InViaggio: Elaborazione - Iterazione 2

G. Messina – S. Squillaci – A. Zarbo

Introduzione

Per l'iterazione 2 sono stati scelti i seguenti requisiti:

- Implementazione del caso d'uso UC3: Creazione corsa.
- Implementazione del caso d'uso UC5: Gestisci cliente.
- Implementazione del caso d'uso UC8: Annulla biglietto.
- Implementazione degli scenari alternativi dei casi d'uso UC2 e UC6 sviluppati nell'iterazione 1.
- Vengono considerate le regole di dominio R1ed R4 nel UC6 (nell'iterazione 1 non erano state applicate), e la regola di dominio R3 nell'UC8.
- Implementazione del caso d'uso d'avviamento necessario per inizializzare questa iterazione.

Per l'iterazione 2 è stata effettuata l'ipotesi che l'amministratore è già loggato per poter effettuare le operazioni da amministratore.

Aggiornamenti elaborati della fase di Ideazione

Per quanto riguarda i casi d'uso UC2 ed UC3 sono state apportate alcune modifiche/correzioni nelle estensioni. Di seguito sono riportate le estensioni aggiornate:

UC2: Creazione nuova tratta

Estensioni	***a.** In un qualsiasi momento il Sistema fallisce e si arresta improvvisamente. 1. L'Amministratore riavvia il software e ripristina lo stato precedente del Sistema; 2. 2. Il Sistema ripristina lo stato. **4a.** L'Amministratore inserisce una tratta già con le città di partenza e arrivo già presenti. 1. Il Sistema genera un messaggio di errore; 2. L'Amministratore ripete il passo 3 cambiando le città. **4b.** L'amministratore inserisce città di partenza e città di arrivo differenti, ma precedentemente aveva indicato tratta urbana (città di partenza e di arrivo devono essere uguali), oppure l'amministratore inserisce la stessa città sia in città di partenza che città di arrivo ma precedentemente aveva indicato tratta extraurbana (le città di partenza e arrivo devono essere differenti).

	1. Il Sistema genera un messaggio di errore; **6a.** L'Amministratore inserisce una corsa per una tratta in una data in cui esiste già una corsa con gli stessi orari di partenza, arrivo e luogo di partenza, arrivo.

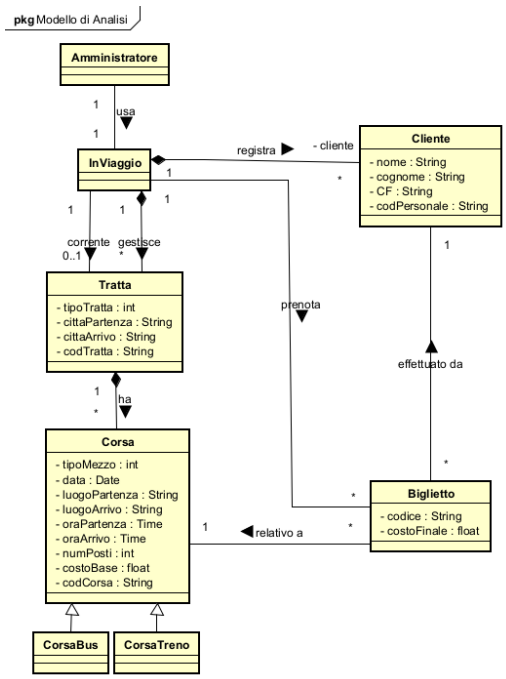
UC3: Creazione corsa

Estensioni	***a.** In un qualsiasi momento il Sistema fallisce e si arresta improvvisamente. 1. L'Amministratore riavvia il software e ripristina lo stato precedente del Sistema 2. Il Sistema ripristina lo stato **5a.** L'Amministratore inserisce una corsa per una tratta in una data in cui esiste già una corsa con gli stessi orari di partenza, arrivo e luogo di partenza, arrivo. 1. Il Sistema genera un messaggio di errore 2. L'Amministratore ripete il passo 5 cambiando le città.

1. Analisi Orientata agli Oggetti

Per descrivere il dominio da un punto di vista ad oggetti per gestire i nuovi requisiti sono stati utilizzati gli stessi strumenti dell'iterazione precedente, ossia modello di dominio, SSD, Diagrammi di sequenza e contratti delle operazioni.

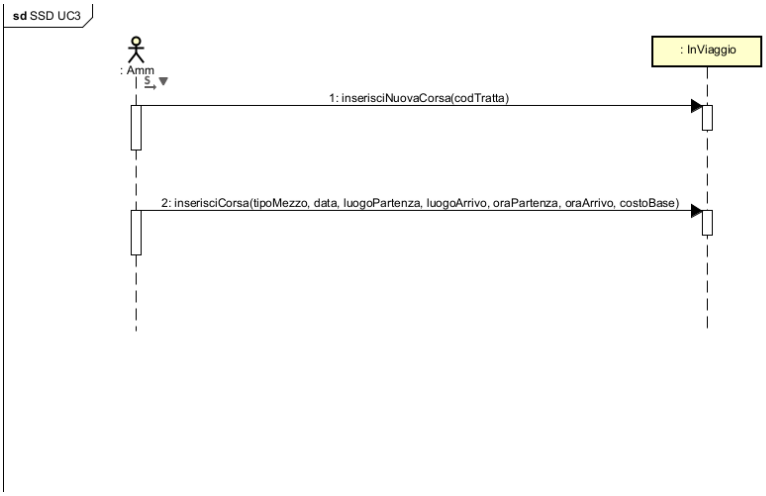
1.1. Modello di dominio



1.2. Diagrammi di Sequenza di Sistema (SSD)

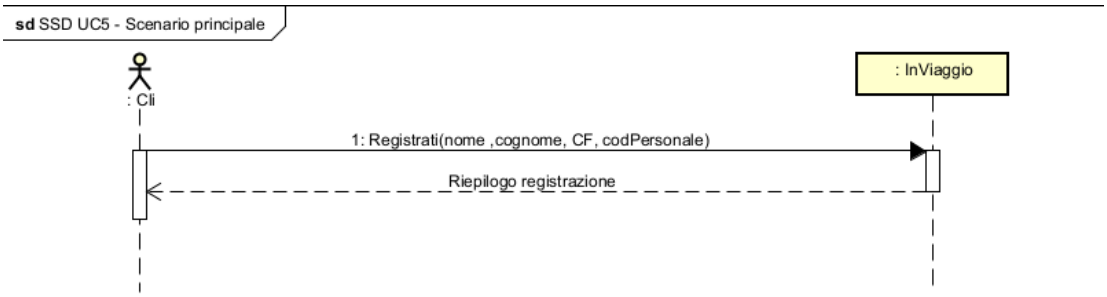
Di seguito sono stati riportati i diagrammi di sequenza relativi ai casi d'uso ed estensioni presi in analisi in questa iterazione.

1.2.1 SSD UC3

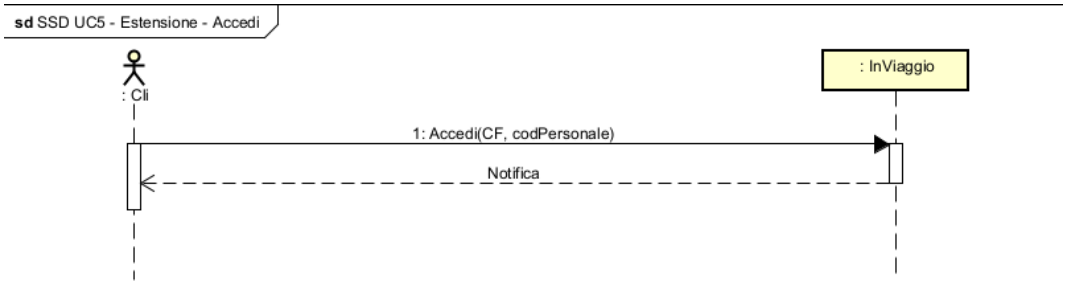


1.2.2 SSD UC5

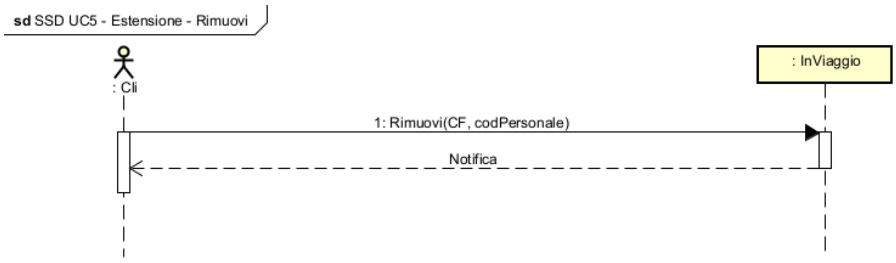
1.2.2.1 Scenario principale (Registrazione)



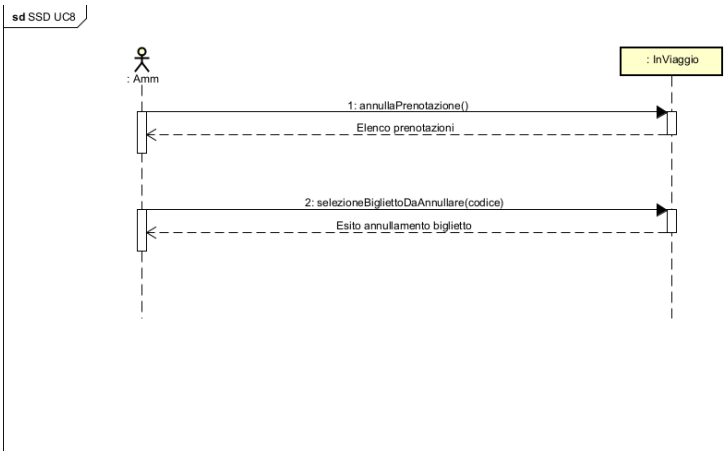
1.2.2.2 Accedi (Estensione)



1.2.2.3 Rimuovi (Estensione)



1.2.3 SSD UC8



1.3. Contratti delle Operazioni

Vengono ora descritte le operazioni svolte dal sistema nei caso d'uso UC3, UC5, UC8 attraverso i contratti:

1.3.1 UC3 inserisciCorsa

Operazione	inserisciCorsa(tipoMezzo:int,data Date,luogoPartenza:String,luogoArrivo:String,oraPartenza :Time, oraArrivo:Time, costoBase: float)
Riferimenti	Caso d'uso UC3: Creazione corsa
Pre-condizioni	È in corso la definizione di Corsa c
Post-condizioni	- È stata creata una nuova istanza c di Corsa: - Gli attributi di c sono stati inizializzati; - c è stata associata a t tramite l'associazione "ha";

1.3.2 UC5 registrati

Operazione	Registrati(nome: String, cognome: String, CF: String, codPeronale: String)
Riferimenti	Caso d'uso UC5: Gestisci cliente
Pre-condizioni	-
Post-condizioni	- È stata creata una nuova istanza cl della classe Cliente: - Gli attributi di cl sono stati inizializzati; - cl è stata associata a inViaggio tramite l'associazione "registra";

1.3.3 UC5 registrati - estensione

Operazione	Registrati(nome: String, cognome: String, CF: String, codPeronale: String)
Riferimenti	Caso d'uso UC5: Gestisci cliente – Estensione "rimuovi Account"
Pre-condizioni	- L'utente deve essere già registrato
Post-condizioni	- È stata recuperata l'istanza cl di Cliente tramite la relazione "registra" sulla base del CF; - L'istanza cl viene dissociata dal Cliente; - L'istanza cl viene eliminata;

1.3.4 UC8 annullaBiglietto

Operazione	selezionaBigliettoDaAnnullare(codice:String)
Riferimenti	Caso d'uso UC8: Annulla biglietto
Pre-condizioni	È stata recuperata l'istanza cl di cliente È stata recuperata l'istanza b di Biglietto
Post-condizioni	- b è stata dissociata dal Cliente cl; - L'istanza b viene eliminata;

1.4. Contratti delle Operazioni aggiornati

Di seguito sono riportati i contratti relativi al caso d'uso UC6 aggiornati considerando adesso le regole di dominio.

1.4.1 UC2 selezionaCorsa

Operazione	selezionaCorsa(codCorsa)
Riferimenti	Caso d'uso: Prenotazione corsa
Pre-condizioni	è stata recuperata l'istanza c della classe Corsa selezionata dall'utente
Post-condizioni	- È stata creata l'istanza b di Biglietto;

	- È stata associata c a b tramite l'associazione "relativo a"; - Viene inizializzato l'attributo costoFinale di b sulla base delle regole di dominio; - Viene inizializzato l'attributo codice di b;

2. Progettazione Orientata agli oggetti

In questa fase si procede con la definizione degli oggetti software a partire dagli oggetti concettuali individuati nella fase precedente. Inoltre, si definiscono anche le loro responsabilità e le loro interazioni al fine di soddisfare i requisiti individuati nei passi precedenti. A seguire sono riportati i diagrammi di sequenza (SD) ed il diagramma delle classi relativi ai casi d'uso UC3, UC5, UC6 ed UC8.

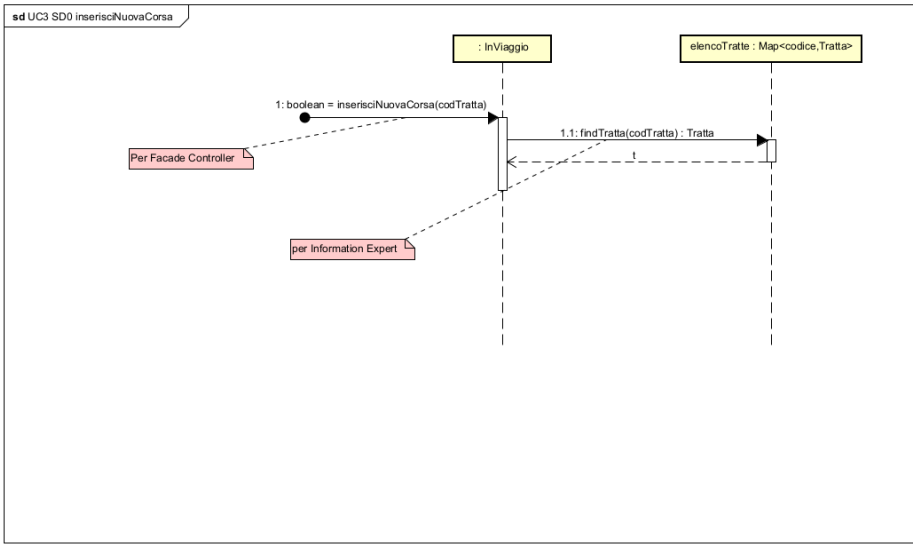
2.1 Pattern applicati

Oltre ai pattern GRASP (Creator, Information Expert, Controller, Low Coupling e High Cohesion) è stato usato il pattern GoF Strategy per applicare le regole di dominio per determinare i prezzi dei biglietti.

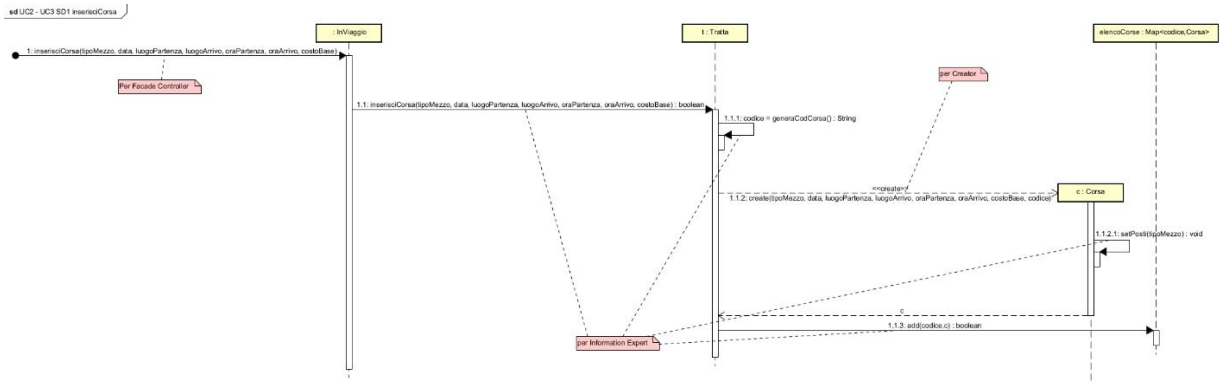
2.2 Diagrammi di sequenza (SD)

È stato scelto di continuare ad utilizzare come Facade Controller l'istanza di InViaggio, ossia la classe concettuale che astrae il sistema.

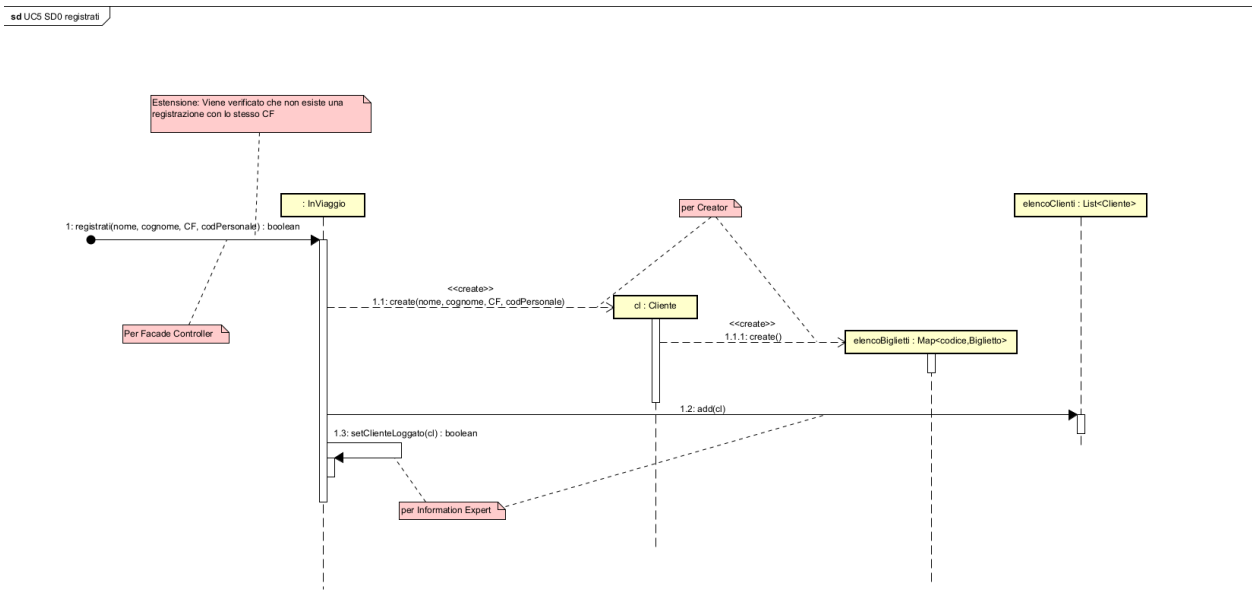
2.2.1 UC3-SD0 inserisciNuovaCorsa



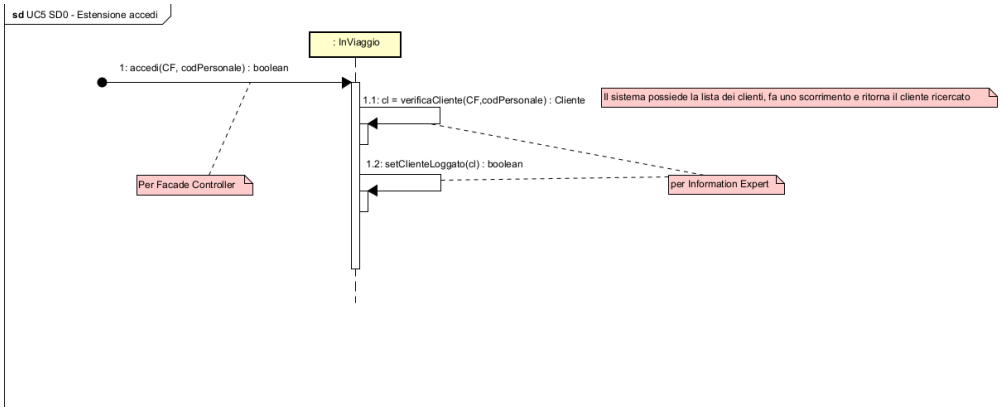
2.2.2 UC3-SD1 inserisciCorsa



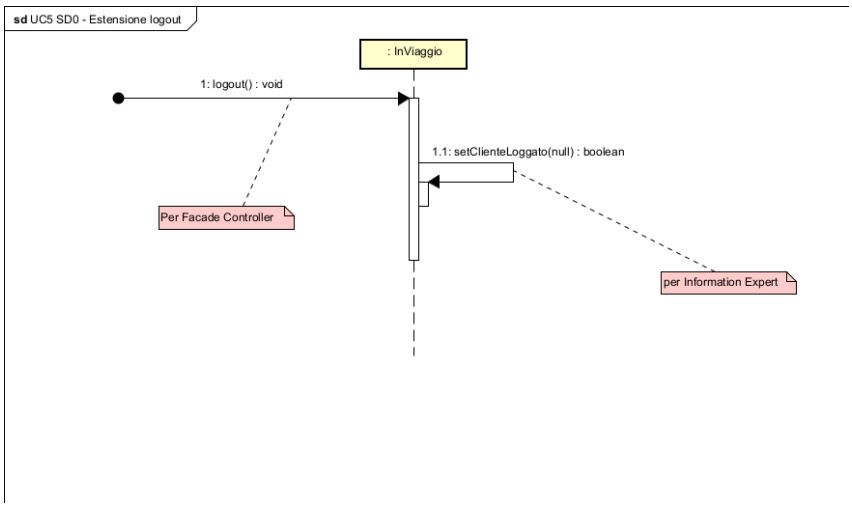
2.2.3 UC5-SD0 registrati (scenario principale)



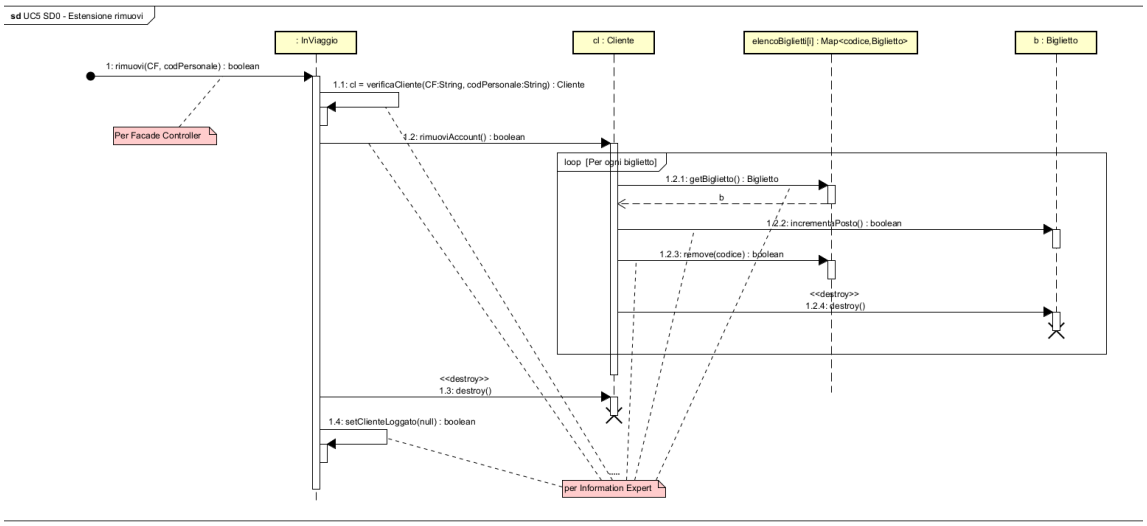
2.2.4 UC5-SD0 accedi (Estensione)



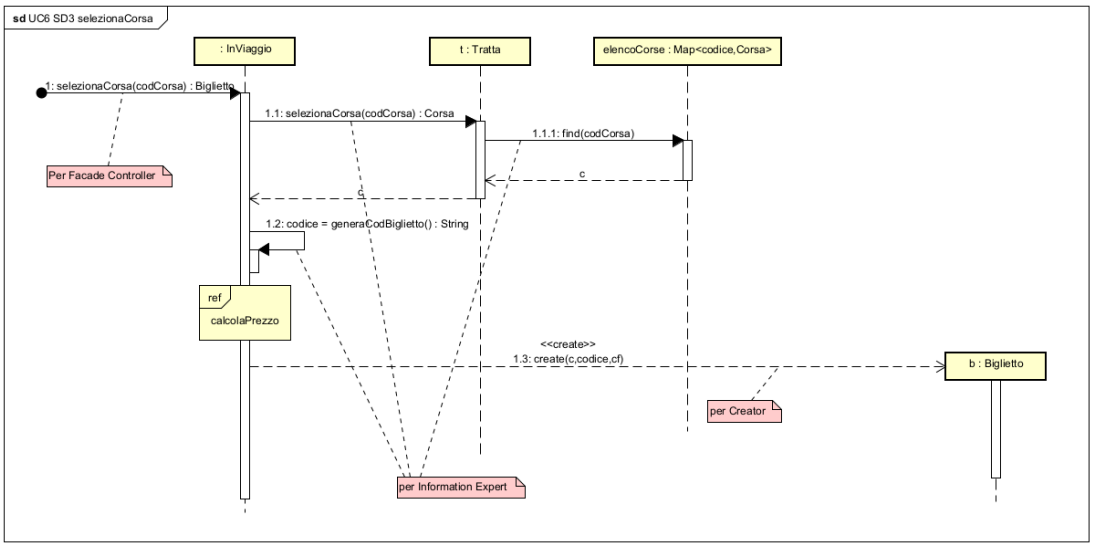
2.2.5 UC5-SD0 logout (Estensione)



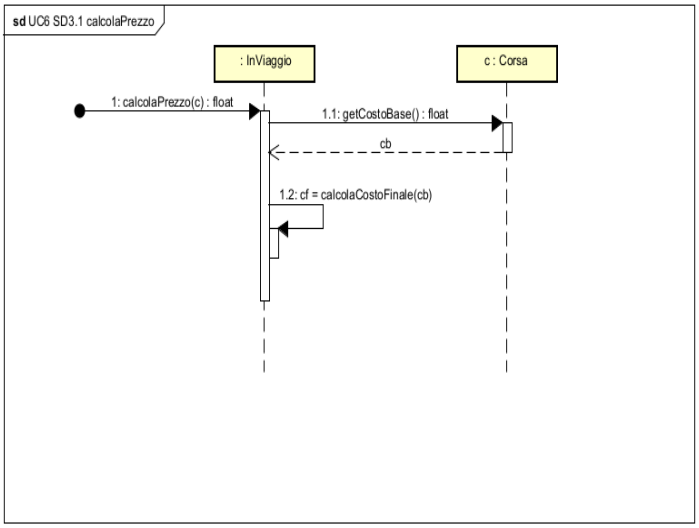
2.2.6 UC5-SD0 rimuovi (Estensione)



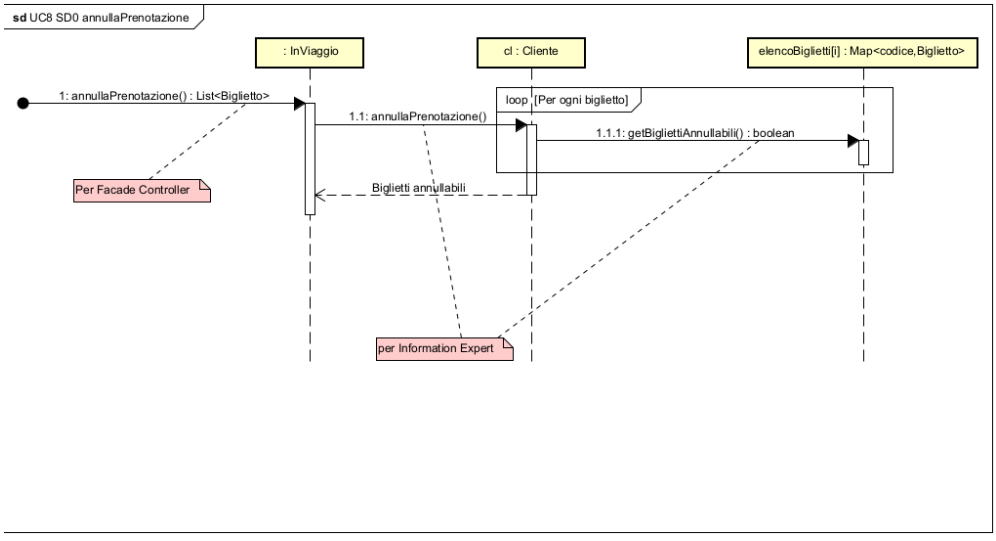
UC6-SD3 selezionaCorsa



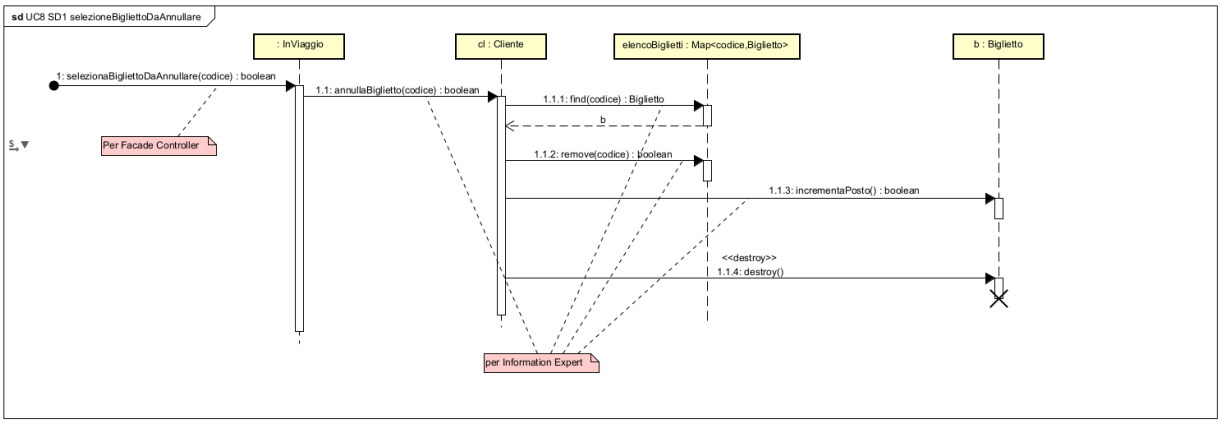
2.2.8 UC6-SD3.1 calcolaPrezzo



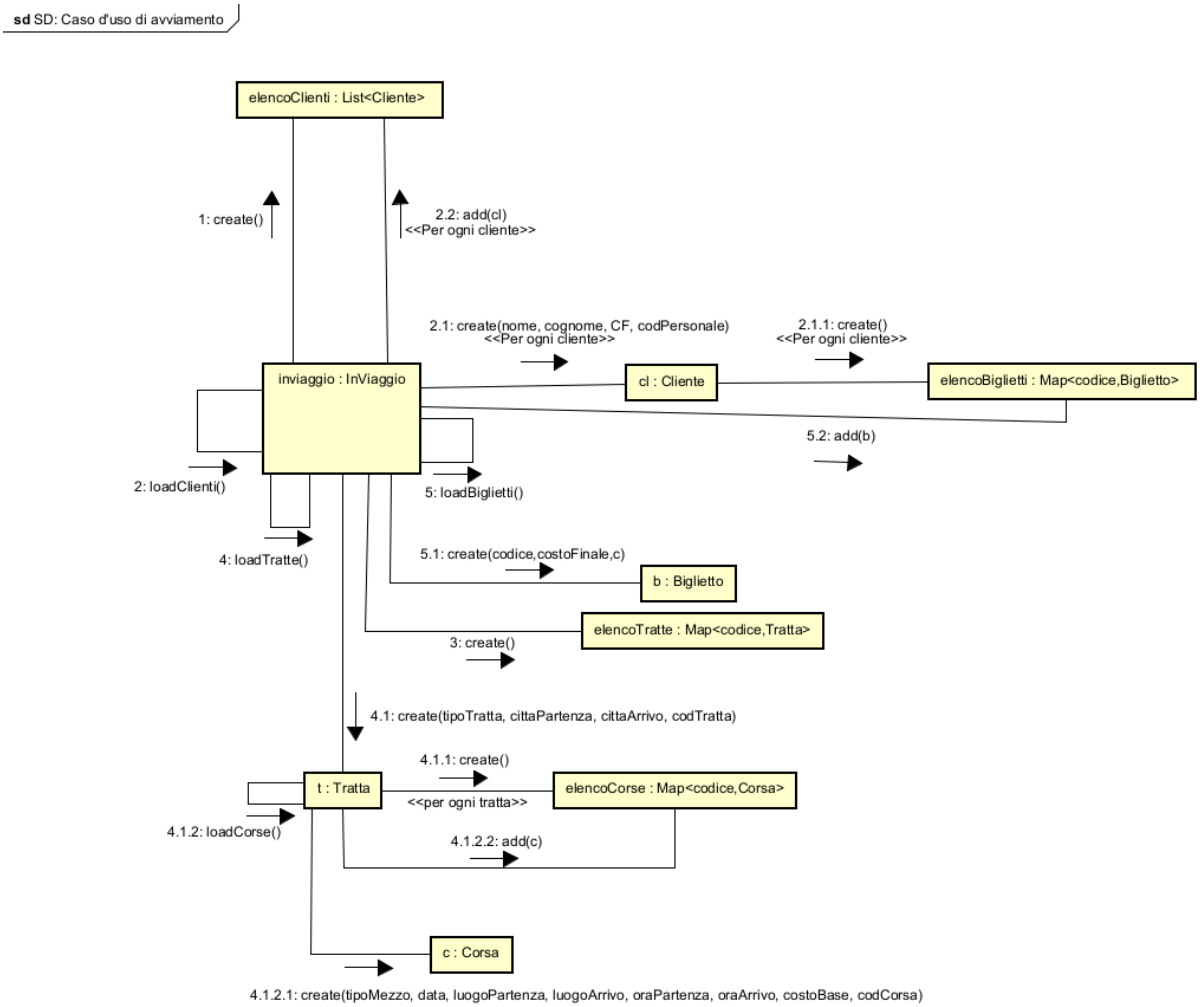
2.2.9 UC8-SD0 annullaPrenotazione



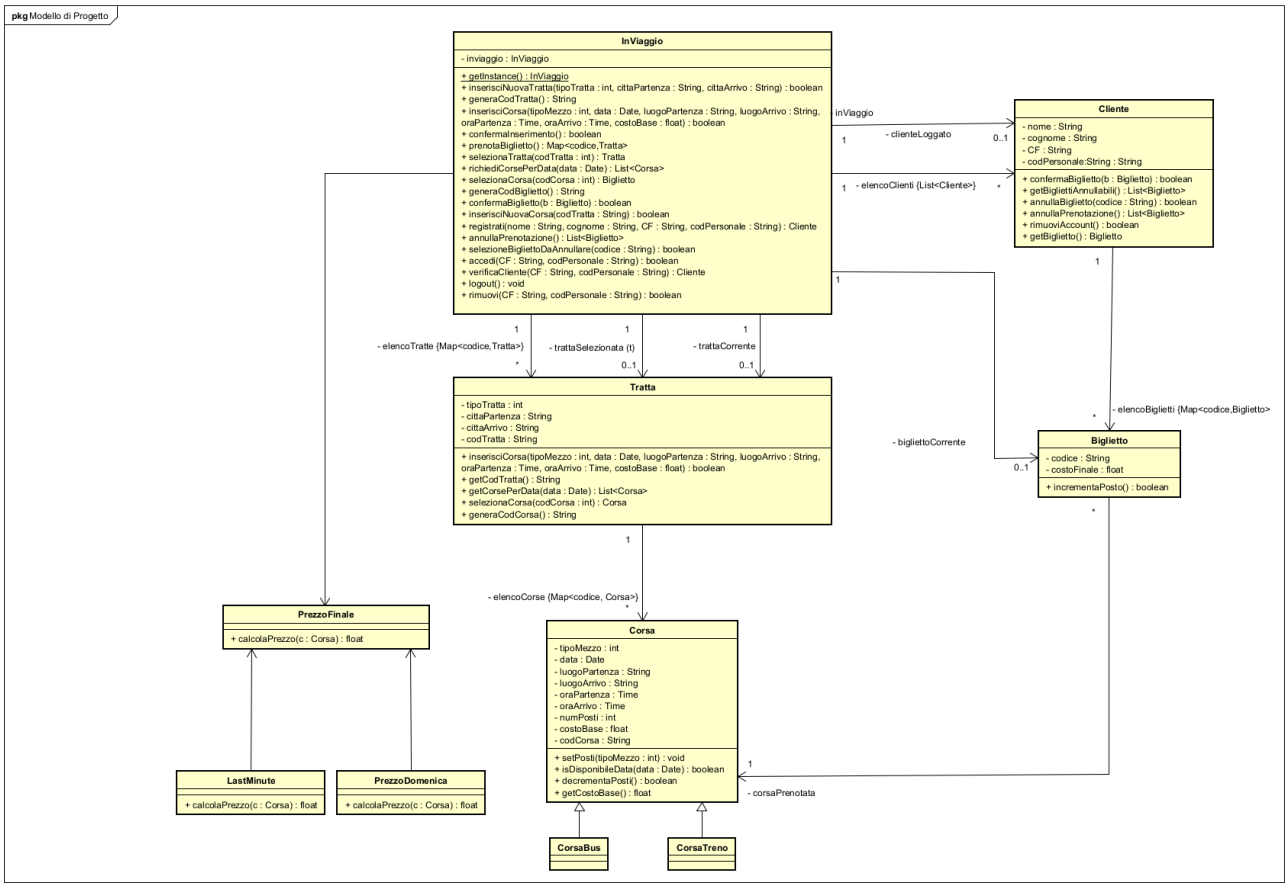
2.2.8 UC8-SD1 selezioneBigliettoDaAnnullare



2.3 SD Caso d'uso d'avviamento



2.4 Modello di Progetto (Diagramma delle Classi di Progetto)



3. Testing

Per la seconda iterazione sono stati fatti i test per verificare il corretto funzionamento dei casi principali UC3, UC5, UC8 con le relative estensioni. Sono state testate anche le estensioni relative ai casi d'uso implementati nell'iterazione precedente UC2 ed UC6. Anche in questa iterazione si è scelto il metodo bottom-up per effettuare i test. Di seguito sono riportati i test dei metodi delle relative classi:

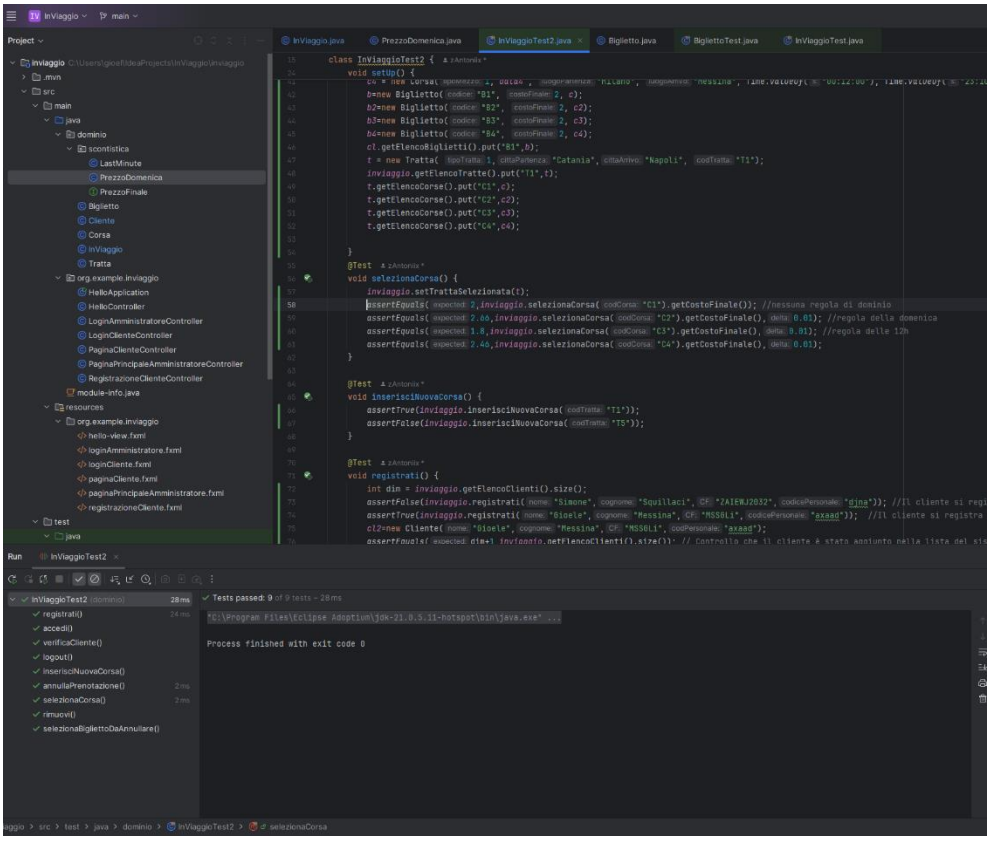
- **Tratta**
  - **inserisciCorsa:** viene verificato che:
    - Il metodo chiamato ritorni "true" quando la corsa è stata inserita correttamente dopo aver effettuato la verifica;
    - Il metodo chiamato ritorni "false" quando la corsa non è stata inserita dopo aver effettuato la verifica.
  - **verificaEsistenzaCorsa:** Viene verificato che:
    - Il metodo chiamato ritorni "true" quando non esiste una corsa con gli stessi dati.

        - Il metodo chiamato ritorni “false” quando esiste una corsa con gli stessi dati.
- **Biglietto**
    - **incrementaPosto:** Viene verificato che:
        - Il metodo ritorni “true”;
        - Il metodo abbia effettivamente aumentato il numero dei posti di 1.
    - **getBigliettiAnnullabili:** Viene verificato che:
        - Il metodo ritorni “false” se il biglietto non è annullabile;
        - Il metodo ritorni “true” se è possibile annullare il biglietto.
- **Cliente**
    - **rimuoviAccount:** Viene verificato che il metodo chiamato ritorni il valore booleano “true”;
    - **annullaPrenotazione:** Viene verificato che:
        - Il metodo ritorni “true”;
        - Il metodo ritorni la lista dei biglietti voluta.
    - **annullaBiglietto:** Viene verificato che:
        - Il metodo chiamato ritorni “true”;
        - Il biglietto venga effettivamente rimosso dalla lista dei biglietti.
- **InViaggio**
    - **inserisciNuovaTratta():** Viene verificato che:
        - il metodo ritorni “true” quando viene inserita una tratta extraurbana con luogoPartenza e luogoArrivo differenti.
        - il metodo ritorni “false” quando viene inserita una tratta extraurbana con luogoPartenza e luogoArrivo uguali.
        - il metodo ritorni “false” quando viene inserita una tratta urbana con luogoPartenza e luogoArrivo differenti.

- il metodo ritorni “true” quando viene inserita una tratta urbana con luogoPartenza e luogoArrivo uguali.

- **selezionaCorsa:** Viene verificato il corretto funzionamento delle regole di dominio testando i 4 casi possibili che si possono avere:
  - La prenotazione non viene effettuata 12h prima della corsa e non viene prenotata una corsa domenicale
  - La prenotazione viene effettuata 12h prima della corsa, ma non è una corsa domenicale
  - La prenotazione effettuata è relativa ad una corsa domenicale, ma non viene fatta 12h prima.
  - La prenotazione viene effettuata 12h prima della corsa e la corsa prenotata avviene di domenica.

Di seguito è riportato il riscontro positivo del test effettuato poiché questo test si basa sulla data attuale in cui viene effettuato il test e quindi non è possibile verificarlo sempre.



- **inserisciNuovaCorsa:** Viene verificato che:
  - Il metodo ritorni “true” quando richiediamo il codice di una tratta esistente;
  - Il metodo ritorni “false” quando richiediamo il codice di una tratta non esistente;
- **registrati:** Viene verificato che:
  - Il metodo ritorni “true” quando l’utente si registra con un codice fiscale non presente nel sistema;
  - Il metodo ritorni “false” quando l’utente si registra con un codice fiscale presente nel sistema;
  - Il cliente sia stato aggiunto alla lista dei clienti gestita dal sistema;
  - Il cliente appena registrato sia stato anche loggato.
- **verificaCliente:** Viene verificato che:
  - il metodo ritorni l’utente associato alle credenziali inserite correttamente;
  - il metodo ritorni “null” quando l’utente inserisce le credenziali di accesso in modo errato, e quindi non esiste nessun cliente con quelle credenziali.
- **accedi:** Viene verificato che:
  - Il metodo chiamato ritorni “true”;
  - Il cliente sia stato correttamente loggato, ossia che la variabile clienteLoggato contenga il cliente che ha effettuato l’accesso;
- **rimuovi:** Viene verificato che:
  - Il metodo ritorni “true” quando l’utente viene rimosso correttamente dal sistema;
  - L’utente sia stato effettivamente rimosso dalla lista dei clienti;
  - Dopo l’eliminazione dell’account non sia più loggato;
  - Il metodo ritorni “false” se l’utente non viene rimosso correttamente a causa di un inserimento errato delle credenziali.

- **logout:** Viene verificato che la variabile clienteLoggato sia “null”;
- **annullaPrenotazione:** Viene verificato che il metodo ritorni la lista desiderata dei biglietti annullabili.
- **selezionaBigliettoDaAnnullare:** Viene verificato che il metodo ritorni “true” se l’eliminazione del biglietto è avvenuta correttamente.