GRUPPO 09
Gianmarco Guerriero
Angelo Di Mieri
Biagio De Martino
Antonio Carbone

REVISIONE

Rispetto alla versione precedente del documento, sono state apportate le seguenti modifiche:

- Nel Class Diagram della sezione 2.1, durante la revisione in vista della fase di implementazione, il team ha individuato incongruenze tra il diagramma e la descrizione a causa di errori di battitura. In particolare, sono state apportate le seguenti modifiche:
  - Aggiunta una relazione di dipendenza tra la classe Vars e la classe BinaryCanonicOperations.
  - Rimossa la relazione di dipendenza tra la classe Vars e la classe UnaryCanonicOperations.
  - Inseriti gli attributi buttonInvio e buttonConferma nella classe Controller, come indicato nella descrizione ma assenti nel diagramma.
- Durante l'implementazione, il team ha deciso di modificare il tipo degli attributi nella classe ComplexNumber (e di conseguenza i tipi ritornati nei metodi) da double a BigDecimal per gestire numeri che superano la capacità di un tipo double.
- Notata la mancanza del metodo toStringArrayList() nella classe Vars, essenziale per mostrare il valore delle variabili tramite interfaccia grafica.
- Modificati i SD 4.2-4.3-4.4, poiché il team ha deciso di non utilizzare più il metodo getValueOf(), preferendo un metodo già fornito dall'interfaccia Map (remove()) utilizzata per la gestione delle variabili nella classe Vars.
- Nella classe UnaryCanonicOperations, cambiato il tipo di ritorno del metodo squareRoot() da ComplexNumber a ComplexNumber[], poiché una radice quadrata produce sempre due risultati.
- Aggiunto un attributo di tipo Alert nella classe Controller per gestire le schermate di errore.
- Cambiato il nome della InvalidOperandsNumberException in InvalidOperandsException.
- Aggiunti i metodi equals() e hashCode() nella classe ComplexNumber per implementare una relazione di uguaglianza tra i numeri complessi.
- Modificato il tipo di ritorno del metodo checkInput() nella classe Checker da boolean a void, poiché ora il metodo, in caso di input errato, lancia una InvalidInputException.

DOCUMENTO DESIGN

INDICE

1. DESIGN ARCHITETTURALE

Il team, visto e considerato che il sistema da sviluppare è risultato essere di una non elevata estensione, ha preso la decisione di saltare questa fase del processo di Design e andare direttamente alla fase dettagliata.

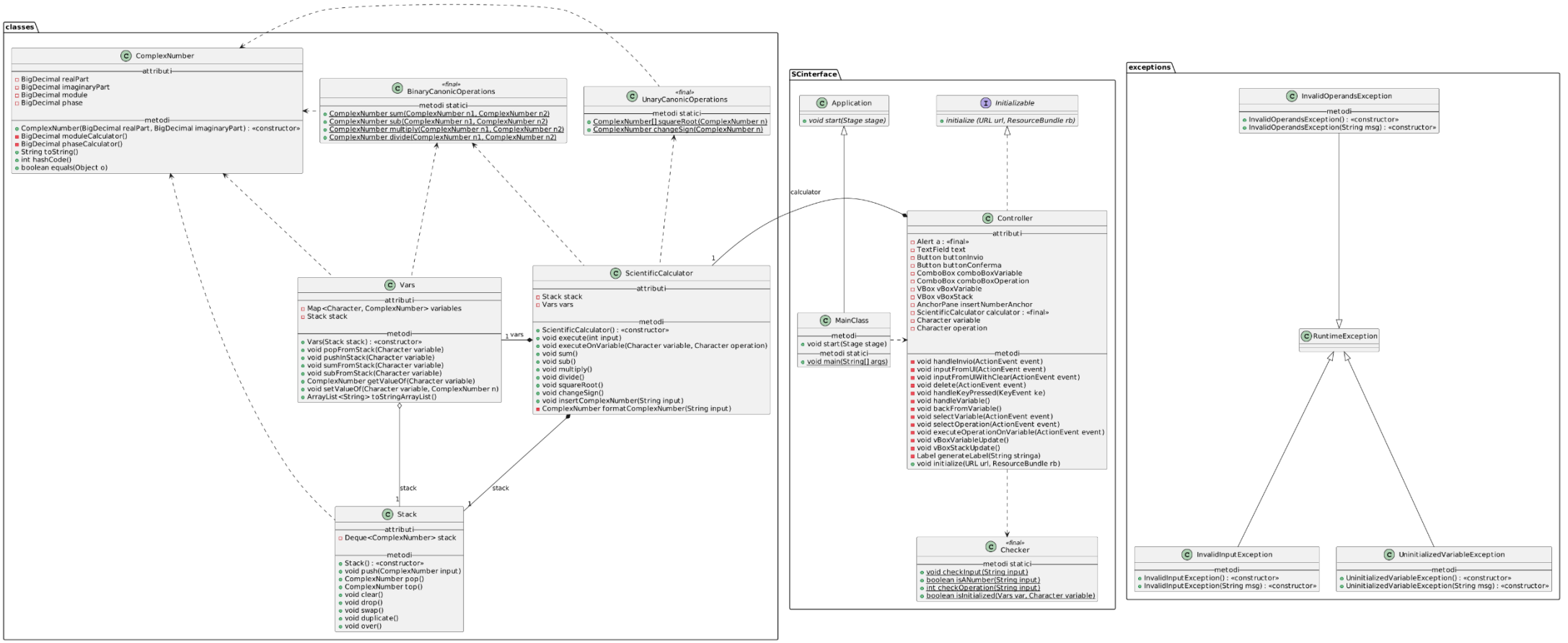
2. DESIGN DETTAGLIATO

In questa fase il team ha decomposto il sistema finale. Basandosi su un design orientato agli oggetti.
In particolare, si è prodotto:

- Un **Class Diagram** per rappresentare le entità funzionali del sistema.
- Dei **Sequence diagram** per descrivere appropriatamente la sequenza e il flusso di messaggi che le diverse classi debbano scambiarsi per poter soddisfare gli Use-Case definiti nel documento dei requisiti.
- Degli **Activity Diagram** per descrivere l'interazione utente-interfaccia grafica.
- Un nuovo **mock-up** dell'interfaccia, più coerente con quella che si auspica sia quella finale.

GRUPPO 09
Gianmarco Guerriero
Angelo Di Mieri
Biagio De Martino
Antonio Carbone

2.1. CLASS DIAGRAM



2.1.1. DESCRIZIONE DEL CLASS DIAGRAM

2.1.1.1. PACKAGE SCINTERFACE

Classe MainClass: la classe gestisce l'avvio dell'applicazione. Estende la classe Application (presente nel package javafx.application.Application) in modo tale da poter supportare l'utilizzo dell'interfaccia grafica.	
METODI	**DESCRIZIONE**
public static void main (String[] args)	Il metodo main avvia l'applicazione.
public void start(Stage stage)	Metodo sovrascritto dalla classe Application, indispensabile per poter gestire la visualizzazione e l'interazione con l'interfaccia grafica

Classe Controller: tale classe è fondamentale per l'interazione tra interfaccia grafica e implementazione delle vere e proprie funzionalità del sistema.	
ATTRIBUTI	**DESCRIZIONE**
private TextField text	Attributo che definisce il collegamento con il campo in cui l'utente inserisce il testo.
private Button buttonInvio	Attributo che definisce il collegamento con il pulsante "Invio" presente sull'interfaccia.
private Button buttonConferma	Attributo che definisce il collegamento con il pulsante "Conferma" presente nella schermata di gestione delle variabili.
private final Alert a	Attributo che definisce il collegamento con l'alert che potrebbe essere mostrato a schermo durante l'esecuzione dell'applicazione.
private ComboBox comboBoxOperation	Attributo che definisce il menu a tendina tramite il quale l'utente può selezionare l'operazione da effettuare su una variabile.
private ComboBox comboBoxVariable	Attributo che definisce il menu a tendina tramite il quale l'utente può selezionare la variabile su cui effettuare l'operazione.
private VBox vBoxVariable	Attributo che gestisce la visualizzazione su interfaccia delle variabili inizializzate.

private VBox vBoxStack	Attributo che gestisce la visualizzazione su interfaccia dei valori contenuti all'interno dello stack.
private AnchorPane insertNumberAnchor	Attribuito che definisce il collegamento con l'AnchorPane che consente di gestire sia il tastierino numerico, che le diverse operazioni (incluse quello sullo stack).
private final ScientificCalculator calculator	Attributo che consente di collegare l'interfaccia grafica al modello (in questo specifico caso le classi implementate nel package "classes").
private Character variable	Attributo che permette di tenere traccia della variabile selezionata.
private Character operation	Attributo che permette di tenere traccia dell'operazione da svolgere sulla variabile selezionata.
METODI	**DESCRIZIONE**
private void handleInvio(ActionEvent event)	Metodo che associa l'azione "premi Invio" alla corrispondente sequenza di operazioni.
private void inputFromUI(ActionEvent event)	Metodo che associa ai tasti, premuti tramite UI, le corrispettive azioni, senza cancellare il testo dal box testuale.
private void inputFromUIWithClear(ActionEvent event)	Metodo che associa ai tasti, premuti tramite UI, le corrispettive azioni cancellando il testo dal box testuale.
private void delete(ActionEvent event)	Metodo che associa all'azione "premi Delete" la cancellazione del testo dal box testuale.
private void handleKeyPressed(KeyEvent ke)	Metodo che cattura la pressione di un tasto tramite tastiera e lo gestisce appropriatamente.
private void handleVariable()	Metodo che cattura la pressione del tasto "Variabile" tramite UI e lo gestisce appropriatamente.

private void backFromVariable()	Metodo che dopo aver cliccato sul tasto "Annulla" dalla sezione delle variabili, permette di rivisualizzare il tastierino numerico, annullando eventuali selezioni svolte.
private void selectVariable(ActionEvent event)	Metodo che permette di salvare nell'attributo 'variable' la variabile selezionata dal menù a tendina.
private void selectOperation(ActionEvent event)	Metodo che permette di salvare nell'attributo 'operation' l'operazione, da effettuare su 'variabile', selezionata dal menù a tendina.
private void executeOperationOnVariable(ActionEvent event)	Metodo che permette di eseguire l'operazione scelta sulla variabile, tramite la pressione del tasto "Conferma".
private void vBoxVariableUpdate()	Metodo che aggiorna la visualizzazione dello stato delle variabili.
private void vBoxStackUpdate()	Metodo che aggiorna la visualizzazione dello stack dopo che è stato modificato.
private Label generateLabel(String stringa)	Metodo che genera le label con gli elementi dello stack o della sezione delle variabili, da inserire nei rispettivi VBox.
public void initialize(URL url, ResourceBundle rb)	Metodo di inizializzazione del controller; vengono impostati gli attributi di cui ha bisogno.

Classe Checker: classe pensata per implementare i differenti controlli sull'input, su che tipo di input è stato inserito (numero o operazione), su quale operazione è stata inserita e per verificare se una variabile è stata inizializzata precedentemente.

METODI	**DESCRIZIONE**
public static void checkInput(String input)	Metodo che controlla che l'input ricevuto sia corretto.
public static boolean isANumber(String input)	Metodo che controlla che l'input, sicuramente corretto in seguito al controllo della checkInput, sia un numero.
public static int checkOperation(String input)	Metodo che restituisce un intero corrispondente all'operazione.
public static boolean isInitialized(Vars var, Character variable)	Metodo che controlla che la variabile non sia già inizializzata prima di sovrascriverla.

2.1.1.2. PACKAGE CLASSES

Classe ComplexNumber: classe che contiene gli attributi e i metodi adatti alla gestione dei numeri complessi.	
ATTRIBUTI	**DESCRIZIONE**
private BigDecimal realPart	Attributo che contiene la parte reale del numero complesso.
private BigDecimal imaginaryPart	Attributo che contiene la parte immaginaria del numero complesso.
private BigDecimal module	Attributo che contiene il modulo del numero complesso.
private BigDecimal phase	Attributo che contiene la fase del numero complesso.
METODI	**DESCRIZIONE**
public ComplexNumber(BigDecimal realPart, BigDecimal imaginaryPart) : <<constructor>>	Costruttore per la classe ComplexNumber a cui vanno passate parte reale e parte immaginaria del numero complesso.
private BigDecimal moduleCalculator()	Metodo che permette di calcolare il modulo di un numero complesso.
private BigDecimal phaseCalculator()	Metodo che permette di calcolare la fase di un numero complesso.
public String toString()	Metodo che permette di restituire sotto forma di stringa il numero complesso.
public boolean equals(Object o)	Metodo che definisce la relazione di uguaglianza tra due ComplexNumber. In particolare, due ComplexNumber sono uguali se sono uguali rispettivamente la loro realPart e la loro imaginaryPart.
public int hashCode()	Metodo che ridefinisce la generazione di un codice hash. Verrà generato a partire dagli attributi realPart e imaginaryPart.

Classe BinaryCanonicOperations: classe final (non potrà essere ereditata). Tale classe implementa i metodi utili ad effettuare le operazioni canoniche binarie come specificato nel documento dei requisiti.	
METODI	**DESCRIZIONE**
public static ComplexNumber sum(ComplexNumber n1, ComplexNumber n2)	Metodo statico che permette di effettuare la somma di due numeri complessi. Restituisce un numero complesso.
public static ComplexNumber sub(ComplexNumber n1, ComplexNumber n2)	Metodo statico che permette di effettuare la sottrazione di due numeri complessi. Viene sottratto al numero n1 il numero n2. Restituisce un numero complesso.
public static ComplexNumber multiply(ComplexNumber n1, ComplexNumber n2)	Metodo statico che permette di effettuare la moltiplicazione di due numeri complessi. Restituisce un numero complesso.
public static ComplexNumber divide(ComplexNumber n1, ComplexNumber n2)	Metodo statico che permette di effettuare la divisione di due numeri complessi. Viene diviso il numero n1, n2 volte. Restituisce un numero complesso.

Classe UnaryCanonicOperations: classe final (non potrà essere ereditata). Tale classe implementa i metodi utili ad effettuare le operazioni canoniche unarie come specificato nel documento dei requisiti.	
METODI	**DESCRIZIONE**
public static ComplexNumber[] squareRoot(ComplexNumber n)	Metodo statico che permette di effettuare la radice quadrata di un numero complesso. Restituisce un numero complesso.
public static ComplexNumber changeSign(ComplexNumber n)	Metodo statico che permette di effettuare l'operazione di cambio segno su un numero complesso. Restituisce un numero complesso.

Classe Stack: classe che implementa la struttura stack che sta alla base del funzionamento della calcolatrice. Tale classe implementa anche i metodi tali da poter realizzare le funzionalità da eseguire sullo stack, come da documento dei requisiti.	
ATTRIBUTI	**DESCRIZIONE**
private Deque<ComplexNumber> stack	Attributo che permette di implementare una struttura dati di tipo stack.
METODI	**DESCRIZIONE**
public Stack() : <<constructor>>	Costruttore della classe Stack; non accetta parametri.
public void push(ComplexNumber input)	Metodo che inserisce un elemento in testa allo stack.
public ComplexNumber pop()	Metodo che preleva un elemento dalla testa dello stack rimuovendolo.
public ComplexNumber top()	Metodo che preleva un elemento dalla testa dello stack senza rimuoverlo.
public void clear()	Metodo che rimuove tutti gli elementi presenti all'interno dello stack.
public void drop()	Metodo che rimuove l'elemento in testa allo stack, senza restituirlo.
public void swap()	Metodo che permette di scambiare di posizione gli ultimi due elementi inseriti nello stack.
public void duplicate()	Metodo che permette di duplicare l'elemento in cima allo stack. La copia viene posizionata, a sua volta, in cima.
public void over()	Metodo che permette di duplicare l'elemento situato in seconda posizione dello stack, partendo della cima. Il duplicato viene posizionato in cima allo stack.

Classe Vars: classe che permette la gestione della sezione delle variabili. In tale classe sono presenti i metodi che implementano le funzionalità utili a manipolare le variabili, come da documento dei requisiti.	
ATTRIBUTI	**DESCRIZIONE**
private Map<Character, ComplexNumber> variables	Struttura dati di tipo Map che permette di memorizzare la coppia <variabile,valore>.
private Stack stack	Attributo che permette di mantenere in memoria un riferimento ad un oggetto di tipo Stack.
METODI	**DESCRIZIONE**
public Vars(Stack stack) : <<constructor>>	Costruttore della classe Vars a cui bisogna passare come attributo un oggetto di tipo Stack.
public void popFromStack(Character variable)	Metodo che permette di salvare all'interno della variabile passata come parametro, la cima dello stack.
public void pushInStack(Character variable)	Metodo che permette di inserire in cima allo stack il valore della variabile passata come parametro.
public void sumFromStack(Character variable)	Metodo che permette di sommare al valore della variabile passata come argomento la cima dello stack.
public void subFromStack(Character variable)	Metodo che permette di sottrarre al valore della variabile passata come argomento la cima dello stack.
public ComplexNumber getValueOf(Character variable)	Metodo che restituisce il valore del numero complesso associato alla variabile passata come argomento.
public void setValueOf(Character variable, ComplexNumber n)	Metodo che assegna il valore del numero complesso passato come argomento alla variabile passata come argomento.
public ArrayList<String> toStringArrayList()	Metodo che restituisce un ArrayList di stringhe il cui contenuto è pari al nome delle variabili inizializzate con il rispettivo valore. Tale metodo permette di avere una struttura iterabile che consenta di aggiornare la visualizzazione dello stato delle variabili su interfaccia grafica.

Classe ScientificCalculator: classe che implementa la calcolatrice scientifica, vista come modello con cui l'interfaccia deve interagire.	
ATTRIBUTI	**DESCRIZIONE**
private Stack stack	Attributo che memorizza il riferimento ad un oggetto di tipo Stack.
private Vars vars	Attributo che memorizza il riferimento ad un oggetto di tipo Vars.
METODI	**DESCRIZIONE**
public ScientificCalculator() : <<constructor>>	Costruttore della classe ScientificCalculator. Non accetta parametri.
public void execute(int input)	Metodo che permette di eseguire l'operazione corrispondente all'intero passato come parametro.
public void executeOnVariable(Character variable, Character operation)	Metodo che permette di eseguire l'operazione passata 'operation' sulla variabile 'variable' (passati come argomento).
public void sum()	Metodo che effettua la somma tra i primi due elementi presenti nello stack, partendo dalla cima. Il risultato viene posto in cima allo stack.
public void sub()	Metodo che effettua la sottrazione tra i primi due elementi presenti nello stack, partendo dalla cima, al secondo elemento viene sottratto il primo. Il risultato viene posto in cima allo stack.
public void multiply()	Metodo che effettua la moltiplicazione tra i primi due elementi presenti nello stack, partendo dalla cima. Il risultato viene posto in cima allo stack.
public void divide()	Metodo che effettua la divisione tra i primi due elementi presenti nello stack, partendo dalla cima, al secondo elemento viene diviso con il primo. Il risultato viene posto in cima allo stack.
public void squareRoot()	Metodo che effettua la radice quadrata dell'elemento in cima allo stack. Il risultato viene posto, a sua volta, in cima allo stack.
public void changeSign()	Metodo che effettua l'operazione di cambio segno dell'elemento in cima allo stack. Il risultato viene posto in cima allo stack.

public void insertComplexNumber(String input)	Metodo che permette l'inserimento di un numero complesso in cima allo stack.
private ComplexNumber formatComplexNumber(String input)	Metodo che restituisce un oggetto di tipo ComplexNumber a partire dalla stringa ricevuta come parametro.

2.1.1.3. PACKAGE EXCEPTIONS

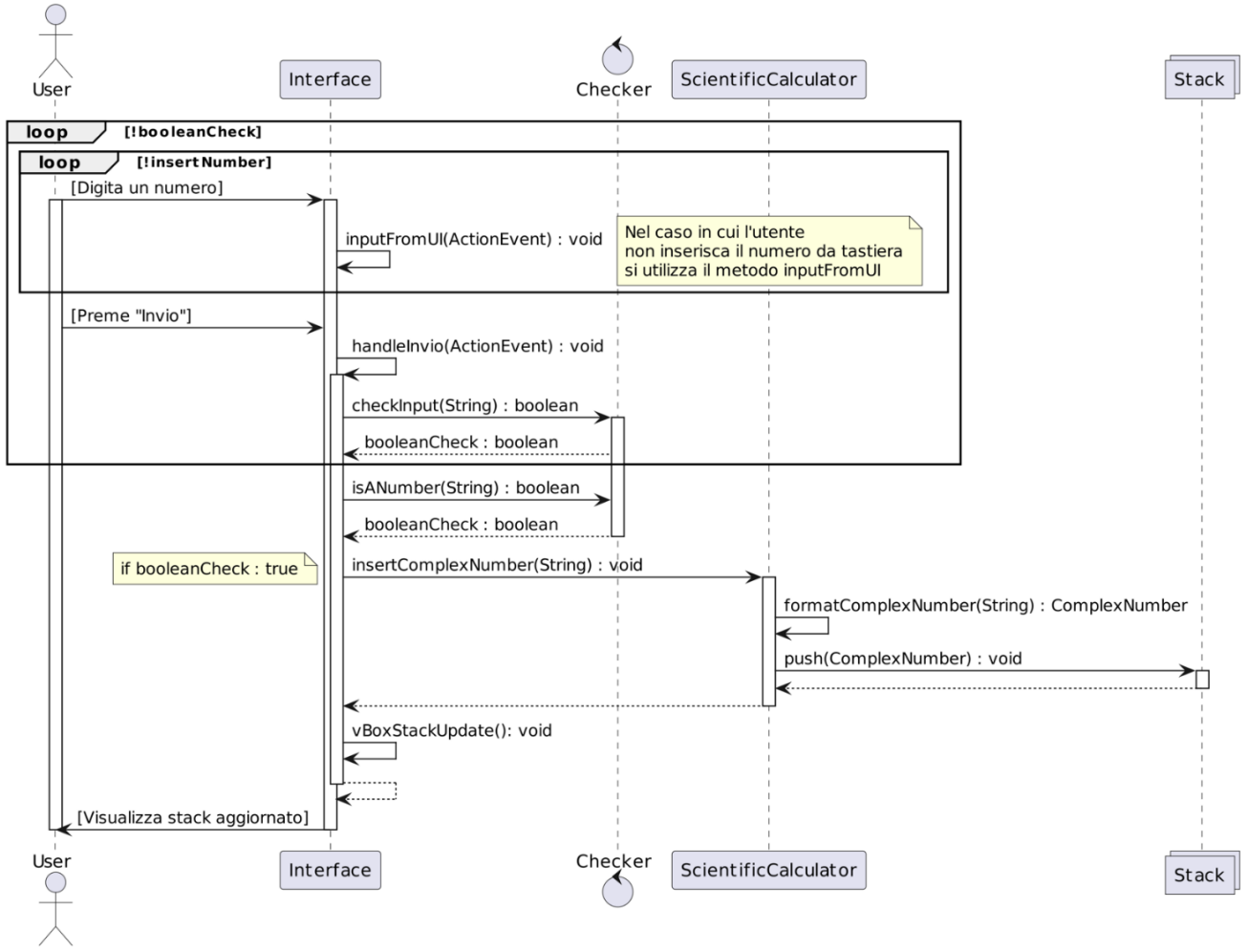
Classe InvalidOperandsException: classe delegata alla gestione dell'errore prodotto da un numero non supportato di operandi. Estende la classe RunTimeException.	
METODI	**DESCRIZIONE**
public InvalidOperandsException(): <<constructor>>	Costruttore per l'eccezione InvalidOperandsException. Non accetta parametri.
public InvalidOperandsException(String msg) : <<constructor>>	Costruttore per l'eccezione InvalidOperandsException. Accetta come parametro una stringa.

Classe InvalidInputException: classe delegata alla gestione dell'errore prodotto da un formato non corretto dell'input. Estende la classe RunTimeException.	
METODI	**DESCRIZIONE**
public InvalidInputException() : <<constructor>>	Costruttore per l'eccezione InvalidInputException. Non accetta parametri.
public InvalidInputException(String msg) : <<constructor>>	Costruttore per l'eccezione InvalidInputException. Accetta come parametro una stringa.

Classe UninitializedVariableException: classe delegata alla gestione dell'errore prodotto dal tentativo di accesso ad una variabile non inizializzata. Estende la classe RunTimeException.	
METODI	**DESCRIZIONE**
public UninitializedVariableException() : <<constructor>>	Costruttore per l'eccezione UninitializedVariableException. Non accetta parametri.
public UninitializedVariableException(String msg) : <<constructor>>	Costruttore per l'eccezione UninitializedVariableException. Accetta come parametro una stringa.

2.2. SEQUENCE DIAGRAMS

2.2.1. Sequence Diagram UC.1



Descrizione Sequence Diagram UC.1

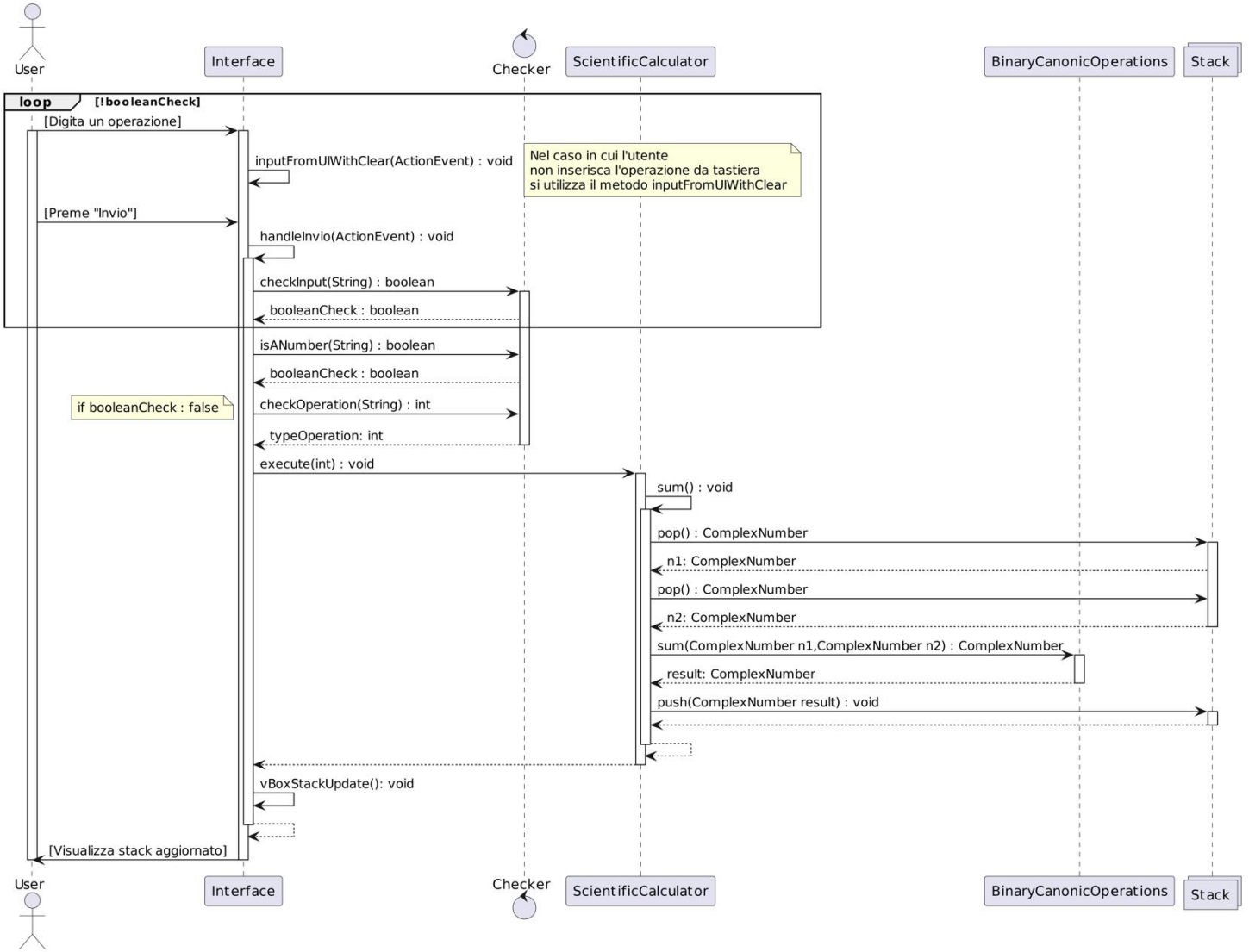
Il diagramma di sequenza rappresenta l'interazione che avviene tra l'attore Utente e la calcolatrice per eseguire l'operazione di inserimento di un numero nello Stack. In questo sequence diagram l'utente interagisce con diversi partecipanti: l'interface, rappresentante l'unione di view e controller, il checker, rappresentante il partecipante delegato al controllo dei vincoli imposti dal Progetto, la classe ScientificCalculator che si occupa di gestire tutte le operazioni effettuate dall'utente sulla calcolatrice, infine lo stack che rappresenta la struttura dati impiegata alla gestione dei numeri inseriti.

All'inizio l'utente inserisce un numero attraverso l'ausilio dell'interfaccia grafica o della tastiera attivando contestualmente il flusso di esecuzione dell'interface. Quest'ultima, sfruttando I metodi adibiti al controllo di inserimento, aggiorna il textfield di input. L'utente a questo punto può completare l'inserimento del numero che terminerà soltanto con la pressione del tasto "Invio". Si tenga presente che in questa fase l'utente potrebbe anche decidere di pulire la visualizzazione del textfield di input attraverso la digitazione del tasto "Delete" (SD 5).

A questo punto, una volta schiacciato "Invio", il checker controllerà il formato del numero inserito che, se dovesse risultare corretto verrebbe immesso nello stack mediante l'utilizzo della classe ScientificCalculator. Il checker potrebbe anche lanciare un alert se il formato del numero inserito fosse inaspettato, in quest'ultimo caso il text-field di input viene ripulito in attesa di una nuova sessione di inserimento.

Se il numero viene aggiunto allo stack, l'interfaccia si aggiorna con una chiamata al metodo vBoxStackUpdate. L'utente può finalmente visualizzare le modifiche.

2.2.2. *Sequence Diagrams UC.2.1*



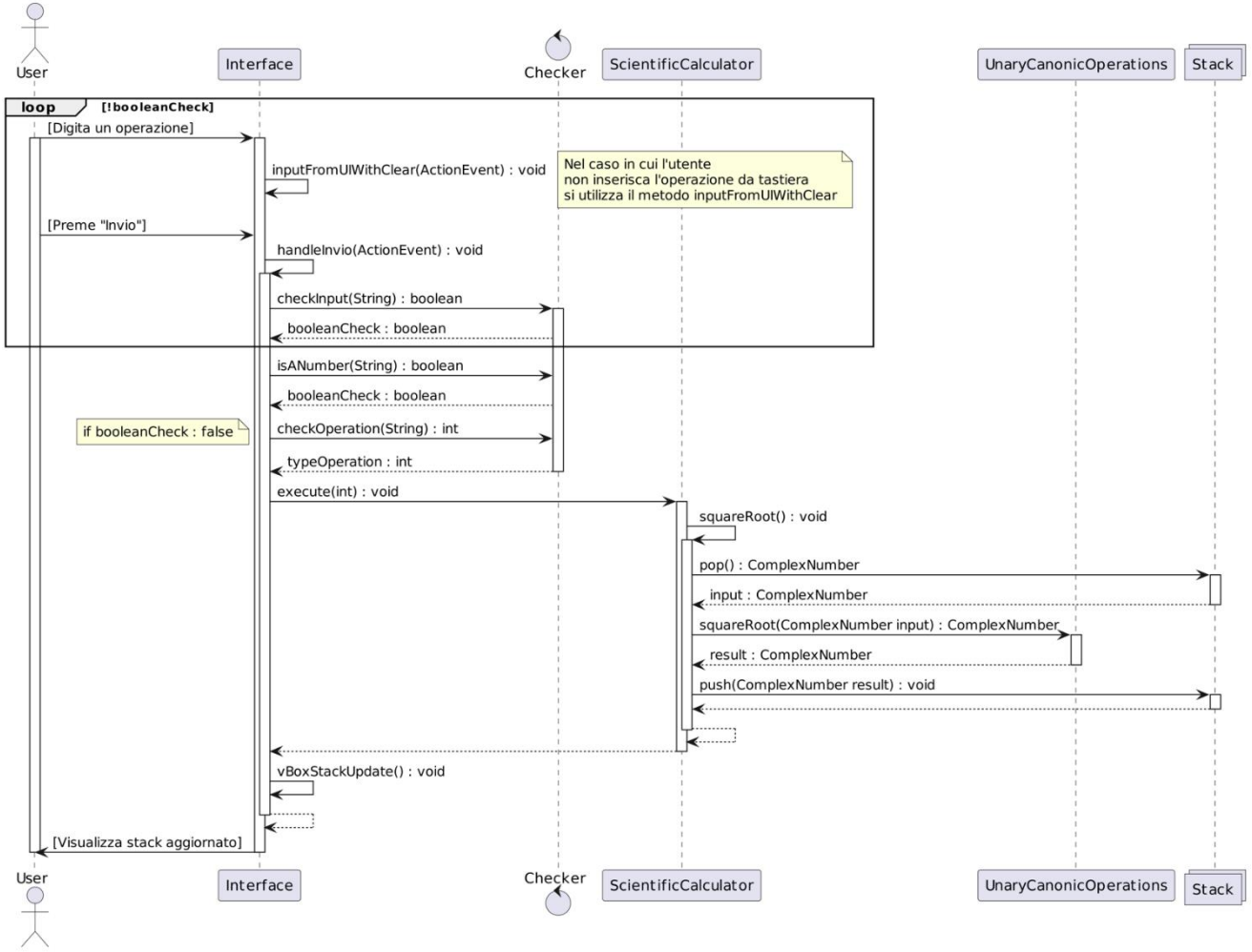
Descrizione Sequence Diagram UC.2.1

Il diagramma di sequenza rappresenta l'interazione che avviene tra l'attore Utente e la calcolatrice per eseguire l'operazione di inserimento di un'operazione. In questo sequence diagram l'utente interagisce con diversi partecipanti: l'interface, rappresentante l'unione di view e controller, il checker, rappresentante il partecipante delegato al controllo dei vincoli imposti dal Progetto, la classe ScientificCalculator che si occupa di gestire tutte le operazioni effettuate dall'utente sulla calcolatrice, la classe BinaryCanonicOperations, la quale gestisce l'esecuzione di operazioni aritmetiche binarie, infine lo stack che rappresenta la struttura dati impiegata alla gestione dei numeri inseriti.

La sequenza comincia con l'inserimento, da parte dell'utente, di un'operazione, questo può avvenire mediante l'interfaccia grafica o la tastiera. I due casi vengono gestiti attraverso metodi diversi dall'interface; ma entrambi i casi rimangono in attesa della pressione del tasto "Invio". A questo punto il cheker controlla che l'operazione può essere eseguita correttamente con I numeri presenti nello stack, se non lo fosse viene lanciata un'eccezione che ha lo scopo di informare l'utente sull'errore: l'errore viene bypassato pulendo il text-field di input.

Se il checker accetta l'operazione (i prerequisiti per l'esecuzione di questa sono verificati), esso deve anche discriminare il tipo di operazione richiesta. L'interface (il suo controller) provvede a richiamare l'operazione che gli è stata precedentemente specificata dalla classe ScientificCalculator. Quest'ultima deve provvedere ad eseguire la logica alla base dell'operazione, aggiornando lo stack con il risultato. Il controllo è passato nuovamente all'interface che si aggiorna con una chiamata al metodo vBoxStackUpdate. L'utente può finalmente visualizzare le modifiche.

2.2.3. *Sequence Diagram UC.2.2*



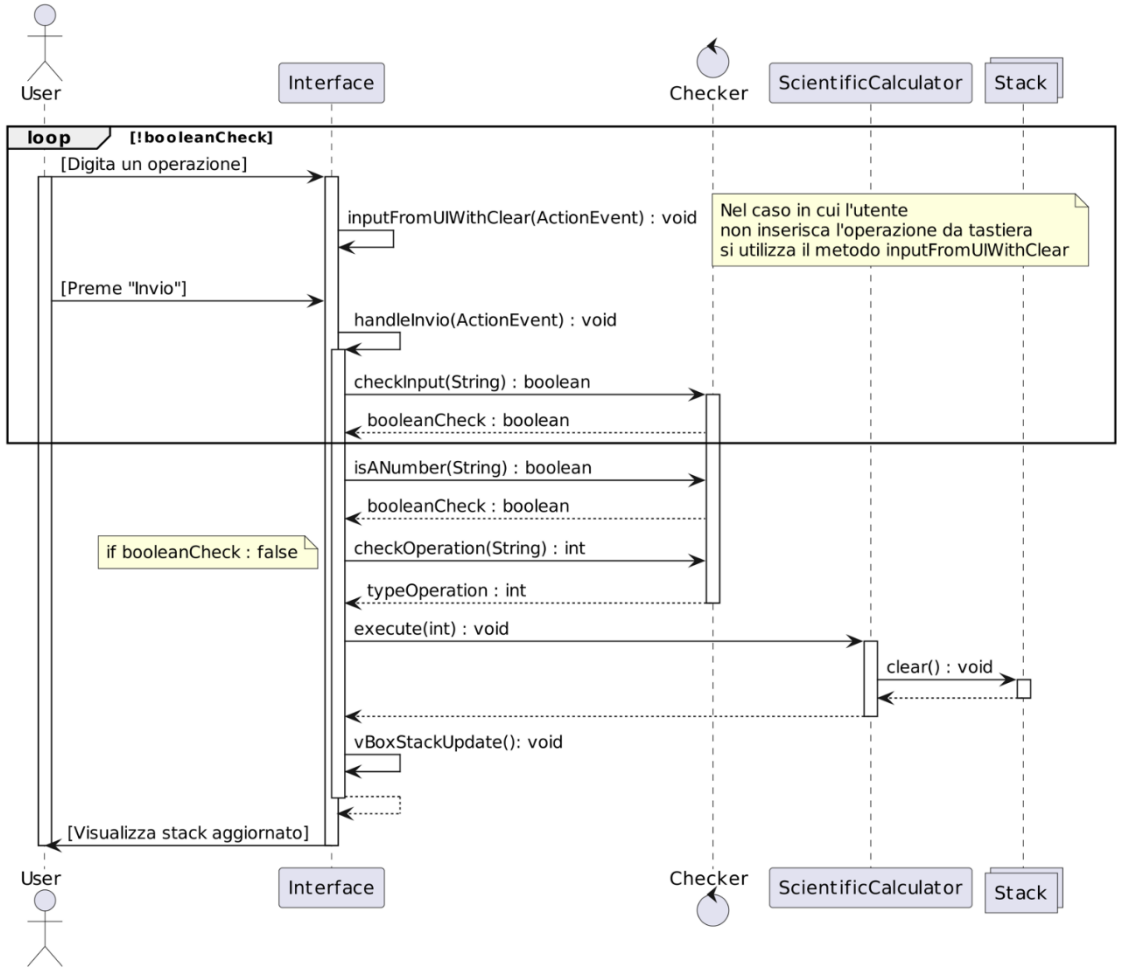
Descrizione Sequence Diagram UC.2.2

Il diagramma di sequenza rappresenta l'interazione che avviene tra l'attore Utente e la calcolatrice per eseguire l'operazione di inserimento di un'operazione. In questo sequence diagram l'utente interagisce con diversi partecipanti: l'interface, rappresentante l'unione di view e controller, il checker, rappresentante il partecipante delegato al controllo dei vincoli imposti dal Progetto, la classe ScientificCalculator che si occupa di gestire tutte le operazioni effettuate dall'utente sulla calcolatrice, la classe UnaryCanonicOperations, la quale gestisce l'esecuzione di operazioni aritmetiche unarie, infine lo stack che rappresenta la struttura dati impiegata alla gestione dei numeri inseriti.

La sequenza comincia con l'inserimento, da parte dell'utente, di un'operazione, questo può avvenire mediante l'interfaccia grafica o la tastiera. I due casi vengono gestiti attraverso metodi diversi dall'interface; ma entrambi i casi rimangono in attesa della pressione del tasto "Invio". A questo punto il cheker controlla che l'operazione può essere eseguita correttamente con I numeri presenti nello stack, se non lo fosse viene lanciata un'eccezione che ha lo scopo di informare l'utente sull'errore: l'errore viene bypassato pulendo il text-field di input.

Se il checker accetta l'operazione (i prerequisiti per l'esecuzione di questa sono verificati), esso deve anche discriminare il tipo di operazione richiesta. L'interface (il suo controller) provvede a richiamare l'operazione che gli è stata precedentemente specificata dalla classe ScientificCalculator. Quest'ultima deve provvedere ad eseguire la logica alla base dell'operazione, aggiornando lo stack con il risultato. Il controllo è passato nuovamente all'interface che si aggiorna con una chiamata al metodo vBoxStackUpdate. L'utente può finalmente visualizzare le modifiche.

2.2.4. Sequence Diagram UC.3.1



Descrizione Sequence Diagram UC.3.1

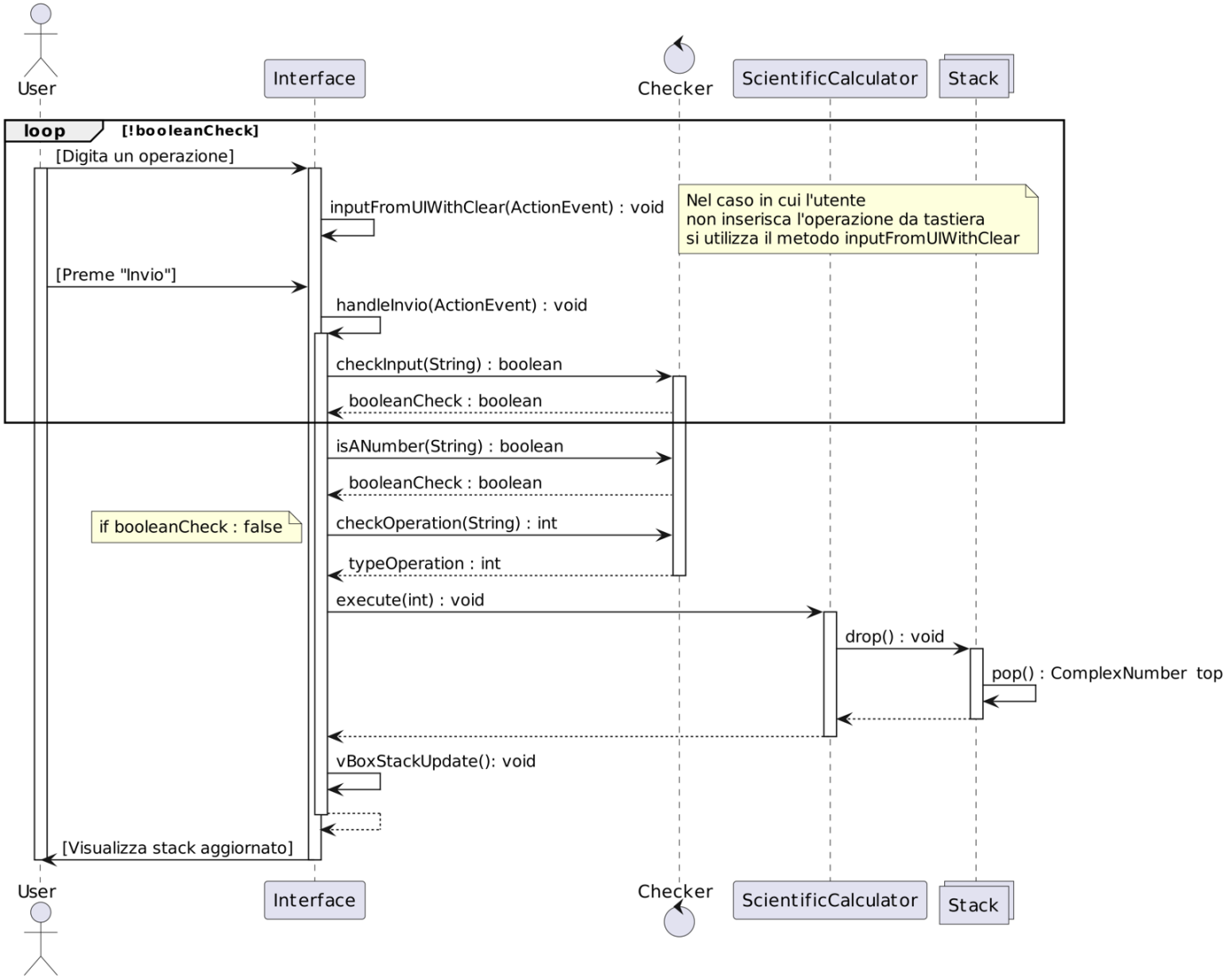
Il diagramma di sequenza rappresenta l'interazione che avviene tra l'attore Utente e la calcolatrice per eseguire l'operazione di clear sullo stack. In questo sequence diagram l'utente interagisce con diversi partecipanti: l'interface, rappresentante l'unione di view e controller, il checker, rappresentante il partecipante delegato al controllo dei vincoli imposti dal Progetto, la classe ScientificCalculator che si occupa di gestire tutte le operazioni effettuate dall'utente sulla calcolatrice, infine lo stack che rappresenta la struttura dati impiegata alla gestione delle operazioni selezionate.

La sequenza comincia con l'inserimento, da parte dell'utente, dell'operazione di clear, questo può avvenire mediante l'interfaccia grafica o la tastiera. I due casi vengono gestiti attraverso metodi diversi dall'interface; ma in entrambi i casi rimangono in attesa della pressione del tasto "Invio". Il checker effettua i controlli che vengono normalmente eseguiti per tutte le operazioni, ma si tenga presente che in questo specifico caso nessuna eccezione viene lanciata, in quanto l'operazione non ne comprende.

Il checker invia un tipo intero di ritorno all'interface associato all'operazione di clear. L'interface provvede quindi a lanciare il metodo appropriato per l'operazione sulla classe ScientificCalculator, che ripulisce lo stack.

Il controllo è passato nuovamente all'interface che si aggiorna con una chiamata al metodo vBoxStackUpdate. L'utente può finalmente visualizzare le modifiche.

2.2.5. *Sequence Diagram UC.3.2*



Descrizione Sequence Diagram UC 3.2

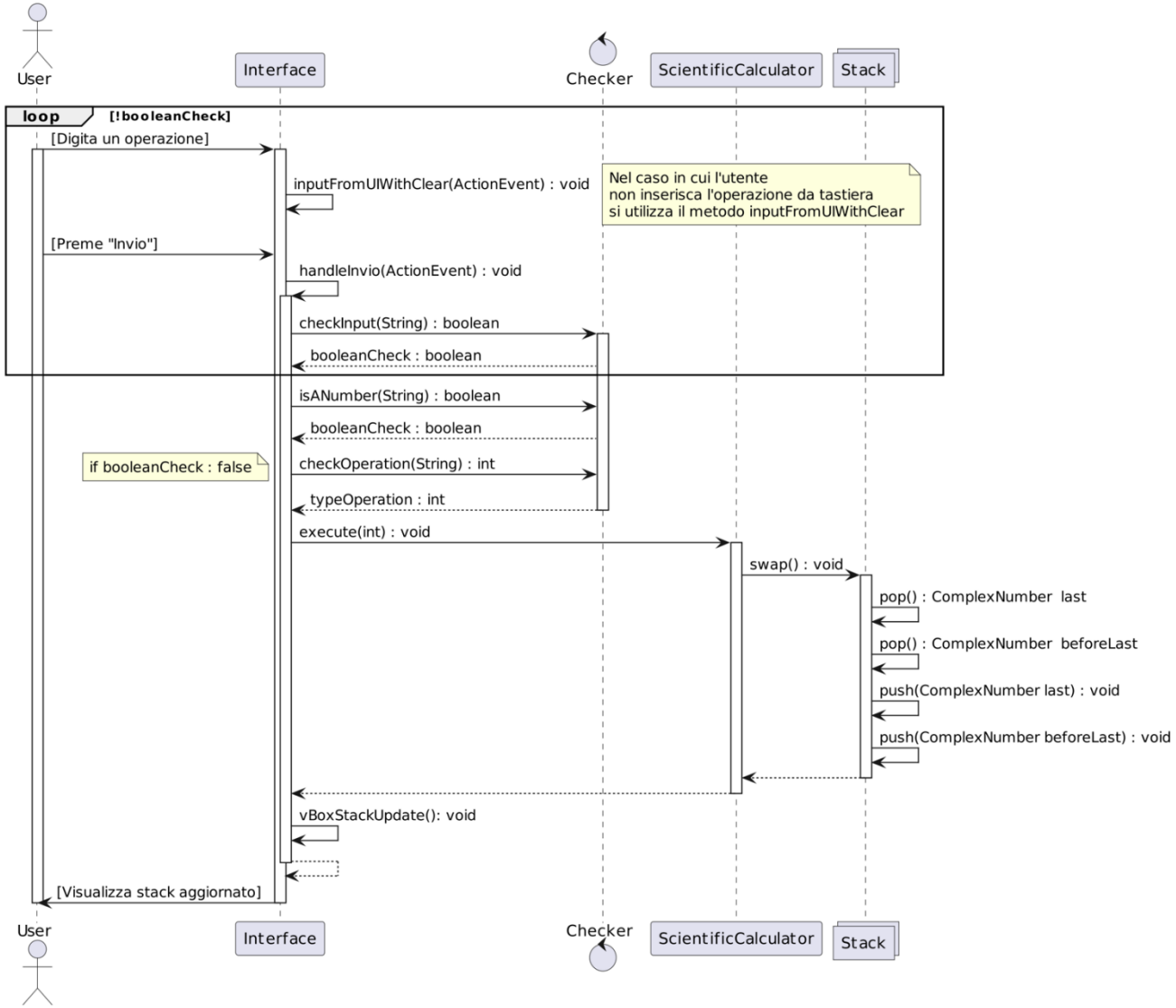
Il diagramma di sequenza rappresenta l'interazione che avviene tra l'attore Utente e la calcolatrice per eseguire l'operazione di drop sullo stack. In questo sequence diagram l'utente interagisce con diversi partecipanti: l'interface, rappresentante l'unione di view e controller, il checker, rappresentante il partecipante delegato al controllo dei vincoli imposti dal Progetto, la classe ScientificCalculator che si occupa di gestire tutte le operazioni effettuate dall'utente sulla calcolatrice, infine lo stack che rappresenta la struttura dati impiegata alla gestione delle operazioni selezionate.

La sequenza comincia con l'inserimento, da parte dell'utente, dell'operazione di drop, questo può avvenire mediante l'interfaccia grafica o la tastiera. I due casi vengono gestiti attraverso metodi diversi dall'interface; ma in entrambi i casi rimangono in attesa della pressione del tasto "Invio". Il checker effettua i controlli che vengono normalmente eseguiti per tutte le operazioni, ma si tenga presente che in questo specifico caso nessuna eccezione viene lanciata, in quanto l'operazione non ne comprende.

Il checker invia un tipo intero di ritorno all'interface associato all'operazione di drop. L'interface provvede quindi a lanciare il metodo appropriato per l'operazione sulla classe ScientificCalculator, che cancella il top dello stack.

Il controllo è passato nuovamente all'interface che si aggiorna con una chiamata al metodo vBoxStackUpdate. L'utente può finalmente visualizzare le modifiche.

2.2.6. *Sequence Diagram UC.3.3*



Descrizione Sequence Diagram UC.3.3

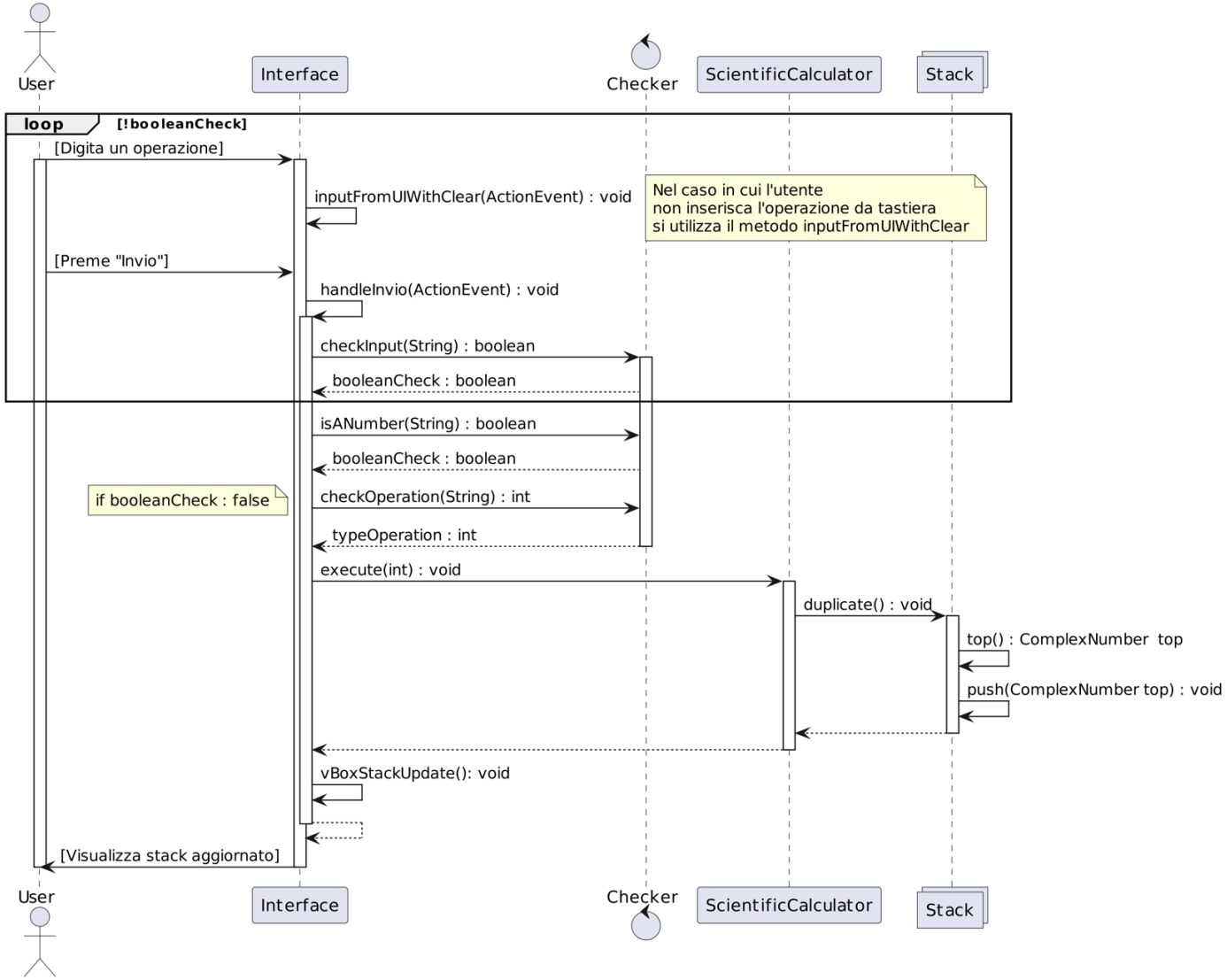
Il diagramma di sequenza rappresenta l'interazione che avviene tra l'attore Utente e la calcolatrice per eseguire l'operazione di swap sullo stack. In questo sequence diagram l'utente interagisce con diversi partecipanti: l'interface, rappresentante l'unione di view e controller, il checker, rappresentante il partecipante delegato al controllo dei vincoli imposti dal Progetto, la classe ScientificCalculator che si occupa di gestire tutte le operazioni effettuate dall'utente sulla calcolatrice, infine lo stack che rappresenta la struttura dati impiegata alla gestione delle operazioni selezionate.

La sequenza comincia con l'inserimento, da parte dell'utente, dell'operazione di swap, questo può avvenire mediante l'interfaccia grafica o la tastiera. I due casi vengono gestiti attraverso metodi diversi dall'interface; ma in entrambi i casi rimangono in attesa della pressione del tasto "Invio". Il checker effettua i controlli che vengono normalmente eseguiti per tutte le operazioni e lancia un'eccezione laddove non siano presenti almeno due elementi nello stack: viene quindi specificata la condizione di errore e il flusso continua con la pulizia del fieldtext di input.

Il checker invia un tipo intero di ritorno all'interface associato all'operazione di swap. L'interface provvede quindi a lanciare il metodo appropriato per l'operazione sulla classe ScientificCalculator, che preleva i primi due elementi dallo stack e li inserisce nuovamente nell'ordine inverso.

Il controllo è passato nuovamente all'interface che si aggiorna con una chiamata al metodo vBoxStackUpdate. L'utente può finalmente visualizzare le modifiche.

2.2.7. *Sequence Diagram UC.3.4*



Descrizione Sequence Diagram UC.3.4

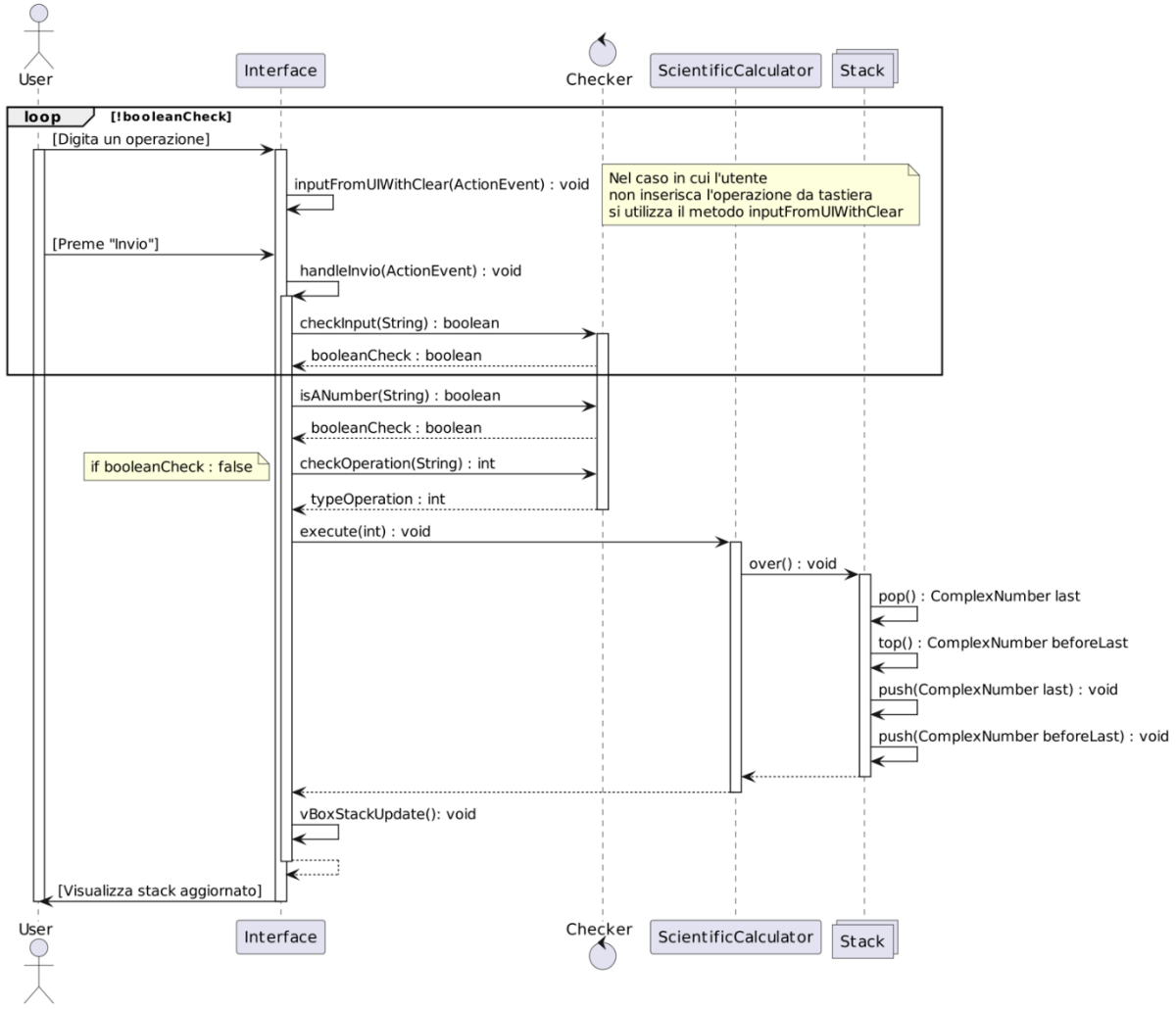
Il diagramma di sequenza rappresenta l'interazione che avviene tra l'attore Utente e la calcolatrice per eseguire l'operazione di duplicate sullo stack. In questo sequence diagram l'utente interagisce con diversi partecipanti: l'interface, rappresentante l'unione di view e controller, il checker, rappresentante il partecipante delegato al controllo dei vincoli imposti dal Progetto, la classe ScientificCalculator che si occupa di gestire tutte le operazioni effettuate dall'utente sulla calcolatrice, infine lo stack che rappresenta la struttura dati impiegata alla gestione delle operazioni selezionate.

La sequenza comincia con l'inserimento, da parte dell'utente, dell'operazione di duplicate, questo può avvenire mediante l'interfaccia grafica o la tastiera. I due casi vengono gestiti attraverso metodi diversi dall'interface; ma in entrambi i casi rimangono in attesa della pressione del tasto "Invio". Il checker effettua i controlli che vengono normalmente eseguiti per tutte le operazioni e lancia un'eccezione laddove non sia presente almeno un elemento nello stack: viene quindi specificata la condizione di errore e il flusso continua con la pulizia del fieldtext di input.

Il checker invia un tipo intero di ritorno all'interface associato all'operazione di duplicate. L'interface provvede quindi a lanciare il metodo appropriato per l'operazione sulla classe ScientificCalculator, che effettua la top sullo stack e ne pusha il valore ottenuto.

Il controllo è passato nuovamente all'interface che si aggiorna con una chiamata al metodo vBoxStackUpdate. L'utente può finalmente visualizzare le modifiche.

2.2.8. *Sequence Diagram UC.3.5*



Descrizione Sequence Diagram UC.3.5

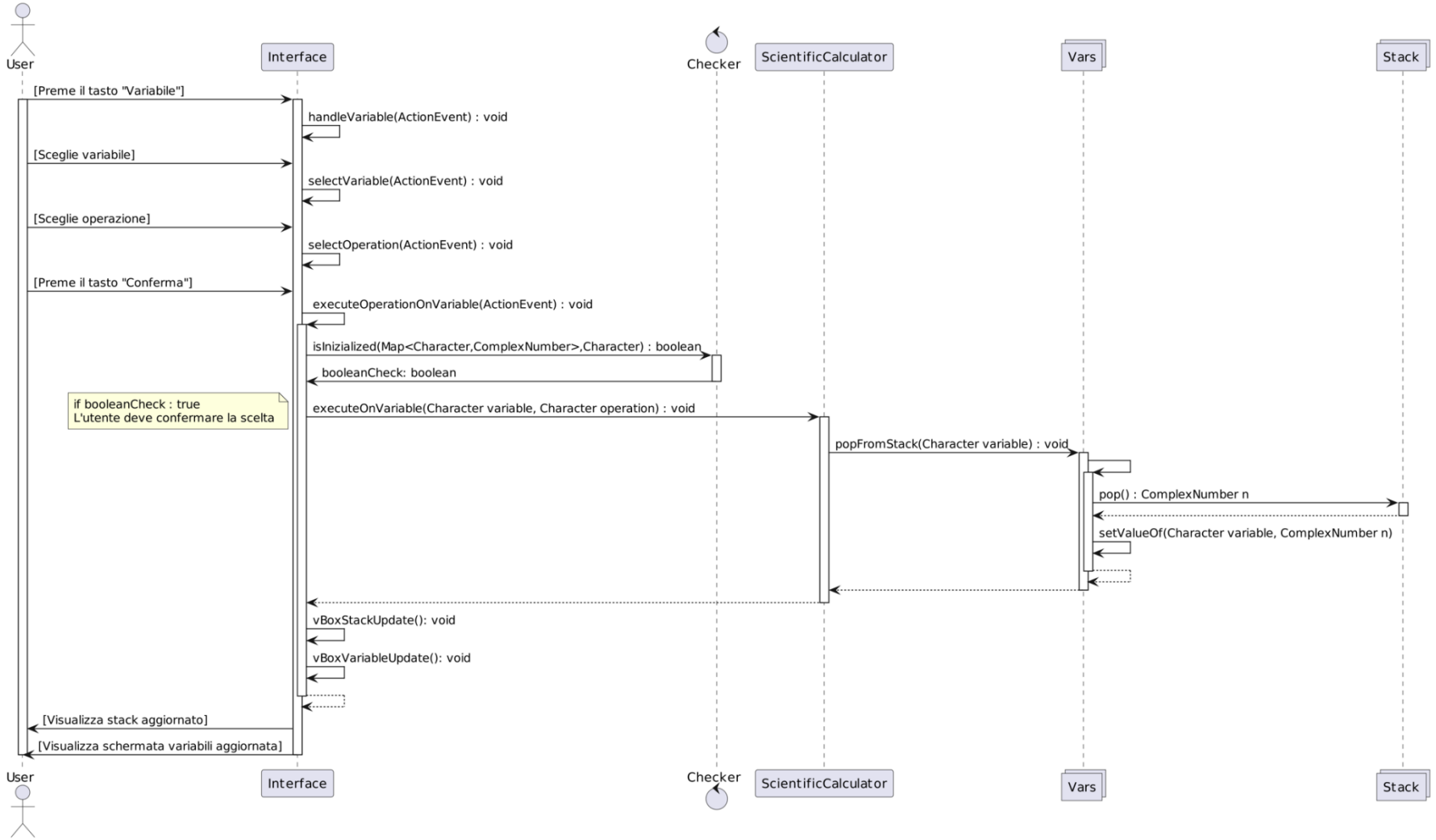
Il diagramma di sequenza rappresenta l'interazione che avviene tra l'attore Utente e la calcolatrice per eseguire l'operazione di over sullo stack. In questo sequence diagram l'utente interagisce con diversi partecipanti: l'interface, rappresentante l'unione di view e controller, il checker, rappresentante il partecipante delegato al controllo dei vincoli imposti dal Progetto, la classe ScientificCalculator che si occupa di gestire tutte le operazioni effettuate dall'utente sulla calcolatrice, infine lo stack che rappresenta la struttuta dati impiegata alla gestione delle operazioni selezionate.

La sequenza comincia con l'inserimento, da parte dell'utente, dell'operazione di over, questo può avvenire mediante l'interfaccia grafica o la tastiera. I due casi vengono gestiti attraverso metodi diversi dall'interface; ma in entrambi i casi rimangono in attesa della pressione del tasto "Invio". Il checker effettua i controlli che vengono normalmente eseguiti per tutte le operazioni e lancia un'eccezione laddove non siano presenti almeno due elementi nello stack: viene quindi specificata la condizione di errore e il flusso continua con la pulizia del fieldtext di input.

Il checker invia un tipo intero di ritorno all'interface associato all'operazione di over. L'interface provvede quindi a lanciare il metodo appropriato per l'operazione sulla classe ScientificCalculator: preleva l'ultimo elemento dallo stack, crea una copia del penultimo elemento, inserisce nuovamente l'ultimo elemento e, infine, reimmette il penultimo elemento. Questo processo assicura che il risultato finale sullo stack sia un duplicato del penultimo elemento.

Il controllo è passato nuovamente all'interface che si aggiorna con una chiamata al metodo vBoxStackUpdate. L'utente può finalmente visualizzare le modifiche.

2.2.9. *Sequence Diagram UC.4.1*



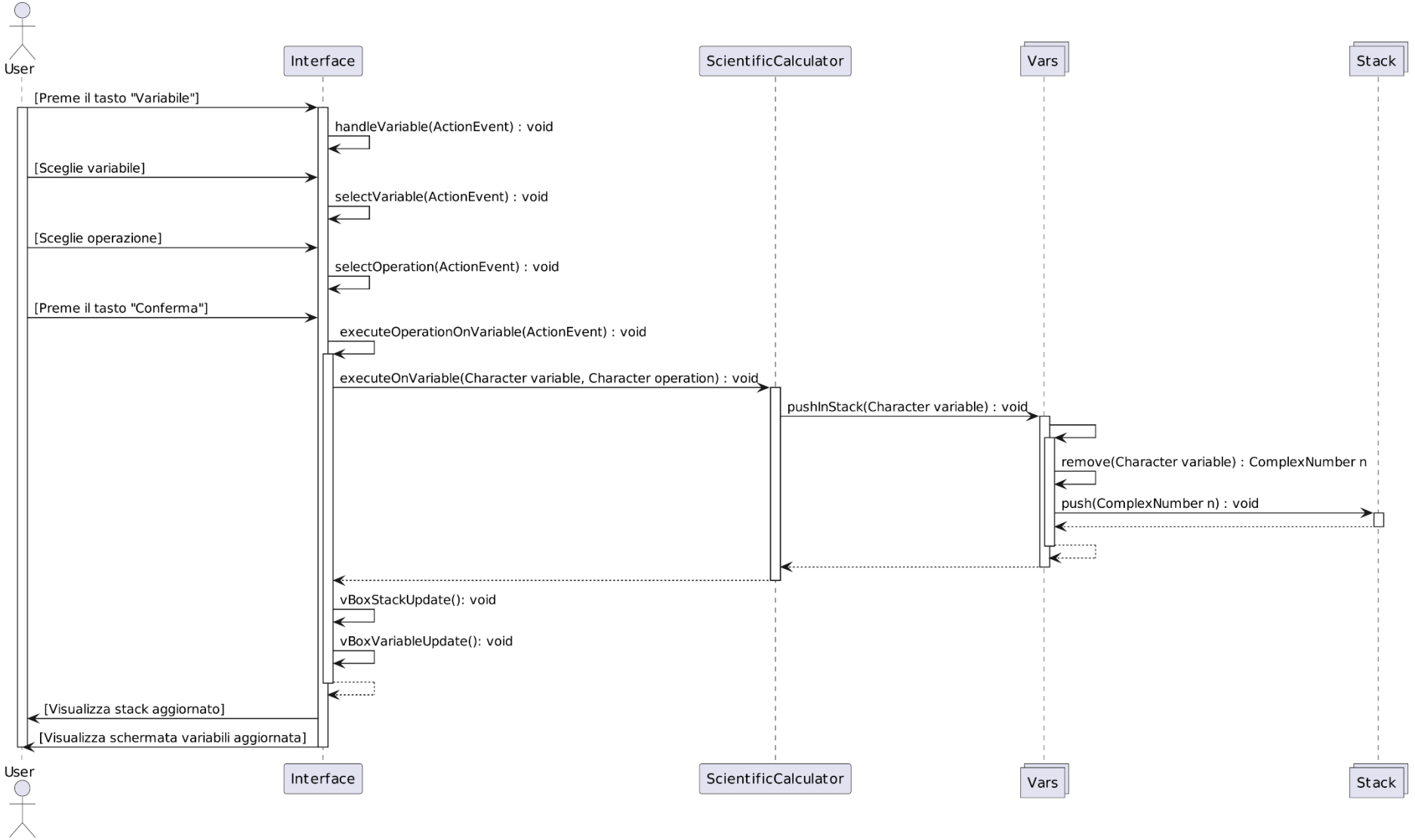
Descrizione Sequence Diagram UC.4.1

Il diagramma di sequenza rappresenta l'interazione che avviene tra l'attore Utente e la calcolatrice per eseguire l'operazione di prelievo di un elemento dallo stack e conseguente salvataggio in una variabile. In questo sequence diagram l'utente interagisce con diversi partecipanti: l'interface, rappresentante l'unione di view e controller, il checker, rappresentante il partecipante delegato al controllo dei vincoli imposti dal Progetto, la classe ScientificCalculator che si occupa di gestire tutte le operazioni effettuate dall'utente sulla calcolatrice, la classe Vars che si occupa della gestione delle variabili e delle loro operazioni, infine, lo stack che rappresenta la struttura dati impiegata alla gestione dei numeri inseriti.

La sequenza comincia con la selezione, da parte dell'utente, del tasto "Variabile", successivamente dovrà selezionare l'operazione da effettuare e su quale variabile. A questo punto l'utente può cliccare su "Annulla", riportando la calcolatrice allo stato iniziale annullando la selezione effettuata, oppure cliccare su "Conferma". Il checker controlla che la variabile selezionata non sia stata precedentemente inizializzata. Nel caso in cui lo sia viene mostrato un alert di conferma alla sovrascrittura della variabile. Se, invece, il controllo effettuato dal checker va a buon fine l'interface provvede quindi a lanciare il metodo appropriato all'operazione sulla classe Vars, che comunica con lo stack prelevando il top e depositandolo nella struttura dedicata all'immagazzinamento delle variabili.

Il controllo è passato nuovamente all'interface che si aggiorna con una chiamata al metodo vBoxStackUpdate e al metodo vBoxVariableUpdate. L'utente può finalmente visualizzare le modifiche.

2.2.10. Sequence Diagram UC.4.2



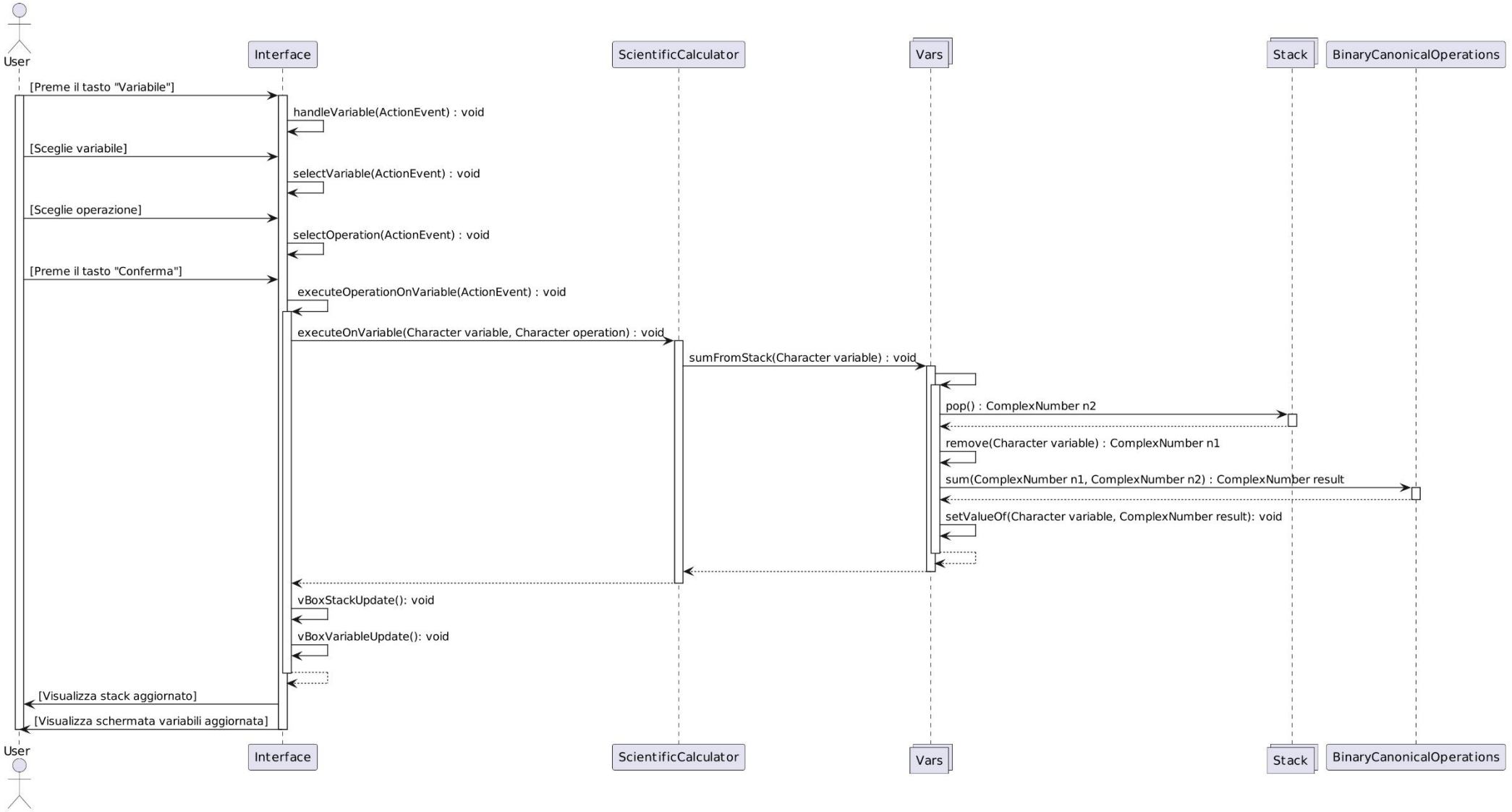
Descrizione Sequence Diagram UC.4.2

Il diagramma di sequenza rappresenta l'interazione che avviene tra l'attore Utente e la calcolatrice per eseguire l'operazione di inserimento del valore associato ad una variabile nel top dello stack. In questo sequence diagram l'utente interagisce con diversi partecipanti: l'interface, rappresentante l'unione di view e controller, la classe ScientificCalculator che si occupa di gestire tutte le operazioni effettuate dall'utente sulla calcolatrice, la classe Vars che si occupa della gestione delle variabili e delle loro operazioni, infine, lo stack che rappresenta la struttura dati impiegata alla gestione dei numeri inseriti.

La sequenza comincia con la selezione, da parte dell'utente, del tasto "Variabile", successivamente dovrà selezionare l'operazione da effettuare e su quale variabile. A questo punto l'utente può cliccare su "Annulla", riportando la calcolatrice allo stato iniziale annullando la selezione effettuata, oppure cliccare su "Conferma". A questo punto l'interface provvede quindi a lanciare il metodo appropriato all'operazione sulla classe Vars, che comunica con la struttura dedicata all'immagazzinamento delle variabili prelevando il valore associato alla variabile selezionata precedentemente dall'utente e lo inserisce nel top dello stack.

Il controllo è passato nuovamente all'interface che si aggiorna con una chiamata al metodo vBoxStackUpdate e al metodo vBoxVariableUpdate. L'utente può finalmente visualizzare le modifiche.

2.2.11. Sequence Diagram UC.4.3



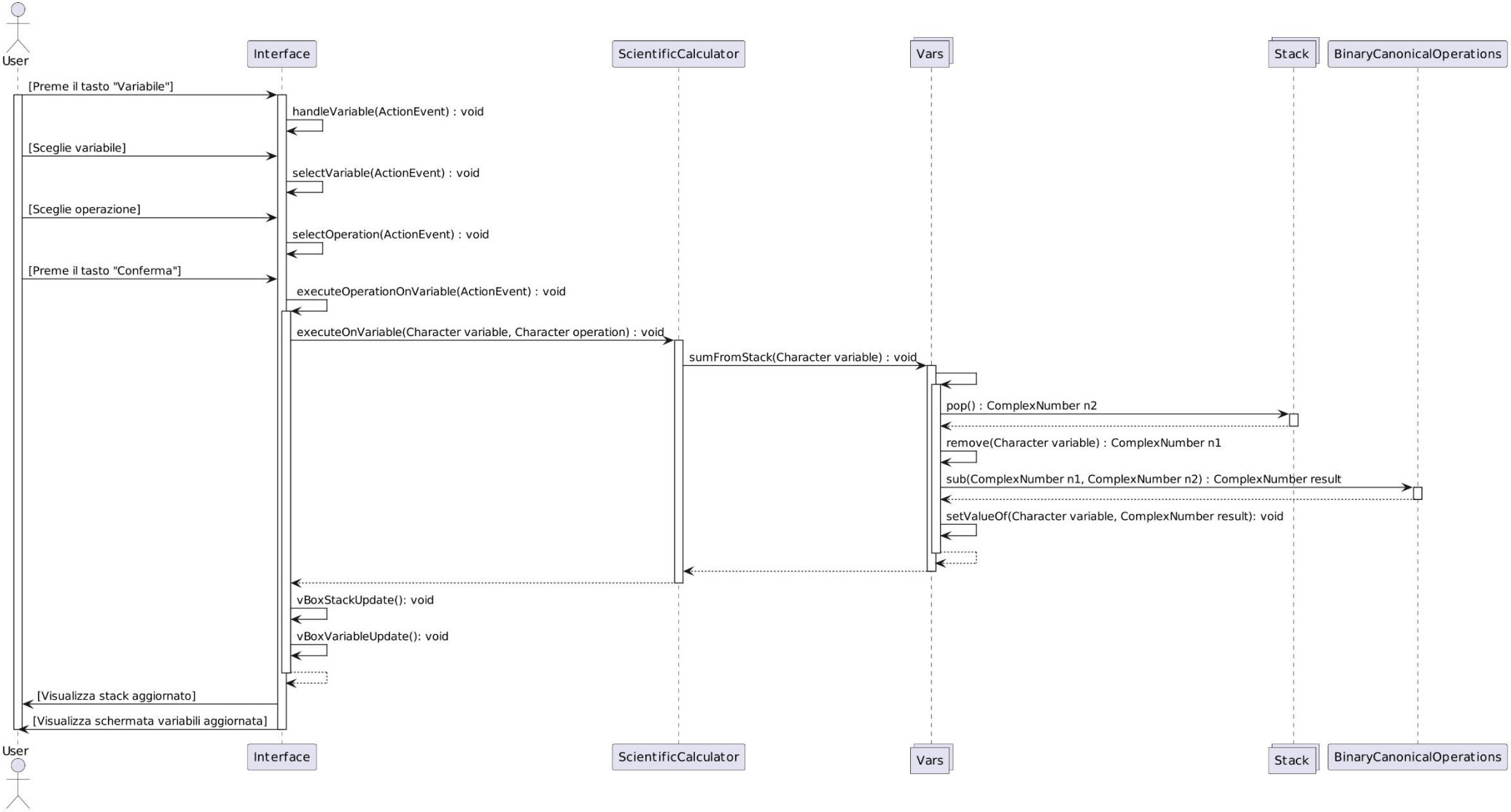
Descrizione Sequence Diagram UC4.3

Il diagramma di sequenza rappresenta l'interazione che avviene tra l'attore Utente e la calcolatrice per eseguire l'operazione di somma del valore associato ad una variabile all'elemento nel top dello stack. In questo sequence diagram l'utente interagisce con diversi partecipanti: l'interface, rappresentante l'unione di view e controller, la classe ScientificCalculator che si occupa di gestire tutte le operazioni effettuate dall'utente sulla calcolatrice, la classe Vars che si occupa della gestione delle variabili e delle loro operazioni, infine, lo stack che rappresenta la struttura dati impiegata alla gestione dei numeri inseriti.

La sequenza comincia con la selezione, da parte dell'utente, del tasto "Variabile", successivamente dovrà selezionare l'operazione da effettuare e su quale variabile. A questo punto l'utente può cliccare su "Annulla", riportando la calcolatrice allo stato iniziale annullando la selezione effettuata, oppure cliccare su "Conferma". A questo punto l'interface provvede quindi a lanciare il metodo appropriato all'operazione sulla classe Vars, che comunica da prima con lo stack prelevando l'elemento nel top e poi con la struttura dedicata all'immagazzinamento delle variabili prelevando il valore associato alla variabile selezionata precedentemente dall'utente. A questo punto la Vars chiamando il metodo sum (della BinaryCanonicOperation) ottiene il risultato e lo salva nella variabile.

Il controllo è passato nuovamente all'interface che si aggiorna con una chiamata al metodo vBoxStackUpdate e al metodo vBoxVariableUpdate. L'utente può finalmente visualizzare le modifiche.

2.2.12. Sequence Diagram UC.4.4



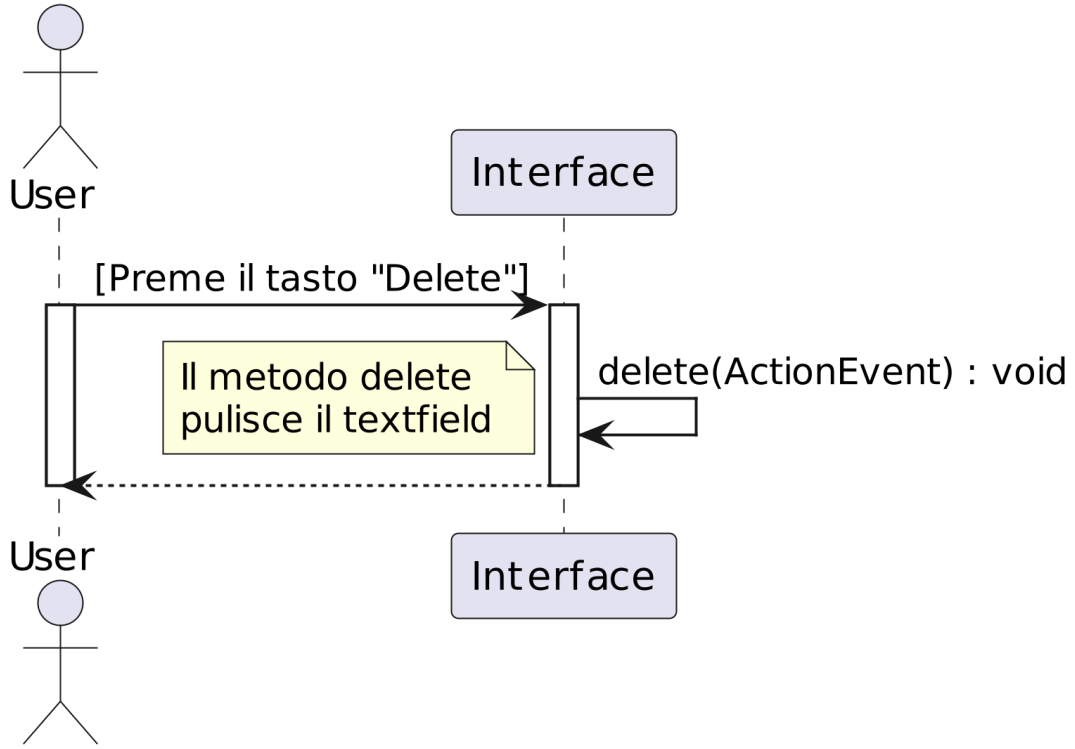
Descrizione Sequence Diagram UC.4.4

Il diagramma di sequenza rappresenta l'interazione che avviene tra l'attore Utente e la calcolatrice per eseguire l'operazione di sottrazione del top dello stack dalla variabile selezionata. In questo sequence diagram l'utente interagisce con diversi partecipanti: l'interface, rappresentante l'unione di view e controller, la classe ScientificCalculator che si occupa di gestire tutte le operazioni effettuate dall'utente sulla calcolatrice, la classe Vars che si occupa della gestione delle variabili e delle loro operazioni, infine, lo stack che rappresenta la struttura dati impiegata alla gestione dei numeri inseriti.

La sequenza comincia con la selezione, da parte dell'utente, del tasto "Variabile", successivamente dovrà selezionare l'operazione da effettuare e su quale variabile. A questo punto l'utente può cliccare su "Annulla", riportando la calcolatrice allo stato iniziale annullando la selezione effettuata, oppure cliccare su "Conferma". A questo punto l'interface provvede quindi a lanciare il metodo appropriato all'operazione sulla classe Vars, che comunica da prima con lo stack prelevando l'elemento nel top e poi con la struttura dedicata all'immagazzinamento delle variabili prelevando il valore associato alla variabile selezionata precedentemente dall'utente. A questo punto la Vars chiamando il metodo sub (della BinaryCanonicOperation) ottiene il risultato e lo salva nella variabile.

Il controllo è passato nuovamente all'interface che si aggiorna con una chiamata al metodo vBoxStackUpdate e al metodo vBoxVariableUpdate. L'utente può finalmente visualizzare le modifiche

2.2.13. Sequence Diagram UC.5



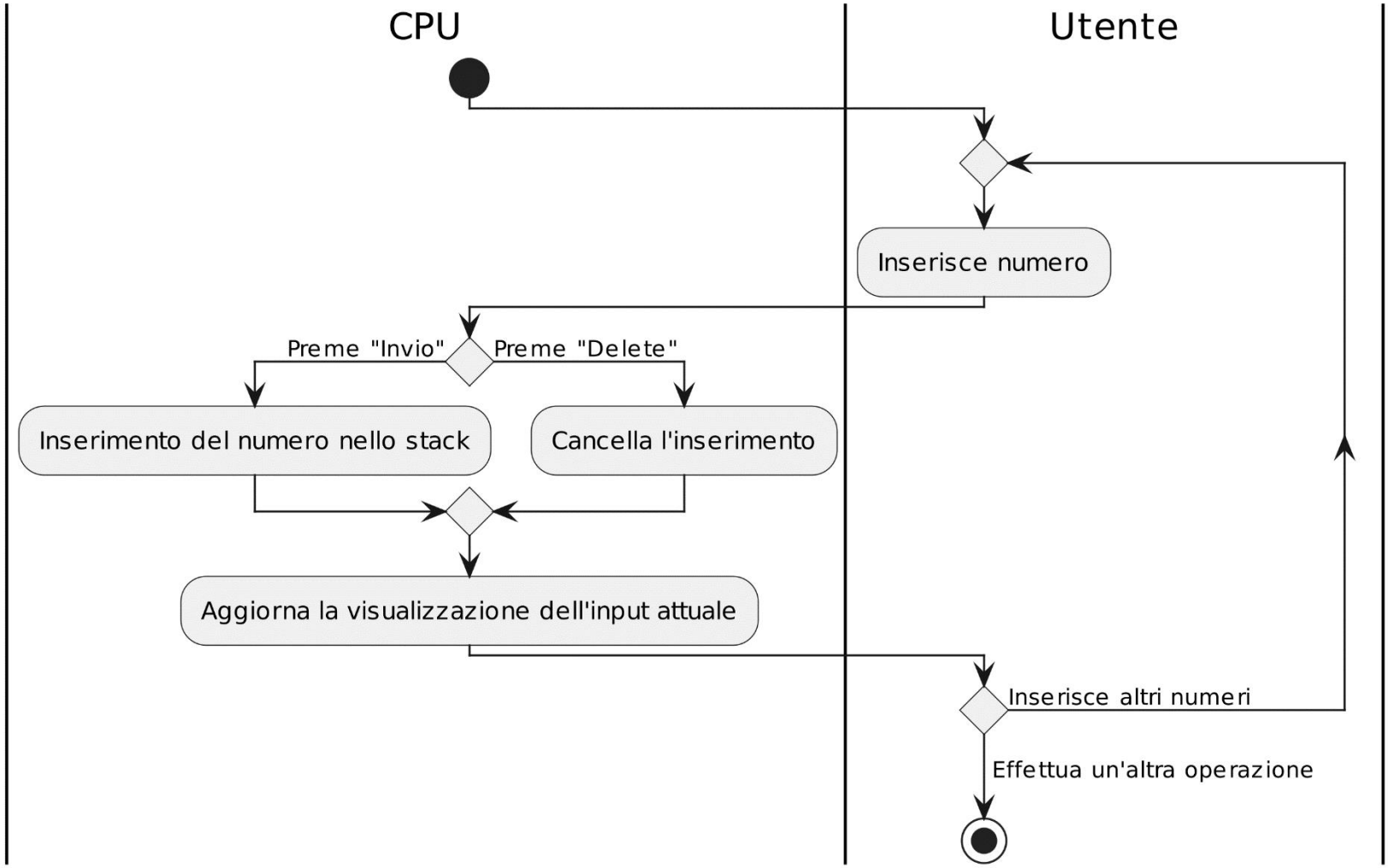
Descrizione Sequence Diagram UC.5

Il sequence diagram definisce la funzionalità di delete del textfield di input.

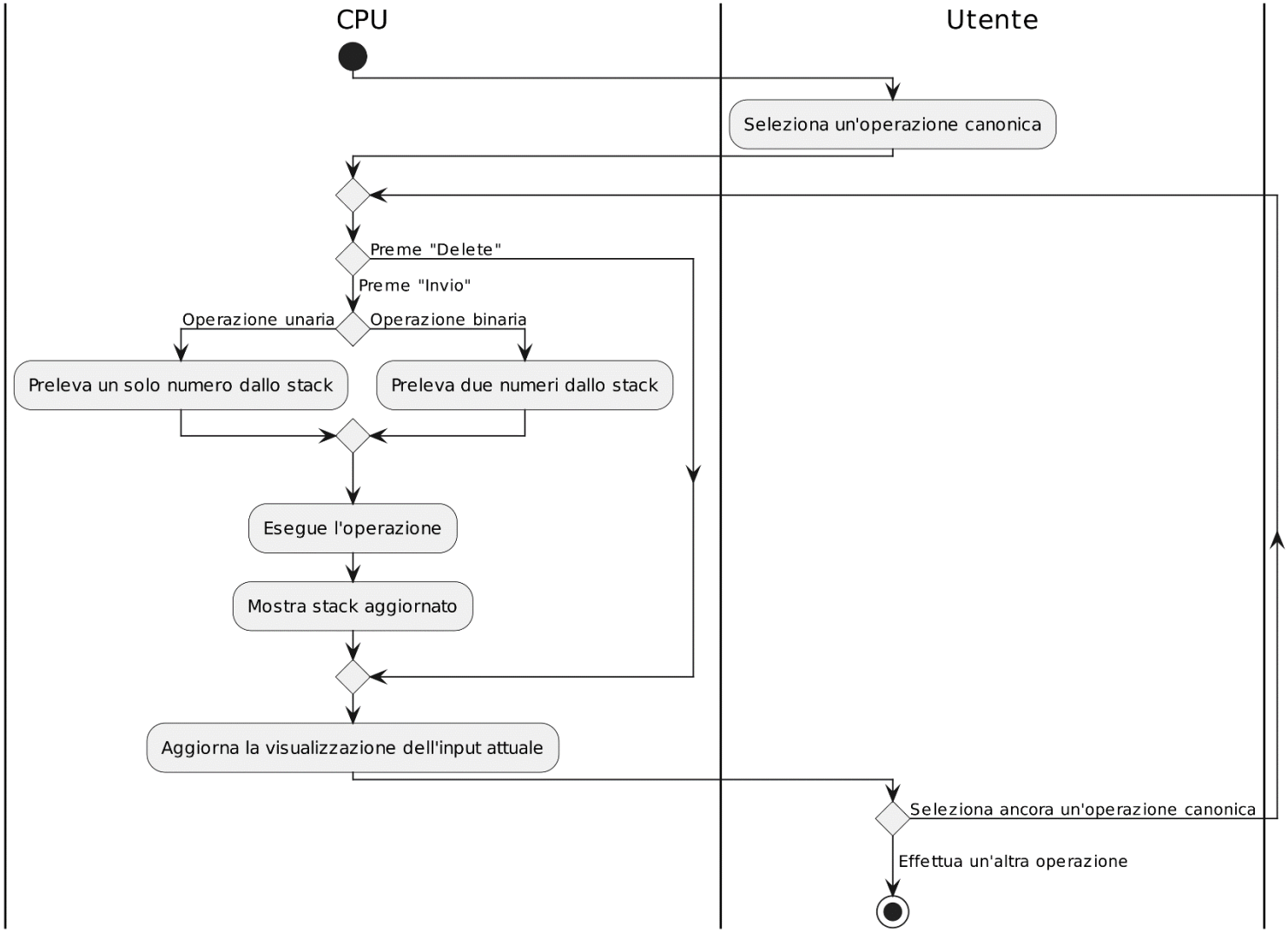
L'utente chiama avvia il flusso dell'unico altro partecipante al sequence, l'interface, mediante la pressione del tasto delete. L'evento viene catturato dal controller di interface che con la chiamata al metodo specifico ripulisce la textfield di input.

2.3. ACTIVITY DIAGRAM

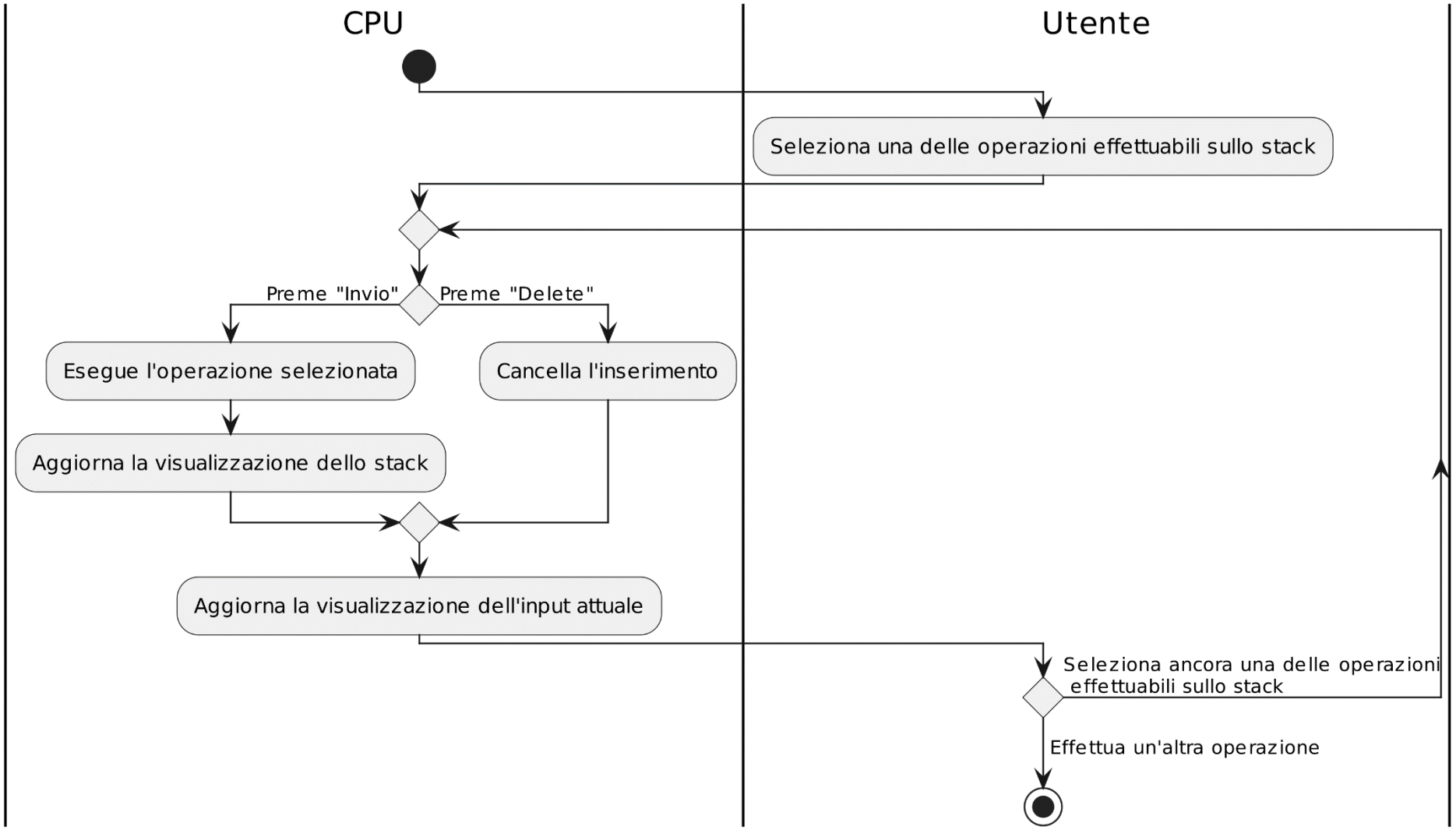
2.3.1. Inserimento numero



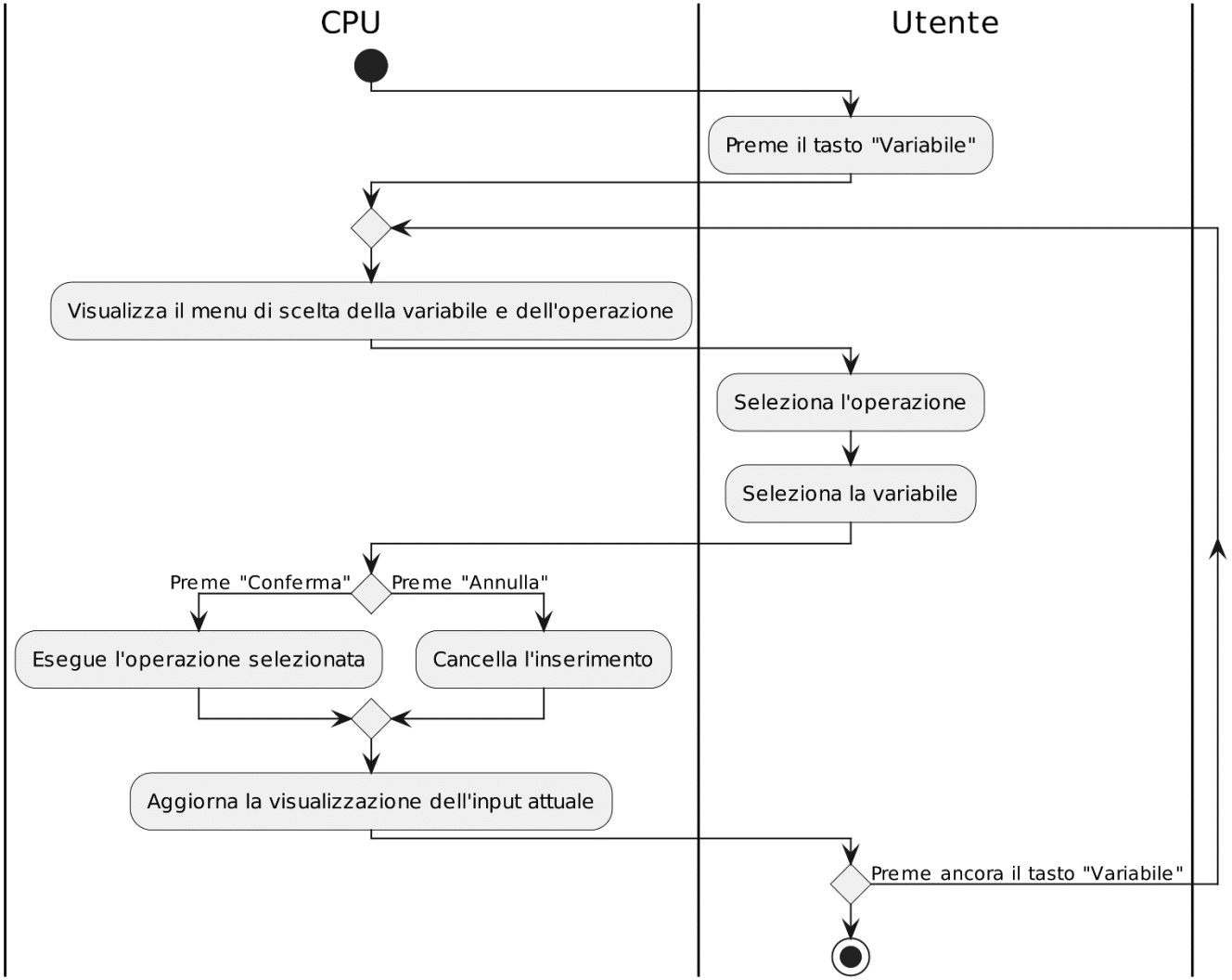
2.3.2. *Operazione canonica*



2.3.3. *Operazioni su stack*

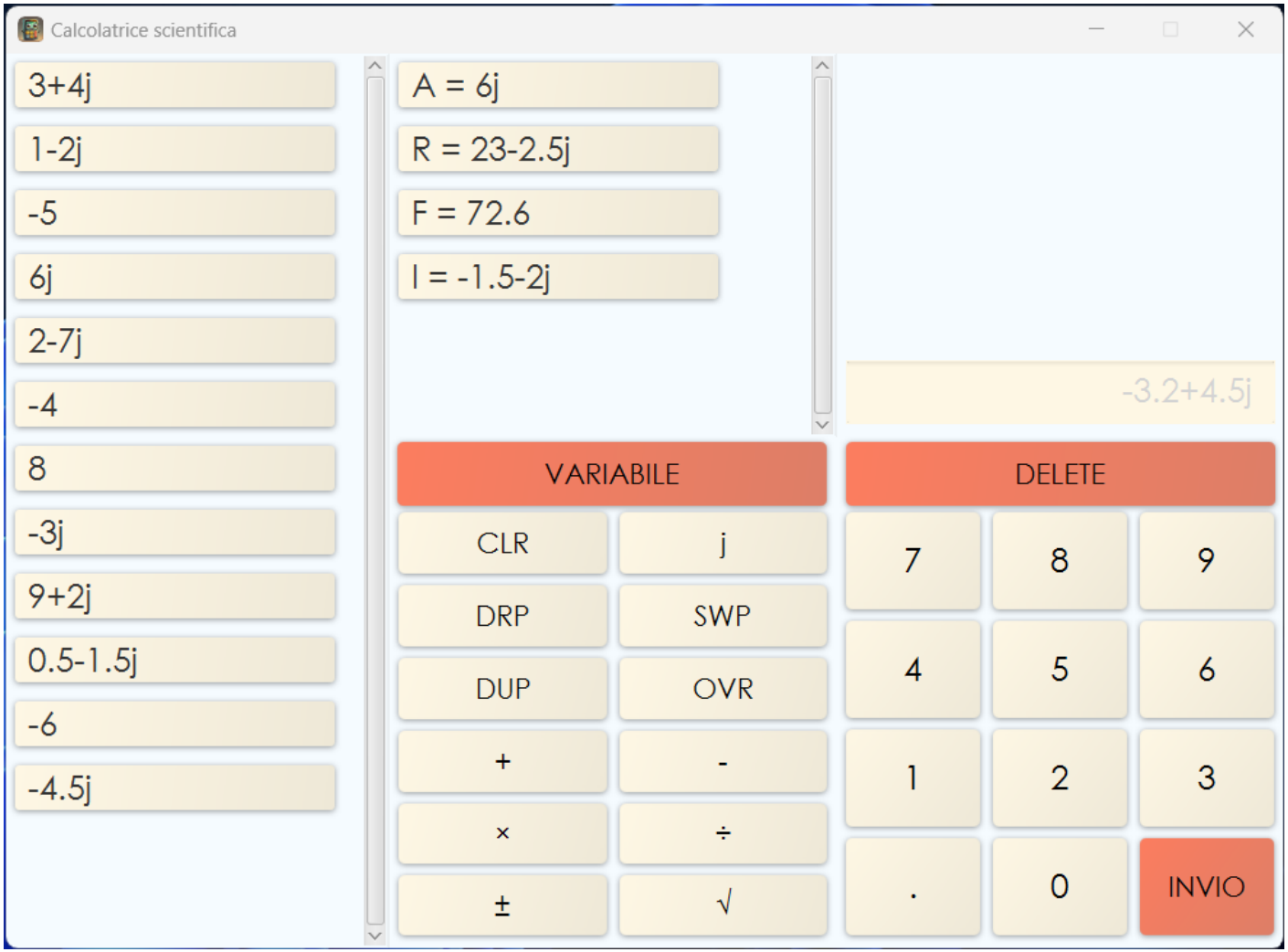


2.3.4. *Operazioni su variabili*

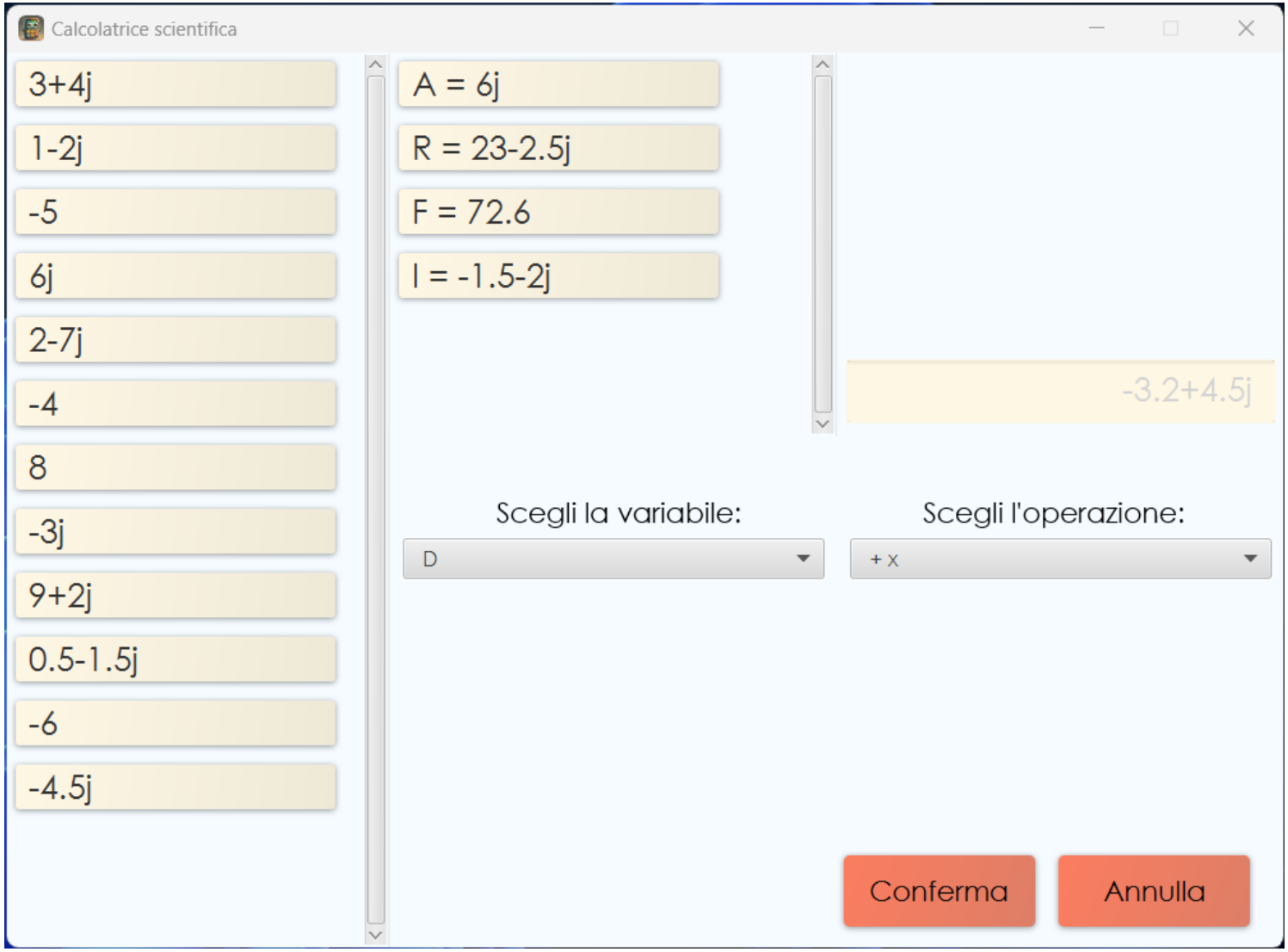


3. MOCK-UP DEFINITIVO

3.1. INTERFACCIA GRAFICA IN MODALITÀ STANDARD



3.2. INTERFACCIA GRAFICA PER EFFETTUARE OPERAZIONI SU VARIABILE



4. MATRICE DI TRACCIABILITÀ

ID Requisito	Design	Implementazione	Testing	Requisiti correlati
IF.1.1	Class BinaryCanonicOperations, SD_UC2.1			IF.1, IF.4, ER.1.1
IF.1.2	Class BinaryCanonicOperations, SD_UC2.1			IF.1, IF.4, ER.1.1
IF.1.3	Class BinaryCanonicOperations, SD_UC2.1			IF.1, IF.4, ER.1.1
IF.1.4	Class BinaryCanonicOperations, SD_UC2.1			IF.1, IF.4, ER.1.1
IF.1.5	Class UnaryCanonicOperations, SD_UC2.2			IF.1, IF.4, ER.1.2
IF.1.6	Class UnaryCanonicOperations, SD_UC2.2			IF.1, IF.4, ER.1.2
IF.2.1	Class Stack, SD_UC3.1			IF.2
IF.2.2	Class Stack, SD_UC3.2			IF.2, IF.4, ER.1.2
IF.2.3	Class Stack, SD_UC3.4			IF.2, IF.4, ER.1.2
IF.2.4	Class Stack, SD_UC3.3			IF.2, IF.4, ER.1.1
IF.2.5	Class Stack, SD_UC3.5			IF.2, IF.4, ER.1.1
IF.3.1	Class Vars, SD_UC4.1			IF.3, IF.4, ER.2.1
IF.3.2	Class Vars, SD_UC4.2			IF.3, ER.2.2
IF.3.3	Class Vars, SD_UC4.3			IF.3, IF.4, IF.1.1, ER.2.2
IF.3.4	Class Vars, SD_UC4.4			IF.3, IF.4, IF.1.1, ER.2.2
IF.4.1	Class Checker, SD_UC1			IF.4
IF.4.2	Class Checker, SD_UC1			IF.4
IF.4.3	Class Checker, SD_UC1			IF.4
IF.4.4	Class Checker, SD_UC1			IF.4

UI.1.1	Class Controller, Mock-up definitivo			UI.1, IF.4, IF.1, IF.2, IF.3
UI.1.2	Class Controller, Mock-up definitivo			UI.1, IF.3
UI.1.3	Class Controller, Mock-up definitivo			UI.1, IF.3, UI.1.6
UI.1.4	Class Controller, Mock-up definitivo			UI.1, UI.1.5, UI.1.6, IF.4
UI.1.5	Class Controller, Mock-up definitivo			UI.1, IF.4
UI.1.6	Class Controller, Mock-up definitivo			UI.1, IF.1, IF.2, IF.3
ER.1.1	Class InvalidOperandsException, SD_UC2.1, SD_UC3.3, SD_UC3.5			ER.1, IF.1.1, IF.1.2, IF.1.3, IF.1.4, IF.2.4, IF.2.5
ER.1.2	Class InvalidOperandsException, SD_UC2.2, SD_UC3.2, SD_UC3.4			ER.1, IF.1.5, IF.1.6, IF.2.2, IF.2.3
ER.2.1	Class Checker, Class Controller, , SD_UC4.1			ER.2, IF.3.1
ER.2.2	Class InvalidOperandsException, Class UninitializedException, SD_UC4.1, SD_UC4.2, SD_UC4.3, SD_UC4.4			ER.2
ER.2.2.1	Class UninitializedException, SD_UC4.1			ER.2.2, IF.3.2
ER.2.2.2	Class InvalidOperandsException, SD_UC4.2			ER.2.2, IF.3.1
ER.2.2.3	Class InvalidOperandsException, Class UninitializedException, SD_UC4.3, SD_UC4.4			ER.2.2, IF.3.3, IF.3.4
ER.3	Class Checker, Class, InvalidImputException, SD_UC1, SD_UC3.1-3.5			IF.4, IF.2