Corso di Laurea in Ingegneria Informatica

Corso di Ingegneria del Software

Progetto di Ingegneria del software

Anno accademico 2023/2024

Pierpaolo Paolino
Filippo Maria Sabatino
Federico Maria Raggio
Daniele Santoro

1. Progettazione

1.1 Architettura del sistema

Il progetto segue il pattern architetturale Model-View-Controller (MVC). Il Model, insieme alla classe OperationComplexNumber, gestisce la logica di calcolo per le operazioni sui numeri complessi e la gestione delle variabili. La View fornisce un'interfaccia utente intuitiva per l'input degli utenti e la visualizzazione dei risultati. Il Controller coordina le interazioni tra Model e View, gestendo gli input utente e interpretando le richieste. Sia il Controller che il Model sono in una relazione di aggregazione con le classi Stack e BufferVariable. Si è cercato di seguire il più possibile l'acronimo SOLID per soddisfare tutti i principi del tema. Questo è stato possibile grazie anche al pattern architetturale utilizzato.

Single Responsibility Principle (SRP): in MVC ogni componente ha una responsabilità specifica come descritto sopra.

Open/Closed Principle (OCP): principio rispettato perché sia le classi del package Entity che MVC permettono l'estensione del comportamento senza modifiche dirette al codice esistente.

Liskov Substitution Principle (LSP): nel nostro contesto questo principio non è stato direttamente applicato perché non abbiamo definito classi che ne estendono altre

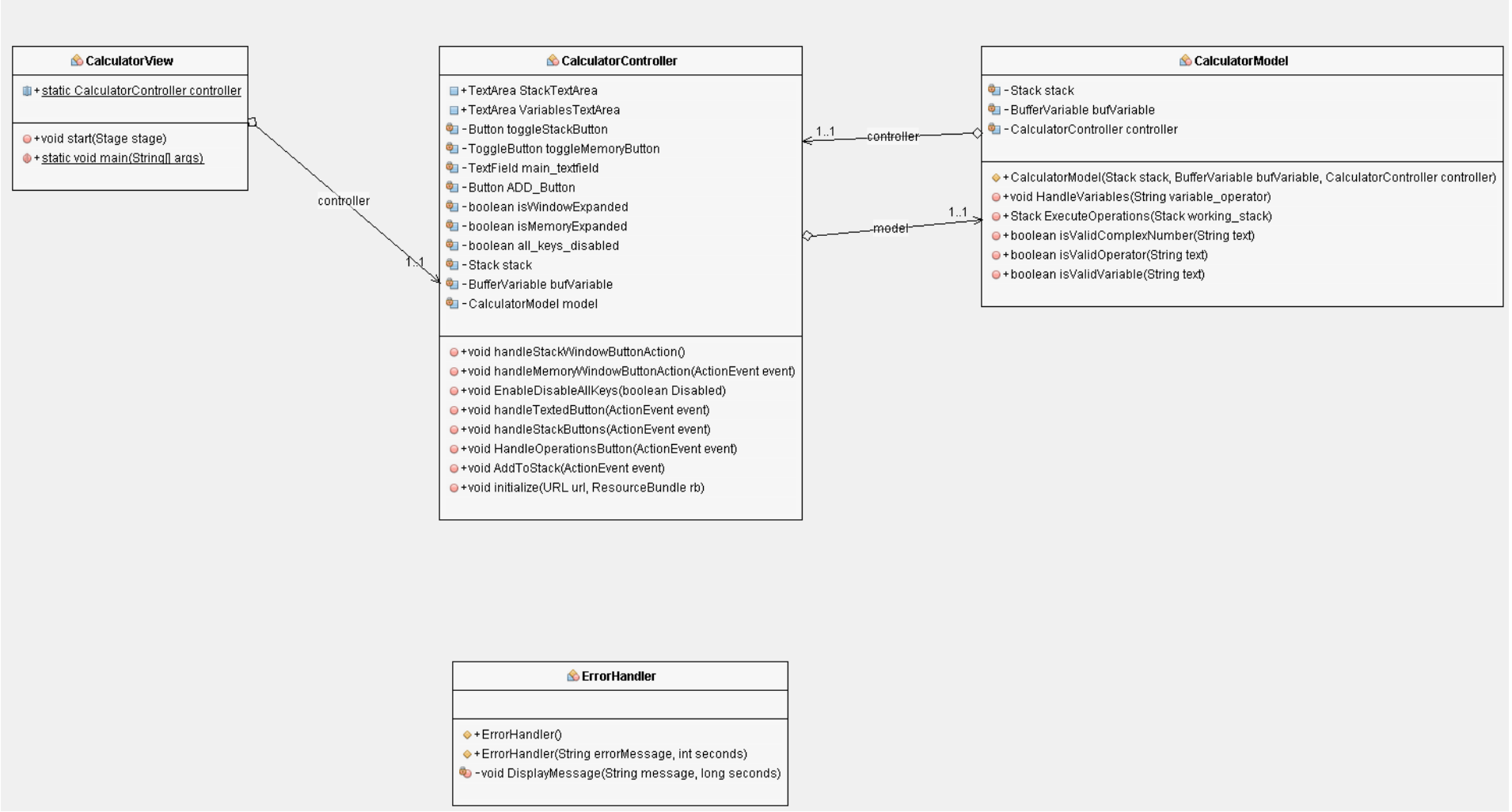
Interface Segregation Principle(ISP): si è cercato di applicare questo principio a priori costruendo la logica delle classi mantenendo un accoppiamento di livello 1, quindi solo sui dati necessari. Per la stessa ragione anche il principio DIP è soddisfatto.

1.2 Interfaccia grafica.

Per l'interfaccia si è cercato di seguire il più possibile il modello di Norman. Lo strumento utilizzato è FXML

2. Diagramma delle classi

2.1 Class diagram package MVC



Classe CalculatorView

Questa classe è progettata per fungere da applicazione JavaFX, in quanto estende la classe Application e sovrascrive il metodo start(). Seguendo il pattern adottato, fa riferimento esclusivamente a CalculatorController tramite l'attributo controller. Quando il metodo start viene invocato, avvia il caricamento del file FXML per la costruzione dell'interfaccia utente completa.

Classe CalculatorController

Metodo initialize: Questo metodo viene chiamato durante l'inizializzazione del controller e imposta vari oggetti di supporto, come lo stack, le variabili e il model.

Metodo HandleOperationsButton: Questo metodo viene chiamato dalla view ogni volta che l'utente seleziona la funzione di esecuzione, "=". Prima di passare la gestione al model dovrà controllare se lo stack è vuoto, se ci sono operandi e operatori. In caso affermativo chiama la funzione ExecuteOperation del model.

Metodo AddToStack: viene chiamato dalla view ogni volta che l'utente seleziona la funzione per lo stack, "ADD". con l'aiuto del model tramite i metodi IsValidVariable, IsValidOperator, isValidComplexNumber effettua l'inserimento nelle strutture dati apposite.

Metodo EnableDisableAllKeys: viene chiamato da ErrorHandle al fine di bloccare la funzione di tutti i tasti quando viene mostrato un messaggio di errore.

Metodo HandleStackButtons: gestisce gli eventi associati ai pulsanti che coinvolgono operazioni sullo stack della calcolatrice. Collabora direttamente con la classe Stack richiamando gli appositi metodi.

I metodi **handleStackWindowButtonAction** e **handleMemoryWindowButtonAction** gestiscono rispettivamente gli eventi associati ai pulsanti per espandere/contrarre la finestra dello stack e della memoria.

Il metodo **handleTextedButton:** gestisce gli eventi associati ai pulsanti numerici e agli operatori quando vengono premuti.

Classe CalculatorModel

Metodo HandleVariables: questo metodo gestisce tutte le operazioni legate alle variabili. Interagisce direttamente con la classe BufferVariable e Stack. Viene chiamato dal controller solo dopo la verifica di IsValidVariable.

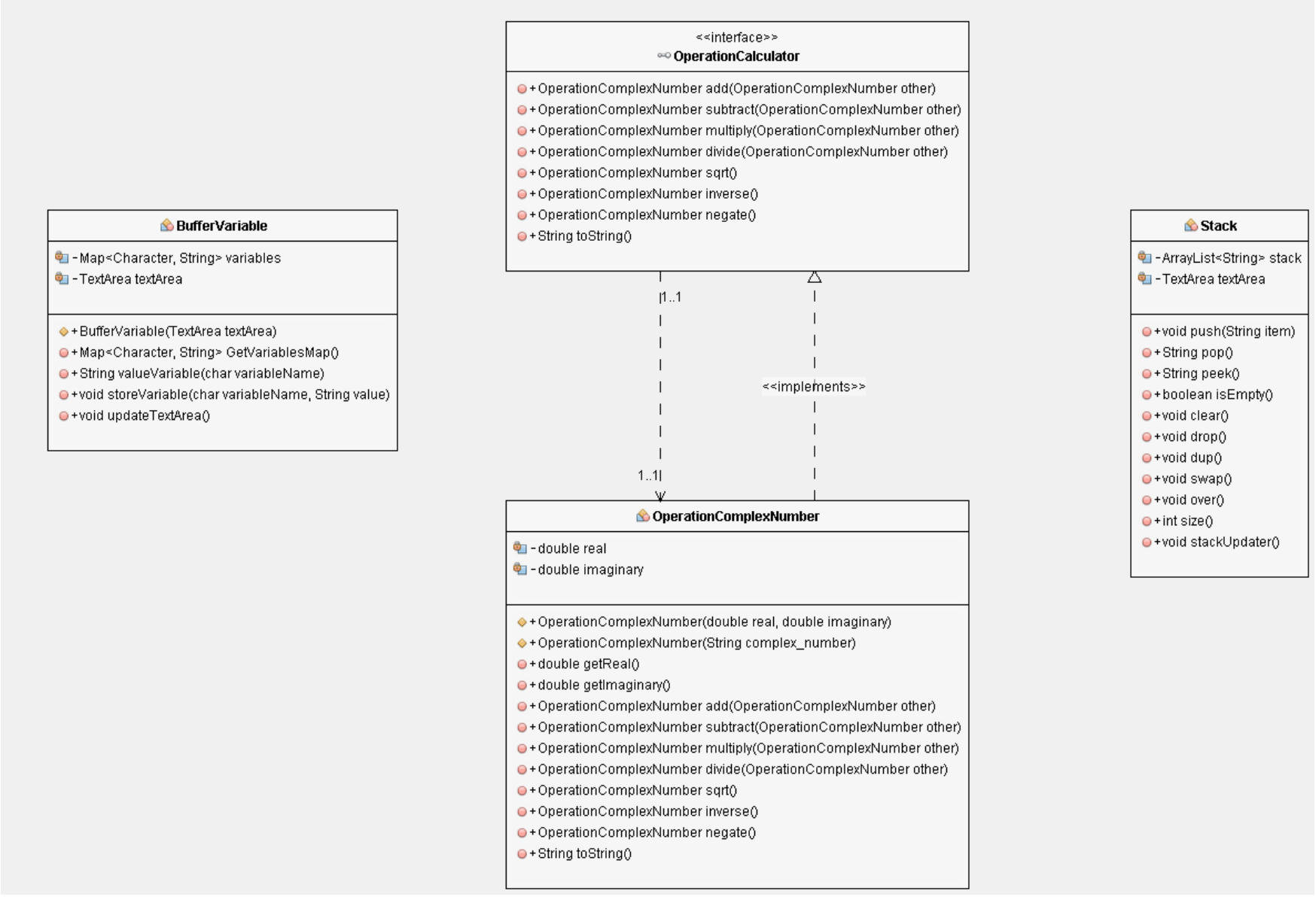
Metodo ExecuteOperations: Questo metodo prende in input una copia dello stack e esegue le operazioni aritmetiche specificate in esso. Alla fine restituisce al chiamante, cioè il metodo HandleOperationButton del controller, un nuovo stack che contiene tutti i risultati delle operazioni eseguite. Il controller si occupa di gestire correttamente lo stack iniziale con quello ritornato permettendo di mantenere coerenza fra i dati.

Classe ErrorHandler

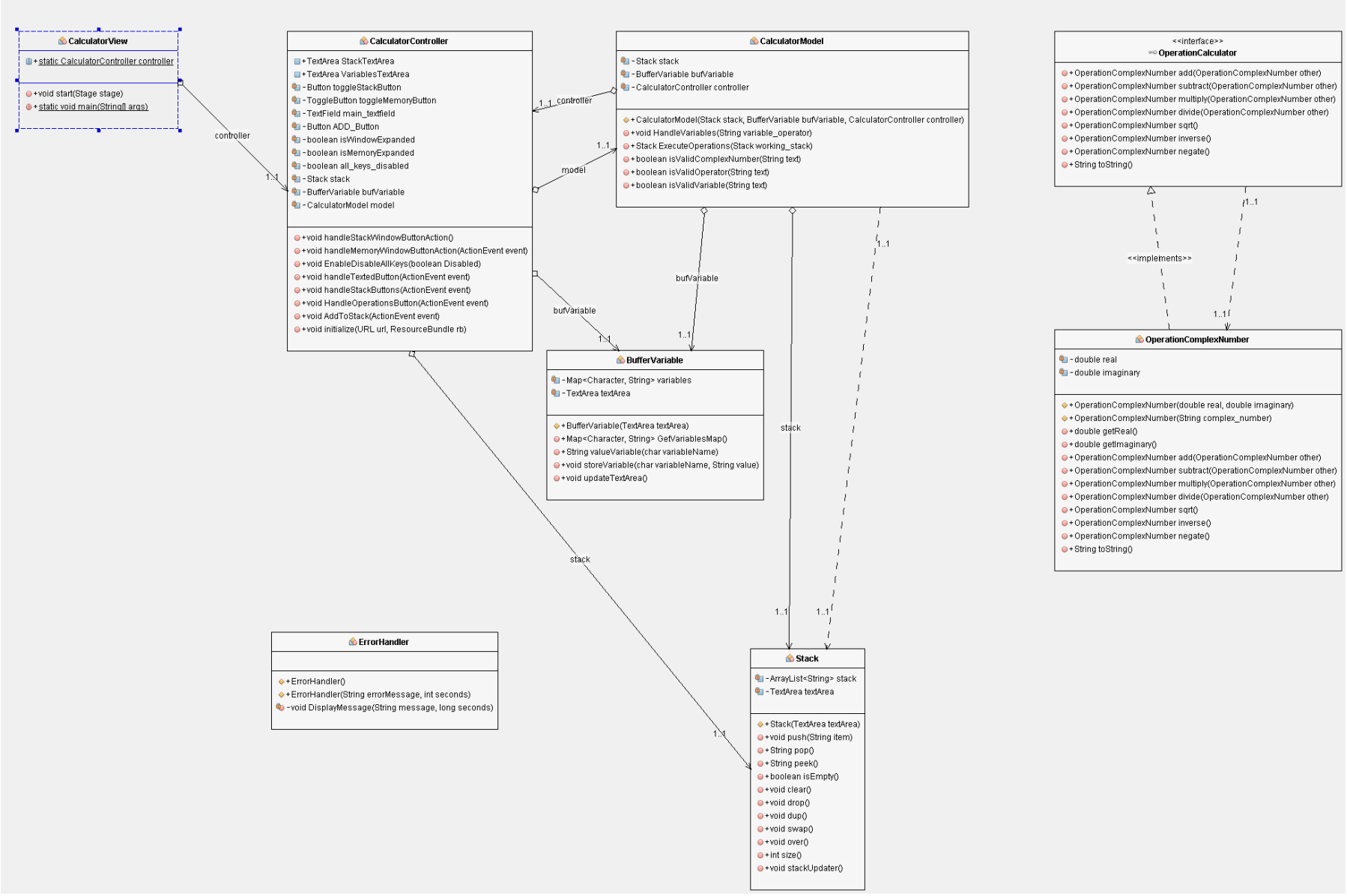
Questa classe estende Exception, quindi ogni volta che viene lanciata un'eccezione di tipo ErrorHandler dal controller o dal model, il costruttore chiama il metodo DisplayMessage(), al quale viene passato il messaggio di errore ricevuto da chi ha lanciato l'eccezione e il tempo massimo per il quale l'errore rimarrà visibile a schermo.

Metodo DisplayMessage: Questo metodo gestisce la visualizzazione del messaggio di errore nell'interfaccia utente. Disabilita temporaneamente l'input dell'utente sulla calcolatrice e imposta il messaggio di errore nella TextField.

2.2 Class diagram package Entity

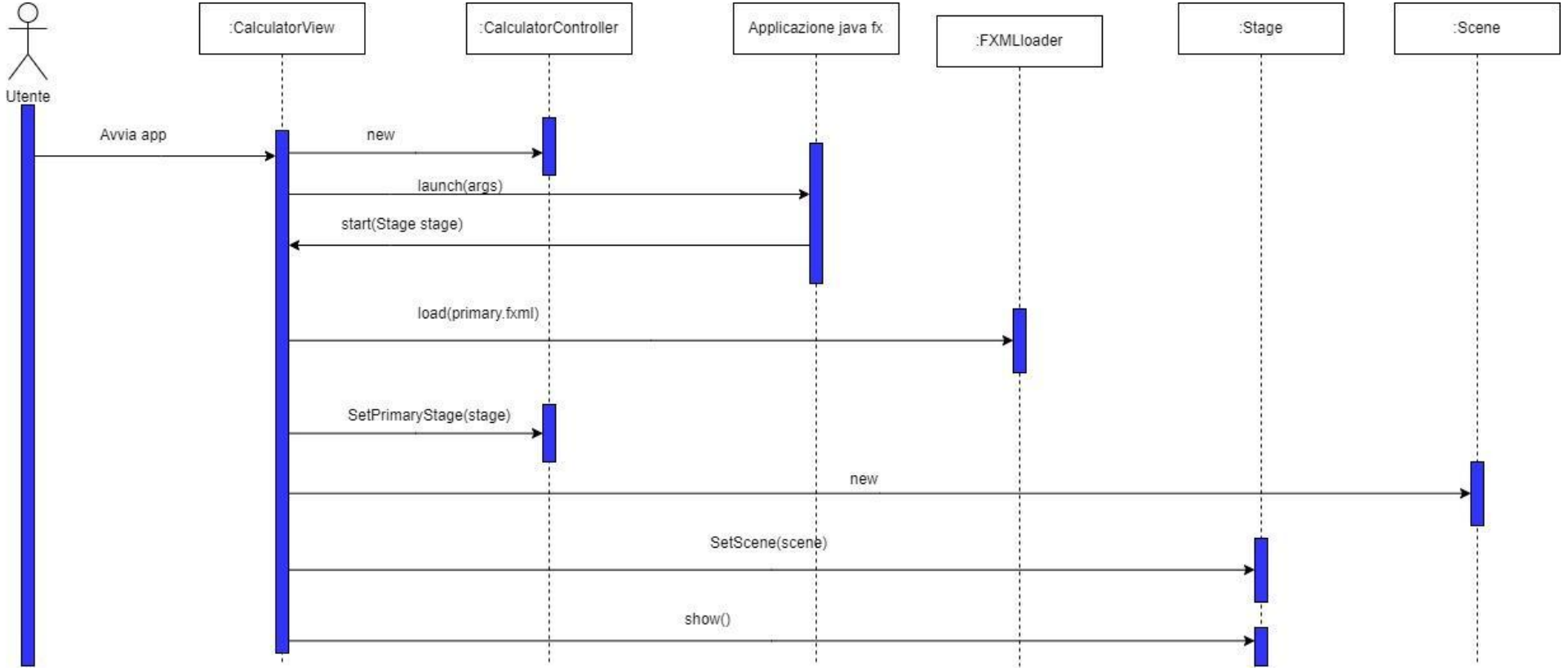


2.3 Class diagram visione completa

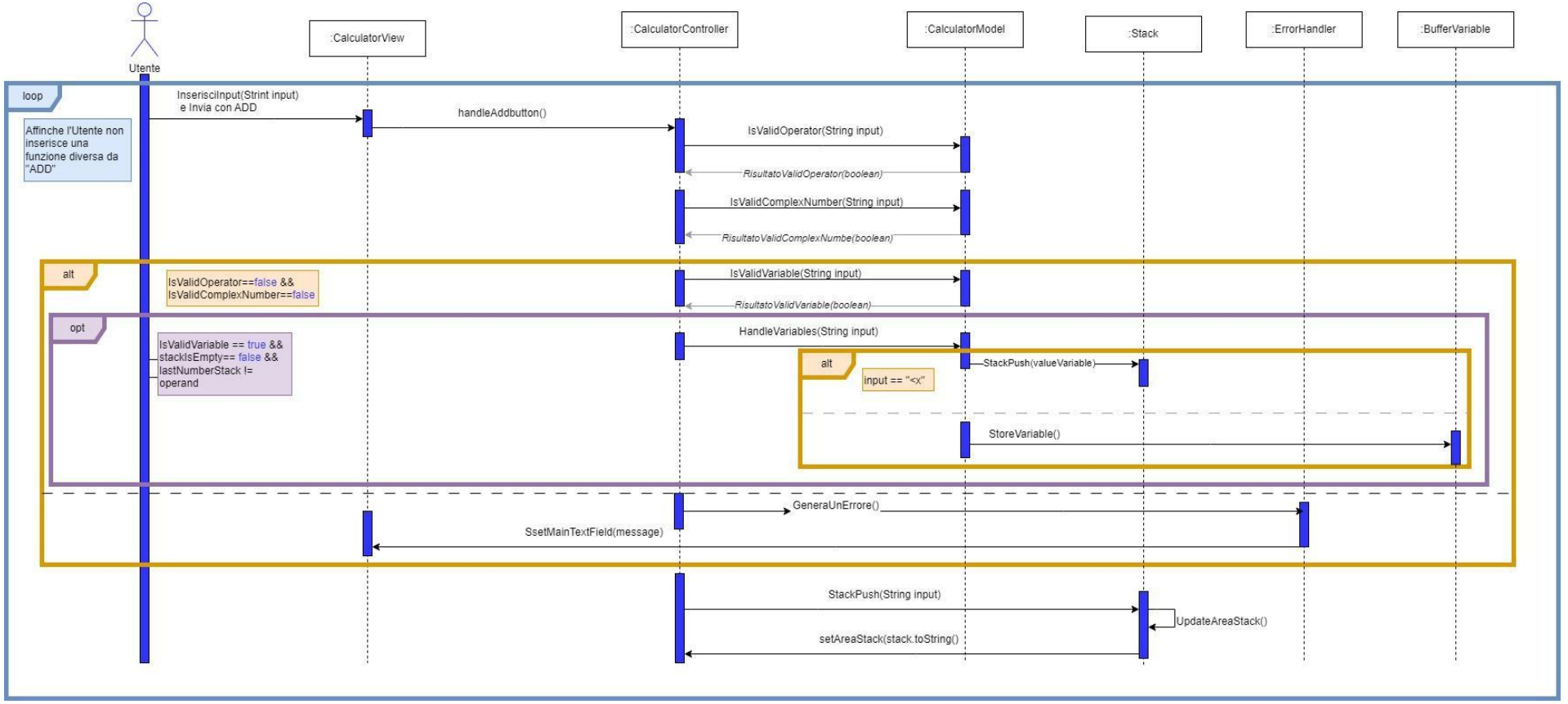


3. Sequence Diagram

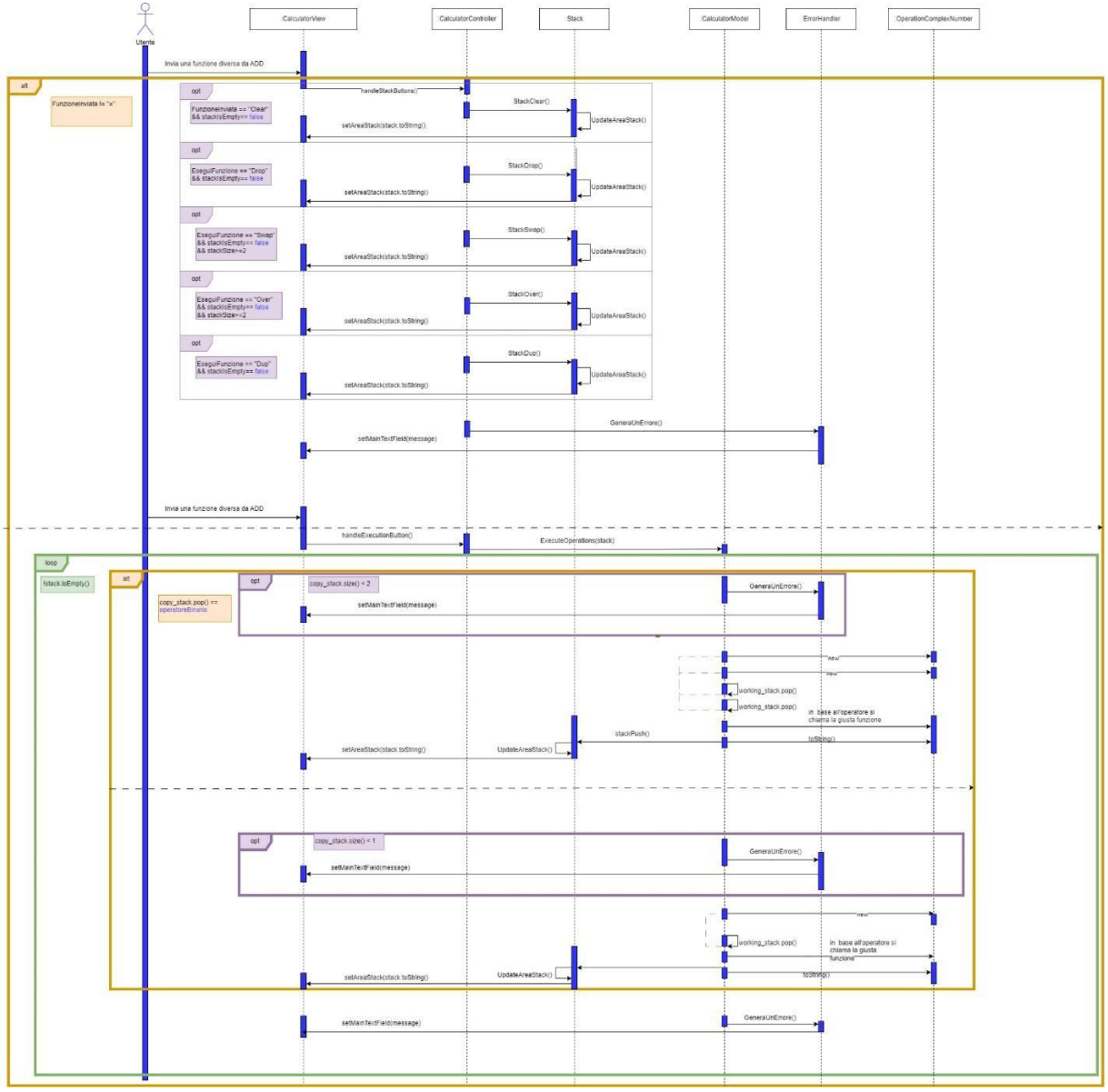
3.1 Sequence diagram per lo scenario “Avvio calcolatrice”



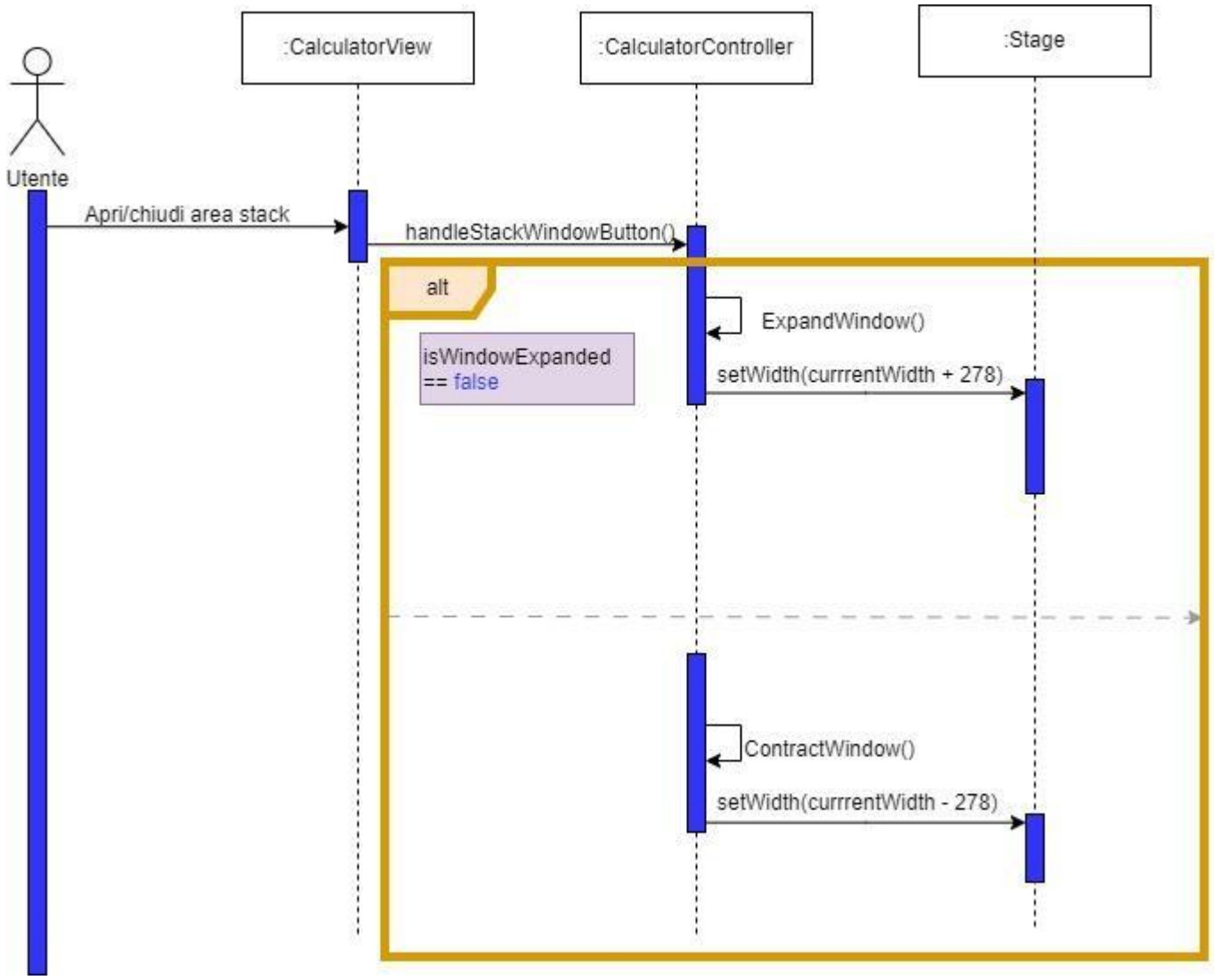
3.2 Sequence diagram per lo scenario “Inserisci input”



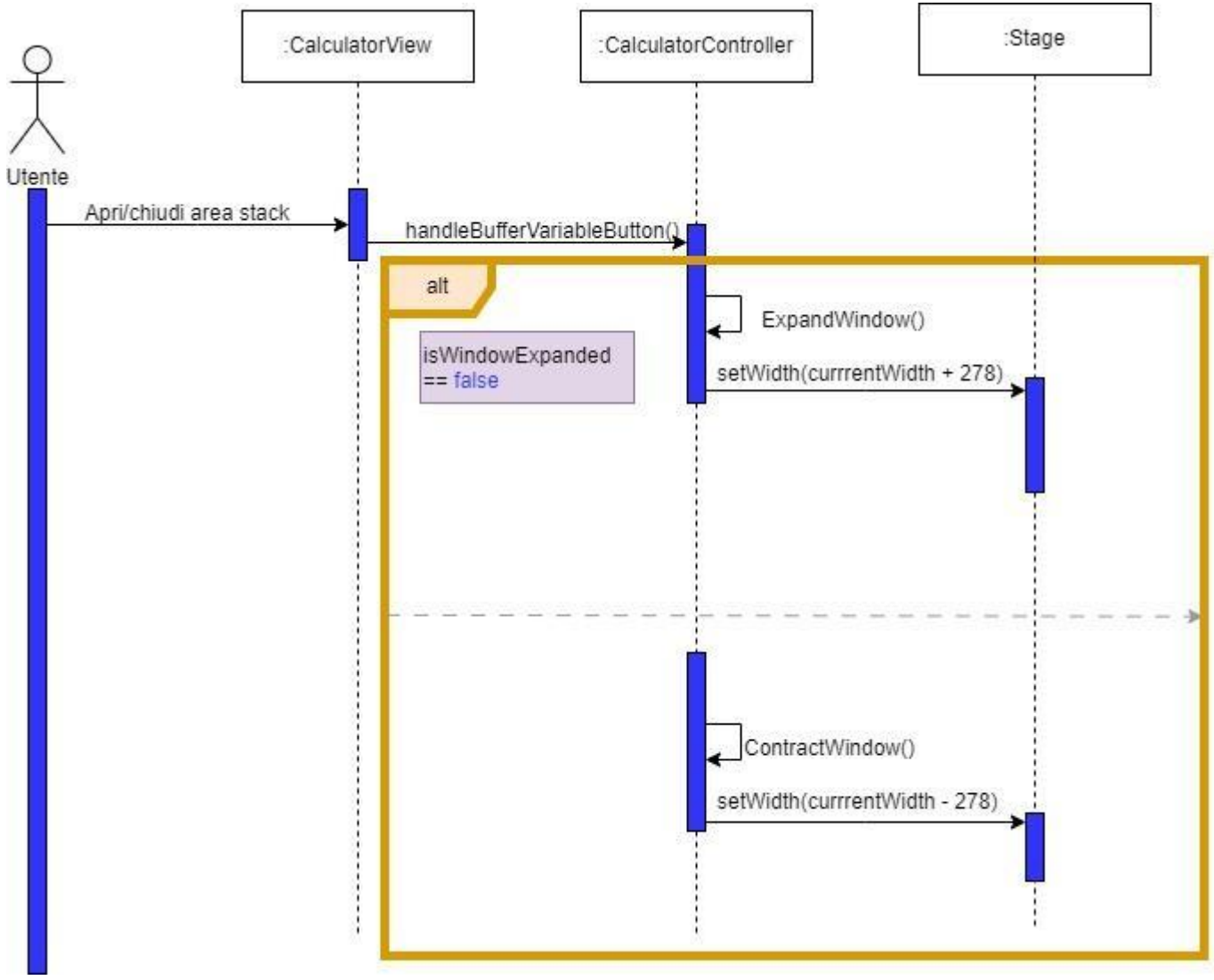
3.3 Sequence diagram per lo scenario “Esegui funzione”



3.4 Sequence diagram per lo scenario “Visualizza stato stack”



3.5 Sequence diagram per lo scenario “Visualizza buffer delle variabili”



4. Matrice di tracciabilità

La matrice illustra le relazioni tra le classi del sistema e i requisiti specificati.

Classe	Requisiti Corrispondenti
CalculatorView	[RF05-RF08], [RF16-R44], RF47
CalculatorController	RF9, RF15, RF45, RF 47 RF48, RF50
CalculatorModel	[RF10-RF14], RF46, RF47
ErrorHandler	RF09, RF11, [RF13-RF15], RF50
Stack	RF02, RF03, RF16, RF49
BufferVariable	RF10, RF18, RF49
OperationComplexNumber	RF4, RF26, RF32, RF34, RF38, RF40, RF43, RF44