

**RELAZIONE INTEGRAZIONE E TEST DI SISTEMI SOFTWARE**

**A.A. 2023/2024**

**STUDENTI:**

**Umberto Messina 675494 u.messina1** **@studenti.uniba.it**

**Omar Balde 759522 o.balde** **@studenti.uniba.it**

Indice

Indice …1

Introduzione …2

Perché fare testing

Il nostro codice

Homework 1

convertRomanToDecimal

convertDecimalToRoman

isValidRomanNumber

areInterdipendent

isPalindrome

Homework 2

…

…

Capitolo 1 – Homework 1

Specification based test - – Approccio a 7 step

Tutti le classi di test vanno inserite e spiegate all’interno di “Specification based test”

Structural Test e Code Coverage

Analisi Code Coverage

Soluzioni trovate

Capitolo 2 – Homework 2

Property based test

…

Conclusioni

INTRODUZIONE

In questa relazione, presenteremo il processo di progettazione e sviluppo della nostra suite di test per la classe RomanToDecimalConverter. Esamineremo le metodologie utilizzate per creare i casi di test, gli strumenti impiegati per valutare la copertura del codice e l'efficacia complessiva della nostra suite di test nel garantire la correttezza e l'affidabilità del software sviluppato.

**PERCHE’ FARE TESTING**

La creazione di una suite di test efficace è un passo cruciale nello sviluppo del software, in quanto ci consente di verificare il corretto funzionamento delle funzionalità implementate e di individuare eventuali errori o comportamenti indesiderati. Nel contesto del nostro progetto, abbiamo sviluppato una suite di test per la classe RomanToDecimalConverter, che offre funzionalità per la conversione di numeri romani in numeri decimali e viceversa.

**IL NOSTRO CODICE**

Il nostro codice si occupa della conversione tra numeri romani e decimali (e viceversa).

Consiste nell'interpretare correttamente una stringa che rappresenta un numero romano e convertirla in un numero decimale equivalente, o viceversa. Ad esempio, il numero romano "IV" deve essere convertito nel numero decimale 4 e il numero decimale 1994 deve essere convertito nel numero romano "MCMXCIV". Inoltre, abbiamo implementato 3 metodi che aggiungo ulteriori funzionalità, ovvero:

1. Verifica se una stringa è un numero romano valido (controlla se contiene solo caratteri validi (I, V, X, L, C, D, M).
2. Verifica se due numeri romani sono interdipendenti (controlla se il resto della loro divisione è 0)
3. Verifica se un numero romano è palindromo (leggibile da sinistra verso destra e da destra verso sinistra).

**HOMEWORK 1**

Per l’homework 1 ci è stato chiesto di sviluppare una suite di test black-box (specification-based testing) con gli structural testing (white-box) e con l’analisi di copertura del codice (code coverage). Nello specifico:

* Per test black box, intendiamo una metodologia di verifica del software che si concentra sul comportamento esterno di un sistema senza considerare il suo funzionamento interno. In altre parole, il tester non ha bisogno di conoscere il codice sorgente o la struttura interna del software per eseguire questi test.
* Per test white box o test strutturale, intendiamo una metodologia di verifica del software che si concentra sulla struttura interna e sul codice sorgente del programma. A differenza del test black box, che si focalizza sul comportamento esterno, il test white box analizza come il codice è stato implementato per individuare potenziali errori.
* Per code coverage intendiamo la percentuale del codice sorgente di un programma che viene effettivamente eseguita durante l'esecuzione dei test. In parole semplici, indica quanto del codice viene testato attivamente.

**METODI DELLA CLASSE RomanToDecimalConverter**

1. convertRomanToDecimal

inserire screen metodo

Il metodo `convertRomanToDecimal` prende in input una stringa `romanNumber`, che rappresenta un numero romano, e restituisce il suo valore decimale corrispondente. Inizia controllando se l'input è valido (non nullo e non vuoto). Successivamente, esegue un'iterazione sui caratteri della stringa dal più significativo al meno significativo. Durante l'iterazione, converte ciascun carattere romano in un valore decimale utilizzando il metodo `getRomanValue`, tenendo conto delle regole di somma e sottrazione. Infine, restituisce il valore decimale totale.

1. getRomanValue

inserire screen metodo

Il metodo `getRomanValue` accetta un carattere `romanChar` che rappresenta un simbolo romano e restituisce il suo valore decimale corrispondente. Utilizza un'istruzione switch per associare ciascun simbolo romano al suo valore decimale. Se il simbolo romano non è riconosciuto, restituisce 0.

1. convertDecimalToRoman

inserire screen

Il metodo `convertDecimalToRoman` converte un numero decimale in un numero romano. Inizia controllando se il numero decimale è inferiore o uguale a zero; in tal caso restituisce una stringa vuota. Successivamente, utilizza due array paralleli `decimalValues` e `romanSymbols` per associare i valori decimali ai simboli romani corrispondenti. Utilizza un ciclo while per iterare attraverso gli array, sottraendo il valore corrente dal numero decimale finché il numero decimale non diventa zero. Durante ogni iterazione, aggiunge il simbolo romano corrispondente al `StringBuilder` `romanNumber`. Infine, restituisce la rappresentazione romana del numero decimale convertito come una stringa.

1. isValidRomanNumber  
   inserire screen  
   Il metodo `isValidRomanNumber` verifica se una stringa rappresenta un numero romano valido. Utilizza un'espressione regolare per definire il pattern di un numero romano valido, che include solo i simboli romani consentiti: I, V, X, L, C, D, e M. Il metodo, quindi, controlla se la stringa `romanNumber` corrisponde al pattern definito. Se la corrispondenza viene trovata, il metodo restituisce true, altrimenti restituisce false.
2. areInterdipedent

inserire screen

Il metodo `areInterdependent` verifica se due numeri romani sono interdipendenti, ossia se uno dei due numeri è divisibile per l'altro. Per fare ciò, converte entrambi i numeri romani in numeri decimali utilizzando il metodo `convertRomanToDecimal`. Quindi, verifica se uno dei numeri decimali è divisibile per l'altro. Se uno dei due numeri decimali è divisibile per l'altro, il metodo restituisce true; altrimenti, restituisce false.

1. isPalindrome

inserire screen

Il metodo `isPalindrome` verifica se un numero romano è un palindromo, cioè se può essere letto allo stesso modo sia da sinistra a destra che da destra a sinistra. Per fare ciò, crea una stringa `reversedRomanNumber` che rappresenta il numero romano invertito utilizzando la classe `StringBuilder` per invertire la stringa `romanNumber`. Quindi, controlla se `romanNumber` è uguale a `reversedRomanNumber`. Se la condizione è vera, il metodo restituisce true, altrimenti restituisce false.

**HOMEWORK 2**

**…….**

CAPITOLO 1 – HOMEWORK 1

Descrizione della creazione della suite di test seguendo i 7 passaggi:

1. Analisi dei requisiti:

Il programma RomanToDecimalConverter deve convertire numeri romani validi in numeri decimali e viceversa.

Deve anche controllare la validità dei numeri romani e verificare se due numeri romani sono interdipendenti (uno divisibile per l'altro) o palindromi (leggibili allo stesso modo da sinistra e da destra).

2. Esplorazione del comportamento del programma:

Sono stati testati diversi input validi e invalidi per le funzioni convertRomanToDecimal, convertDecimalToRoman, isValidRomanNumber, areInterdependent, e isPalindrome.

Si è verificato che il programma producesse i risultati attesi per gli input validi e gestisse correttamente gli input invalidi.

3. Identificazione di partizioni e casi limite:

Partizioni:

Input validi e invalidi per convertRomanToDecimal e convertDecimalToRoman.

Numeri romani validi e invalidi per isValidRomanNumber.

Coppie di numeri romani interdipendenti e non interdipendenti per areInterdependent.

Stringhe palindrome e non palindrome per isPalindrome.

Casi limite:

Input nullo o vuoto.

Negativi per convertDecimalToRoman.

Caratteri non validi in numeri romani.

Numeri romani con ripetizioni o sottrazioni.

4. Identificazione dei casi di test:

Sono stati definiti casi di test specifici per coprire le partizioni e i casi limite identificati.

I test verificano sia i comportamenti attesi che quelli non attesi (es. input invalidi che dovrebbero generare errori).

5. Definizione dei casi di test:

I casi di test sono stati scritti utilizzando il framework JUnit e le sue annotazioni (@Test, @DisplayName).

Ogni test case ha un nome descrittivo che indica lo scenario che sta testando.

I test utilizzano affermazioni di JUnit (Assertions) per verificare i risultati attesi (es. assertEquals, assertTrue).

6. Automatizzazione dei casi di test:

I casi di test sono stati eseguiti automaticamente utilizzando un runner di test JUnit.

Questo consente di eseguire i test in modo rapido e affidabile, riducendo il rischio di errori umani.

7. Miglioramento della suite di test con creatività ed esperienza:

La suite di test può essere ulteriormente migliorata includendo:

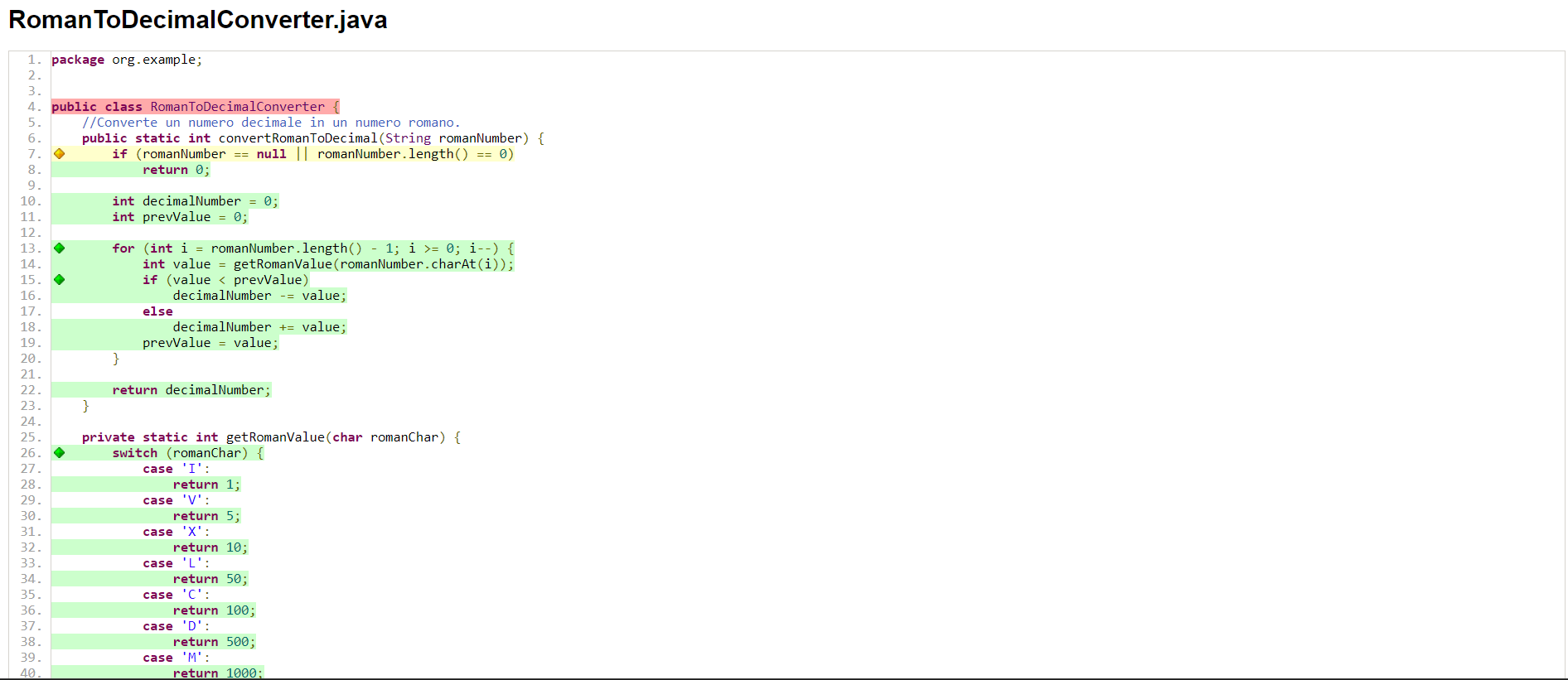
Test per scenari di combinazione o edge case più complessi.

Mock o stub per isolare singole unità e testare la loro interazione.

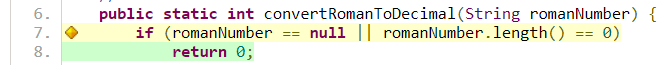
Test di regressione per garantire che le modifiche al codice non introducano regressioni.

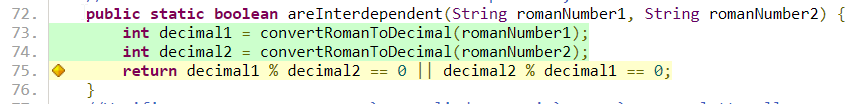
La suite di test attuale copre diverse partizioni e casi limite, dimostrando una buona comprensione dei requisiti e delle funzionalità del programma. L'applicazione dei suggerimenti nel punto 7 può contribuire a rafforzare ulteriormente la qualità e la completezza della suite di test.

**ANALISI CODE COVERAGE - INIZIALE**



  
come possiamo vedere dagli screen la code coverage è quasi totalmente verde tranne che per due linee





**SOLUZIONI TROVATE**

L’unica soluzione che abbiamo implementato copre soltanto la riga di codice numero 7; la riga 75 non può essere coperta totalmente dai test, perché i casi che mancano nella nostra suite sono quelli che gestiscono le divisioni con 0 o null e, a meno di refactor del codice sorgente, non è possibile implementarli nella classe di test.

Inserire screen soluzione implementata (testConvertRomanToDecimal\_NullOrEmptyInput)

**ANALISI CODE COVERAGE – FINALE**

Dopo aver implementato la soluzione, la code coverage si presenta in questo modo

