UniCTest

Ingegneria del Software – A.A. 2021/2022

Progetto d’esame di: Blanco Francesco Giulio, Anzalone Dario Giuseppe

Fase di Elaborazione – Iterazione 6

Sommario

[1 Versione 2](#_Toc95165835)

[2 Introduzione 2](#_Toc95165836)

[3 Analisi orientata agli oggetti 3](#_Toc95165837)

[3.1 Modello di dominio finale 3](#_Toc95165838)

[4 Progettazione Orientata agli Oggetti 4](#_Toc95165839)

[4.1.1 Diagrammi di interazione 4](#_Toc95165840)

[4.2 DCD finale 6](#_Toc95165841)

[4.2.1 Implementazione e Testing 7](#_Toc95165842)

# Versione

|  |  |  |  |
| --- | --- | --- | --- |
| Versione | Data | Descrizione | Autori |
| Elaborazione 6 | 2021/02/06 | Sesta iterazione della fase di Elaborazione. | Anzalone Dario Giuseppe, Blanco Francesco Giulio. |

# Introduzione

I Casi d’Uso già esplorati coprono il *core* del progetto, cioè soddisfano le motivazioni cardine per cui il cliente aveva deciso di commissionare UniCTest: permettere agli studenti di esercitarsi mediante lo svolgimento di simulazioni sul proprio calcolatore, consentendogli di personalizzarle, e permettere ai tutor di creare, svolgere, correggere le simulazioni cartacee con il minore sforzo possibile, lasciando la possibilità di riutilizzare il lavoro fatto in altre simulazioni, collaborare con i colleghi nella realizzazione delle stesse, ecc.

Il cuore del progetto è stato sviluppato facendo affidamento al Caso d’Uso di Avviamento, tramite il quale venivano inseriti nel sistema Tutor, Studenti e Tutor con responsabilità (Ammministratori e TutorSimulazione). Tuttavia, non è ancora possibile aggiungere tali Utenti al Sistema né è possibile aggiungere/rimuovere responsabilità a *runtime*, come promesso dal pattern Decorator applicato nella Iterazione 4 della fase di Elaborazione.

Inoltre, manca lo sviluppo di alcune primitive che consentano di visualizzare il lavoro svolto, ad esempio i test effettuati dallo Studente e i quesiti già inseriti dal Tutor.

Nella Iterazione 6 della fase di Elaborazione, dunque, ci si occuperà di

* UC6: Nuovo Tutor
* UC13: Aggiungi materia insegnata
  + Dovrà prevedere l’applicazione del pattern Flyweight
* UC14: Aggiungi responsabilità   
  (Aggiungi Amministratore/Aggiungi TutorSimulazione)
* UC15: Rimuovi responsabilità   
  (Rimuovi Amministratore/Rimuovi TutorSimulazione)
* UC5: Nuovo Studente
* UC16: Visualizza quesiti creati (CRUD)
  + Visualizzazione quesiti creati da parte del Tutor
* UC17: Visualizza test svolti (CRUD)
  + Visualizzazione test svolti da parte dello Studente

È stato deciso di non sviluppare i Diagrammi di Sequenza di Sistema in quanto i Casi d’Uso citati prevederanno l’uso di una singola operazione e dunque di una singola interazione Attore-Sistema software, pertanto i Diagrammi di Sequenza di Sistema risulterebbero ovvi.

In questa Iterazione vengono mostrati i diagrammi finali, in quanto si prevede che questa sia l’ultima Iterazione del progetto.

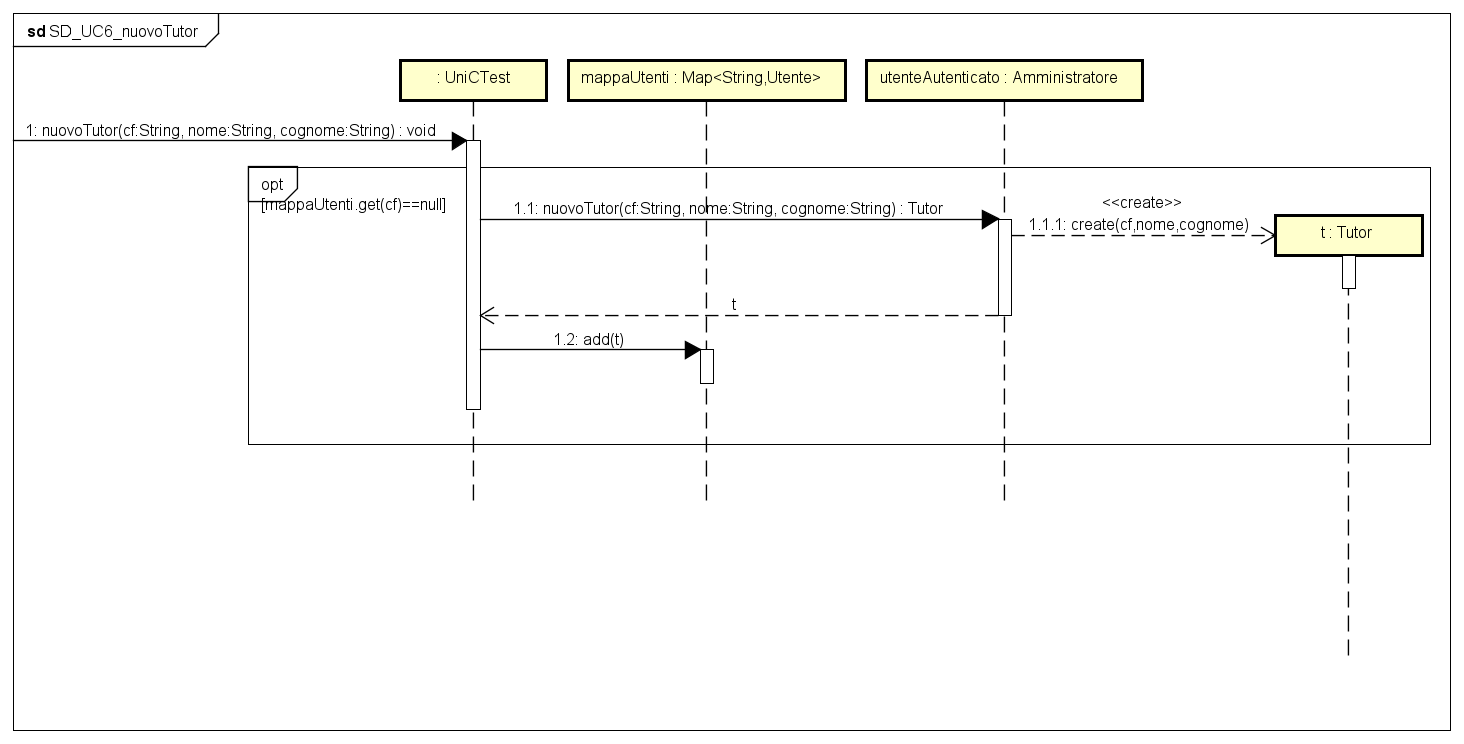
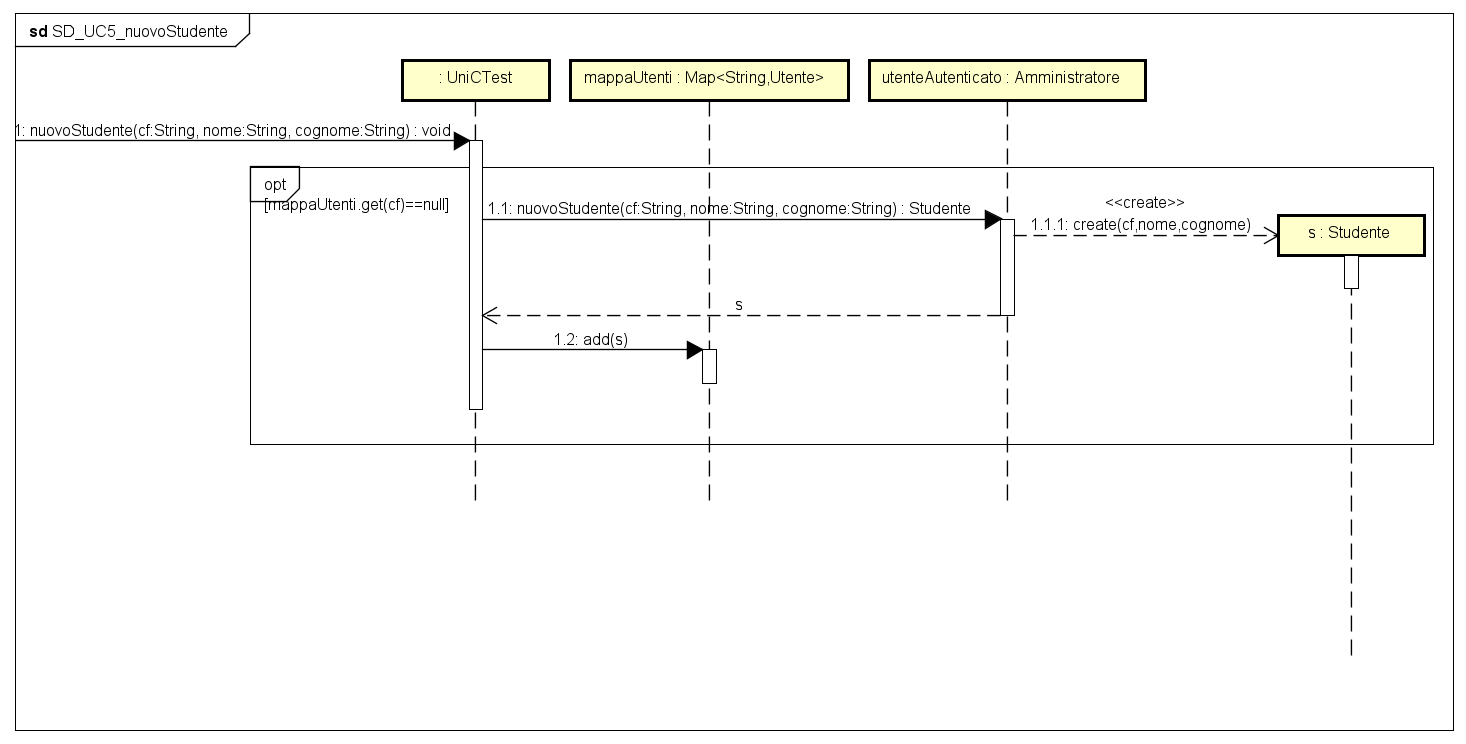
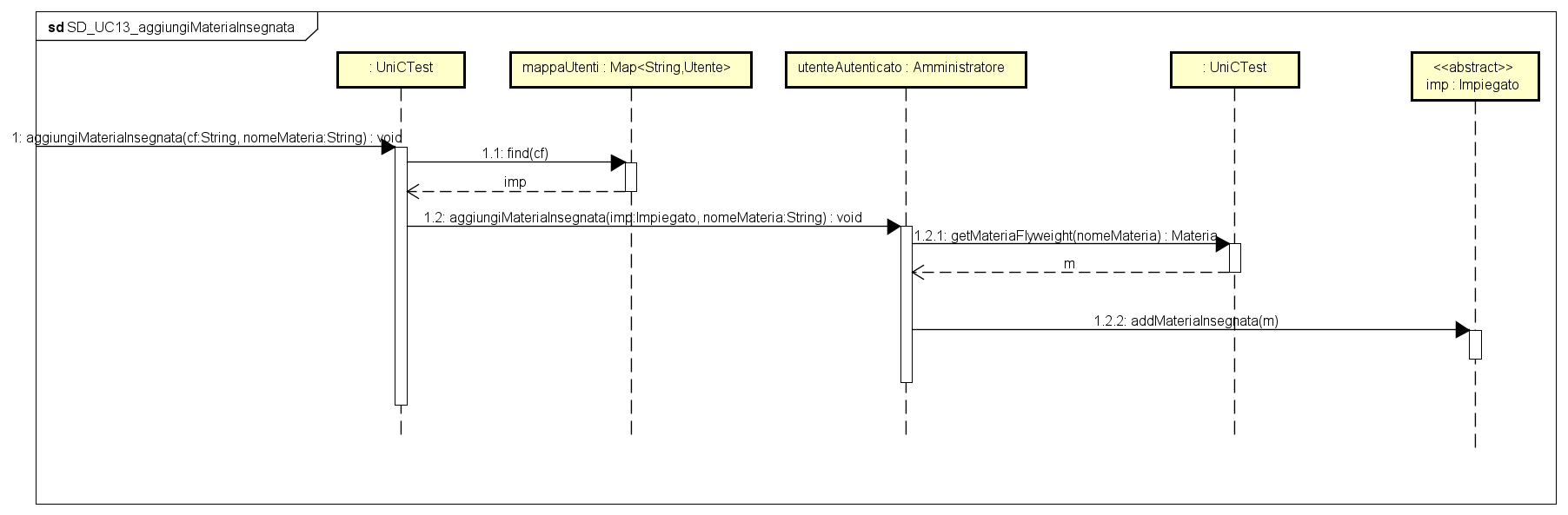
# Analisi orientata agli oggetti

## Modello di dominio finale

# Progettazione Orientata agli Oggetti

Vengono di seguito riportati i Diagrammi di Sequenza relativi alle operazioni reputate più complesse o con maggiore richiesta di dettagli. Per le operazioni notoriamente più semplici viene riportata direttamente l’Implementazione.

### Diagrammi di interazione

1. **SD\_UC5\_nuovoStudente**
2. **SD\_UC6\_nuovoTutor**
3. **SD\_UC\_13\_aggiungiMateriaInsegnata**

## DCD finale

## Implementazione e Testing

### Implementazione

Per l’Implementazione si rimanda al codice in allegato.

### Testing

**TestAggiungiResponsabilità.java**

**Si tratta di test sviluppati per testare i Casi d’Uso UC14 e UC15, ovvero l’aggiunta e la rimozione delle responsabilità addizionali di un Tutor (pattern GoF Decorator).**

Sono state sviluppate due tipologie di test differenti:

* Test a **scatola nera (black-box)**. Lo sviluppo dei test black-box è stato promosso dalle pratiche di test-driven development introdotte nell’Iterazione 4 della fase di Elaborazione. I test effettuati sono stati sviluppati prescindendo dalla conoscenza dei dettagli implementativi del codice, ma hanno analizzato il blocco software di assegnazione delle responsabilità addizionali ai Tutor come un sistema ingresso-uscita.
* Testa a **scatola bianca (white-box)**. Sono stati effettuati test a scatola bianca con ***copertura delle decisioni*** (che include la copertura delle istruzioni).   
  L’obiettivo della copertura delle decisioni è individuare un insieme di casi di test in modo da garantire che ogni decisione assuma il valore vero almeno una volta.  
  In sostanza è stato deciso di testare se un Tutor che era stato reso prima di tutto TutorSimulazione e solo dopo Amministratore veniva correttamente visto come TutorSimulazione. Allo stesso modo doveva essere verificato che un Tutor che era stato reso prima di tutto TutorSimulazione e solo dopo Amministratore veniva correttamente NON visto come TutorSimulazione una volta chiamata la funzione rimuoviTutorSimulazione.  
  Questi test non sarebbero stati effettuati se fossero stati elaborati solo test a scatola nera. Infatti la decisione di testare questa condizione deriva dalla conoscenza del workflow interno delle funzioni per aggiungere e rimuovere le responsabilità addizionali. Infatti:

@Override  
public Impiegato rimuoviAmministratore(Impiegato imp){  
 Impiegato attuale = imp;  
 Impiegato precedente = null;  
 Impiegato successivo = imp.getImpiegato();   
 //Struttura: PREC(ATTUALE(SUCC))  
 while(successivo != null){  
 if(attuale instanceof Amministratore){  
 if(precedente!=null){  
 precedente.setImpiegato(successivo);  
 return imp;  
 } else {  
 return successivo;  
 }  
 } else { //Scaliamo tutti al successivo  
 precedente = attuale; //Il PREC diventa l'ATT  
 attuale = successivo; //L'ATT diventa il SUCC  
 successivo = successivo.getImpiegato(); //Il SUCC diventa più SUCC  
 }  
 } //Interrompo se lo trovo o se non ho più successivi (cioè più interni)! --> Infatti il Tutor restituisce null perché non ha interni.  
 System.*out*.println("Non sei Amministratore, quindi non rimuovo nulla");  
 return imp;  
}

Dunque bisogna prevedere:

* + Il caso in cui *successivo*=*null*, e cioè il caso in cui l’Impiegato è un Tutor, dunque non ha responsabilità addizionali.
  + Nel caso in cui *successivo* non sia *null*,
    - Il caso in cui il *successivo* sia un Amministratore
      * Il caso in cui il precedente sia *null*, e cioè il caso (banale) in cui l’elemento passato è esso stesso un Amministratore che *wrappa* altre responsabilità al proprio interno
      * Il caso in cui il precedente non sia *null*, e dunque bisogna ricostruire la catena di impiegati rimuovendo quello *attuale*, e cioè PREC(ATT(SUCC))) diventa PREC(SUCC).
    - Il caso in cui i*successivo* non sia un Amministratore, in tal caso la ricerca deve procedere esplorando le responsabilità più interne.
* Di conseguenza, vengono analizzati i casi in cui la responsabilità cercata sia “esterna” e in cui sia “interna”.

Funzionalità testate:

* Test a scatola nera:
  + Diritti di Amministratore: è impossibile per un non-Amministratore aggiungere o rimuovere responsabilità.
  + Un Tutor reso Amministratore viene visto correttamente come Amministratore.
  + Un Tutor reso TutorSimulazione viene visto correttamente come TutorSimulazione.
  + Un Tutor reso Amministratore non può auto-rimuovere i diritti di Amministratore.
* Test a scatola bianca:
  + Un Tutor reso Amministratore e in seguito TutorSimulazione viene visto correttamente come Amministratore
  + Un Tutor reso Amministratore e in seguito TutorSimulazione viene visto correttamente come TutorSimulazione