05 – Elaborazione- Iterazione 4

2.1 Introduzione

Durante questa quarta iterazione ci si concentrerà su:

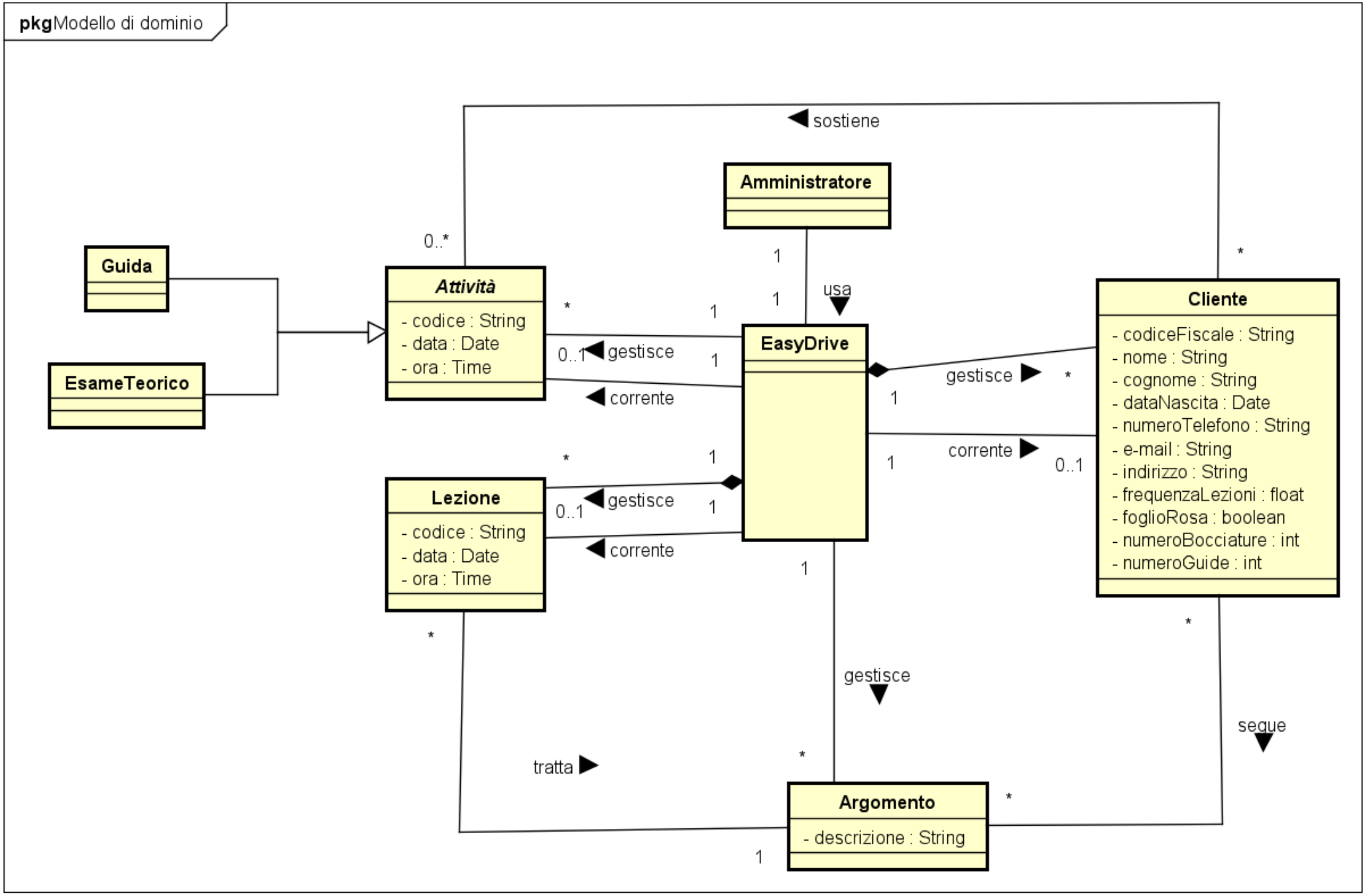
- Implementare lo scenario principale di successo e tutte le estensioni finora individuate riguardante il caso d'uso **UC7**: ***Gestisci programmazione guide (CRUD);***
- Implementare lo scenario principale di successo e tutte le estensioni finora individuate riguardante il caso d'uso **UC8**: ***Prenota guida;***
- Implementare lo scenario principale di successo e tutte le estensioni finora individuate riguardante il caso d'uso **UC9**: ***Aggiorna numero guide sostenute;***

2.2 Analisi Orientata agli Oggetti

Al fine di descrivere il dominio da un punto di vista ad oggetti e gestire ulteriori requisiti, saranno utilizzati nuovamente gli stessi strumenti dell'iterazione precedente (Modello di Dominio, SSD - Sequence System Diagram e Contratti delle operazioni). In particolare, i paragrafi seguenti permettono di evidenziare i cambiamenti che tali elaborati hanno subito rispetto alla fase precedente.

2.2.1 Modello di Dominio

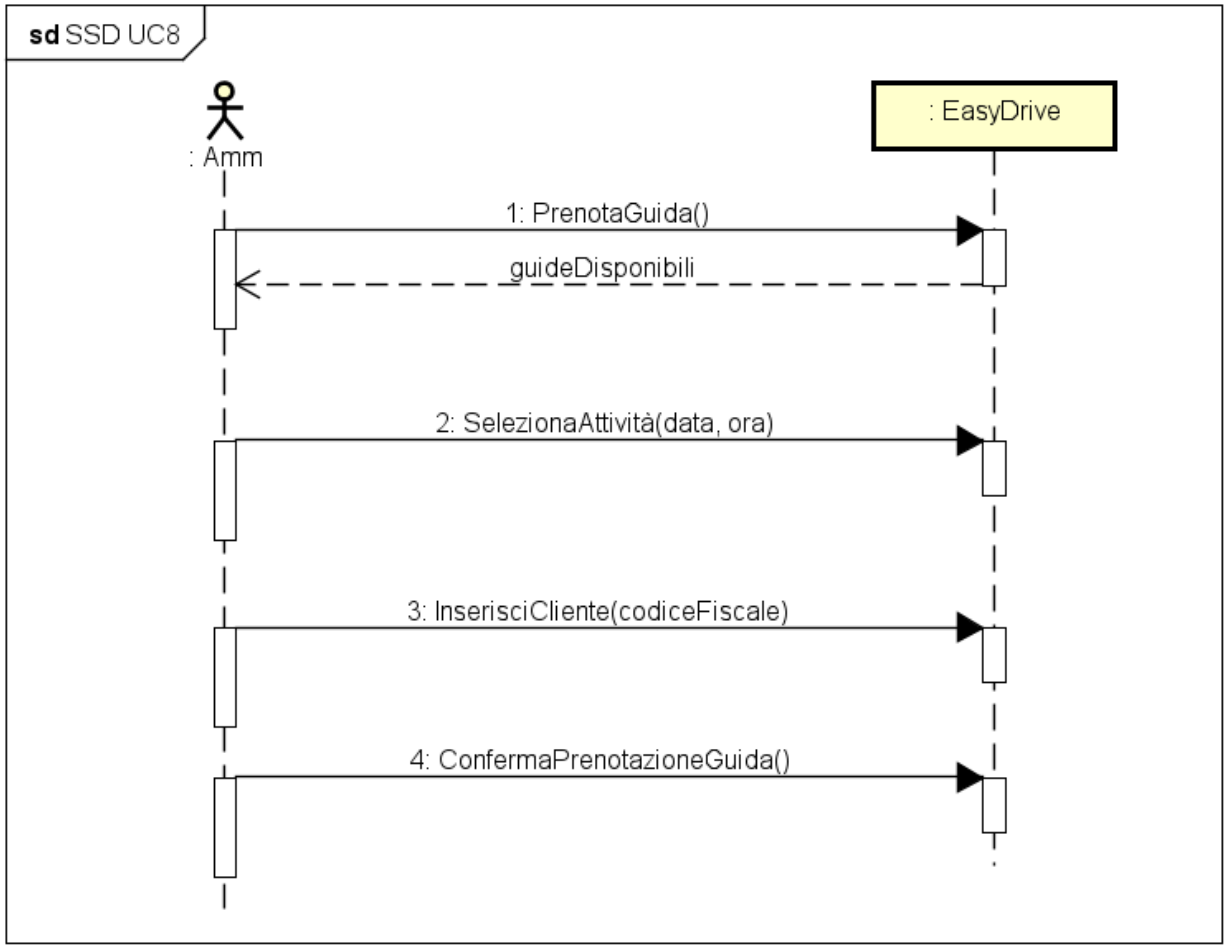
Relativamente ai casi d'uso in esame **(UC7, UC8, UC9)**, nasce l'esigenza di creare una nuova classe "**Guida**" e, poiché abbiamo riscontrato caratteristiche comuni tra **"Guida"** ed **"EsameTeorico"**, abbiamo deciso di creare una generalizzazione attraverso una nuova classe astratta che prenderà il nome di **"Attività"**. Inoltre, è stato aggiunto anche l'attributo *numeroGuide* in **Cliente.**


pkgModello di dominio
sostiene
Amministratore
0..*
1
1
usa
Guida
Attività
- codice : String
- data : Date
- ora : Time
*
1
EasyDrive
Cliente
- codiceFiscale : String
- nome : String
- cognome : String
- dataNascita : Date
- numeroTelefono : String
- e-mail : String
- indirizzo : String
- frequenzaLezioni : float
- foglioRosa : boolean
- numeroBocciature : int
- numeroGuide : int
EsameTeorico
0..1
gestisce
1
corrente
gestisce
*
1
1
corrente
0..1
*
1
Lezione
- codice : String
- data : Date
- ora : Time
0..1
gestisce
1
corrente
1
*
gestisce
segue
tratta
*
*
Argomento
- descrizione : String
1

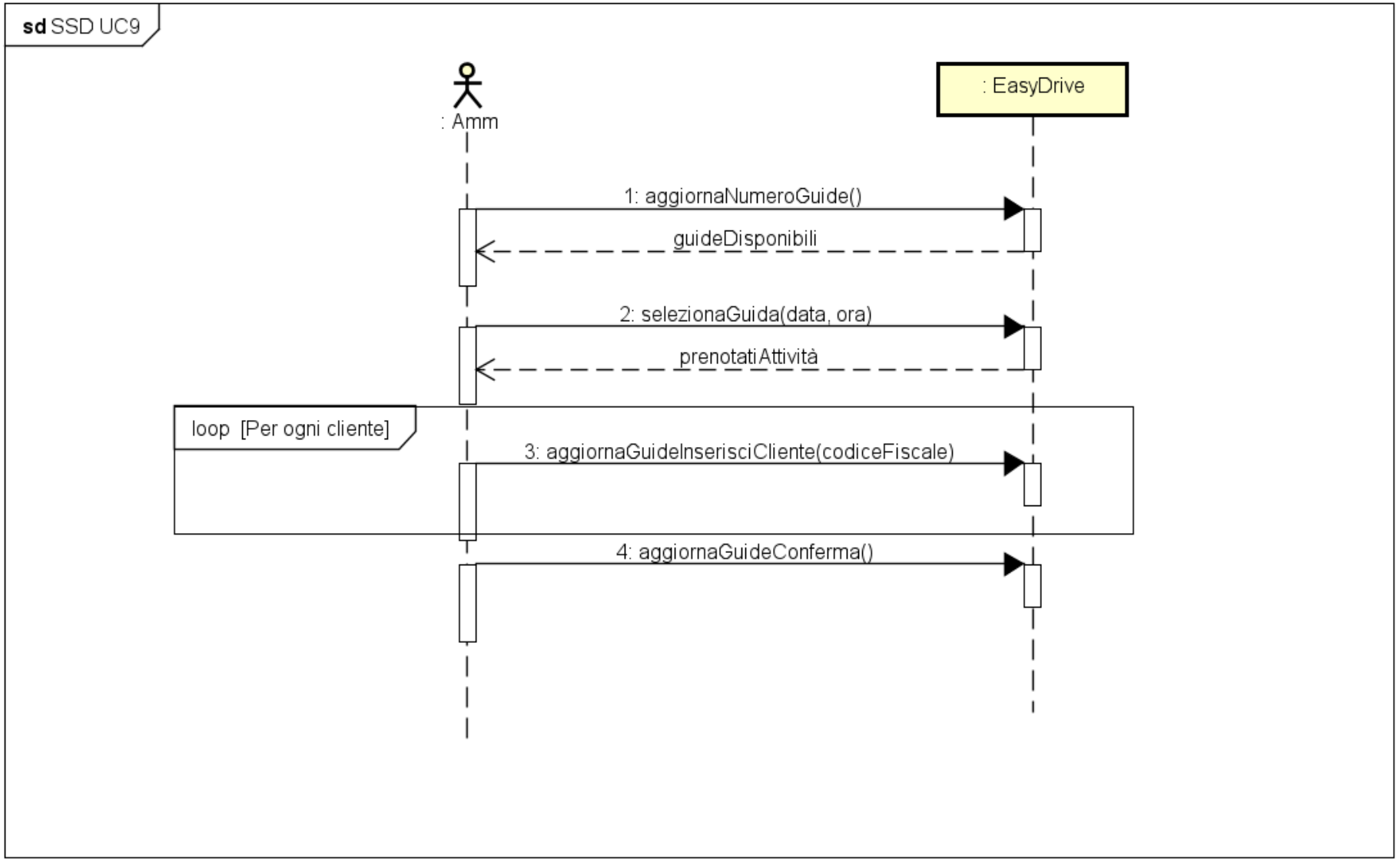
2.2.2 Diagramma di sequenza di sistema

Per il caso d'uso **UC7 (CRUD) non è** stato creato l' SSD.

SSD UC8:



SSD UC9:


sd SSD UC9
: Amm
: EasyDrive
1: aggiornaNumeroGuide()
guideDisponibili
2: selezionaGuida(data, ora)
prenotatiAttività
loop [Per ogni cliente]
3: aggiornaGuideInserisciCliente(codiceFiscale)
4: aggiornaGuideConferma()

2.2.3 Contratti delle operazioni

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

UC8:

Contratto CO1: prenotaGuida

Operazione: prenotaGuida();

Riferimenti: caso d'uso: Prenota Guida;

Pre-condizioni:

Post-Condizioni: - sono state recuperate le istanze di "Guida" sulla base della data e ora attuali;

Contratto CO2: selezionaAttività

Operazione: selezionaAttività(data, ora);

Riferimenti: caso d'uso: Prenota Guida;

Pre-condizioni: - È noto l'elenco delle guide disponibili;

Post-Condizioni: - È stata recuperata l'istanza a di Guida sulla base di data e ora;

- "a" è stata associata a EasyDrive tramite l'associazione "corrente";

Contratto CO3: InserisciCliente

Operazione: InserisciCliente(codiceFiscale);

Riferimenti: caso d'uso: Prenota Guida;

Pre-condizioni: - È in corso la prenotazione di un cliente ad una guida;

Post-Condizioni: - È stata recuperata l'istanza c di Cliente sulla base di codiceFiscale;

- "c" è stata associata a EasyDrive tramite l'associazione "corrente";

Contratto CO4: confermaPrenotazioneGuida

Operazione: confermaPrenotazioneGuida()

Riferimenti: caso d'uso: Prenota Guida;

Pre-condizioni: - È in corso l'aggiornamento del numero di guide di un cliente;

Post-Condizioni: - È stato recuperato l'attributo foglioRosa di Cliente c;

- "c" è stata associata ad Attività tramite l'associazione "sostiene" nel caso in cui l'attributo "foglioRosa" sia "true"

UC9:

Contratto CO1: aggiornaNumeroGuide

Operazione: aggiornaNumeroGuide();

Riferimenti: caso d'uso: Aggiorna Numero Guide Sostenute;

Pre-condizioni:

Post-Condizioni: - sono state recuperate le istanze di "Guida" svolte in passato sulla base della data e ora attuali;

Contratto CO2: selezionaGuida

Operazione: selezionaGuida(data, ora);

Riferimenti: caso d'uso: Aggiorna Numero Guide Sostenute;

Pre-condizioni: - È noto l'elenco delle guide svolte in passato;

Post-Condizioni: - È stata recuperata l'istanza "a" di Guida sulla base di data e ora;

- "a" è stata associata a EasyDrive tramite l'associazione "corrente";
- Sono state recuperate le istanze di "Cliente" che hanno partecipato alla guida "a".

Contratto CO3: aggiornaGuideInserisciCliente

Operazione: aggiornaGuideInserisciCliente(codiceFiscale);

Riferimenti: caso d'uso: Aggiorna Numero Guide Sostenute;

Pre-condizioni: - È in corso l'aggiornamento del numero di guide di un cliente;

Post-Condizioni: - È stata recuperata l'istanza c di Cliente sulla base di codiceFiscale;

- "c" è stata associata a EasyDrive tramite l'associazione "corrente";

Contratto CO4: aggiornaGuideConferma

Operazione: aggiornaGuideConferma()

Riferimenti: caso d'uso: Aggiorna Numero Guide Sostenute;

Pre-condizioni: - È in corso l'aggiornamento del numero di guide di un cliente;

PostCondizioni: - È stato incrementato l'attributo numeroGuide del cliente c;

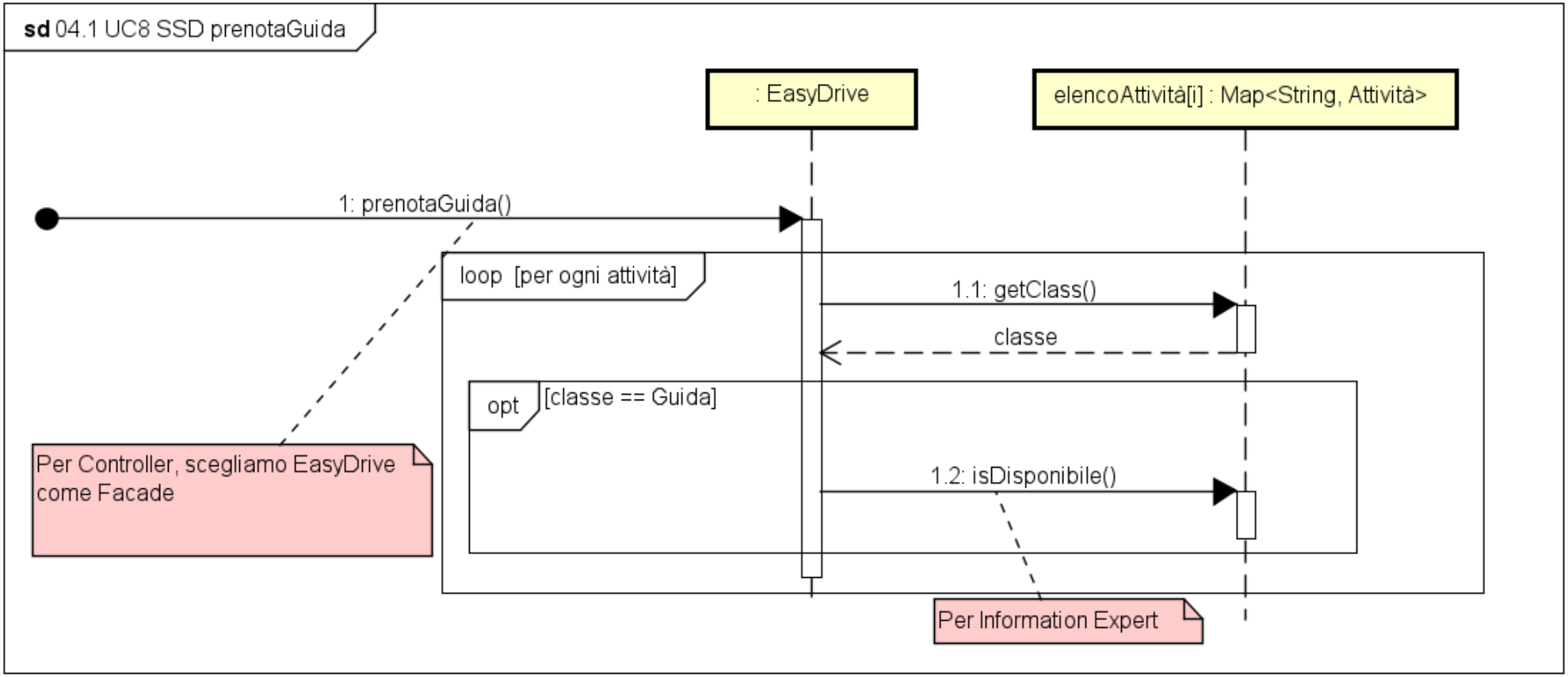
2.3 Progettazione

La progettazione orientata agli oggetti è la disciplina di UP interessata alla definizione degli oggetti software, delle loro responsabilità e a come questi collaborano per soddisfare i requisiti individuati nei passi precedenti. Seguono dunque i diagrammi di Interazione più significativi e il diagramma delle Classi relativi al caso d'uso **UC8** e **UC9**, determinati a seguito di un attento studio degli elaborati scritti in precedenza.

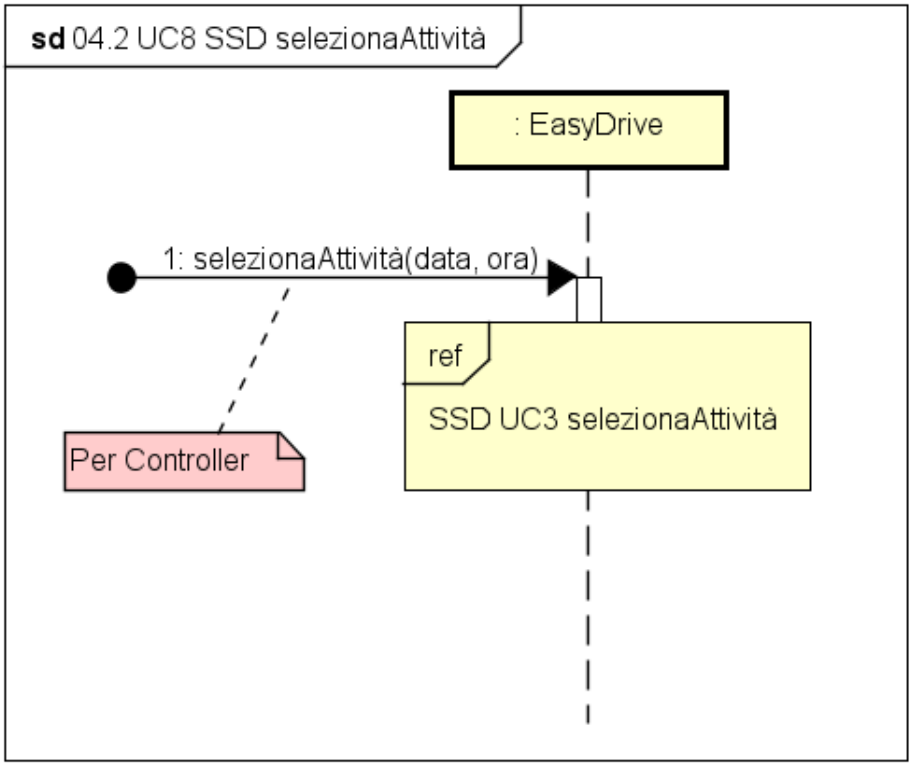
2.3.1 Diagrammi di sequenza

UC8:

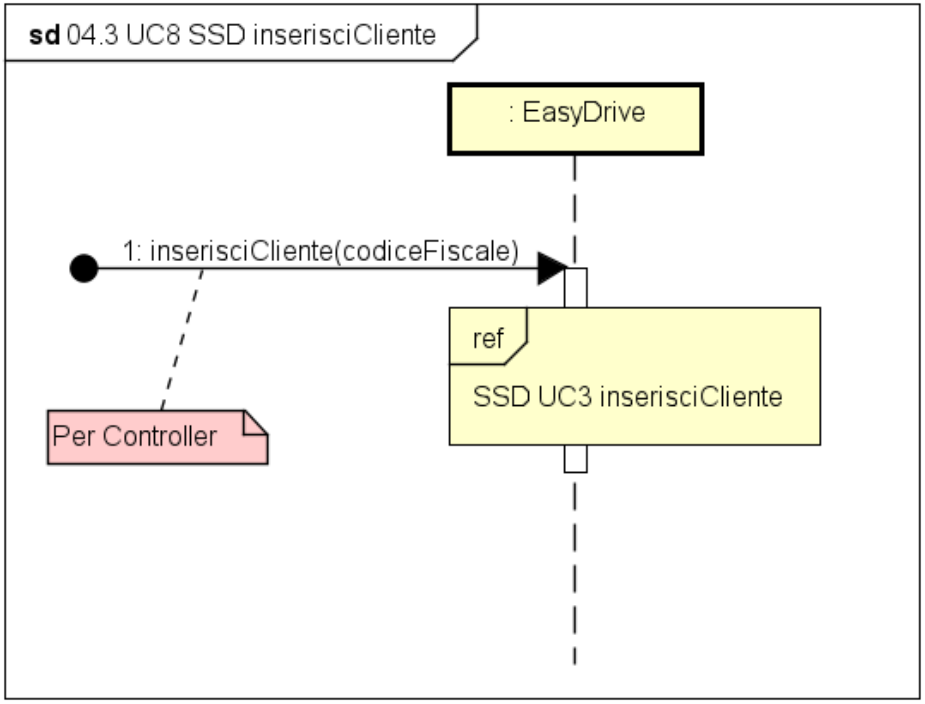
- **Prenota Guida**



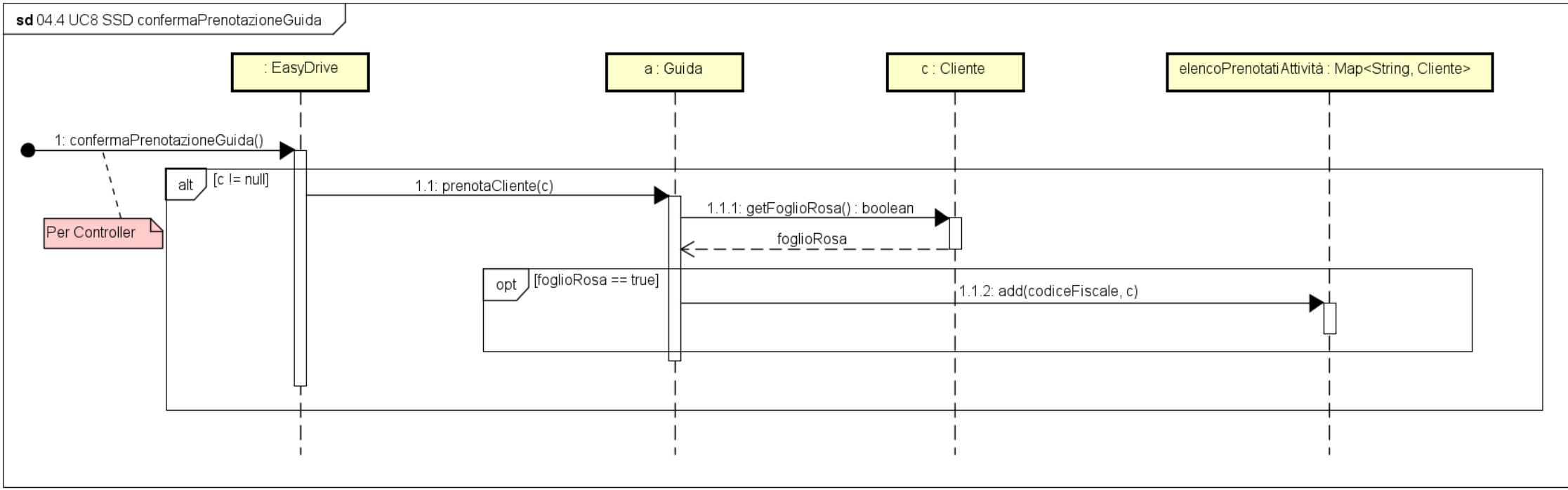
- **Seleziona Attività**



- **Inserisci Cliente**

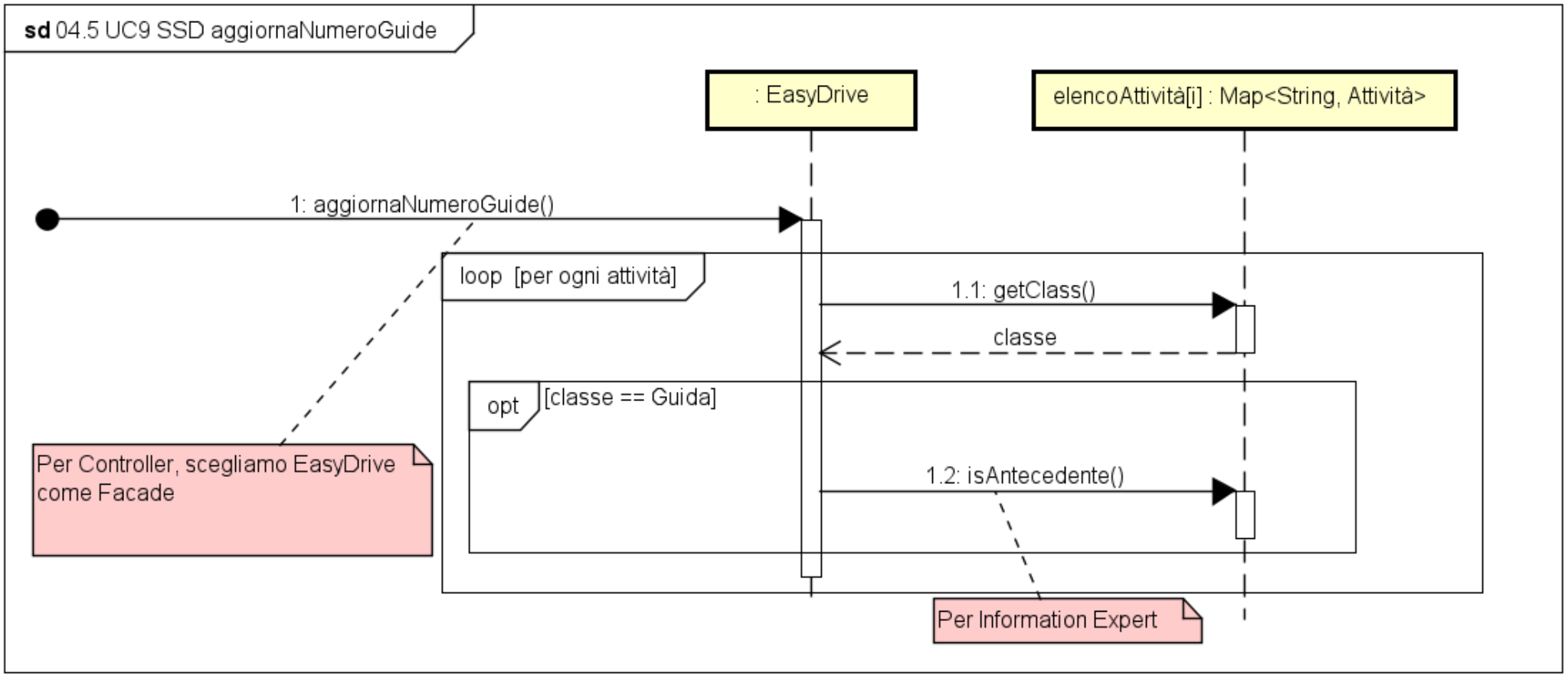


- **Conferma Prenotazione Guide**

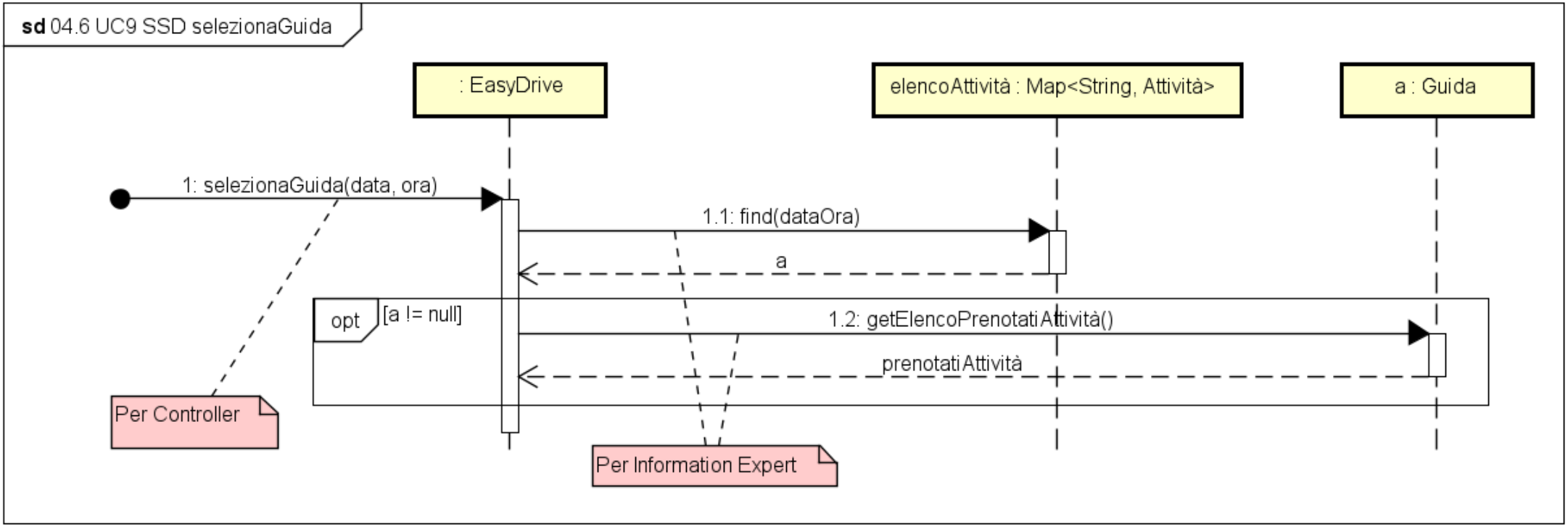


UC9:

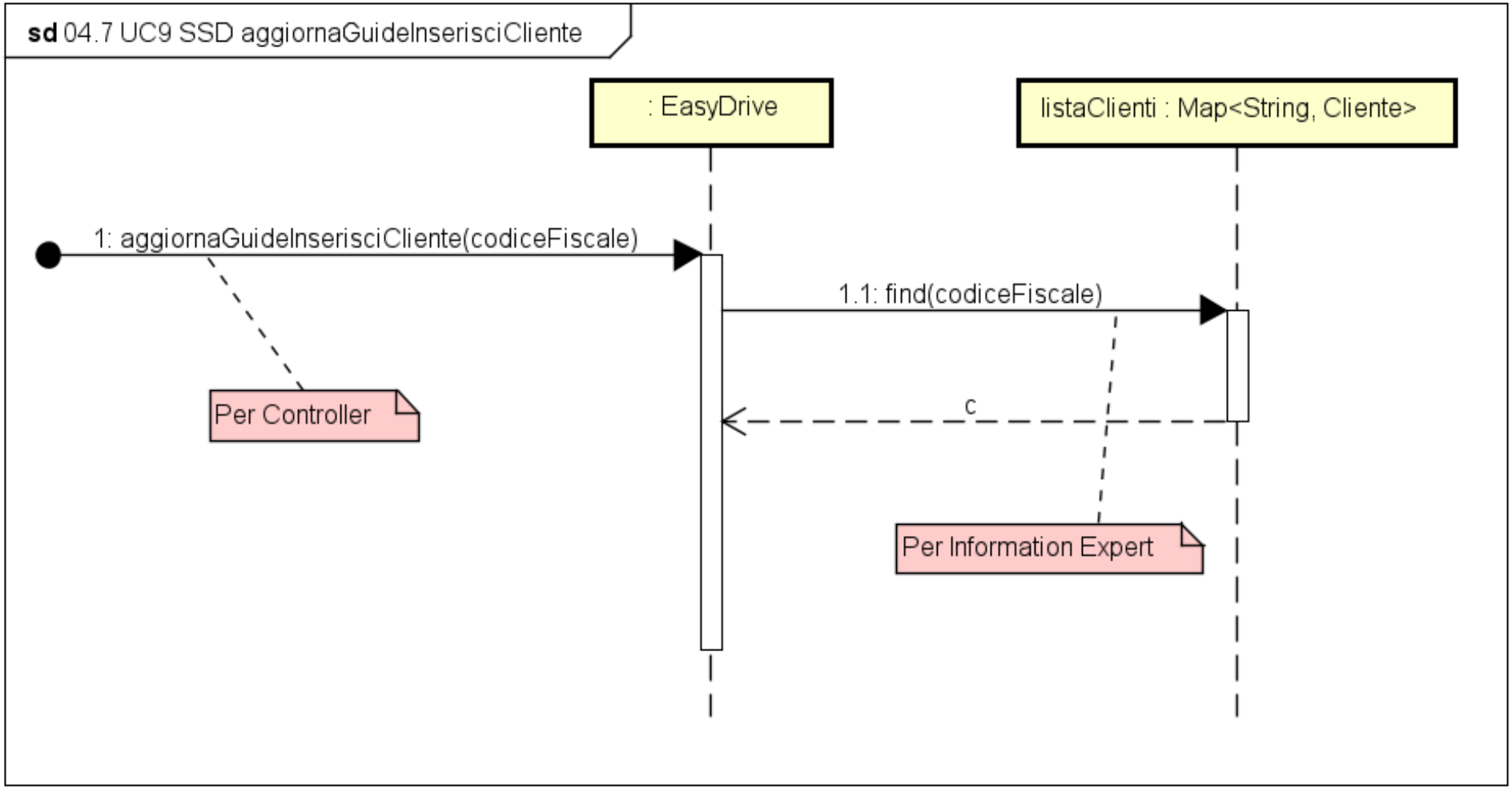
- **Aggiorna Numero Guide**



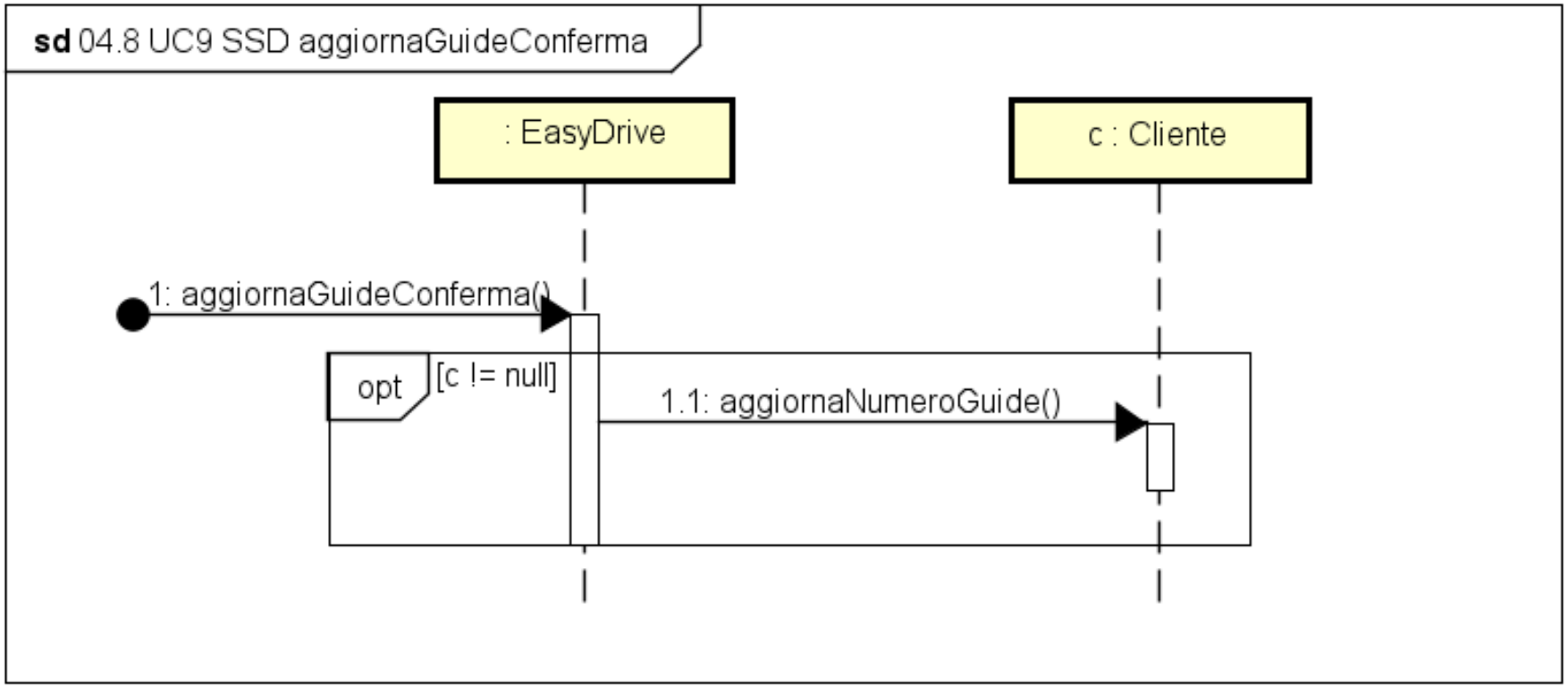
- **Seleziona Guida**



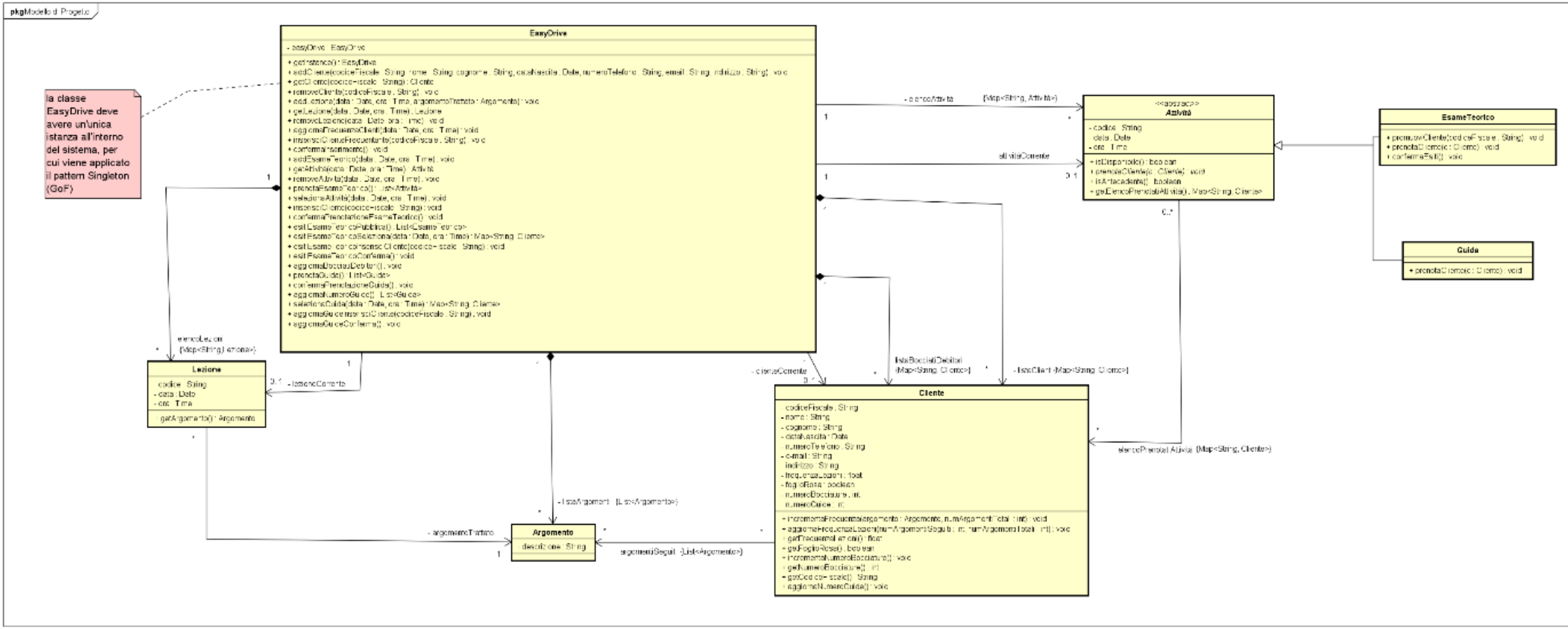
- **Aggiorna Guide Inserisci Cliente**



- **Aggiorna Guide Conferma**



2.3.2 Diagramma delle Classi



2.4 Implementazione

Il codice è stato scritto in linguaggio Java utilizzando l'IDE Eclipse ed il framework di testing JUnit.

2.5 Test

Per verificare che i metodi e le classi da noi implementate siano funzionanti abbiamo creato dei test automatizzati. Segue un elenco puntato con la descrizione della metodologia di testing utilizzati per la seconda iterazione:

- **EasyDrive (testEasyDrive.java)**:
  - **testAddGuida:** Per prima cosa aggiungiamo tramite il metodo "*addGuida*" 3 guide specificando la loro data e ora. Successivamente ci facciamo tornare l'elenco di attività e se tale elenco è vuoto vuol dire che non vi è nessuna guida inserita in lista, in caso contrario vengono stampate le guide inserite in lista (in questo caso 3). Verifichiamo, inoltre, la correttezza anche tramite l'utilizzo di "*AssertNotNull*" e "*AssertNull*", passando rispettivamente prima una data di un esame esistente e poi la data di un esame non esistente.
  - **testPrenotaGuida:** Inizialmente aggiungiamo 3 guide tramite il metodo "*addGuida*", successivamente creiamo un ArrayList di tipo guida (nominato "*guideDisponibili*") e chiameremo il metodo "*prenotaGuida*" per riempire proprio quest'ultimo. Vedremo che una volta chiamato questo metodo, "*guideDisponibili*" avrà all'interno solo la data successiva alla data odierna e al suo interno non vi saranno quelle con date antecedenti.
  - **testConfermaPrenotazioneGuida:** In questo test prima di tutto aggiungiamo una guida, un cliente e un *esameTeorico* utilizzando rispettivamente il metodo "*addGuida*", "*addCliente*" e "*addEsameTeorico*". Faremo, inoltre, in modo che la frequenza delle lezioni sia tale da poter permettere la prenotazione all'esame teorico (>70%). Subito dopo faremo una prova per verificare che se viene selezionata un'attività con data non registrata nel sistema verremo avvisati. Invece, nel momento in cui selezioniamo un'attività con data presente nel sistema saremo in grado di poter prenotare il cliente passandogli il codice fiscale del cliente che vogliamo prenotare tramite il metodo "*inserisciCliente*"; tuttavia, viene dimostrato come il cliente non può essere prenotato se non ha ancora superato l'esame teorico, invece, se facciamo la stessa procedura e il cliente ha superato l'esame teorico è possibile prenotarlo. In particolare, viene mostrato come in quest'ultimo caso se viene inserito erroneamente un cliente non registrato nel sistema ci verrà restituito un messaggio di avviso e non potremo procedere con la prenotazione, mentre se eseguiamo tutta la procedura per bene, inserendo anche il codice fiscale di un cliente davvero registrato nel sistema e che rispetti tutti i vincoli, sarà possibile prenotarlo alla guida e verrà stampato l'elenco dei prenotati.
  - **testAggiornaNumeroGuide:** Innanzitutto aggiungiamo 3 guide nel sistema, tra cui una con una data successiva a quella odierna, dunque non verrà inserita all'interno dell'ArrayList "*guideDisponibili*". Nota bene, in questo caso "*guideDisponibili*" ha esattamente il significato opposto a quello che aveva prima, infatti verrà riempito dalle guide con data ANTECEDENTE a quella odierna. Verrà chiamato il metodo "*aggiornaNumeroGuide*" e in questo modo verrà riempito l'ArrayList. Successivamente verranno stampate solo le guide disponibili, che per l'appunto sono antecedenti, dunque ne verranno stampate solo 2.

- **testSelezionaGuida:** Prima di tutto aggiungiamo una guida, un cliente e un esameTeorico utilizzando rispettivamente il metodo "*addGuida*", "*addCliente*" e "*addEsameTeorico*". Faremo, inoltre, in modo che la frequenza delle lezioni sia tale da poter permettere la prenotazione all'esame teorico (>70%). A questo punto prenotiamo e facciamo superare l'esame teorico al cliente in modo da poter prenotare una guida. Dopo di che viene creata un HashMap di nome "prenotati" e verrà verificato prima di tutto che se inseriamo una data di una guida non registrata nel sistema verremo avvertiti, mentre se inseriamo una data corretta allora potremo ricavare l'elenco dei prenotati a tale guida. Inoltre, ci aspettiamo che la guida selezionata diventi *attivitàCorrente* e che l'hashmap "*prenotati*" non sia vuoto verificando entrambe le cose tramite l'utilizzo degli *assertNotNull*.
- **testAggiornaGuideInserisciCliente:** Prima di tutto aggiungiamo una guida, un cliente e un *esameTeorico* utilizzando rispettivamente il metodo "*addGuida*", "*addCliente*" e "*addEsameTeorico*". Faremo, inoltre, in modo che la frequenza delle lezioni sia tale da poter permettere la prenotazione all'esame teorico (>70%). A questo punto prenotiamo e facciamo superare l'esame teorico al cliente in modo da poterlo prenotare ad una guida. Una volta fatto ciò selezioniamo una guida alla quale far partecipare il cliente e chiamiamo il metodo "*aggiornaNumeroGuideInserisciCliente*" e, nel caso in cui venga selezionato un cliente non registrato nel sistema verremo avvisati, altrimenti verrà selezionato il relativo cliente che ci aspettiamo diventi *clienteCorrente*.
- **testAggiornaGuideConferma:** Anche in quest'ultimo caso, prima di tutto aggiungiamo una guida, un cliente e un *esameTeorico* utilizzando rispettivamente il metodo "*addGuida*", "*addCliente*" e "*addEsameTeorico*". Faremo, inoltre, in modo che la frequenza delle lezioni sia tale da poter permettere la prenotazione all'esame teorico (>70%). A questo punto prenotiamo e facciamo superare l'esame teorico al cliente e, una volta fatto ciò, lo prenotiamo ad una guida. Successivamente procediamo con l'aggiornamento del numero di guide effettuate, selezioniamo la guida, chiamiamo il metodo "*aggiornaGuideInserisciCliente*" al quale passiamo il codice fiscale e ci aspettiamo che il cliente selezionato diventi *clienteCorrente*. In tal modo chiamando il metodo "*aggiornaGuideConferma*" saremo in grado di incrementare il numero di guide del cliente inserito. Per verificare ciò effettuiamo un *assert* e stampiamo tutti i dati del cliente selezionato.

- **Cliente(TestCliente.java):**
  - **testAggiornaNumeroGuide:** Inizialmente stampiamo in console l'attributo "*numeroGuide*" dell'oggetto Cliente c, ci aspettiamo un valore pari a zero. Successivamente chiamiamo il metodo *"aggiornaNumeroGuide"* della classe Cliente e stampiamo nuovamente in console l'attributo. Ci accorgeremo che ogni volta che verrà chiamato tale metodo, il numero di bocciature sarà incrementato.
- **Guida(TestGuida.java):**
  - **testIsDisponibile:** Chiamiamo il metodo *"isDisponibile"* della classe *Guida*, se tutto è andato a buon fine ci aspettiamo che il metodo ritoni *false* poiché la data dell'oggetto di tipo *Guida* chiamante è antecedente a quella odierna.
  - **testPrenotaCliente:** Per prima cosa creiamo un oggetto di tipo *Cliente*, il quale non avrà ancora ottenuto il foglio rosa, e lo passiamo come parametro nella funzione *"prenotaCliente"* della classe *Guida*. Poiché è necessario il possesso del foglio rosa per poter effettuare le guide, ci aspettiamo che la prenotazione non vada a buon fine e che quindi l'elenco dei prenotati alla guida rimanga vuoto. In un secondo momento

settiamo l'attributo *foglioRosa* del cliente a *true* e richiamiamo il metodo "*penotaCliente*" della classe Guida, in questo caso ci aspettiamo che la prenotazione vada a buon fine e che l'elenco dei prenotati alla guida non sia vuoto.

- **TestIsAntecedente:** Chiamiamo il metodo *"isAntecedente"* della classe *Guida*, se tutto è andato a buon fine ci aspettiamo che il metodo ritoni *true* poiché la guida ha una data antecedente al momento della chiamata della funzione.