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

Università degli Studi di Salerno

Test Planning

INTRODUZIONE

Il processo di testing, nel contesto dello sviluppo software, si propone di raggiungere due obiettivi cruciali: dimostrare la conformità del software ai requisiti stabiliti e individuare eventuali deviazioni dal comportamento desiderato. Pur riconoscendo che i test non possono eliminare completamente i difetti, essi costituiscono un pilastro essenziale per garantire che il software rispetti le specifiche previste.

Consapevole dell'importanza rivestita da questa fase dello sviluppo software il team è stato attento a non ridurre questa fase ad una formalità ma a trasformala in un momento profondo di riflessione, arricchito da accesi dibattiti che hanno messo più volte in discussione la fase di implementazione.

Nonostante il fatto che il processo si sia rilevato ben poco lineare, il team si è sforzato di mantenere sempre un approccio sistematico rivolto al modello di processo a cascata assunto nella fase embrionale del progetto. Questo significa che ogni ripensamento sulla fase di implementazione è stato prima confermato da un attento studio delle fasi precedenti.

BREVE RIEPILOGO DEL SISTEMA

Il sistema implementato è una calcolatrice scientifica, che può operare su numeri complessi ed esegue le operazioni seguendo la notazione polacca inversa. Inoltre, deve poter gestire 26 variabili, identificate dalle 26 lettere dell'alfabeto, potendole manipolare con operazioni ad-hoc.

FUNZIONALITA' DA TESTARE

Il focus sul test si è concentrato sul verificare:

1- La possibilità di poter avere all'interno del sistema dei numeri complessi
2- Il corretto funzionamento delle operazioni sui numeri complessi (quali somma, sottrazione, moltiplicazione, divisione, radice quadrata e cambio segno).
3- La corretta gestione dell'esecuzione delle operazioni in base alla notazione polacca inversa
4- La corretta gestione delle variabili
5- La visualizzazione di schermate di errori in caso di funzionamenti non previste dal sistema.
6- Il corretto funzionamento dell'interfaccia grafica.

APPROCCIO UTILIZZATO

Per poter avere un test quanto più affidabile possibile, il team ha deciso di procedere in maniera incrementale con la stipulazione e l'esecuzione dei test. In particolare, è stato data priorità alle classi con il minor numero di dipendenze. Dunque, una volta testata la singola classe, in caso di presenza di relazioni di qualche tipo con altre classi, è stato testato il funzionamento congiunto.

DEFINIZIONE DEI CASI DI TEST E DELLE CLASSI DI EQUIVALENZA

Poiché i principali dati su cui deve operare il sistema sono dei numeri complessi, sono state definite delle classi di test esemplificative dei casi di maggiore interesse. Si fa presente che gli input usati nei test non sono solo quelli specificati nelle tabelle, ma anche numeri che possono ricondursi ad una delle categorie della tabella.

Classi	Valori
Numeri complessi aventi parte reale e parte immaginaria con segno concorde	**31.6238+22.729j, 23+59j, 12.528+74j, 9+13.729j, -31.6238-22j, -23-59j, -12.528-74j, -9-13.729j.**
Numeri complessi aventi parte reale e parte immaginaria con segno discorde	**31.6238-22.729j, 23-59j, 12.528-74j, 9-13.729j, -31.6238+22j, -23+59j, -12.528+74j, -9+13.729j.**
Numeri reali	**176, -176, 23.618, -23.618.**
Numeri immaginari	**176j, -176j, 23.618j, -23.618j.**
Casi limite	**0, 0j, -0j, 0.0j, -0.0j,**

	1j, **-1j,** **0.0+0.0j,** **11111111111111111,** **2.1584926487859.**
Input errati	**31.6238+,** **2j-1,** **j163,** **7ab4r9,** **j,** **-j**

FUNCTIONAL TEST CASES

FTC-1	**Inserimento di un numero**
Condizioni di ingresso	La calcolatrice è stata avviata.
Flusso di eventi	1 – Viene digitato "7ab4r9 e viene premuto invio (errore). 2 – Viene digitato "23-59j" e viene premuto invio. 3 – Il numero viene inserito nel top dello stack.
Condizioni di uscita	Viene aggiornata la visualizzazione dello stack.
Input	-Il test può essere effettuato su tutti i dati definiti dalle classi di equivalenza.
Use-Cases	Inserisce un numero.

FTC-2	**Operazioni canoniche binarie della calcolatrice**
Condizioni di ingresso	-L'utente ha avviato il software della calcolatrice. -Lo stack è vuoto. -I numeri inseriti risultano validi dal FTC-1.
Flusso di eventi	1 – Viene digitata l'operazione "+" e viene digitato Invio (condizione di errore). 2 – Viene inserito il numero "31.6238-22.729j" e viene digitato invio. 3 - Viene digitata l'operazione "+" e viene digitato Invio (condizione di errore). 4 – Viene inserito il numero "12.528+74j" e viene digitato Invio. 5 - Viene digitata l'operazione "+" e viene digitato Invio. 6 – Il software calcola il risultato ed inserisce il valore corretto nello stack. Ripetere i passi 1-6 per tutte le rimanenti operazioni binarie, in una nuova esecuzione del test.
Condizioni di uscita	-L'utente avrà visualizzato correttamente tutti i risultati nello stack mostrato dall'interfaccia.
Input	I valori inseriti nello stack saranno quelli definiti nelle classi di equivalenza, evitando i formati errati in quanto già testati nella funzionalità di inserimento.
Use-Cases	Inserisce un numero – Operazione aritmetica binaria

FTC-3	Operazioni canoniche unarie della calcolatrice
Condizioni di ingresso	-L'utente ha avviato il software della calcolatrice. -Lo stack è vuoto. -I numeri inseriti risultano validi dal FTC-1.
Flusso di eventi	1 – Viene digitata l'operazione ""√"" e viene digitato Invio (condizione di errore). 2 – Viene inserito il numero "-31.6238-22j" e viene digitato Invio. 3 – Viene selezionata l'operazione unaria "√" e viene digitato Invio. 4 – Il software calcola il risultato ed inserisce il valore corretto nello stack. Ripetere i passi 1-4 per l'altra operazione unaria (±), in una nuova esecuzione del test.
Condizioni di uscita	-L'utente avrà visualizzato correttamente tutti i risultati nello stack mostrato dall'interfaccia.
Input	I valori inseriti nello stack saranno quelli definiti nelle classi di equivalenza, evitando i formati errati in quanto già testati nella funzionalità di inserimento.
Use-Cases	Inserisce un numero – Operazione aritmetica unaria.

FTC-4	Operazioni sullo stack
Condizioni di ingresso	-L'utente ha avviato il software della calcolatrice. -Lo stack è vuoto. -I numeri che verranno inseriti devono risultare validi (FTC-1).
Flusso di eventi	1 – Viene selezionata l'operazione "Drop" (Caso di errore). 2 – Viene selezionata l'operazione "Duplicate" (Caso di errore). 3 – Viene selezionata l'operazione "Swap" (Caso di errore). 4 – Viene selezionata l'operazione "Over" (Caso di errore). 5 – Viene inserito un qualsiasi numero. 6 – ripete passo 2. 7 – Il numero viene correttamente duplicato ed inserito nello stack. 8 – Ripete passo 1. 9 – Il numero in testa viene correttamente eliminato dallo stack. 10 – Ripete passo 3 (Caso di errore). 11 – Ripete passo 4 (Caso di errore). 12 – Ripete passo 5. 13 - Ripete passo 3. 14 – I due numeri presenti nello stack vengono correttamente invertiti.

	15 - Ripete passo 4. 16 – Il secondo elemento presente nello stack viene correttamente duplicato e messo in testa. 17 - Viene selezionata l'operazione "Clear". 18 – Lo stack viene correttamente svuotato.
Condizioni di uscita	L'utente avrà visualizzato correttamente tutti i risultati nello stack mostrato dall'interfaccia.
Input	I valori inseriti nello stack saranno quelli definiti nelle classi di equivalenza, evitando i formati errati in quanto già testati nella funzionalità di inserimento.
Use-Cases	Inserisci un numero – Esegue un'operazione sullo stack (UC 3.1-3.5)

FTC-5	**Operazioni su variabili**
Condizioni di ingresso	-L'utente ha avviato il software della calcolatrice. -Lo stack è vuoto. -I numeri che verranno inseriti devono risultare validi (FTC-1). -L'utente ha selezionato il tasto "Variabile".
Flusso di eventi	1 – Viene selezionata una qualsiasi variabile. 2 – Viene selezionata l'operazione ">x" e il tasto "Conferma" (Caso di errore). 3 - Viene selezionata l'operazione "<x" e il tasto "Conferma" (Caso di errore). 4 – Viene selezionata l'operazione "+x" e il tasto "Conferma" (Caso di errore). 5 - Viene selezionata l'operazione "-x" e il tasto "Conferma" (Caso di errore). 6 – Viene selezionato il tasto "Annulla". 7 – Viene inserito un qualsiasi numero nello stack. 8 – Viene selezionato il tasto "Variabile". 9 – Ripete passo 1. 10 – Ripete passo 2. 11 – Viene inserito correttamente il valore presente nello stack all'interno della variabile selezionata. 12 – Ripete passi 6-8. 13 – Viene selezionata la stessa variabile del passo 9. 14 – Ripete passo 4. 15 – Viene correttamente sommato il valore presente nello stack al valore presente nella variabile ed il risultato conservato nella stessa. 16 – Ripete passi 6-8. 17 – Ripete passo 13. 18 – Ripete passo 5.

	19 – Viene correttamente sottratto il valore presente nello stack al valore presente nella variabile ed il risultato conservato nella stessa. 20 – Ripete passo 3. 21 – Viene inserito correttamente il valore presente nella variabile all'interno dello stack.
Condizioni di uscita	L'utente avrà visualizzato correttamente tutti i risultati nello stack mostrato dall'interfaccia e lo storico delle variabili modificate.
Use-Cases	Inserisce un numero - Esegue un'operazione su una variabile (UC 4.1-4.4).

FTC-6	**Delete**
Condizioni di ingresso	-L'utente ha avviato il software della calcolatrice.
Flusso di eventi	1 – L'utente inserisce un qualsiasi input senza premere "Invio". 2 – L'utente preme "Delete". 3 – Il textfield viene correttamente ripulito.
Condizioni di uscita	L'utente vede correttamente la digitazione nel textfield e premuto il tasto "Delete" ne verifica lo svuotamento.
Input	Qualsiasi carattere alfanumerico riconducibile all'inserimento nel textfield.
Use-Cases	Delete

UNIT TESTS

1- Classe ComplexNumber

UTC-1.1.1	**Test ComplexNumber.toString() : Numero complesso con parte reale e parte immaginaria decimali positive.**
Test Items	Classe **ComplexNumber**, metodo **toString()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("31.6238"), new BigDecimal("22.729"));
Oracle	num.toString().equals("31.624 + 22.729j") == true

UTC-1.1.2	**Test ComplexNumber.toString() : Numero complesso con parte reale e parte immaginaria intere positive.**
Test Items	Classe **ComplexNumber**, metodo **toString()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("23"), new BigDecimal("59"));
Oracle	num.toString().equals("23 + 59j") == true

UTC-1.1.3	**Test ComplexNumber.toString() : Numero complesso con parte reale decimale positiva e parte immaginaria intera positiva.**
Test Items	Classe **ComplexNumber**, metodo **toString()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("12.528"), new BigDecimal("74"));
Oracle	num.toString().equals("12.528 + 74j") == true

UTC-1.1.4	**Test ComplexNumber.toString() : Numero complesso con parte reale intera e parte complessa decimale entrambe positive.**
Test Items	Classe **ComplexNumber**, metodo **toString()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("-9"), new BigDecimal("-13.729"));
Oracle	num.toString().equals("-9 - 13.729j") == true

UTC-1.1.5	**Test ComplexNumber.toString() : Numero reale intero positivo.**
Test Items	Classe **ComplexNumber**, metodo **toString()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("+176"), new BigDecimal("0"));
Oracle	num.toString().equals("176") == true

UTC-1.1.6	**Test ComplexNumber.toString() : Numero nullo.**
Test Items	Classe **ComplexNumber**, metodo **toString()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("0"), new BigDecimal("0"));
Oracle	num.toString().equals("0") == true

UTC-1.1.7	**Test ComplexNumber.toString() : Numero immaginario puro decimale con tante cifre dopo la virgola.**
Test Items	Classe **ComplexNumber**, metodo **toString()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("0"), new BigDecimal("2.1584926487859"));
Oracle	num.toString().equals("2.158j") == true

UTC-1.1.8	**Test ComplexNumber.toString() : Numero reale intero molto grande.**
Test Items	Classe **ComplexNumber**, metodo **toString()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("1111111111111"), new BigDecimal("0"));
Oracle	num.toString().equals("1111111111111") == true

UTC-1.1.9	**Test ComplexNumber.toString() : 1j**
Test Items	Classe **ComplexNumber**, metodo **toString()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("0"), new BigDecimal("1"));
Oracle	num.toString().equals("j") == true

UTC-1.1.10	**Test ComplexNumber.toString() : -1j**
Test Items	Classe **ComplexNumber**, metodo **toString()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("0"), new BigDecimal("-1"));
Oracle	num.toString().equals("-j") == true

UTC-1.2.1	**Test ComplexNumber.getRealPart() : Numero complesso con parte reale positiva e parte immaginaria negativa decimali.**
Test Items	Classe **ComplexNumber**, metodo **getRealPart()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("31.624"), new BigDecimal("22.729"));
Oracle	num.getRealPart().equals("31.624") == true

UTC-1.2.2	**Test ComplexNumber.getRealPart() : Numero complesso con parte reale e parte immaginaria intere positive.**
Test Items	Classe **ComplexNumber**, metodo **getRealPart()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("23"), new BigDecimal("59"));
Oracle	num.getRealPart().equals("23.000") == true

UTC-1.2.3	**Test ComplexNumber.getRealPart() : Numero complesso con parte reale decimale negativa e parte immaginaria intera positiva.**
Test Items	Classe **ComplexNumber**, metodo **getRealPart()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("-12.528"), new BigDecimal("74"));
Oracle	num.getRealPart().equals("-12.528") == true

UTC-1.2.4	**Test ComplexNumber.getRealPart() : Numero reale intero negativo.**
Test Items	Classe **ComplexNumber**, metodo **getRealPart()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("-176"), new BigDecimal("0"));
Oracle	num.getRealPart().equals("-176.000") == true

UTC-1.2.5	**Test ComplexNumber.getRealPart() : Numero nullo.**
Test Items	Classe **ComplexNumber**, metodo **getRealPart()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("0"), new BigDecimal("0"));
Oracle	num.getRealPart().equals("0.000") == true

UTC-1.2.6	**Test ComplexNumber.getRealPart() : Numero immaginario puro decimale positivo.**
Test Items	Classe **ComplexNumber**, metodo **getRealPart()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("0"), new BigDecimal("23.618"));
Oracle	num.getRealPart().equals("0.000") == true

UTC-1.3.1	**Test ComplexNumber.getImaginaryPart() : Numero complesso con parte reale e parte immaginaria decimali positive.**
Test Items	Classe **ComplexNumber**, metodo **getImaginaryPart()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("31.624"), new BigDecimal("22.729"));
Oracle	num.getImaginarylPart().equals("22.729") == true

UTC-1.3.2	**Test ComplexNumber.getImaginaryPart() : Numero complesso con parte reale e parte immaginaria intere positive.**
Test Items	Classe **ComplexNumber**, metodo **getImaginaryPart()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("23"), new BigDecimal("59"));
Oracle	num.getImaginarylPart().equals("59.000") == true

UTC-1.3.3	**Test ComplexNumber.getImaginaryPart() : Numero complesso con parte reale intera positiva e parte immaginaria decimale negativa.**
Test Items	Classe **ComplexNumber**, metodo **getImaginaryPart()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("74"), new BigDecimal("-12.528"));
Oracle	num.getImaginarylPart().equals("-12.528") == true

UTC-1.3.4	**Test ComplexNumber.getImaginaryPart() : Numero immaginario puro intero negativo.**
Test Items	Classe **ComplexNumber**, metodo **getImaginaryPart()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("0"), new BigDecimal("-176"));
Oracle	num.getImaginarylPart().equals("-176.000") == true

UTC-1.3.5	**Test ComplexNumber.getImaginaryPart() : Numero nullo.**
Test Items	Classe **ComplexNumber**, metodo **getImaginaryPart()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("0"), new BigDecimal("0"));
Oracle	num.getImaginarylPart().equals("0.000") == true

UTC-1.3.6	**Test ComplexNumber.getImaginaryPart() : Numero reale decimale positivo.**
Test Items	Classe **ComplexNumber**, metodo **getImaginaryPart()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("23.618"), new BigDecimal("0"));
Oracle	num.getImaginarylPart().equals("0.000") == true

UTC-1.4.1	**Test ComplexNumber.getModule() : Numero complesso con parte reale e parte immaginaria decimali positive.**
Test Items	Classe **ComplexNumber**, metodo **getModule()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("31.624"), new BigDecimal("22.729"));
Oracle	num.getModule().equals("38.945") == true

UTC-1.4.2	**Test ComplexNumber.getModule() : Numero complesso con parte reale e parte immaginaria intere positive.**
Test Items	Classe **ComplexNumber**, metodo **getModule()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("23"), new BigDecimal("59"));
Oracle	Num.getModule().equals("63.325") == true

UTC-1.4.3	**Test ComplexNumber.getModule() : Numero con parte reale intera positiva e parte immaginaria decimale negativa.**
Test Items	Classe **ComplexNumber**, metodo **getModule()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("74"), new BigDecimal("-12.528"));
Oracle	num.getModule().equals("75.053") == true

UTC-1.4.4	**Test ComplexNumber.getModule() : Numero immaginario puro intero negativo.**
Test Items	Classe **ComplexNumber**, metodo **getModule()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("0"), new BigDecimal("-176"));
Oracle	num.getModule().equals("176.000") == true

UTC-1.4.5	**Test ComplexNumber.getModule() : Numero immaginario puro decimale negativo.**
Test Items	Classe **ComplexNumber**, metodo **getModule()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("0"), new BigDecimal("-23.618"));
Oracle	num.getModule().equals("23.618") == true

UTC-1.4.6	**Test ComplexNumber.getModule() : Numero nullo.**
Test Items	Classe **ComplexNumber**, metodo **getModule()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("0"), new BigDecimal("0"));
Oracle	num.getModule().equals("0.000") == true

UTC-1.4.7	**Test ComplexNumber.getModule() : Numero reale decimale positivo.**
Test Items	Classe **ComplexNumber**, metodo **getModule()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("23.618"), new BigDecimal("0"));
Oracle	num.getModule().equals("23.618") == true

UTC-1.4.8	**Test ComplexNumber.getModule() : Numero con parte reale intera positiva molto grande e parte immaginaria negativa decimale con molti numeri dopo la virgola.**
Test Items	Classe **ComplexNumber**, metodo **getModule()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("11111111111111111"), new BigDecimal("2.1584926487859"));
Oracle	num.getModule().equals("11111111111111111.000") == true

UTC-1.5.1	**Test ComplexNumber.getPhase() : Numero con parte reale e parte immaginaria decimali positive.**
Test Items	Classe **ComplexNumber**, metodo **getPhase()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("31.624"), new BigDecimal("22.729"));
Oracle	num.getPhase().equals("0.623") == true

UTC-1.5.2	**Test ComplexNumber.getPhase() : Numero complesso con parte reale e parte immaginaria intere positive.**
Test Items	Classe **ComplexNumber**, metodo **getPhase()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("23"), new BigDecimal("59"));
Oracle	num.getPhase().equals("1.199") == true

UTC-1.5.3	**Test ComplexNumber.getPhase() : Numero complesso con parte reale intera positiva e parte immaginaria decimale negativa.**
Test Items	Classe **ComplexNumber**, metodo **getPhase()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("74"), new BigDecimal("-12.528"));
Oracle	num.getPhase().equals("-0.168") == true

UTC-1.5.4	**Test ComplexNumber.getPhase() : Numero immaginario puro intero negativo.**
Test Items	Classe **ComplexNumber**, metodo **getPhase()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("0"), new BigDecimal("-176"));
Oracle	num.getPhase().equals("-1.571") == true

UTC-1.5.5	**Test ComplexNumber.getPhase() : Numero immaginario puro decimale negativo.**
Test Items	Classe **ComplexNumber**, metodo **getPhase()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("0"), new BigDecimal("-23.618"));
Oracle	num.getPhase().equals("-1.571") == true

UTC-1.5.6	**Test ComplexNumber.getPhase() : Numero nullo.**
Test Items	Classe **ComplexNumber**, metodo **getPhase()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("0"), new BigDecimal("0"));
Oracle	num.getPhase().equals("0.000") == true

UTC-1.5.7	**Test ComplexNumber.getPhase() : Numero reale decimale positive.**
Test Items	Classe **ComplexNumber**, metodo **getPhase()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("23.618"), new BigDecimal("0"));
Oracle	num.getPhase().equals("0.000") == true

UTC-1.5.8	**Test ComplexNumber.getPhase() : Numero complesso con parte reale intera positiva molto grande e parte immaginaria decimale positiva con tanti numeri dopo la virgola.**
Test Items	Classe **ComplexNumber**, metodo **getPhase()**
Input	ComplexNumber num = new ComplexNumber(new BigDecimal("11111111111111111"), new BigDecimal("2.1584926487859"));
Oracle	num.getPhase().equals("0.000") == true

2- Classe BinaryCanonicOperations

UTC-2.1.1	**Test BinaryCanonicOperations.sum(ComplexNumber n1, ComplexNumber n2) : verifica correttezza somma tra:** **Primo numero: complesso con parte reale e immaginaria positiva e decimale** **Secondo numero: complesso con parte reale e immaginaria positiva e intera**
Test Items	Classe **BinaryCanonicOperations**, metodo **sum(ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("23"),new BigDecimal("59")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("54.624"),new BigDecimal("81.729")); ComplexNumber result = BinaryCanonicOperations.sum(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.1.2	**Test BinaryCanonicOperations.sum(ComplexNumber n1, ComplexNumber n2) : verifica correttezza somma tra:** **Primo numero: complesso con parte reale e immaginaria positiva e decimale** **Secondo numero: complesso con parte reale intera negativa e parte immaginaria decimale negativa**
Test Items	Classe **BinaryCanonicOperations**, metodo **sum (ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("-9"),new BigDecimal("-13.729")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("22.624"),new BigDecimal("9")); ComplexNumber result = BinaryCanonicOperations.sum(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.1.3	**Test BinaryCanonicOperations.sum(ComplexNumber n1, ComplexNumber n2) : verifica correttezza somma tra: Primo numero: complesso con parte reale e immaginaria positiva e decimale Secondo numero: coniugato del primo numero**
Test Items	Classe **BinaryCanonicOperations**, metodo **sum (ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("-22.729")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("63.248"),new BigDecimal("0")); ComplexNumber result = BinaryCanonicOperations.sum(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.1.4	**Test BinaryCanonicOperations.sum(ComplexNumber n1, ComplexNumber n2) : verifica correttezza somma tra: Primo numero: complesso con parte reale e immaginaria positiva e decimale Secondo numero: numero reale intero positivo**
Test Items	Classe **BinaryCanonicOperations**, metodo **sum (ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("176"),new BigDecimal("0")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("207.624"),new BigDecimal("22.729")); ComplexNumber result = BinaryCanonicOperations.sum(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.1.5	**Test BinaryCanonicOperations.sum(ComplexNumber n1, ComplexNumber n2) : verifica correttezza somma tra: Primo numero: complesso con parte reale e immaginaria positiva e decimale Secondo numero: numero immaginario decimale negativo**
Test Items	Classe **BinaryCanonicOperations**, metodo **sum (ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("0"),new BigDecimal("-23.618")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("-0.889")); ComplexNumber result = BinaryCanonicOperations.sum(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.1.6	**Test BinaryCanonicOperations.sum(ComplexNumber n1, ComplexNumber n2) : verifica correttezza somma tra: Primo numero: complesso con parte reale e immaginaria positiva e decimale Secondo numero: 0**
Test Items	Classe **BinaryCanonicOperations**, metodo **sum (ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("0"),new BigDecimal("0")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber result = BinaryCanonicOperations.sum(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.1.7	**Test BinaryCanonicOperations.sum(ComplexNumber n1, ComplexNumber n2) : verifica correttezza somma tra:** **Primo numero: complesso con parte reale e immaginaria positiva e decimale** **Secondo numero: numero reale intero positivo molto grande**
Test Items	Classe **BinaryCanonicOperations**, metodo **sum (ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("11111111111111111"),new BigDecimal("0")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("111111111111111142.624"),new BigDecimal("22.729")); ComplexNumber result = BinaryCanonicOperations.sum(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.1.8	**Test BinaryCanonicOperations.sum(ComplexNumber n1, ComplexNumber n2) : verifica correttezza somma tra:** **Primo numero: numero complesso con parte reale e immaginaria intera positiva** **Secondo numero: coniugato del primo numero**
Test Items	Classe **BinaryCanonicOperations**, metodo **sum (ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("23"),new BigDecimal("59")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("23"),new BigDecimal("-59")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("46"),new BigDecimal("0")); ComplexNumber result = BinaryCanonicOperations.sum(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.1.9	**Test BinaryCanonicOperations.sum(ComplexNumber n1, ComplexNumber n2) : verifica correttezza somma tra:** **Primo numero: numero reale intero positivo** **Secondo numero: numero immaginario intero positivo**
Test Items	Classe **BinaryCanonicOperations**, metodo **sum (ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("176"),new BigDecimal("0")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("0"),new BigDecimal("176")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("176"),new BigDecimal("176")); ComplexNumber result = BinaryCanonicOperations.sum(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.1.10	**Test BinaryCanonicOperations.sum(ComplexNumber n1, ComplexNumber n2) : verifica correttezza somma tra:** **Primo numero: 0** **Secondo numero: 0**
Test Items	Classe **BinaryCanonicOperations**, metodo **sum (ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("0"),new BigDecimal("0")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("0"),new BigDecimal("0")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("0"),new BigDecimal("0")); ComplexNumber result = BinaryCanonicOperations.sum(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.2.1	**Test BinaryCanonicOperations.sub (ComplexNumber n1, ComplexNumber n2) : verifica correttezza differenza tra:** **Primo numero: numero complesso a parte reale e immaginaria decimale positiva** **Secondo numero: numero complesso a parte reale e immaginaria intera positiva**
Test Items	Classe **BinaryCanonicOperations**, metodo **sub (ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("23"),new BigDecimal("59")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("8.624"),new BigDecimal("-36.271")); ComplexNumber result = BinaryCanonicOperations.sub(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.2.2	**Test BinaryCanonicOperations.sub (ComplexNumber n1, ComplexNumber n2) : verifica correttezza differenza tra:** **Primo numero: numero complesso a parte reale e immaginaria decimale positiva** **Secondo numero: numero complesso a parte reale intera negativa e parte immaginaria decimale negativa**
Test Items	Classe **BinaryCanonicOperations**, metodo **sub (ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("-9"),new BigDecimal("-13.729")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("40.624"),new BigDecimal("36.458")); ComplexNumber result = BinaryCanonicOperations.sub(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.2.3	**Test BinaryCanonicOperations.sub (ComplexNumber n1, ComplexNumber n2) : verifica correttezza differenza tra:** **Primo numero: numero complesso a parte reale e immaginaria decimale positiva** **Secondo numero: coniugato del primo numero**
Test Items	Classe **BinaryCanonicOperations**, metodo **sub (ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("-22.729")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("0"),new BigDecimal("45.458")); ComplexNumber result = BinaryCanonicOperations.sub(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.2.4	**Test BinaryCanonicOperations.sub (ComplexNumber n1, ComplexNumber n2) : verifica correttezza differenza tra:** **Primo numero: numero complesso a parte reale e immaginaria decimale positiva** **Secondo numero: numero reale intero positivo**
Test Items	Classe **BinaryCanonicOperations**, metodo **sub (ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("176"),new BigDecimal("0")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("-144.376"),new BigDecimal("22.729")); ComplexNumber result = BinaryCanonicOperations.sub(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.2.5	**Test BinaryCanonicOperations.sub (ComplexNumber n1, ComplexNumber n2) : verifica correttezza differenza tra:** **Primo numero: numero complesso a parte reale e immaginaria decimale positiva** **Secondo numero: numero immaginario decimale positivo**
Test Items	Classe **BinaryCanonicOperations**, metodo **sub (ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("0"),new BigDecimal("-23.618")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("46.347")); ComplexNumber result = BinaryCanonicOperations.sub(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.2.6	**Test BinaryCanonicOperations.sub (ComplexNumber n1, ComplexNumber n2) : verifica correttezza differenza tra:** **Primo numero: numero complesso a parte reale e immaginaria decimale positiva** **Secondo numero: 0**
Test Items	Classe **BinaryCanonicOperations**, metodo **sub (ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("0"),new BigDecimal("0")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber result = BinaryCanonicOperations.sub(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.2.7	**Test BinaryCanonicOperations.sub (ComplexNumber n1, ComplexNumber n2) : verifica correttezza differenza tra:** **Primo numero: numero complesso a parte reale e immaginaria decimale positiva** **Secondo numero: numero reale intero positivo molto grande**
Test Items	Classe **BinaryCanonicOperations**, metodo **sub (ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("11111111111111111"),new BigDecimal("0")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("-111111111111111079.376"),new BigDecimal("22.729")); ComplexNumber result = BinaryCanonicOperations.sub(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.2.8	**Test BinaryCanonicOperations.sub (ComplexNumber n1, ComplexNumber n2) : verifica correttezza differenza tra:** **Primo numero: numero complesso a parte reale e immaginaria intera positiva** **Secondo numero: coniguato del primo numero**
Test Items	Classe **BinaryCanonicOperations**, metodo **sub (ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("23"),new BigDecimal("59")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("23"),new BigDecimal("-59")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("0"),new BigDecimal("118")); ComplexNumber result = BinaryCanonicOperations.sub(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.2.9	**Test BinaryCanonicOperations.sub (ComplexNumber n1, ComplexNumber n2) : verifica correttezza differenza tra:** **Primo numero: numero reale intero positivo** **Secondo numero: numero immaginario intero positivo**
Test Items	Classe **BinaryCanonicOperations**, metodo **sub (ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("176"),new BigDecimal("0")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("0"),new BigDecimal("176")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("176"),new BigDecimal("-176")); ComplexNumber result = BinaryCanonicOperations.sub(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.2.10	**Test BinaryCanonicOperations.sub (ComplexNumber n1, ComplexNumber n2) : verifica correttezza differenza tra:** **Primo numero: 0** **Secondo numero: 0**
Test Items	Classe **BinaryCanonicOperations**, metodo **sub (ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("0"),new BigDecimal("0")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("0"),new BigDecimal("0")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("0"),new BigDecimal("0")); ComplexNumber result = BinaryCanonicOperations.sub(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.3.1	**Test BinaryCanonicOperations.multiply (ComplexNumber n1, ComplexNumber n2) : verifica correttezza prodotto tra: Primo numero: numero complesso a parte reale e immaginaria decimale positiva Secondo numero: numero complesso a parte reale e immaginaria intera positiva**
Test Items	Classe **BinaryCanonicOperations**, metodo multiply **(ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("23"),new BigDecimal("59")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("-613.659"),new BigDecimal("2388.583")); ComplexNumber result = BinaryCanonicOperations.multiply(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.3.2	**Test BinaryCanonicOperations.multiply (ComplexNumber n1, ComplexNumber n2) : verifica correttezza prodotto tra: Primo numero: numero complesso a parte reale e immaginaria decimale positiva Secondo numero: numero complesso con parte reale intera negativa e parte immaginaria decimale negativa**
Test Items	Classe **BinaryCanonicOperations**, metodo multiply **(ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("-9"),new BigDecimal("-13.729")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("27.430"),new BigDecimal("-638.727")); ComplexNumber result = BinaryCanonicOperations.multiply(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.3.3	**Test BinaryCanonicOperations.multiply (ComplexNumber n1, ComplexNumber n2) : verifica correttezza prodotto tra: Primo numero: numero complesso a parte reale e immaginaria decimale positiva Secondo numero: il coniguato del primo numero**
Test Items	Classe **BinaryCanonicOperations**, metodo multiply **(ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("-22.729")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("1516.685"),new BigDecimal("0")); ComplexNumber result = BinaryCanonicOperations.multiply(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.3.4	**Test BinaryCanonicOperations.multiply (ComplexNumber n1, ComplexNumber n2) : verifica correttezza prodotto tra: Primo numero: numero complesso a parte reale e immaginaria decimale positiva Secondo numero: numero reale intero positivo**
Test Items	Classe **BinaryCanonicOperations**, metodo multiply **(ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("176"),new BigDecimal("0")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("5565.824"),new BigDecimal("4000.304")); ComplexNumber result = BinaryCanonicOperations.multiply(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.3.5	**Test BinaryCanonicOperations.multiply (ComplexNumber n1, ComplexNumber n2) : verifica correttezza prodotto tra: Primo numero: numero complesso a parte reale e immaginaria decimale positiva Secondo numero: numero immaginario decimale negativo**
Test Items	Classe **BinaryCanonicOperations**, metodo multiply **(ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("0"),new BigDecimal("-23.618")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("536.814"),new BigDecimal("-746.896")); ComplexNumber result = BinaryCanonicOperations.multiply(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.3.6	**Test BinaryCanonicOperations.multiply (ComplexNumber n1, ComplexNumber n2) : verifica correttezza prodotto tra: Primo numero: numero complesso a parte reale e immaginaria decimale positiva Secondo numero: 0**
Test Items	Classe **BinaryCanonicOperations**, metodo multiply **(ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("0"),new BigDecimal("0")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("0"),new BigDecimal("0")); ComplexNumber result = BinaryCanonicOperations.multiply(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.3.7	**Test BinaryCanonicOperations.multiply (ComplexNumber n1, ComplexNumber n2) : verifica correttezza prodotto tra: Primo numero: numero complesso a parte reale e immaginaria intera positiva Secondo numero: il coniugato del primo numero**
Test Items	Classe **BinaryCanonicOperations**, metodo multiply **(ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("23"),new BigDecimal("59")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("23"),new BigDecimal("-59")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("4010"),new BigDecimal("0")); ComplexNumber result = BinaryCanonicOperations.multiply(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.3.8	**Test BinaryCanonicOperations.multiply (ComplexNumber n1, ComplexNumber n2) : verifica correttezza prodotto tra: Primo numero: numero reale intero positivo Secondo numero: numero immaginario intero positivo**
Test Items	Classe **BinaryCanonicOperations**, metodo multiply **(ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("176"),new BigDecimal("0")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("0"),new BigDecimal("176")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("0"),new BigDecimal("30976")); ComplexNumber result = BinaryCanonicOperations.multiply(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.3.9	**Test BinaryCanonicOperations.multiply (ComplexNumber n1, ComplexNumber n2) : verifica correttezza prodotto tra:** **Primo numero: 0** **Secondo numero: 0**
Test Items	Classe **BinaryCanonicOperations**, metodo multiply **(ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("0"),new BigDecimal("0")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("0"),new BigDecimal("0")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("0"),new BigDecimal("0")); ComplexNumber result = BinaryCanonicOperations.multiply(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.4.1	**Test BinaryCanonicOperations.divide (ComplexNumber n1, ComplexNumber n2) : verifica correttezza prodotto tra:** **Primo numero: numero complesso con parte reale e immaginaria decimale positiva** **Secondo numero: numero complesso con parte reale e immaginaria intera positiva**
Test Items	Classe **BinaryCanonicOperations**, metodo divide **(ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("23"),new BigDecimal("59")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("0.516"),new BigDecimal("-0.335")); ComplexNumber result = BinaryCanonicOperations.divide(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.4.2	**Test BinaryCanonicOperations.divide (ComplexNumber n1, ComplexNumber n2) : verifica correttezza prodotto tra: Primo numero: numero complesso con parte reale e immaginaria decimale positiva Secondo numero: numero complesso con parte reale intera negativa e parte immaginaria decimale negativa**
Test Items	Classe **BinaryCanonicOperations**, metodo divide **(ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("-9"),new BigDecimal("-13.729")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("-2.214"),new BigDecimal("0.852")); ComplexNumber result = BinaryCanonicOperations.divide(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.4.3	**Test BinaryCanonicOperations.divide (ComplexNumber n1, ComplexNumber n2) : verifica correttezza prodotto tra: Primo numero: numero complesso con parte reale e immaginaria decimale positiva Secondo numero: coniguato del primo numero**
Test Items	Classe **BinaryCanonicOperations**, metodo divide **(ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("-22.729")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("0.319"),new BigDecimal("0.948")); ComplexNumber result = BinaryCanonicOperations.divide(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.4.4	**Test BinaryCanonicOperations.divide (ComplexNumber n1, ComplexNumber n2) : verifica correttezza prodotto tra: Primo numero: numero complesso con parte reale e immaginaria decimale positiva Secondo numero: numero reale intero positivo**
Test Items	Classe **BinaryCanonicOperations**, metodo divide **(ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("176"),new BigDecimal("0")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("0.180"),new BigDecimal("0.129")); ComplexNumber result = BinaryCanonicOperations.divide(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.4.5	**Test BinaryCanonicOperations.divide (ComplexNumber n1, ComplexNumber n2) : verifica correttezza prodotto tra: Primo numero: numero complesso con parte reale e immaginaria decimale positiva Secondo numero: numero immaginario decimale negativo**
Test Items	Classe **BinaryCanonicOperations**, metodo divide **(ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("0"),new BigDecimal("-23.618")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("-0.962"),new BigDecimal("1.339")); ComplexNumber result = BinaryCanonicOperations.divide(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.4.6	**Test BinaryCanonicOperations.divide (ComplexNumber n1, ComplexNumber n2) : verifica correttezza prodotto tra: Primo numero: numero complesso con parte reale e immaginaria decimale positiva Secondo numero: 0**
Test Items	Classe **BinaryCanonicOperations**, metodo divide **(ComplexNumber n1, ComplexNumber n2)**

Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("31.624"),new BigDecimal("22.729"));
	
	ComplexNumber n2 = new ComplexNumber(new BigDecimal("0"),new BigDecimal("0"));
	
	boolean thrown = false;
	try{
	ComplexNumber result = BinaryCanonicOperations.divide(n1, n2);
	}catch(ArithmeticException exc){
	thrown = true;
	}

Oracle	thrown == true

UTC-2.4.7	**Test BinaryCanonicOperations.divide (ComplexNumber n1, ComplexNumber n2) : verifica correttezza prodotto tra: Primo numero: numero complesso con parte reale e immaginaria intera positiva Secondo numero: coniugato del primo numero**
Test Items	Classe **BinaryCanonicOperations**, metodo divide **(ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("23"),new BigDecimal("59")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("23"),new BigDecimal("-59")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("-0.736"),new BigDecimal("0.677")); ComplexNumber result = BinaryCanonicOperations.divide(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.4.8	**Test BinaryCanonicOperations.divide (ComplexNumber n1, ComplexNumber n2) : verifica correttezza prodotto tra: Primo numero: numero reale intero positivo Secondo numero: numero immaginario intero positivo**
Test Items	Classe **BinaryCanonicOperations**, metodo divide **(ComplexNumber n1, ComplexNumber n2)**
Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("176"),new BigDecimal("0")); ComplexNumber n2 = new ComplexNumber(new BigDecimal("0"),new BigDecimal("176")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("0"),new BigDecimal("-1")); ComplexNumber result = BinaryCanonicOperations.divide(n1, n2);
Oracle	expResult.equals(result) == true

UTC-2.4.9	**Test BinaryCanonicOperations.divide (ComplexNumber n1, ComplexNumber n2) : verifica correttezza prodotto tra: Primo numero: 0 Secondo numero: 0**
Test Items	Classe **BinaryCanonicOperations**, metodo divide **(ComplexNumber n1, ComplexNumber n2)**

Input	ComplexNumber n1 = new ComplexNumber(new BigDecimal("0"),new BigDecimal("0"));
	
	ComplexNumber n2 = new ComplexNumber(new BigDecimal("0"),new BigDecimal("0"));
	
	boolean thrown = false;
	try{
	ComplexNumber result = BinaryCanonicOperations.divide(n1, n2);
	}catch(ArithmeticException exc){
	thrown = true;
	}

Oracle	thrown == true

3- Class UnaryCanonicOperations

UTC-3.1.1	**Test UnaryCanonicOperations.squareRoot(ComplexNumber n) : Numero inserito con parte reale positiva e parte immaginaria positiva.**
Test Items	Classe **UnaryCanonicOperations** , metodo **squareRoot(ComplexNumber n)**

Input	ComplexNumber n = new ComplexNumber(new BigDecimal("2.5"),new BigDecimal("3.5"));
	
	ComplexNumber[] expResult = new ComplexNumber[2];
	expResult[0] = new ComplexNumber(new BigDecimal("1.844"),new BigDecimal("0.949"));
	expResult[1] = new ComplexNumber(new BigDecimal("-1.844"),new BigDecimal("-0.949"));
	
	ComplexNumber[] result = UnaryCanonicOperations.squareRoot(n);

Oracle	expResult.equals(result) == true;

UTC-3.1.2	**Test UnaryCanonicOperations.squareRoot(ComplexNumber n) : Numero inserito con parte reale positive e parte immaginaria negative.**
Test Items	Classe **UnaryCanonicOperations** , metodo **squareRoot(ComplexNumber n)**

Input	ComplexNumber n = new ComplexNumber(new BigDecimal("-4.824"),new BigDecimal("-2.32"));
	
	ComplexNumber[] expResult = new ComplexNumber[2];
	expResult[0] = new ComplexNumber(new BigDecimal("0.514"),new BigDecimal("-2.256"));
	expResult[1] = new ComplexNumber(new BigDecimal("-0.514"),new BigDecimal("2.256"));
	
	ComplexNumber[] result = UnaryCanonicOperations.squareRoot(n);

Oracle	expResult.equals(result) == true;

UTC-3.1.3	**Test UnaryCanonicOperations.squareRoot(ComplexNumber n) : Numero inserito con parte reale nulla e parte immaginaria positiva.**
Test Items	Classe **UnaryCanonicOperations** , metodo **squareRoot(ComplexNumber n)**

Input	ComplexNumber n = new ComplexNumber(new BigDecimal("0"),new BigDecimal("5.214"));
	
	ComplexNumber[] expResult = new ComplexNumber[2];
	expResult[0] = new ComplexNumber(new BigDecimal("1.615"),new BigDecimal("1.615"));
	expResult[1] = new ComplexNumber(new BigDecimal("-1.615"),new BigDecimal("-1.615"));
	
	ComplexNumber[] result = UnaryCanonicOperations.squareRoot(n);

Oracle	expResult.equals(result) == true;

UTC-3.1.4	**Test UnaryCanonicOperations.squareRoot(ComplexNumber n) : Numero inserito con parte reale positiva e parte immaginaria nulla.**
Test Items	Classe **UnaryCanonicOperations** , metodo **squareRoot(ComplexNumber n)**

Input	ComplexNumber n = new ComplexNumber(new BigDecimal("15.457"),new BigDecimal("0.000"));
	
	ComplexNumber[] expResult = new ComplexNumber[2];
	expResult[0] = new ComplexNumber(new BigDecimal("3.932"),new BigDecimal("0"));
	expResult[1] = new ComplexNumber(new BigDecimal("-3.932"),new BigDecimal("0"));
	
	ComplexNumber[] result = UnaryCanonicOperations.squareRoot(n);

Oracle	expResult.equals(result) == true;

UTC-3.1.5	**Test UnaryCanonicOperations.squareRoot(ComplexNumber n) : Numero inserito nullo.**
Test Items	Classe **UnaryCanonicOperations** , metodo **squareRoot(ComplexNumber n)**

Input	ComplexNumber n = new ComplexNumber(new BigDecimal("0.00006"),new BigDecimal("0"));
	
	ComplexNumber[] expResult = new ComplexNumber[2];
	expResult[0] = new ComplexNumber(new BigDecimal("0"),new BigDecimal("0"));
	expResult[1] = new ComplexNumber(new BigDecimal("0"),new BigDecimal("0"));
	
	ComplexNumber[] result = UnaryCanonicOperations.squareRoot(n);

Oracle	expResult.equals(result) == true;

UTC-3.1.6	**Test UnaryCanonicOperations.squareRoot(ComplexNumber n) : Numero inserito con parte reale positiva e parte immaginaria negativa.**
Test Items	Classe **UnaryCanonicOperations** , metodo **squareRoot(ComplexNumber n)**

Input	ComplexNumber n = new ComplexNumber(new BigDecimal("35469.214"),new BigDecimal("-514.2416"));
	
	ComplexNumber[] expResult = new ComplexNumber[2];
	expResult[0] = new ComplexNumber(new BigDecimal("188.338"),new BigDecimal("-1.365"));
	expResult[1] = new ComplexNumber(new BigDecimal("-188.338"),new BigDecimal("1.365"));
	
	ComplexNumber[] result = UnaryCanonicOperations.squareRoot(n);

Oracle	expResult.equals(result) == true;

UTC-3.1.7	**Test UnaryCanonicOperations.squareRoot(ComplexNumber n) : Numero inserito con parte reale negative e parte immaginaria positiva.**
Test Items	Classe **UnaryCanonicOperations** , metodo **squareRoot(ComplexNumber n)**

Input	ComplexNumber n = new ComplexNumber(new BigDecimal("-2145.98468"),new BigDecimal("14"));
	
	ComplexNumber[] expResult = new ComplexNumber[2];
	expResult[0] = new ComplexNumber(new BigDecimal("0.151"),new BigDecimal("46.325"));
	expResult[1] = new ComplexNumber(new BigDecimal("-0.151"),new BigDecimal("-46.325"));
	
	ComplexNumber[] result = UnaryCanonicOperations.squareRoot(n);

Oracle	expResult.equals(result) == true;

UTC-3.2.1	**Test UnaryCanonicOperations.changeSign(ComplexNumber n) : Numero inserito con parte reale positiva e parte immaginaria positiva.**
Test Items	Classe **UnaryCanonicOperations** , metodo **changeSign(ComplexNumber n)**
Input	ComplexNumber n = new ComplexNumber(new BigDecimal("31.6238"),new BigDecimal("22.729")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("-31.6238"),new BigDecimal("-22.729")); ComplexNumber result = UnaryCanonicOperations.changeSign(n);
Oracle	expResult.equals(result) == true;

UTC-3.2.2	**Test UnaryCanonicOperations.changeSign(ComplexNumber n) : Numero inserito con parte reale negativa e parte immaginaria negativa.**
Test Items	Classe **UnaryCanonicOperations** , metodo **changeSign(ComplexNumber n)**
Input	ComplexNumber n = new ComplexNumber(new BigDecimal("-12.528"),new BigDecimal("-74")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("12.528"),new BigDecimal("74")); ComplexNumber result = UnaryCanonicOperations.changeSign(n);
Oracle	expResult.equals(result) == true;

UTC-3.2.3	**Test UnaryCanonicOperations.changeSign(ComplexNumber n) : Numero inserito con parte reale positiva e parte immaginaria negativa.**
Test Items	Classe **UnaryCanonicOperations** , metodo **changeSign(ComplexNumber n)**
Input	ComplexNumber n = new ComplexNumber(new BigDecimal("23"),new BigDecimal("-13.729")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("-23"),new BigDecimal("13.729")); ComplexNumber result = UnaryCanonicOperations.changeSign(n);
Oracle	expResult.equals(result) == true;

UTC-3.2.4	**Test UnaryCanonicOperations.changeSign(ComplexNumber n) : Numero inserito con parte reale negativa e parte immaginaria positiva.**
Test Items	Classe **UnaryCanonicOperations** , metodo **changeSign(ComplexNumber n)**
Input	ComplexNumber n = new ComplexNumber(new BigDecimal("-12.528"),new BigDecimal("74")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("12.528"),new BigDecimal("-74")); ComplexNumber result = UnaryCanonicOperations.changeSign(n);
Oracle	expResult.equals(result) == true;

UTC-3.2.5	**Test UnaryCanonicOperations.changeSign (ComplexNumber n) : Numero inserito con parte reale positiva e parte immaginaria nulla.**
Test Items	Classe **UnaryCanonicOperations** , metodo **changeSign(ComplexNumber n)**
Input	ComplexNumber n = new ComplexNumber(new BigDecimal("176"),new BigDecimal("0")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("-176"),new BigDecimal("0")); ComplexNumber result = UnaryCanonicOperations.changeSign(n);
Oracle	expResult.equals(result) == true;

UTC-3.2.6	**Test UnaryCanonicOperations.changeSign (ComplexNumber n) : Numero inserito con parte reale negativa e parte immaginaria nulla.**
Test Items	Classe **UnaryCanonicOperations** , metodo **changeSign(ComplexNumber n)**
Input	ComplexNumber n = new ComplexNumber(new BigDecimal("-176"),new BigDecimal("0")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("176"),new BigDecimal("0")); ComplexNumber result = UnaryCanonicOperations.changeSign(n);
Oracle	expResult.equals(result) == true;

UTC-3.2.7	**Test UnaryCanonicOperations.changeSign (ComplexNumber n) : Numero inserito con parte reale nulla e parte immaginaria positiva.**
Test Items	Classe **UnaryCanonicOperations** , metodo **changeSign(ComplexNumber n)**
Input	ComplexNumber n = new ComplexNumber(new BigDecimal("0"),new BigDecimal("23.618")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("0"),new BigDecimal("-23.618")); ComplexNumber result = UnaryCanonicOperations.changeSign(n);
Oracle	expResult.equals(result) == true;

UTC-3.2.8	**Test UnaryCanonicOperations.changeSign (ComplexNumber n) : Numero inserito con parte reale nulla e parte immaginaria negativa.**
Test Items	Classe **UnaryCanonicOperations** , metodo **changeSign(ComplexNumber n)**
Input	ComplexNumber n = new ComplexNumber(new BigDecimal("0"),new BigDecimal("-23.618")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("0"),new BigDecimal("23.618")); ComplexNumber result = UnaryCanonicOperations.changeSign(n);
Oracle	expResult.equals(result) == true;

UTC-3.2.9	**Test UnaryCanonicOperations.changeSign (ComplexNumber n) : Numero inserito nullo.**
Test Items	Classe **UnaryCanonicOperations** , metodo **changeSign(ComplexNumber n)**
Input	ComplexNumber n = new ComplexNumber(new BigDecimal("0"),new BigDecimal("0")); ComplexNumber expResult = new ComplexNumber(new BigDecimal(""),new BigDecimal("0")); ComplexNumber result = UnaryCanonicOperations.changeSign(n);
Oracle	expResult.equals(result) == true;

UTC-3.2.10	**Test UnaryCanonicOperations.changeSign (ComplexNumber n) : Numero immaginario con parte decimale molto grande**
Test Items	Classe **UnaryCanonicOperations** , metodo **changeSign(ComplexNumber n)**
Input	ComplexNumber n = new ComplexNumber(new BigDecimal("11111111111111111111111111"),new BigDecimal("0")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("-11111111111111111111111111"),new BigDecimal("0")); ComplexNumber result = UnaryCanonicOperations.changeSign(n);
Oracle	expResult.equals(result) == true;

UTC-3.2.11	**Test UnaryCanonicOperations.changeSign (ComplexNumber n) : Numero inserito con parte reale negativa e parte immaginaria positiva.**
Test Items	Classe **UnaryCanonicOperations** , metodo **changeSign(ComplexNumber n)**
Input	ComplexNumber n = new ComplexNumber(new BigDecimal("0 "),new BigDecimal("2.1584926487859")); ComplexNumber expResult = new ComplexNumber(new BigDecimal("0"),new BigDecimal("-2.1584926487859")); ComplexNumber result = UnaryCanonicOperations.changeSign(n);
Oracle	expResult.equals(result) == true;

4- Classe Stack

UTC-4.1.1	**Test Stack.getStack() : verifica della correttezza del riferimento allo stack restituito dal metodo getStack()**
Test Items	Classe **Stack**, metodo **getStack()**

Input	Stack instance = new Stack();
	
	Field field = instance.getClass().getDeclaredField("stack");
	
	field.setAccessible(true);
	
	Deque<ComplexNumber> testGet = new LinkedList<>();
	
	field.set(instance, testGet);
	
	Deque<ComplexNumber> result = instance.getStack();

Oracle	testGet.equals(result) == true;

UTC-4.2.1	**Test Stack.push() : inserimento nello stack di un numero complesso a parte reale e immaginaria decimale e negativa**
Test Items	Classe **Stack**, metodo **push()**
Input	ComplexNumber input = new ComplexNumber(new BigDecimal(-12.5284),new BigDecimal(-74.5269)); Stack instance = new Stack(); instance.push(input);
Oracle	testGet.equals(result) == true;

UTC-4.2.2	**Test Stack.push() : inserimento nello stack di un numero complesso a parte reale decimale positiva e parte immaginaria decimale negativa**
Test Items	Classe **Stack**, metodo **push()**

Input	ComplexNumber input = new ComplexNumber(new BigDecimal(31.6238),new BigDecimal(-22.729));
	
	Stack instance = new Stack();
	
	instance.push(input);
	
	assertEquals(input,instance.getStack().getFirst());

Oracle	input.equals(instance.getStack().getFirst()) == true;

UTC-4.2.3	**Test Stack.push() : inserimento nello stack di un numero complesso a parte reale decimale negativa e parte immaginaria decimale positiva**
Test Items	Classe **Stack**, metodo **push()**
Input	ComplexNumber input = new ComplexNumber(new BigDecimal(-12.5284),new BigDecimal(74.5269)); Stack instance = new Stack(); instance.push(input);
Oracle	input.equals(instance.getStack().getFirst()) == true;

UTC-4.2.4	**Test Stack.push() : inserimento nello stack di un numero reale intero positivo**
Test Items	Classe **Stack**, metodo **push()**
Input	ComplexNumber input = new ComplexNumber(new BigDecimal(176),new BigDecimal(0)); Stack instance = new Stack(); instance.push(input);
Oracle	input.equals(instance.getStack().getFirst()) == true;

UTC-4.2.5	Test Stack.push() : inserimento nello stack di un numero reale intero negativo
Test Items	Classe **Stack**, metodo **push()**
Input	ComplexNumber input = new ComplexNumber(new BigDecimal(-176),new BigDecimal(0)); Stack instance = new Stack(); instance.push(input);
Oracle	input.equals(instance.getStack().getFirst()) == true;

UTC-4.2.6	Test Stack.push() : inserimento nello stack di un numero reale decimale positivo
Test Items	Classe **Stack**, metodo **push()**
Input	ComplexNumber input = new ComplexNumber(new BigDecimal(23.6189),new BigDecimal(0)); Stack instance = new Stack(); instance.push(input);
Oracle	input.equals(instance.getStack().getFirst()) == true;

UTC-4.2.7	Test Stack.push() : inserimento nello stack di un numero reale decimale negativo
Test Items	Classe **Stack**, metodo **push()**
Input	ComplexNumber input = new ComplexNumber(new BigDecimal(-23.6189),new BigDecimal(0)); Stack instance = new Stack(); instance.push(input);
Oracle	input.equals(instance.getStack().getFirst()) == true;

UTC-4.2.8	Test Stack.push() : inserimento nello stack di un numero immaginario intero positivo
Test Items	Classe **Stack**, metodo **push()**
Input	ComplexNumber input = new ComplexNumber(new BigDecimal(0),new BigDecimal(176)); Stack instance = new Stack(); instance.push(input);
Oracle	input.equals(instance.getStack().getFirst()) == true;

UTC-4.2.9	Test Stack.push() : inserimento nello stack di un numero immaginario intero negativo
Test Items	Classe **Stack**, metodo **push()**
Input	ComplexNumber input = new ComplexNumber(new BigDecimal(0),new BigDecimal(-176)); Stack instance = new Stack(); instance.push(input);
Oracle	input.equals(instance.getStack().getFirst()) == true;

UTC-4.2.10	Test Stack.push() : inserimento nello stack di un numero immaginario decimale positivo
Test Items	Classe **Stack**, metodo **push()**
Input	ComplexNumber input = new ComplexNumber(new BigDecimal(0),new BigDecimal(23.6189)); Stack instance = new Stack(); instance.push(input);
Oracle	input.equals(instance.getStack().getFirst()) == true;

UTC-4.2.11	**Test Stack.push() : inserimento nello stack di un numero immaginario decimale negativo**
Test Items	Classe **Stack**, metodo **push()**
Input	ComplexNumber input = new ComplexNumber(new BigDecimal(0),new BigDecimal(-23.6189)); Stack instance = new Stack(); instance.push(input);
Oracle	input.equals(instance.getStack().getFirst()) == true;

UTC-4.2.12	**Test Stack.push() : inserimento nello stack di un numero reale intero positivo molto grande**
Test Items	Classe **Stack**, metodo **push()**
Input	ComplexNumber input = new ComplexNumber(new BigDecimal("1111111111111111"),new BigDecimal(0)); Stack instance = new Stack(); instance.push(input);
Oracle	input.equals(instance.getStack().getFirst()) == true;

UTC-4.2.13	**Test Stack.push() : inserimento nello stack di un numero reale positivo con molte cifre decimali**
Test Items	Classe **Stack**, metodo **push()**
Input	ComplexNumber input = new ComplexNumber(new BigDecimal("2.1584926487859"),new BigDecimal(0)); Stack instance = new Stack(); instance.push(input);
Oracle	input.equals(instance.getStack().getFirst()) == true;

UTC-4.2.14	**Test Stack.push() : inserimento nello stack dell'unità immaginaria**
Test Items	Classe **Stack**, metodo **push()**
Input	ComplexNumber input = new ComplexNumber(new BigDecimal(0),new BigDecimal(-1)); Stack instance = new Stack(); instance.push(input);
Oracle	input.equals(instance.getStack().getFirst()) == true;

UTC-4.2.15	**Test Stack.push() : inserimento nello stack di 0**
Test Items	Classe **Stack**, metodo **push()**
Input	ComplexNumber input = new ComplexNumber(new BigDecimal(0.0),new BigDecimal(0.0)); Stack instance = new Stack(); instance.push(input);
Oracle	input.equals(instance.getStack().getFirst()) == true;

UTC-4.3.1	**Test Stack.pop() : verifica che il metodo pop restituisca un numero precedentemente inserito nello stack con push()**
Test Items	Classe **Stack**, metodo **pop()**
Input	Stack instance = new Stack(); ComplexNumber expResult = new ComplexNumber(new BigDecimal(31.6238),new BigDecimal(22.729)); instance.push(expResult); ComplexNumber result = instance.pop();
Oracle	expResult.equals(result) == true;

UTC-4.3.2	**Test Stack.pop() : verifica che il numero restituito dal metodo pop() sia effettivamente quello presente nella cima dello stack coerentemente alla logica LIFO di uno stack**
Test Items	Classe **Stack**, metodo **pop()**

Input	Stack instance = new Stack();
	
	ComplexNumber expResult1 = new ComplexNumber(new BigDecimal(31.6238),new BigDecimal(22.729));
	
	instance.push(expResult1);
	
	ComplexNumber expResult2 = new ComplexNumber(new BigDecimal(-12.5284),new BigDecimal(74.5269));
	
	instance.push(expResult2);
	
	ComplexNumber result2 = instance.pop();
	
	ComplexNumber result1 = instance.pop();
	
	assertEquals(expResult1, result1);
	
	assertEquals(expResult2, result2);

Oracle	expResult1.equals(result1) == true && expResult.equals(result) == true

UTC-4.3.3	**Test Stack.pop() : caso di pop() su stack vuoto (Lancia eccezione)**
Test Items	Classe **Stack**, metodo **pop()**

Input	boolean thrown = false;
	
	Stack instance = new Stack();
	try{
	ComplexNumber result = instance.pop();
	}catch(InvalidOperandsException ex){
	thrown = true;
	}

Oracle	thrown == true

UTC-4.4.1	**Test Stack.top() : verifica che il numero restituito dal metodo top() sia effettivamente quello presente nella cima dello stack coerentemente alla logica LIFO di uno stack**
Test Items	Classe **Stack**, metodo **top()**
Input	Stack instance = new Stack(); ComplexNumber expResult = new ComplexNumber(new BigDecimal(31.6238),new BigDecimal(22.729)); instance.push(expResult); ComplexNumber result = instance.top();
Oracle	expResult.equals(result) == true

UTC-4.4.2	**Test Stack.top() : caso di top() su stack vuoto (Lancia eccezione)**
Test Items	Classe **Stack**, metodo **top()**

Input	boolean thrown = false;
	
	Stack instance = new Stack();
	
	try{
	  ComplexNumber result = instance.top();
	}catch(InvalidOperandsException ex){
	  thrown = true;
	}

Oracle	thrown == true

UTC-4.5.1	**Test Stack.clear() : verifica che il metodo clear ripulisca correttamente lo stack**
Test Items	Classe **Stack**, metodo **clear()**

Input	Stack instance = new Stack();
	
	ComplexNumber n1 = new ComplexNumber(new BigDecimal(31.6238),new BigDecimal(22.729));
	
	ComplexNumber n2 = new ComplexNumber(new BigDecimal(-12.5284),new BigDecimal(74.5269));
	
	instance.push(n1);
	
	instance.push(n2);
	
	instance.clear();

Oracle	instance.getStack().isEmpty() == true

UTC-4.6.1	**Test Stack.drop () : verifica che il metodo drop lanci un'eccezione nel caso di stack vuoto**
Test Items	Classe **Stack**, metodo **drop()**

Input	boolean thrown = false;
	
	Stack instance = new Stack();
	
	try{
	instance.drop();
	}catch(InvalidOperandsException ex){
	thrown = true;
	}

Oracle	thrown == true

UTC-4.6.2	**Test Stack.drop() : verifica che il metodo elimini correttamente dalla cima dello stack l'ultimo numero inserito**
Test Items	Classe **Stack**, metodo **drop()**

Input	Stack instance = new Stack();
	
	ComplexNumber n1 = new ComplexNumber(new BigDecimal(31.6238),new BigDecimal(22.729));
	
	instance.push(n1);
	
	instance.drop();

Oracle	instance.getStack().isEmpty();== true

UTC-4.6.3	**Test Stack.drop () : verifica che dopo la drop il numero di elementi nello stack sia coerente**
Test Items	Classe **Stack**, metodo **drop()**

Input	int before;
	
	Stack instance = new Stack();
	
	ComplexNumber n1 = new ComplexNumber(new BigDecimal(31.6238),new BigDecimal(22.729));
	
	ComplexNumber n2 = new ComplexNumber(new BigDecimal(-12.5284),new BigDecimal(74.5269));
	
	instance.push(n1);
	
	instance.push(n2);
	
	before = instance.getStack().size();
	
	instance.drop();
	
	before-=1;

Oracle	Before == instance.getStack().size();

UTC-4.7.1	**Test Stack.swap() : verifica che la swap() sollevi un'eccezione nel caso lo stack sia vuoto**
Test Items	Classe **Stack**, metodo **swap()**

Input	boolean thrown = false;
	
	Stack instance = new Stack();
	
	try{
	instance.swap();
	}catch(InvalidOperandsException ex){
	thrown = true;
	}

Oracle	thrown == true

UTC-4.7.2	**Test Stack.swap() : verifica che la swap() sollevi un'eccezione nel caso ci sia un solo element nello stack**
Test Items	Classe **Stack**, metodo **swap()**

Input	boolean thrown = false;
	
	Stack instance = new Stack();
	
	ComplexNumber n1 = new ComplexNumber(new BigDecimal(31.6238),new BigDecimal(22.729));
	
	instance.push(n1);
	
	try{
	instance.swap();
	}catch(InvalidOperandsException ex){
	thrown = true;
	}

Oracle	thrown == true

UTC-4.7.3	**Test Stack.swap() : verifichi che la swap inverta correttamente l'ordine degli ultimi due elementi inseriti nello stack**
Test Items	Classe **Stack**, metodo **swap()**

Input	Stack instance = new Stack();
	
	ComplexNumber n1 = new ComplexNumber(new BigDecimal(31.6238),new BigDecimal(22.729));
	
	ComplexNumber n2 = new ComplexNumber(new BigDecimal(-12.5284),new BigDecimal(74.5269));
	
	instance.push(n1);
	
	instance.push(n2);
	
	instance.swap();

Oracle	n1.equals(instance.pop()) == true && n2.equals(instance.pop()) == true

UTC-4.8.1	**Test Stack.duplicate() : verifica che il duplicate lanci un'eccezione se lo stack è vuoto**
Test Items	Classe **Stack**, metodo **duplicate()**

Input	boolean thrown = false;
	
	Stack instance = new Stack();
	
	try{
	instance.duplicate();
	}catch(InvalidOperandsException ex){
	thrown = true;
	}

Oracle	thrown == true

UTC-4.8.2	**Test Stack.duplicate() : verifica che il duplicate duplichi correttmente l'elemento nella cima dello stack**
Test Items	Classe **Stack**, metodo **duplicate()**

Input	Stack instance = new Stack();
	
	ComplexNumber n1 = new ComplexNumber(new BigDecimal(31.6238),new BigDecimal(22.729));
	
	instance.push(n1);
	
	instance.duplicate();

Oracle	n1.equals(instance.pop()) == true && n1.equals(instance.pop())

UTC-4.9.1	**Test Stack.over() : verifica che la over lanci correttamente un'eccezione se lo stack è vuoto**
Test Items	Classe **Stack**, metodo **over()**

Input	boolean thrown = false;
	Stack instance = new Stack();
	try{
	instance.over();
	}catch(InvalidOperandsException ex){
	thrown = true;
	}

Oracle	thrown == true

UTC-4.9.2	**Test Stack.over() : verifica che la over lanci correttamente un'eccezione se nello stack è inserito un solo elemento**
Test Items	Classe **Stack**, metodo **over()**

Input	boolean thrown = false;
	
	Stack instance = new Stack();
	
	ComplexNumber n1 = new ComplexNumber(new BigDecimal(31.6238),new BigDecimal(22.729));
	
	instance.push(n1);
	
	try{
	instance.over();
	}catch(InvalidOperandsException ex){
	thrown = true;
	}

Oracle	thrown == true

UTC-4.9.3	**Test Stack.over() : verifica che la over() duplichi correttamente l'elemento nella seconda posizione dalla cima**
Test Items	Classe **Stack**, metodo **over()**

Input	Stack instance = new Stack();
	
	ComplexNumber n1 = new ComplexNumber(new BigDecimal(31.6238),new BigDecimal(22.729));
	
	ComplexNumber n2 = new ComplexNumber(new BigDecimal(-12.5284),new BigDecimal(74.5269));
	
	instance.push(n1);
	
	instance.push(n2);
	
	instance.over();

Oracle	n1.equals(instance.pop()) == true && n2.equals(instance.pop()) == true && n1.equals(instance.pop()) == true

5- Classe Vars

UTC-5.1.1	**Test Vars.getStack() : verifica della correttezza del riferimento allo stack restituito dal metodo getStack()**
Test Items	Classe **Vars**, metodo **getStack()**
Input	Stack expResult = new Stack(); Vars instance = new Vars(expResult); Stack result = instance.getStack(); assertEquals(expResult, result);
Oracle	expResult.equals(result) == true

UTC-5.2.1	**Test Vars.getVariables() : verifica della correttezza del riferimento all'HashMap restituita dal metodo getVariables()**
Test Items	Classe **Vars**, metodo **getVariables()**

Input	Vars instance = new Vars(new Stack());
	Map<Character, ComplexNumber> expResult = new HashMap<>();
	Field field = instance.getClass().getDeclaredField("variables");
	field.setAccessible(true);
	field.set(instance,expResult);
	Map<Character, ComplexNumber> result = instance.getVariables();
	assertEquals(expResult, result);

Oracle	expResult.equals(result) == true

UTC-5.3.1	**Test Vars.getValueOf(Character variable) : verifico la correttezza del valore di ritorno del metodo getValueOf(Character variable)**
Test Items	Classe **Vars**, metodo **getValueOf()**
Input	Character variable = 'A'; Vars instance = new Vars(new Stack()); ComplexNumber expResult = new ComplexNumber(new BigDecimal(-1.5), new BigDecimal(2.3)); instance.getVariables().put(variable, expResult); ComplexNumber result = instance.getValueOf(variable);
Oracle	expResult.equals(result)

UTC-5.4.1	**Test Vars.setValueOf(Character variable) : verifico l'uguaglianza tra il numero complesso passato come parametro e quello presente nell'HashMap della classe Vars**
Test Items	Classe **Vars**, metodo **setValueOf(Character variable, ComplexNumber n)**
Input	Character variable = 'A'; ComplexNumber n = new ComplexNumber(new BigDecimal(-1.5), new BigDecimal(2.3)); Vars instance = new Vars(new Stack()); instance.setValueOf(variable, n);
Oracle	n.equals(instance.getValueOf(Character variable))

UTC-5.5.1	**Test Vars.popFromStack(Character variable) : verifico l'uguaglianza tra il numero complesso salvato nella variabile e quello presente nel top dello stack.**
Test Items	Classe **Vars**, metodo **popFromStack(Character variable)**

Input	Character variable = 'A';
	
	Stack s = new Stack();
	
	Vars instance = new Vars(s);
	
	ComplexNumber n = new ComplexNumber(new BigDecimal(-1.5), new BigDecimal(2.3));
	
	s.push(n);
	
	instance.popFromStack(variable);

Oracle	n.equals(instance.getValueOf(Character variable))

UTC-5.5.2	**Test Vars.popFromStack(Character variable) : verifico che in caso di stack vuoto venga lanciata l'eccezione correlata**
Test Items	Classe **Vars**, metodo **popFromStack(Character variable)**

Input	boolean thrown = false;
	
	Character variable = 'A';
	
	Vars instance = new Vars(new Stack());
	
	try{
	  instance.popFromStack(variable);
	}catch(InvalidOperandsException ex){
	  thrown = true;
	}

Oracle	thrown == true

UTC-5.6.1	**Test Vars.pushInStack(Character variable) : verifico l'uguaglianza tra il numero complesso rimosso dalla variabile e quello inserito nel top dello stack. Inoltre verifico che essa venga rimossa dall'HashMap (Caso : 1 variabile inizializzata)**
Test Items	Classe **Vars**, metodo **pushInStack(Character variable)**

Input	Character variable = 'A';
	
	ComplexNumber n = new ComplexNumber(new BigDecimal(-1.5),
	new BigDecimal(2.3));
	
	Stack s = new Stack();
	
	Vars instance = new Vars(s);
	
	instance.setValueOf(variable, n);
	
	instance.pushInStack(variable);

Oracle	n.equals(s.pop()) == true && instance.getVariables().isEmpty() == true

UTC-5.6.2	**Test Vars.pushInStack(Character variable) : verifico l'uguaglianza tra il numero complesso rimosso dalla variabile e quello inserito nel top dello stack. Inoltre verifico che essa venga rimossa dall'HashMap (Caso : >1 variabile inizializzata)**
Test Items	Classe **Vars**, metodo **pushInStack(Character variable)**

Input	Character variable = 'A';
	
	ComplexNumber n = new ComplexNumber(new BigDecimal(-1.5),
	new BigDecimal(2.3));
	
	Stack s = new Stack();
	
	Vars instance = new Vars(s);
	
	instance.setValueOf(variable, n);
	
	variable = 'B';
	
	n = new ComplexNumber(new BigDecimal(2.5),new BigDecimal(4.3));
	
	instance.setValueOf(variable, n);
	
	int before = instance.getVariables().size();

	instance.pushInStack(variable);
Oracle	n.equals(s.pop()) == true && instance.getVariables().size() == (before - 1)

UTC-5.6.3	**Test Vars.pushInStack(Character variable) : verifico che venga lanciata l'eccezione nel caso in cui la variabile non è inizializzata**
Test Items	Classe **Vars**, metodo **pushInStack(Character variable)**

Input	boolean thrown = false;
	
	Character variable = 'A';
	
	Vars instance = new Vars(new Stack());
	try{
	instance.pushInStack(variable);
	}catch(UninitializedVariableException ex){
	thrown = true;
	}

Oracle	thrown == true

UTC-5.7.1	**Test Vars.sumFromStack(Character variable) : verifico il nuovo valore della variabile sia effettivamente uguale alla somma, e che venga rimosso l'elemento dallo stack. Inoltro controllo che non vengano lanciate le eccezioni.**
Test Items	Classe **Vars**, metodo **sumFromStack(Character variable)**

Input	boolean thrownStack = false;
	
	boolean thrownVars = false;
	
	Character variable = 'A';
	
	Stack s = new Stack();
	
	Vars instance = new Vars(s);
	
	ComplexNumber n1 = new ComplexNumber(new BigDecimal(-1.5), new BigDecimal(2.3));
	
	ComplexNumber n2 = new ComplexNumber(new BigDecimal(2.5), new BigDecimal(-4.6));

	instance.setValueOf(variable, n1);
	
	s.push(n2);
	
	int before = s.getStack().size();
	
	try{
	  instance.sumFromStack(variable);
	}catch(UninitializedVariableException ex){
	  thrownVars = true;
	}catch(InvalidOperandsException ex){
	  thrownStack = true;
	}

Oracle	thrownVars == false && thrownStack == false && BinaryCanonicOperations.sum(n1,n2).equals(instance.getValueOf(variable)) && (before -1) == s.getStack().size()

UTC-5.7.2	**Test Vars.sumFromStack(Character variable) : verifico che nel caso di stack vuoto non venga modificato l'attuale valore della variabile e che venga lanciata la giusta eccezione**
Test Items	Classe **Vars**, metodo **sumFromStack(Character variable)**

Input	boolean thrownStack = false;
	
	boolean thrownVars = false;
	
	Character variable = 'A';
	
	Stack s = new Stack();
	
	Vars instance = new Vars(s);
	
	ComplexNumber n = new ComplexNumber(new BigDecimal(-1.5),
	new BigDecimal(2.3));
	
	instance.setValueOf(variable,n);
	
	int stackSize = s.getStack().size();
	ComplexNumber before = instance.getValueOf(variable);
	
	try{
	  instance.sumFromStack(variable);
	}catch(UninitializedVariableException ex){

	thrownVars = true;
	}catch(InvalidOperandsException ex){
	thrownStack = true;
	}

Oracle	thrownVars == false && thrownStack == true && before.equals(instance.getvalueOf(variable)) && stackSize == s.getStack().size()

UTC-5.7.3	**Test Vars.sumFromStack(Character variable) : verifico che nel caso di variabile non inizializzata, non venga modificato lo stack e che venga lanciata la giusta eccezione**
Test Items	Classe **Vars**, metodo **sumFromStack(Character variable)**

Input	boolean thrownStack = false;
	
	boolean thrownVars = false;
	
	Character variable = 'A';
	
	Stack s = new Stack();
	
	Vars instance = new Vars(s);
	
	ComplexNumber n = new ComplexNumber(new BigDecimal(-1.5),
	new BigDecimal(2.3));
	
	s.push(n);
	
	int stackSize = s.getStack().size();
	
	ComplexNumber before = instance.getValueOf(variable);
	
	try{
	instance.sumFromStack(variable);
	}catch(UninitializedVariableException ex){
	thrownVars = true;
	}catch(InvalidOperandsException ex){
	thrownStack = true;
	}

Oracle	thrownVars == true && thrownStack == false && before.equals(instance.getvalueOf(variable)) && stackSize == s.getStack().size()

UTC-5.7.4	**Test Vars.sumFromStack(Character variable) : verifico che nel caso di variabile non inizializzata e stack vuoto, non vengano modificate le strutture dati**
Test Items	Classe **Vars**, metodo **sumFromStack(Character variable)**

Input	boolean thrownStack = false;
	
	boolean thrownVars = false;
	
	Character variable = 'A';
	
	Stack s = new Stack();
	
	Vars instance = new Vars(s);
	
	int stackSize = s.getStack().size();
	
	ComplexNumber before = instance.getValueOf(variable);
	
	try{
	instance.sumFromStack(variable);
	}catch(UninitializedVariableException ex){
	thrownVars = true;
	}catch(InvalidOperandsException ex){
	thrownStack = true;
	}

Oracle	thrownVars == true && thrownStack == false && before.equals(instance.getvalueOf(variable)) && stackSize == s.getStack().size()

UTC-5.8.1	**Test Vars.subFromStack(Character variable) : verifico il nuovo valore della variabile sia effettivamente uguale alla differenza, e che venga rimosso l'elemento dallo stack. Inoltro controllo che non vengano lanciate le eccezioni.**
Test Items	Classe **Vars**, metodo **subFromStack(Character variable)**

Input	boolean thrownStack = false;
	
	boolean thrownVars = false;
	
	Character variable = 'A';
	
	Stack s = new Stack();
	
	Vars instance = new Vars(s);
	
	ComplexNumber n1 = new ComplexNumber(new BigDecimal(-1.5), new BigDecimal(2.3));
	
	ComplexNumber n2 = new ComplexNumber(new BigDecimal(2.5), new BigDecimal(-4.6));
	
	instance.setValueOf(variable, n1);
	
	s.push(n2);
	
	int before = s.getStack().size();
	
	try{
	instance.subFromStack(variable);
	}catch(UninitializedVariableException ex){
	thrownVars = true;
	}catch(InvalidOperandsException ex){
	thrownStack = true;
	}

Oracle	thrownVars == false && thrownStack == false && BinaryCanonicOperations.sub(n1,n2).equals(instance.getValueOf(variable)) && (before -1) == s.getStack().size()

UTC-5.8.2	**Test Vars.subFromStack(Character variable) : verifico che nel caso di stack vuoto non venga modificato l'attuale valore della variabile e che venga lanciata la giusta eccezione**
Test Items	Classe **Vars**, metodo **subFromStack(Character variable)**

Input	boolean thrownStack = false;
	
	boolean thrownVars = false;
	
	Character variable = 'A';
	
	Stack s = new Stack();
	
	Vars instance = new Vars(s);
	
	ComplexNumber n = new ComplexNumber(new BigDecimal(-1.5),
	new BigDecimal(2.3));
	
	instance.setValueOf(variable,n);
	
	int stackSize = s.getStack().size();
	
	ComplexNumber before = instance.getValueOf(variable);
	
	try{
	  instance.subFromStack(variable);
	}catch(UninitializedVariableException ex){
	  thrownVars = true;
	}catch(InvalidOperandsException ex){
	  thrownStack = true;
	}

Oracle	thrownVars == false && thrownStack == true && before.equals(instance.getvalueOf(variable)) && stackSize == s.getStack().size()

UTC-5.8.3	**Test Vars.subFromStack(Character variable) : verifico che nel caso di variabile non inizializzata, non venga modificato lo stack e che venga lanciata la giusta eccezione**
Test Items	Classe **Vars**, metodo **sumFromStack(Character variable)**

Input	boolean thrownStack = false;
	
	boolean thrownVars = false;
	
	Character variable = 'A';
	
	Stack s = new Stack();
	
	Vars instance = new Vars(s);
	
	ComplexNumber n = new ComplexNumber(new BigDecimal(-1.5), new BigDecimal(2.3));
	
	s.push(n);
	
	int stackSize = s.getStack().size();
	
	ComplexNumber before = instance.getValueOf(variable);
	
	try{
	instance.subFromStack(variable);
	}catch(UninitializedVariableException ex){
	thrownVars = true;
	}catch(InvalidOperandsException ex){
	thrownStack = true;
	}

Oracle	thrownVars == true && thrownStack == false && before.equals(instance.getvalueOf(variable)) && stackSize == s.getStack().size()

UTC-5.8.4	**Test Vars.subFromStack(Character variable) : verifico che nel caso di variabile non inizializzata e stack vuoto, non vengano modificate le strutture dati**
Test Items	Classe **Vars**, metodo **subFromStack(Character variable)**

Input	boolean thrownStack = false;
	
	boolean thrownVars = false;
	
	Character variable = 'A';
	
	Stack s = new Stack();
	
	Vars instance = new Vars(s);
	
	int stackSize = s.getStack().size();
	
	ComplexNumber before = instance.getValueOf(variable);
	
	try{
	instance.subFromStack(variable);
	}catch(UninitializedVariableException ex){
	thrownVars = true;
	}catch(InvalidOperandsException ex){
	thrownStack = true;
	}

Oracle	thrownVars == true && thrownStack == false && before.equals(instance.getvalueOf(variable)) && stackSize == s.getStack().size()

UTC-5.9.1	**Test Vars.toStringArrayList() : verifico che le ArrayList siano uguali (Caso : entrambe vuote)**
Test Items	Classe **Vars**, metodo **toStringArrayList()**
Input	Vars instance = new Vars(new Stack()); ArrayList<String> expResult = new ArrayList(); ArrayList<String> result = instance.toStringArrayList();
Oracle	expResult.equals(result) == true

UTC-5.9.2	**Test Vars.toStringArrayList() : verifico che le ArrayList siano uguali (Caso : entrambe con un elemento)**
Test Items	Classe **Vars**, metodo **toStringArrayList()**

Input	Vars instance = new Vars(new Stack());
	
	Character variable = 'A';
	
	ComplexNumber n = new ComplexNumber(new BigDecimal(-1.5),
	new BigDecimal(2.3));
	
	ArrayList<String> expResult = new ArrayList();
	
	String e = variable + " = " + n.toString();
	
	expResult.add(e);
	
	expResult.sort(null);
	
	instance.setValueOf(variable, n);
	
	ArrayList<String> result = instance.toStringArrayList();

Oracle	expResult.equals(result) == true

UTC-5.9.3	**Test Vars.toStringArrayList() : verifico che le ArrayList siano uguali (Caso : entrambe con due elementi e non inserite in ordine alfabetico)**
Test Items	Classe **Vars**, metodo **toStringArrayList()**

Input	Vars instance = new Vars(new Stack());
	
	ArrayList<String> expResult = new ArrayList();
	
	Character variable = 'D';
	
	ComplexNumber n = new ComplexNumber(new BigDecimal(-1.5),
	new BigDecimal(2.3));
	
	instance.setValueOf(variable, n);
	
	String e = variable + " = " + n.toString();
	
	expResult.add(e);

	variable = 'A';
	n = new ComplexNumber(new BigDecimal(8.985),new BigDecimal(-23.4582));
	instance.setValueOf(variable, n);
	e = variable + " = " + n.toString();
	expResult.add(e);
	expResult.sort(null);
	ArrayList<String> result = instance.toStringArrayList();

Oracle	expResult.equals(result) == true

6- Classe ScientificCalculator

UTC-6.1.1	**Test ScientificCalculator.execute(int input) : con input=1**
Test Items	Classe **ScientificCalculator**, metodo **execute()**

Input	s.push(new ComplexNumber(new BigDecimal("12.528"), new BigDecimal("74")));
	s.push(new ComplexNumber(new BigDecimal("-23"), new BigDecimal("-59")));
	sc.execute(1); //sc.sum()

Oracle	s.pop().equals(new ComplexNumber(new BigDecimal("-10.472"), new BigDecimal("15")) == true

Lo stesso viene fatto per gli input=2, 3, 4 (UTC-6.1.2, UTC-6.1.3, UTC-6.1.4 richiamano metodi di BinaryCanonicOperations).

UTC-6.1.5	**Test ScientificCalculator.execute(int input) : con input=5**
Test Items	Classe **ScientificCalculator**, metodo **execute()**

Input	s.push(new ComplexNumber(new BigDecimal("12.528"), new BigDecimal("74")));
	s.push(new ComplexNumber(new BigDecimal("-23"), new BigDecimal("-59")));
	sc.execute(5); //sc.changeSign()

Oracle	s.pop().equals(new ComplexNumber(new BigDecimal("23"), new BigDecimal("59")) == true

Lo stesso viene fatto anche per input=6 (UTC-6.1.6 richiamano metodi di UnaryCanonicOperations).

UTC-6.1.7	**Test ScientificCalculator.execute(int input) : con input=11**
Test Items	Classe **ScientificCalculator**, metodo **execute()**

Input	s.push(new ComplexNumber(new BigDecimal("12.528"), new BigDecimal("74")));
	
	s.push(new ComplexNumber(new BigDecimal("-23"), new BigDecimal("-59")));
	
	sc.execute(11); //sc.Stack.clear()

Oracle	s.getStack().isEmpty() == true

Lo stesso viene fatto per input=12 … 15 (UTC-6.1.8 – UTC-6.1.11 richiamano metodi di Stack).

UTC-6.2.1	**Test ScientificCalculator.executeOnVariable(Character variable, Character operation) : con input =('A', '>')**
Test Items	Classe **ScientificCalculator**, metodo **executeOnVariable(Character variable, Character operation)**

Input	ScientificCalculator sc = new ScientificCalculator();
	
	Stack s = sc.getStack();
	
	Vars v = sc.getVars();
	
	s.push(new ComplexNumber(new BigDecimal("12.528"), new BigDecimal("74")));
	
	sc.executeOnVariable('A', '>');

Oracle	s.getStack().isEmpty() == true && v.getVariables().containsKey('A') == true && v.getVariables().containsValue(new ComplexNumber(new BigDecimal("12.528"), new BigDecimal("74")))

Lo stesso viene fatto per input='<', '+', '-' (UTC-6.2.2 – UTC-6.2.4 richiamano metodi di Vars).

UTC-6.3.1	**Test ScientificCalculator.sum() : Nel caso in cui lo stack sia vuoto (caso di errore)**
Test Items	Classe **ScientificCalculator**, metodo **sum()**

Input	ScientificCalculator sc = new ScientificCalculator();
	
	int test=0;
	
	try{
	  sc.sum();
	} catch (InvalidOperandsException exc) {
	  test=1;
	}

Oracle	Test == 1

UTC-6.3.2	**Test ScientificCalculator.sum() : Nel caso in cui nello stack ci sia un solo elemento (caso di errore)**
Test Items	Classe **ScientificCalculator**, metodo **sum()**

Input	ScientificCalculator sc = new ScientificCalculator();
	
	Stack s = sc.getStack();
	
	s.push(new ComplexNumber(new BigDecimal("12.528"), new BigDecimal("74")));
	
	int test=0;
	
	try{
	  sc.sum();
	} catch (InvalidOperandsException exc) {
	  test=1;
	}

Oracle	Test == 1

UTC-6.3.3	**Test ScientificCalculator.sum() : Nel caso in cui nello stack ci siano due elementi**
Test Items	Classe **ScientificCalculator**, metodo **sum()**

Input	ScientificCalculator sc = new ScientificCalculator();
	
	Stack s = sc.getStack();
	
	s.push(new ComplexNumber(new BigDecimal("12.528"), new BigDecimal("74")));
	
	s.push(new ComplexNumber(new BigDecimal("-23"), new BigDecimal("-59")));
	
	sc.sum();

Oracle	s.pop().equals(new ComplexNumber(new BigDecimal("-10.472"), new BigDecimal("15")) == true

UTC-6.4.1	**Test ScientificCalculator.sub() : Nel caso in cui lo stack sia vuoto (caso di errore)**
Test Items	Classe **ScientificCalculator**, metodo **sub()**

Input	ScientificCalculator sc = new ScientificCalculator();
	
	int test=0;
	
	try{
	  sc.sub();
	} catch (InvalidOperandsException exc) {
	  test=1;
	}

Oracle	Test == 1

UTC-6.4.2	**Test ScientificCalculator.sub() : Nel caso in cui nello stack ci sia un solo elemento (caso di errore)**
Test Items	Classe **ScientificCalculator**, metodo **sub()**

Input	ScientificCalculator sc = new ScientificCalculator();
	
	Stack s = sc.getStack();
	
	s.push(new ComplexNumber(new BigDecimal("12.528"), new BigDecimal("74")));
	
	int test=0;
	
	try{
	  sc.sub();
	} catch (InvalidOperandsException exc) {
	  test=1;
	}

Oracle	Test == 1

UTC-6.4.3	**Test ScientificCalculator.sub() : Nel caso in cui nello stack ci siano due elementi**
Test Items	Classe **ScientificCalculator**, metodo **sub()**

Input	ScientificCalculator sc = new ScientificCalculator();
	
	Stack s = sc.getStack();
	
	s.push(new ComplexNumber(new BigDecimal("12.528"), new BigDecimal("74")));
	
	s.push(new ComplexNumber(new BigDecimal("-23"), new BigDecimal("-59")));
	
	sc.sub();

Oracle	s.pop().equals(new ComplexNumber(new BigDecimal("35.528"), new BigDecimal("133")) == true

UTC-6.5.1	**Test ScientificCalculator.multiply() : Nel caso in cui lo stack sia vuoto (caso di errore)**
Test Items	Classe **ScientificCalculator**, metodo **multiply()**

Input	ScientificCalculator sc = new ScientificCalculator();
	
	int test=0;
	
	try{
	  sc.multiply();
	} catch (InvalidOperandsException exc) {
	  test=1;
	}

Oracle	Test == 1

UTC-6.5.2	**Test ScientificCalculator.multiply() : Nel caso in cui nello stack ci sia un solo elemento (caso di errore)**
Test Items	Classe **ScientificCalculator**, metodo **multiply()**

Input	ScientificCalculator sc = new ScientificCalculator();
	
	Stack s = sc.getStack();
	
	s.push(new ComplexNumber(new BigDecimal("12.528"), new BigDecimal("74")));
	
	int test=0;
	
	try{
	  sc.multiply();
	} catch (InvalidOperandsException exc) {
	  test=1;
	}

Oracle	Test == 1

UTC-6.5.3	**Test ScientificCalculator.multiply() :** **Nel caso in cui nello stack ci siano due elementi**
Test Items	Classe **ScientificCalculator**, metodo **multiply()**

Input	ScientificCalculator sc = new ScientificCalculator();
	
	Stack s = sc.getStack();
	
	s.push(new ComplexNumber(new BigDecimal("12.528"), new BigDecimal("74")));
	
	s.push(new ComplexNumber(new BigDecimal("-23"), new BigDecimal("-59")));
	
	sc.multiply();

Oracle	s.pop().equals(new ComplexNumber(new BigDecimal("4077.856"), new BigDecimal("-2441.152")) == true

UTC-6.6.1	**Test ScientificCalculator.divide() :** **Nel caso in cui lo stack sia vuoto (caso di errore)**
Test Items	Classe **ScientificCalculator**, metodo **divide()**

Input	ScientificCalculator sc = new ScientificCalculator();
	
	int test=0;
	
	try{
	  sc.divide();
	} catch (InvalidOperandsException exc) {
	  test=1;
	}

Oracle	Test == 1

UTC-6.6.2	**Test ScientificCalculator.divide() : Nel caso in cui nello stack ci sia un solo elemento (caso di errore)**
Test Items	Classe **ScientificCalculator**, metodo **divide()**

Input	ScientificCalculator sc = new ScientificCalculator();
	
	Stack s = sc.getStack();
	
	s.push(new ComplexNumber(new BigDecimal("12.528"), new BigDecimal("74")));
	
	int test=0;
	
	try{
	  sc.divide();
	} catch (InvalidOperandsException exc) {
	  test=1;
	}

Oracle	Test == 1

UTC-6.6.3	**Test ScientificCalculator.divide() : Nel caso in cui nello stack ci siano due elementi**
Test Items	Classe **ScientificCalculator**, metodo **divide()**

Input	ScientificCalculator sc = new ScientificCalculator();
	
	Stack s = sc.getStack();
	
	s.push(new ComplexNumber(new BigDecimal("12.528"), new BigDecimal("74")));
	
	s.push(new ComplexNumber(new BigDecimal("-23"), new BigDecimal("-59")));
	
	sc.divide();

Oracle	s.pop().equals(new ComplexNumber(new BigDecimal("-1.161"), new BigDecimal("-0.240")) == true

UTC-6.7.1	**Test ScientificCalculator.squareRoot() : Nel caso in cui lo stack sia vuoto (caso di errore)**
Test Items	Classe **ScientificCalculator**, metodo **squareRoot()**

Input	ScientificCalculator sc = new ScientificCalculator();
	
	int test=0;
	
	try{
	sc.squareRoot();
	} catch (InvalidOperandsException exc) {
	test=1;
	}

Oracle	Test == 1

UTC-6.7.2	**Test ScientificCalculator. squareRoot() : Nel caso in cui nello stack ci sia un solo elemento (caso di errore)**
Test Items	Classe **ScientificCalculator**, metodo **squareRoot()**

Input	ScientificCalculator sc = new ScientificCalculator();
	
	Stack s = sc.getStack();
	
	s.push(new ComplexNumber(new BigDecimal("12.528"),
	new BigDecimal("74")));
	
	sc.squareRoot();

Oracle	s.pop().equals(new ComplexNumber(new BigDecimal("-6.617"), new BigDecimal("-5.591")) == true s.pop().equals(new ComplexNumber(new BigDecimal("6.617"), new BigDecimal("5.591")) == true

UTC-6.8.1	**Test ScientificCalculator.changeSign() :** **Nel caso in cui lo stack sia vuoto (caso di errore)**
Test Items	Classe **ScientificCalculator**, metodo **changeSign()**

Input	ScientificCalculator sc = new ScientificCalculator();
	
	int test=0;
	
	try{
	  sc.changeSign();
	} catch (InvalidOperandsException exc) {
	  test=1;
	}

Oracle	Test == 1

UTC-6.8.2	**Test ScientificCalculator.changeSign() :** **Nel caso in cui nello stack ci sia un solo elemento (caso di errore)**
Test Items	Classe **ScientificCalculator**, metodo **changeSign()**

Input	ScientificCalculator sc = new ScientificCalculator();
	
	Stack s = sc.getStack();
	
	s.push(new ComplexNumber(new BigDecimal("12.528"),
	new BigDecimal("74")));
	
	sc.changeSign();

Oracle	s.pop().equals(new ComplexNumber(new BigDecimal("-12.528"), new BigDecimal("-74")) == true

Il metodo insertComplexNumber della classe ScientificCalculator utilizza il metodo privato formatComplexNumber(), perciò i test seguenti verificheranno il funzionamento di entrambi i metodi.

UTC-6.9.1	**Test ScientificCalculator.insertComplexNumber(String input) : Con input complesso decimale positivo**
Test Items	Classe **ScientificCalculator**, metodo **insertComplexNumber(String input)**
Input	ScientificCalculator sc = new ScientificCalculator(); Stack s = sc.getStack(); sc.insertComplexNumber("31.6238+22.729j");
Oracle	s.pop().equals(new ComplexNumber(new BigDecimal("31.6238"), new BigDecimal("22.729")) == true

UTC-6.9.2	**Test ScientificCalculator.insertComplexNumber(String input) : Con input decimale negativo**
Test Items	Classe **ScientificCalculator**, metodo **insertComplexNumber(String input)**
Input	ScientificCalculator sc = new ScientificCalculator(); Stack s = sc.getStack(); sc.insertComplexNumber("-31.6238-22.729j");
Oracle	s.pop().equals(new ComplexNumber(new BigDecimal("-31.6238"), new BigDecimal("-22.729")) == true

UTC-6.9.3	**Test ScientificCalculator.insertComplexNumber(String input) : Con input decimale segni discordi**
Test Items	Classe **ScientificCalculator**, metodo **insertComplexNumber(String input)**
Input	ScientificCalculator sc = new ScientificCalculator(); Stack s = sc.getStack(); sc.insertComplexNumber("+31.6238-22.729j");
Oracle	s.pop().equals(new ComplexNumber(new BigDecimal("31.6238"), new BigDecimal("-22.729")) == true

UTC-6.9.4	**Test ScientificCalculator.insertComplexNumber(String input) : Con input intero negativo**
Test Items	Classe **ScientificCalculator**, metodo **insertComplexNumber(String input)**
Input	ScientificCalculator sc = new ScientificCalculator(); Stack s = sc.getStack(); sc.insertComplexNumber("-23-59j");
Oracle	s.pop().equals(new ComplexNumber(new BigDecimal("-23"), new BigDecimal("-59")) == true

UTC-6.9.5	**Test ScientificCalculator.insertComplexNumber(String input) : Con input intero positivo**
Test Items	Classe **ScientificCalculator**, metodo **insertComplexNumber(String input)**
Input	ScientificCalculator sc = new ScientificCalculator(); Stack s = sc.getStack(); sc.insertComplexNumber("+23+59j");
Oracle	s.pop().equals(new ComplexNumber(new BigDecimal("23"), new BigDecimal("59")) == true

UTC-6.9.6	**Test ScientificCalculator.insertComplexNumber(String input) : Con input intero segni discordi**
Test Items	Classe **ScientificCalculator**, metodo **insertComplexNumber(String input)**
Input	ScientificCalculator sc = new ScientificCalculator(); Stack s = sc.getStack(); sc.insertComplexNumber("-23+59j");
Oracle	s.pop().equals(new ComplexNumber(new BigDecimal("-23"), new BigDecimal("59")) == true

UTC-6.9.7	**Test ScientificCalculator.insertComplexNumber(String input) : Con input reale decimale positivo**
Test Items	Classe **ScientificCalculator**, metodo **insertComplexNumber(String input)**
Input	ScientificCalculator sc = new ScientificCalculator(); Stack s = sc.getStack(); sc.insertComplexNumber("23.618");
Oracle	s.pop().equals(new ComplexNumber(new BigDecimal("23.618"), new BigDecimal("0.000")) == true

UTC-6.9.8	**Test ScientificCalculator.insertComplexNumber(String input) : Con input reale decimale negativo**
Test Items	Classe **ScientificCalculator**, metodo **insertComplexNumber(String input)**
Input	ScientificCalculator sc = new ScientificCalculator(); Stack s = sc.getStack(); sc.insertComplexNumber("-23.618");
Oracle	s.pop().equals(new ComplexNumber(new BigDecimal("-23.618"), new BigDecimal("0.000")) == true

UTC-6.9.9	**Test ScientificCalculator.insertComplexNumber(String input) : Con input reale intero positivo**
Test Items	Classe **ScientificCalculator**, metodo **insertComplexNumber(String input)**
Input	ScientificCalculator sc = new ScientificCalculator(); Stack s = sc.getStack(); sc.insertComplexNumber("176");
Oracle	s.pop().equals(new ComplexNumber(new BigDecimal("176 "), new BigDecimal("0.000")) == true

UTC-6.9.10	**Test ScientificCalculator.insertComplexNumber(String input) : Con input reale intero negativo**
Test Items	Classe **ScientificCalculator**, metodo **insertComplexNumber(String input)**
Input	ScientificCalculator sc = new ScientificCalculator(); Stack s = sc.getStack(); sc.insertComplexNumber("-176");
Oracle	s.pop().equals(new ComplexNumber(new BigDecimal("-176 "), new BigDecimal("0.000")) == true

UTC-6.9.11	**Test ScientificCalculator.insertComplexNumber(String input) : Con input immaginario puro decimale positivo**
Test Items	Classe **ScientificCalculator**, metodo **insertComplexNumber(String input)**
Input	ScientificCalculator sc = new ScientificCalculator(); Stack s = sc.getStack(); sc.insertComplexNumber("23.618j");
Oracle	s.pop().equals(new ComplexNumber(new BigDecimal("0.000"), new BigDecimal("23.618")) == true

UTC-6.9.12	**Test ScientificCalculator.insertComplexNumber(String input) : Con input immaginario puro decimale negativo**
Test Items	Classe **ScientificCalculator**, metodo **insertComplexNumber(String input)**
Input	ScientificCalculator sc = new ScientificCalculator(); Stack s = sc.getStack(); sc.insertComplexNumber("-23.618j");
Oracle	s.pop().equals(new ComplexNumber(new BigDecimal("0.000"), new BigDecimal("-23.618")) == true

UTC-6.9.13	**Test ScientificCalculator.insertComplexNumber(String input) : Con input immaginario puro intero positivo**
Test Items	Classe **ScientificCalculator**, metodo **insertComplexNumber(String input)**
Input	ScientificCalculator sc = new ScientificCalculator(); Stack s = sc.getStack(); sc.insertComplexNumber("176j");
Oracle	s.pop().equals(new ComplexNumber(new BigDecimal("0.000"), new BigDecimal("176")) == true

UTC-6.9.14	**Test ScientificCalculator.insertComplexNumber(String input) : Con input immaginario puro intero negativo**
Test Items	Classe **ScientificCalculator**, metodo **insertComplexNumber(String input)**
Input	ScientificCalculator sc = new ScientificCalculator(); Stack s = sc.getStack(); sc.insertComplexNumber("-176j");
Oracle	s.pop().equals(new ComplexNumber(new BigDecimal("0.000"), new BigDecimal("-176")) == true

UTC-6.10.1	**Test ScientificCalculator.getStack() : Test della validità del reference restituito dal metodo getStack()**
Test Items	Classe **ScientificCalculator**, metodo **getStack()**

Input	ScientificCalculator sc = new ScientificCalculator();
	Field field = sc.getClass().getDeclaredField("stack");
	field.setAccessible(true);
	Stack s = new Stack();
	field.set(sc, s);

Oracle	Test.equals(sc.getStack()) == true

UTC-6.11.1	**Test ScientificCalculator.getVars() :** **Test della validità del reference restituito dal metodo getVars()**
Test Items	Classe **ScientificCalculator**, metodo **getVars()**

Input	ScientificCalculator sc = new ScientificCalculator();
	
	Field field = sc.getClass().getDeclaredField("vars");
	
	field.setAccessible(true);
	
	Vars test = new Vars(new Stack());
	
	field.set(sc, test);

Oracle	test.equals(sc.getVars()) == true

7- Class Checker

UTC-7.1.1	**Test Checker.isANumber(String input) : Numero complesso con parte reale e parte immaginaria positiva**
Test Items	Classe **Checker**, metodo **isANumber(String input)**

Input	String input = "31.6238+22.729j";
	
	boolean expResult = true;
	boolean result = Checker.isANumber(input);

Oracle	expResult == result

UTC-7.1.2	**Test Checker.isANumber(String input) : Numero complesso con parte negativa e parte immaginaria negativa**
Test Items	Classe **Checker**, metodo **isANumber(String input)**

Input	String input = "-31.6238-22.729j";
	
	boolean expResult = true;
	boolean result = Checker.isANumber(input);

Oracle	expResult == result

UTC-7.1.3	**Test Checker.isANumber(String input) : Numero intero reale negativo**
Test Items	Classe **Checker**, metodo **isANumber(String input)**

Input	String input = "-176";
	
	boolean expResult = true;
	boolean result = Checker.isANumber(input);

Oracle	expResult == result

UTC-7.1.4	**Test Checker.isANumber(String input) : Numero reale negativo con molte cifre decimali**
Test Items	Classe **Checker**, metodo **isANumber(String input)**

Input	String input = "-23.816215";
	
	boolean expResult = true;
	boolean result = Checker.isANumber(input);

Oracle	expResult == result

UTC-7.1.5	**Test Checker.isANumber(String input) : Numero puramente immaginario positivo**
Test Items	Classe **Checker**, metodo **isANumber(String input)**

Input	String input = "176j";
	
	boolean expResult = true;
	boolean result = Checker.isANumber(input);

Oracle	expResult == result

UTC-7.1.6	**Test Checker.isANumber(String input) : Numero puramente immaginario negativo con molte cifre decimali**
Test Items	Classe **Checker**, metodo **isANumber(String input)**

Input	String input = "-23.816215j";
	
	boolean expResult = true;
	boolean result = Checker.isANumber(input);

Oracle	expResult == result

UTC-7.1.7	**Test Checker.isANumber(String input) : Inserimento di 0**
Test Items	Classe **Checker**, metodo **isANumber(String input)**

Input	String input = "0";
	
	boolean expResult = true;
	boolean result = Checker.isANumber(input);

Oracle	expResult == result

UTC-7.1.8	**Test Checker.isANumber(String input) : Inserimento di 0j**
Test Items	Classe **Checker**, metodo **isANumber(String input)**

Input	String input = "0j";
	
	boolean expResult = true;
	boolean result = Checker.isANumber(input);

Oracle	expResult == result

UTC-7.1.9	**Test Checker.isANumber(String input) : Inserimento di 1j**
Test Items	Classe **Checker**, metodo **isANumber(String input)**

Input	String input = "1j";
	
	boolean expResult = true;
	boolean result = Checker.isANumber(input);

Oracle	expResult == result

UTC-7.1.10	**Test Checker.isANumber(String input) : Inserimento di 0.0+0.0j**
Test Items	Classe **Checker**, metodo **isANumber(String input)**

Input	String input = "0.0+0.0j";
	
	boolean expResult = true;
	boolean result = Checker.isANumber(input);

Oracle	expResult == result

UTC-7.1.11	**Test Checker.isANumber(String input) : Inserimento di 1111111111111111111111111111111111111111111111111**
Test Items	Classe **Checker**, metodo **isANumber(String input)**

Input	String input = "1111111111111111111111111111111111111111111111111";
	
	boolean expResult = true;
	boolean result = Checker.isANumber(input);

Oracle	expResult == result

UTC-7.1.12	**Test Checker.isANumber(String input) : Inserimento dell'operazione "-"**
Test Items	Classe **Checker**, metodo **isANumber(String input)**

Input	String input = "-";
	
	boolean expResult = false;
	boolean result = Checker.isANumber(input);

Oracle	expResult == result

UTC-7.1.13	**Test Checker.isANumber(String input) : Inserimento dell'operazione "DUP"**
Test Items	Classe **Checker**, metodo **isANumber(String input)**

Input	String input = "DUP";
	
	boolean expResult = false;
	boolean result = Checker.isANumber(input);

Oracle	expResult == result

UTC-7.1.14	**Test Checker.isANumber(String input) : Inserimento dell'operazione "OVR"**
Test Items	Classe **Checker**, metodo **isANumber(String input)**

Input	String input = "OVR";
	
	boolean expResult = false;
	boolean result = Checker.isANumber(input);

Oracle	expResult == result

UTC-7.1.15	**Test Checker.isANumber(String input) : Inserimento di stringa errata**
Test Items	Classe **Checker**, metodo **isANumber(String input)**

Input	String input = "ciao";
	
	boolean expResult = false;
	boolean result = Checker.isANumber(input);

Oracle	expResult == result

UTC-7.1.16	**Test Checker.isANumber(String input) : Inserimento di una stringa contenente un numero immaginario**
Test Items	Classe **Checker**, metodo **isANumber(String input)**

Input	String input = "test27j";
	
	boolean expResult = false;
	boolean result = Checker.isANumber(input);

Oracle	expResult == result

UTC-7.1.17	**Test Checker.isANumber(String input) : Inserimento di un numero con spazi tra il segno di separazione**
Test Items	Classe **Checker**, metodo **isANumber(String input)**

Input	String input = " 253.265 - 251j";
	
	boolean expResult = true;
	boolean result = Checker.isANumber(input);

Oracle	expResult == result

UTC-7.2.1	**Test Checker.checkOperation(String input) : Inserimento dell'operatore "+"**
Test Items	Classe **Checker**, metodo **checkOperation (String input)**

Input	String input = "+";
	
	int expResult =1;
	int result = Checker.checkOperation(input);

Oracle	expResult == result

UTC-7.2.2	**Test Checker.checkOperation(String input) : Inserimento dell'operatore "-"**
Test Items	Classe **Checker**, metodo **checkOperation (String input)**

Input	String input = "-";
	
	int expResult =2;
	int result = Checker.checkOperation(input);

Oracle	expResult == result

UTC-7.2.3	**Test Checker.checkOperation(String input) : Inserimento dell'operatore "×"**
Test Items	Classe **Checker**, metodo **checkOperation (String input)**

Input	String input = "×";
	
	int expResult =3;
	int result = Checker.checkOperation(input);

Oracle	expResult == result

UTC-7.2.4	**Test Checker.checkOperation(String input) : Inserimento dell'operatore "÷"**
Test Items	Classe **Checker**, metodo **checkOperation (String input)**

Input	String input = "÷";
	
	int expResult =4;
	int result = Checker.checkOperation(input);

Oracle	expResult == result

UTC-7.2.5	**Test Checker.checkOperation(String input) : Inserimento dell'operatore "±"**
Test Items	Classe **Checker**, metodo **checkOperation (String input)**

Input	String input = "±";
	
	int expResult =5;
	int result = Checker.checkOperation(input);

Oracle	expResult == result

UTC-7.2.6	**Test Checker.checkOperation(String input) : Inserimento dell'operatore "√"**
Test Items	Classe **Checker**, metodo **checkOperation (String input)**

Input	String input = "√";
	
	int expResult =6;
	int result = Checker.checkOperation(input);

Oracle	expResult == result

UTC-7.2.7	**Test Checker.checkOperation(String input) : Inserimento dell'operatore "CLR"**
Test Items	Classe **Checker**, metodo **checkOperation (String input)**

Input	String input = "CLR";
	
	int expResult =11;
	int result = Checker.checkOperation(input);

Oracle	expResult == result

UTC-7.2.8	**Test Checker.checkOperation(String input) : Inserimento dell'operatore "DRP"**
Test Items	Classe **Checker**, metodo **checkOperation (String input)**

Input	String input = "DRP";
	
	int expResult =12;
	int result = Checker.checkOperation(input);

Oracle	expResult == result

UTC-7.2.9	**Test Checker.checkOperation(String input) : Inserimento dell'operatore "SWP"**
Test Items	Classe **Checker**, metodo **checkOperation (String input)**

Input	String input = "SWP";
	
	int expResult =13;
	int result = Checker.checkOperation(input);

Oracle	expResult == result

UTC-7.2.10	**Test Checker.checkOperation(String input) : Inserimento dell'operatore "DUP"**
Test Items	Classe **Checker**, metodo **checkOperation (String input)**

Input	String input = "DUP";
	
	int expResult =14;
	int result = Checker.checkOperation(input);

Oracle	expResult == result

UTC-7.2.11	**Test Checker.checkOperation(String input) : Inserimento dell'operatore "OVR"**
Test Items	Classe **Checker**, metodo **checkOperation (String input)**

Input	String input = "OVR";
	
	int expResult =15;
	int result = Checker.checkOperation(input);

Oracle	expResult == result

UTC-7.2.12	**Test Checker.checkOperation(String input) : Inserimento di una stringa**
Test Items	Classe **Checker**, metodo **checkOperation (String input)**

Input	String input = "più";
	
	int expResult =-1;
	int result = Checker.checkOperation(input);

Oracle	expResult == result

UTC-7.2.13	**Test Checker.checkOperation(String input) : Inserimento di un numero corretto**
Test Items	Classe **Checker**, metodo **checkOperation (String input)**

Input	String input = "2+3j";
	
	int expResult =-1;
	int result = Checker.checkOperation(input);

Oracle	expResult == result

UTC-7.3.1	**Test Checker.checkInput(String input) : Inserimento di un numero corretto**
Test Items	Classe **Checker**, metodo **checkInput (String input)**

Input	String input = "22.2541-3.1452j";
	
	boolean thrown = false;
	try {
	Checker.checkInput(input);
	} catch (InvalidInputException ex) {
	thrown = true;
	}

Oracle	thrown == false

UTC-7.3.2	**Test Checker.checkInput(String input) : Inserimento di un operazione corretta**
Test Items	Classe **Checker**, metodo **checkInput (String input)**

Input	String input = "OVR";
	
	boolean thrown = false;
	try {
	Checker.checkInput(input);
	} catch (InvalidInputException ex) {
	thrown = true;
	}

Oracle	thrown == false

UTC-7.3.3	**Test Checker.checkInput(String input) : Inserimento di un input completamente errato**
Test Items	Classe **Checker**, metodo **checkInput (String input)**

Input	String input = "test input errato";
	
	boolean thrown = false;
	try {
	Checker.checkInput(input);
	} catch (InvalidInputException ex) {
	thrown = true;
	}

Oracle	thrown == true

UTC-7.3.4	**Test Checker.checkInput(String input) : Inserimento di un input errato contenente simboli che potrebbero essere corretti**
Test Items	Classe **Checker**, metodo **checkInput (String input)**

Input	String input = " + +j";
	
	boolean thrown = false;
	try {
	Checker.checkInput(input);
	} catch (InvalidInputException ex) {
	thrown = true;
	}

Oracle	thrown == true

UTC-7.4.1	**Test Checker.isInitialized (Vars var,Character variable) : Variabile già inizializzata**
Test Items	Classe **Checker**, metodo **isInitialized (Vars var,Character variable)**

Input	s.push(new ComplexNumber(new BigDecimal("3"), new BigDecimal("8")));
	
	var.popFromStack('A');
	Character variable = 'A';
	
	boolean expResult = true;
	boolean result = Checker.isInitialized(var, variable);

Oracle	expResult == result

UTC-7.4.2	**Test Checker.isInitialized (Vars var,Character variable) : Variabile non presente**
Test Items	Classe **Checker**, metodo **isInitialized(Vars var,Character variable)**

Input	Character variable = 'A';
	
	boolean expResult = false;
	boolean result = Checker.isInitialized(var, variable);

Oracle	expResult == result

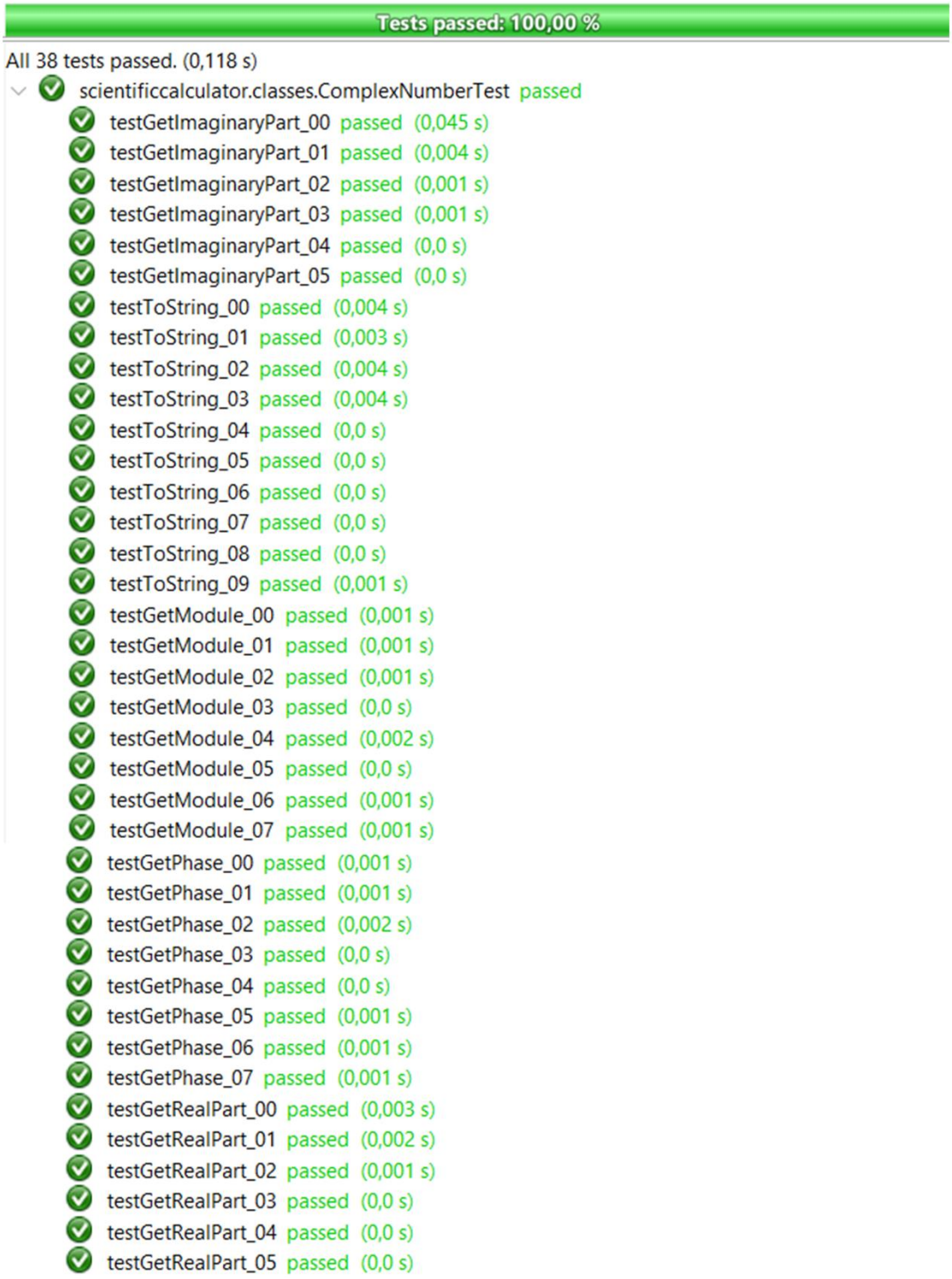
UTC-7.4.3	**Test Checker.isInitialized (Vars var,Character variable) : Variabile presente ma senza alcun valore**
Test Items	Classe **Checker**, metodo **isInitialized(Vars var,Character variable)**

Input	var.getVariables().put('A', null);
	
	Character variable = 'A';
	
	boolean expResult = false;
	boolean result = Checker.isInitialized(var, variable);

Oracle	expResult == result

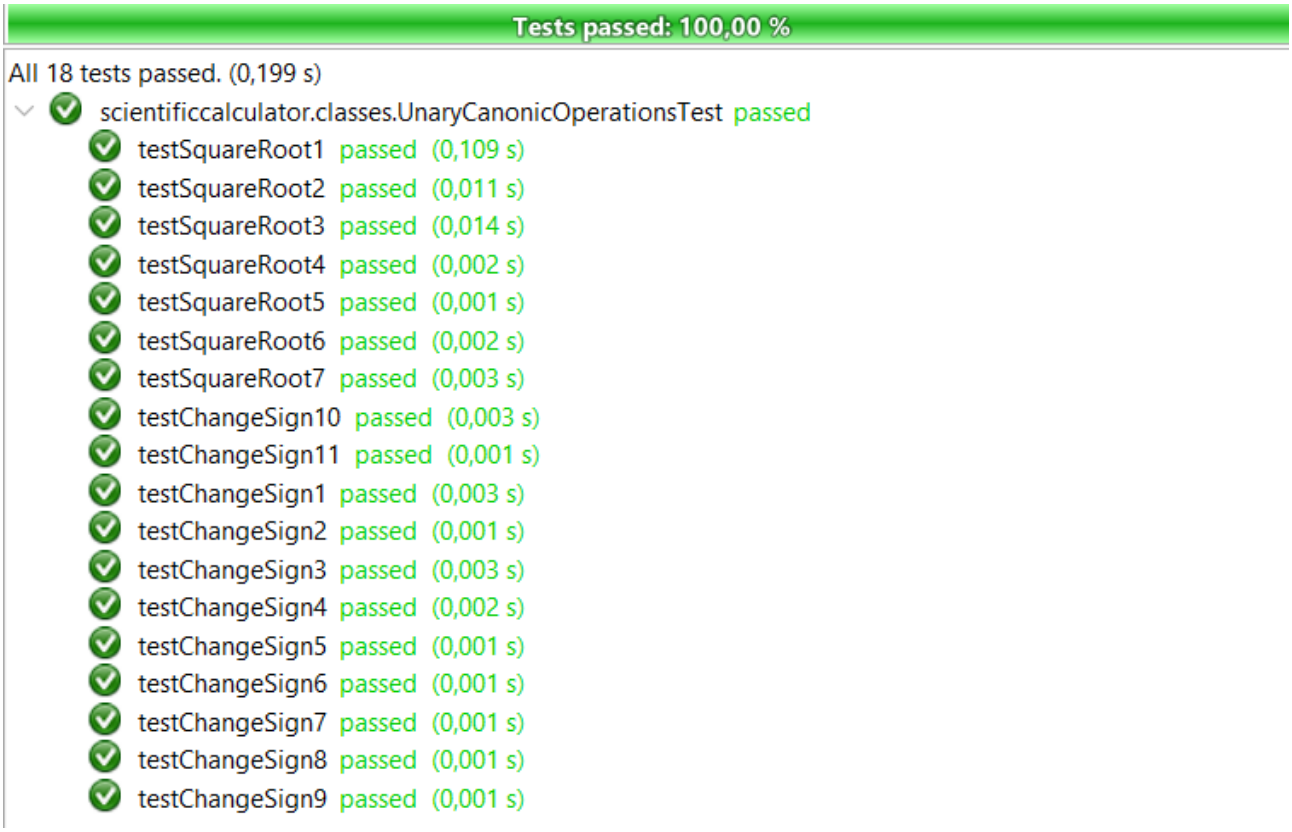
RISULTATI DEI TEST

1- Risultati test ComplexNumber

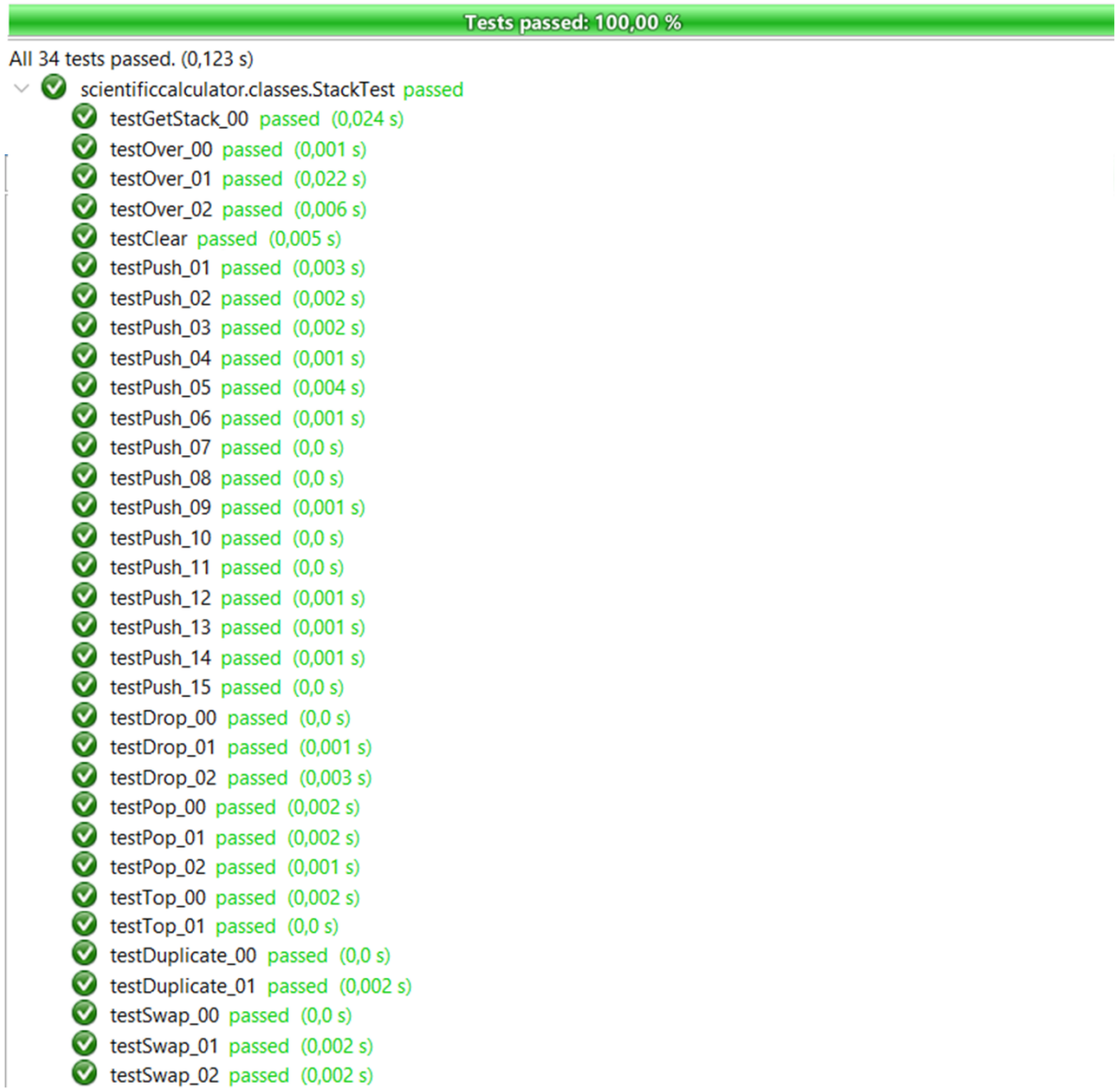


2- Risultati test BinaryCanonicOperations

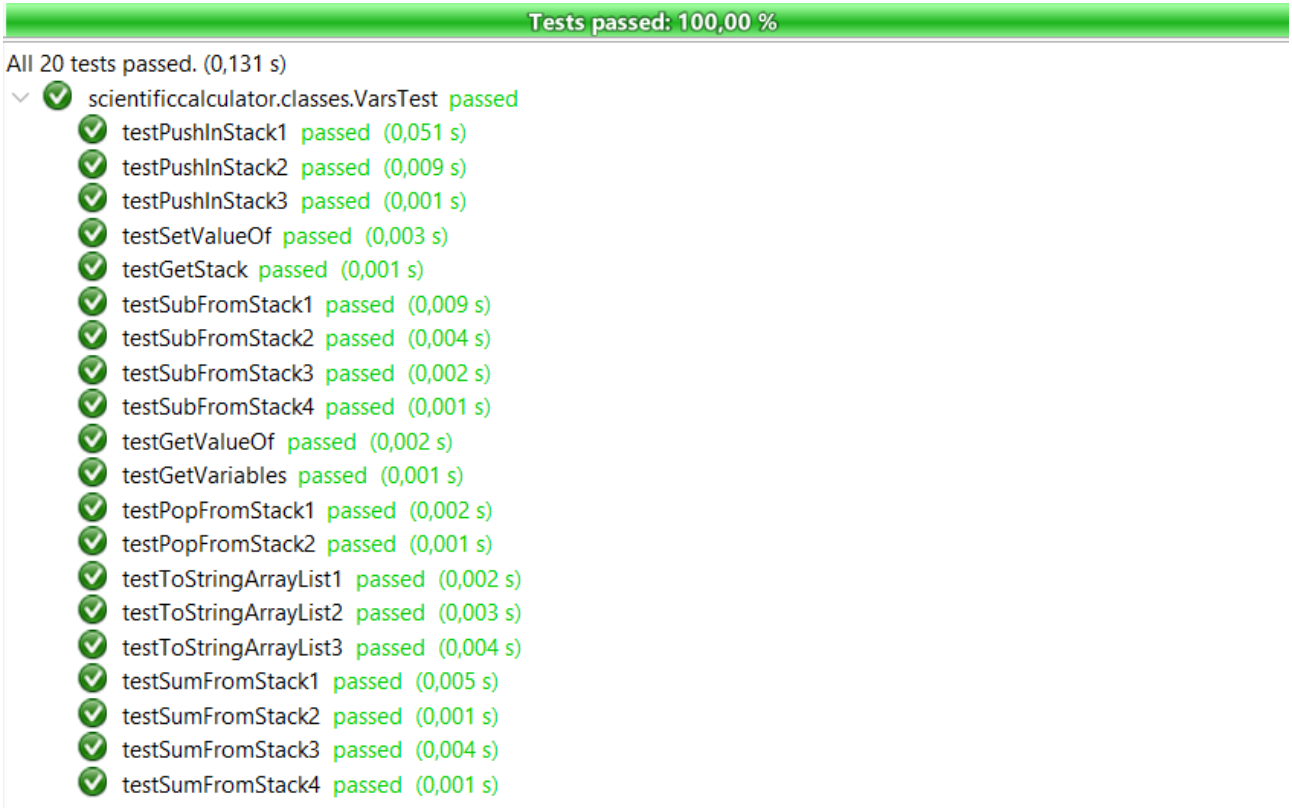
3- Risultati test UnaryCanonicOperations



4- Risultati classe Stack

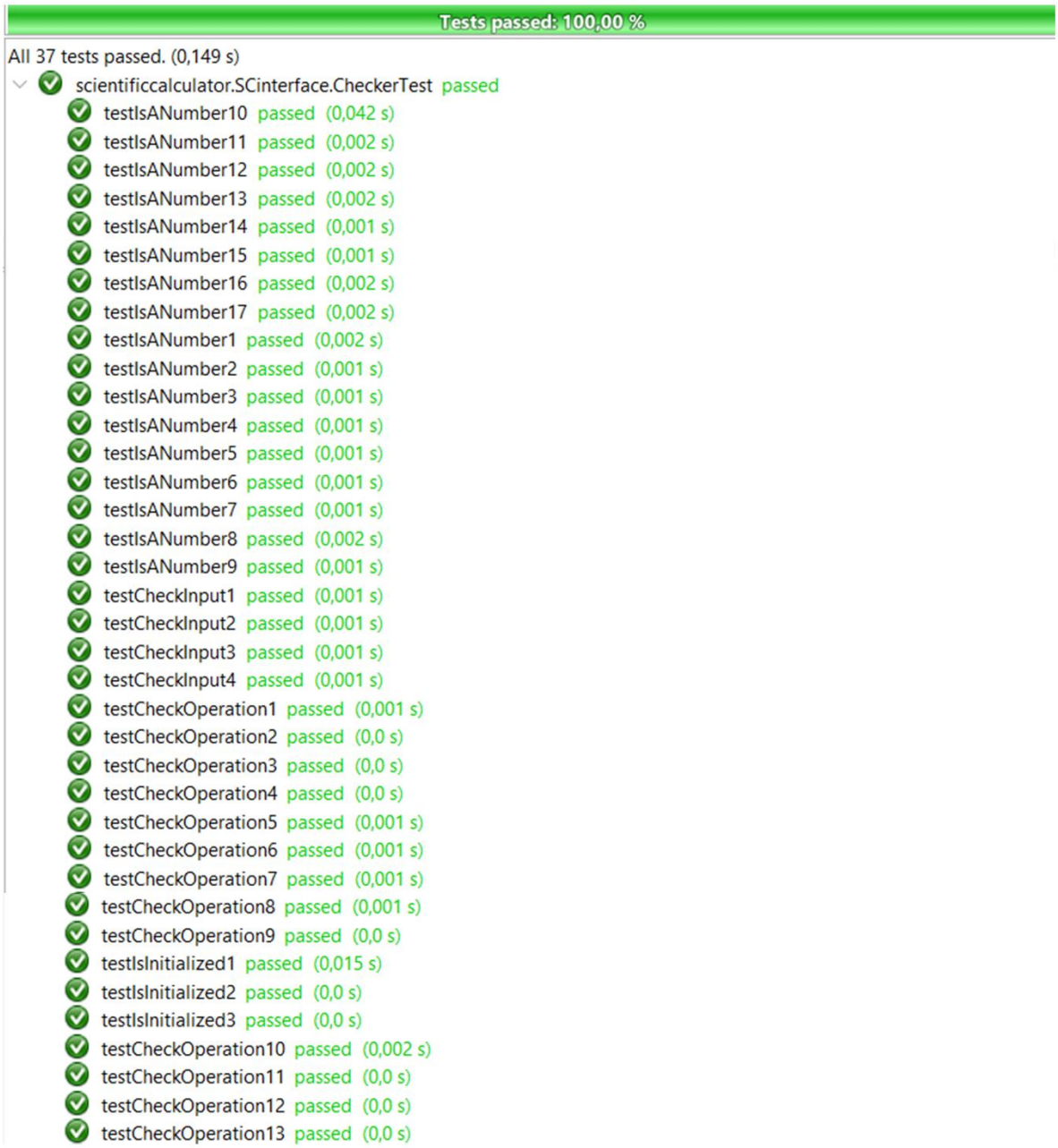


5- Risultati classe Vars



6- Risultati classe ScientificCalculator

7- Risultati classe Checker



TEST SULL'INTERFACCIA UTENTE (TUI)

Presa coscienza dell'importanza assunta dall'implementazione di un'interfaccia grafica con cui l'utente può facilmente sfruttare le funzionalità offerte dal sistema, il team ha attentamente riservato parte della fase di test alla verifica del suo corretto funzionamento. In particolare si è verificato (manualmente):

- il corretto funzionamento dei bottoni presenti;
- la possibilità di poter inserire un input da tastiera;
- la possibilità di visualizzare almeno gli ultimi 12 elementi inseriti nella struttura;
- la possibilità di visualizzare le variabili precedentemente inizializzate;
- la possibilità di inserire numeri complessi secondo diversi formati e di visualizzare una schermata di errore nel caso di formato non permesso;
- la possibilità di inserire delle operazioni;
- la possibilità di eseguire le operazioni prefissate e di visualizzare una schermata di errore nel caso di anomalie;
- la possibilità di aprire una schermata apposita per la gestione delle variabili;
- il corretto funzionamento dei componenti inseriti nella schermata delle variabili;
- la disabilitazione del bottone Invio quando non è stato inizializzato il textfield di inserimento input
- la disabilitazione del bottone Conferma quando nella schermata dedicata alle variabili non è stata ancora selezionata dal menu a tendina la variabile sulla quale lavorare o l'operazione da fare su di essa;
- la visualizzazione dinamica della sezione dedicata allo stack (ogni cambiamento sulla struttura dati viene riportata all'utente);
- la possibilità di sfruttare la proprietà di scrolling sia per la sezione dedicata allo stack sia per quella dedicata alle variabili inizializzate.

I test sopracitati sono stati tutti superati (si riporta che è possibile visualizzare più di 12 elementi nello stack).

MATRICE DI TRACCIABILITÀ

ID Requisito	Design	Implementazione	Testing	Requisiti correlati
IF.1.1	Classe BinaryCanonicOperations, SD_UC2.1	Classe BinaryCanonicOperations, Metodo sum()	UTC-2.1.1 … UTC-2.1.10, UTC-6.1.1, UTC-7.2.1	IF.1, IF.4, ER.1.1
IF.1.2	Classe BinaryCanonicOperations, SD_UC2.1	Classe BinaryCanonicOperations, Metodo sub()	UTC-2.2.1 … UTC-2.2.10, UTC-6.1.2, UTC-7.2.2	IF.1, IF.4, ER.1.1
IF.1.3	Classe BinaryCanonicOperations, SD_UC2.1	Classe BinaryCanonicOperations, Metodo multiply()	UTC-2.3.1 … UTC-2.3.9, UTC-6.1.3, UTC-7.2.3	IF.1, IF.4, ER.1.1
IF.1.4	Classe BinaryCanonicOperations, SD_UC2.1	Classe BinaryCanonicOperations, Metodo divide()	UTC-2.4.1 … UTC-2.4.9, UTC-6.1.4, UTC-7.2.4	IF.1, IF.4, ER.1.1

IF.1.5	Classe UnaryCanonicOperations, SD_UC2.2	Classe UnaryCanonicOperations, Metodo squareRoot()	UTC-3.1.1 … UTC-3.1.7, UTC-6.1.6, UTC-7.2.6	IF.1, IF.4, ER.1.2
IF.1.6	Classe UnaryCanonicOperations, SD_UC2.2	Classe UnaryCanonicOperations, Metodo changeSign()	UTC-3.2.1 … UTC-3.2.11, UTC-6.1.5, 7.2.5.	IF.1, IF.4, ER.1.2
IF.2.1	Classe Stack, SD_UC3.1	Classe Stack, Metodo clear()	UTC-4.5.1, UTC-6.1.7, UTC-7.2.7	IF.2
IF.2.2	Classe Stack, SD_UC3.2	Classe Stack, Metodo drop()	UTC-4.6.1 … UTC-4.6.3, UTC-6.1.8, UTC-7.2.8	IF.2, IF.4, ER.1.2
IF.2.3	Classe Stack, SD_UC3.4	Classe Stack, Metodo duplicate()	UTC-4.8.1, UTC-4.8.2, UTC-6.1.10, UTC-7.2.10	IF.2, IF.4, ER.1.2
IF.2.4	Classe Stack, SD_UC3.3	Classe Stack, Metodo swap()	UTC-4.7.1 … UTC-4.7.3, UTC-6.1.9, UTC-7.2.9	IF.2, IF.4, ER.1.1

IF.2.5	Classe Stack, SD_UC3.5	Classe Stack, Metodo over()	UTC-4.9.1 … UTC-4.9.3, UTC-6.1.11, UTC-7.2.11	IF.2, IF.4, ER.1.1
IF.3.1	Classe Vars, SD_UC4.1	Classe Vars, Metodo popFromStack()	UTC-5.5.1, UTC-5.5.2, UTC-6.2.1 UTC-7.4.1.	IF.3, IF.4, ER.2.1
IF.3.2	Classe Vars, SD_UC4.2	Classe Vars, Metodo pushInStack()	UTC-5.6.1 … UTC-5.6.3, UTC-6.2.2 UTC-7.4.2, UTC-7.4.3.	IF.3, ER.2.2
IF.3.3	Classe Vars, SD_UC4.3	Classe Vars, Metodo sumFromStack()	UTC-5.7.1 … UTC-5.7.4, UTC-6.2.3, UTC-7.4.2, UTC-7.4.3.	IF.3, IF.4, IF.1.1, ER.2.2
IF.3.4	Classe Vars, SD_UC4.4	Classe Vars, Metodo subFromStack()	UTC-5.8.1 … UTC-5.8.4, UTC-6.2.4, UTC-7.4.2, UTC-7.4.3.	IF.3, IF.4, IF.1.1, ER.2.2

IF.4.1	Classe Checker, SD_UC1	Classe ComplexNumber, Metodo toString(), Classe Checker, Metodo isANumber(), Classe ScientificCalculator, Metodo formatComplexNumber().	UTC-1.1.1 … UTC-1.1.4, UTC-6.9.1 … UTC-6.9.6, UTC-7.1.1, UTC-7.1.2, UTC-7.1.10, UTC-7.1.17.	IF.4
IF.4.2	Classe Checker, SD_UC1	Classe ComplexNumber, Metodo toString(), Classe Checker, Metodo isANumber(), Classe ScientificCalculator, Metodo formatComplexNumber().	UTC-1.1.5, UTC-1.1.6, UTC-1.1.8, UTC-6.9.7 … UTC-6.9.10, UTC-7.1.3, UTC-7.1.4, UTC-7.1.7, UTC-7.1.11.	IF.4
IF.4.3	Classe Checker, SD_UC1	Classe ComplexNumber, Metodo toString(), Classe Checker, Metodo isANumber(), Classe ScientificCalculator, Metodo formatComplexNumber().	UTC-1.1.7, UTC-1.1.9, UTC-1.1.10, UTC-6.9.11 … UTC-6.9.14, UTC-7.1.5, UTC-7.1.6,	IF.4

			UTC-7.1.8, UTC-7.1.9.	
IF.4.4	Classe Checker, SD_UC1	Classe ComplexNumber, Metodo toString(), Classe Checker, Metodo isANumber(), Classe ScientificCalculator, Metodo formatComplexNumber().	UTC-6.9.3, UTC-6.9.5.	IF.4
UI.1.1	Classe Controller, Mock-up definitivo	Classe Controller, Metodo vBoxStackUpdate(). Classe Stack. Classe ScientificCalculator, Metodi getStack(), generateLabel().	TUI	UI.1, IF.4, IF.1, IF.2, IF.3
UI.1.2	Classe Controller, Mock-up definitivo	Classe Controller, Metodo vBoxVariableUpdate(). Classe Vars. Classe ScientificCalculator, Metodi getVars(), generateLabel().	TUI	UI.1, IF.3
UI.1.3	Classe Controller, Mock-up definitivo	Classe Controller, Metodo handleVariable().	TUI	UI.1, IF.3, UI.1.6
UI.1.4	Classe Controller, Mock-up definitivo	Classe Controller, Metodi inputFromUI(),inputFromUIWithClear(), delete().	TUI	UI.1, UI.1.5, UI.1.6, IF.4
UI.1.5	Classe Controller, Mock-up definitivo	Classe Controller, Metodo inputFromUI().	TUI	UI.1, IF.4
UI.1.6	Classe Controller, Mock-up definitivo	Classe Controller, Metodo inputFromUIWithClear().	TUI	UI.1, IF.1, IF.2, IF.3
ER.1.1	Classe InvalidOperandsException, SD_UC2.1, SD_UC3.3, SD_UC3.5	Classe InvalidOperandsException,	UTC-4.7.1, UTC-4.7.2,	ER.1, IF.1.1, IF.1.2, IF.1.3,

		Classe ScientificCalculator, Metodi sum(), sub(), multiply(), divide(), Classe Stack, Metodi swap(), over().	UTC-4.9.1, UTC-4.9.2, UTC-6.3.1, UTC-6.3.2, UTC-6.4.1, UTC-6.4.2, UTC-6.5.1, UTC-6.5.2, UTC-6.6.1, UTC-6.6.2, UTC-7.2.1 … UTC-7.2.4, UTC-7.2.9, UTC-7.2.11.	IF.1.4, IF.2.4, IF.2.5
ER.1.2	Classe InvalidOperandsException, SD_UC2.2, SD_UC3.2, SD_UC3.4	Classe InvalidOperandsException, Classe ScientificCalculator, Metodi squareRoot(), ChangeSign(), Classe Stack, Metodi drop(), duplicate().	UTC-4.6.1, UTC-4.8.1, UTC-6.7.1, UTC-6.8.1, UTC-7.2.5, UTC-7.2.6, UTC-7.2.8, UTC-7.2.10.	ER.1, IF.1.5, IF.1.6, IF.2.2, IF.2.3
ER.2.1	Classe Checker, Class Controller, SD_UC4.1	Classe Vars, Metodo popFromStack().	UTC-7.4.1.	ER.2, IF.3.1
ER.2.2	Classe InvalidOperandsException, Classe UninitializedException,	Classe InvalidOperandsException, Classe UninitializedException.	UTC-7.4.1, UTC-4.3.3.	ER.2

	SD_UC4.1, SD_UC4.2, SD_UC4.3, SD_UC4.4			
ER.2.2.1	Classe UninitializedException, SD_UC4.1	Classe UninitializedException. Classe Vars, Metodo pushInStack().	UTC-5.6.3, UTC-7.4.2, UTC-7.4.3.	ER.2.2, IF.3.2
ER.2.2.2	Classe InvalidOperandsException, SD_UC4.2	Classe InvalidOperandsException, Classe Vars, Metodo popFromStack().	UTC-4.3.3, UTC-5.5.2, UTC-7.4.2, UTC-7.4.3.	ER.2.2, IF.3.1
ER.2.2.3	Classe InvalidOperandsException, Classe UninitializedException, SD_UC4.3, SD_UC4.4	Classe InvalidOperandsException, Classe UninitializedException, Classe Vars, Metodo sumFromStack(), Classe Vars, Metodo subFromStack().	UTC-4.3.3, UTC-5.7.3, UTC-5.8.2, UTC-7.4.2, UTC-7.4.3.	ER.2.2, IF.3.3, IF.3.4
ER.3	Classe Checker, Classe InvalidInputException, SD_UC1, SD_UC3.1-3.5	Classe InvalidInputException, Classe Checker, Metodo isANumber().	UTC-7.1.12 … UTC-7.1.17	IF.4, IF.2