InViaggio: Elaborazione - Iterazione 1

G. Messina – S. Squillaci – A. Zarbo

Introduzione

Finita la fase di ideazione si passa alla fase di elaborazione. Lo scopo delle iterazioni seguenti sarà quello di raffinare la Visione ed in generale raffinare gli elaborati prodotti durante la fase di ideazione, applicare l'analisi e la progettazione orientata agli oggetti.

Per l'iterazione 1 sono stati scelti i seguenti requisiti su cui concentrarsi:

- Implementazione dello scenario principale di successo del caso d'uso UC2: Creazione nuova tratta.
- Implementazione dello scenario principale di successo del caso d'uso UC6: Prenotazione corsa. In questa prima iterazione non vengono applicate le regole di dominio per il calcolo del prezzo.
- Implementazione del caso d'uso d'avviamento necessario per inizializzare questa iterazione.

Per l'iterazione 1 sono state anche effettuate delle ipotesi, ossia l'utente è già registrato e loggato nel sistema.

Aggiornamenti elaborati della fase di Ideazione

Una prima correzione è stata fatta al documento di visione al fine di specificare meglio alcuni aspetti.
Relativamente al caso d'uso preso in esame UC2: Creazione nuova tratta, sono stati individuati alcuni passi poco chiari ed altri errati; pertanto, sono state apportate le dovute correzioni e modifiche allo scenario principale. Di seguito vengono riportati UC2 e UC6 aggiornati:

UC2: Creazione nuova tratta

Nome caso d'uso	Creazione nuova tratta
Portata	Applicazione In Viaggio
Livello	Obiettivo utente
Attore primario	Amministratore
Parti interessate	Amministratore: vuole gestire la creazione di una nuova tratta specificando il mezzo utilizzato e la tipologia
Pre-condizioni	L'amministratore ha stipulato degli accordi con la società privata o pubblica di trasporto
Garanzia di successo	Le informazioni relative alla nuova tratta sono inserite con successo nel sistema

Scenario principale di successo	1. L'Amministratore vuole inserire una nuova tratta; 2. L'Amministratore sceglie l'attività "Inserimento nuova tratta"; 3. L'Amministratore inserisce se la tratta è urbana o extraurbana; 4. L'Amministratore inserisce città di partenza, città di arrivo; 5. Il sistema genera un codice univoco per la tratta. 6. L'amministratore inserisce per la tratta appena aggiunta le relative corse specificando: la data, il mezzo di trasporto (treno o autobus) che andrà a definire il numero di posti disponibili, luogo di partenza e luogo di arrivo, orario di partenza, orario di arrivo ed il costo base. Il sistema genererà un codice univoco per la corsa. Il Sistema registra le informazioni relative alla corsa. *Il passo 5 viene ripetuto fin quando serve.* 7. L'Amministratore indica di aver finito.
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 **3a.** 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.
Requisiti speciali	
Elenco delle variazioni tecnologiche e dei dati	
Frequenza di ripetizioni	Legata ad ogni nuovo accordo stipulato o a periodi festivi
Varie	

UC6: Prenotazione corsa

Nome caso d'uso	Prenotazione corsa
Portata	Applicazione In Viaggio
Livello	Obiettivo utente
Attore primario	Cliente
Parti interessate	Cliente: vuole prenotare un biglietto relativo ad una corsa di una specifica tratta
Pre-condizioni	Il cliente è già registrato e loggato

Garanzia di successo	Il cliente riceve nel suo profilo la prenotazione effettuata con le relative informazioni
Scenario principale di successo	1. Il cliente vuole prenotare un nuovo biglietto; 2. Il cliente sceglie l'attività "Prenota biglietto"; 3. Il sistema mostra l'elenco di tutte le tratte disponibili; 4. Il cliente sceglie la tratta che vuole effettuare; 5. Il cliente sceglie il giorno in cui vuole effettuare la corsa selezionata; 6. Il sistema mostra al cliente le corse relative alla tratta scelta con ancora dei posti disponibili; 7. Il cliente sceglie la corsa che preferisce; 8. Il sistema calcola il prezzo finale secondo le regole di dominio e genera un codice univoco per il biglietto; 9. Il sistema mostra il riepilogo della prenotazione. 10. Il cliente conferma la prenotazione; 11. Il sistema conferma la prenotazione aggiornando la disponibilità dei posti; 12. Il sistema inserisce il biglietto nel profilo del cliente 13. Il sistema mostra il biglietto con le relative informazioni
Estensioni	***a.** In un qualsiasi momento il Sistema fallisce e si arresta improvvisamente. 1. Il cliente riavvia il software e deve ripetere la procedura.
Requisiti speciali	
Elenco delle variazioni tecnologiche e dei dati	
Frequenza di ripetizioni	Legata all'esigenza del cliente
Varie	

1. Analisi Orientata agli Oggetti

1.1. Modello di Dominio

In questo elaborato grafico vengono identificate le classi, gli attributi e le associazioni dei concetti significativi all'interno del dominio del problema. Di seguito vengono elencate le classi concettuali individuate

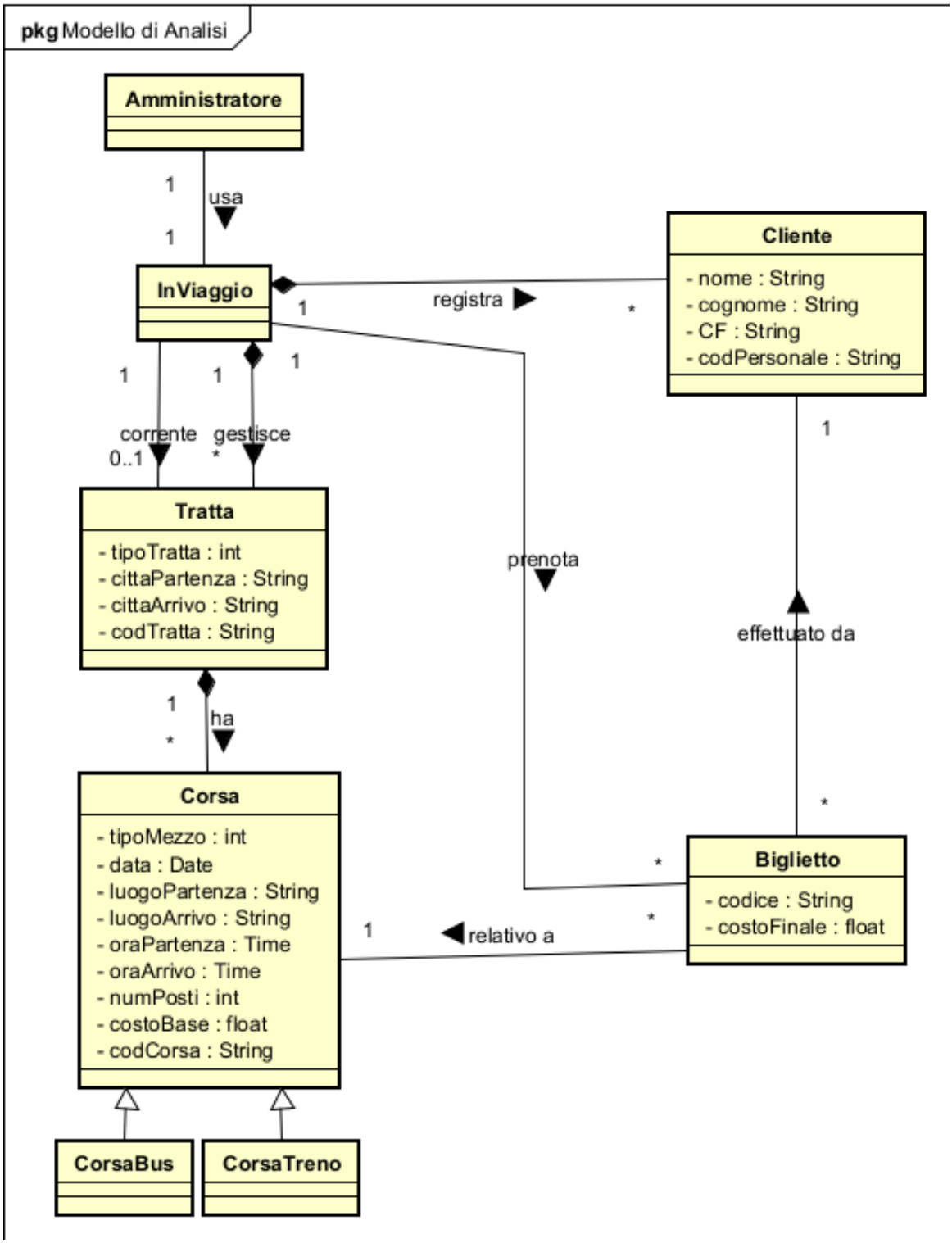
dopo un’attenta analisi dello scenario principale di successo relativo al caso d’uso UC2: *Creazione nuova tratta*:

- Amministratore
- InViaggio
- Tratta
- Corsa (individuando due specializzazioni della classe concettuale: corsaBus e corsaTreno)

Si procede con l’analisi del flusso base di successo del caso d’uso UC6: *Prenotazione corsa.* Nello specifico sono state individuate le seguenti classi concettuali:

- Cliente
- Biglietto

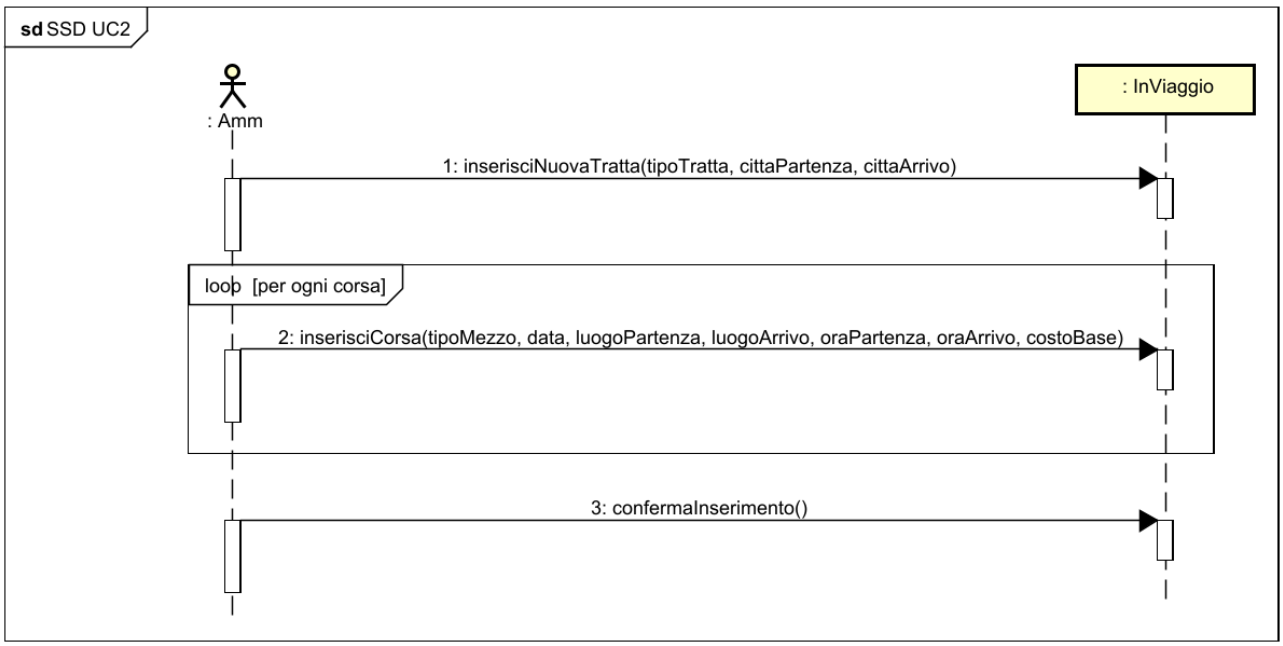
Dall’analisi dei due casi d’uso presi in esame, tenendo anche in considerazione i relativi attributi e associazioni, è stato ricavato il seguente Modello di Dominio:



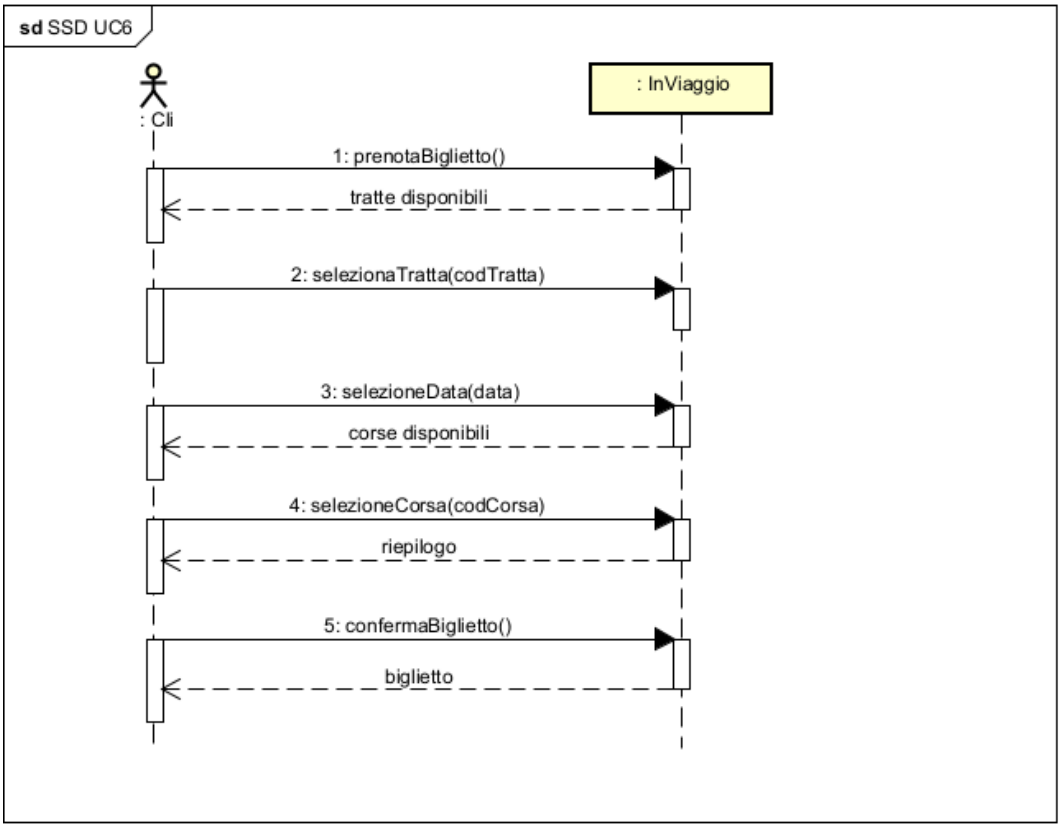
1.2. Diagrammi di Sequenza di Sistema (SSD)

Il passo successivo dell'analisi orienta agli oggetti è la realizzazione dei Diagrammi di Sequenza di Sistema relativi ai due casi d'uso presi in esame, UC2 e UC6.

1.2.1 SSD UC2



1.2.2 SSD UC6



1.3. Contratti delle Operazioni

Vengono ora descritte le operazioni svolte dal sistema nel caso d'uso UC2 attraverso i Contratti.

1.3.1 inserisciNuovaTratta

Operazione	inserisciNuovaTratta(tipoTratta:int,cittaPartenza:String, cittaArrivo:String)
Riferimenti	Caso d'uso: Creazione nuova tratta
Pre-condizioni	-
Post-condizioni	- È stata creata una nuova istanza t di Tratta: - Gli attributi di t sono stati inizializzati; - t è stata associata a inViaggio tramite l'associazione "corrente";

1.3.2 inserisciCorsa

Operazione	inserisciCorsa(tipoMezzo:int,date Date,luogoPartenza:String,luogoArrivo:String,oraPartenza :Time, oraArrivo:Time, costoBase: float)
Riferimenti	Caso d'uso: Creazione nuova tratta
Pre-condizioni	È in corso la definizione di Tratta t
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";

1.3.3 confermaInserimento

Operazione	confermaInserimento()
Riferimenti	Caso d'uso: Creazione nuova tratta
Pre-condizioni	È in corso la definizione di Tratta t
Post-condizioni	- è stata associata t a inViaggio tramite l'associazione"gestisci"; - la relazione "corrente" tra t e c viene eliminata.

Vengono ora descritte le operazioni svolte dal sistema nel caso d'uso UC6 attraverso i Contratti.

1.3.4 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; - Viene inizializzato l'attributo codice di b;

1.3.5 confermaBiglietto

Operazione	confermaBiglietto()
Riferimenti	Caso d'uso: Prenotazione corsa
Pre-condizioni	È in corso la prenotazione del biglietto
Post-condizioni	- Viene aggiornato (decrementato) c.numPosti; - b viene associata a cl tramite l'associazione "effettuato da";

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 UC2 e UC6.

2.1 Pattern applicati

Sono stati applicati i principali pattern GRASP come Controller, Information Expert, basso accoppiamento (low Coupling) e alta coesione (High cohesion). Sono stati inoltre utilizzati anche alcuni patter GoF in particolare sono stati utilizzati:

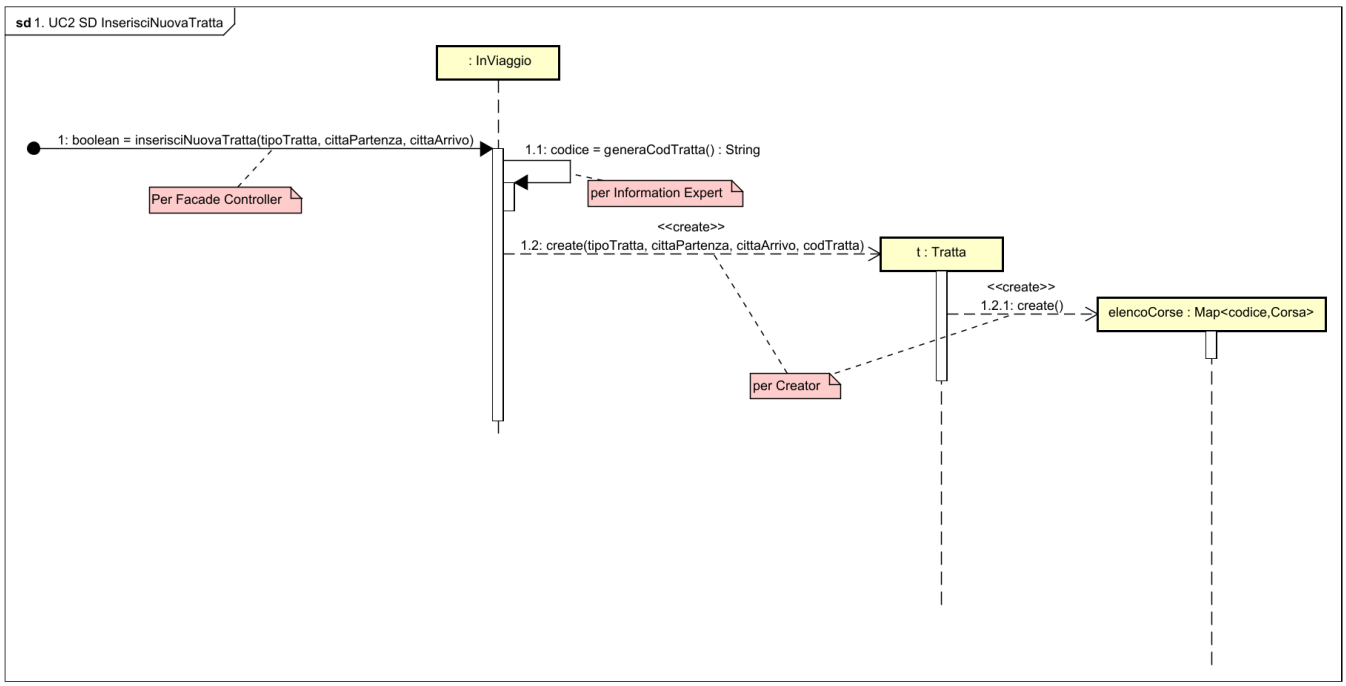
- Singleton: utilizzato per la classe software inViaggio al fine di avere un'unica istanza della classe all'interno del sistema.

- Facade: utilizzato per fornire un’unica interfaccia per un insieme di funzionalità sparse su più classi.

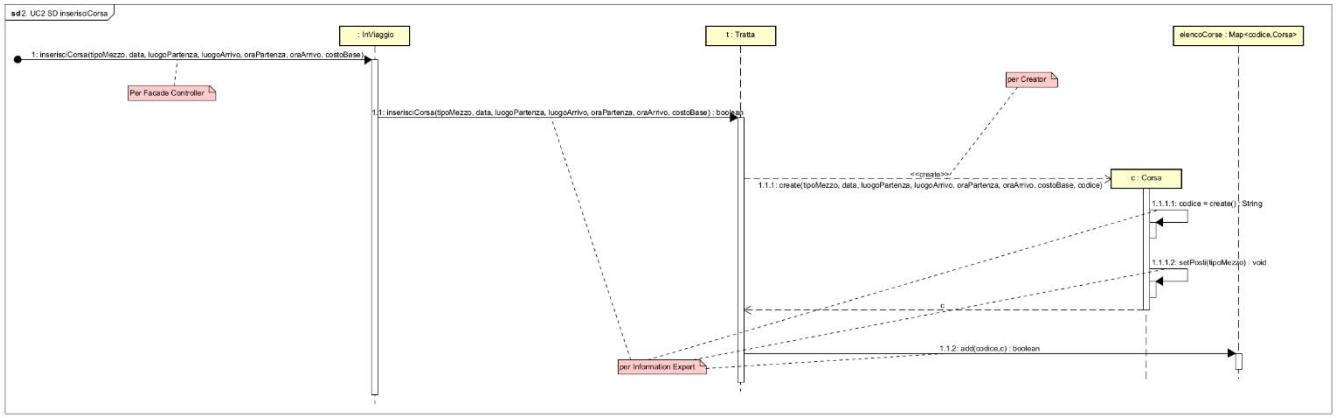
2.2 Diagrammi di Sequenza

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

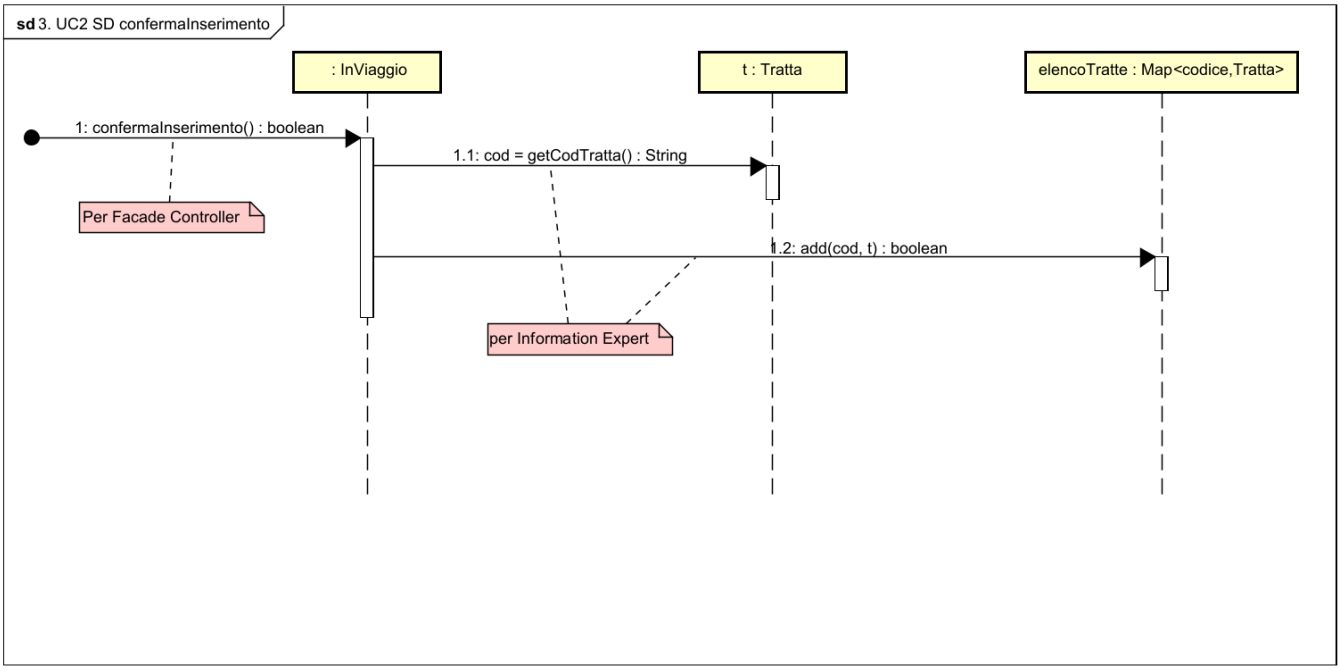
2.2.1 UC2 inserisciNuovaTratta



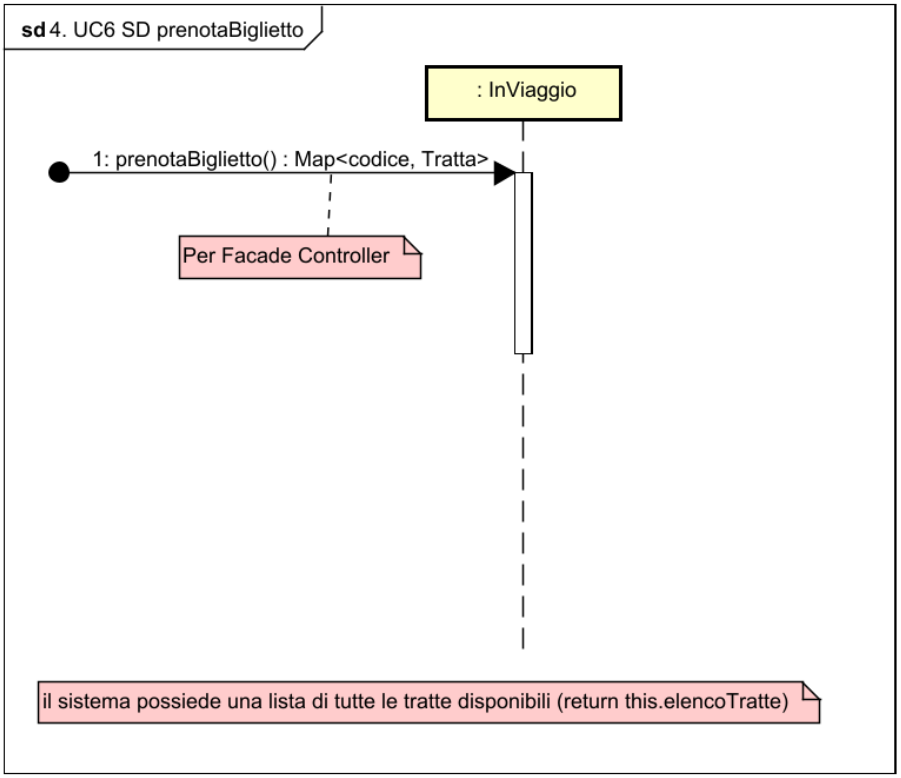
2.2.2 UC2 inserisciCorsa



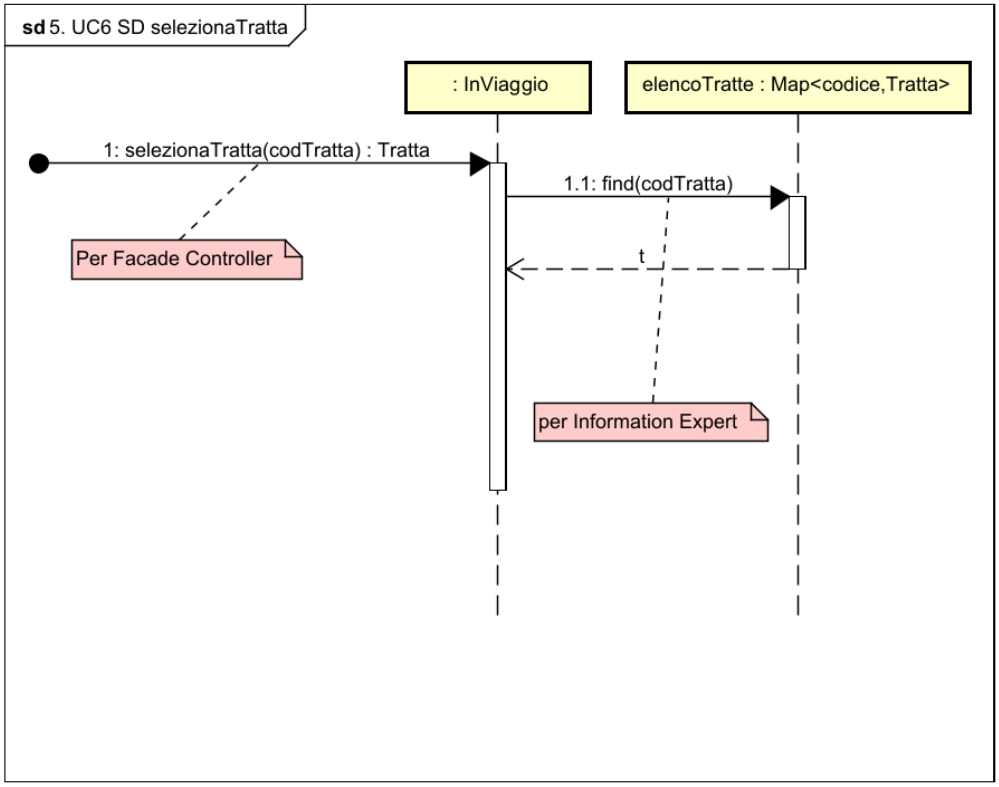
2.2.3 UC2 confermaInserimento



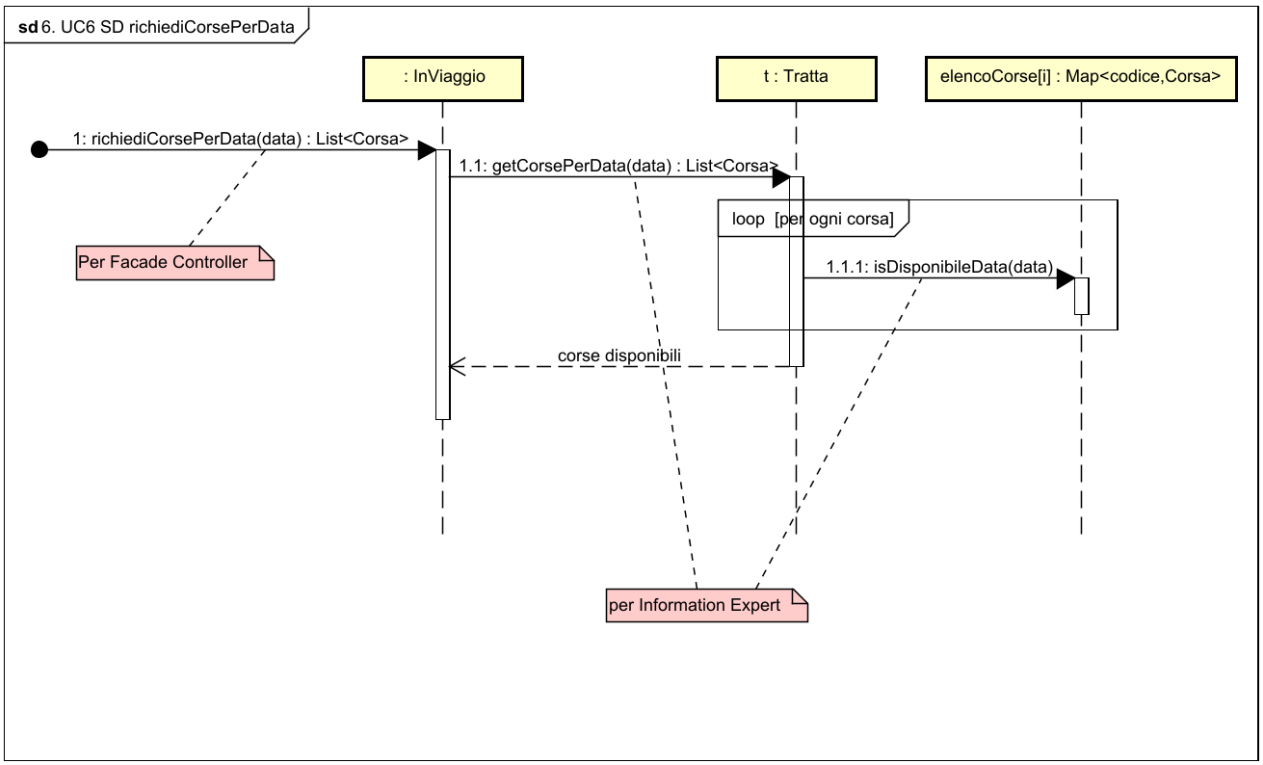
2.2.4 UC6 prenotaBiglietto



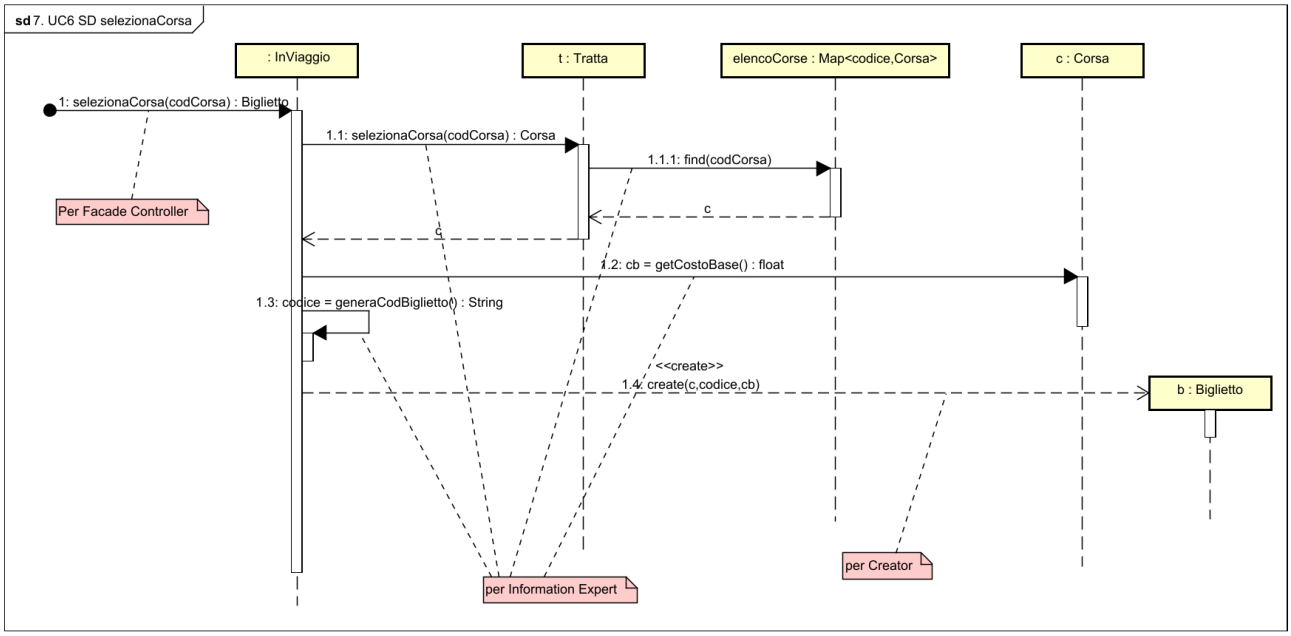
2.2.5 UC6 selezionaTratta



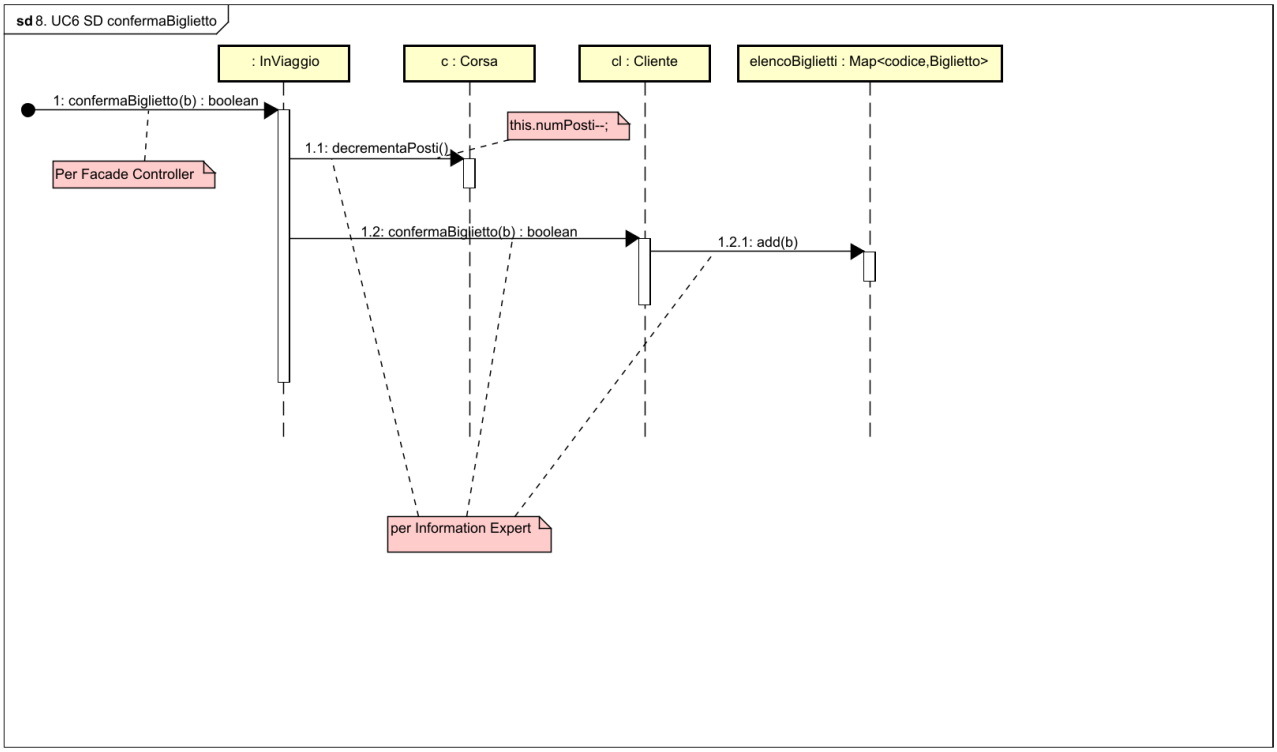
2.2.6 UC6 richiediCorsePerData



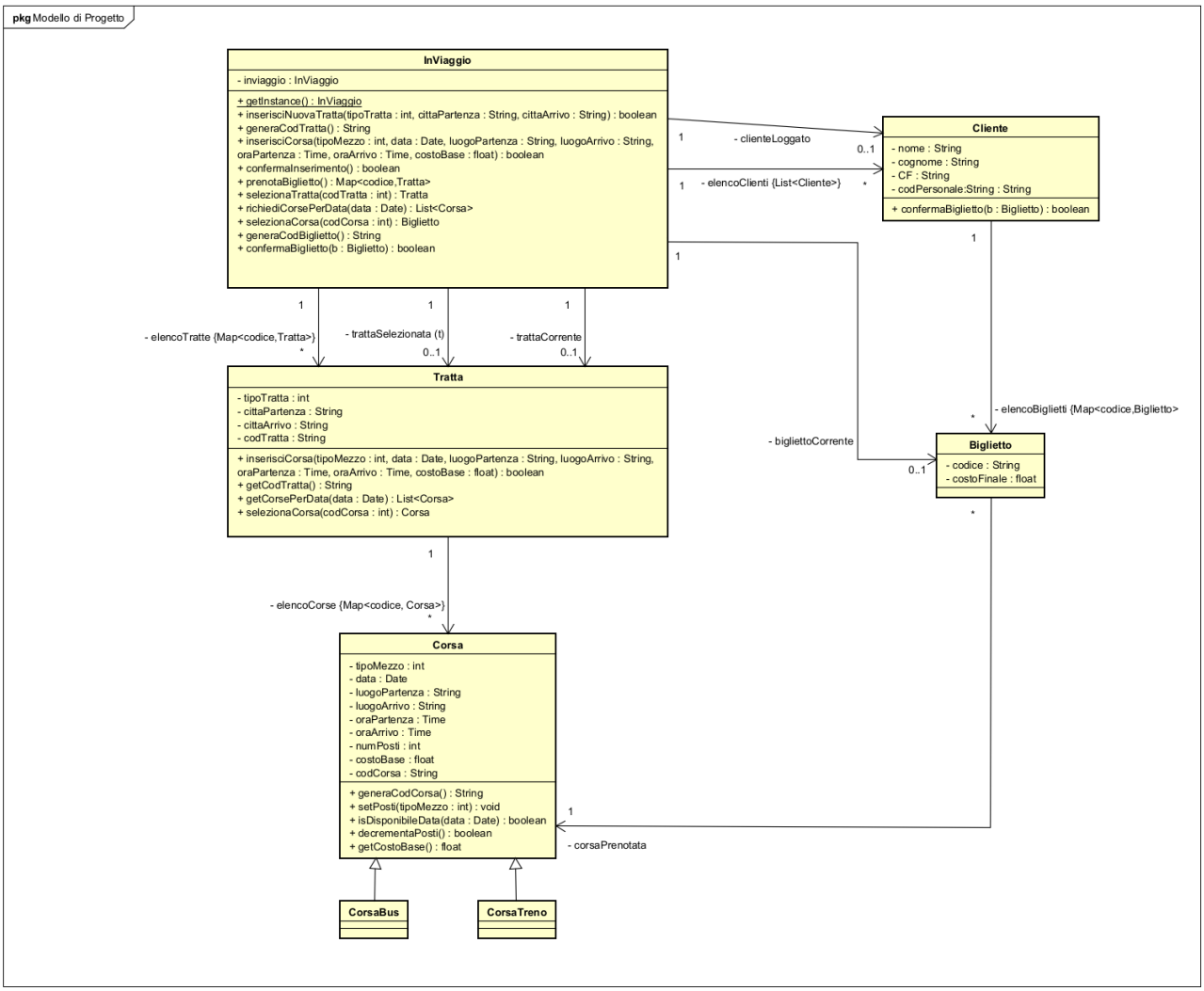
2.2.7 UC6 selezionaCorsa



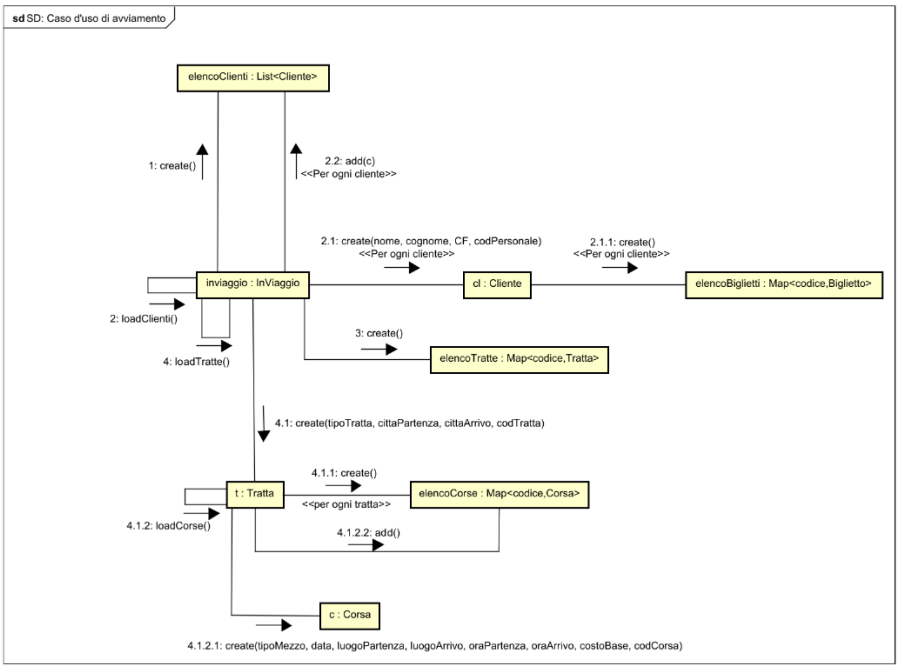
2.2.8 UC6 confermaBiglietto



2.3 Modello di Progetto (Diagramma delle Classi di Progetto)



2.4 SD Caso d'uso d'avviamento



3. Testing

Il testing è un processo fondamentale nello svilupo del software in quando consente di esaminare e verificare la funzionalità e la correttezza del programma. Per effettuare il testing sono stati utilizzati i test unitari con un approccio Bottom-Up, quindi per testarli si è partiti dalle classi meno accoppiate verso le classi più accoppiate.

In particolare, sono stati testati i seguenti metodi delle relative classi:

- Corsa
  - setPosti(): viene verificato che in base al valore assunto dall'attributo "tipoMezzo" venga settato in modo opportuno l'attributo "numPosti" (viene settato a 52 se tipoMezzo=1, a 100 altirmenti).
  - isDisponibileData():viene verificato:
    - che venga ritornato il valore "true" quando viene chiamato il metodo fornendo una data in cui è presente una corsa con posti disponibili.
    - che venga ritornato il valore "false" quando viene chiamato il metodo fornendo una data in cui non presente una corsa.
    - che venga ritornato il valore "false" quando viene chiamato il metodo fornendo una data in cui è presente una corsa ma senza posti disponibili.
  - decrementaPosti():viene verificato che il metodo chiamato torni "true" quando il numero dei posti disponibili di una data corsa è maggiore di "0", false altrimenti.
- Tratta
  - inserisciCorsa(): Viene verificato che:
    - il metodo chiamato ritorni "true".
    - viene verificato se è avvenuto il corretto inserimento della corsa nell'elenco corse posseduto della tratta specifica, verificando che la dimensione della mappa sia aumenta di 1
  - generaCodiceCorsa(): viene verificato che il metodo generi il codice corsa in modo corretto.

- getCorsePerData(): viene verificato che il metodo ritorni il valore booleano “true” se la lista ritornata dal metodo relativa alle corse disponibili nella data specificata sia quella attesa.
- selezionaCorsa(): viene verificato che il metodo ritorni un’istanza di tipo Corsa, e viene anche verificato che l’istanza ritornata sia quella attesa.

- Cliente
  - confermaBiglietto(): viene verificato:
    - il corretto inserimento del biglietto nell’elenco dei biglietti dello specifico utente, verificando che il metodo chiamato ritorni “true”.
    - Che la dimensione della mappa contenente i biglietti dell’utente sia aumenta di 1.
  - getElencoBiglietti(): viene verificato che il metodo chiamato ritorni il valore booleano “true”, che indica la corretta restituzione dell’elenco dei biglietti dello specifico utente.
- inViaggio
  - getInstance(): Viene verificato che il metodo ritorni l’unica istanza del sistema.
  - inserisciNuovaTratta(): Viene verificato che il metodo chiamato ritorni “true” quando vengono passati come parametri i dati della tratta che si vuole creare.
  - generaCodTratta(): Viene verificato che il metodo generi il codice tratta in modo corretto.
  - inserisciCorsa(): Viene verificato che il metodo chiamato passando i valori dei parametri relativi alla corsa che si vuole creare ritorni “true”.
  - confermaInserimento(): Viene verificato che:
    - il metodo ritorni “true” dopo l’inserimento della tratta nell’elenco delle tratte posseduto dal sistema.
    - la dimensione della mappa posseduta dal sistema contenente la lista delle tratte disponibili abbia un elemento in più.
  - prenotaBiglietto(): Viene verificato che il metodo chiamato ritorni una mappa contenente oggetti di tipo Tratta.

- selezionaTratta(): Viene verificato che il metodo ritorni un’istanza di tipo Tratta, e viene anche verificato che la tratta tornata abbia il codice desiderato.
- richiediCorsePerData():Viene verificato che il metodo chiamato ritorni la lista delle corse disponibili per una data specifica passata come parametro.
- generaCodBiglietto(): Viene verificato che il metodo generi il codice del biglietto in modo corretto.
- selezionaCorsa(): Viene verificato che il metodo chiamato fornendo come parametro il codice della corsa selezionata ritorni un’istanza della classe Biglietto.
- confermaBiglietto(): Viene verificato che il metodo ritorni “true” dopo l’inserimento del biglietto nella lista dei biglietti posseduti dal cliente.