Fase di Elaborazione
Iterazione 1

Sommario

1. Analisi

1.1. Introduzione

Per l'iterazione 1, si è scelto di attenzionare i seguenti requisiti:

- Lo scenario principale di successo del caso d'uso UC2 (*Gestisci Proiezioni*) e alcuni dei controlli richiesti nella specifica dei requisiti ovvero:
    - eventuali sovrapposizioni tra la nuova proiezione e le altre già presenti,
    - la nuova proiezione deve avvenire in una data futura,
    - vincolo sulla tipologia di proiezione (2D/3D) e la sala scelta.

  Tali vincoli sono stati attenzionati maggiormente a livello di implementazione.
- Lo scenario principale di successo del caso d'uso UC11 (*Effettua Prenotazione*). Non vengono considerate le promozioni, le varie tipologie di sale e il concetto di disabilità (sia a livello di cliente che dei posti in sala); Viene effettuato il controllo sul credito del cliente e l'acquisto sarà possibile entro i 15 minuti successivi all'inizio della proiezione (infatti questo intervallo di tempo è riservato alla pubblicità).
- Caso d'uso d'avviamento per supportare i requisiti di inizializzazione dell'iterazione;
- Non viene attenzionato per il momento l'aspetto legato all'autenticazione.
- Dati solo in memoria principale.

In questo capitolo viene descritta l'analisi svolta nell'iterazione 1, considerando separatamente i due casi d'uso di interesse.

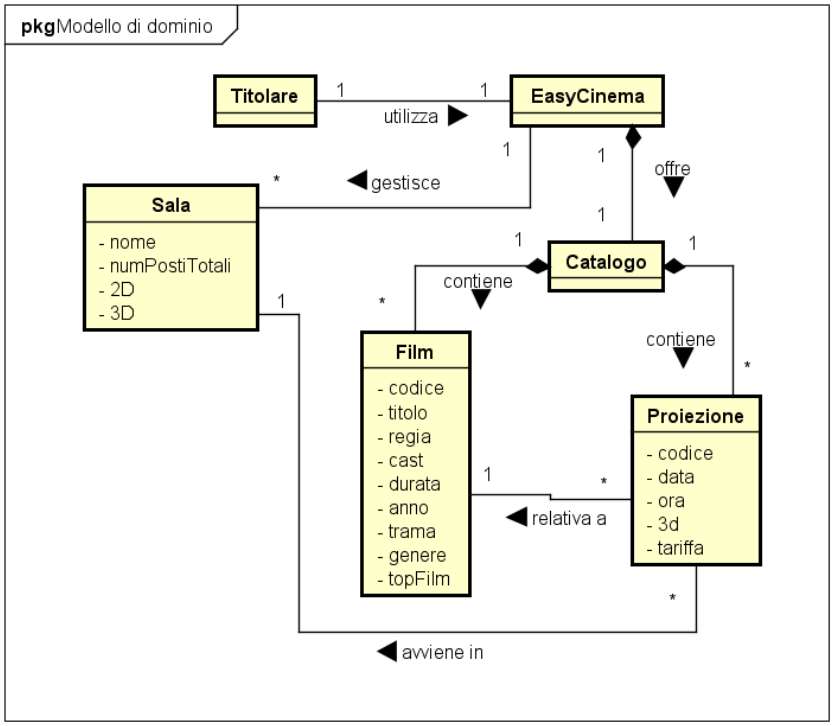
NOTA: tutti i digrammi UML di seguito riportati sono stati realizzati mediante Astah UML.

1.2. Caso d'uso UC2, Modello di dominio

Le classi concettuali del dominio del problema limitatamente ai requisiti dell'iterazione corrente sono:

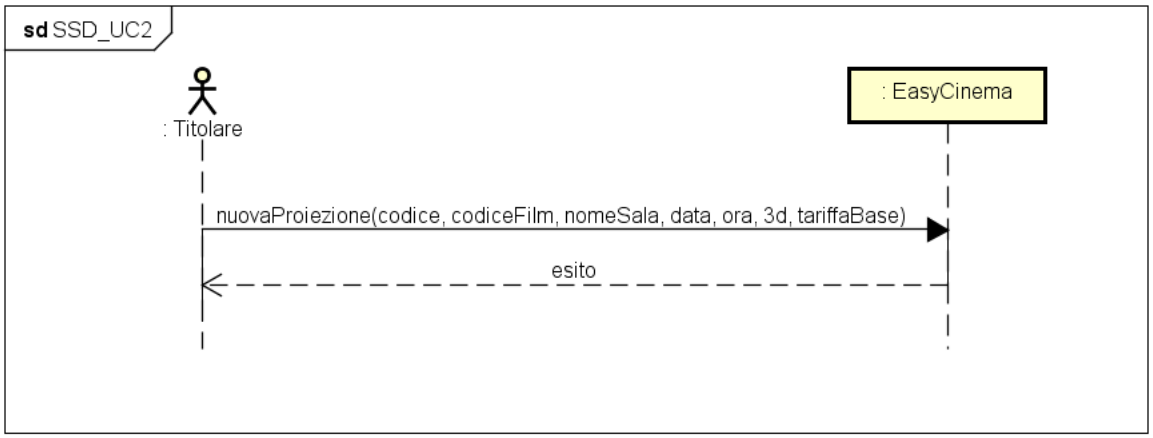
- *Titolare*: attore primario, interagisce direttamente con il sistema.
- *EasyCinema*: rappresenta il sistema Easy Cinema;
- *Catalogo*: contiene le informazioni sui film e sulle proiezioni esistenti;
- *Sala*: luogo in cui avviene la proiezione.
- *Proiezione*: una proiezione di un film che avviene in una sala ben precisa.
- *Film*: rappresenta l'oggetto della proiezione.

Tali classi e le loro relazioni costituiscono il modello di dominio. Esso è un elaborato della disciplina della *modellazione del business* in UP ed è rappresentato mediante il diagramma delle classi di UML utilizzato secondo il punto di vista concettuale e riportato di seguito.



1.3. Caso d'uso UC2, Diagramma di sequenza di sistema

Il diagramma di sequenza di sistema riporta gli eventi di input e output del sistema in discussione al fine di descrivere che cosa sa fare il sistema ovvero qual è il suo comportamento e attraverso l'interazione con quali attori (primari e di supporto) esso è definito. Tale artefatto rientra nel Modello dei casi d'uso.



1.4. Caso d'uso UC2, Contratti delle operazioni

I contratti delle operazioni di sistema permettono di descrivere il comportamento del sistema in modo più dettagliato. Sono complementari ai casi d'uso e sono utili per la descrizione delle operazioni più complesse o meno chiare.

Guardando al SSD sopra riportato viene definito il seguente contratto:

1.4.1. Nuova proiezione

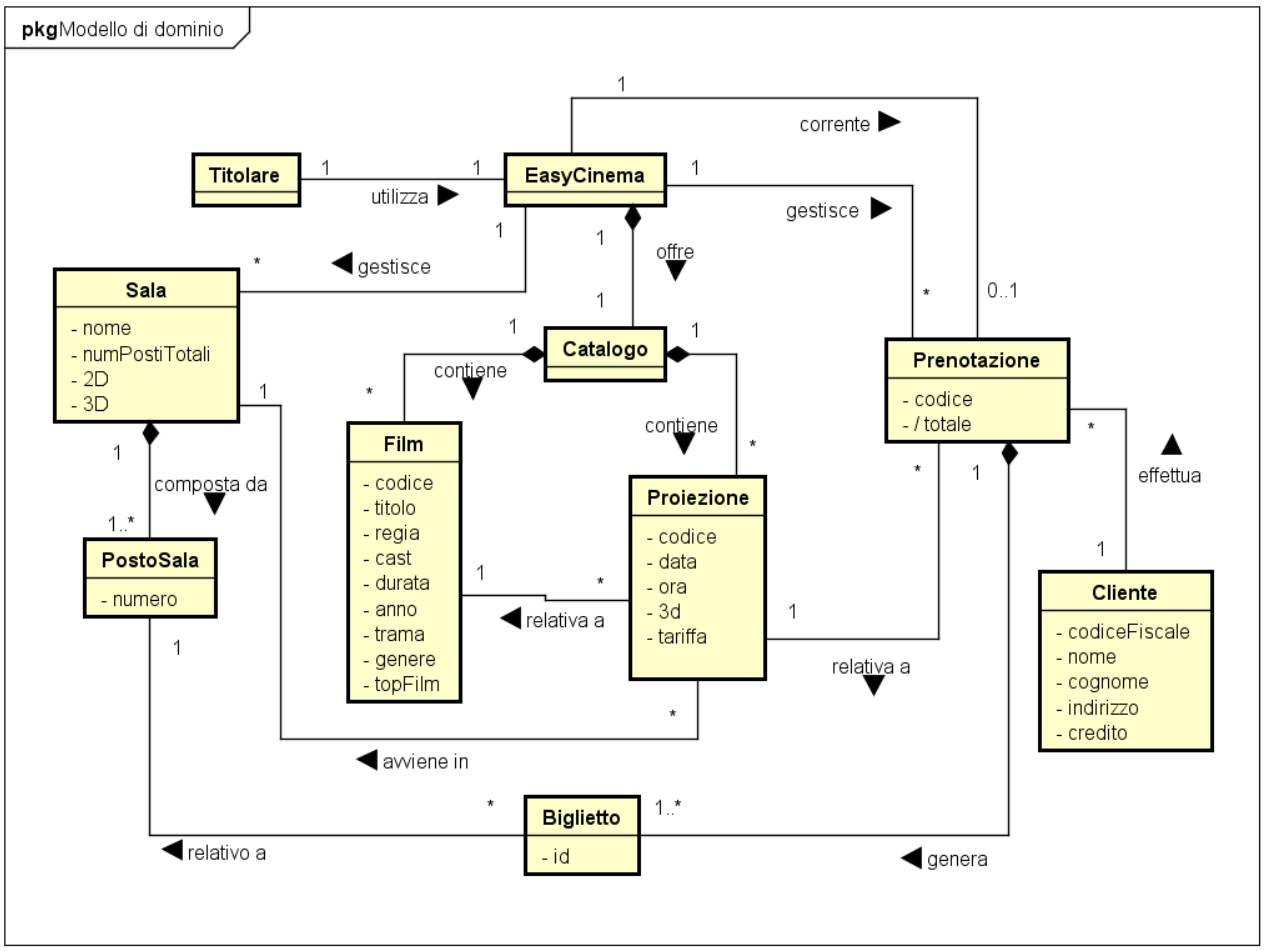
Operazione	nuovaProiezione(codice:String, codiceFilm:String, nomeSala:String, data:Date, ora:Time, 3d:boolean, tariffaBase:double)
Riferimenti	Caso d'uso: Gestisci Proiezioni
Pre-condizioni	- il Titolare t sta utilizzando il sistema
Post-condizioni	- è stata creata un'istanza pr di Proiezione; - pr è stata associata con il Catalogo tramite l'associazione "contiene"; - pr è stata associata con un Film, in base alla corrispondenza con codiceFilm; - pr è stata associata con una Sala, in base alla corrispondenza con nomeSala; - gli attributi di pr sono stati inizializzati.

1.5. Caso d'uso UC11, Modello di dominio

Analizzando lo scenario principale di successo del caso d'uso UC11 (Gestisci Proiezioni) occorre introdurre le seguenti classi concettuali:

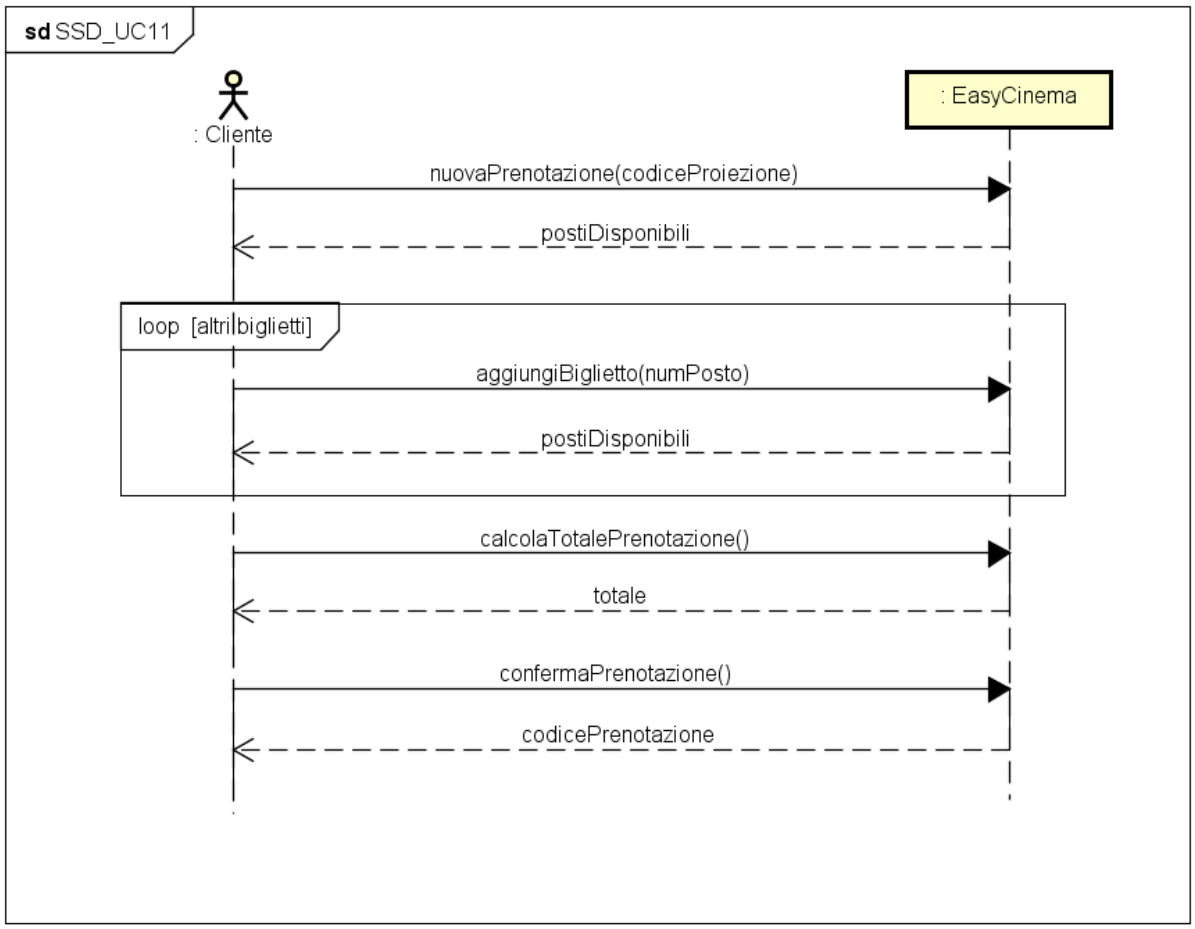
- *Cliente*: attore primario, che interagisce direttamente con il sistema;
- *PostoSala*: rappresenta un singolo posto a sedere all'interno della sala cinematografica.
- *Prenotazione*: rappresenta la modalità attraverso la quale il cliente può prendere parte ad una proiezione.
- *Biglietto*: risultato della prenotazione, assicura un posto per una proiezione.

Otteniamo in tal modo il seguente modello di dominio (comprensivo delle considerazioni fatte per il caso d'uso UC2):



1.6. Caso d'uso UC11, Diagramma di sequenza di sistema

Il diagramma di sequenza di sistema per lo scenario principale di successo del caso d'uso UC11 è riportato di seguito.



1.7. Caso d'uso UC11, Contratti delle operazioni

Guardando al SSD di cui sopra vengono definiti i seguenti contratti:

1.7.1. Nuova prenotazione

Operazione	nuovaPrenotazione(codiceProiezione:String)
Riferimenti	Caso d'uso: Effettua Prenotazione
Pre-condizioni	- Il cliente c sta utilizzando il sistema
Post-condizioni	- è stata creata un'istanza p di Prenotazione - p è stata associata con il Cliente c - p è stata associata con una Proiezione pr, in base alla corrispondenza con codiceProiezione

	- p è stata associata a EasyCinema tramite l'associazione "corrente" - l'attributo codice dell'istanza p è stato inizializzato

1.7.2. Aggiungi biglietto

Operazione	aggiungiBiglietto(numPosto:int)
Riferimenti	Caso d'uso: Effettua Prenotazione
Pre-condizioni	- È in corso una Prenotazione p.
Post-condizioni	- è stata creata un'istanza b di Biglietto - b è stata associata con un PostoSala (tra quelli che fanno parte della Sala s in cui avverrà la Proiezione p), in base alla corrispondenza con numPosto - b è stata associata alla Prenotazione corrente p - l'attributo id dell'istanza b è stato inizializzato

1.7.3. Conferma prenotazione

Operazione	confermaPrenotazione()
Riferimenti	Caso d'uso: Effettua Prenotazione
Pre-condizioni	- È in corso una Prenotazione p.
Post-condizioni	- è stata associata l'istanza p di Prenotazione corrente a EasyCinema tramite l'associazione gestisce. - L'attributo credito del Cliente c è stato modificato di un valore pari al totale della Prenotazione.

2. Progettazione

2.1. Introduzione

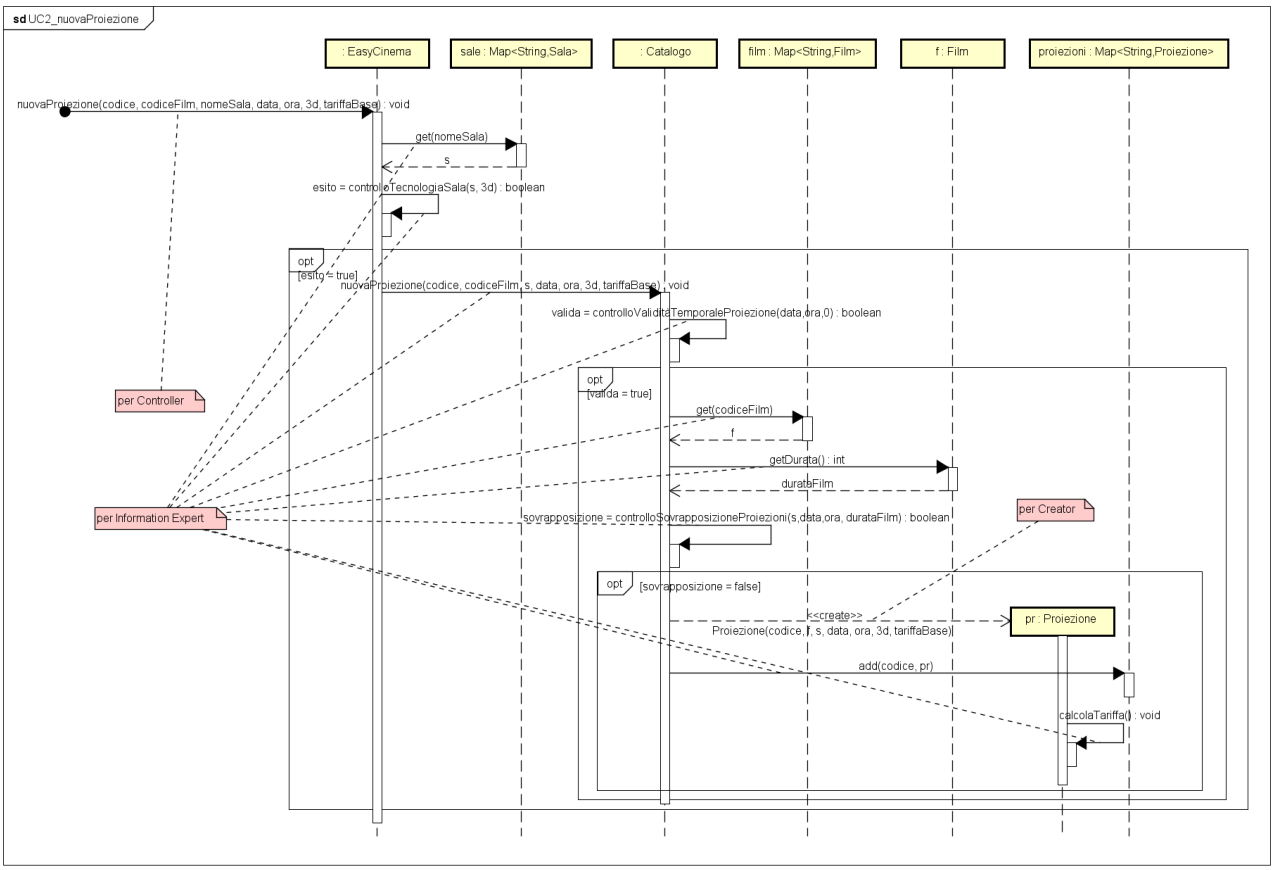
L'architettura software adottata è un'architettura a strati e il focus del progetto verte sullo sviluppo dello strato della logica applicativa. Segue dunque la progettazione a oggetti per lo strato del dominio distinguendo tra modellazione statica (diagramma delle classi) e dinamica (diagrammi di interazione). Tali diagrammi nella loro totalità realizzano il *Modello di Progetto* elaborato fondamentale per la disciplina di *Design* in UP.

Poiché le operazioni di sistema di entrambi i casi d'uso non sono molte si sceglie di usare un facade controller, la classe scelta a tal fine è EasyCinema in quanto rappresenta il sistema complessivo (Pattern GRASP *Controller*). Sarà essa a ricevere e coordinare le operazioni di sistema.

2.2. Caso d'uso UC2, Diagrammi di interazione

I diagrammi di interazione illustrano come gli oggetti collaborano per realizzare uno scenario di caso d'uso.

2.2.1. nuovaProiezione(codice:String, codiceFilm:String, nomeSala:String, data:Date, ora:Time, 3d:boolean, tariffaBase:double)



Guardando al contratto dell'operazione di sistema notiamo la necessità di creare un'istanza di Proiezione. Analizzando il modello di dominio ci si accorge che Catalogo contiene istanze di Proiezione e quindi per Low

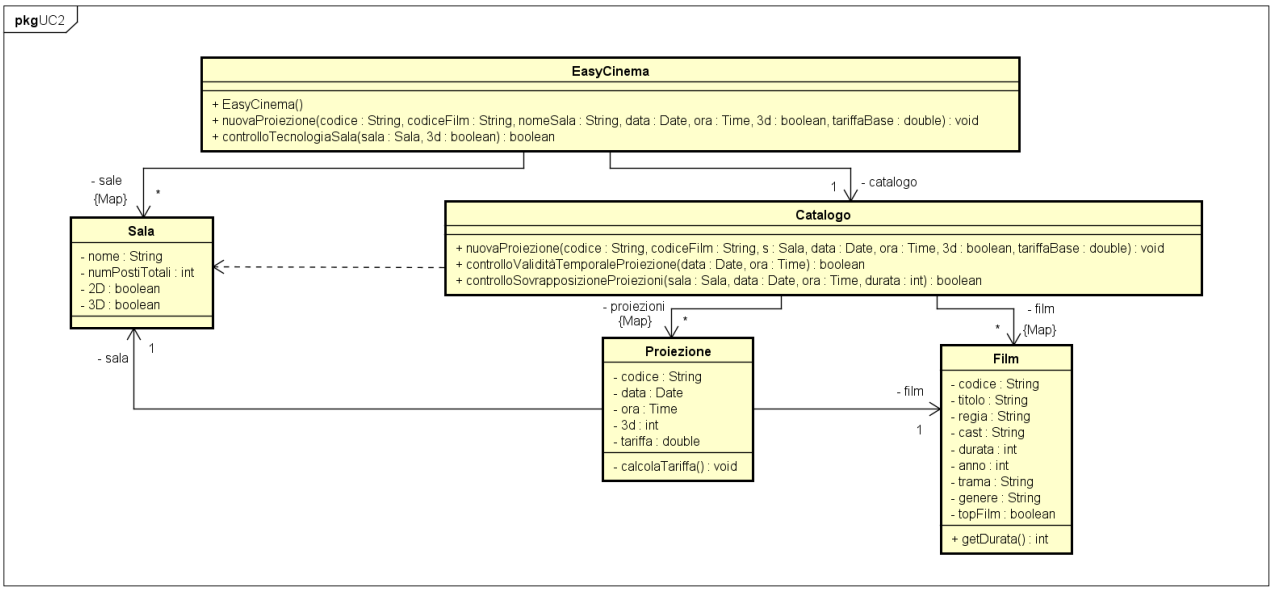
Representational Gap si aggiunge la classe software Proiezione e applicando il pattern GRASP *Creator* si giunge alla decisione di assegnare proprio a Catalogo la responsabilità di creare tali istanze.

Continuando a guardare al modello di dominio e utilizzando il pattern GRASP *Information Expert* si può constatare che la responsabilità di ricercare una Sala dato il suo nome è da assegnare a EasyCinema in quanto gestisce tutte le sale. Per quanto riguarda invece la ricerca del Film a partire dal suo codice è da assegnare a Catalogo in quanto contiene tutti i film.

Sono state aggiunte delle operazioni di controllo per verificare la correttezza delle informazioni inserite.

2.3. Caso d'uso UC2, Diagramma delle classi di progetto

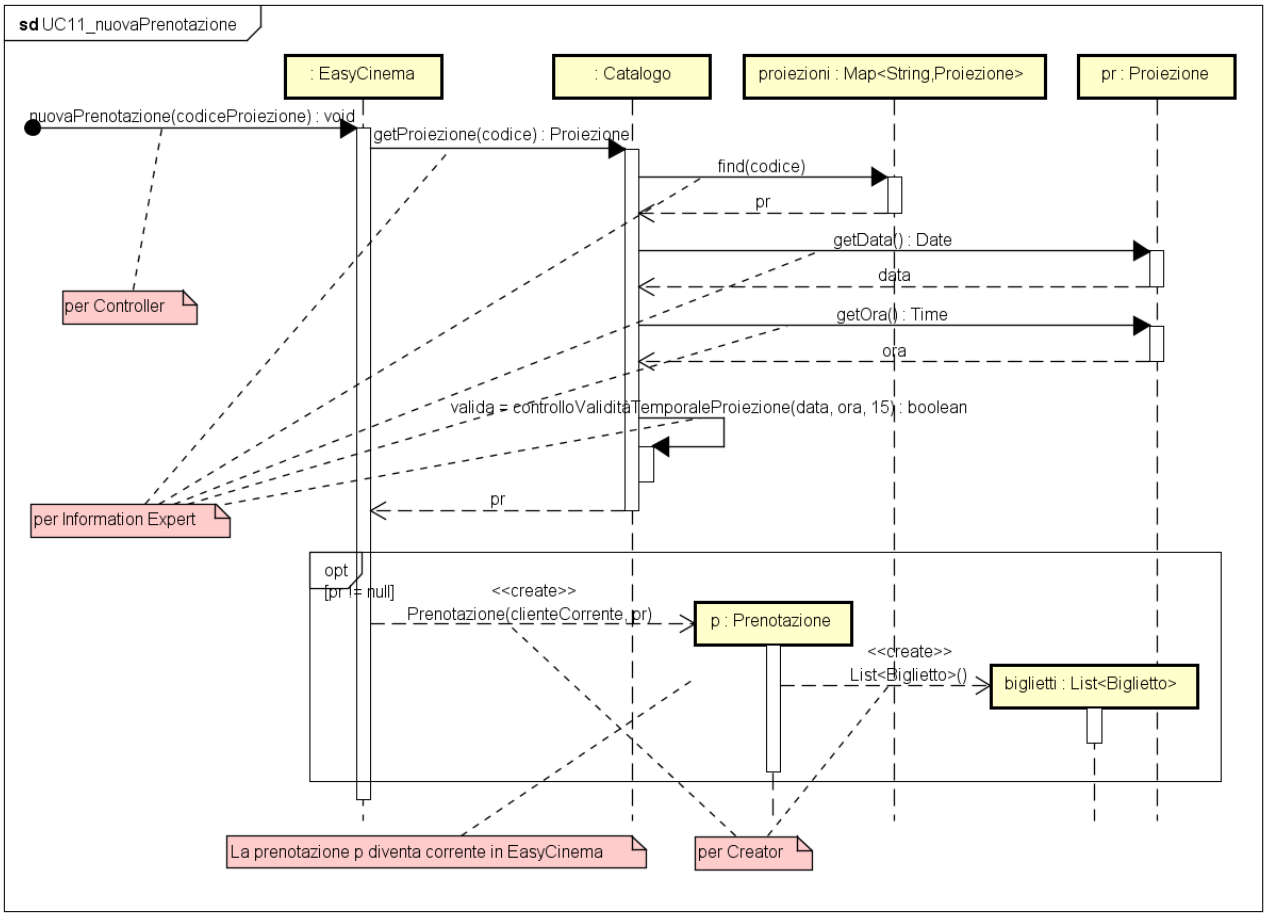
La modellazione statica avviene attraverso i diagrammi delle classi di progetto ovvero diagrammi delle classi di UML utilizzati non più dal punto di vista concettuale bensì della specifica. Le responsabilità assegnate nella progettazione statica vengono adesso sintetizzate nella modellazione statica andando ad illustrare le caratteristiche comportamentali e strutturali delle classi software.



Per motivi di spazio non viene riportato il costruttore con attributi della classe Film (che sarà invece presente nel DCD complessivo più avanti riportato).

2.4. Caso d'uso UC11, Diagrammi di interazione

2.4.1. nuovaPrenotazione(codiceProiezione:String)

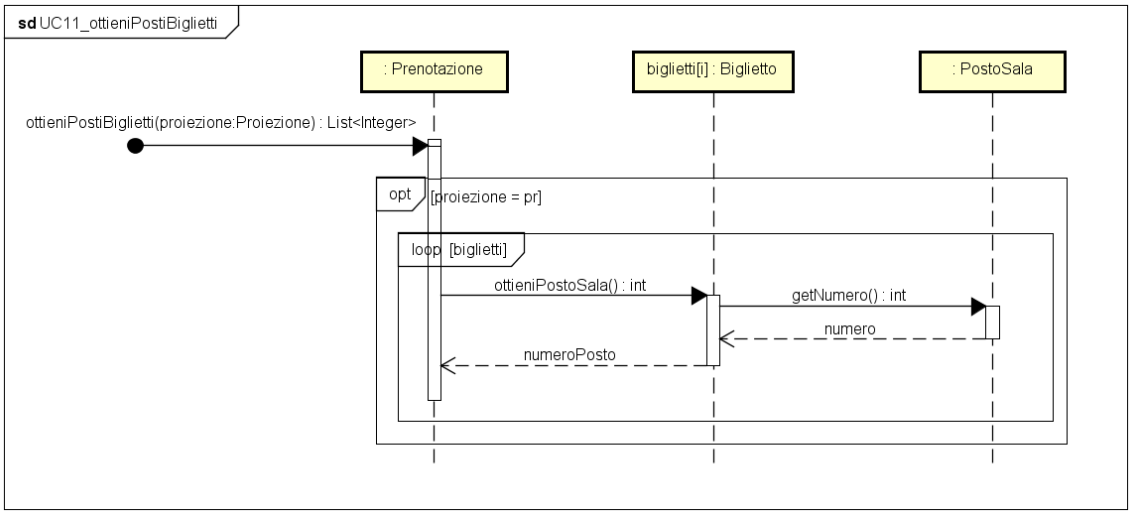


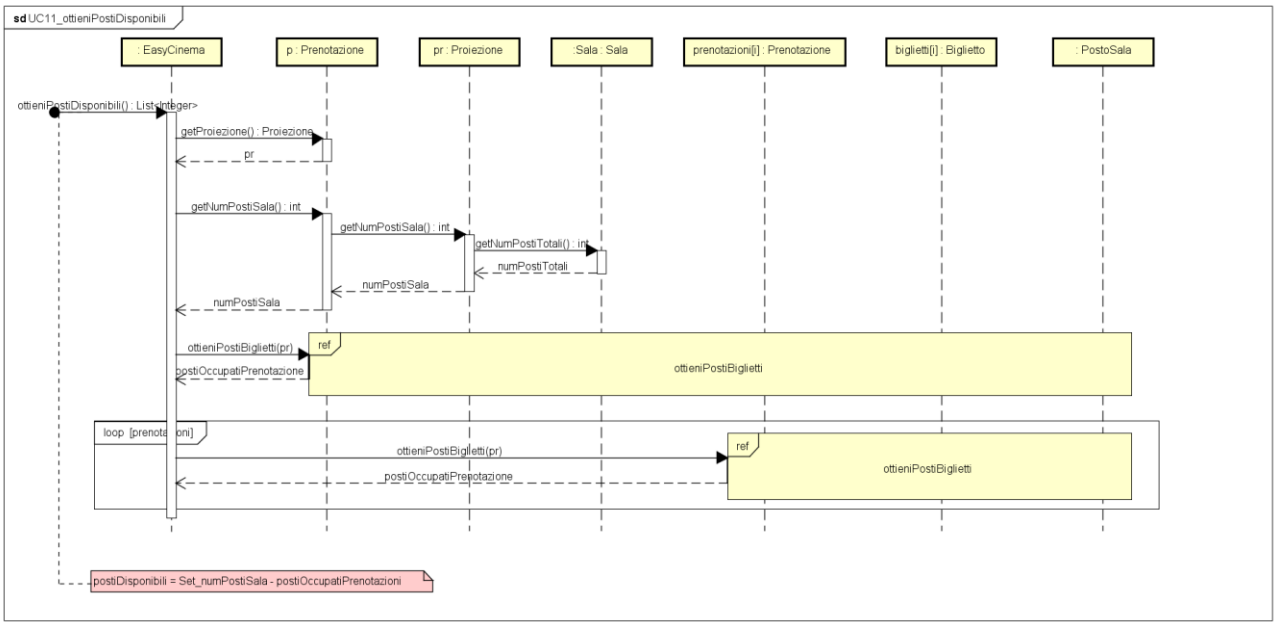
Attenzionando il contratto di tale operazione di sistema notiamo il bisogno di creare un'istanza di Prenotazione, questa responsabilità per il pattern *Creator* spetta a EasyCinema in quanto gestisce le prenotazioni e in particolare l'istanza di Prenotazione p diverrà corrente per EasyCinema. La prenotazione appena creata deve a sua volta occuparsi di creare la struttura dati relativa ai Biglietti che utilizzerà successivamente. Per quanto riguarda la ricerca della Proiezione sulla base del suo codice per *Information Expert* se ne occupa Catalogo, il quale conosce tutte le proiezioni presenti. Infine, per associare p con il particolare Cliente occorre recuperare l'Utente che sta utilizzando il sistema software e tale responsabilità è affidata a EasyCinema.

2.4.2. ottieniPostiDisponibili()

Questa non è un'operazione di sistema bensì un'interrogazione al fine di determinare quali sono i posti disponibili all'interno della sala in cui avrà luogo la proiezione corrente.

Poiché la rappresentazione è piuttosto ampia e l'operazione di ricerca dei posti relativi a ciascuna prenotazione viene ripetuta più volte si è scelto di realizzare un primo diagramma di interazione che verrà referenziato all'interno di un altro diagramma di interazione creando così una correlazione tra i due.

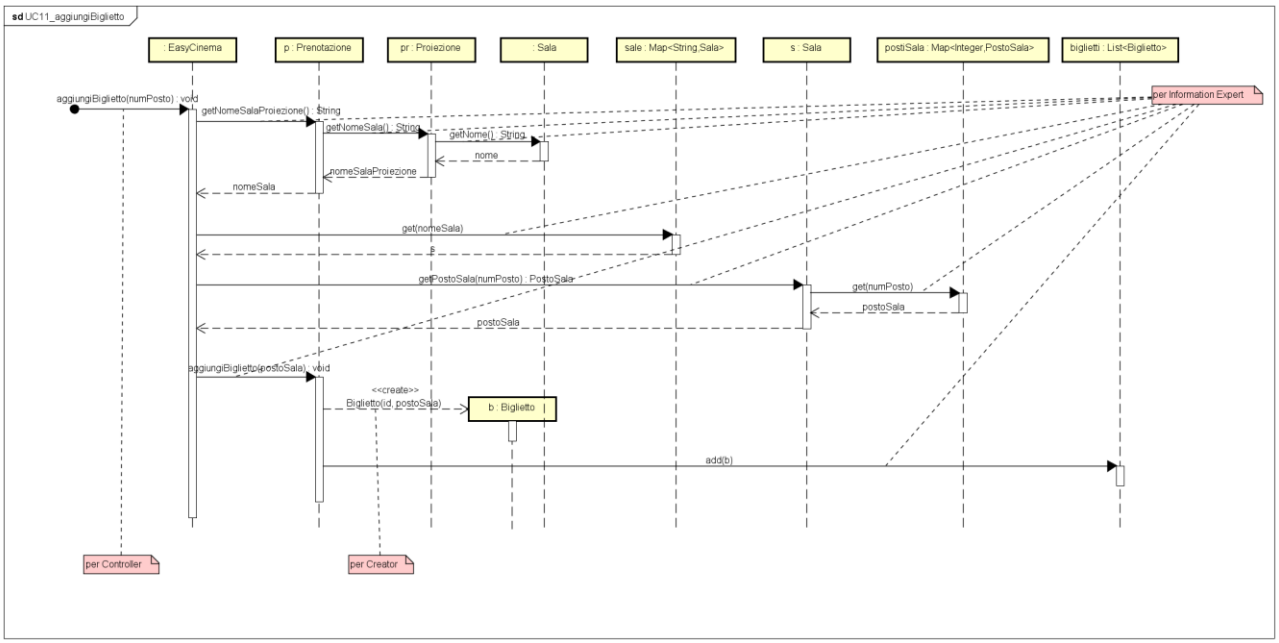




La classe software EasyCinema che gestisce le varie prenotazioni si occupa di risalire ai posti occupati e sottraendoli a quelli liberi ottiene i posti disponibili.

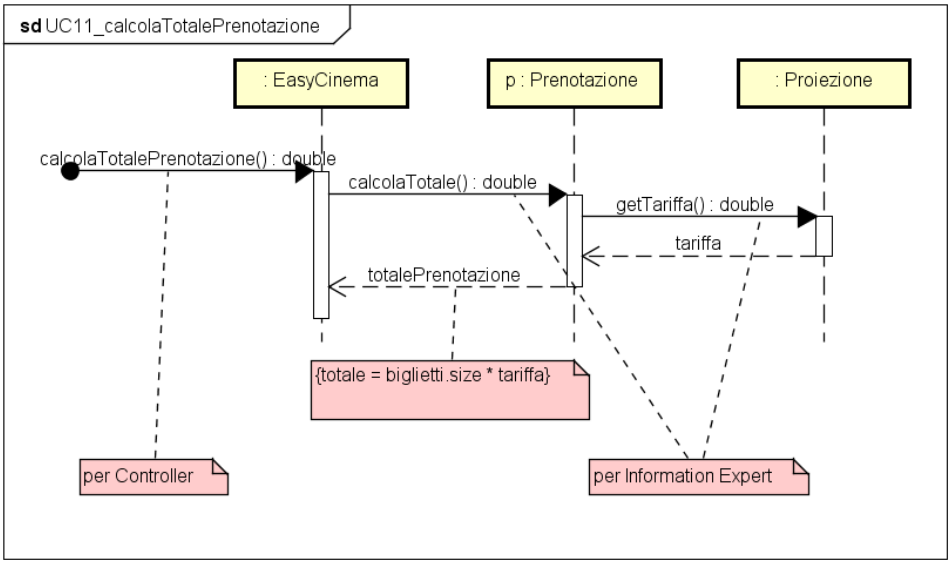
Oltre a tenere conto dei posti riservati dalle altre prenotazioni occorre considerare anche quelli momentaneamente riservati dalla prenotazione corrente e che diverranno definitivi una volta confermata la prenotazione.

2.4.3. aggiungiBiglietto(numPosto: Integer)



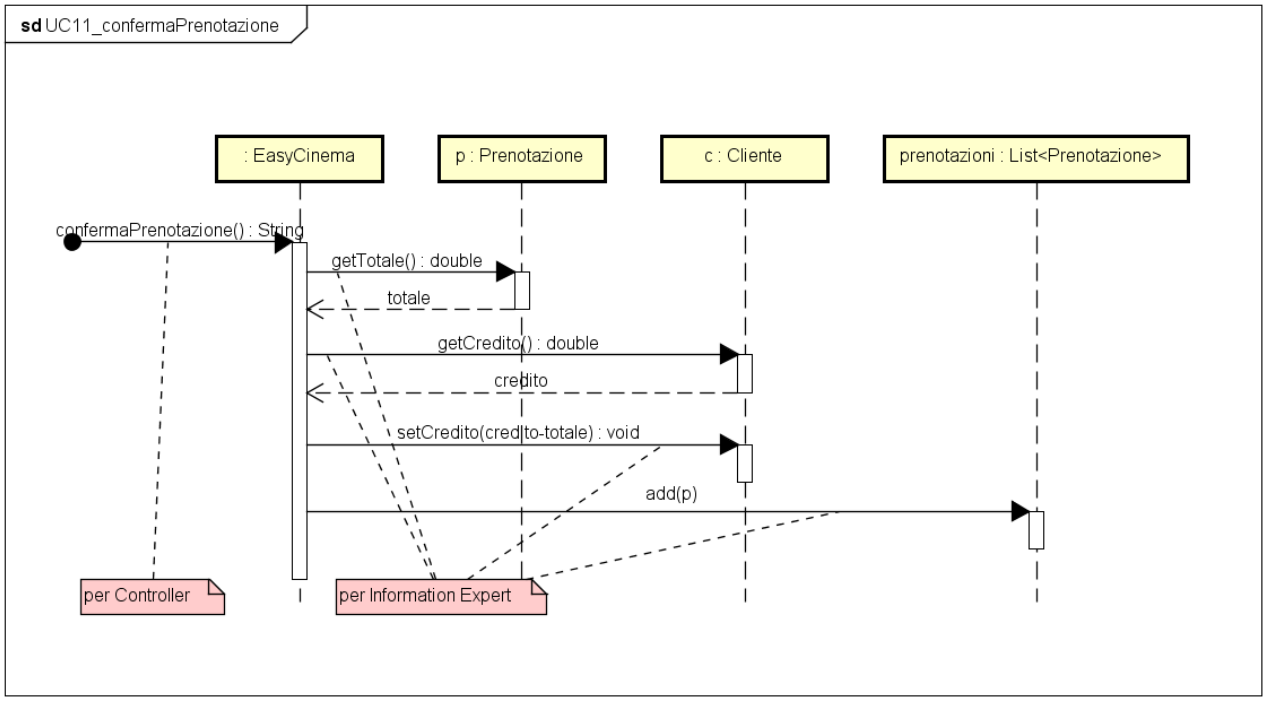
Guardando alle post-condizioni del contratto relative all'operazione di sistema occorre creare un'istanza b di Biglietto, tale responsabilità viene assegnata a Prenotazione per il pattern *Creator* in quanto aggrega i biglietti relativi ad una prenotazione.
Inoltre, è necessario associare il Biglietto b alla particolare istanza di PostoSala la quale è, a sua volta, associata alla Sala in cui ha luogo la Proiezione a cui la Prenotazione corrente fa riferimento. Per fare ciò occorre recuperare l'istanza in questione e tale responsabilità è stata assegnata, per Information Expert, a EasyCinema che gestisce le sale.

2.4.4. calcolaTotalePrenotazione()



La responsabilità di calcolare il totale della Prenotazione corrente per il pattern GRASP *Information Expert* spetta proprio a Prenotazione in quanto conosce la Proiezione a cui fa riferimento e da cui ottiene la tariffa e il numero di biglietti, informazioni necessarie per ottenere il risultato finale.

2.4.5. confermaPrenotazione()



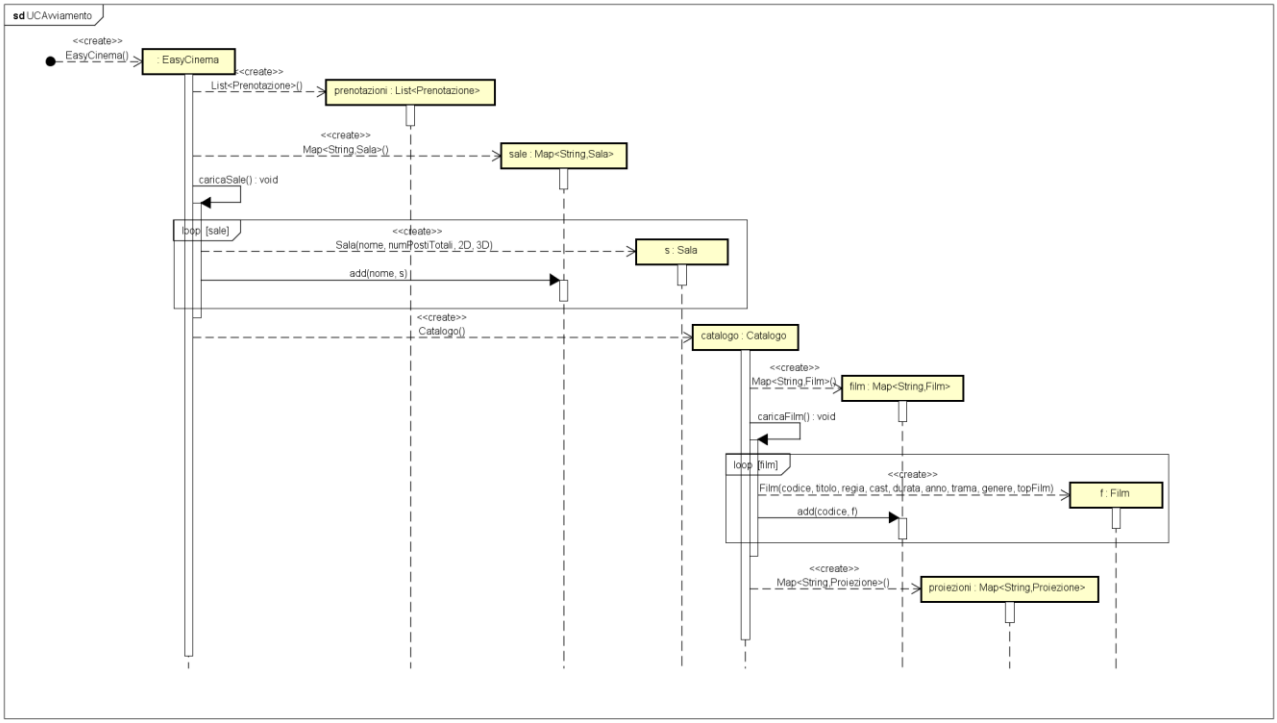
EasyCinema gestisce le prenotazioni e dunque sarà esso ad aggiungere alla sua collezione la Prenotazione corrente rendendola definitiva. Occorre prima però controllare il credito del cliente, tale informazione è contenuta all'interno di Cliente.

2.5. Caso d'uso di avviamento, Diagramma di interazione

Per concludere la modellazione della progettazione OO, dopo aver compreso cosa occorre creare ed inizializzare dalle altre realizzazioni di scenario di caso d'uso, si considera la progettazione delle inizializzazioni per l'avviamento.

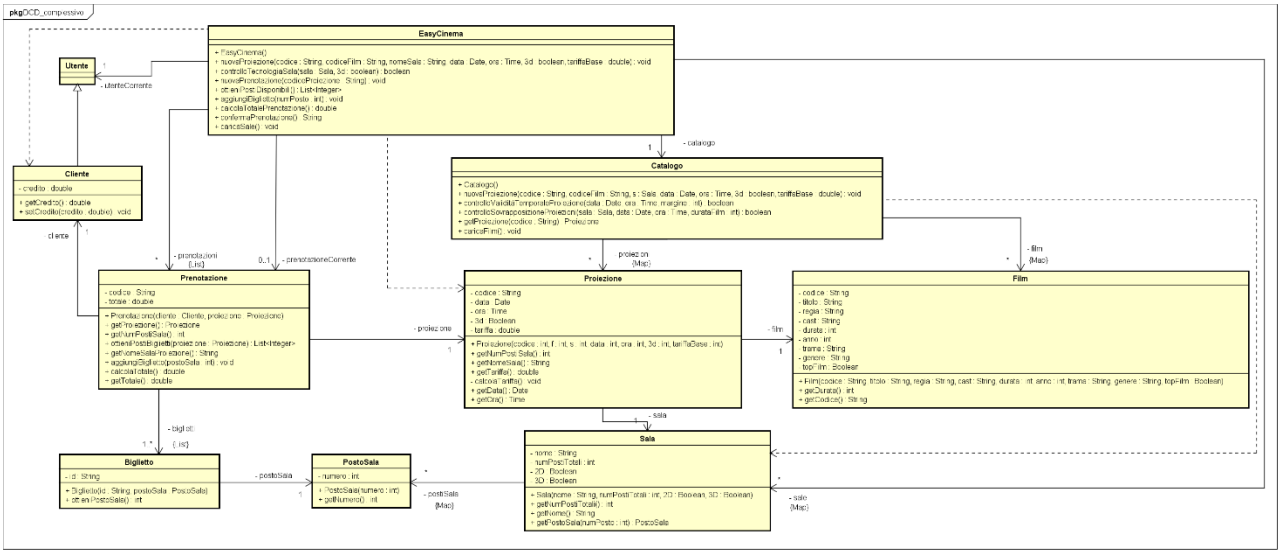
L'*oggetto di dominio iniziale* ovvero il primo oggetto software a livello di dominio ad essere creato, responsabile della creazione di altri oggetti del dominio è stato individuato nella figura del facade controller EasyCinema in quanto oggetto radice rispetto alla gerarchia di aggregazione degli oggetti del dominio.

Il diagramma di interazione per l'operazione di avviamento mostra cosa succede quando l'oggetto di dominio iniziale viene creato ed è di seguito riportato.



2.6. Diagramma delle classi complessivo

Segue il diagramma delle classi complessivo ottenuto dalle considerazioni relative agli scenari dei casi d'uso analizzati in questa iterazione.



3. Implementazione

A partire dai diagrammi di interazione e dal diagramma delle classi complessivo si procede ad implementare il progetto software per l'iterazione corrente.

Per l'implementazione del sistema software si è fatto uso dell'IDE Eclipse, del VCS Git e il progetto è ospitato su GitHub.

L'interfaccia utente adottata è una semplice interfaccia utente a caratteri basata su quella di "Colazione di Rosa".

4. Testing

È lo strumento principale per la verification del software. Si ricorre a criteri sistematici (esplorazioni mirate del dominio di input) al fine di ricercare i test cases tali da massimizzare la probabilità di rilevare errori presenti nel software.

Per realizzare il testing si è utilizzato il framework di unit testing per Java JUnit e il framework di mocking Mockito.

Progettazione dei casi di test:

- Cliente
    - **setCredito**(double credito): utilizzando la tecnica di copertura delle classi di equivalenza e in particolare ricorrendo al criterio del numero di valori si ottiene come classe valida quella per cui credito>=0 mentre quella non valida per credito <=0, applicando inoltre la tecnica di analisi dei valori estremi sull'input è stata considerata la condizione credito=0 direttamente sull'estremo. Per quanto riguarda invece il white-box testing le condizioni individuate sono sufficiente a garantire la copertura delle istruzioni.
      Per verificare che il credito è stato settato correttamente si ricorre al metodo getCredito.
      Le condizioni di interesse individuate sono:
        1. *testSetCreditoPositivo*
            - Input: credito positivo (9.5).
            - Output previsto: 9.5.
        2. *testSetCreditoNullo*
            - Input: credito nullo (0).
            - Output previsto: 0.
        3. *testSetCreditoNegativo*
            - Input: credito negativo (-10).
            - Output previsto: generazione eccezione e credito invariato.

- Sala
    - setNumPostiTotali(int numPostiTotali): sono state adottate le stesse tecniche utilizzate per setCredito, stavolta però, la classe di equivalenza valida è per numPostiTotali>0 mentre quella invalida per numPostiTotali<=0.
      Essendo il metodo in esame privato i test sono effettuati tramite il metodo pubblico che lo invoca ovvero il costruttore di Sala.

      Le condizioni di interesse individuate sono:
        1. *testSetNumPostiTotaliPositivo*
            - Input: numero positivo (1) - immediatamente sopra l'estremo.
            - Output previsto: 1.
        2. *testSetNumPostiTotaliNullo*
            - Input: numero nullo (0) - estremo.
            - Output previsto: generazione eccezione, istanza di Sala non viene creata.
        3. *testSetNumPostiTotaliNegativo*
            - Input: numero negativo (-1) - immediatamente sotto l'estremo.
            - Output previsto: generazione eccezione, istanza di Sala non viene creata.
    - creaPostiSala(): obiettivo del test è verificare che il numero di oggetti PostoSala creati sia pari al numero di posti totali della Sala. Anch'esso è un metodo privato e verrà testato passando dal metodo che lo invoca ovvero il costruttore.
      Il test in esame non è propriamente uno unit test in quanto subentra l'interazione con un altro componente ovvero PostoSala (creazione e interrogazione), essendo tuttavia le operazioni coinvolte elementari non si è ritenuto necessario introdurre uno stub rendendolo di fatto un test di integrazione.

      Vengono riutilizzate le stesse condizioni utilizzate per il setNumPostiTotali.

        1. *testCreaPostiSalaNumPostiTotaliPositivo*
            - Input: numero positivo (1) - immediatamente sopra l'estremo.
            - Output previsto: 1.
        2. *testCreaPostiSalaNumPostiTotaliNullo*
            - Input: numero nullo (0) - estremo.
            - Output previsto: generazione eccezione, istanza di Sala non viene creata.
        3. *testCreaPostiSalaNumPostiTotaliNegativo*
            - Input: numero negativo (-1) - immediatamente sotto l'estremo.
            - Output previsto: generazione eccezione, istanza di Sala non viene creata.

- Proiezione
    - setTariffa(double tariffaBase): una proiezione deve avere una tariffa positiva dunque possiamo individuare i test cases similmente a quanto visto finora ovvero un valore positivo, un valore nullo e un valore negativo di tariffaBase (black-box). Guardando inoltre alle specifiche si nota un comportamento diverso, nel caso di tariffa base positiva, a seconda che il film associato alla proiezione sia un top film e se la proiezione è 3D o meno (in entrambi i casi vi sarà un incremento del 15% della tariffa).
      Volendo realizzare uno unit test le classi Film e Sala, da cui Proiezione dipende, sono state implementate come mocks.

Essendo il metodo in esame privato i test sono effettuati tramite il metodo pubblico che lo invoca ovvero il costruttore di Proiezione.

Le condizioni di interesse individuate sono:

1. *testSetTariffaPositiva_NoProiezione3D_NoTopFilm*
   - Input: valore positivo (2.5); inoltre 3D=false, topFilm=false.
   - Output previsto: 2.5.
2. *testSetTariffaPositiva_Proiezione3D_NoTopFilm*
   - Input: valore positivo (3); inoltre 3D=true, topFilm=false.
   - Output previsto: 3.45.
3. *testSetTariffaPositiva_NoProiezione3D_TopFilm*
   - Input: valore positivo (9.6); inoltre 3D=false, topFilm=false.
   - Output previsto: 11.04.
4. *testSetTariffaPositiva_Proiezione3D_TopFilm*
   - Input: valore positivo (10); inoltre 3D=true, topFilm=true.
   - Output previsto: 13.23 (arrotondamento a due cifre decimali).
5. *testSetTariffaNulla*
   - Input: valore nullo (0).
   - Output previsto: generazione eccezione, istanza di Proiezione non viene creata.
6. *testSetTariffaNegativa*
   - Input: valore nullo (-1).
   - Output previsto: generazione eccezione, istanza di Proiezione non viene creata.

- Catalogo
  - controlloValiditaTemporaleProiezione (LocalDate data, LocalTime ora, int margine): una proiezione può essere programmata solo per una data futura. Da queste considerazioni il metodo verrà testato rispetto al minuto successivo, al minuto corrente e al minuto precedente a quello attuale. Il concetto di margine viene tralasciato dal test.
    Le condizioni di interesse individuate sono:
    1. *testControlloValiditaTemporaleProiezione*_Valida_MinutoSuccessivo
       - Input: minuto successivo rispetto alla data e ora corrente.
       - Output previsto: true.
    2. *testControlloValiditaTemporaleProiezione_NonValida_MinutoCorrente*
       - Input: stesso minuto della data e ora corrente.
       - Output previsto: false.
    3. *testControlloValiditaTemporaleProiezione_NonValida_MinutoPrecedente*
       - Input: minuto antecedente rispetto alla data e ora corrente.
       - Output previsto: false.

- controlloSovrapposizioneProiezione (LocalDate data, LocalTime ora, int durataFilm, Proiezione proiezioneEsistente): una nuova proiezione non può sovrapporsi temporalmente ad un'altra. Una proiezione ha un inizio (data e ora) e una fine (inizio + 15 minuti di pubblicità + durata del film). È stato generato un mock per la classe Proiezione. Ricorrendo al criterio di copertura delle istruzioni (white-box testing) e alla tecnica di analisi dei valori estremi sono stati individuati i seguenti scenari di interesse (il risultato atteso è true se vi è sovrapposizione, false altrimenti):

    1. *testControlloSovrapposizioneProiezione_InteramenteContenuta*
        - Input: la nuova potenziale proiezione è contenuta temporalmente da una proiezione esistente.
        - Output previsto: true.
    2. *testControlloSovrapposizioneProiezione_ParzialmenteContenuta*
        - Input: La nuova potenziale proiezione è contenuta parzialmente da una proiezione esistente.
        - Output previsto: true.
    3. *testControlloSovrapposizioneProiezione_ContieneInteramente*
        - Input: La nuova potenziale proiezione contiene temporalmente una proiezione esistente.
        - Output previsto: true.
    4. *testControlloSovrapposizioneProiezione_ContieneParzialmente*
        - Input: La nuova potenziale proiezione contiene parzialmente una proiezione esistente.
        - Output previsto: true.
    5. *testControlloSovrapposizioneProiezione_Coincidenza*
        - Input: La nuova potenziale proiezione coincide esattamente con una proiezione esistente.
        - Output previsto: true.
    6. *testControlloSovrapposizioneProiezione_NuovaImmediatamentePrima*
        - Input: La nuova potenziale proiezione termina immediatamente prima dell'inizio di una proiezione esistente.
        - Output previsto: false.
    7. *testControlloSovrapposizioneProiezione_NuovaImmediatamenteDopo*
        - Input: La nuova potenziale proiezione inizia immediatamente dopo la fine di una proiezione esistente.
        - Output previsto: false.

- Prenotazione
    - ottieniPostiBiglietti(Proiezione proiezione): obiettivo del metodo è restituire l'insieme dei posti occupati dalla Prenotazione per una specifica Proiezione. Sono state individuate le due condizioni possibili ovvero la prenotazione è relativa alla Proiezione o non lo è. Per la prima condizione si è considerato il caso estremo di un solo posto prenotato e un ulteriore caso con 3 posti.
      Le condizioni di interesse individuate sono:
        1. *testOttieniPostiBiglietti_StessaProiezione1Biglietto*

- Input: un biglietto per il posto 2 per la proiezione di interesse.
- Output previsto: [2].

2. *testOttieniPostiBiglietti_StessaProiezione3Biglietti*
   - Input: tre biglietti per i posti 2,40,17 per la proiezione di interesse.
   - Output previsto: [2,40,17].

3. *testOttieniPostiBiglietti_DiversaProiezione*
   - Input: un biglietto per il posto 2 per una proiezione diversa da quella di interesse.
   - Output previsto: [].

- EasyCinema

I metodi da testare di tale classe sono stati realizzati come Nested tests (JUnit5) in modo da avere un'organizzazione dei test migliore e un setup distinto laddove richiesto.

- controlloTecnologiaSala(Sala sala, boolean _3D): una proiezione (sia essa 2D o 3D) può avere luogo unicamente in una sala in grado di supportare tale tecnologia. È stato utilizzato un mock per la classe Sala e si è condizionato per ciascun test case il risultato dei suoi metodi is_2D e is_3D.
  Le condizioni di interesse individuate sono:

  1. *testControlloTecnologiaSala_Proiezione3DSalaAbilitata*
     - Input: Sala che supporta il 3D, true.
     - Output previsto: True.

  2. *testControlloTecnologiaSala_Proiezione3DSalaNonAbilitata*
     - Input: Sala che non supporta il 3D, true.
     - Output previsto: False.

  3. *testControlloTecnologiaSala_Proiezione2DSalaAbilitata*
     - Input: Sala che supporta il 2D, false.
     - Output previsto: True.

  4. *testControlloTecnologiaSala_Proiezione2DSalaNonAbilitata*
     - Input: Sala che non supporta il 2D, false.
     - Output previsto: False.

- nuovaPrenotazione(String codiceProiezione): una nuova prenotazione deve fare riferimento ad una proiezione valida (ovvero esistente ed entro i 15 minuti successivi all'inizio della proiezione stessa. Essendo uno unit test si è scelto di testare il comportamento del componente EasyCinema in corrispondenza dei due casi possibili: proiezione valida e non valida. Il controllo sulla validità della proiezione spetta al componente Catalogo e dunque essendo al di fuori del componente sotto test viene "imitato".
  Le condizioni di interesse individuate sono:

  1. *testNuovaPrenotazioneProiezioneValida*
     - Input: codice proiezione ritenuta valida.
     - Output previsto: nessuna eccezione.

  2. *testNuovaPrenotazioneProiezioneNonValida*
     - Input: codice proiezione ritenuta non valida.
     - Output previsto: generazione eccezione.

- confermaPrenotazione(): E' possibile confermare correttamente una prenotazione solo se c'è già una prenotazione in corso e se il credito del cliente è sufficiente a coprire l'importo. Obiettivo del test è verificare l'aggiunta della prenotazione corrente alle prenotazioni già esistenti nel caso in cui le due condizioni sopra riportate siano rispettate.
  Le condizioni di interesse individuate sono:

  1. *testConfermaPrenotazione_PrenotazioneInCorso*
     Tale test è stato realizzato come test parametrizzato in modo da eseguire lo stesso test per valori del totale della prenotazione diversi (il credito del cliente è quello di default ovvero 10.0); i valori del totale scelti sono:
     - Input: 5.0.
     - Output previsto: il numero di prenotazioni esistenti aumenta di una unità.
     - Input: 10.0.
     - Output previsto: il numero di prenotazioni esistenti aumenta di una unità.
     - Input: 15.0.
     - Output previsto: il numero di prenotazioni esistenti resta invariato.
  2. *testConfermaPrenotazione_NessunaPrenotazioneInCorso*
     - Input: nessuna prenotazione in corso.
     - Output previsto: generazione eccezione.

Per le classi dello strato di dominio: Film, PostoSala, Biglietto, Utente non sono state rilevate operazioni di interesse per la fase di testing nell'iterazione corrente.

Utilizzando lo strumento di code coverage fornito da Eclipse per i test implementati si è raggiunta, per lo strato di dominio, una copertura del 53.1%.