SERLAB
Software Engineering Research
UNIVERSITÀ DEGLI STUDI DI BARI

ALETHEIA

Software Testing

Versione 1.0.0

Data di rilascio: 01/01/2025

Integrazione e Test di Sistemi Software

A.A. 2024-2025

[Informatica e Tecnologie per la Produzione del Software]

Realizzato da

Lorenzo Calabrese - 779727 - l.calabrese28@studenti.uniba.it

Francesco Conforti - 776628- f.conforti9@studenti.uniba.it

Giuseppe Pio De Biase - 774957- g.debiase5@studenti.uniba.it

Alexandru Gheorghe Zaharia - 774747 - a.zaharia@studenti.uniba.it

Dipartimento di Informatica - Università degli Studi di Bari
Via Orabona, 4 - 70125 - Bari
Tel: +39.080.5443270 • +39.080.5442300 | Fax: +39.080.5442536
serlab.di.uniba.it
Aletheia Software StartUp aletheia.aziendale@gmail.com

Sommario

SPECIFICATION-BASED TESTING

TASK 1

1. Specification Based Testing - Black Box

```java
public static String replace(String original, String target, String replacement) {
    if (original == null || target == null || replacement == null) {
        throw new NullPointerException("original, target e replacement non devono essere null");
    }
    if (target.isEmpty()) {
        throw new IllegalArgumentException("target non deve essere vuoto");
    }

    // Usa un StringBuilder per costruire il risultato
    StringBuilder result = new StringBuilder();
    int start = 0;
    int index;

    // Cerca tutte le occorrenze di target e le sostituisce con replacement
    while ((index = original.indexOf(target, start)) != -1) {
        result.append(original, start, index); // Aggiunge la parte prima di target
        result.append(replacement);            // Aggiunge replacement
        start = index + target.length();       // Aggiorna il punto di partenza
    }

    // Aggiunge la parte rimanente della stringa originale
    result.append(original.substring(start));
    return result.toString();
}
```

1.1. Comprensione delle richieste

1.1.1. Obiettivi

Il metodo **`replace()`** è progettato per sostituire tutte le occorrenze di una sottostringa (**`target`**) con un'altra sottostringa (**`replacement`**) all'interno di una stringa originale (**`original`**). Si tratta di un'operazione comunemente utilizzata per manipolare stringhe in diversi contesti, come la formattazione di testo, la pulizia di dati o la personalizzazione di messaggi.

1.1.2. Parametri

I parametri richiesti dal metodo **replace()** sono:

Parametro	Descrizione	Valore Ammesso	Eccezioni Lanciate
`target`	Stringa originale su cui operare	Non può essere `null` o `""`	NullPointerException, IllegalArgumentException
`replacement`	Sottostringa da sostituire	Non può essere `null`	NullPointerException
`original`	Sottostringa con cui sostituire il `target`	Non può essere `null`	NullPointerException

1.1.3. Output

Il metodo `replace()` restituisce un valore di tipo `String` che rappresenta il risultato della sostituzione delle occorrenze della sottostringa `target` con la sottostringa `replacement` nella stringa originale `original`.

1.2. Analizza il comportamento del programma con diversi input

Sono stati eseguiti 4 test in cui sono state verificate le funzionalità principali del metodo. In particolare sono stato testati i casi in cui:

1. Caso con **`valori corretti`** e sostituzioni effettuate
2. Eccezione **`NullPointerException`** su original null
3. Eccezione **`IllegalArgumentException`** su target vuoto
4. Nessuna sostituzione effettuata quando **`target`** non esiste

```java
public class BlackBoxTest {

    @DisplayName("Caso semplice con tutti i valori corretti")
    @Test
    public void correctCaseTest() {
        // Input
        String original = "hello world", target = "world", replacement = "everyone";
        // Expected output
        String expected = "hello everyone";
        Assertions.assertEquals(expected, StringUtils.replace(original, target, replacement));
    }

    @DisplayName("Eccezione NullPointerException su original null")
    @Test
    public void nullOriginalTest() {
        // Input
        String original = null, target = "world", replacement = "everyone";
        Assertions.assertThrows(NullPointerException.class, () -> StringUtils.replace(original, target, replacement));
    }

    @DisplayName("Eccezione IllegalArgumentException su target vuoto")
    @Test
    public void emptyTargetTest() {
        // Input
        String original = "hello world", target = "", replacement = "everyone";
        Assertions.assertThrows(IllegalArgumentException.class, () -> StringUtils.replace(original, target, replacement));
    }

    @DisplayName("Nessuna sostituzione effettuata quando target non esiste")
    @Test
    public void noReplacementNeededTest() {
        // Input
        String original = "hello world", target = "planet", replacement = "everyone";

        // Expected output
        String expected = "hello world";
        Assertions.assertEquals(expected, StringUtils.replace(original, target, replacement));
    }
}
```

BlackBoxTest.correctCaseTest	passed	8 ms
BlackBoxTest.noReplacementNeededTest	passed	0 ms
BlackBoxTest.emptyTargetTest	passed	2 ms
BlackBoxTest.nullOriginalTest	passed	1 ms

1.3. Esplora gli input, gli output e identifica le partizioni

1.3.1. Input individuali (classi di input)

`original`:

- empty;
- null;
- length>=1

`target`:

- empty;
- null;
- length>=1

`replacement`:

- empty;
- null;
- length >=1;

1.3.2. Combinazione dei valori di input

Dati 3 insiemi, corrispondenti ai parametri, con:

- `original` 3 valori;
- `target` 3 valori;
- `replacement` 3 valori;

è possibile ottenere un massimo di 27 combinazioni:

T	original	target	replacement
1	null	null	null
2	null	null	empty
3	null	null	length >= 1
4	null	empty	null
5	null	empty	empty
6	null	empty	length >= 1
7	null	length>=1	null
8	null	length>=1	empty
9	null	length>=1	length >= 1

10	empty	null	null
11	empty	null	empty
12	empty	null	length>=1
13	empty	empty	null
14	empty	empty	empty
15	empty	empty	length >= 1
16	empty	length>=1	null
17	empty	length>=1	empty
18	empty	length >= 1	length >= 1
19	length >= 1	null	null
20	length >= 1	null	empty
21	length >= 1	null	length >= 1
22	length >= 1	empty	null
23	length >= 1	empty	empty
24	length >= 1	empty	length>=1
25	length >= 1	length >= 1	null
26	length >= 1	length >= 1	empty
27	length >= 1	length >= 1	length >= 1

1.3.3 Classi di output

Le classi di outputs sono:

- stringa nulla
- stringa vuota
- string di lunghezza >= 1

1.4 Identificazione dei Boundary Cases

I boundary cases sono:

1. `original`
   a. on point: `original` >= 0
   b. off point: `original` = null
2. `target`
   a. on point: `target` >= 1
   b. off point:`target` = null | vuota
3. `replacement`
   a. on point: `replacement` >= 0
   b. off point:`replacement` = null

1.5 Creazione dei casi di test

In questa fase, l'obiettivo è quello di individuare i test essenziali, facendo una selezione di 27 test totali individuati in precedenza.

I criteri di selezione adottati sono:

- testare i casi eccezionali, come `null` ed `empty`, soltanto una volta, senza combinarli
- testare i casi che evidenziano singolarmente gli `off points`

Quindi, sono stati selezionati i seguenti test:

T	original	target	replacement
9	null	length>=1	length >= 1
21	length >= 1	null	length >= 1
24	length >= 1	empty	length>=1
25	length >= 1	length >= 1	null

Con i test cases:

- Il test **9** individua `l'off point` del parametro `original`
- I test **21** e **24** individuano `l'off point` del parametro `target`
- Il test **25** individua `l'off point` del parametro `replacement`

Per effettuare un testing efficace, la test suite è completata dai seguenti test:

T	original	target	replacement
1	null	null	null
18	empty	length >= 1	length >= 1
26	length >= 1	length >= 1	empty
27	length >= 1	length >= 1	length >= 1

In aggiunta ai test case eccezionali abbiamo scelto di eseguire anche i test cases che mostrano gli on point dei parametri come:

- Il test **18** individua `l'on point` del parametro `original`
- Il test **26** individua `l'on point` del parametro `replacement`
- Il test **27** individua `l'on point` del parametro `target`

Per comprendere al meglio il funzionamento del metodo `replace()` abbiamo deciso di esaminare ulteriori scenari eccezionali:

- `target` non esistente nella stringa `original`
- `original` con `target` case sensitive
- `original` con caratteri speciali
- `target` e `replacement` uguali
- `original`, `target` e `replacement` di lunghezza 1
- `original` di lunghezza di 1.000.000 di caratteri

Dopo aver rielaborato la test suite abbiamo rilevato i seguenti test da effettuare:

T	original	target	replacement
1	null	length>=1	length >= 1
2	length >= 1	null	length >= 1
3	length >= 1	empty	length>=1
4	length >= 1	length >= 1	null

5	null	null	null
6	empty	length >= 1	length >= 1
7	length >= 1	length >= 1	empty
8	length >= 1	length >= 1	length >= 1
9	length >= 1	length >= 1 (target non esistente nella stringa original)	length >= 1
10	length >= 1	length >= 1 (target case sensitive)	length >= 1
11	length >= 1 (original con caratteri speciali)	length >= 1	length >= 1
12	length >= 1	length = 1 (uguale a replacement)	length >= 1 (uguale a target)
13	length = 1	length = 1	length = 1
14	length = 1.000.000	length = 1	length = 1

1.6 Automatizzazione della test suite

Abbiamo implementato i test utilizzando i test parametrici di JUnit5.

```java
public class StringUtilsTest {

    @DisplayName("Combinazione di possibili valori.")
    @ParameterizedTest
    @MethodSource("provideReplaceTestCases")
    void testSimpleReplace(String original, String target, String replacement, String expected) {
        Assertions.assertEquals(expected, StringUtils.replace(original, target, replacement));
    }

    static Stream<Arguments> provideReplaceTestCases() {  1 usage
        return Stream.of(
                // T8 - Caso con valori corretti e sostituzioni effettuate
                Arguments.of( ...arguments: "Ciao mondo, mondo bello!", "mondo", "universo", "Ciao universo, universo bello!"),
                // T9 - Caso in cui il target non è presente nella stringa originale
                Arguments.of( ...arguments: "Ciao mondo!", "galassia", "universo", "Ciao mondo!"),
                // T6 - Caso in cui original è vuoto
                Arguments.of( ...arguments: "", "mondo", "universo", ""),
                // T10 - Caso con target case sensitive
                Arguments.of( ...arguments: "Ciao mondo, Mondo bello!", "mondo", "universo", "Ciao universo, Mondo bello!"),
                // T11 - Caso con original con caratteri speciali
                Arguments.of( ...arguments: "Ciao@mondo#mondo", "mondo", "$", "Ciao@$#$"),
                // T7 - Caso con il replacement vuoto
                Arguments.of( ...arguments: "Integrazione e test di sistemi software", "test", "", "Integrazione e  di sistemi software"),
                // T12 -Caso in cui replacement e target sono uguali
                Arguments.of( ...arguments: "Integrazione e test di sistemi software", "software", "software", "Integrazione e test di sistemi software"),
                // T13 - Caso con stringa di lunghezza 1
                Arguments.of( ...arguments: "m", "m", "M", "M"),
                // T14 - Caso con stringhe molto lunghe
                Arguments.of( ...arguments: "a".repeat( count: 1_000_000), "a", "b", "b".repeat( count: 1_000_000))
        );
    }

    @DisplayName("1: Target nullo, 2: Original nullo, 3: Replacement nullo, 4: Parametri nulli")
    @ParameterizedTest
    @MethodSource("provideNullValuesTestCases")
    void testValuesNull(String original, String target, String replacement) {
        Assertions.assertThrows(NullPointerException.class, () -> StringUtils.replace(original, target, replacement));
    }

    static Stream<Arguments> provideNullValuesTestCases() {  1 usage
        return Stream.of(
                // T2 - Caso con target nullo
                Arguments.of( ...arguments: "Ciao mondo, mondo bello!", null, "universo"),
                // T1 - Caso con original nullo
                Arguments.of( ...arguments: null, "mondo", "universo"),
                // T4 - Caso con replacement nullo
                Arguments.of( ...arguments: "Ciao mondo, mondo bello!", "mondo", null),
                // T5 - Caso in cui original, target e replacement sono nulli
                Arguments.of( ...arguments: null, null, null)
        );
    }

    @DisplayName("1: Target vuoto")
    @ParameterizedTest
    @CsvSource({
            // T3 - Caso con target vuoto
            "'Ciao mondo, mondo bello!', '' , 'universo'",
    })
    void testTargetEmpty(String original, String target, String replacement) {
        Assertions.assertThrows(IllegalArgumentException.class, () -> StringUtils.replace(original, target, replacement));
    }
}
```

La suite di test non ha rilevato alcun bug all'interno del metodo analizzato.

```
✓ StringUtilsTest (org.task1)
  ✓ Combinazione di possibili valori.
    ✓ [1] Ciao mondo, mondo bello!, mondo, universo, Ciao universo, universo bello!
    ✓ [2] Ciao mondo!, galassia, universo, Ciao mondo!
    ✓ [3] , mondo, universo,
    ✓ [4] Ciao mondo, Mondo bello!, mondo, universo, Ciao universo, Mondo bello!
    ✓ [5] Ciao@mondo#mondo, mondo, $, Ciao@$#$
    ✓ [6] Integrazione e test di sistemi software, test, , Integrazione e  di sistemi software
    ✓ [7] Integrazione e test di sistemi software, software, software, Integrazione e test di sistemi software
    ✓ [8] m, m, M, M
    ✓ [9] aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  ✓ 1: Target nullo, 2: Original nullo, 3: Replacement nullo, 4: Parametri nulli
    ✓ [1] Ciao mondo, mondo bello!, null, universo
    ✓ [2] null, mondo, universo
    ✓ [3] Ciao mondo, mondo bello!, mondo, null
    ✓ [4] null, null, null
  ✓ 1: Target vuoto
    ✓ [1] Ciao mondo, mondo bello!, , universo
```

1.7 Espansione della test suite

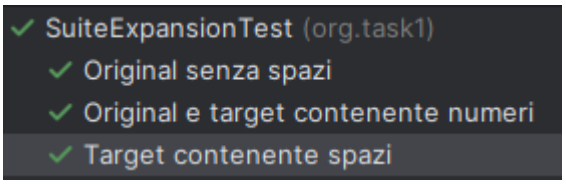
Sono stati considerati anche altri scenari che includono situazioni comuni, come stringhe semplici, stringhe con spazi e numeri, verificando che il comportamento del metodo sia affidabile in contesti differenti. Questo assicura la robustezza e la versatilità del metodo in casi d'uso reali.

Quindi, a completamento della test suite si aggiungono i seguenti 3 test cases.

T	original	target	replacement
1	senza spazi	length>=1	length >= 1
2	contenente numeri	contenente numeri	length >= 1
3	length >= 1	contenente spazi	length>=1

```java
public class SuiteExpansionTest {

    @DisplayName("Combinazione di possibili valori.")
    @ParameterizedTest
    @MethodSource("provideReplaceTestCases")
    void testSimpleReplace(String original, String target, String replacement, String expected) {
        Assertions.assertEquals(expected, StringUtils.replace(original, target, replacement));
    }

    static Stream<Arguments> provideReplaceTestCases() {  1 usage
        return Stream.of(
                // Original senza spazi
                Arguments.of( ...arguments: "helloworld", "world", "everyone", "helloeveryone"),
                // Original e target contenente numeri
                Arguments.of( ...arguments: "JUnit5 Java Framework", "5", "Testing", "JUnitTesting Java Framework"),
                // target contenente spazi
                Arguments.of( ...arguments: "JUnit5 Java Framework", " Java ", "Testing", "JUnit5TestingFramework")
        );
    }

}
```

I test aggiuntivi hanno avuto anch'essi un esito positivo.



1.8 Pianificazione test cases

La progettazione della Test Suite è stata sviluppata in modo sistematico per garantire la copertura delle funzionalità chiave del metodo `StringUtils.replace()`. Sono stati identificati i principali scenari d'uso e le combinazioni di input che potrebbero influire sul comportamento della funzione. Attraverso l'uso di test parametrizzati, ogni caso è stato pianificato per verificare situazioni comuni e potenziali edge case, come stringhe con spazi, numeri o caratteri speciali. Questo approccio permette di mantenere la test suite chiara, riutilizzabile e facilmente espandibile per futuri aggiornamenti o nuove esigenze. La pianificazione ha tenuto conto della leggibilità e della modularità dei test per facilitare la manutenzione. Si può visualizzare la pianificazione al seguente link Progettazione Test

STRUCTURAL TESTING

AND

CODE COVERAGE

Task 1

2. Structural testing and Code Coverage

2.1. Esecuzione dei test con analisi della copertura del codice

Con l'applicazione di un approccio step-by-step del **Specification-based testing** e un'analisi approfondita dell'implementazione del metodo, abbiamo eseguito la test suite utilizzando **JaCoCo** come tool per la **code coverage**. Come evidenziato dai risultati, la copertura della classe testata è pari al 100% sia per il criterio **Line** che per il criterio **Branch**.

Current scope: all classes

Overall Coverage Summary

Package	Class, %	Method, %	Branch, %	Line, %
all classes	100% (1/1)	100% (1/1)	100% (10/10)	100% (12/12)

Coverage Breakdown

Package	Class, %	Method, %	Branch, %	Line, %
org.task1	100% (1/1)	100% (1/1)	100% (10/10)	100% (12/12)

generated on 2024-12-30 11:06

Coverage Summary for Class: StringUtils (org.task1)

Class	Class, %	Method, %	Branch, %	Line, %
StringUtils	100% (1/1)	100% (1/1)	100% (10/10)	100% (12/12)

```java
package org.task1;

public class StringUtils {

    private StringUtils() { }

    /**
     * Sostituisce tutte le occorrenze di una sottostringa (`target`) con un'altra sottostringa (`replacement`)
     * all'interno di una stringa originale (`original`).
     *
     * @param original    La stringa originale in cui effettuare la sostituzione.
     * @param target      La sottostringa da sostituire.
     * @param replacement La nuova sottostringa con cui sostituire.
     * @return Una nuova stringa con tutte le sostituzioni effettuate.
     * @throws NullPointerException     Se `original`, `target` o `replacement` sono null.
     * @throws IllegalArgumentException Se `target` è vuoto.
     */

    public static String replace(String original, String target, String replacement) {
        if (original == null || target == null || replacement == null) {
            throw new NullPointerException("original, target e replacement non devono essere null");
        }
        if (target.isEmpty()) {
            throw new IllegalArgumentException("target non deve essere vuoto");
        }

        // Usa un StringBuilder per costruire il risultato
        StringBuilder result = new StringBuilder();
        int start = 0;
        int index;

        // Cerca tutte le occorrenze di target e le sostituisce con replacement
        while ((index = original.indexOf(target, start)) != -1) {
            result.append(original, start, index); // Aggiunge la parte prima di target
            result.append(replacement);            // Aggiunge replacement
            start = index + target.length();       // Aggiorna il punto di partenza
        }

        // Aggiunge la parte rimanente della stringa originale
        result.append(original.substring(start));
        return result.toString();
    }
}
```

2.2 Mutation testing

Il **Mutation Testing** è un metodo di testing strutturale che prevede l'introduzione intenzionale di errori, noti come **"mutanti"**, all'interno del codice al fine di valutare l'efficacia di una suite di test nel rilevarli. Questo approccio si basa sul principio del **Coupling Effect**: se una suite di test è in grado di individuare errori di bassa complessità, si ritiene che possa anche rilevare anomalie più complesse.

Per l'implementazione del Mutation Testing è stato utilizzato **PIT Mutation Testing**, che ha generato 7 mutanti, appartenenti a diverse tipologie, come dettagliato negli screenshot allegati.

Pit Test Coverage Report

Package Summary

org.task1

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
1	100%	13/13	100%	7/7	100%	7/7

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
StringUtils.java	100%	13/13	100%	7/7	100%	7/7

Report generated by PIT 1.15.8

Come mostrato dal **report generato dal tool**, la test suite sviluppata è stata in grado di rilevare e contrastare tutte le mutazioni inserite nel codice da testare.

Pertanto, sebbene non possa esistere una suite di test completamente esaustiva a causa di elevati costi ed un elevato tempo necessario, si può concludere che quella sviluppata dimostri un elevato **grado di efficacia e robustezza**.

Di seguito, si presenta un resoconto dettagliato delle mutazioni introdotte, comprensivo del numero di riga in cui è stato inserito ciascun errore e della specifica tipologia di mutazione.

Code Coverage

```
1   package org.task1;
2
3   public class StringUtils {
4
5       private StringUtils() { }
6
7       /**
8        * Sostituisce tutte le occorrenze di una sottostringa (`target`) con un'altra sottostringa (`replacement`)
9        * all'interno di una stringa originale (`original`).
10       *
11       * @param original    La stringa originale in cui effettuare la sostituzione.
12       * @param target      La sottostringa da sostituire.
13       * @param replacement La nuova sottostringa con cui sostituire.
14       * @return Una nuova stringa con tutte le sostituzioni effettuate.
15       * @throws NullPointerException     Se `original`, `target` o `replacement` sono null.
16       * @throws IllegalArgumentException Se `target` è vuoto.
17       */
18
19      public static String replace(String original, String target, String replacement) {
20 3        if (original == null || target == null || replacement == null) {
21              throw new NullPointerException("original, target e replacement non devono essere null");
22          }
23 1        if (target.isEmpty()) {
24              throw new IllegalArgumentException("target non deve essere vuoto");
25          }
26
27          // Usa un StringBuilder per costruire il risultato
28          StringBuilder result = new StringBuilder();
29          int start = 0;
30          int index;
31
32          // Cerca tutte le occorrenze di target e le sostituisce con replacement
33 1        while ((index = original.indexOf(target, start)) != -1) {
34              result.append(original, start, index); // Aggiunge la parte prima di target
35              result.append(replacement);           // Aggiunge replacement
36 1            start = index + target.length();      // Aggiorna il punto di partenza
37          }
38
39          // Aggiunge la parte rimanente della stringa originale
40          result.append(original.substring(start));
41 1        return result.toString();
42      }
43  }
```

Mutations

```
    1. negated conditional → KILLED
20  2. negated conditional → KILLED
    3. negated conditional → KILLED
23  1. negated conditional → KILLED
33  1. negated conditional → KILLED
36  1. Replaced integer addition with subtraction → TIMED_OUT
41  1. replaced return value with "" for org/task1/StringUtils::replace → KILLED
```

Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

Tests examined

- org.task1.StringUtilsTest.[engine:junit-jupiter]/[class:org.task1.StringUtilsTest]/[test-template:testValuesNull(java.lang.String, java.lang.String, java.lang.String)]/[test-template-invocation:#3] (0 ms)
- org.task1.StringUtilsTest.[engine:junit-jupiter]/[class:org.task1.StringUtilsTest]/[test-template:testValuesNull(java.lang.String, java.lang.String, java.lang.String)]/[test-template-invocation:#4] (0 ms)
- org.task1.StringUtilsTest.[engine:junit-jupiter]/[class:org.task1.StringUtilsTest]/[test-template:testSimpleReplace(java.lang.String, java.lang.String, java.lang.String, java.lang.String)]/[test-template-invocation:#3] (0 ms)
- org.task1.StringUtilsTest.[engine:junit-jupiter]/[class:org.task1.StringUtilsTest]/[test-template:testSimpleReplace(java.lang.String, java.lang.String, java.lang.String, java.lang.String)]/[test-template-invocation:#11] (0 ms)
- org.task1.StringUtilsTest.[engine:junit-jupiter]/[class:org.task1.StringUtilsTest]/[test-template:testSimpleReplace(java.lang.String, java.lang.String, java.lang.String, java.lang.String)]/[test-template-invocation:#8] (0 ms)
- org.task1.StringUtilsTest.[engine:junit-jupiter]/[class:org.task1.StringUtilsTest]/[test-template:testSimpleReplace(java.lang.String, java.lang.String, java.lang.String, java.lang.String)]/[test-template-invocation:#7] (0 ms)
- org.task1.StringUtilsTest.[engine:junit-jupiter]/[class:org.task1.StringUtilsTest]/[test-template:testTargetEmpty(java.lang.String, java.lang.String, java.lang.String)]/[test-template-invocation:#1] (1 ms)
- org.task1.StringUtilsTest.[engine:junit-jupiter]/[class:org.task1.StringUtilsTest]/[test-template:testValuesNull(java.lang.String, java.lang.String, java.lang.String)]/[test-template-invocation:#2] (1 ms)
- org.task1.StringUtilsTest.[engine:junit-jupiter]/[class:org.task1.StringUtilsTest]/[test-template:testSimpleReplace(java.lang.String, java.lang.String, java.lang.String, java.lang.String)]/[test-template-invocation:#2] (1 ms)
- org.task1.StringUtilsTest.[engine:junit-jupiter]/[class:org.task1.StringUtilsTest]/[test-template:testSimpleReplace(java.lang.String, java.lang.String, java.lang.String, java.lang.String)]/[test-template-invocation:#4] (1 ms)
- org.task1.StringUtilsTest.[engine:junit-jupiter]/[class:org.task1.StringUtilsTest]/[test-template:testSimpleReplace(java.lang.String, java.lang.String, java.lang.String, java.lang.String)]/[test-template-invocation:#6] (1 ms)
- org.task1.StringUtilsTest.[engine:junit-jupiter]/[class:org.task1.StringUtilsTest]/[test-template:testSimpleReplace(java.lang.String, java.lang.String, java.lang.String, java.lang.String)]/[test-template-invocation:#10] (1 ms)
- org.task1.StringUtilsTest.[engine:junit-jupiter]/[class:org.task1.StringUtilsTest]/[test-template:testSimpleReplace(java.lang.String, java.lang.String, java.lang.String, java.lang.String)]/[test-template-invocation:#5] (1 ms)
- org.task1.StringUtilsTest.[engine:junit-jupiter]/[class:org.task1.StringUtilsTest]/[test-template:testSimpleReplace(java.lang.String, java.lang.String, java.lang.String, java.lang.String)]/[test-template-invocation:#12] (1 ms)
- org.task1.StringUtilsTest.[engine:junit-jupiter]/[class:org.task1.StringUtilsTest]/[test-template:testValuesNull(java.lang.String, java.lang.String, java.lang.String)]/[test-template-invocation:#1] (5 ms)
- org.task1.StringUtilsTest.[engine:junit-jupiter]/[class:org.task1.StringUtilsTest]/[test-template:testSimpleReplace(java.lang.String, java.lang.String, java.lang.String, java.lang.String)]/[test-template-invocation:#1] (21 ms)
- org.task1.StringUtilsTest.[engine:junit-jupiter]/[class:org.task1.StringUtilsTest]/[test-template:testSimpleReplace(java.lang.String, java.lang.String, java.lang.String, java.lang.String)]/[test-template-invocation:#9] (56 ms)

SPECIFICATION-BASED TESTING

TASK 2

1. Specification Based Testing - Black Box

```java
package org.task2;

public class BMIUtils {  3 usages
    /**
     * Calcola l'Indice di Massa Corporea (BMI) dato il peso e l'altezza.
     *
     * @param peso    Peso in chilogrammi.
     * @param altezza Altezza in metri.
     * @return Il BMI calcolato e arrotondato a due decimali.
     * @throws IllegalArgumentException se il peso è <= 1.50 o altezza è <= 0.40 i valori non sono realistici.
     */
    public static double calcolaBMI(double peso, double altezza) {  3 usages
        // Controllo dei valori non validi per peso e altezza
        if (peso <= 1.50 || altezza <= 0.40) {
            throw new IllegalArgumentException("Il peso o l'altezza devono essere valori reali");
        }

        // Calcolo del BMI
        double bmi = peso / (altezza * altezza);

        // Limiti per BMI (range ragionevole)
        if (bmi < 10 || bmi > 80) {
            throw new IllegalArgumentException("Il BMI calcolato non è realistico.");
        }

        return Math.ceil(bmi * 100.0) / 100.0;
    }

}
```

1.1. Comprensione delle richieste

1.1.1. Obiettivi

Il metodo **`calcolaBMI()`** è progettato per calcolare **`l'Indice di Massa Corporea (BMI)`** dato un **`peso`** e un' **`altezza`** forniti come parametri. Questa funzione consente di valutare la relazione tra peso e altezza di una persona in modo standardizzato.
È utile in diversi contesti, come la valutazione dello stato di salute, la pianificazione di programmi di fitness o la ricerca medica.

1.1.2. Parametri

I parametri richiesti dal metodo `calcolaBMI()` sono:

Parametro	Descrizione	Valore Ammesso	Eccezioni Lanciate
`peso`	Peso corporeo su cui operare	Non può essere `< 2.40 kg`	IllegalArgumentException
`altezza`	Altezza corporea su cui operare	Non può essere `< 0.40 m`	IllegalArgumentException

1.1.3. Output

Il metodo `calcolaBMI()` restituisce un valore di tipo `double` che rappresenta il risultato del calcolo tra `il peso` e `l'altezza.` | `BMI = (peso/(altezza*altezza)`

1.2. Analizza il comportamento del programma con diversi input

Sono stati eseguiti 3 test in cui sono state verificate le funzionalità principali del metodo.
In particolare sono stato testati i casi in cui:

1. Caso con `valori corretti` e BMI calcolato correttamente
2. Caso in cui il `peso` non sia valido
3. Caso in cui il `BMI` non è valido

```java
package org.task2;

import org.junit.Test;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.DisplayName;

public class BlackBoxTest {  new *

    @DisplayName("Caso semplice in cui tutti i valori sono corretti")  new *
    @Test
    public void correctCaseTest(){
        double peso = 80.00; double altezza = 1.87;
        double bmi = Math.ceil(peso / (altezza * altezza)*100.0)/100.0;
        Assertions.assertEquals(bmi, BMIUtils.calcolaBMI(peso,altezza));
    }

    @DisplayName("Caso in cui il peso non sia valido")  new *
    @Test
    public void throwsEceptionTest(){
        double peso = 0.50; double altezza = 1.87;
        Assertions.assertThrows(IllegalArgumentException.class, () -> BMIUtils.calcolaBMI(peso,altezza));
    }

    @DisplayName("Caso in cui il BMI non è valido")  new *
    @Test
    public void throwsEceptionOnBMITest(){
        double peso = 180.00; double altezza = 1.45;
        Assertions.assertThrows(IllegalArgumentException.class, () -> BMIUtils.calcolaBMI(peso,altezza));
    }
}
```

BlackBoxTest.correctCaseTest	passed	8 ms
BlackBoxTest.throwsEceptionOnBMITest	passed	3 ms
BlackBoxTest.throwsEceptionTest	passed	1 ms

1.3. Esplora gli input, gli output e identifica le partizioni

1.3.1. Input Individuali (classi di Input)

peso:

- peso >= 2.4 kg
- peso < 2.4 kg

altezza:

- altezza >= 0.40 m
- altezza < 0.40 m

1.3.2. Combinazione dei valori di input

Dati 2 insiemi, corrispondenti ai parametri,con:

- `peso` 2 valori
- `altezza` 2 valori

è possibile ottenere un massimo di 4 combinazioni

T	peso	altezza
1	>= 2.4 kg	>= 0.40 m
2	>= 2.4 kg	< 0.40 m
3	< 2.4 kg	>= 0.40 m
4	< 2.4 kg	< 0.40 m

1.3.3. Classi di Output

Le classi di outputs sono:

- `BMI < 10`
- `BMI >= 10 && <= 80`
- `BMI > 80`

1.4. Identificazione dei Boundary Cases

I boundary cases sono:

- `peso`
  a. on point: `peso` >= 2.4 kg
  b. off point: `peso` < 2.4 kg
- `altezza`
  a. on point: `altezza` >= 0.4 m
  b. off point: `altezza` < 0.4 m

1.5. Creazione dei casi di test

In questa fase, l'obiettivo è quello di individuare i test essenziali, facendo una selezione di 4 test totali individuati in precedenza.

I criteri di selezione adottati sono:

- testare i casi eccezionali, come `< 2.4 kg` ed `< 0.4 m`, soltanto una volta, senza combinarli
- testare i casi che evidenziano singolarmente gli `off points`

Quindi, sono stati selezionati il seguente test:

T	Peso	Altezza
2	>= 2.4 kg	< 0.4 m
3	< 2.4 kg	>= 0.4 m

Con i test cases:

- Il test **2** individua l'off point del parametro `altezza`
- Il test **3** individua l'off point del parametro `peso`

Per effettuare un testing efficace, la test suite è completata dal seguente test:

T	Peso	Altezza
1	>= 2.4 kg	>= 0.4 m

In aggiunta ai test case eccezionali abbiamo scelto di eseguire anche i test cases che mostrano gli on point dei parametri come:

- Il test **1** individua l'on point del parametro `altezza` e `peso`

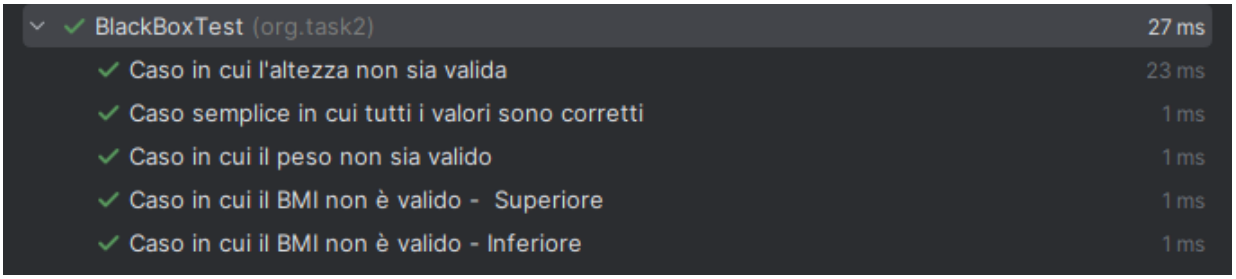
Dopo aver rielaborato la test suite abbiamo rilevato i seguenti test da effettuare:

T	Peso	Altezza
1	>= 2.4kg	>= 0.4 m
2	>= 2.4 kg	< 0.4 m
3	< 2.4 kg	>= 0.4 m

1.6. Automatizzazione della test suite

L'automatizzazione del codice con un approccio **`Specification Based`** sul metodo preso in esame risulta poco esaustivo e, a causa di ciò, abbiamo deciso di implementare logiche di test per garantire il la **`Path Coverage Massima`**.

```java
public class BlackBoxTest {

    @DisplayName("Caso semplice in cui tutti i valori sono corretti")
    @Test
    public void correctCaseTest(){
        double peso = 80.00; double altezza = 1.87;
        double bmi = Math.ceil(peso / (altezza * altezza)*100.0)/100.0;
        Assertions.assertEquals(bmi, BMIUtils.calcolaBMI(peso,altezza));
    }

    @DisplayName("Caso in cui il peso non sia valido")
    @Test
    public void throwsExceptionPesoTest(){
        double peso = 0.50; double altezza = 1.87;
        Assertions.assertThrows(IllegalArgumentException.class, () -> BMIUtils.calcolaBMI(peso,altezza));
    }

    @DisplayName("Caso in cui l'altezza non sia valida")
    @Test
    public void throwsExceptionAltezzaTest(){
        double peso = 55.00; double altezza = 0.25;
        Assertions.assertThrows(IllegalArgumentException.class, () -> BMIUtils.calcolaBMI(peso,altezza));
    }

    @DisplayName("Caso in cui il BMI non è valido -  Superiore")
    @Test
    public void throwsExceptionOnOverBMITest(){
        double peso = 180.00; double altezza = 1.45;
        Assertions.assertThrows(IllegalArgumentException.class, () -> BMIUtils.calcolaBMI(peso,altezza));
    }

    @DisplayName("Caso in cui il BMI non è valido - Inferiore")
    @Test
    public void throwsExceptionOnUnderBMITest(){
        double peso = 30.00; double altezza = 2.15;
        Assertions.assertThrows(IllegalArgumentException.class, () -> BMIUtils.calcolaBMI(peso,altezza));
    }
}
```



1.7. Pianificazione test cases

La progettazione della Test Suite è stata sviluppata in modo sistematico per garantire la copertura delle funzionalità chiave del metodo `BMIUtils.calcolaBMI`. Sono stati identificati i principali scenari d'uso e le combinazioni di input che potrebbero influire sul comportamento della funzione. Ogni caso è stato pianificato per verificare situazioni comuni e potenziali edge case, come l'altezza inferiore al **0.40 m** e il peso inferiore al **2.40 kg**. Questo approccio permette di mantenere la test suite chiara, riutilizzabile e facilmente espandibile per futuri aggiornamenti o nuove esigenze. La pianificazione ha tenuto conto della leggibilità e della modularità dei test per facilitare la manutenzione. Si può visualizzare la pianificazione al seguente link Progettazione Test

STRUCTURAL TESTING AND CODE COVERAGE

Task 2

2. STRUCTURAL TESTING AND CODE COVERAGE

2.1. Esecuzione dei test con analisi della copertura del codice

Con l'applicazione di un approccio step-by-step del **Specification-based testing** e un'analisi approfondita dell'implementazione del metodo, abbiamo eseguito la test suite utilizzando **JaCoCo** come tool per la code coverage. Come evidenziato dai risultati,la copertura della classe testata è pari al 100% sia per il criterio **Line** che per il criterio **Path**.

Coverage Summary for Package: org.task2

Package	Class, %	Method, %	Branch, %	Line, %
org.task2	100% (1/1)	100% (1/1)	100% (8/8)	100% (6/6)

Class	Class, %	Method, %	Branch, %	Line, %
BMIUtils	100% (1/1)	100% (1/1)	100% (8/8)	100% (6/6)

generated on 2024-12-30 18:09

```java
package org.task2;

public class BMIUtils {

    private BMIUtils() { }

    /**
     * Calcola l'Indice di Massa Corporea (BMI) dato il peso e l'altezza.
     *
     * @param peso    Peso in chilogrammi.
     * @param altezza Altezza in metri.
     * @return Il BMI calcolato e arrotondato a due decimali.
     * @throws IllegalArgumentException se il peso è <= 1.50 o altezza è <= 0.40 i valori non sono realistici.
     */
    public static double calcolaBMI(double peso, double altezza) {
        // Controllo dei valori non validi per peso e altezza
        if (peso <= 1.50 || altezza <= 0.40) {
            throw new IllegalArgumentException("Il peso o l'altezza devono essere valori reali");
        }

        // Calcolo del BMI
        double bmi = peso / (altezza * altezza);

        // Limiti per BMI (range ragionevole)
        if (bmi < 10 || bmi > 80) {
            throw new IllegalArgumentException("Il BMI calcolato non è realistico.");
        }

        return Math.ceil(bmi * 100.0) / 100.0;
    }

}
```

2.2. Mutation testing

Il **Mutation Testing** è un metodo di testing strutturale che prevede l'introduzione intenzionale di errori, noti come **"mutanti"**, all'interno del codice al fine di valutare l'efficacia di una suite di test nel rilevarli. Questo approccio si basa sul principio del **Coupling Effect**: se una suite di test è in grado di individuare errori di bassa complessità, si ritiene che possa anche rilevare anomalie più complesse.

Per l'implementazione del Mutation Testing è stato utilizzato **PIT Mutation Testing**, che ha generato 13 mutanti, appartenenti a diverse tipologie, come dettagliato negli screenshot allegati.

Pit Test Coverage Report

Package Summary

org.task2

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
1	86%	6/7	69%	9/13	69%	9/13

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
BMIUtils.java	86%	6/7	69%	9/13	69%	9/13

Report generated by PIT 1.15.8

BMIUtils.java

```
1  package org.task2;
2
3  public class BMIUtils {
4
5      private BMIUtils() { }
6
7      /**
8       * Calcola l'Indice di Massa Corporea (BMI) dato il peso e l'altezza.
9       *
10      * @param peso    Peso in chilogrammi.
11      * @param altezza Altezza in metri.
12      * @return Il BMI calcolato e arrotondato a due decimali.
13      * @throws IllegalArgumentException se il peso è <= 1.50 o altezza è <= 0.40 i valori non sono realistici.
14      */
15     public static double calcolaBMI(double peso, double altezza) {
16         // Controllo dei valori non validi per peso e altezza
17 4       if (peso <= 1.50 || altezza <= 0.40) {
18             throw new IllegalArgumentException("Il peso o l'altezza devono essere valori reali");
19         }
20
21         // Calcolo del BMI
22 2       double bmi = peso / (altezza * altezza);
23
24         // Limiti per BMI (range ragionevole)
25 4       if (bmi < 10 || bmi > 80) {
26             throw new IllegalArgumentException("Il BMI calcolato non è realistico.");
27         }
28
29 3       return Math.ceil(bmi * 100.0) / 100.0;
30     }
31
32 }
```

Mutations

17	1. negated conditional → KILLED 2. negated conditional → KILLED 3. changed conditional boundary → SURVIVED 4. changed conditional boundary → SURVIVED
22	1. Replaced double division with multiplication → KILLED 2. Replaced double multiplication with division → KILLED
25	1. changed conditional boundary → SURVIVED 2. negated conditional → KILLED 3. changed conditional boundary → SURVIVED 4. negated conditional → KILLED
29	1. replaced double return with 0.0d for org/task2/BMIUtils::calcolaBMI → KILLED 2. Replaced double division with multiplication → KILLED 3. Replaced double multiplication with division → KILLED

Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

Tests examined

- org.task2.BlackBoxTest.[engine:junit-jupiter]/[class:org.task2.BlackBoxTest]/[method:correctCaseTest()] (1 ms)
- org.task2.BlackBoxTest.[engine:junit-jupiter]/[class:org.task2.BlackBoxTest]/[method:throwsExceptionOnUnderBMITest()] (0 ms)
- org.task2.BlackBoxTest.[engine:junit-jupiter]/[class:org.task2.BlackBoxTest]/[method:throwsExceptionOnOverBMITest()] (0 ms)
- org.task2.BlackBoxTest.[engine:junit-jupiter]/[class:org.task2.BlackBoxTest]/[method:throwsExceptionAltezzaTest()] (35 ms)
- org.task2.BlackBoxTest.[engine:junit-jupiter]/[class:org.task2.BlackBoxTest]/[method:throwsExceptionPesoTest()] (0 ms)

Report generated by PIT 1.15.8

Fix bug su mutazioni non coperte

Dopo aver analizzato le mutazioni eseguite da **PIT** abbiamo notato che il metodo preso in esame possiede numerosi bug e, a causa di ciò, è stato possibile apportare le seguenti modifiche:

- Da **peso <= 2.40 kg** a **peso < 2.40 kg**
- Abbiamo aggiunto un peso massimo inseribile: **peso <= 500 kg**
- Abbiamo aggiunto un'altezza massima inseribile**: altezza > 2.50 m**
- Da **altezza <= 0.40 m** ad **altezza < 0.40 m**
- Controllo sul **BMI** arrotondato invece del **BMI** calcolato

```java
package org.task2;

public class BMIUtils {  45 usages  checcoconf +1

    private BMIUtils() { }  no usages  checcoconf

    /**
     * Calcola l'Indice di Massa Corporea (BMI) dato il peso e l'altezza.
     *
     * @param peso    Peso in chilogrammi.
     * @param altezza Altezza in metri.
     * @return Il BMI calcolato e arrotondato a due decimali.
     * @throws IllegalArgumentException se il peso è <= 1.50 o altezza è <= 0.40 i valori non sono realistici.
     */
    public static double calcolaBMI(double peso, double altezza) {  13 usages  checcoconf +1
        // Controllo dei valori non validi per peso e altezza
        if (peso < 2.40 || peso > 500) {
            throw new IllegalArgumentException("Il peso deve avere valori reali");
        }

        if (altezza < 0.40 || altezza > 2.50) {
            throw new IllegalArgumentException("L'altezza deve avere valori reali");
        }

        // Calcolo del BMI
        double bmi = peso / (altezza * altezza);
        double ceiledBMI = Math.ceil(bmi * 100.0) / 100.0;

        // Limiti per BMI (range ragionevole)
        if (ceiledBMI < 10 || ceiledBMI > 80) {
            throw new IllegalArgumentException("Il BMI calcolato non è realistico.");
        }
        return ceiledBMI;
    }
}
```

Dopo aver espanso la **Test Suite** con test mirati progettati in base ai risultati del tool **PIT** abbiamo effettuato nuovamente il test del metodo, riportando i risultati sottostanti:

Report

Pit Test Coverage Report

Package Summary

org.task2

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
1	100%	10/10	100%	17/17	100%	17/17

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
BMIUtils.java	100%	10/10	100%	17/17	100%	17/17

Report generated by PIT 1.15.8

Code Coverage

```
1  package org.task2;
2
3  public class BMIUtils {
4
5      private BMIUtils() { }
6
7      /**
8       * Calcola l'Indice di Massa Corporea (BMI) dato il peso e l'altezza.
9       *
10      * @param peso    Peso in chilogrammi.
11      * @param altezza Altezza in metri.
12      * @return Il BMI calcolato e arrotondato a due decimali.
13      * @throws IllegalArgumentException se il peso è <= 1.50 o altezza è <= 0.40 i valori non sono realistici.
14      */
15     public static double calcolaBMI(double peso, double altezza) {
16         // Controllo dei valori non validi per peso e altezza
17         if (peso < 2.40 || peso > 500) {
18             throw new IllegalArgumentException("Il peso deve avere valori reali");
19         }
20
21         if (altezza < 0.40 || altezza > 2.50) {
22             throw new IllegalArgumentException("L'altezza deve avere valori reali");
23         }
24
25         // Calcolo del BMI
26         double bmi = peso / (altezza * altezza);
27         double ceiledBMI = Math.ceil(bmi * 100.0) / 100.0;
28
29         // Limiti per BMI (range ragionevole)
30         if (ceiledBMI < 10 || ceiledBMI > 80) {
31             throw new IllegalArgumentException("Il BMI calcolato non è realistico.");
32         }
33         return ceiledBMI;
34     }
35 }
```

Mutations

17	1. negated conditional → KILLED 2. negated conditional → KILLED 3. changed conditional boundary → KILLED 4. changed conditional boundary → KILLED
21	1. changed conditional boundary → KILLED 2. changed conditional boundary → KILLED 3. negated conditional → KILLED 4. negated conditional → KILLED
26	1. Replaced double multiplication with division → KILLED 2. Replaced double division with multiplication → KILLED
27	1. Replaced double division with multiplication → KILLED 2. Replaced double multiplication with division → KILLED
30	1. changed conditional boundary → KILLED 2. changed conditional boundary → KILLED 3. negated conditional → KILLED 4. negated conditional → KILLED
33	1. replaced double return with 0.0d for org/task2/BMIUtils::calcolaBMI → KILLED

Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

Tests examined

- org.task2.BMIUtilsTest.[engine:junit-jupiter]/[class:org.task2.BMIUtilsTest]/[test-template:testCorrectValues(double, double)]/[test-template-invocation:#3] (0 ms)
- org.task2.BMIUtilsTest.[engine:junit-jupiter]/[class:org.task2.BMIUtilsTest]/[test-template:testCorrectValues(double, double)]/[test-template-invocation:#2] (0 ms)
- org.task2.BMIUtilsTest.[engine:junit-jupiter]/[class:org.task2.BMIUtilsTest]/[test-template:testCorrectValues(double, double)]/[test-template-invocation:#4] (1 ms)
- org.task2.BMIUtilsTest.[engine:junit-jupiter]/[class:org.task2.BMIUtilsTest]/[test-template:testCorrectValues(double, double)]/[test-template-invocation:#6] (1 ms)
- org.task2.BMIUtilsTest.[engine:junit-jupiter]/[class:org.task2.BMIUtilsTest]/[test-template:testCorrectValues(double, double)]/[test-template-invocation:#5] (1 ms)
- org.task2.BMIUtilsTest.[engine:junit-jupiter]/[class:org.task2.BMIUtilsTest]/[test-template:testCorrectValues(double, double)]/[test-template-invocation:#1] (11 ms)
- org.task2.BMIUtilsTest.[engine:junit-jupiter]/[class:org.task2.BMIUtilsTest]/[test-template:testMultipleValues(double, double)]/[test-template-invocation:#4] (0 ms)
- org.task2.BMIUtilsTest.[engine:junit-jupiter]/[class:org.task2.BMIUtilsTest]/[test-template:testMultipleValues(double, double)]/[test-template-invocation:#10] (0 ms)
- org.task2.BMIUtilsTest.[engine:junit-jupiter]/[class:org.task2.BMIUtilsTest]/[test-template:testMultipleValues(double, double)]/[test-template-invocation:#3] (1 ms)
- org.task2.BMIUtilsTest.[engine:junit-jupiter]/[class:org.task2.BMIUtilsTest]/[test-template:testMultipleValues(double, double)]/[test-template-invocation:#9] (0 ms)
- org.task2.BMIUtilsTest.[engine:junit-jupiter]/[class:org.task2.BMIUtilsTest]/[test-template:testMultipleValues(double, double)]/[test-template-invocation:#8] (0 ms)
- org.task2.BMIUtilsTest.[engine:junit-jupiter]/[class:org.task2.BMIUtilsTest]/[test-template:testMultipleValues(double, double)]/[test-template-invocation:#6] (0 ms)
- org.task2.BMIUtilsTest.[engine:junit-jupiter]/[class:org.task2.BMIUtilsTest]/[test-template:testMultipleValues(double, double)]/[test-template-invocation:#2] (1 ms)
- org.task2.BMIUtilsTest.[engine:junit-jupiter]/[class:org.task2.BMIUtilsTest]/[test-template:testMultipleValues(double, double)]/[test-template-invocation:#7] (1 ms)
- org.task2.BMIUtilsTest.[engine:junit-jupiter]/[class:org.task2.BMIUtilsTest]/[test-template:testMultipleValues(double, double)]/[test-template-invocation:#5] (1 ms)
- org.task2.BMIUtilsTest.[engine:junit-jupiter]/[class:org.task2.BMIUtilsTest]/[test-template:testMultipleValues(double, double)]/[test-template-invocation:#1] (51 ms)

Test Suite

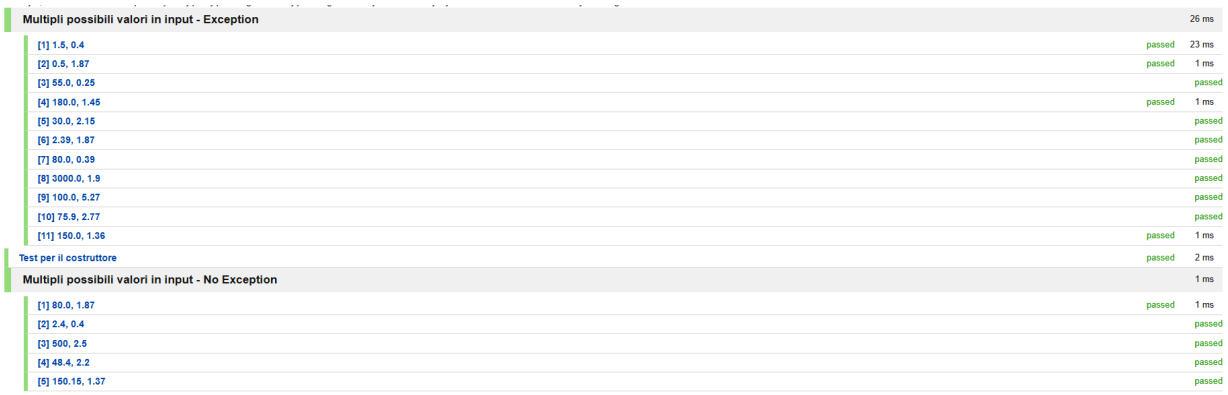
la **`Test Suite`** completa è la seguente:

```java
public class BMIUtilsTest {

    @DisplayName("Test per il costruttore")
    @Test
    void testPrivateConstructor() throws Exception {
        Constructor<BMIUtils> constructor = BMIUtils.class.getDeclaredConstructor();
        Assertions.assertTrue(Modifier.isPrivate(constructor.getModifiers()));
        constructor.setAccessible(true);
        Assertions.assertDoesNotThrow(() -> constructor.newInstance());
    }

    @DisplayName("Multipli possibili valori in input - No Exception")
    @ParameterizedTest
    @MethodSource("provideCorrectValues")
    void testCorrectValues(double peso, double altezza) {
        double bmi = Math.ceil(peso / (altezza * altezza)*100.0)/100.0;
        Assertions.assertEquals(bmi, BMIUtils.calcolaBMI(peso,altezza));
    }

    1 usage
    static Stream<Arguments> provideCorrectValues() {
        return Stream.of(
                // T1 - Valori Validi
                Arguments.of( ...arguments: 80.00, 1.87),
                // T13 - Limite Peso
                Arguments.of( ...arguments: 2.40, 0.40),
                // T14 - Limite Peso Superiore
                Arguments.of( ...arguments: 500, 2.50),
                // T15 Limite Inferiore BMI
                Arguments.of(  ...arguments: 48.4, 2.20),
                // T16 Limite Superiore BMI
                Arguments.of( ...arguments: 150.15, 1.37)
        );
    }
```

```java
    @DisplayName("Multipli possibili valori in input - Exception")
    @ParameterizedTest
    @MethodSource("provideMultipleValues")
    void testMultipleValues(double peso, double altezza) {
        Assertions.assertThrows(IllegalArgumentException.class, () -> BMIUtils.calcolaBMI(peso, altezza));
    }

    1 usage
    static Stream<Arguments> provideMultipleValues() {
        return Stream.of(
                // T12 - Limite Altezza e BMI non realistico
                Arguments.of( ...arguments: 1.50, 0.40),
                // T2 - Caso in cui il peso non sia valido
                Arguments.of( ...arguments: 0.50, 1.87),
                // T3 - Caso in cui l'altezza non sia valida
                Arguments.of( ...arguments: 55.00, 0.25),
                // T4 - Caso in cui il BMI non è valido - Superiore
                Arguments.of( ...arguments: 180.00, 1.45),
                // T5 - Caso in cui il BMI non è valido - Inferiore
                Arguments.of( ...arguments: 30.00, 2.15),
                // Casi di test rilevati nel PIT
                // T8 - Caso sotto limite peso
                Arguments.of( ...arguments: 2.39, 1.87),
                // T9 - Caso sotto limite altezza
                Arguments.of( ...arguments: 80.00, 0.39),
                // T6 - Caso in cui si superi il peso massimo
                Arguments.of( ...arguments: 3000.00,1.90),
                // T7 - Caso in cui si superi l'altezza massima
                Arguments.of( ...arguments: 100.00, 5.27),
                // T10 - BMI Fuori limite inferiore
                Arguments.of( ...arguments: 75.9, 2.77),
                // T11 - BMI Fuori limite superiore
                Arguments.of( ...arguments: 150.0, 1.36)

        );
    }
}
```

Code Coverage

Dopo aver completato la test suite abbiamo eseguito nuovamente i test con la coverage di JaCoCo:



3. PROPERTY-BASED TESTING

3.1. Comprensione delle Task

3.1.1. Obiettivi

L'obiettivo della Funzione `calcolaBMI_valoriValidi()` è verificare che, data una combinazione di peso e altezza, il metodo `calcolaBMI()` calcoli correttamente il `BMI`, generando eccezioni quando necessario. Il test esegue il calcolo con valori di peso compresi tra `2.40 kg e 500.0 kg` e valori di altezza compresi tra `0.40 metri e 2.50 metri`. In questo caso, il test si aspetta che il `BMI` venga calcolato senza errori quando i valori sono nel range corretto (e che il risultato rientri nel range 10-80). Tuttavia, se i valori risultano non realistici (ad esempio, fuori dal range `BMI` accettato), deve essere lanciata un'eccezione `IllegalArgumentException`, che viene gestita correttamente tramite un `blocco try-catch`**.** Questo test conferma che il sistema gestisce correttamente tanto i valori `validi` quanto quelli `non validi`, restituendo il risultato atteso o lanciando l'eccezione appropriata.

3.1.2. Parametri

I parametri della funzione `calcolaBMI_valoriValidi()` sono i seguenti:

- `Peso`: un valore tipo `double` che rappresenta il peso della persona, compreso tra `min (2.40)` e `max(500.0)`kg.
- `Altezza`:un valore di tipo `double` che rappresenta l'altezza della persona, compreso tra `min (0.40)` e `max(2.50)`m.
- Questi parametri vengono passati alla funzione per calcolare il `BMI`, che deve rientrare nell'intervallo valido tra `10` e `80`, se i valori di peso e altezza sono corretti.

3.1.3. Output

Il nostro metodo ritorna un valore numerico di tipo `double` che è il `BMI` risultante del prodotto tra $altezza^2 * peso$. `L'output` riguarda principalmente la `validazione del BMI`.

3.2. Identificazione delle proprietà sulle richieste

- **`Valid`**

  Questa partizione include tutti i casi in cui i parametri di peso e altezza sono validi i quali il BMI viene calcolato correttamente e rientra nel range realistico di 10-80.

  - Esempio: **`peso`**: 70 kg, **`altezza`** 1.75 m
  - Risposta: Un valore di **`BMI`** valido calcolato, ad esempio 22.86

- **`Invalid`**

  Questa partizione include tutti i casi in cui i parametri di **`peso`**, **`altezza`** e **`BMI`** sono fuori dai range validi. In questi casi, il sistema deve lanciare un'eccezione.

- **`IllegalArgumentException`**

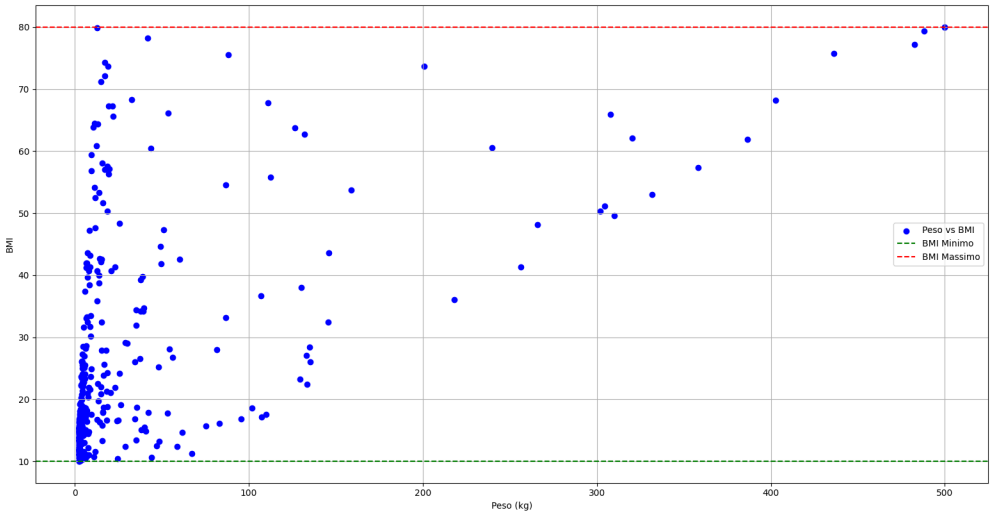
  - Esempio: **`peso`** = 0.0 kg , **`altezza`** = 1.75 m , **`BMI`** = 5 | **`BMI`** = 85
  - Risposta: L'eccezione **`IllegalArgumentException`**.

- **`Boundary Cases`**

  Questa partizione riguarda i casi in cui i dati sono esattamente ai limiti dei range accettabili. Anche in questi casi, il **`BMI`** deve essere calcolato correttamente o l'eccezione deve essere lanciata se i dati sono invalidi.

3.3. Statistiche

3.3.1. Report dei BMI validi in base al peso



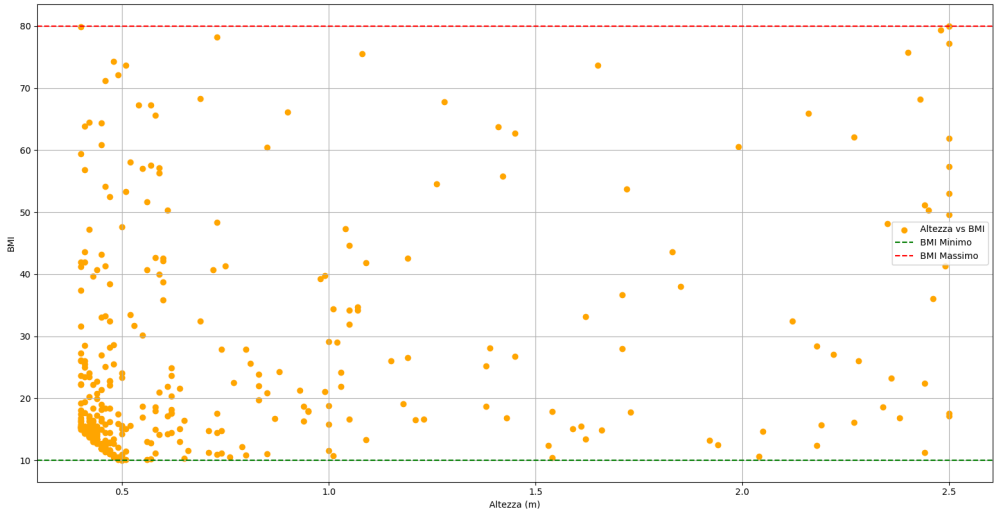
Il grafico rappresenta i risultati di un test di tipo **`property-based`**, eseguito su 1000 simulazioni, con valori di **`peso`** generati tra i limiti estremi consentiti. I dati riflettono un'ampia gamma di combinazioni, esplorando sia i casi estremi che i più comuni.

La distribuzione dei dati riflette come il metodo reagisce in modo positivo qualora i dati in input calcolino un **`BMI`** che raggiunga i limiti impostati.

Questa analisi permette di verificare che il sistema gestisce correttamente sia i valori normali che i casi limite, garantendo stabilità e affidabilità dei risultati.

Come si riesce a notare dal grafico il metodo tende ad effettuare la maggior parte dei test con valori vicini al limite inferiore impostato per il **`peso`**.

3.3.2. Report dei BMI validi in base all'altezza

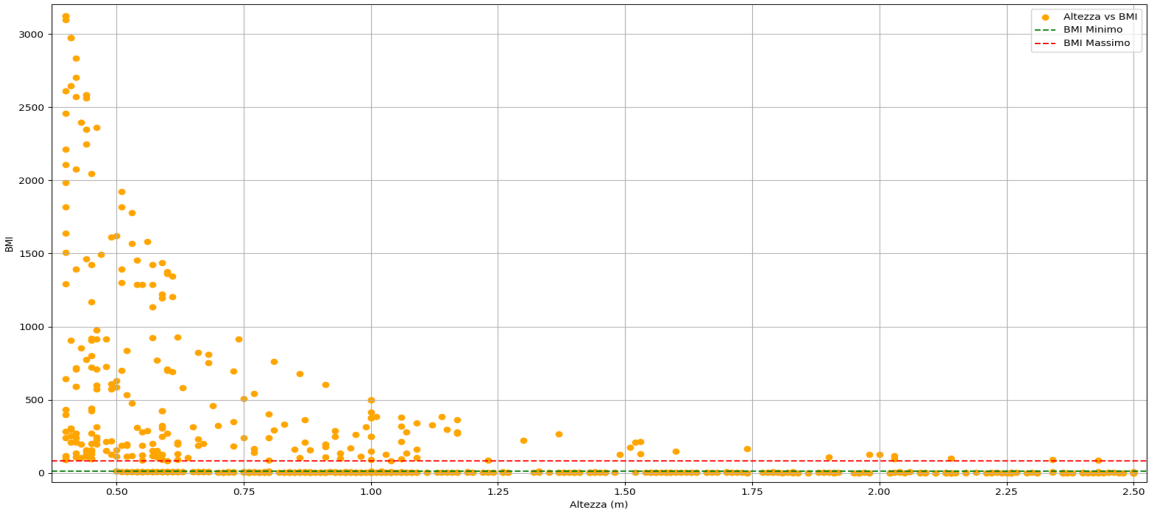


Il grafico rappresenta i risultati di un test di tipo **`property-based`** eseguito su 1000 simulazioni. In questo caso prende in esame **`l'altezza`**.

I dati riflettono una vasta gamma di combinazioni, inclusi casi estremi, per verificare la capacità del sistema di gestire input che portano il **`BMI`** ai limiti impostati, confermando stabilità e affidabilità nei risultati.

Come si può notare i valori tendono ad essere molto vicini al limite inferiore **`dell'altezza`**.

3.3.3. Report dei BMI non validi in base al all'altezza

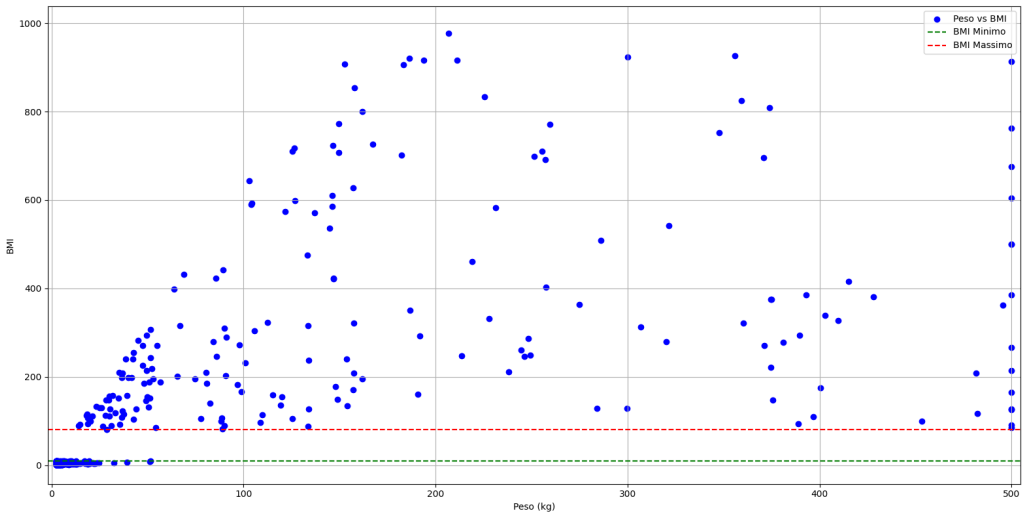


Il grafico rappresenta i risultati di un test di tipo `property-based`, eseguito su 1000 simulazioni, questa volta ponendo in esame i valori errati del `BMI`. Come possiamo notare, pur effettuando i calcoli con valori dell'`altezza` nei range consentiti i `BMI` risultanti tendono ad essere fuori range. Dei 1000 test effettuati circa 300 (test mostrati nel grafico sovrastante) sono fuori range.

Anche qui, la distribuzione dei dati riflette come il metodo reagisce in modo positivo qualora i dati in input calcolino un `BMI` che superino i limiti impostati.

Come si riesce a notare dal grafico, valori molto piccoli dell'`altezza` tendono ad influenzare altamente il risultato del `BMI`, portandolo a superare di molto il range del `BMI`.

3.3.4. Report dei BMI non validi in base al peso



Il grafico rappresenta i risultati di un test di tipo **`property-based`**, eseguito su 1000 simulazioni, questa volta ponendo in esame i valori errati del **`BMI`**. Come possiamo notare, pur effettuando i calcoli con valori del **`peso`** nei range consentiti i **`BMI`** risultanti tendono ad essere fuori range. Dei 1000 test effettuati circa 300 (test mostrati nel grafico sovrastante) sono fuori range.

Anche qui, la distribuzione dei dati riflette come il metodo reagisce in modo positivo qualora i dati in input calcolino un **`BMI`** errato.

Come si riesce a notare dal grafico, valori molto piccoli del **`peso`** tendono ad influenzare altamente il risultato del **`BMI`**.

4. Risorse

4.1. Repository

Le classi testate e i relativi metodi di test sono disponibili all'interno della seguente repository Github: https://github.com/Aletheia/Software_Testing

4.2. Google Sheet

Progettazione Test Pianificazione ed esecuzione dei casi di test

4.3. Tools

Per l'esecuzione della Code Coverage è stato utilizzato JaCoCo

JACOCO Java Code Coverage https://www.eclemma.org/jacoco/

Per l'esecuzione del Mutation Testing è stato utilizzato PIT Mutation Testing.

https://pitest.org/

Per l'esecuzione del Property-based Testing è stato utilizzato jqwik.

https://jqwik.net/

5. Glossario dei termini

5.1. Definizione dei termini

5.1.1. BMI

è un indicatore di densità corporea utile per capire se abbiamo un peso adeguato alla nostra altezza