Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative

SUPSI

Filesystem Command-line Simulator

Studente/i	Relatore	Correlatore
Daniele Cereghetti Niccolò Pasqualetti Sebastiano Piubellini	Giancarlo Corti	

Corso di laurea	Modulo / Codice Progetto	Anno
Ingegneria informatica TP	Laboratorio ingegneria del software 2	2025 - 2026

Committente	Data
Giancarlo Corti	16.12.2025

STUDENTSUPSI

Indice

- Contesto
- Problema
- Stato dell’arte
- Approccio
- Risultati
- Conclusioni

Contesto

- Applicazione delle conoscenze acquisite nel corso:
    - Agile
    - Unit Testing
    - Ecc.
- Implementare un interprete di comandi per un filesystem simulato.

Problema

Implementazione di un filesystem simulato

Requisiti generali

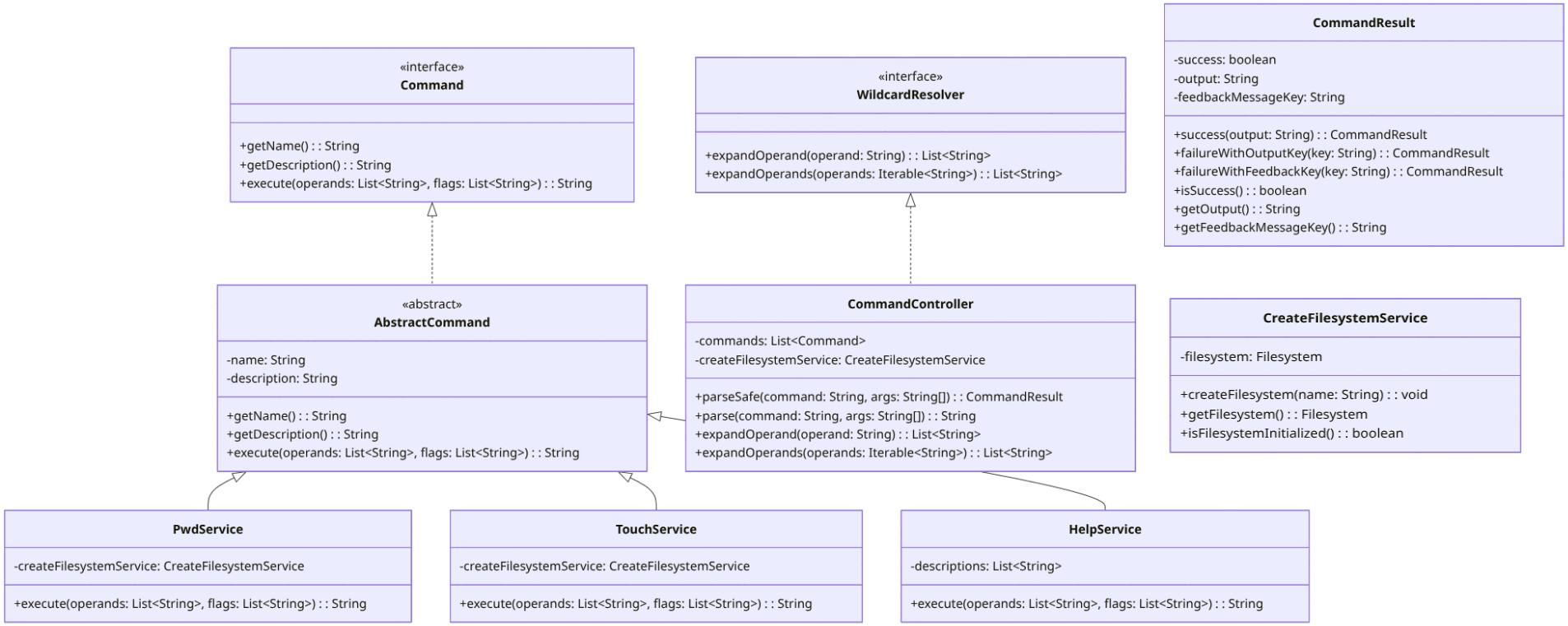
- GUI desktop stand-alone (single user).
  - Interazione testuale stile UNIX
- Funzioni principali:
  - creare un nuovo filesystem *simulato*
  - salvare lo stato del filesystem su file
  - caricare un filesystem da file;
  - help e credits in GUI;
  - feedback per azioni GUI e per l'interpretazione comandi.

UML Use Case (1)

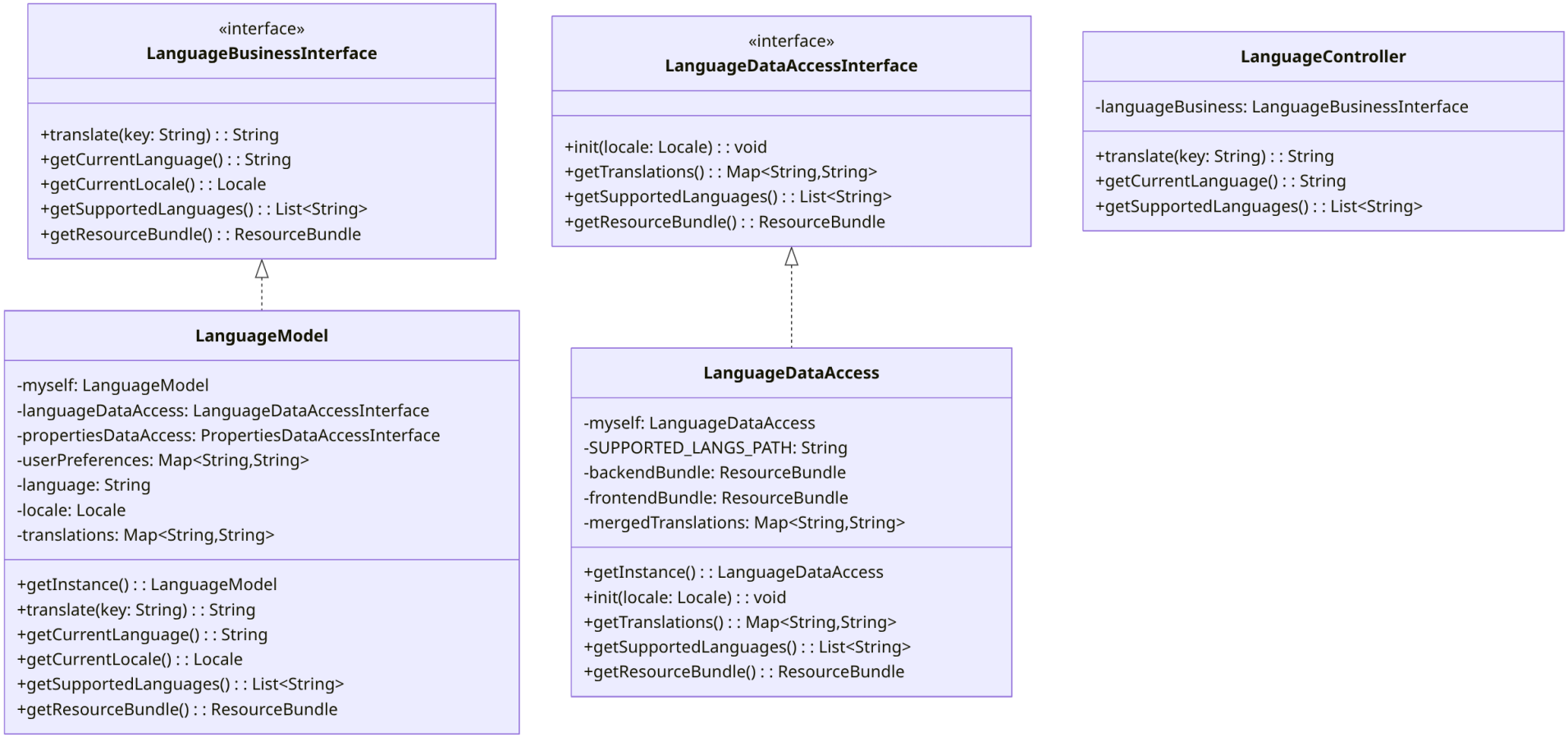
UML Use Case (2)

UML Class Diagram (Preferences)

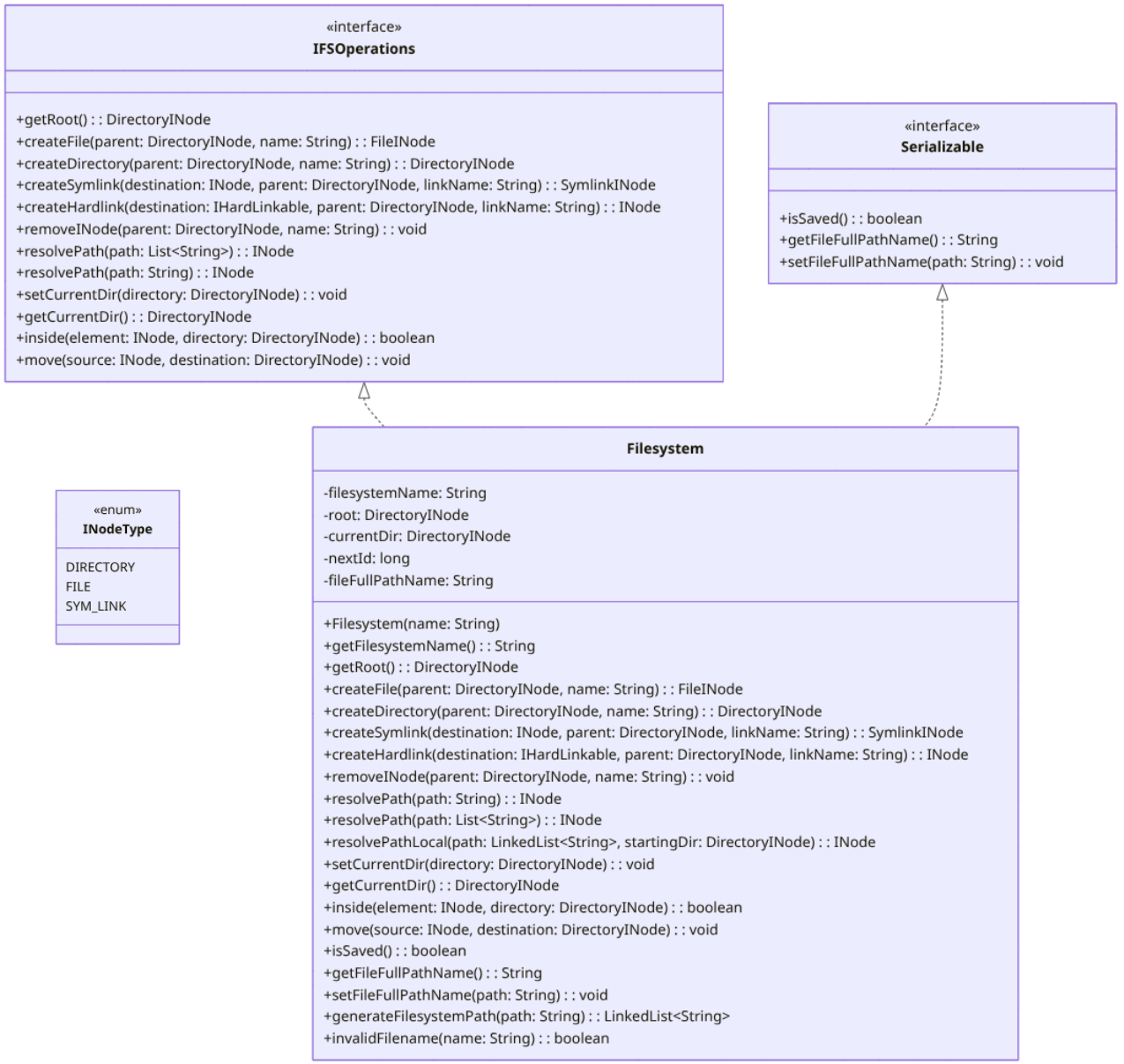
UML Class Diagram (Commands)



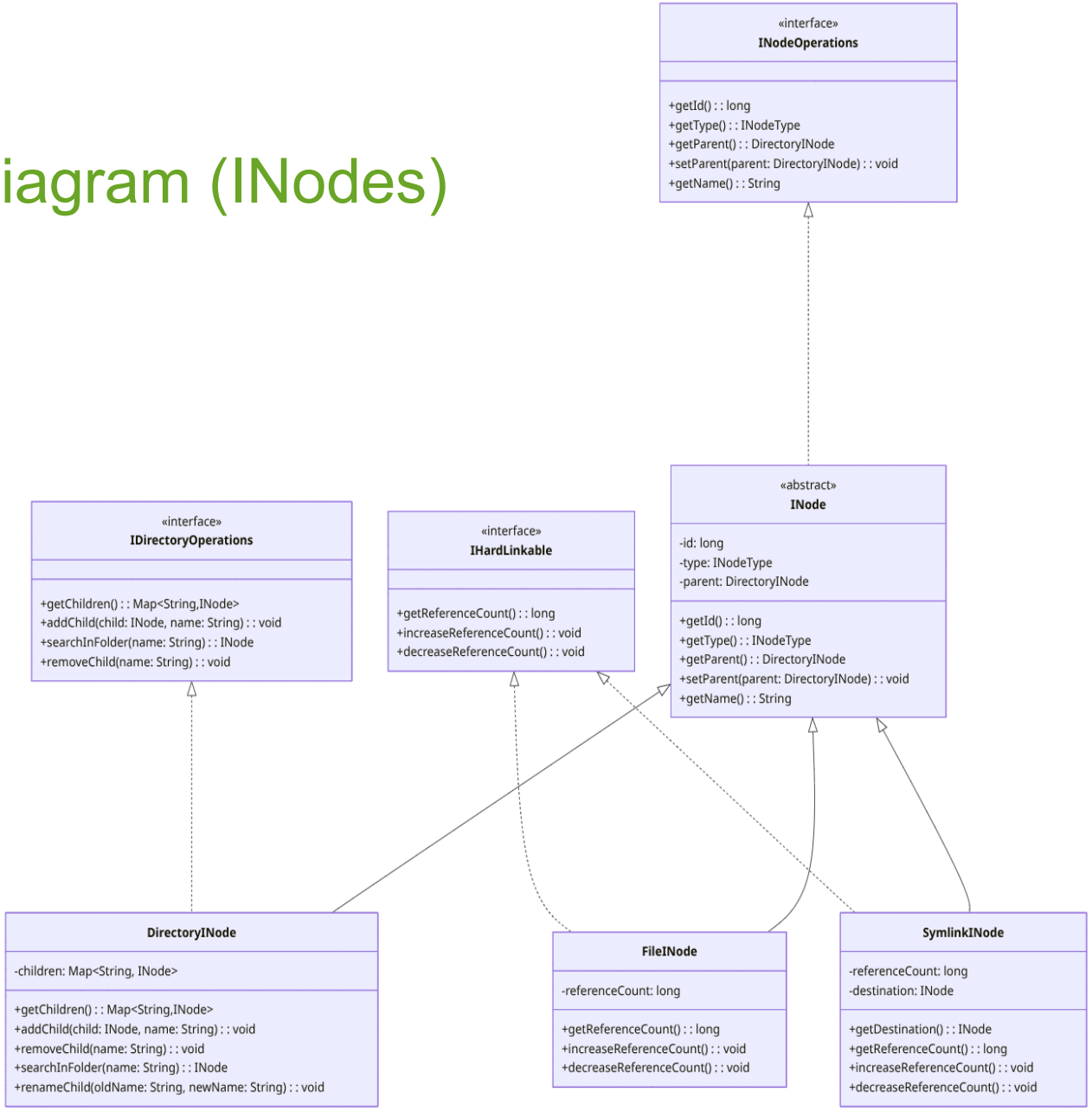
UML Class Diagram (Language)



UML Class Diagram (Filesystem)

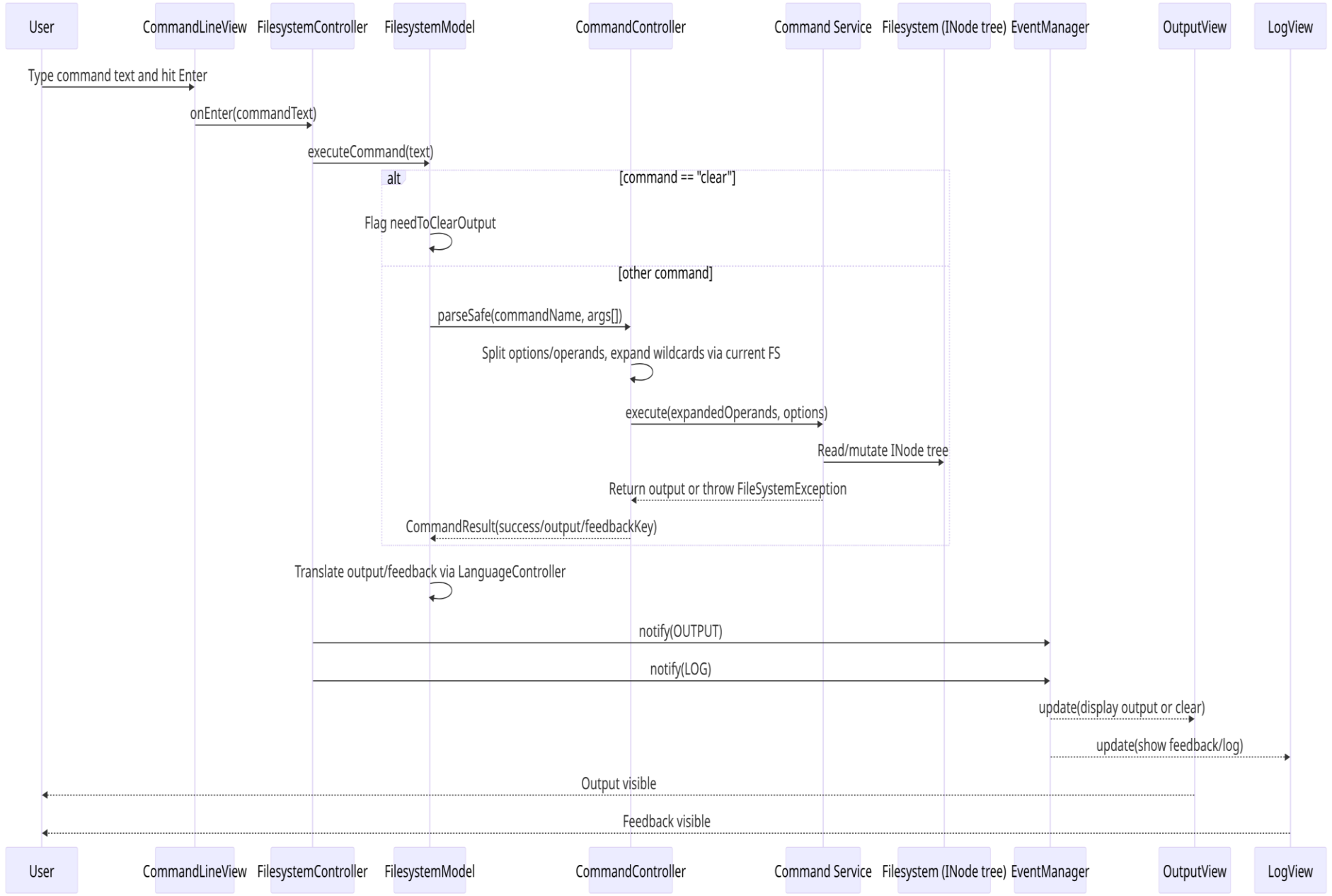


UML Class Diagram (INodes)



SUPSI

UML Sequence Diagram



Stato dell'arte

Non esiste niente che rispetti i nostri requisiti

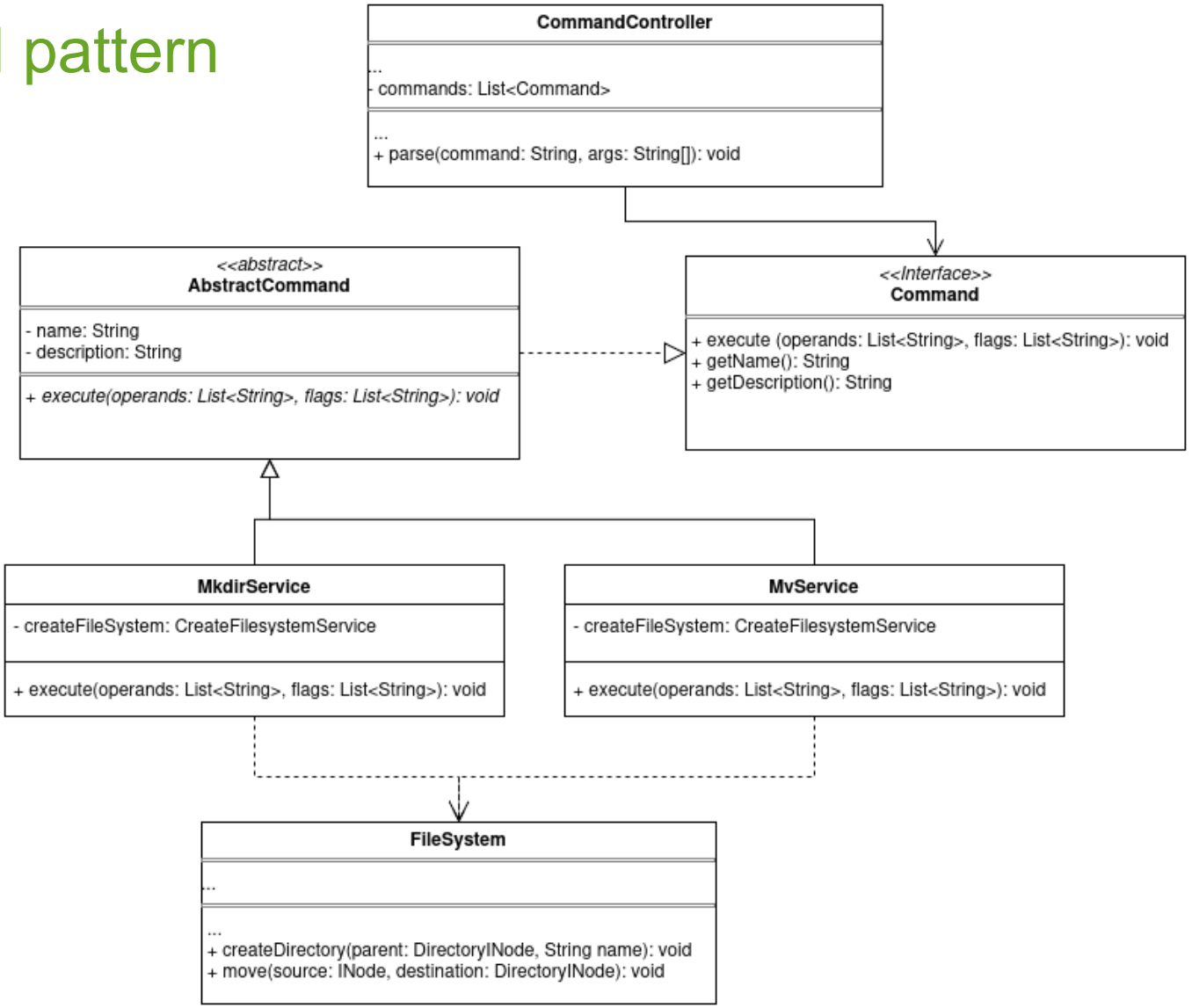
Approccio

- Divisione del lavoro con metodo AGILE
- Pattern implementati
- Persistenza
- Testing

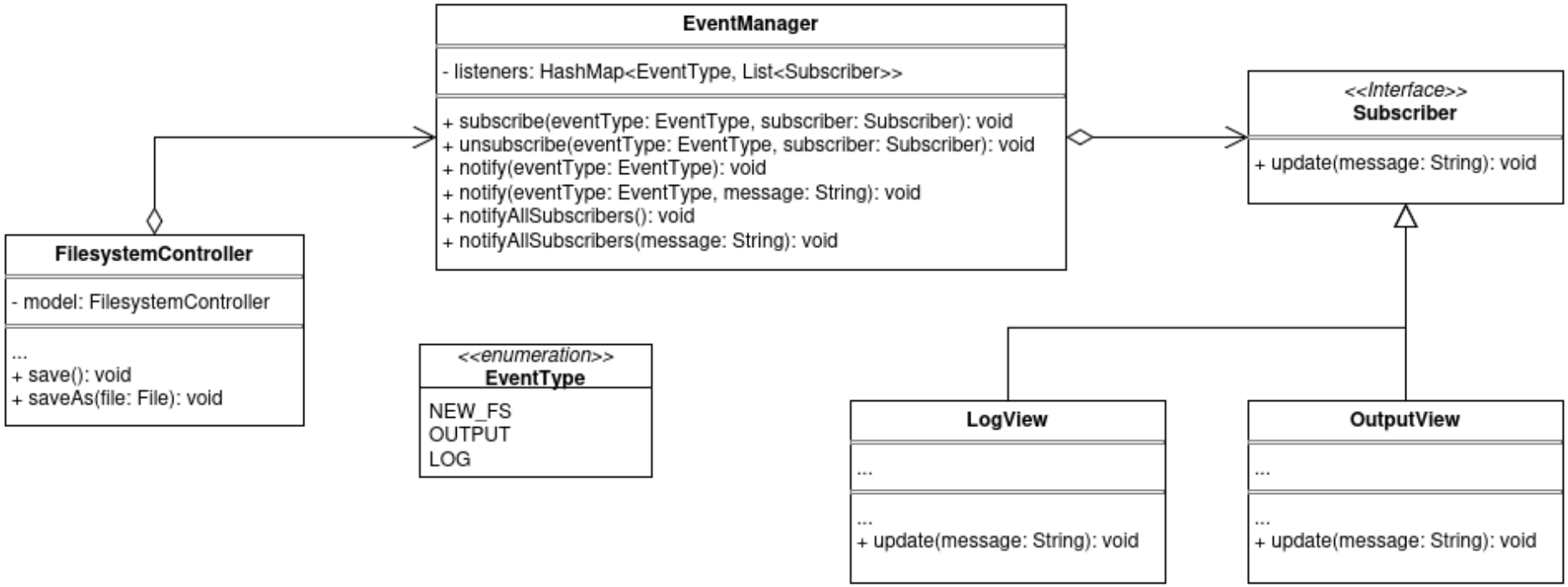
Divisione del lavoro con metodo AGILE

- Sprints di due settimane
- 4 sprints
- Meetings settimanali
    - In supsi (sprint start, ogni due settimane)
    - Online Discord (ogni settimana, controllo progresso, sincronizzare strategie)
- Per ogni sprint
    - Pianificazione
    - Assegnazione tasks
    - Sviluppo
    - Test
    - Review

Command pattern



Observer pattern



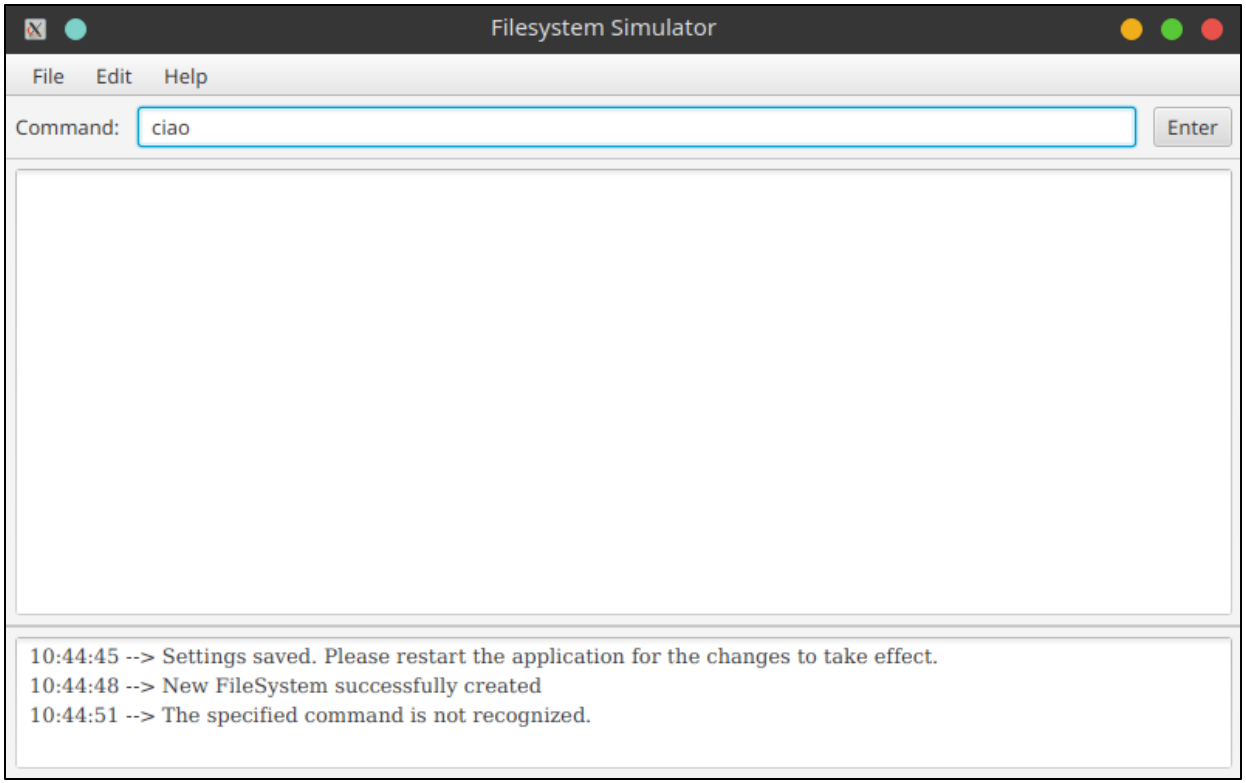
Persistenza

- Sfrutta **GSON** per la serializzazione oggetto -> json
- Funziona con l'oggetti che implementano l'interfaccia *Serializable*
- Ricostruzione del *parent* degli elementi del filesystem alla lettura

Testing

- **JUnit:** sviluppati test unitari per le classi backend per garantire correttezza e stabilità.
- **Mockito:** utilizzato per creare mock e isolare le dipendenze nei test, simulando comportamenti senza accedere a risorse reali.
- **JaCoCo:** integrato nel processo di build per generare report di copertura del codice e monitorare la qualità dei test.
- **TestFx:** utilizzato per automatizzare i test dell'interfaccia grafica

Risultati



Conclusioni

- Obiettivo raggiunto realizzando un software completo
- Approccio efficace tramite metodologia Agile
- Valore aggiunto
    - Esperienza pratica con testing e CI automatico su GitLab