UNIVERSITÀ DEGLI STUDI DI SALERNO

Corso di Laurea in Ingegneria Informatica
Corso di Ingegneria del Software

Progetto di Ingegneria del software

Anno accademico 2023/2024

Studenti
Pierpaolo Paolino
Filippo Maria Sabatino
Federico Maria Raggio
Daniele Santoro

1.Test planning.....2
1.1 Test unit class CalculatorModel..... 2
1.2 Test unit class OperationComplexNumber..... 9
1.3 Test unit class Stack..... 18
1.4 Integration test CalculatorController-CalculatorModel..... 21
2.0 Matrice di tracciabilità per i test.....27
3.0Matrice di tracciabilità per l'implementazione..... 28

1.Test planning

1.1 Test unit class CalculatorModel

TestCaseID TN	Descrizione	Metodo testato	Input	Output atteso
TN01	Parte reale > 0, parte immaginaria > 0	isValidComplex Number	3+4j	True
TN02	Parte reale < 0, parte immaginaria < 0	isValidComplex Number	-2.5 - 1.7j	True
TN03	Parte reale = 0, parte immaginaria > 0	isValidComplex Number	5j	True
TN04	Parte reale > 0, parte immaginaria = 0	isValidComplex Number	7	True
TN05	Numero non valido	isValidComplex Number	invalid	False
TN06	Parte reale = 0, parte immaginaria > 0	isValidComplex Number	0 + 4j	True
TN07	Parte reale < 0, parte immaginaria = 0	isValidComplex Number	-3,6	True
TN08	Parte reale = 0, parte immaginaria < 0	isValidComplex Number	0 - 2j	True
TN09	Parte reale < 0, parte immaginaria = 0	isValidComplex Number	-1.2 + 0j	True
TN10	Parte reale < 0, parte immaginaria < 0	isValidComplex Number	-1.5 - 2.5j	True
TN11	Parte reale > 0, parte immaginaria < 0	isValidComplex Number	+5.5 - 1.2j	True

TN12	Parte reale < 0, parte immaginaria > 0	isValidComplex Number	-4.2 + 3.8j	True
TN13	Parte reale = 0, parte immaginaria = 0	isValidComplex Number	0+0j	True

TestCaseID TV	**Descrizione**	**Metodo testato**	**Input**	**Output atteso**
TV01	input <x con x qualsiasi variabile	isValidVariable	<a	True
TV02	input >x con x qualsiasi variabile	isValidVariable	>b	True
TV03	input +x con x qualsiasi variabile	isValidVariable	+c	True
TV04	input -x con x qualsiasi variabile	isValidVariable	-z	True
TV05	nessun input	isValidVariable	“”	False
TV06	input spazio “ “	isValidVariable	“ “	False
TV07	input <1 con 1 qualsiasi numero	isValidVariable	<1	False
TV08	input +	isValidVariable	+	False

TV09	input -	isValidVariable	-	False
TV10	input <ab con “a” e “b” variabili	isValidVariable	<ab	False
TV11	input <>	isValidVariable	<>	False
TV12	input ++	isValidVariable	++	False
TV13	input –	isValidVariable	–	False
TV14	input ÷	isValidVariable	÷	False
TV15	input ×	isValidVariable	×	False
TV16	input <+	isValidVariable	<+	False
TV17	input >-	isValidVariable	>-	False
TV18	input <÷	isValidVariable	<÷	False

TV19	input >× con × operatore per le moltiplicazioni	isValidVariable	>×	False
TV20	input <+-5 con 5 qualsiasi numero	isValidVariable	<+-5	False
TV21	input >+-1 con 1 qualsiasi numero	isValidVariable	>+-1	False
TV22	input <a1 con a qualsiasi variabile e 1 qualsiasi numero	isValidVariable	<a1	False
TV23	input >a1 con a qualsiasi variabile e 1 qualsiasi numero	isValidVariable	>a1	False

TestCaseID TO	**Descrizione**	**Metodo testato**	**Input**	**Output atteso**
TO01	Simbolo addizione	isValidOperator	+	True
TO02	Simbolo sottrazione	isValidOperator	-	True
TO03	Simbolo moltiplicazione	isValidOperator	×	True
TO04	Simbolo divisione	isValidOperator	÷	True
TO05	Simbolo radice quadrata	isValidOperator	√	True

TO06	Simbolo inversione del segno	isValidOperator	+/-	True
TO07	Simbolo errato inversione del segno	isValidOperator	-/+	False
TO08	Simbolo errato inversione del segno	isValidOperator	+//-	False
TO09	Simbolo errato radice quadrata	isValidOperator	sqrt	False
TO10	Simbolo errato divisione	isValidOperator	/	False
TO11	Simbolo errato moltiplicazione (lettera x)	isValidOperator	x	False
TO12	Simbolo errato moltiplicazione	isValidOperator	*	False

TestCaseID TE	**Descrizione**	**Metodo testato**	**Input**	**Output atteso**
TE01	Verifica dell'operazione di somma tra due numeri complessi	executeOperations	3+2j; 1+4.2j; +	4+6.2j
TE02	Verifica dell'operazione di sottrazione tra due numeri complessi	executeOperations	5+3j; 2+1j; -	3+2j

TE03	Verifica dell’operazione di moltiplicazione tra due numeri complessi;	executeOperations	3+2j; 1+4j; x	-5+14j
TE04	Verifica dell’operazione di divisione tra due numeri complessi	executeOperations	3+2j; 1+4j; ÷	0.647-0.588j
TE05	Verifica dell’operazione di divisione tra due numeri complessi, con divisore nullo	executeOperations	3+2j; 0+0j; ÷	throw new ErrorHandler
TE06	Verifica dell’operazione di radice quadrata di un numero complesso	executeOperations	4+2j	2.058+0.486j -2.058-0.486j
TE07	Verifica dell’operazione di inversione del segno di un numero complesso	executeOperations	1+1j	1-1j

TE08	Verifica di una sequenza di operazioni tra numeri complessi	executeOperations	3+2j; 1+1j; 4+0j; +; -; 2+2j; ÷	-0.25+0.75j

TestCaseID TH	**Descrizione**	**Metodo testato**	**Input**	**Output atteso**
TH01	input inviato “>x” il top dello stack è un numero valido	handleVariables	“>z”	alla variabile x viene assegnato il valore del top dello stack.
TH02	Input inviato “>x” e il top dello stack è un operatore	handleVariables	“>x”	throw ErrorHandler
TH03	input inviato “<x” ma lo stack è vuoto	handleVariables	“<x”	throw ErrorHandler
TH04	input inviato “>x” ma la variabile x non è stata ancora inizializzata	handleVariables	“>x”	throw ErrorHandler
TH05	storeVariable di x e 5+2j e input “<x”	handleVariables	“<x”	allo stack viene aggiunto l’elemento 5+2j come top
TH06	input “+x” ma lo stack è vuoto	handleVariables	“+x”	throw ErrorHandler
TH07	storeVariable di x e 5+2j e stack push di 5-j. input “+x”	handleVariables	“+x”	il top dello stack sarà 8+j
TH08	input “-x” ma lo stack è vuoto	handleVariables	“-x”	throw ErrorHandler

TH09	storeVariable di x e 7+4j e stack push di 2+j. input “-x”	handleVariables	“-x”	il top dello stack sarà 5+3j

1.2 Test unit class OperationComplexNumber

TestCaseID TA	**Descrizione**	**Metodo testato**	**Input**	**Output atteso**
TA01	verifica dell’operazione somma tra due numeri entrambi con : parte reale >0 parte immaginaria > 0	add	cn1 = 1+2j cn2 = 3+4j	4+6j
TA02	verifica dell’operazione somma tra due numeri primo numero : parte reale >0 parte immaginaria > 0 secondo numero: parte reale > 0 parte immaginaria < 0	add	cn1 = 1+2j cn2 = 3-4j	3-2j
TA03	verifica dell’operazione somma tra due numeri primo numero : parte reale >0 parte immaginaria > 0 secondo numero:	add	cn1 = 1+2j cn2 = -3+4j	-2+6j

	parte reale < 0 parte immaginaria > 0			
TA04	verifica dell'operazione somma tra due numeri primo numero : parte reale >0 parte immaginaria > 0 secondo numero: parte reale < 0 parte immaginaria < 0	add	cn1 = 1+2j cn2 = -3-4j	-2-2j
TA05	verifica dell'operazione somma tra due numeri primo numero : parte reale >0 parte immaginaria < 0 secondo numero: parte reale > 0 parte immaginaria > 0	add	cn1 = 1-2j cn2 = 3+4j	4+2j
TA06	verifica dell'operazione somma tra due numeri primo numero : parte reale >0 parte immaginaria < 0 secondo numero: parte reale > 0 parte immaginaria < 0	add	cn1 = 1-2j cn2 = 3-4j	4-6j

TA07	verifica dell'operazione somma tra due numeri primo numero : parte reale >0 parte immaginaria < 0 secondo numero: parte reale < 0 parte immaginaria > 0	add	cn1 = 1-2j cn2 = -3+4j	-2+2j
TA08	verifica dell'operazione somma tra due numeri primo numero : parte reale >0 parte immaginaria > 0 secondo numero: parte reale < 0 parte immaginaria < 0	add	cn1 = -1-2j cn2 = -3-4j	-4-6j
TA09	verifica dell'operazione somma tra due numeri primo numero : parte reale <0 parte immaginaria > 0 secondo numero: parte reale > 0 parte immaginaria > 0	add	cn1 = -1+2j cn2 = 3+4j	2+6j
TA10	verifica dell'operazione somma tra due numeri	add	cn1 = -1+2j cn2 = 3-4j	2-2j

	primo numero : parte reale <0 parte immaginaria > 0 secondo numero: parte reale > 0 parte immaginaria < 0			
TA11	verifica dell'operazione somma tra due numeri primo numero : parte reale <0 parte immaginaria > 0 secondo numero: parte reale < 0 parte immaginaria > 0	add	cn1 = -1+2j cn2 = -3+4j	-4+6j
TA12	verifica dell'operazione somma tra due numeri primo numero : parte reale <0 parte immaginaria > 0 secondo numero: parte reale < 0 parte immaginaria < 0	add	cn1 = -1+2j cn2 = -3-4j	-4-2j
TA13	verifica dell'operazione somma tra due numeri primo numero : parte reale < 0	add	cn1 = -1-2j cn2 = 3+4j	2+2j

	parte immaginaria < 0 secondo numero: parte reale > 0 parte immaginaria > 0			
TA14	verifica dell'operazione somma tra due numeri primo numero : parte reale <0 parte immaginaria < 0 secondo numero: parte reale > 0 parte immaginaria < 0	add	cn1 = -1-2j cn2 = 3-4j	2-6j
TA15	verifica dell'operazione somma tra due numeri primo numero : parte reale < 0 parte immaginaria < 0 secondo numero: parte reale < 0 parte immaginaria > 0	add	cn1 = -1-2j cn2 = -3+4j	-4-2j
TA16	verifica dell'operazione somma tra due numeri primo numero : parte reale < 0 parte immaginaria < 0 secondo numero: parte reale < 0	add	cn1 = -1-2j cn2 = -3-4j	-4-6j

	parte immaginaria < 0			
TA17	verifica dell'operazione somma tra due numeri entrambi con parte reale = 0 parte immaginaria = 0	add	cn1 = 0+0j cn2 = 0+0j	0+0j

TestCaseID TS	**Descrizione**	**Metodo testato**	**Input**	**Output atteso**
TS01	Verifica dell'operazione di sottrazione tra due numeri complessi	subtract	5+6j; 3+4j;	2+2j
TS02	Verifica dell'operazione di sottrazione tra zero e un numero complesso	subtract	0+0j; 3+4j;	-3-4j;

TestCaseID TM	Descrizione	Metodo testato	Input	Output atteso
TM01	Verifica dell'operazione di moltiplicazione tra due numeri complessi con parte reale > 0, parte immaginaria > 0	multiply	1+2j; 3+4j;	-5+10j
TM02	Verifica dell'operazione di moltiplicazione tra due numeri complessi con parte reale < 0, parte immaginaria < 0	multiply	-1-2j; -3-4j;	5-10j
TM03	Verifica dell'operazione di moltiplicazione di un numero complesso per zero	multiply	0+0j; 1+2j;	0+0j

TestCaseID TD	Descrizione	Metodo testato	Input	Output atteso
TD01	Divisione fra numero1: Re>0, Im>0 numero2: Re>0, Im>0	divide	1+2j 3+4j	0,44+0,08j
TD02	Divisione fra numero1: Re=0, Im>0 numero2: Re=0, Im>0	divide	+2j +4j	0,5

TD03	Divisione fra numero1: Re>0, Im<0 numero2: Re=0, Im>0	divide	10-3j 4j	-0,75-2,5j
TD04	Divisione fra numero1: Re<0, Im=0 numero2: Re=0, Im>0	divide	-7 4j	1,75j
TD05	Divisione fra numero1: Re>0, Im<0 numero2: Re=0, Im<0	divide	-0.75 -1.5j -4j	0,375-0,1875j

TestCaseID TR	**Descrizione**	**Metodo testato**	**Input**	**Output atteso**
TR01	radice di un numero con : parte reale > 0 parte immaginaria > 0	sqrt	1+j	radice “principale” 1.09868+0.45509j
TR02	radice di un numero con : parte reale > 0 parte immaginaria < 0	sqrt	1-j	radice “principale” 1.09868 - 0.45509j
TR03	radice di un numero con : parte reale < 0 parte immaginaria > 0	sqrt	-1+j	radice “principale” 0.45509j+1.09868
TR04	radice di un numero con : parte reale < 0	sqrt	-1-j	radice “principale” 0.45509-1.09868j

	parte immaginaria < 0			
TR05	radice di un numero con : parte reale = 0 parte immaginaria > 0	sqrt	2j	radice “principale” 1+j
TR06	radice di un numero con : parte reale = 0 parte immaginaria < 0	sqrt	-2j	radice “principale” 1-j
TR07	radice di un numero con : parte reale > 0 parte immaginaria = 0	sqrt	2	radice “principale” 1.41421
TR08	radice di un numero con : parte reale < 0 parte immaginaria = 0	sqrt	-2	radice “principale” 1.41421j

TestCaseID TI	**Descrizione**	**Metodo testato**	**Input**	**Output atteso**
TI01	Verifica del calcolo dell’opposto di un numero complesso	inverse	1+2j	-1-2j
TI02	Verifica del calcolo dell’opposto di un numero complesso con parte reale = 0, parte immaginaria = 0	inverse	0+0j	0+0j

1.3 Test unit class Stack

TestCaseID TSC	Descrizione	Metodo testato	Input	Output atteso
TSC01	si fornisce come input uno stack non vuoto, lo svuota	clear	Stack con 3 valori	Stack vuoto
TSC02	si fornisce come input uno stack non vuoto, lo svuota	clear	Stack con 1 valore	Stack vuoto
TSC03 (*non automatizzato*)	si fornisce come input uno stack vuoto	clear	-	il controller lancia un errore di tipo ErrorHandler

TestCaseID TSD	Descrizione	Metodo testato	Input	Output atteso
TSD01	si fornisce come input uno stack non vuoto, ne elimina il top	drop	2 (top) 1	1 (top)

TSD02	si fornisce come input uno stack non vuoto, ne elimina il top	drop	1 (top)	stack vuoto
TSD03 (*non automatizzato*)	si fornisce come input uno stack vuoto	drop	-	il controller lancia un errore di tipo ErrorHandler

TestCaseID TSP	**Descrizione**	**Metodo testato**	**Input**	**Output atteso**
TSP01	si fornisce come input uno stack non vuoto, ne duplica il top	dup	1 (top)	1 (top) 1
TSP02	si fornisce come input uno stack non vuoto, ne duplica il top	dup	3 (top) 2	3 (top) 3 2

TSP03 (*non automatizzato*)	si fornisce come input uno stack vuoto	dup		il controller lancia un errore di tipo ErrorHandler

TestCaseID TSW	**Descrizione**	**Metodo testato**	**Input**	**Output atteso**
TSW01	si fornisce come input uno stack non vuoto, ne scambia il primo con il secondo elemento e viceversa	swap	2 (top) 1	1 (top) 2
TSW02 (*non automatizzato*)	si fornisce come input uno stack con un solo elemento	swap	1 (top)	il controller lancia un errore di tipo ErrorHandler

TestCaseID TSO	**Descrizione**	**Metodo testato**	**Input**	**Output atteso**
TSO01	si fornisce come input uno stack non vuoto, ne inserisce una copia del penultimo elemento	over	2 (top) 1	1 (top) 2 1
TSO02 (*non automatizzato*)	si fornisce come input uno stack con un solo elemento	over	1 (top)	il controller lancia un errore di tipo ErrorHandler

1.4 Integration test CalculatorController-CalculatorModel

Per questi test, abbiamo adottato l'approccio white box poichè, in quanto metodi centrali responsabili dell'aggiunta di elementi allo stack, è essenziale assicurarsi che il codice gestisca correttamente una varietà di casi, inclusi input validi e invalidi, e che interagisca correttamente con altri componenti del sistema come il modello e le variabili.

TestCaseID THE	**Descrizione**	**Metodo testato**	**Scenario**
THE01	(non automatizzato)	HandelOperationButtons - ExecuteOperation	1.Nessun operatore valido nello stack
THE02	(non automatizzato)	HandelOperationButtons - ExecuteOperation	2.Esecuzione di un'operazione binaria
THE03	(non automatizzato) il top dello stack non è un operatore	HandelOperationButtons - ExecuteOperation	3.Esecuzione di un'operazione binaria

Output Atteso TestCaseID THE01:

HandleOperationButtons - Controllo dello Stack:

Prima di eseguire qualsiasi operazione, il metodo HandleOperationButtons verifica se lo stack non è vuoto.

Scenario Passato: Se lo stack è vuoto, il metodo termina senza eseguire ulteriori controlli o operazioni.

Scenario Fallito: Se lo stack non è vuoto, il controllo passa al passo successivo.

HandleOperationButtons - Controllo degli Operatori Validi:

Dopo aver confermato che lo stack non è vuoto, il metodo verifica se nello stack c'è almeno un operatore valido.

Scenario Passato: Se c'è almeno un operatore valido, il controllo passa al metodo ExecuteOperation.

Scenario Fallito: Se non ci sono operatori validi, il metodo genera un'eccezione di tipo ErrorHandler con il messaggio "No valid operation in the stack" e l'esecuzione termina.

Risultato Finale:

Se il flusso arriva fino a questo punto senza generare eccezioni, significa che c'è almeno un operatore valido nello stack e il controllo passa ad ExecuteOperations

Output Atteso TestCaseID THE02:

HandleOperationButtons - Controllo dello Stack:
Prima di eseguire qualsiasi operazione, il metodo HandleOperationButtons verifica se lo stack non è vuoto.
Scenario Fallito: Se lo stack è vuoto, il metodo termina senza eseguire ulteriori controlli o operazioni.
Scenario Passato: Se lo stack non è vuoto, il controllo passa al passo successivo.
HandleOperationButtons - Controllo degli Operatori Validi:
Dopo aver confermato che lo stack non è vuoto, il metodo verifica se nello stack c'è almeno un operatore valido.
Scenario Passato: Se c'è almeno un operatore valido, il controllo passa al metodo ExecuteOperation.
Scenario Fallito: Se non ci sono operatori validi, il metodo genera un'eccezione (ErrorHandler) con il messaggio "Nessuna operazione valida nello stack" e l'esecuzione termina.
ExecuteOperation - Analisi dell'Operatore Binario:
All'interno del metodo ExecuteOperation, viene analizzato il top dello stack.
Scenario Operatore Binario: Se il top dello stack è un operatore binario, il metodo verifica se ci sono almeno altri 2 elementi nello stack.
Scenario Passato: Se ci sono almeno 2 elementi, il metodo preleva le prossime due stringhe dallo stack. Se entrambe sono numeri complessi, esegue l'operazione binaria (somma nel nostro caso) tra di esse. Il risultato dell'operazione viene poi pushato nuovamente nello stack. L'esecuzione continua con eventuali ulteriori operazioni.

Scenario Fallito: Se non ci sono almeno 2 elementi nello stack, il metodo genera un'eccezione (ErrorHandler) con il messaggio "Operandi insufficienti nello stack" e il codice di errore 1, e l'esecuzione termina.

Risultato Finale: Se il flusso arriva fino a questo punto senza generare eccezioni, significa che le operazioni sono state eseguite correttamente e lo stack è stato aggiornato secondo le specifiche.

Output Atteso TestCaseID THE03:

HandleOperationButtons - Controllo dello Stack:

Prima di eseguire qualsiasi operazione, il metodo HandleOperationButtons verifica se lo stack non è vuoto.

Scenario Fallito: Se lo stack è vuoto, il metodo termina senza eseguire ulteriori controlli o operazioni.

Scenario Passato: Se lo stack non è vuoto, il controllo passa al passo successivo.

HandleOperationButtons - Controllo degli Operatori Validi:

Dopo aver confermato che lo stack non è vuoto, il metodo verifica se nello stack c'è almeno un operatore valido.

Scenario Passato: Se c'è almeno un operatore valido, il controllo passa al metodo ExecuteOperation.

Scenario Fallito: Se non ci sono operatori validi, il metodo genera un'eccezione (ErrorHandler) con il messaggio "Nessuna operazione valida nello stack" e l'esecuzione termina.

ExecuteOperation - Analisi dello stack:

Il metodo inizia ad analizzare il top dello stack e verifica il tipo di elemento.

Diversi Scenari

1. Se il tipo di elemento è un operatore allora lo scenario è lo stesso di THE02.

2B. Se il tipo di elemento non è un operatore, il metodo fa il pop dallo stack e inserisce l'elemento in uno stack di appoggio e continua l'esecuzione analizzando il successivo elemento, che sarà il top.

ExecuteOperation - Esecuzione delle operazioni:

Appena verrà trovato un operatore verrà eseguito lo stesso flusso descritto nello scenario THE02.

Se nello stack non ci sono più operatori e lo stack di appoggio non è vuoto,

Il metodo fa il pop dallo stack di appoggio e pusha la stringa nello stack originale.

Questa operazione viene effettuata affinché non si pusha un operatore.

Diversi scenari:

1. Si è pushato un operatore. Il controllo passa allo stack principale che esegue l'operazione inserita, pusha il risultato e poi inizia a ricerca un nuovo operatore ritornando al punto 2B descritto sopra.
2. lo stack di appoggio è vuoto oppure contiene elementi ma nessun operatore. Questo è il punto di uscita da questo metodo perchè non ci sono più operazioni da effettuare quindi il flusso termina.

TestCaseID TAM	**Descrizione**	**Metodo testato**	**Scenario**
TAM01	(non automatizzato)	addToStack - isValidVariable HandleVariable	1.Aggiunta di variabili al buffer delle variabili
TAM02	(non automatizzato)	addToStack - isValidComplexNumber	2.Aggiunta di numeri complessi allo stack
TAM03	(non automatizzato)	addToStack - isValidOperator	3.Aggiunta di operatori allo stack.

Output Atteso TestCaseID TAM01:

AddToStack - Gestione delle variabili:

Quando si inserisce una variabile, come “>x”, il metodo AddToStack invoca “isValidVariable” per validare l’input.

Scenario Fallito: Se lo stack è vuoto, o la variabile non valida, il metodo termina senza eseguire ulteriori controlli o operazioni.

Scenario Passato: Se lo stack non è vuoto e la variabile valida, il controllo passa al passo successivo attraverso la funzione “HandleVariables”. Si verifica se l’ulitmo elemento dello stack (TOP) è un numero complesso valido o meno.

HandleVariables:

Scenario Fallito: Se l’ultimo elemento dello stack (TOP) non è un numero complesso, il metodo genera un'eccezione (ErrorHandler) con il messaggio "Last element in the stack is not a complex number" e l'esecuzione termina.

Scenario Passato: Se l’ultimo elemento dello stack (TOP) è un numero complesso valido, gestisce i vari casi di assegnazione e utilizzo delle variabili tramite gli operatori “<”, “>”, “+”, “-”, eseguendo i vari casi, rispettivamente:

<: Salvataggio dell’elemento contenuto della variabile, sullo stack.

>: Inserimento dell’ultimo elemento dello stack, nel buffer delle variabili

+: Somma l’ultimo elemento dello stack, con quello nel buffer delle variabili, e lo salva nuovamente nel buffer delle variabili.

-: Somma l’ultimo elemento dello stack, con quello nel buffer delle variabili, e lo salva nuovamente nel buffer delle variabili.

Risultato Finale: La mappa delle variabili viene aggiornata con la nuova variabile o si solleva un’eccezione in caso di errore.

Output Atteso TestCaseID TAM02:

AddToStack - Gestione dei numeri complessi:

Quando si inserisce un numero complesso, come “3+7j”, il metodo AddToStack invoca “isValidComplexNumber” per validare l’input.

Scenario Fallito: Se il numero complesso non è valido, il metodo termina senza eseguire ulteriori operazioni, sollevando un’eccezione “Invalid complex number or operator”.

Scenario Passato: Se il numero complesso è valido, viene aggiunto allo stack.

Risultato Finale: Lo stack viene aggiornato con il nuovo numero complesso oppure si solleva un'eccezione in caso di numero complesso non valido.

Output Atteso TestCaseID TAM03:

AddToStack - Gestione degli Operatori:

Quando si inserisce un operatore, come “+”, il metodo AddToStack invoca “isValidOperator” per validare l’input.

Scenario Fallito: Se l'operatore non è valido, il metodo termina senza eseguire ulteriori operazioni, sollevando un'eccezione "Invalid complex number or operator".

Scenario Passato: Se l'operatore è valido, viene aggiunto allo stack.

Risultato Finale: Lo stack viene aggiornato con il nuovo operatore oppure si solleva un'eccezione in caso di operatore non valido.

2.0 Matrice di tracciabilità per i test

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

Classe	**Requisiti Corrispondenti**
Integration test: CalculatorController-CalculatorModel	RF9, RF15, RF45, RF 47 RF48, RF50
Test Unit: CalculatorModelTest	[RF10-RF14], RF18, RF46, RF47, RF49
Test unit: StackTest	RF02, RF03, RF16, RF22, RF28, RF29, RF34, RF40, RF49
Test unit: OperationComplexNumbesTest	RF4, RF26, RF32, RF34, RF38, RF40, RF43, RF44

3.0Matrice di tracciabilità per l'implementazione

La matrice illustra le relazioni fra i metodi delle classi e i requisiti specificati.

Classe	Metodo	Requisiti corrispondenti
CalculatorController	handleStackWindowButtonAction () expandWindow()	RF16
CalculatorController	handleOperationsButtons() ExecuteOperations()	RF17, RF50
CalculatorController	handleMemoryWindowButtonAct ion() contractWindow()	RF18
CalculatorController	clearTextfield()	RF19
CalculatorController	addToStack()	RF22
CalculatorController	handleStackButtons()	RF21 RF27, RF28, RF33, RF39, RF50
CalculatorController - CalculatorModel	addToStack() isValidOperator() isValidComplexNUmber	RF09
CalculatorController - CalculatorModel	addToStack() isValidVariable() handleVariables	[RF10-RF15]
CalculatorModel	ExecuteOperations()	RF45, RF46
CalculatorView	-	[RF05-RF08], [RF16-R44], RF47
OperationComplexN Umber	OperationComplexNumber(String)	RF04
OperationComplexN Umber	divide()	RF26
OperationComplexN Umber	multiply()	RF32

OperationComplexNUmber	inverse()	RF34
OperationComplexNUmber	substract()	RF38
OperationComplexNUmber	sqrt()	RF40
OperationComplexNUmber	add()	RF44
Stack	clear()	RF21
Stack	drop()	RF27
Stack	over()	RF28
Stack	dup()	RF33
Stack	swap()	RF39
BufferVariable	-	RF10, RF18, RF49
ErrorHandler	-	RF09, RF11, [RF13-RF15], RF50