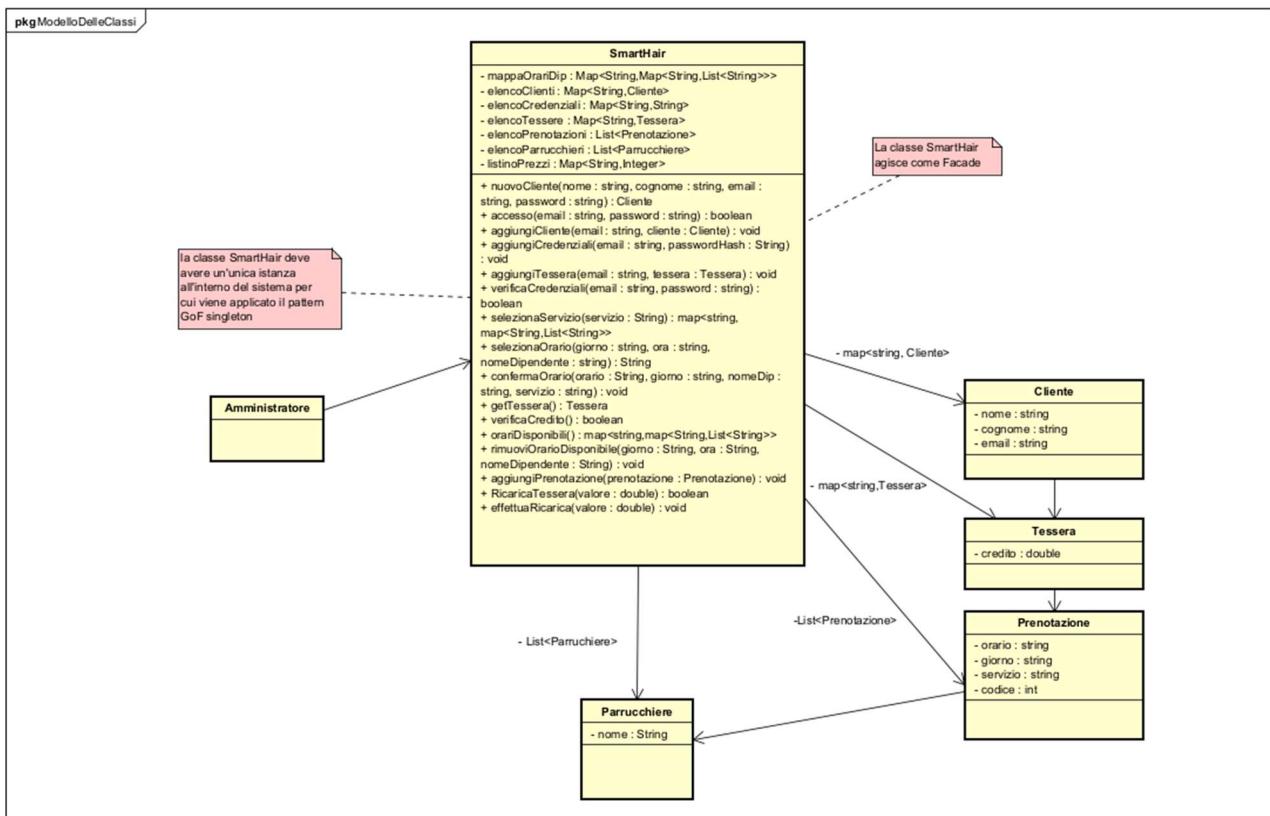
• RicaricaTessera



Nota:

nella fase di ideazione avevamo inserito il caso d'uso 7 che trattava l'assegnazione del servizio al dipendente tramite smarthair, ma eseguendo questa prima iterazione abbiamo notato che risulta essere inutile perché offriamo la possibilità al cliente di scegliere il dipendente che deve eseguire il particolare servizio ed ogni dipendente può eseguire qualsiasi servizio tra quelli indicati.

Diagramma delle classi



Test

Il testing rappresenta una fase fondamentale nello sviluppo del sistema SmartHair, poiché garantisce che le funzionalità implementate rispondano ai requisiti richiesti. Testare il software in maniera sistematica ha permesso non solo di intercettare e correggere eventuali errori durante le varie fasi dello sviluppo, ma anche di costruire un'applicazione più solida e conforme ai requisiti definiti. Un processo di testing efficace consente inoltre di ridurre i costi di manutenzione a lungo termine, prevenendo l'insorgenza di errori nelle fasi successive dello sviluppo. Abbiamo realizzato i test del progetto SmartHair utilizzando il framework JUnit, che consente di automatizzare la verifica del corretto funzionamento delle classi e dei metodi sviluppati. L'approccio seguito è di tipo white-box testing, poiché i test sono stati progettati conoscendo la struttura interna del codice e verificando che ogni metodo produca i risultati attesi in base alla logica implementata. In particolare, sono stati eseguiti test unitari per verificare in maniera isolata il corretto comportamento di ogni funzione rispetto ai valori aspettati .

Individuazione dei casi di test:

SmartHairTest:

-setUp():

azzerà forzatamente l'istanza SmartHair.instance (è un Singleton), così ogni test parte da uno stato pulito. Poi crea una nuova istanza e registra un cliente.

-testNuovoClienteRegistraCorrettamente():

verifica che il costruttore/registrazione di un nuovo cliente inizializza correttamente:

- nome = "Mario"
- tessera a 0.0
- nessuna prenotazione iniziale

-testAccessoCorrettoElImpostaClienteCorrente():

verifica il login con credenziali valide, infatti:

- ritorna true
- imposta clienteCorrente
- il cliente corrente è proprio "Mario"

questo conferma che l'autenticazione e lo stato di sessione funzionano.

-testAccessoFallisceConPasswordErrata():

con password sbagliata il login fallisce e non viene impostato clienteCorrente

-testRicaricaTesseraAggiornaCredito():

la ricarica:

- richiede utente loggato
- aumenta il credito della tessera della cifra indicata
- restituisce true

questo garantisce che il saldo venga gestito correttamente.

-testRicaricaTesseraRifiutaValoriNegativi():

verifica che le ricariche non valide (negative) sono rifiutate.

-testLogoutReimpostaClienteCorrente():

verifica che il logout azzerà clienteCorrente.

-testConfermaPrenotazioneCreaNuovaPrenotazione():

Confermando un orario valido viene **creata e aggiunta** una Prenotazione al cliente.

-testCancellaPrenotazioneFunzionaCorrettamente():

data una prenotazione appena creata, cancellaPrenotazione(codice) la rimuove e ritorna true

-testStampaPrenotazioniClienteMostraPrenotazioniCorrette():

l'output a console di stampaPrenotazioniCliente() contiene:

- l'intestazione (es. “le tue prenotazioni”),
- il giorno (“martedì”),
- il nome del parrucchiere (“vincenzo”).

Lo fa reindirizzando temporaneamente System.out su un buffer per ispezionare il testo stampato. Lo facciamo perché conferma che la stampa mostra info essenziali della prenotazione.

ClienteTest:

-setUp():

viene eseguito prima di ogni test. Serve per creare un nuovo oggetto Cliente “pulito” in modo che ogni test parta da zero (nessuna prenotazione, tessera inizializzata, ecc.).

-testCostruttoreEGetter():

Controlla che il costruttore di Cliente imposti correttamente:

- nome, cognome, email
- la tessera non sia null
- il credito iniziale sia 0.0
- la lista di prenotazioni sia vuota all'inizio

In pratica verifica che la creazione dell'oggetto Cliente sia corretta.

- testAggiungiPrenotazione():

Crea un parrucchiere e una prenotazione. Aggiunge la prenotazione al cliente con cliente.aggiungiPrenotazione(p). Poi controlla:

- che la lista delle prenotazioni contenga esattamente 1 elemento;
- che quell'elemento sia esattamente la stessa istanza (assertSame, non solo uguale nei valori).

Quindi serve per verificare che il metodo aggiungiPrenotazione() funzioni correttamente e aggiunga l'oggetto giusto.

- testAggiungiPrenotazioneNull():

Prova ad aggiungere una prenotazione null. Controlla che la lista **resti vuota**. Quindi serve per verificare che aggiungiPrenotazione() sia **robusto** e **ignori input nulli**.

-testRimuoviPrenotazione():

Aggiunge una prenotazione. Poi la rimuove. Verifica che, dopo la rimozione, **la lista sia vuota**. Serve per testare che rimuoviPrenotazione() funzioni e **tolga effettivamente** l'oggetto dalla lista del cliente.

ParrucchiereTest:

-setUp():

Viene creato un'istanza di parrucchiere prima di ogni test, così i test sono indipendenti.

-testGetNomeRestituisceValoreCorretto:

Verifica che getNome() non restituisce null e restituisce il valore atteso.

-testOggettiDiversiConStessoNomeSonoUgualiSoloSeEqualsDefinito:

Verifica che due istanze diverse con lo stesso nome risultano diverse se equals non è ridefinito.

PrenotazioneTest:

-setUp():

Inizializza prima di ogni test un cliente e un parrucchiere, così i test non si influenzano tra loro.

-testCostruttoreAssegnaDatiCorrettamente():

il costruttore di Prenotazione assegna correttamente tutti i campi (orario, giorno, servizio, cliente, parrucchiere).

-testCodiceIncrementaleUnivoco():

verifica che il campo codice delle prenotazioni è univoco e incrementale

TesseraTest:

-setUp():

Crea una istanza di tessera con credito iniziale 50.0 prima di ogni test. Così i test sono indipendenti.

-testCostruttoreEGetter():

Verifica che il costruttore assegna correttamente il credito iniziale e getCredito() lo restituisce senza errori.

-testSetterAggiornaCredito():

Verifica che setCredito() aggiorna davvero il valore memorizzato.

-testSetterAccettaZero():

il credito può essere impostato a zero senza eccezioni o blocchi.

Pattern GoF usati:

in questa prima iterazione abbiamo fatto uso di due pattern GoF. Il primo è il pattern Singleton mentre il secondo è il pattern Facade. La classe SmartHair usa il pattern Singleton perché deve esistere una sola istanza centrale dell'applicazione che gestisce tutti i dati condivisi (clienti, tessere, prenotazioni, parrucchieri). Il costruttore è privato e l'accesso avviene tramite getInstance(), garantendo un unico punto di controllo. SmartHair agisce anche come Facade perché fornisce un'interfaccia semplificata per interagire con più sottosistemi (classi Cliente, Tessera, Prenotazione, Parrucchiere), nascondendo la complessità interna e coordinando tutte le operazioni del sistema da un unico punto.

-Elaborazione – Iterazione 2

Introduzione

Conclusa la prima fase di iterazione, si passa alla seconda. Durante questa seconda iterazione ci si concentrerà su:

- Implementare lo scenario principale di successo dei casi d'uso
 - UC4: Gestione delle pulizie
 - UC5: Gestione pagamento
 - UC6: Generazione consiglio

L'integrazione di questi casi d'uso consentirà di completare il flusso totale all'interno del sistema SmartHair, migliorando l'efficienza operativa del salone e l'esperienza utente per i clienti.

Modello dei casi d'uso

UC4: Gestione delle pulizie

Nome caso d'uso	<i>UC4: Gestione delle pulizie</i>
Portata	Applicazione SmartHair
Livello	Obiettivo amministratore
Attore primario	Amministratore
Parti interessate e interessi	Amministratore: vuole assegnare il turno di pulizia ad un dipendente Parrucchiere: gli viene assegnato il turno di pulizia da eseguire
Pre-Condizioni	L'applicazione SmartHair è attiva, funzionante e l'amministratore si è loggato
Garanzia di successo	L'amministratore effettua l'assegnazione con successo.

Scenario principale di successo	<p>7. L'amministratore accede all'applicazione.</p> <p>8. Inserisce i suoi dati personali.</p> <p>9. Il sistema verifica la validità dei dati.</p> <p>10. Il Sistema recupera i giorni a cui non è stato assegnato il turno di pulizia</p> <p>11. L'amministratore inserisce il nome del dipendente ed il Giorno di pulizia</p> <p>12. Il Sistema controlla i dati inseriti e carica il tutto.</p>
Estensioni	*a. i dati inseriti sono incompleti o errati: il sistema notifica l'errore e richiede la correzione.
Requisiti speciali	
Elenco delle varianti tecnologie e dei dati	
Frequenza di ripetizione	Ogni volta che l'amministratore desidera assegnare un turno di pulizia settimanale al dipendente desiderato.

UC5: Gestione pagamento

Nome caso d'uso	UC5: Gestione pagamento
Portata	Applicazione SmartHair
Livello	Obiettivo utente
Attore primario	Cliente
Parti interessate e interessi	Cliente: effettua/cancella una prenotazione Sistema: vuole detrarre/rimborsare il credito dalla tessera
Pre-Condizioni	Il cliente è registrato e autenticato nel Sistema.
Garanzia di successo	Il Sistema detrae o rimborsa il cliente con successo nella sua tessera.
Scenario principale di successo	<p>9. Il cliente accede al proprio profilo.</p> <p>10. Sceglie l'azione desiderata (aggiungi, elimina prenotazione).</p> <p>11. Il sistema aggiunge/ cancella i dati necessari.</p> <p>12. Il sistema detrae/rimborsa nella tessera del cliente.</p>

Estensioni	
Requisiti speciali	
Elenco delle varianti tecnologie e dei dati	
Frequenza di ripetizione	Ogni volta che un cliente desidera aggiungere/ eliminare una prenotazione

UC6: Generazione consiglio

Nome caso d'uso	UC6: Generazione consiglio
Portata	Applicazione SmartHair
Livello	Obiettivo utente
Attore primario	Sistema
Parti interessate e interessi	Cliente: vuole effettuare una prenotazione. Sistema: vuole generare un Consiglio in funzione delle vecchie prenotazioni del cliente.
Pre-Condizioni	Il cliente si è loggato, ha almeno una vecchia prenotazione e non ha ancora ricevuto un Consiglio.
Garanzia di successo	Il Sistema genera un consiglio sulla prossima prenotazione che il cliente dovrà fare.
Scenario principale di successo	<ol style="list-style-type: none"> 6. Il cliente accede al proprio profilo. 7. Il Sistema, se il cliente ha delle vecchie prenotazioni, genera un consiglio. 8. Il cliente accetta o rifiuta il consiglio. 9. Se il cliente accetta il consiglio, allora il Sistema aggiunge una nuova prenotazione con i dati inerenti al consiglio generato dal sistema. 10. Il sistema inserisce la prenotazione nell'elenco di prenotazioni.
Estensioni	*a. i dati inseriti sono incompleti o errati: il sistema notifica l'errore e richiede la correzione.
Requisiti speciali	
Elenco delle varianti tecnologie e dei dati	
Frequenza di ripetizione	Ogni volta che un cliente desidera prenotarsi al servizio e possiede almeno una Vecchia prenotazione

Refactoring

A differenza dell'iterazione 1, in questa iterazione che stiamo trattando abbiamo modificato parte del caso d'uso 2(Gestione Prenotazione), in particolare il metodo confermaOrario, per andare a detrarre/rimborsare il credito qualora l'utente effettui/elimini una prenotazione.

Analisi orientata agli oggetti

Al fine di descrivere il dominio da un punto di vista orientato agli oggetti e per gestire i nuovi requisiti introdotti in questa iterazione, saranno nuovamente utilizzati gli stessi strumenti impiegati nella fase precedente, ovvero il Modello di Dominio, il Sequence System Diagram (SSD) e i Contratti delle operazioni. Nei paragrafi seguenti verranno evidenziati i cambiamenti e le estensioni apportate a questi strumenti rispetto all'iterazione precedente, in relazione ai nuovi casi d'uso implementati.

Modello di Dominio

Relativamente ai casi d'uso scelti per questa iterazione (UC5, UC6 e UC7), dopo un'attenta analisi dello scenario principale di successo, è stato possibile identificare il seguente modello di dominio:

- **SmartHair:** Rappresenta l'intero sistema software per la gestione della parrucchieria.
- **Amministratore di sistema:** Gestore del salone che si occupa della programmazione dei servizi.
- **Cliente:** Utente registrato su SmartHair che può prenotare un servizio, visualizzare le prenotazioni, eliminare le prenotazioni.
- **Parrucchiere:** Professionista qualificato che si occupa di eseguire il servizio indicato dal cliente ed i servizi di pulizia indicati dall'amministratore.
- **Tessera:** componente che serve per mantenere un conto da parte dell'utente.
- **Prenotazione:** Procedura attraverso la quale un cliente sceglie il servizio e la data.
- **Consiglio:** generato automaticamente da SmartHair, associato a un Cliente.

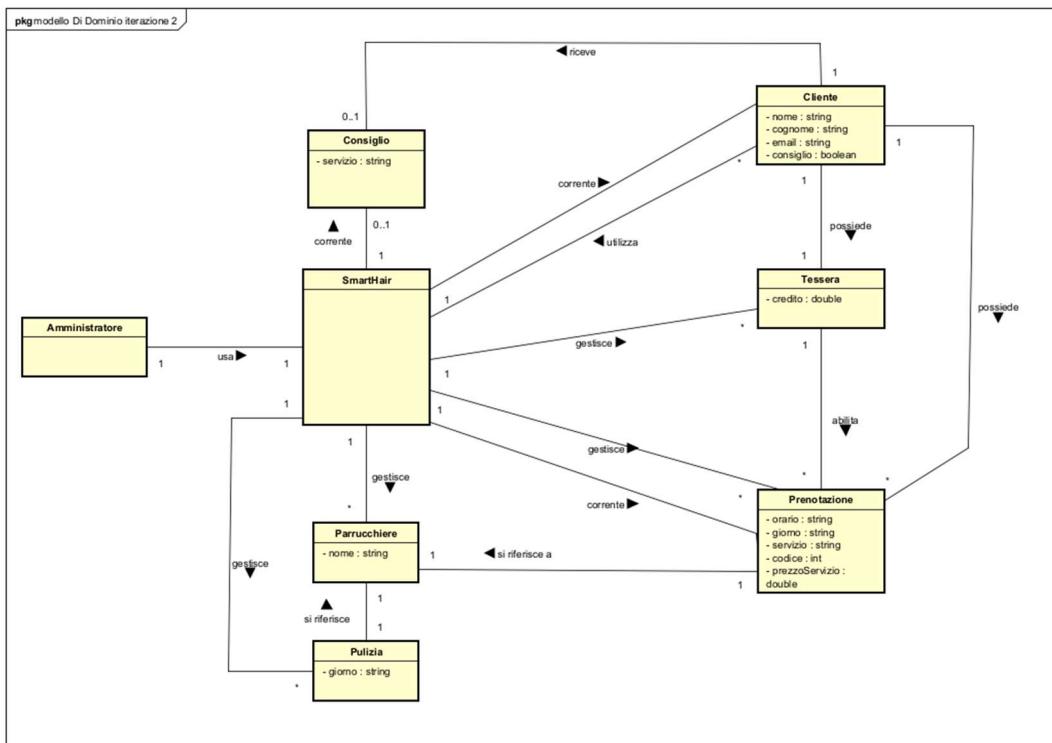
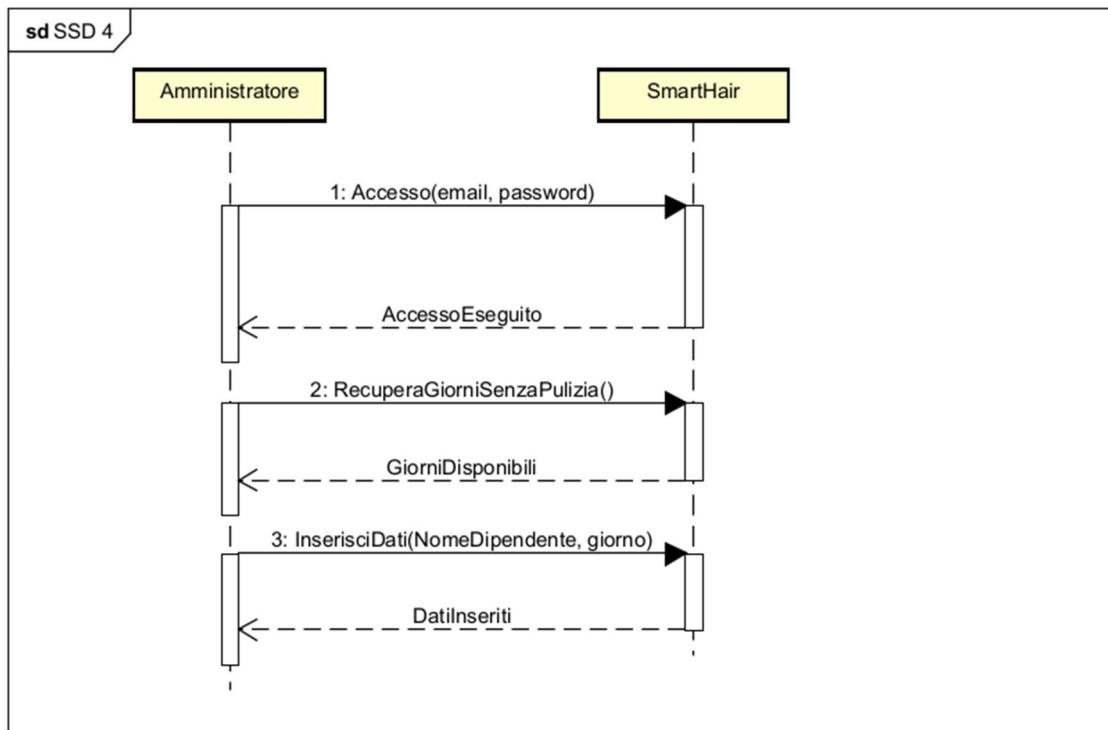


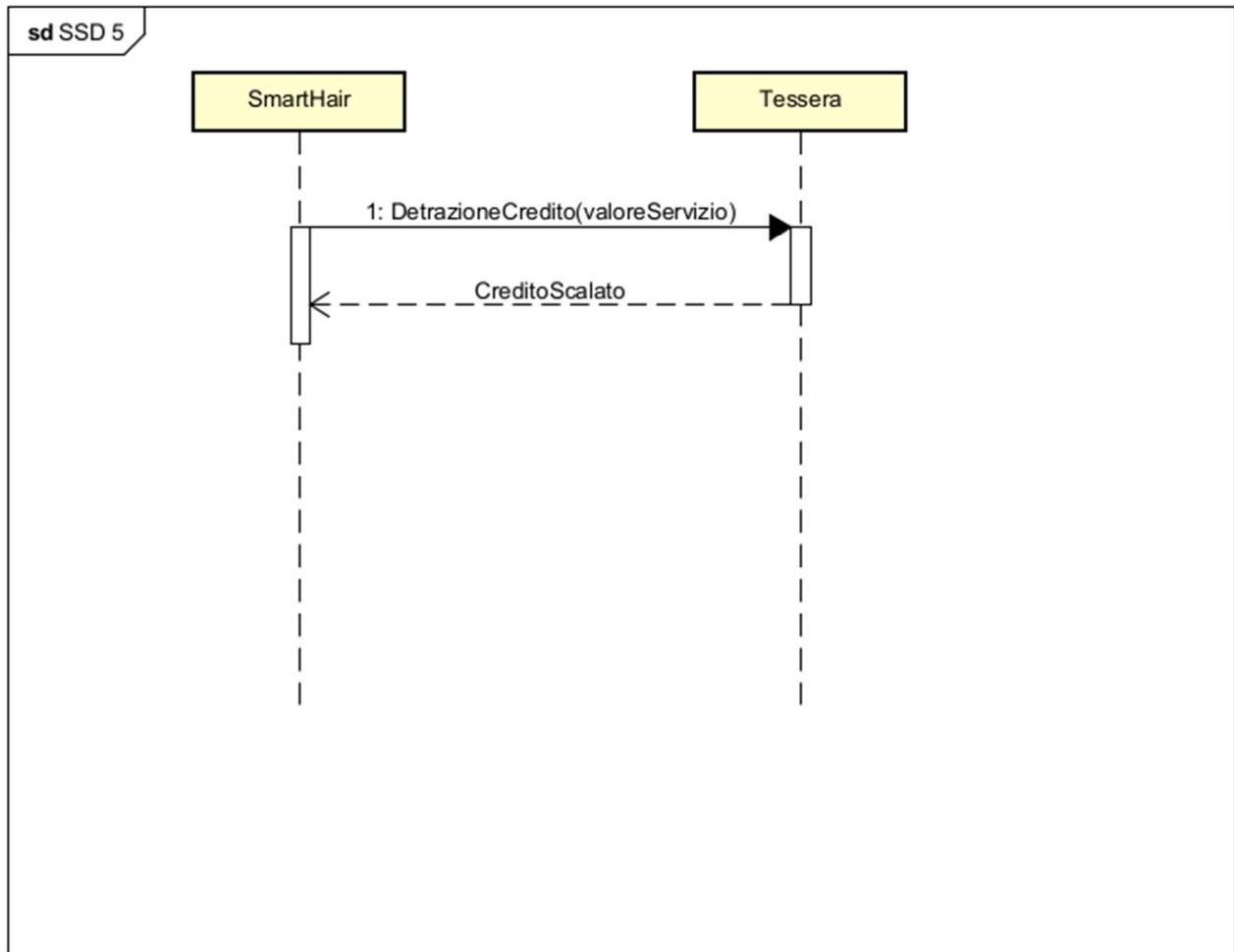
Diagramma di sequenza di sistema

Procedendo con l'analisi Orientata agli Oggetti, il passo successivo è la creazione del Diagramma di Sequenza di Sistema (SSD) al fine di illustrare il corso degli eventi di input e di output degli ultimi casi d'uso (UC4, UC5 e UC6), quindi avremo:

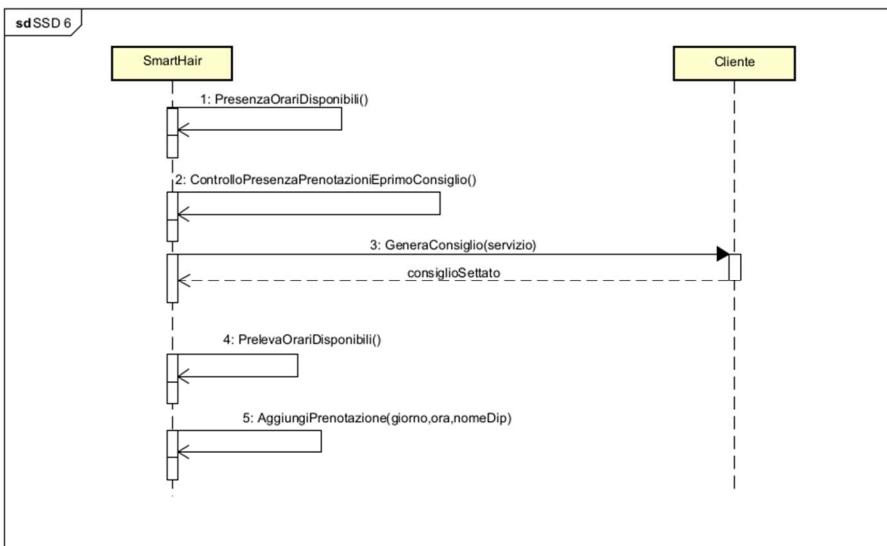
SSD- Caso d'uso UC4: Gestione delle pulizie



SSD- Caso d'uso UC5: Gestione pagamento



SSD- Caso d'uso UC6: Generazione consiglio



Contratti delle operazioni

Vengono ora descritte attraverso i contratti le principali operazioni di sistema che si occupano di gestire gli eventi di sistema individuati negli SSD.

Contratti Operazioni UC4

Contratto CO1: Accesso

Operazione	Accesso(email: String, password: String)
Riferimenti	Caso d'uso UC4: Gestione delle pulizie
Pre-Condizioni	<ul style="list-style-type: none">L'applicazione SmartHair deve essere avviata.
Post-Condizioni	<ul style="list-style-type: none">L'amministratore ha effettuato l'accesso con successo

Contratto CO2: RecuperaGiorniSenzaPulizia

Operazione	RecuperaGiorniSenzaPulizia ()
Riferimenti	Caso d'uso UC4: Gestione delle pulizie
Pre-Condizioni	<ul style="list-style-type: none">L'applicazione SmartHair deve essere avviata.L'amministratore si è loggato
Post-Condizioni	<ul style="list-style-type: none">L'amministratore vede tutti i giorni disponibili a cui assegnare al dipendente desiderato il turno di pulizia

Contratto CO3: InserisciDati

Operazione	InserisciDati (NomeDipendente:string, Giorno:string)
Riferimenti	Caso d'uso UC4: Gestione delle pulizie
Pre-Condizioni	<ul style="list-style-type: none">L'applicazione SmartHair deve essere avviata.L'amministratore si è loggatoCi sono giorni disponibili per assegnare il turno di pulizia
Post-Condizioni	<ul style="list-style-type: none">L'amministratore assegna al dipendente desiderato il turno di pulizia

Contratti Operazioni UC5

Contratto CO1: DetrazioneCredito

Operazione	DetrazioneCredito (valoreServizio: double)
Riferimenti	Caso d'uso UC5: Gestione pagamento
Pre-Condizioni	<ul style="list-style-type: none">• L'applicazione SmartHair deve essere avviata.• Il cliente deve essere loggato• Il cliente ha effettuato una prenotazione
Post-Condizioni	<ul style="list-style-type: none">• Il Sistema ha detratto il valore del servizio dal credito della tessera del cliente.

Contratti Operazioni UC6

Contratto CO1: PresenzaOrariDisponibili

Operazione	PresenzaOrariDisponibili ()
Riferimenti	Caso d'uso UC6: Generazione consiglio
Pre-Condizioni	<ul style="list-style-type: none">• L'applicazione SmartHair deve essere avviata.• Il cliente deve essere loggato
Post-Condizioni	<ul style="list-style-type: none">• Il Sistema ha restituito se ci sono orari disponibili.

Contratto CO2: ControlloPresenzaPrenotazioniEprimoConsiglio

Operazione	ControlloPresenzaPrenotazioniEprimoConsiglio()
Riferimenti	Caso d'uso UC6: Generazione consiglio
Pre-Condizioni	<ul style="list-style-type: none">• L'applicazione SmartHair deve essere avviata.• Il cliente deve essere loggato
Post-Condizioni	<ul style="list-style-type: none">• Il Sistema ha verificato la presenza di vecchie prenotazioni(almeno una) del cliente, verifica che non sia stato generato ancora un Consiglio appena si è loggato e il Sistema ha generato il servizio che consiglia al cliente per la sua prossima prenotazione.

Contratto CO3: GeneraConsiglio

Operazione	GeneraConsiglio (servizio:string)
Riferimenti	Caso d'uso UC6: Generazione consiglio
Pre-Condizioni	<ul style="list-style-type: none"> • L'applicazione SmartHair deve essere avviata. • Il cliente deve essere loggato • Il Sistema ha generato un Consiglio
Post-Condizioni	<ul style="list-style-type: none"> • Il cliente ha accettato il Consiglio e di conseguenza viene settato il servizio da eseguire nella prossima prenotazione.

Contratto CO4: PrelevaOrariDisponibili

Operazione	PrelevaOrariDisponibili()
Riferimenti	Caso d'uso UC6: Generazione consiglio
Pre-Condizioni	<ul style="list-style-type: none"> • L'applicazione SmartHair deve essere avviata. • Il cliente deve essere loggato • Il cliente ha accettato il consiglio
Post-Condizioni	<ul style="list-style-type: none"> • Il Sistema mostra tutti gli orari disponibili per una prenotazione.

Contratto CO5: AggiungiPrenotazione

Operazione	AggiungiPrenotazione (giorno:string, ora:string, nomeDip:string)
Riferimenti	Caso d'uso UC6: Generazione consiglio
Pre-Condizioni	<ul style="list-style-type: none"> • L'applicazione SmartHair deve essere avviata. • Il cliente deve essere loggato • Il cliente ha accettato il consiglio
Post-Condizioni	<ul style="list-style-type: none"> • Il Sistema ha effettuato la prenotazione con successo.

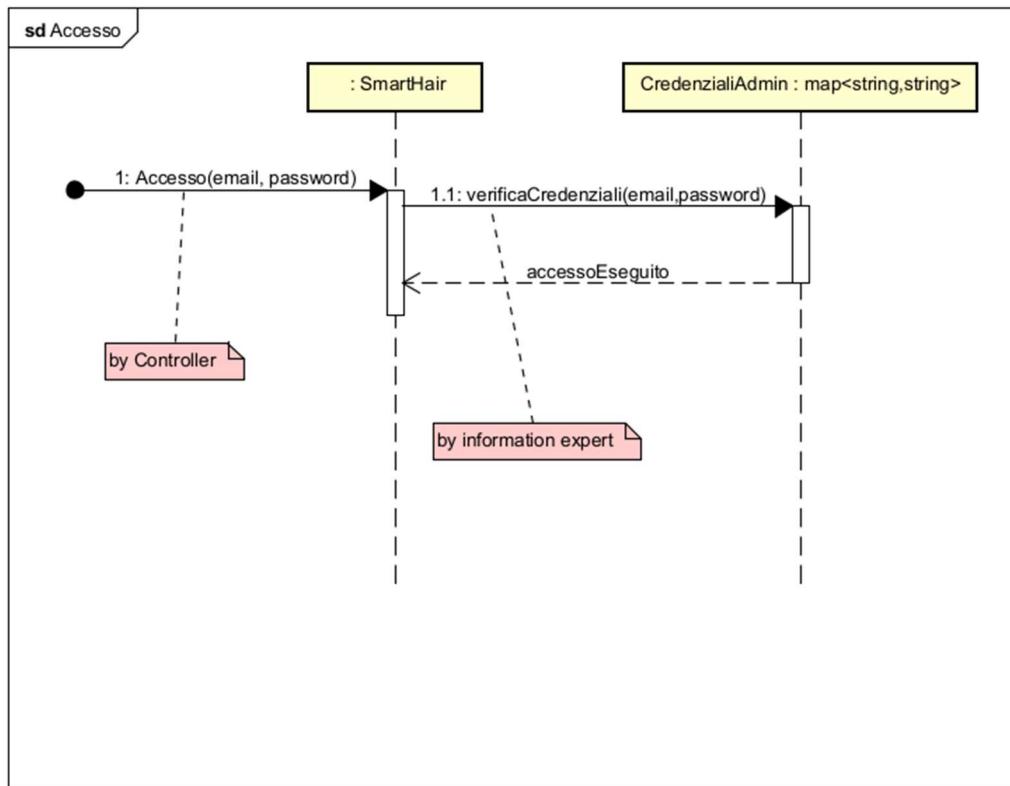
Progettazione

La progettazione orientata agli oggetti è la disciplina interessata alla definizione degli oggetti software, delle loro responsabilità e a come questi collaborano per soddisfare i requisiti individuati nei passi precedenti. L'elaborato principale di questa fase che è stato preso in considerazione è il Modello di Progetto, ovvero l'insieme dei diagrammi che descrivono la progettazione logica sia da un punto di vista dinamico (Diagrammi di Interazione) che da un punto di vista statico (Diagramma delle Classi). Seguono dunque i diagrammi di Interazione più significativi e il diagramma delle Classi.

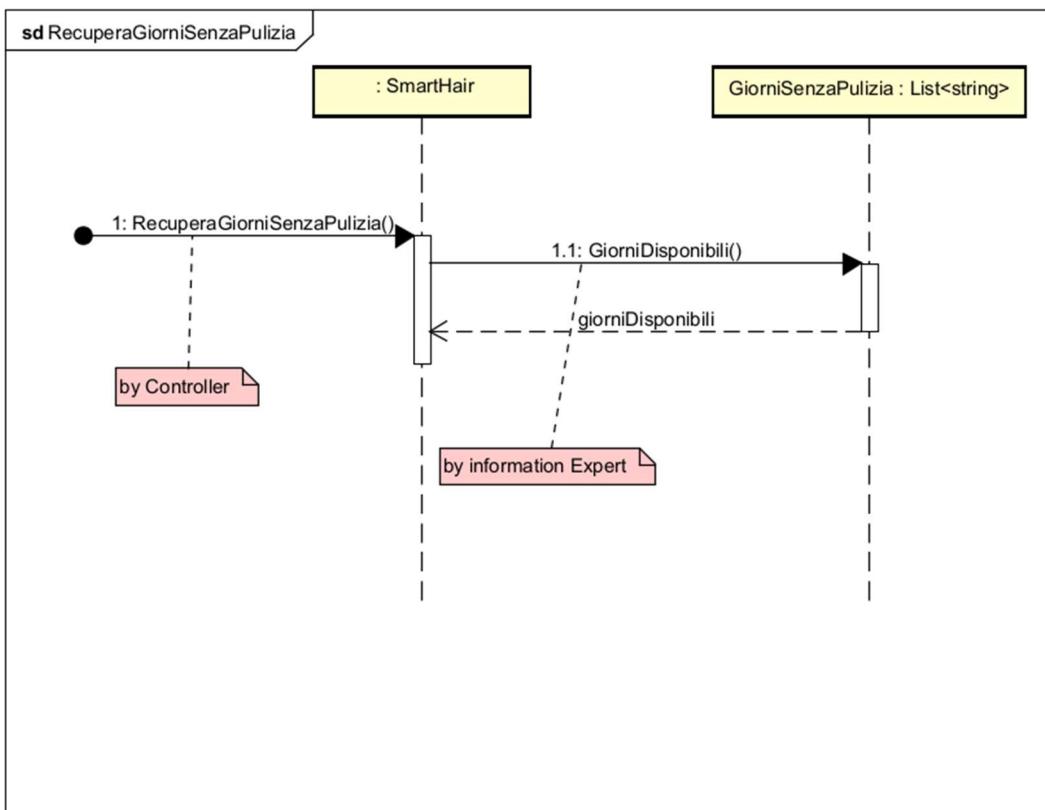
Diagrammi di Sequenza

UC4

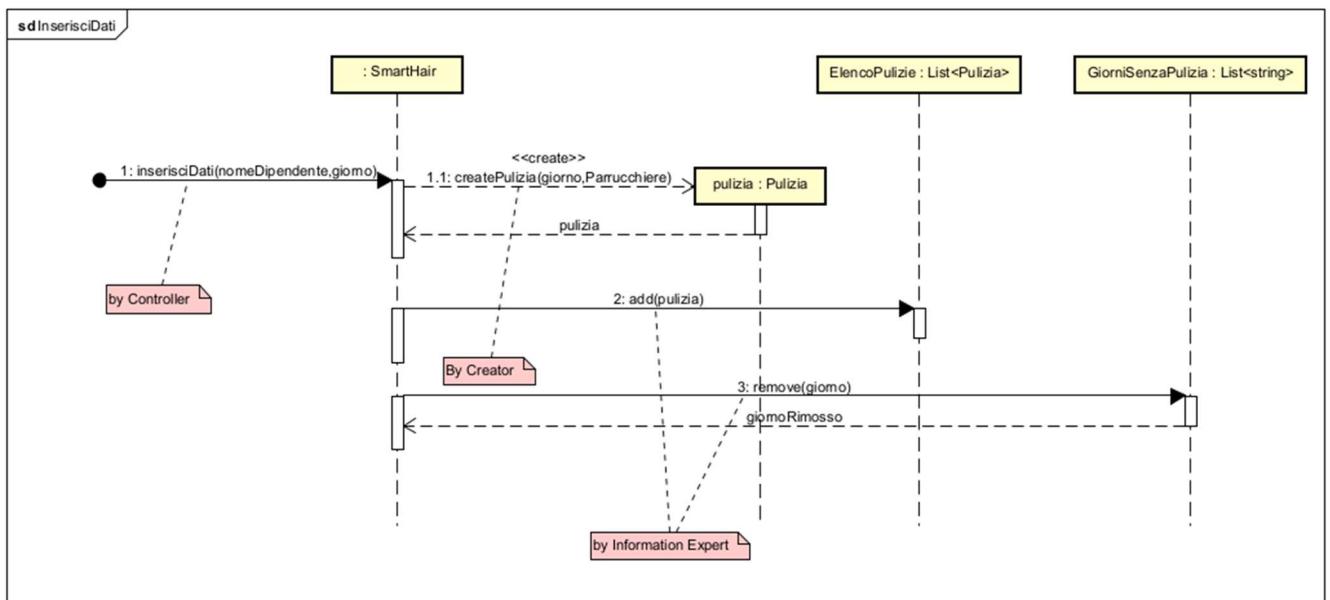
- Accesso



• RecuperaGiorniSenzaPulizia

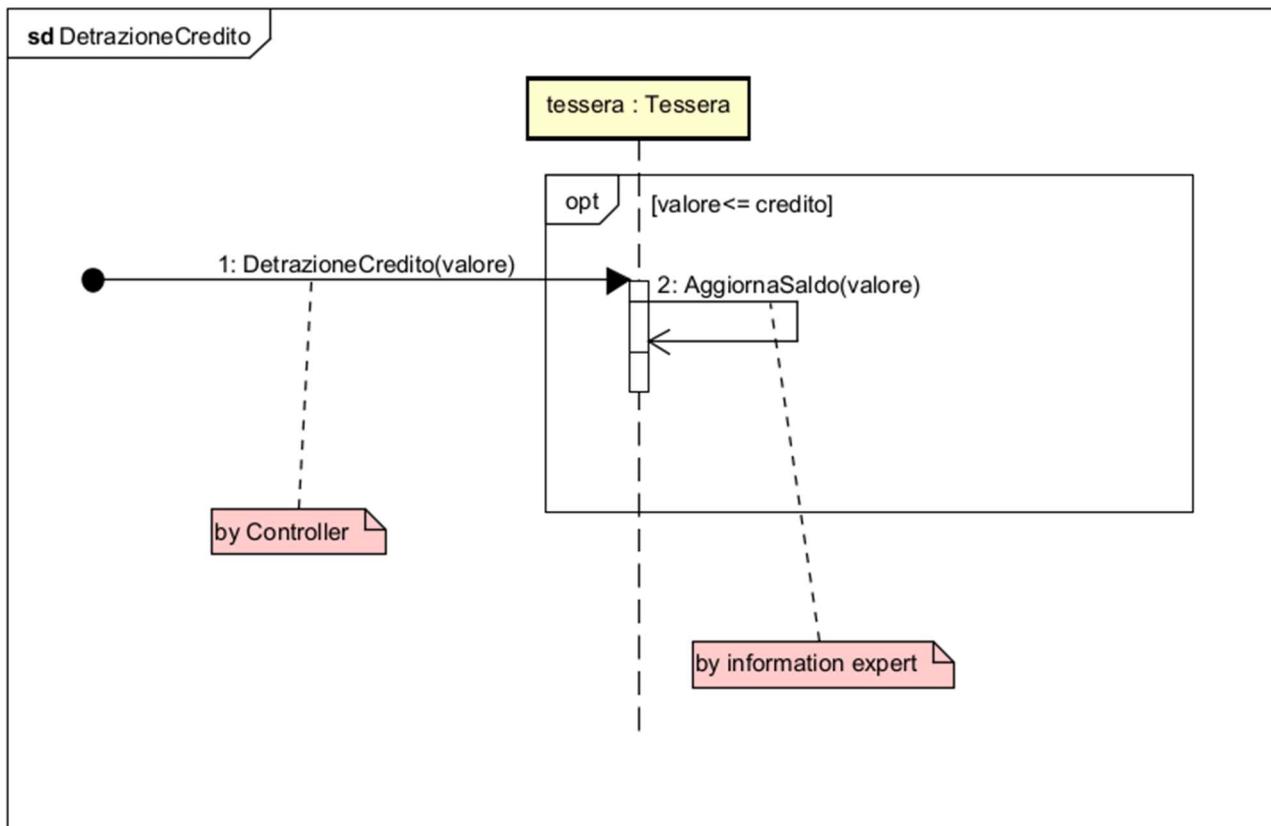


• InserisciDati



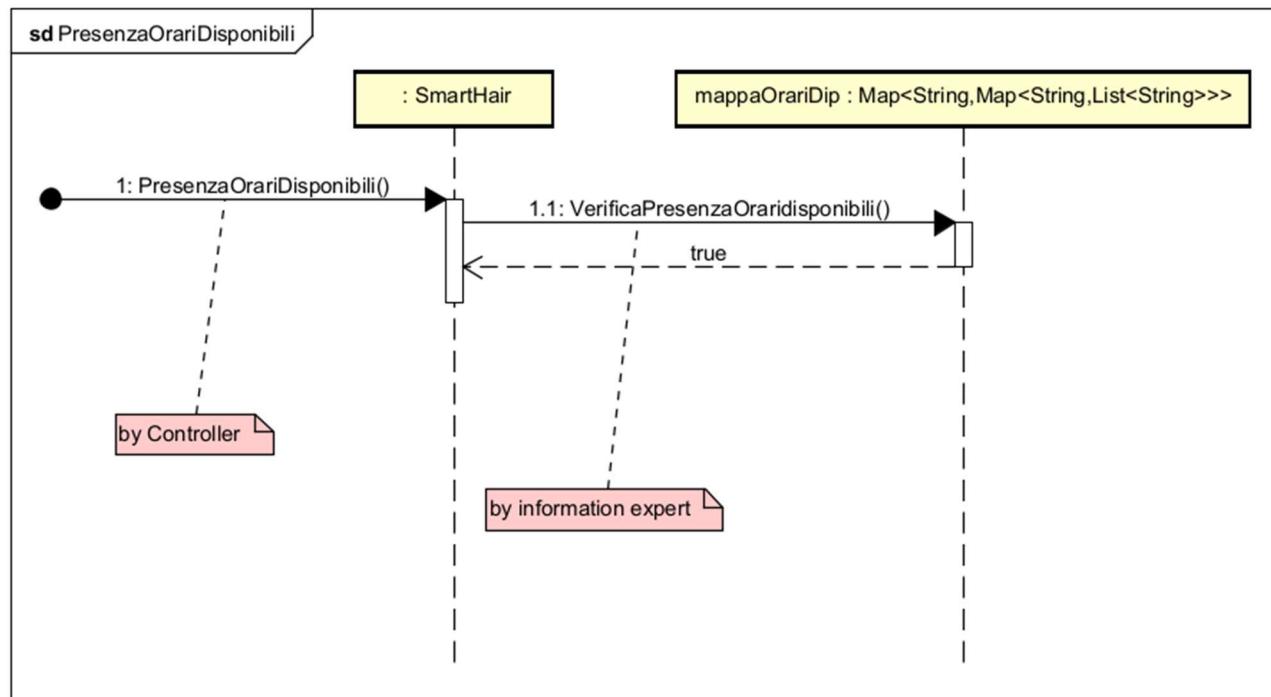
UC5

- DetrazioneCredito

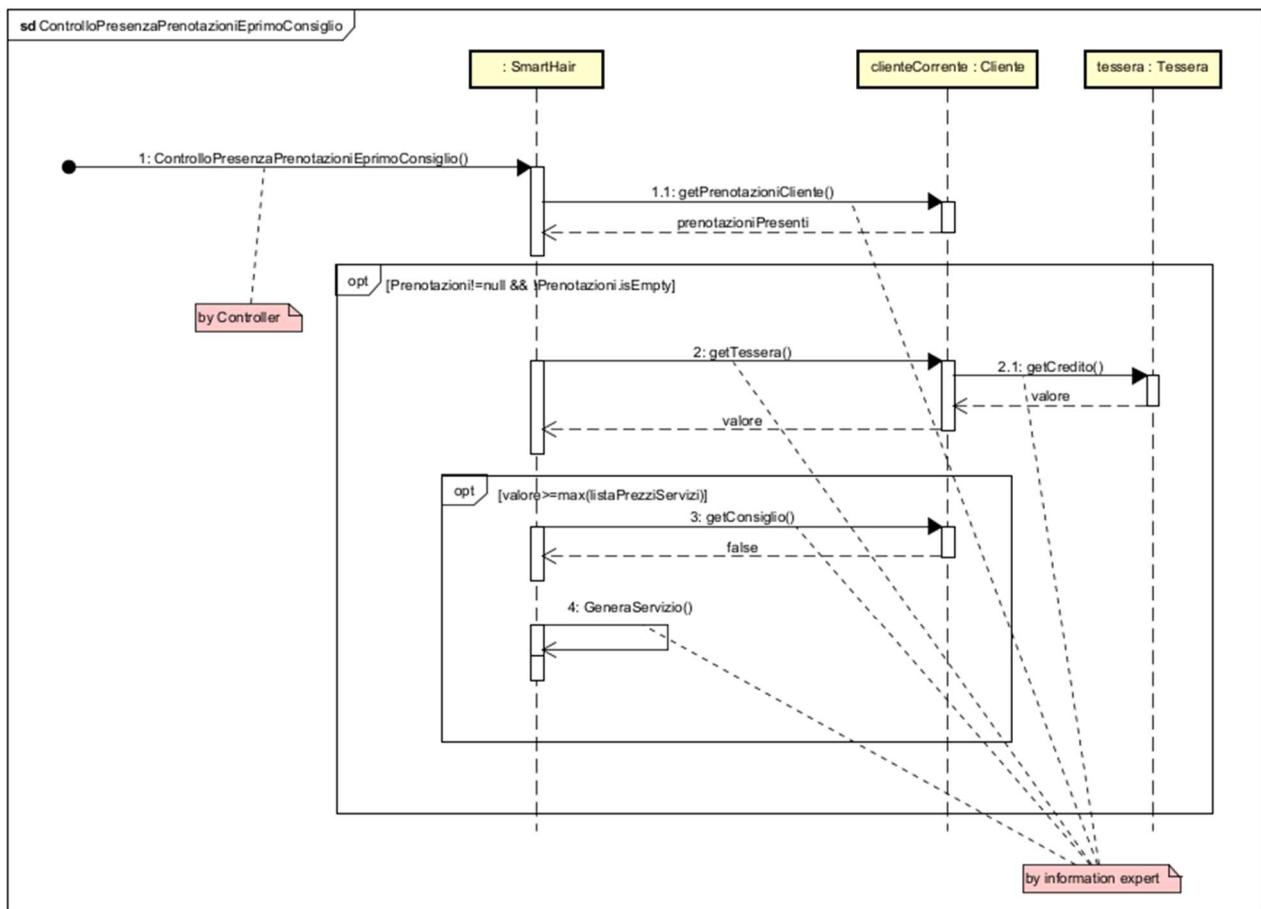


UC6

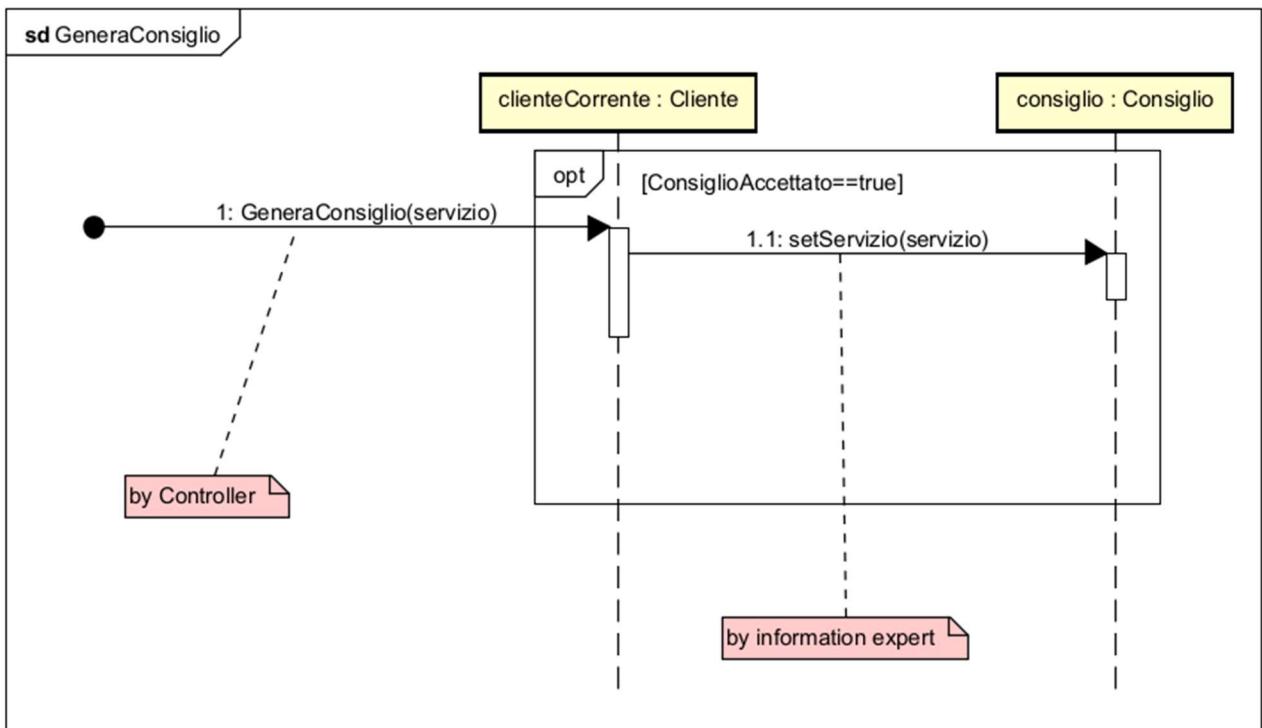
- PresenzaOrariDisponibili



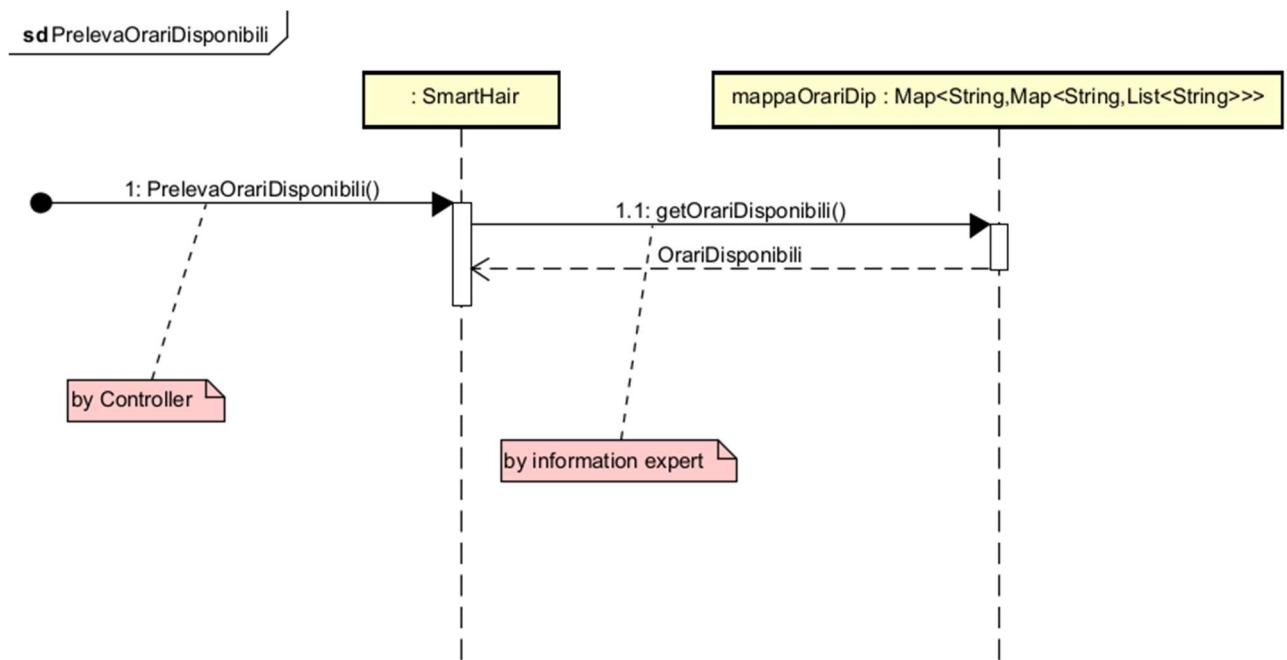
• ControlloPresenzaPrenotazioniEPrimoConsiglio



• GeneraConsiglio



• PrelevaOrariDisponibili



• AggiungiPrenotazione

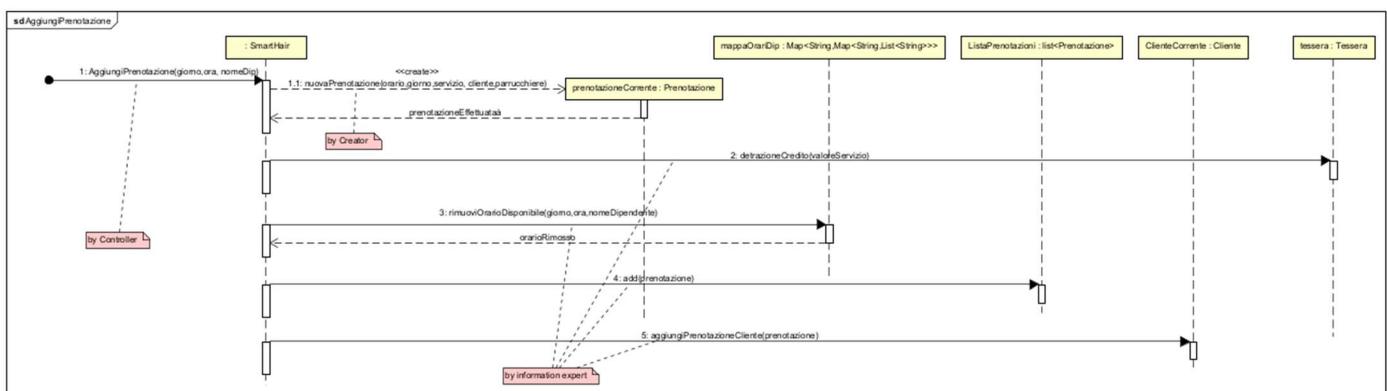
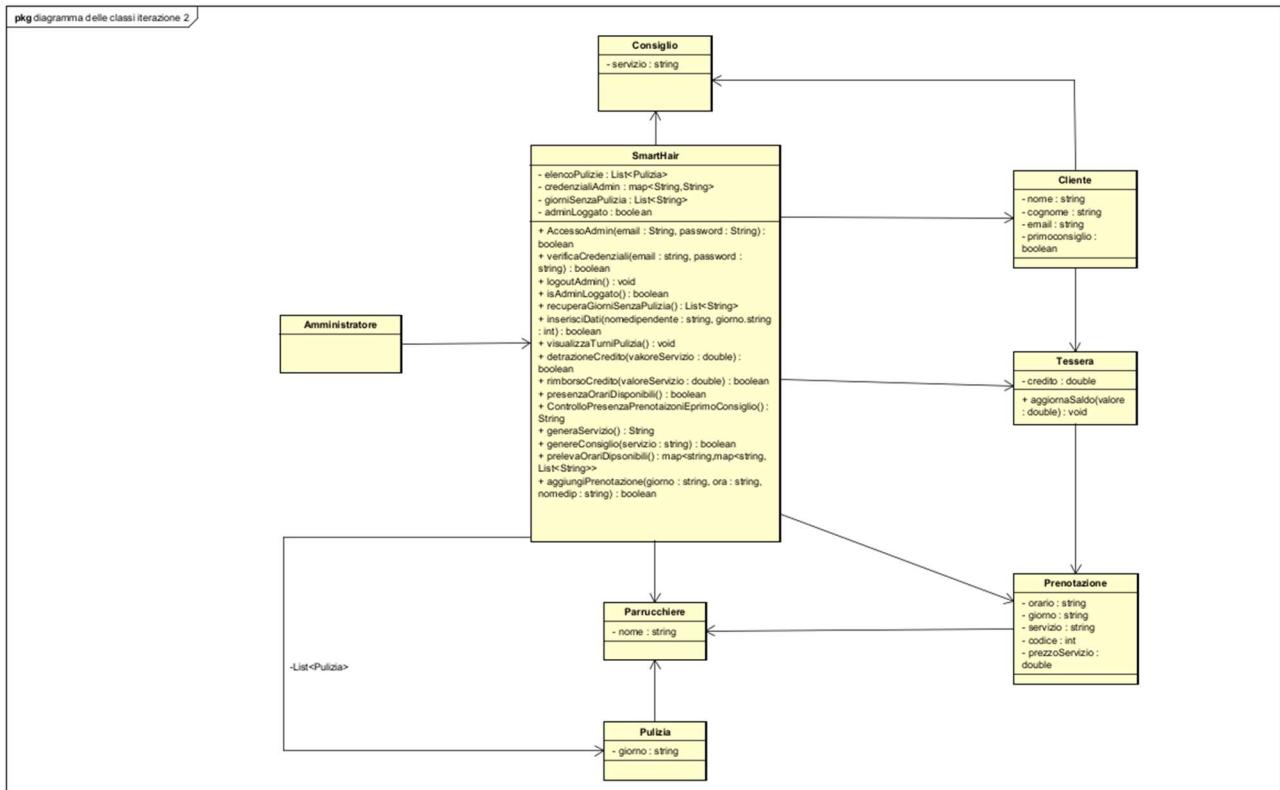


Diagramma delle classi



Test

Abbiamo realizzato i test del progetto SmartHair utilizzando il framework JUnit, che consente di automatizzare la verifica del corretto funzionamento delle classi e dei metodi sviluppati. L'approccio seguito è di tipo white-box testing, poiché i test sono stati progettati conoscendo la struttura interna del codice e verificando che ogni metodo produca i risultati attesi in base alla logica implementata.

Individuazione dei casi di test:

SmartHairTest:

-setup():

ottiene l'istanza singleton con `SmartHair.getInstance()`, svuota la lista interna dei giorni senza pulizia tramite `recuperaGiorniSenzaPulizia().clear()` e la ripopola con `["martedì", "mercoledì", "giovedì", "venerdì", "sabato"]`, così ogni test parte da uno stato noto in cui tutti quei giorni risultano disponibili per l'assegnazione delle pulizie.

- testSingleton():

chiama due volte `SmartHair.getInstance()` assegnando a a e b e verifica con `assertSame(a,b)` che puntino alla stessa istanza, confermando che SmartHair implementa correttamente il pattern Singleton (un'unica istanza condivisa).

- **testAccessoAdminCorretto():**

invoca sistema.AccessoAdmin("admin@gmail.com","admin1234") aspettandosi true e poi verifica che sistema.isAdminLoggato() sia true, infine chiama logoutAdmin() per ripulire lo stato; il test conferma che le credenziali corrette autenticano e impostano il flag di sessione admin.

-**testAccessoAdminEmailErrata():**

invoca sistema.AccessoAdmin("fake@gmail.com","admin1234") e si aspetta false, poi verifica che sistema.isAdminLoggato() sia false, mostrando che un'email errata impedisce login e non modifica lo stato di autenticazione.

-**testLogoutAdmin():**

effettua prima un login con credenziali valide, chiama sistema.logoutAdmin() e controlla assertFalse(sistema.isAdminLoggato()), accertando che il logout resetti correttamente lo stato dell'admin loggato.

-**testInserisciDatiValido():**

crea Parrucchiere("Luca"), mediante reflection imposta il campo privato elencoParrucchieri di SmartHair a una lista contenente "Luca" e imposta giorniSenzaPulizia a ["martedì", "mercoledì", "giovedì", "venerdì", "sabato"], quindi chiama sistema.inserisciDati("Luca","martedì") e si aspetta true, verificando che un parrucchiere esistente e un giorno disponibile producano un inserimento valido del turno di pulizia.

-**testInserisciDatiParrucchierenesistente():**

chiama sistema.inserisciDati("Fake","mercoledì") e si aspetta false, controllando che la funzione fallisca quando il nome indicato non è presente nell'elenco parrucchieri gestito dal sistema.

- **testInserisciDatiGiornoNonDisponibile():**

crea Parrucchiere("Anna") (senza aggiungerla al sistema), rimuove "sabato" da recuperagiorniSenzaPulizia() per simulare indisponibilità del giorno, invoca sistema.inserisciDati("Anna","sabato") e si aspetta false, verificando che una richiesta su un giorno non disponibile venga rifiutata indipendentemente dal parrucchiere.

- **testDetrazioneCredito():**

crea Cliente("Mario","Rossi","mario@rossi.it"), ricarica la tessera a 50.0 con aggiornaSaldo, poi via reflection imposta clienteCorrente del sistema su quel cliente, invoca sistema.detrazioneCredito(20.0) aspettandosi true e verifica che il credito residuo sia 30.0, confermando che la detrazione avviene quando il saldo è sufficiente e che il saldo viene aggiornato correttamente.

-**testDetrazioneCreditoinSufficiente():**

crea Cliente("Luca","Bianchi","luca@gmail.com"), imposta saldo tessera a 10.0, setta via reflection clienteCorrente, invoca sistema.detrazioneCredito(40.0) aspettandosi false e verifica che il credito resti 10.0, accertando che la funzione rifiuti l'addebito e non modifichi il saldo se il credito è insufficiente.

-testRimborsoCredito():

crea Cliente("Anna","Verdi","anna@verdi.it") con saldo iniziale 30.0, imposta via reflection clienteCorrente, invoca sistema.rimborsoCredito(20.0) aspettandosi true e verifica saldo 50.0, mostrando che il rimborso con importo positivo viene accettato e accreditato.

-testRimborsoCreditoNegativo():

invoca sistema.rimborsoCredito(-10.0) senza settare un cliente specifico e si aspetta false, verificando la validazione che rifiuta importi negativi per rimborso.

-testPresenzaOrariDisponibiliVuota():

via reflection imposta il campo privato mappaOrariDip a una new HashMap<>() vuota, chiama sistema.presenzaOrariDisponibili() e si aspetta false, confermando che senza disponibilità memorizzate la funzione segnala assenza di orari.

-testPresenzaOrariDisponibiliPiena():

costruisce una mappa annidata con chiave parrucchiere "Luca" che ha per "martedì" gli orari ["9","10"], setta via reflection mappaOrariDip con tale struttura e verifica che sistema.presenzaOrariDisponibili() ritorni true, dimostrando che la presenza di almeno un orario disponibile viene rilevata.

-testGeneraServizioDaUltimaPrenotazione():

crea Cliente("Mario",...) e Parrucchiere("Luca"), aggiunge al cliente una Prenotazione("10","venerdì","taglio",c,p,30.0), imposta via reflection clienteCorrente a quel cliente e chiama sistema.generaServizio(), verificando che il risultato non sia null e che appartenga all'insieme {"piega","colore","taglio"}, attestando che il sistema produce un consiglio coerente con lo storico (almeno nell'insieme previsto).

AmministratoreTest:

-testGetters():

Crea un oggetto Amministratore con username "admin" e password "1234" e verifica che i metodi getter (getUsername() e getPassword()) restituiscano i valori corretti.

- testVerificaCredenzialiCorrette():

Verifica che, passando le stesse credenziali memorizzate nell'oggetto, il metodo verificaCredenziali() ritorni true

- testVerificaCredenzialiUsernameErrato():

Verifica che, se si inserisce un username sbagliato, il metodo restituisca false

- testVerificaCredenzialiPasswordErrata():

Verifica che, con password sbagliata, il metodo ritorni false

- testVerificaCredenzialiEntrambeErrate():

Passa credenziali completamente errate (sia username che password). Si aspetta che il risultato sia false.

ClienteTest:

-testCostruttoreEGetter():

Controlla che il costruttore di Cliente verifichi:

- nome, cognome, email
- la tessera non sia null
- il credito iniziale sia 0.0
- la lista di prenotazioni sia vuota all'inizio
- isPrimoConsiglio sia false di default

In pratica verifica che la creazione dell'oggetto Cliente sia corretta.

- testSetEGetConsiglio():

Crea un Consiglio, imposta servizio = "taglio". andiamo a fare cliente.setConsiglio(c) e poi cliente.getConsiglio().getServizio(). Quindi si aspetta "taglio". Quindi implica che il cliente mantenga un riferimento a un Consiglio impostabile dall'esterno. getConsiglio() non deve essere null dopo il set.

- testSetPrimoConsiglio():

Controlla che isPrimoConsiglio() sia false inizialmente. Lo imposta a true con setPrimoConsiglio(true). Verifica che poi sia true.

ConsiglioTest:

- setUp():

viene eseguito prima di ogni test e crea un'istanza di un nuovo oggetto Consiglio() per garantire che ogni test parta da un'istanza pulita e indipendente.

- testCostruttoreVuoto():

verifica che, creando un oggetto con il costruttore vuoto new Consiglio(), il valore del campo servizio sia null. Utilizza assertNull(consiglio.getServizio()) per controllare che il costruttore non assegni alcun valore di default.

- testCostruttoreConParametroValido():

crea un oggetto Consiglio passando una stringa valida "Taglio" al costruttore e verifica con assertEquals("Taglio", c.getServizio()) che il campo servizio venga inizializzato correttamente.

- testSetServizioValido():

chiama consiglio.setServizio("Piega") e verifica con assertEquals("Piega", consiglio.getServizio()) che il metodo setter aggiorni correttamente il valore del campo servizio.

ParrucchiereTest:

-testCostruttoreValido():

crea new Parrucchiere("Vincenzo"), verifica che l'istanza non sia null (assertNotNull) e che getNome() restituisca esattamente "Vincenzo" (assertEquals), confermando inizializzazione e mapping del costruttore al campo.

PrenotazioneTest:

-testCostruttoreAssegnaDatiCorrettamente():

il costruttore di Prenotazione assegna correttamente tutti i campi (orario, giorno, servizio, cliente, parrucchiere, prezzoServizio).

- testCostruttoreOrarioNull():

accerta la validazione sull'argomento "orario" aspettandosi NullPointerException quando si invoca new Prenotazione(null,"martedì","taglio", cliente, parrucchiere, 25.0), quindi il costruttore non consente orario nullo e deve fallire immediatamente.

- testCostruttoreGiornoNull():

verifica che il costruttore rifiuti un giorno nullo lanciando NullPointerException all'invocazione new Prenotazione("10", null, "taglio", cliente, parrucchiere, 25.0), garantendo la non-nullability del campo giorno.

- testCostruttoreServizioNull():

controlla che passare servizio uguale a null a new Prenotazione("10","martedì",null, cliente, parrucchiere, 25.0) provochi NullPointerException, quindi il tipo di servizio è obbligatorio e validato in costruzione.

-testCostruttoreClienteNull():

verifica che il riferimento al cliente sia obbligatorio richiedendo NullPointerException quando si costruisce con cliente nullo (new Prenotazione("10","martedì","taglio", null, parrucchiere, 25.0)), impedendo prenotazioni senza titolare.

- testGetters():

crea Prenotazione p = new Prenotazione("11","venerdì","colore", cliente, parrucchiere, 40.0) e usa assertAll(...) per raggruppare asserzioni indipendenti su tutti i getter (orario, giorno, servizio, cliente, parrucchiere, prezzo), così se più valori sono errati li si vede tutti in un'unica esecuzione del test, confermando il corretto mapping costruttore → stato interno → getter.

- testCodiciDiversi():

istanzia due prenotazioni con dati diversi (p1 e p2) e verifica con assertEquals(p1.getCodice(), p2.getCodice()) che il campo codice sia un identificativo univoco generato per istanza (non derivato banalmente dagli input), requisito utile per distinguere prenotazioni anche con campi simili.

PuliziaTest:

- testCostruttoreValido():

crea un Parrucchiere("Luca"), istanzia Pulizia("venerdì", p) e verifica con due assertEquals che i campi siano mappati correttamente dal costruttore (giorno uguale a "venerdì" e assegnato uguale all'istanza p), confermando inizializzazione e associazione corretta al parrucchiere.

- testGettersValoriCorrettamenteRestituiti():

crea Parrucchiere("Giulia"), istanzia Pulizia("sabato", p) e controlla con assertEquals che getGiorno() restituisca "sabato" e che getAssegnato().getNome() restituisca "Giulia", validando che i getter espongano fedelmente i valori impostati in costruzione.

Pattern GoF usati:

anche in questa iterazione abbiamo fatto uso dei due pattern GoF usati nella prima. Il primo è il pattern Singleton mentre il secondo è il pattern Facade.