



**UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO**

Relazione

Integrazione e Test di Sistemi Software A.A 22/23

Gruppo: e_tech - Test

Realizzato da

Bilanzuoli Filippo 736640 f.bilanzuoli1@studenti.uniba.it

Sabatelli Francesco 744409 f.sabatelli11@studenti.uniba.it



INDICE

Homework 1 - Test Case del metodo ListOps()	3
Prima Parte 1	3
1. Understanding the requirements	3
Selezione delle tecniche di test	4
2. Explore what the program does for various inputs	5
3. Explore inputs, outputs and identify partitions	5
4. Identify boundary cases	6
5. Devise test cases	7
6. Automate test cases	9
7. Augment the test suite with creativity and experience	10
9. Code Coverage case test	11
Seconda parte – White-box	12
1. Understanding the requirements	12
2. Explore what the program does for various inputs	12
3. Explore inputs, outputs and identify partitions	12
4. Identify boundary cases	13
5. Devise test cases	13
6. Automate test cases	15
7. Valutazione dei risultati dei test	16
9. Code Coverage case test	18
Homework 2 - Test case del metodo stringProperties()	19
1. Understanding the requirements	19
2. Explore what the program does for various inputs	19
3. Explore inputs, outputs and identify partitions	20
4. Identify boundary cases	21
5. Devise test cases	21
6. Code Coverage case test and Test	24
Homework 3 - Test case del metodo	25
Tests implemented property-based-testing	25
Conclusione e valutazione dei property-based-testing.....	27

Homework 1 - Test Case del metodo ListOps()

Prima Parte 1

1. Understanding the requirements

Nel primo Homework abbiamo testato il metodo `list_intersection()` del file `ListOps.java`. Ecco il codice da cui abbiamo dedotto i test di tipo **black-box**:

```
public class ListOps {

    /** <code>ListUtils</code> should not normally be instantiated. */

    private ListOps() {
    }

    /**
     * Returns a new list containing all elements that are contained in
     * both given
     * lists.
     *
     * @param <E>
     *         the element type
     * @param list1
     *         the first list
     * @param list2
     *         the second list
     * @return the intersection of those two lists
     * @throws NullPointerException
     *         if either list is null
     */

    public static <E> List<E> list_intersection(final List<? extends E> list1, final
List<? extends E> list2) {
        final List<E> result = new ArrayList<>();
        List<? extends E> smaller = list1;
        List<? extends E> larger = list2;
        if (list1.size() > list2.size()) {
            smaller = list2;
            larger = list1;
        }
        final HashSet<E> hashSet = new HashSet<>(smaller);
        for (final E e : larger) {
            if (hashSet.contains(e)) {
                result.add(e);
                hashSet.remove(e);
            }
        }
        return result;
    }
}
```



Per il metodo `list_intersection()` abbiamo definito i seguenti casi di test:

1. Il primo caso di test sarà **`testIntersectionEmptyLists()`**:
questo test viene utilizzato per testare il comportamento del metodo `list_intersection()` quando entrambi gli elenchi di input sono vuoti. Si creano due liste, cioè **`list1`** e **`list2`**, e vengono inizializzati come `ArrayList` vuoti.
Inoltre, viene inizializzato anche il valore **`expected`** passato come `ArrayList`.
Nel valore **`result`** viene assegnato il risultato del metodo `list_intersection()` richiamando le due liste(`list1` e `list2`).
Infine, dovrà essere implementato il metodo `Assertions.assertEquals()`, dove verranno passati `expected` e risultato, verificando che i due valori di output siano uguali;
2. Il secondo caso di test sarà **`testIntersectionSameList()`**:
viene utilizzato per testare il comportamento del metodo `list_intersection()`, quando gli elenchi entrambi gli elenchi di input sono identici.
Si deve dichiarare due liste, cioè `list1` e `list2`, inserendo gli stessi valori. Poi avremo il valore `expected` dove verranno passate le due liste. Nel valore di `result` viene assegnato il risultato del metodo `list_intersection()` e quindi saranno `list1` e `list2`. Infine, viene inserito `Assertions.assertEquals()` con i valori `expected` e `result`.
3. Il terzo caso di test sarà **`testIntersectionDifferentLists()`**:
viene utilizzato per testare il comportamento del metodo `list_intersection()` quando entrambe le liste che si hanno input sono diverse. Si dichiarano le due liste, `list1` e `list2`, inserendo valori diversi. Poi verrà dichiarato il valore `expected` dove verranno inseriti i valori uguali tra le due liste. Mentre nel valore `result` verrà assegnato il risultato del metodo `list_intersection()` delle due liste. Infine, verrà inserito `Assertions.assertEquals()` dove verranno passati il valore `expected` e `result` che risulteranno uguali;
4. Il quarto caso di test sarà **`testIntersectionWithDuplicates()`**:
viene utilizzato per verificare che due liste di input contengono valori duplicati. Nella dichiarazione delle liste, `list1` e `list2`, vengono inseriti valori duplicati. Poi viene inizializzata la variabile `expected` con gli elementi comuni tra `list1` e `list2` senza duplicati. In seguito, viene inizializzata la variabile `result` dove viene inserito il risultato delle due liste. Infine, viene implementato `Assertions.assertEquals()` dove vengono passate le variabili `expected` e `result` che risulteranno uguali;
5. Il quinto caso di test sarà **`testIntersectionWithNull()`**:
verrà utilizzato per testare il comportamento quando due elenchi di input contengono elementi nulli. Poi viene inizializzata la variabile `expected` che contiene gli elementi comuni tra `list1` e `list2`. Successivamente viene inizializzato `result` che contiene il risultato tra `list1` e `list2`. Infine, ci sarà `Assertions.assertEquals()` dove vengono passati i valori `expected` e `result`.
6. Il sesto e ultimo caso di test che è possibile effettuare sul metodo `list_intersection()` sarà **`testIntersectionWithDifferent()`**:
viene utilizzato per testare il comportamento del metodo `list_intersection()` quando due liste di input contengano elementi comuni creando un'unica lista con gli elementi comuni tra loro.

Selezione delle tecniche di test

I test che sono stati spiegati in precedenza sono composti quindi da elenchi vuoti, elenchi diversi, elenchi uguali e diversi, elenchi uguali ed elenchi con valori nulli, testando allo stesso modo diversi elementi negli elenchi.

Abbiamo utilizzato le asserzioni affinché il test restituisca i risultati previsti dai vari scenari.

Però per migliorare ulteriormente la copertura del test, può essere utile aggiungere più casi di test per casi limite o scenari specifici che attualmente non sono coperti, come ad esempio:

- Prova con un elenco vuoto e l'altro non vuoto
- Test con elenchi contenenti solo valori nulli
- Test con liste contenenti un solo elemento



- Eseguire il test con liste molto grandi per assicurarsi che il metodo li gestisca correttamente
- Eseguire il test con liste contenenti elementi di una classe personalizzata per assicurarsi che il metodo li gestisca correttamente.

Infine, è possibile aggiungere in futuro anche altri test per aumentare la copertura del test e garantire ancor di più la correttezza del metodo `list_intersection()`.

2. Explore what the program does for various inputs

Il programma restituisce un nuovo elenco contenente tutti gli elementi presenti in entrambi gli elenchi di input. Innanzitutto, determina quale elenco di input è più piccolo e utilizza i suoi elementi per creare un HashSet. Quindi, scorre gli elementi dell'elenco di input più grande e controlla se ogni elemento è presente nell'HashSet. Se un elemento è presente, viene aggiunto all'elenco dei risultati e rimosso dall'HashSet per evitare duplicati.

Ad esempio, creiamo il test con il seguente metodo inserendo degli input otterremo:

@Test

```
void NewTest(){
    List<Integer> list1 = Arrays.asList(1, 2, 3, 4);
    List<Integer> list2 = Arrays.asList(3, 4, 5, 6);
    List<Integer> intersection = ListOps.list_intersection(list1, list2);
    System.out.println(intersection);
}
```

Dove In output avremo la seguente lista: [3, 4].

3. Explore inputs, outputs and identify partitions

T1: input:

- list1[]
- list2[]

output atteso:

- []

partizione: chiamiamo il metodo `list_intersection` con le liste vuote in input. In seguito, si verifica che il risultato ottenuto sia una lista vuota.

T2: input:

- list1[1,2,3,4]
- list2[1,2,3,4]

output atteso:

- [1, 2, 3, 4]

partizione: chiamiamo il metodo `list_intersection()` con le liste `list1` e `list2` come input. In seguito, si verifica che il risultato ottenuto sia una lista che contiene solo elementi comuni tra le due liste.

T3: input:

- list1: [1, 2, 3, 4]
- list2: [2, 3, 4, 5]

output atteso:

- [2, 3, 4]

partizione: chiamiamo il metodo `list_intersection()` con le liste `list1` e `list2` come input. In seguito, verificare che il risultato ottenuto sia una lista che contiene solo elementi comuni tra le due liste.



T4: input:

- list1: [1, 2, 2, 3, 4, 4]
- list2: [2, 2, 3, 3, 4, 5]

output atteso:

- [2, 3, 4]

partizione: chiamiamo il metodo list_intersection() con le liste list1 e list2 come input. In seguito verificare che il risultato ottenuto sia una lista che contiene elementi comuni tra le due liste che si ripetono.

T5: input:

- list1: [1, 2, null, 3]
- list2: [null, 3, 4, 5]

output atteso:

- [null, 3]

partizione: chiamiamo il metodo list_intersection() con le liste list1 e list2 come input. In seguito verificare che il risultato ottenuto sia una lista che contiene solo elementi comuni tra le due liste, prendendo in considerazione anche gli elementi null.

T6: input:

- list1: [1, 2.0, 3.5]
- list2: [1, 2, 3]

output atteso:

- [1]

partizione: chiamiamo il metodo list_intersection() con le liste list1 e list2. In seguito verificare che il risultato ottenuto sia una lista che contiene solo elementi interi e comuni tra le due liste.

4. Identify boundary cases

Per quanto riguarda i boundary cases possiamo inserire i test case black-box, perché coprono gran parte degli scenari importanti. Tuttavia, ne abbiamo individuati altri di boundary cases che potrebbero essere implementati:

- Large lists:
 - T7: **list1** e **list2** sono liste molto grandi.
 - T8: una tra **list1** e **list2** è una lista molto grande e l'altra molto piccola.
- Lists with Null elements:
 - T9: Sia **list1** che **list2** contengono elementi null, ma l'intersezione non riporta gli elementi di tipo null.
 - T10: **list1** e **list2** contengono elementi null e nell'intersezione riportano gli elementi di tipo null.
 - T11: Una tra **list1** e **list2** contiene elementi null, mentre l'altra no.
- Lists with Different Element Types:
 - T12: **list1** e **list2** contengono elementi di tipo diverso che non sono compatibili.



5. Devise test cases

TITOLO DEL TEST	TEST CASE ID	NUMERO DEL TEST	DATA DEL TEST
testIntersectionEmptyLists	01	01	21-04-2023
DESCRIZIONE DEL TEST	TEST PROGETTATO DA	TEST ESEGUITO DA	DATA DI ESECUZIONE
Questo caso di test verifica se la funzione gestisce correttamente lo scenario quando entrambi gli elenchi di input sono vuoti.	Francesco Sabatelli	Francesco Sabatelli	21-04-2023

ID CASE	DATI INPUT	DATA DEL TEST	RISULTATI ATTESI	RISULTATI EFFETTIVI	PASSA / FALLISCI	COMMENTO
01	list1[] list2[]	21-04-2023	[]	[]	passato	/

TITOLO DEL TEST	TEST CASE ID	NUMERO DEL TEST	DATA DEL TEST
testIntersectionSameList	02	02	21-04-2023
DESCRIZIONE DEL TEST	TEST PROGETTATO DA	TEST ESEGUITO DA	DATA DI ESECUZIONE
Questo caso di test verifica se la funzione restituisce il risultato corretto quando entrambi gli elenchi di input sono uguali.	Francesco Sabatelli	Francesco Sabatelli	21-04-2023

ID CASE	DATI INPUT	DATA DEL TEST	RISULTATI ATTESI	RISULTATI EFFETTIVI	PASSA / FALLISCI	COMMENTO
02	list1[1, 2, 3, 4] list2[1, 2, 3, 4]	21-05-2023	[1, 2, 3, 4]	[1, 2, 3, 4]	passato	/

TITOLO DEL TEST	TEST CASE ID	NUMERO DEL TEST	DATA DEL TEST
testIntersectionDifferentLists	03	03	21-04-2023
DESCRIZIONE DEL TEST	TEST PROGETTATO DA	TEST ESEGUITO DA	DATA DI ESECUZIONE
Questo caso di test garantisce che la funzione identifichi correttamente e restituisca l'intersezione di liste diverse.	Francesco Sabatelli	Francesco Sabatelli	21-04-2023

ID CASE	DATI INPUT	DATA DEL TEST	RISULTATI ATTESI	RISULTATI EFFETTIVI	PASSA / FALLISCI	COMMENTO
03	list1[1, 2, 3, 4] list2[2, 3, 4, 5]	21-04-2023	[2, 3, 4]	[2, 3, 4]	passato	/



TITOLO DEL TEST	TEST CASE ID	NUMERO DEL TEST	DATA DEL TEST
testIntersectionWithDuplicates	04	01	22-04-2023
DESCRIZIONE DEL TEST	TEST PROGETTATO DA	TEST ESEGUITO DA	DATA DI ESECUZIONE
Questo caso di test controlla se la funzione gestisce correttamente gli elementi duplicati e restituisce l'intersezione senza duplicati.	Filippo Bilanzuoli	Filippo Bilanzuoli	22-04-2023

ID CASE	DATI INPUT	DATA DEL TEST	RISULTATI ATTESI	RISULTATI EFFETTIVI	PASSA / FALLISCI	COMMENTO
04	list1[1, 2, 2, 3, 4, 4] list2[2, 2, 3, 3, 4, 5]	22-04-2023	[2, 3, 4]	[2, 3, 4]	passato	/

TITOLO DEL TEST	TEST CASE ID	NUMERO DEL TEST	DATA DEL TEST
testIntersectionWithNull	05	01	22-04-2023
DESCRIZIONE DEL TEST	TEST PROGETTATO DA	TEST ESEGUITO DA	DATA DI ESECUZIONE
Questo caso di test verifica se la funzione gestisce correttamente nulli valori e restituisce l'intersezione di conseguenza.	Filippo Bilanzuoli	Filippo Bilanzuoli	22-04-2023

ID CASE	DATI INPUT	DATA DEL TEST	RISULTATI ATTESI	RISULTATI EFFETTIVI	PASSA / FALLISCI	COMMENTO
05	list1[1, 2, null, 3] list2[null, 3, 4, 5]	22-04-2023	[null, 3]	[null, 3]	passato	/

TITOLO DEL TEST	TEST CASE ID	NUMERO DEL TEST	DATA DEL TEST
testIntersectionWithDifferentTypes	06	01	22-04-2023
DESCRIZIONE DEL TEST	TEST PROGETTATO DA	TEST ESEGUITO DA	DATA DI ESECUZIONE
Questo caso di test verifica se la funzione gestisce correttamente diversi tipi di elenchi di input e restituisce l'intersezione con il tipo appropriato.	Filippo Bilanzuoli	Filippo Bilanzuoli	22-04-2023

ID CASE	DATI INPUT	DATA DEL TEST	RISULTATI ATTESI	RISULTATI EFFETTIVI	PASSA / FALLISCI	COMMENTO
06	list1[1, 2.0, 3.5] list2[1, 2, 3]	22-04-2023	[1]	[1]	passato	/



6. Automate test cases

Abbiamo dunque implementato i test dedotti nella parte iniziale:

```
@Test
void testIntersectionEmptyLists() {
    List<Integer> list1 = new ArrayList<>();
    List<Integer> list2 = new ArrayList<>();
    List<Integer> expected = new ArrayList<>();
    List<Integer> result = ListOps.list_intersection(list1, list2);
    Assertions.assertEquals(expected, result);
}

@Test
void testIntersectionSameList() {
    List<Integer> list1 = Arrays.asList(1, 2, 3, 4);
    List<Integer> list2 = Arrays.asList(1, 2, 3, 4);
    List<Integer> expected = Arrays.asList(1, 2, 3, 4);
    List<Integer> result = ListOps.list_intersection(list1, list2);
    Assertions.assertEquals(expected, result);
}

@Test
void testIntersectionDifferentLists() {
    List<Integer> list1 = Arrays.asList(1, 2, 3, 4);
    List<Integer> list2 = Arrays.asList(2, 3, 4, 5);
    List<Integer> expected = Arrays.asList(2, 3, 4);
    List<Integer> result = ListOps.list_intersection(list1, list2);
    Assertions.assertEquals(expected, result);
}

@Test
void testIntersectionWithDuplicates() {
    List<Integer> list1 = Arrays.asList(1, 2, 2, 3, 4, 4);
    List<Integer> list2 = Arrays.asList(2, 2, 3, 3, 4, 5);
    List<Integer> expected = Arrays.asList(2, 3, 4);
    List<Integer> result = ListOps.list_intersection(list1, list2);
    Assertions.assertEquals(expected, result);
}

@Test
void testIntersectionWithNull() {
    List<Integer> list1 = Arrays.asList(1, 2, null, 3);
    List<Integer> list2 = Arrays.asList(null, 3, 4, 5);
    List<Integer> expected = Arrays.asList(null, 3);
    List<Integer> result = ListOps.list_intersection(list1, list2);
    Assertions.assertEquals(expected, result);
}

@Test
void testIntersectionWithDifferentTypes() {
    List<? extends Number> list1 = Arrays.asList(1, 2.0, 3.5);
    List<Integer> list2 = Arrays.asList(1, 2, 3);
    List<Integer> expected = Arrays.asList(1);
    List<Number> result = ListOps.list_intersection(list1, list2);
    Assertions.assertEquals(expected, result);
}
```



7. Augment the test suite with creativity and experience

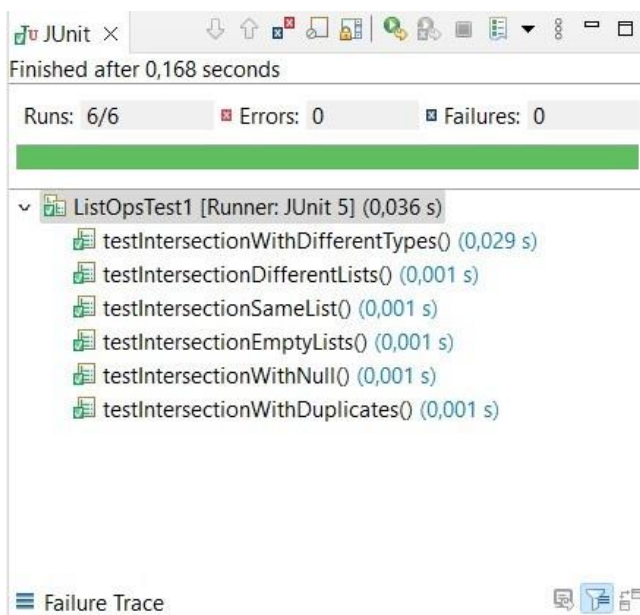
In questo punto abbiamo pensato di includere un altro test case che all'inizio della progettazione non avevamo considerato.

Il test considera la situazione in cui si hanno due liste con nessun elemento in comune.

```
@Test
void testListIntersectionWithNoIntersection() {
    List<Integer> list1 = new ArrayList<>(Arrays.asList(4, 5, 6));
    List<Integer> list2 = new ArrayList<>(Arrays.asList(1, 2, 3));
    List<Integer> result = ListOps.list_intersection(list1, list2);
    assertEquals(0, result.size());
}
```

8. Valutazione dei risultati dei test

Dopo aver progettato e sviluppato dettagliatamente i vari test gli abbiamo eseguiti sul compilatore(Eclipse) e l'esecuzione dei vari test è stata superata al primo tentativo di testing.

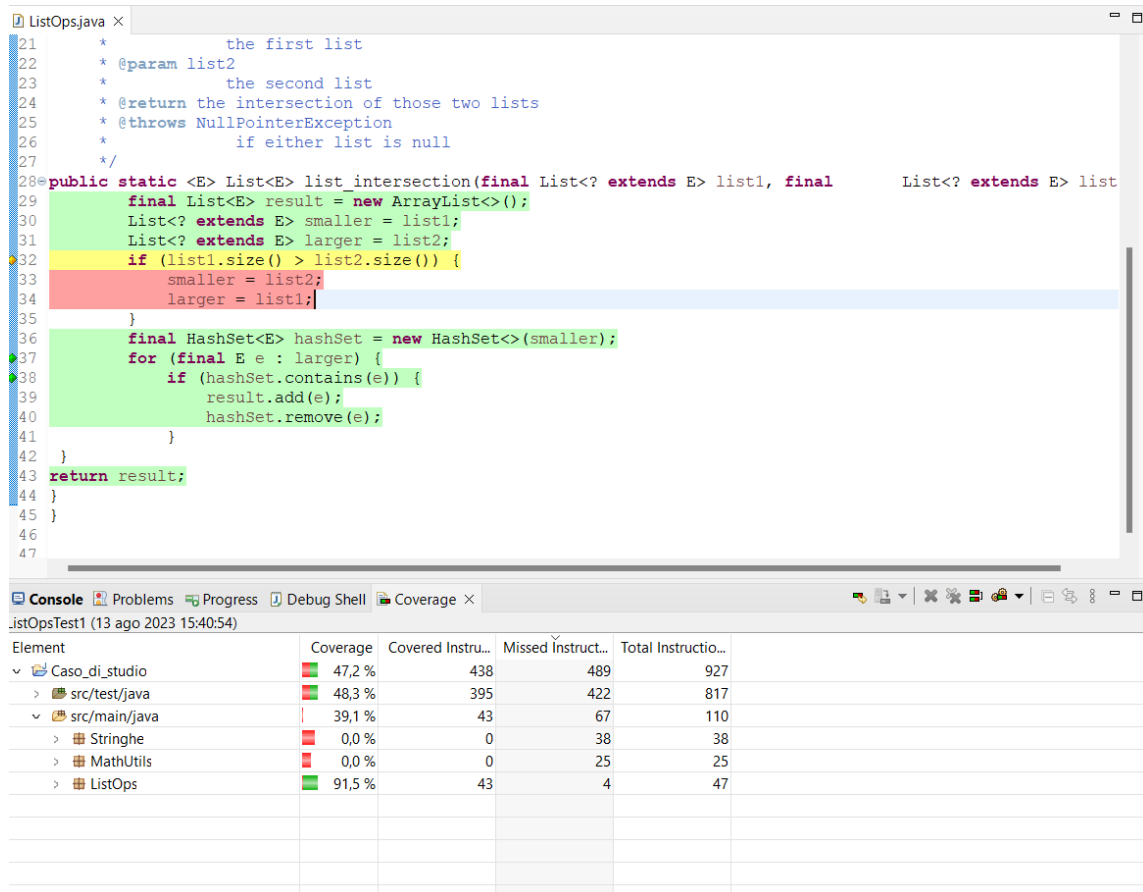


9. Code Coverage case test

La maggior parte delle righe del code coverage sono verdi quindi sono state coperte dai test implementati. Come si può notare dall'immagine presente, le uniche righe non verdi sono le seguenti: 32, 33 e 34.

La riga 32 è gialla, questo indica che è stato parzialmente coperto o che ci sia un problema nella riga. In questo caso il problema è che non è stato testato il caso in cui la grandezza della lista superi il valore Integer.MAX_VALUE. Inoltre, non è testato il caso in cui list2 sia maggiore di list1.

Le righe 33 e 34 non sono state eseguite nei test.



The screenshot shows an IDE window with a Java file named `ListOps.java`. The code implements a `listIntersection` method that takes two lists and returns their intersection. The code is annotated with Javadoc comments. The IDE highlights lines 32, 33, and 34 in yellow, indicating they are not fully covered by tests. Below the code editor, a 'Coverage' tab is active, displaying a table with the following data:

Element	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
Case di studio	47,2 %	438	489	927
> src/test/java	48,3 %	395	422	817
> src/main/java	39,1 %	43	67	110
> Stringhe	0,0 %	0	38	38
> MathUtils	0,0 %	0	25	25
> ListOps	91,5 %	43	4	47



Seconda parte – White-box

1. Understanding the requirements

Nella seconda parte dell'Homework 1 abbiamo creato la parte di test case di tipo **white-box**, revisionando il metodo `list_intersection()`, trovando altri case test per migliorare il testing del metodo preso in considerazione.

2. Explore what the program does for various inputs

Il metodo inizia creando un ArrayList vuoto chiamato `result` per memorizzare gli elementi comuni. Determina gli elenchi più piccoli e più grandi confrontando le loro dimensioni. Questo viene fatto per ottimizzare il processo di iterazione, iterando sull'elenco più piccolo e controllando il contenimento nell'elenco più grande.

Successivamente, viene creato un HashSet che utilizza l'elenco più piccolo. Lo scopo dell' HashSet è controllare in modo efficiente il contenimento degli elementi durante l'iterazione.

Il programma itera su ciascun elemento **e** nell'elenco più grande. Controlla se `hashSet` contiene **e**. In caso affermativo, **e** viene aggiunto all'elenco `result` e quindi rimosso `hashSet` per evitare occorrenze duplicate.

Infine, quando tutti gli elementi nell'elenco più grande sono stati elaborati, viene restituito l'elenco `result`.

3. Explore inputs, outputs and identify partitions

T7: input:

- `list1[null]`
- `list2[1, 2, 3]`

output atteso:

- `[NullPointerException]`

partizione: testare il comportamento quando il primo elenco è null.

T8: input:

- `list1[]`
- `list2[1, 2, 3]`

output atteso:

- `[]`

partizione: testare il comportamento quando uno degli elenchi è vuoto.

T9: input:

- `list1[4, 5, 6]`
- `list2[1, 2, 3]`

output atteso:

- `[]`

partizione: testare il comportamento quando non c'è intersezione tra le liste.

T10: input:

- `list1[1, 2, 3]`
- `list2[2, 3, 4]`

output atteso:

- `[2, 3]`

partizione: testare il comportamento quando c'è un'intersezione tra le liste.

T11: input:

- `list1[1, 2, 2, 3]`
- `list2[2, 2, 3, 4]`

output atteso:

- `[2, 3]`



partizione: testare il comportamento quando ci sono elementi duplicati negli elenchi.

T12: input:

- list1[1, 2, 3, 4, 5]
- list2[3, 4, 5]

output atteso:

- [3, 4, 5]

partizione: in questo test si verifica se entrambe le liste contengono elementi e hanno elementi in comune e l'output dovrebbe essere una lista che contiene gli elementi comuni.

4. Identify boundary cases

Dei test di tipo white-box abbiamo incontrato dei test che sono identificati come “bondary case” e sono:

- **testListIntersectionWithNullList** : questo test riceve una lista nulla con un'eccezione.
- **testListIntersectionWithEmptyList**: questo test ha due liste e riceve come risultato una lista vuota.

5. Devise test cases

TITOLO DEL TEST	TEST CASE ID	NUMERO DEL TEST	DATA DEL TEST
testListIntersectionWithNullList	07	01	28-05-2023
DESCRIZIONE DEL TEST	TEST PROGETTATO DA	TEST ESEGUITO DA	DATA DI ESECUZIONE
Il test verifica il comportamento del metodo list_intersection , quando viene passata una lista null come primo argomento.	Francesco Sabatelli	Francesco Sabatelli	28-05-2023

ID CASE	DATI INPUT	DATA DEL TEST	RISULTATI ATTESI	RISULTATI EFFETTIVI	PASSA / FALLISCI	COMMENTO
07	[null] list2[1,2,3]	28-05-2023	[NullPointerException]	[NullPointerException]	passato	Il test fa parte dei boundary case.

TITOLO DEL TEST	TEST CASE ID	NUMERO DEL TEST	DATA DEL TEST
testListIntersectionWithEmptyList	08	02	28-05-2023
DESCRIZIONE DEL TEST	TEST PROGETTATO DA	TEST ESEGUITO DA	DATA DI ESECUZIONE
Il test verifica che, quando ha due liste(list1 e list2) se c'è una lista vuota restituirà una lista vuota.	Filippo Bilanzuoli	Filippo Bilanzuoli	28-05-2023



ID CASE	DESCRIZIONE PASSO	DATA DEL TEST	RISULTATI ATTESI	RISULTATI EFFETTIVI	PASSA / FALLISCI	COMMENTO
08	List1[] List2[1, 2, 3]	28-05-2023	list []	list []	passato	Il test fa parte dei boundary cases.

TITOLO DEL TEST	TEST CASE ID	NUMERO DEL TEST	DATA DEL TEST
testListIntersectionWithNoIntersection	09	03	28-05-2023
DESCRIZIONE DEL TEST	TEST PROGETTATO DA	TEST ESEGUITO DA	DATA DI ESECUZIONE
Il test verifica che quando si hanno due liste(list1 e list2), che contengono elementi diversi tra loro restituirà una lista vuota.	Francesco Sabatelli	Francesco Sabatelli	28-05-2023

ID CASE	DESCRIZIONE PASSO	DATA DEL TEST	RISULTATI ATTESI	RISULTATI EFFETTIVI	PASSA / FALLISCI	COMMENTO
09	List1[4, 5, 6] List2[1, 2, 3]	28-05-2023	list []	list []	passato	/

TITOLO DEL TEST	TEST CASE ID	NUMERO DEL TEST	DATA DEL TEST
testListIntersectionWithIntersection	10	01	29-05-2023
DESCRIZIONE DEL TEST	TEST PROGETTATO DA	TEST ESEGUITO DA	DATA DI ESECUZIONE
Il test verifica che, quando ci sono elementi in comune tra le due liste di input, la funzione list_intersection() restituisce una lista contenente gli elementi comuni.	Filippo Bilanzuoli	Filippo Bilanzuoli	29-05-2023

ID CASE	DESCRIZIONE PASSO	DATA DEL TEST	RISULTATI ATTESI	RISULTATI EFFETTIVI	PASSA / FALLISCI	COMMENTO
10	list1[1, 2, 3] List2[2, 3, 4]	29-05-2023	list [2, 3]	list [2, 3]	passato	/

TITOLO DEL TEST	TEST CASE ID	NUMERO DEL TEST	DATA DEL TEST
testListIntersectionWithDuplicateElements	11	01	29-05-2023
DESCRIZIONE DEL TEST	TEST PROGETTATO DA	TEST ESEGUITO DA	DATA DI ESECUZIONE
Il test verifica che la funzione list_intersection() gestisca correttamente gli elementi duplicati nelle liste di input.	Filippo Bilanzuoli	Filippo Bilanzuoli	29-05-2023

ID CASE	DESCRIZIONE PASSO	DATA DEL TEST	RISULTATI ATTESI	RISULTATI EFFETTIVI	PASSA / FALLISCI	COMMENTO
11	List1[1, 2, 2, 3] List2[2, 2, 3, 4]	29-05-2023	list [2, 3]	list [2, 3]	passato	/

TITOLO DEL TEST	TEST CASE ID	NUMERO DEL TEST	DATA DEL TEST
testListIntersectionWithLargerFirstList	12	01	13-08-2023
DESCRIZIONE DEL TEST	TEST PROGETTATO DA	TEST ESEGUITO DA	DATA DI ESECUZIONE
Questo test verifica il comportamento della funzione list_intersection con particolari input e verifica se l'output è conforme alle aspettative.	Filippo Bilanzuoli	Filippo Bilanzuoli	13-08-2023

ID CASE	DESCRIZIONE PASSO	DATA DEL TEST	RISULTATI ATTESI	RISULTATI EFFETTIVI	PASSA / FALLISCI	COMMENTO
12	List1[1, 2, 3, 4, 5] List2[3, 4, 5]	13-08-2023	list [3, 4, 5]	list [3, 4, 5]	passato	/

6. Automate test cases

```

@Test
void testListIntersectionWithNullList() {
    assertThrows(NullPointerException.class, () -> {
        ListOps.list_intersection(null, Arrays.asList(1, 2, 3));
    });
}

@Test
void testListIntersectionWithEmptyList() {
    List<Integer> list1 = new ArrayList<>();
    List<Integer> list2 = new ArrayList<>(Arrays.asList(1, 2, 3));
    List<Integer> result = ListOps.list_intersection(list1, list2);
    assertEquals(0, result.size());
}

@Test
void testListIntersectionWithNoIntersection() {
    List<Integer> list1 = new ArrayList<>(Arrays.asList(4, 5, 6));
    List<Integer> list2 = new ArrayList<>(Arrays.asList(1, 2, 3));
    List<Integer> result = ListOps.list_intersection(list1, list2);
    assertEquals(0, result.size());
}

```



```

@Test
void testListIntersectionWithIntersection() {
    List<Integer> list1 = new ArrayList<>(Arrays.asList(1, 2, 3));
    List<Integer> list2 = new ArrayList<>(Arrays.asList(2, 3, 4));
    List<Integer> result = ListOps.list_intersection(list1, list2);
    assertEquals(2, result.size());
    assertEquals(true, result.contains(2));
    assertEquals(true, result.contains(3));
}

@Test
void testListIntersectionWithDuplicateElements() {
    List<Integer> list1 = new ArrayList<>(Arrays.asList(1, 2, 2, 3));
    List<Integer> list2 = new ArrayList<>(Arrays.asList(2, 2, 3, 4));
    List<Integer> result = ListOps.list_intersection(list1, list2);
    assertEquals(2, result.size());
    assertEquals(true, result.contains(2));
    assertEquals(true, result.contains(3));
}

@Test
void testListIntersectionWithLargerFirstList() {
    List<Integer> list1 = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5));
    List<Integer> list2 = new ArrayList<>(Arrays.asList(3, 4, 5));

    List<Integer> intersection = ListOps.list_intersection(list1, list2);

    assertEquals(3, intersection.size());
    assertTrue(intersection.contains(3));
    assertTrue(intersection.contains(4));
    assertTrue(intersection.contains(5));
}

```

7. Valutazione dei risultati dei test

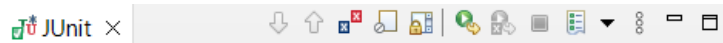
I risultati ottenuti dall'implementazione dei test di tipo white-box sono:

1. **testListIntersectionWithNullList:** Il test verifica se viene generata correttamente un'eccezione `NullPointerException` quando la prima lista è null. Il test ha successo se l'eccezione viene generata correttamente.
2. **testListIntersectionWithEmptyList:** Il test verifica se la funzione `list_intersection()` restituisce una lista vuota quando una delle due liste di input è vuota. Il test ha successo se la lista risultante ha dimensione 0.
3. **testListIntersectionWithNoIntersection:** Il test verifica se la funzione `list_intersection()` restituisce una lista vuota quando non ci sono elementi in comune tra le due liste di input. Il test ha successo se la lista risultante ha dimensione 0.
4. **testListIntersectionWithIntersection:** Il test verifica se la funzione `list_intersection()` restituisce correttamente gli elementi in comune tra le due liste di input. Il test ha successo se la lista risultante ha dimensione 2 e contiene correttamente gli elementi 2 e 3.
5. **testListIntersectionWithDuplicateElements:** Il test verifica se la funzione `list_intersection()` gestisce correttamente gli elementi duplicati nelle liste di input, restituendo solo gli elementi in comune senza duplicati. Il test ha successo se la lista risultante ha dimensione 2 e contiene correttamente gli elementi 2 e 3 senza duplicati.

La valutazione dei test aggiuntivi mostra che la funzione `list_intersection()` gestisce correttamente una serie di casi, inclusi casi limite come liste vuote e valori nulli. La funzione



restituisce risultati corretti quando ci sono intersezioni tra le liste e gestisce correttamente gli elementi duplicati.



Finished after 0,274 seconds

Runs: 6/6 Errors: 0 Failures: 0

▼ ListOpsTest2 [Runner: JUnit 5] (0,053 s)

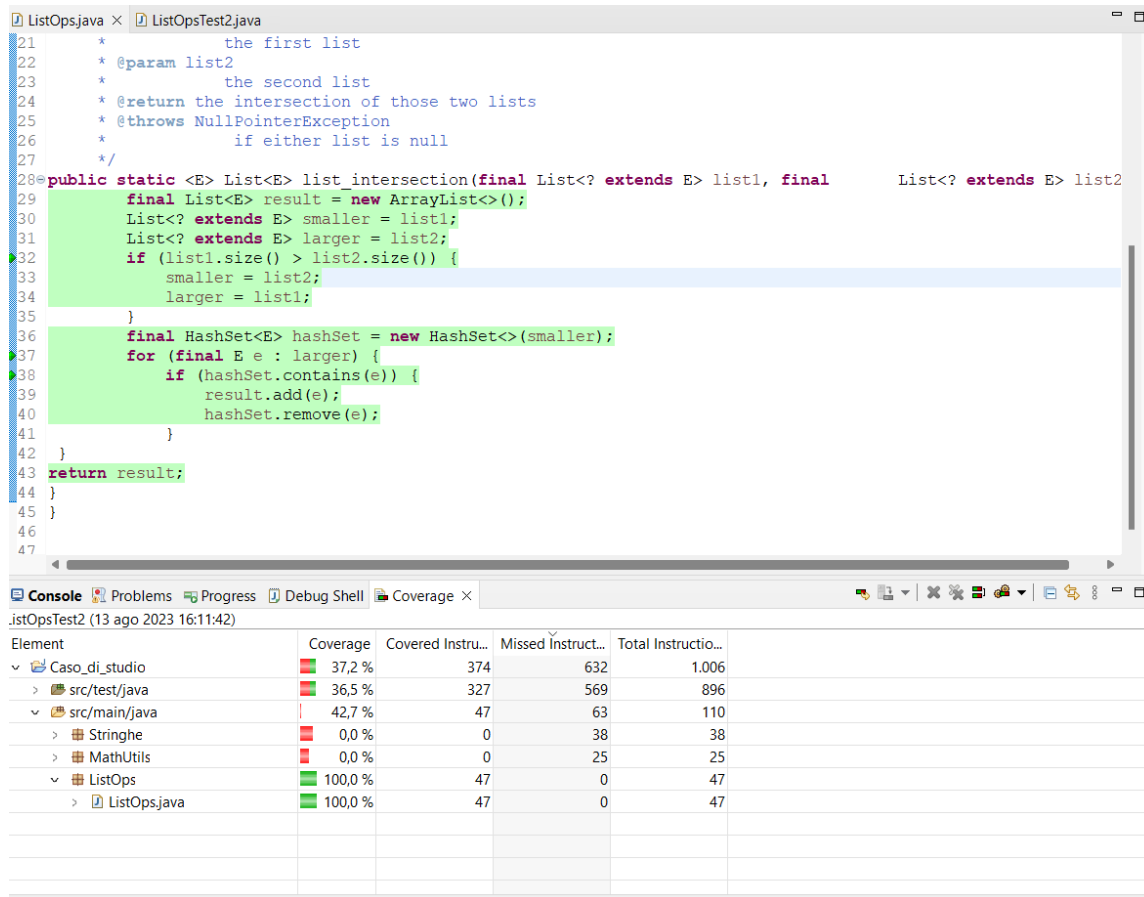
- testListIntersectionWithIntersection() (0,029 s)
- testListIntersectionWithEmptyList() (0,001 s)
- testListIntersectionWithLargerFirstList() (0,013 s)
- testListIntersectionWithNoIntersection() (0,001 s)
- testListIntersectionWithDuplicateElements() (0,001 s)
- testListIntersectionWithNullList() (0,004 s)

≡ Failure Trace



9. Code Coverage case test

Nella seconda parte aggiungendo ulteriori test, abbiamo raggiunto il 100% di code coverage.



The screenshot displays an IDE with two tabs: `ListOps.java` and `ListOpsTest2.java`. The `ListOps.java` file contains a `listIntersection` method that takes two lists and returns their intersection. The code is as follows:

```
21 * the first list
22 * @param list2
23 * the second list
24 * @return the intersection of those two lists
25 * @throws NullPointerException
26 * if either list is null
27 */
28 public static <E> List<E> listIntersection(final List<? extends E> list1, final List<? extends E> list2)
29 {
30     final List<E> result = new ArrayList<>();
31     List<? extends E> smaller = list1;
32     List<? extends E> larger = list2;
33     if (list1.size() > list2.size()) {
34         smaller = list2;
35         larger = list1;
36     }
37     final HashSet<E> hashSet = new HashSet<>(smaller);
38     for (final E e : larger) {
39         if (hashSet.contains(e)) {
40             result.add(e);
41             hashSet.remove(e);
42         }
43     }
44     return result;
45 }
```

Below the code editor, the `Coverage` tab is active, showing a table of coverage data for the project.

Element	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
Caseo_di_studio	37,2 %	374	632	1.006
src/test/java	36,5 %	327	569	896
src/main/java	42,7 %	47	63	110
Stringhe	0,0 %	0	38	38
MathUtils	0,0 %	0	25	25
ListOps	100,0 %	47	0	47
ListOps.java	100,0 %	47	0	47



Homework 2 - Test case del metodo stringProperties()

1. Understanding the requirements

Nel secondo homework abbiamo testato il metodo stringProperties() della classe String.java.

```
public class Stringhe {  
  
    public static String stringProperties(String str) {  
        if (str == null) {  
            return "String is null";  
        } else if (str.isEmpty()) {  
            return "String is empty";  
        } else if (str.length() < 5) {  
            return "String length is less than 5";  
        } else if (str.matches("[0-9]+")) {  
            return "String contains only numbers";  
        } else if (str.matches("[a-zA-Z]+")) {  
            return "String contains only letters";  
        } else if (str.matches("[a-zA-Z0-9]+")) {  
            return "String contains only letters and numbers";  
        } else {  
            return "String contains special characters";  
        }  
    }  
}
```

2. Explore what the program does for various inputs

Il programma della classe Stringhe definisce un metodo chiamato stringProperties() che accetta String come input e restituisce un messaggio che descrive alcune proprietà della stringa. Esploriamo cosa fa il programma per vari input:

1. Se la stringa di input è **null**, restituisce il messaggio "String is null".
2. Se la stringa di input è vuota (ha una lunghezza pari a 0), restituisce il messaggio "String is empty".
3. Se la stringa di input ha una lunghezza inferiore a 5 caratteri, restituisce il messaggio "String length is less than 5".
4. Se la stringa di input contiene solo caratteri numerici (0-9), restituisce il messaggio "String contains only numbers".
5. Se la stringa di input contiene solo caratteri alfabetici (az o AZ), restituisce il messaggio "String contains only letters".
6. Se la stringa di input contiene solo caratteri alfanumerici (lettere o numeri), restituisce il messaggio "String contains only letters and numbers".
7. Se la stringa di input contiene caratteri speciali (caratteri diversi da lettere e numeri), restituisce il messaggio "String contains special characters".

Di seguito abbiamo individuato alcuni esempi per capire il comportamento del programma:

1. **Stringhe.stringProperties(null)** restituisce "String is null" perché la stringa di input è **null**.
2. **Stringhe.stringProperties("")** restituisce "La stringa è vuota" perché la stringa di input è vuota.
3. **Stringhe.stringProperties("abc")** restituisce "La lunghezza della stringa è minore di 5" perché la stringa di input ha una lunghezza di 3, che è minore di 5.
4. **Stringhe.stringProperties("12345")** restituisce "La stringa contiene solo numeri" perché la stringa di input è composta solo da caratteri numerici.



5. **Stringhe.stringProperties("abcdef")** restituisce "La stringa contiene solo lettere" perché la stringa di input è composta solo da caratteri alfabetici.
6. **Stringhe.stringProperties("abc123")** restituisce "La stringa contiene solo lettere e numeri" perché la stringa di input è composta solo da caratteri alfanumerici.
7. **Stringhe.stringProperties("abc@123")** restituisce "La stringa contiene caratteri speciali" perché la stringa di input contiene un carattere speciale (@) insieme a lettere e numeri.

Inoltre, si può notare come il programma utilizza espressioni regolari per determinare i modelli di caratteri nella stringa. Le espressioni regolari utilizzate sono:

- [0-9]+ corrisponde a uno o più caratteri numerici.
- [a-zA-Z]+ corrisponde a uno o più caratteri alfabetici (maiuscolo o minuscolo).
- [a-zA-Z0-9]+ corrisponde a uno o più caratteri alfanumerici (lettere o numeri).

3. Explore inputs, outputs and identify partitions

Possiamo esaminare i vari casi di test, distinguendone gli input, output e le partizioni. Di seguito riportiamo l'analisi dei test sviluppati:

T1: input:

- result[null]

output atteso:

- ["String is null"]

partizione: la stringa di input è **null**.

T2: input:

- result[" "] → Stringa vuota

output atteso:

- ["String is empty"]

partizione: la stringa di input è vuota.

T3: input:

- result["abcd"]

output atteso:

- ["String length is less than 5"]

partizione: la stringa di input ha una lunghezza inferiore a 5.

T4: input:

- result["12345"]

output atteso:

- ["String contains only numbers"]

partizione: la stringa di input contiene solo caratteri numerici.

T5: input:

- result["abcABC"]

output atteso:

- ["String contains only letters"]

partizione: la stringa di input contiene solo caratteri alfabetici.

T6: input:

- result["abc123"]

output atteso:

- ["String contains only letters and numbers"]

partizione: la stringa di input contiene caratteri alfanumerici (lettere e numeri).

T7: input:

- result["abc!@# "]

output atteso:

- ["String contains special characters"]

partizione: la stringa di input contiene caratteri speciali.



4. Identify boundary cases

Dai casi di test forniti, possiamo identificare i seguenti casi limite:

1. **testStringProperties_NullString:**

- Boundary case: la stringa di input è **null**.
Questo verifica la gestione da parte del programma di un input nullo.

2. **testStringProperties_EmptyString:**

- Boundary case: la stringa di input è una stringa vuota (" ").
Questo verifica la gestione da parte del programma di un input vuoto.

3. **testStringProperties_ShortString:**

- Boundary Case: la stringa di input ha una lunghezza minima consentita inferiore a 5 caratteri. In questo caso, la stringa di input è **"abcd"**.
Questo verifica il comportamento del programma quando la lunghezza della stringa di input è al limite inferiore.

5. Devise test cases

TITOLO DEL TEST	TEST CASE ID	NUMERO DEL TEST	DATA DEL TEST
testStringProperties_NullString	01	01	10-06-2023
DESCRIZIONE DEL TEST	TEST PROGETTATO DA	TEST ESEGUITO DA	DATA DI ESECUZIONE
Verifica il comportamento della funzione "stringProperties" quando viene fornita una stringa nulla (null) come input. L'obiettivo del test è verificare se la funzione gestisce correttamente questa situazione restituendo il messaggio "String is null".	Francesco Sabatelli	Francesco Sabatelli	10-06-2023

ID CASE	DESCRIZIONE PASSO	DATA DEL TEST	RISULTATI ATTESI	RISULTATI EFFETTIVI	PASSA / FALLISCI	COMNETO
01	String result = [null]	10-06-2023	[null]	[String is null]	passato	/

TITOLO DEL TEST	TEST CASE ID	NUMERO DEL TEST	DATA DEL TEST
testStringProperties_EmptyString	02	01	10-06-2023
DESCRIZIONE DEL TEST	TEST PROGETTATO DA	TEST ESEGUITO DA	DATA DI ESECUZIONE
Verifica il comportamento della funzione "stringProperties" quando viene fornita una stringa vuota come input.	Francesco Sabatelli	Francesco Sabatelli	10-06-2023



ID CASE	DESCRIZIONE PASSO	DATA DEL TEST	RISULTATI ATTESI	RISULTATI EFFETTIVI	PASSA / FALLISCI	COMMNETO
02	String result= [" "]	10-06-2023	[null]	[String is empty]	passato	/

TITOLO DEL TEST	TEST CASE ID	NUMERO DEL TEST	DATA DEL TEST
testStringProperties_ShortString	03	01	11-06-2023
DESCRIZIONE DEL TEST	TEST PROGETTATO DA	TEST ESEGUITO DA	DATA DI ESECUZIONE
Verifica il comportamento della funzione "stringProperties" quando viene fornita una stringa di lunghezza inferiore a 5 come input.	Francesco Sabatelli	Francesco Sabatelli	11-06-2023

ID CASE	DESCRIZIONE PASSO	DATA DEL TEST	RISULTATI ATTESI	RISULTATI EFFETTIVI	PASSA / FALLISCI	COMMNETO
03	String result = [abcd]	11-06-2023	[String length is less than 5]	[String length is less than 5]	passato	/

TITOLO DEL TEST	TEST CASE ID	NUMERO DEL TEST	DATA DEL TEST
testStringProperties_OnlyNumbers	04	1	11-06-2023
DESCRIZIONE DEL TEST	TEST PROGETTATO DA	TEST ESEGUITO DA	DATA DI ESECUZIONE
Verifica il comportamento della funzione "stringProperties" quando viene fornita una stringa composta solo da numeri come input.	Francesco Sabatelli	Francesco Sabatelli	11-06-2023

ID CASE	DESCRIZIONE PASSO	DATA DEL TEST	RISULTATI ATTESI	RISULTATI EFFETTIVI	PASSA / FALLISCI	COMMNETO
04	String result = [12345]	10-06-2023	[String contains only numbers]	[String contains only numbers]	passato	/



TITOLO DEL TEST	TEST CASE ID	NUMERO DEL TEST	DATA DEL TEST
testStringProperties_OnlyLetters	05	01	12-06-2023
DESCRIZIONE DEL TEST	TEST PROGETTATO DA	TEST ESEGUITO DA	DATA DI ESECUZIONE
Verifica il comportamento della funzione "stringProperties" quando viene fornita una stringa composta solo da lettere.	Filippo Bilanzuoli	Filippo Bilanzuoli	12-06-2023

ID CASE	DESCRIZIONE PASSO	DATA DEL TEST	RISULTATI ATTESI	RISULTATI EFFETTIVI	PASSA / FALLISCI	COMMNETO
05	String result = [abcABC]	12-06-2023	[String contains only letters]	[String contains only letters]	passato	/

TITOLO DEL TEST	TEST CASE ID	NUMERO DEL TEST	DATA DEL TEST
testStringProperties_LettersAndNumbers	06	01	12-06-2023
DESCRIZIONE DEL TEST	TEST PROGETTATO DA	TEST ESEGUITO DA	DATA DI ESECUZIONE
Verifica il comportamento della funzione "stringProperties" quando viene fornita una stringa composta sia da lettere che da numeri.	Filippo Bilanzuoli	Filippo Bilanzuoli	12-06-2023

ID CASE	DESCRIZIONE PASSO	DATA DEL TEST	RISULTATI ATTESI	RISULTATI EFFETTIVI	PASSA / FALLISCI	COMMNETO
06	String result = [abc123]	12-06-2023	[String contains only letters and numbers]	[String contains only letters and numbers]	passato	/

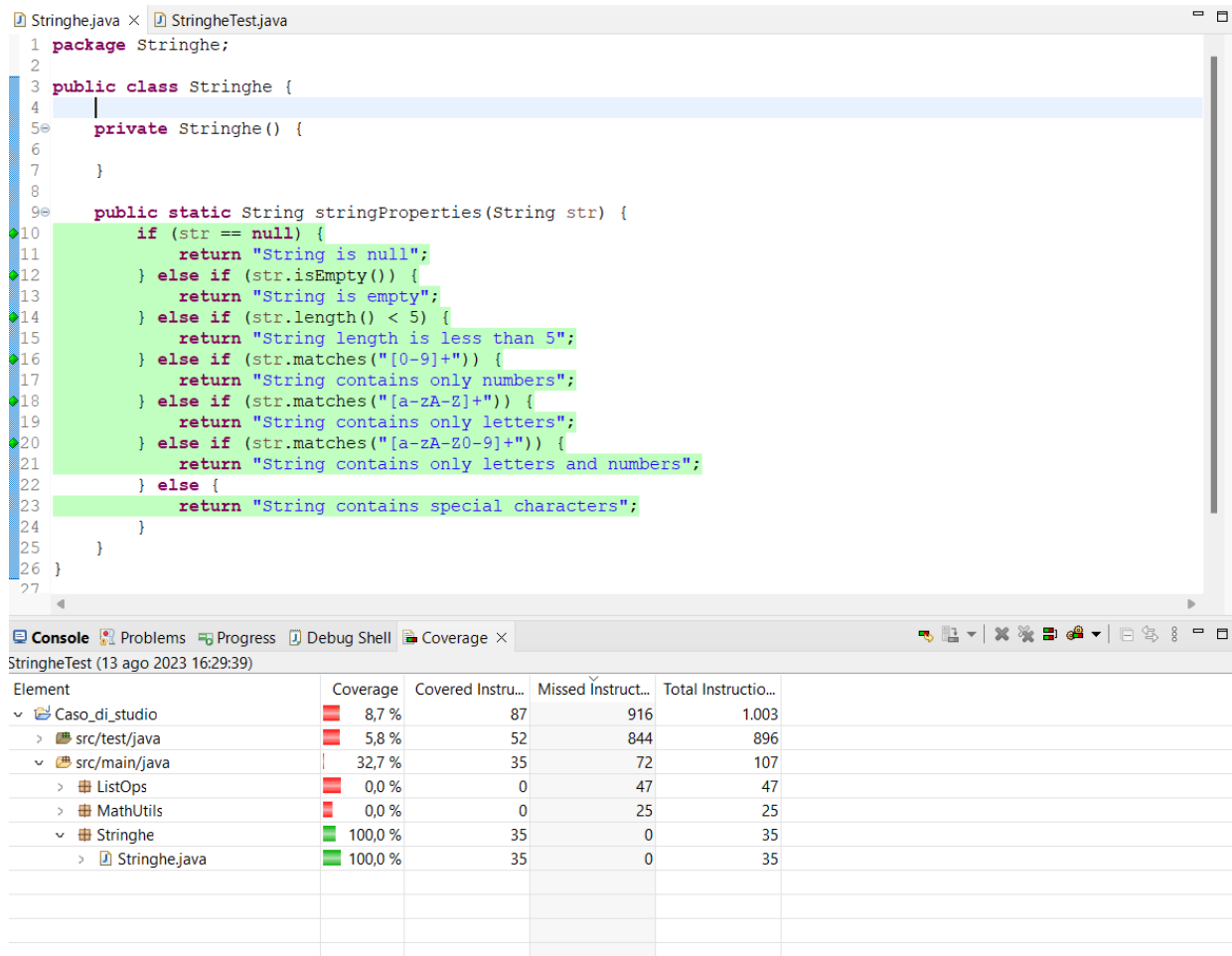
TITOLO DEL TEST	TEST CASE ID	NUMERO DEL TEST	DATA DEL TEST
testStringProperties_SpecialCharacters	07	01	12-06-2023
DESCRIZIONE DEL TEST	TEST PROGETTATO DA	TEST ESEGUITO DA	DATA DI ESECUZIONE
Verifica il comportamento della funzione "stringProperties" quando viene fornita una stringa contenente caratteri speciali come punti esclamativi, chiocciola e cancelletti.	Filippo Bilanzuoli	Filippo Bilanzuoli	12-06-2023

ID CASE	DESCRIZIONE PASSO	DATA DEL TEST	RISULTATI ATTESI	RISULTATI EFFETTIVI	PASSA / FALLISCI	COMMNETO
07	String result = [abc!@#]	12-06-2023	[String contains special characters]	[String contains special characters]	passato	/



6. Code Coverage case test and Test

I test implementati sono stati necessari per avere il 100% di code coverage.



```
1 package Stringhe;
2
3 public class Stringhe {
4     |
5     private Stringhe() {
6     }
7
8
9     public static String stringProperties(String str) {
10        if (str == null) {
11            return "String is null";
12        } else if (str.isEmpty()) {
13            return "String is empty";
14        } else if (str.length() < 5) {
15            return "String length is less than 5";
16        } else if (str.matches("[0-9]+")) {
17            return "String contains only numbers";
18        } else if (str.matches("[a-zA-Z]+")) {
19            return "String contains only letters";
20        } else if (str.matches("[a-zA-Z0-9]+")) {
21            return "String contains only letters and numbers";
22        } else {
23            return "String contains special characters";
24        }
25    }
26 }
27
```

StringheTest (13 ago 2023 16:29:39)

Element	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
✓ Caso_di_studio	8,7 %	87	916	1.003
> src/test/java	5,8 %	52	844	896
✓ src/main/java	32,7 %	35	72	107
> ListOps	0,0 %	0	47	47
> MathUtils	0,0 %	0	25	25
✓ Stringhe	100,0 %	35	0	35
> Stringhe.java	100,0 %	35	0	35



Homework 3 - Test case del metodo

Per l'homework 3 abbiamo scelto di testare la funzione `addition()` della classe `Addition` che calcola il la somma di due numeri di tipo `float` e riporta il risultato in virgola mobile. Di seguito riportiamo la classe scelta, ovvero `Addition`:

```
public class Addition{  
  
    public static float addition(float n1, float n2) {  
        return n1 + n2;  
    }  
}
```

Tests implemented property-based-testing

Di seguito abbiamo testato le proprietà della funzione addizione con i seguenti test method:

```
@Provide  
private Arbitrary<Float> values(){  
    return Arbitraries.oneOf(  
        Arbitraries.floats().between(-50, true, -1, true),  
        Arbitraries.floats().between(1, true, 50, true));  
}
```

Il metodo `values`, avendo annotazione `@Provide`, genera dei dati da usare nei test. È configurato per generare `float` casuali in due intervalli distinti: uno tra -50 e -1 inclusi, e l'altro tra 1 e 50 inclusi. Abbiamo escluso lo zero in quanto elemento neutro dell'addizione che verrà testato in un metodo apposito.

```
@Property  
@Report(Reporting.GENERATED)  
void commutativeProperty(@ForAll("values") float n1, @ForAll("values") float n2) {  
    float function = Addition.addition(n1, n2);  
    float res = n1 + n2;  
    Assertions.assertEquals(function, res);  
    Statistics.collect(function);  
}
```

Il test `commutativeProperty` è progettato per verificare se la funzione di addizione implementata rispetta la proprietà commutativa dell'addizione, che afferma che cambiare l'ordine degli addendi non cambia il risultato. Il test genera coppie casuali di numeri in virgola mobile, ne calcola la somma in entrambi gli ordini e verifica che i due risultati siano uguali.

Inoltre, in tutti i test, abbiamo aggiunto la notazione `@Report` e la funzione `Statistics.collect()` per generare le statistiche dei risultati della funzione.



```

@property
@Report(Reporting.GENERATED)
void associativeProperty(@ForAll("values") float n1, @ForAll("values") float n2,
@ForAll("values") float n3) {
    float function = Addition.addition(Addition.addition(n1, n2), n3);
    float res = n1 + n2 + n3;
    Assertions.assertEquals(function, res);
    Statistics.collect(function);
}

```

Il test associativeProperty ha lo scopo di verificare che l'implementazione dell'addizione rispetti la proprietà associativa, la quale afferma che l'ordine delle operazioni di somma non influisce sul risultato finale. Il test genera tre numeri casuali, esegue due somme in modi diversi (function e res) e verifica che i due risultati siano uguali.

```

@property
@Report(Reporting.GENERATED)
void neutralProperty(@ForAll("values") float n1) {
    float n_z = 0;
    float function = Addition.addition(n1, n_z);
    Assertions.assertEquals(function, n1);
    Statistics.collect(function);
}

```

Il test neutralProperty ha lo scopo di verificare che l'implementazione dell'addizione rispetti la proprietà dell'elemento neutro. L'elemento neutro dell'addizione è lo zero, quindi il test verifica che sommando zero a qualsiasi numero, esso non cambia.

```

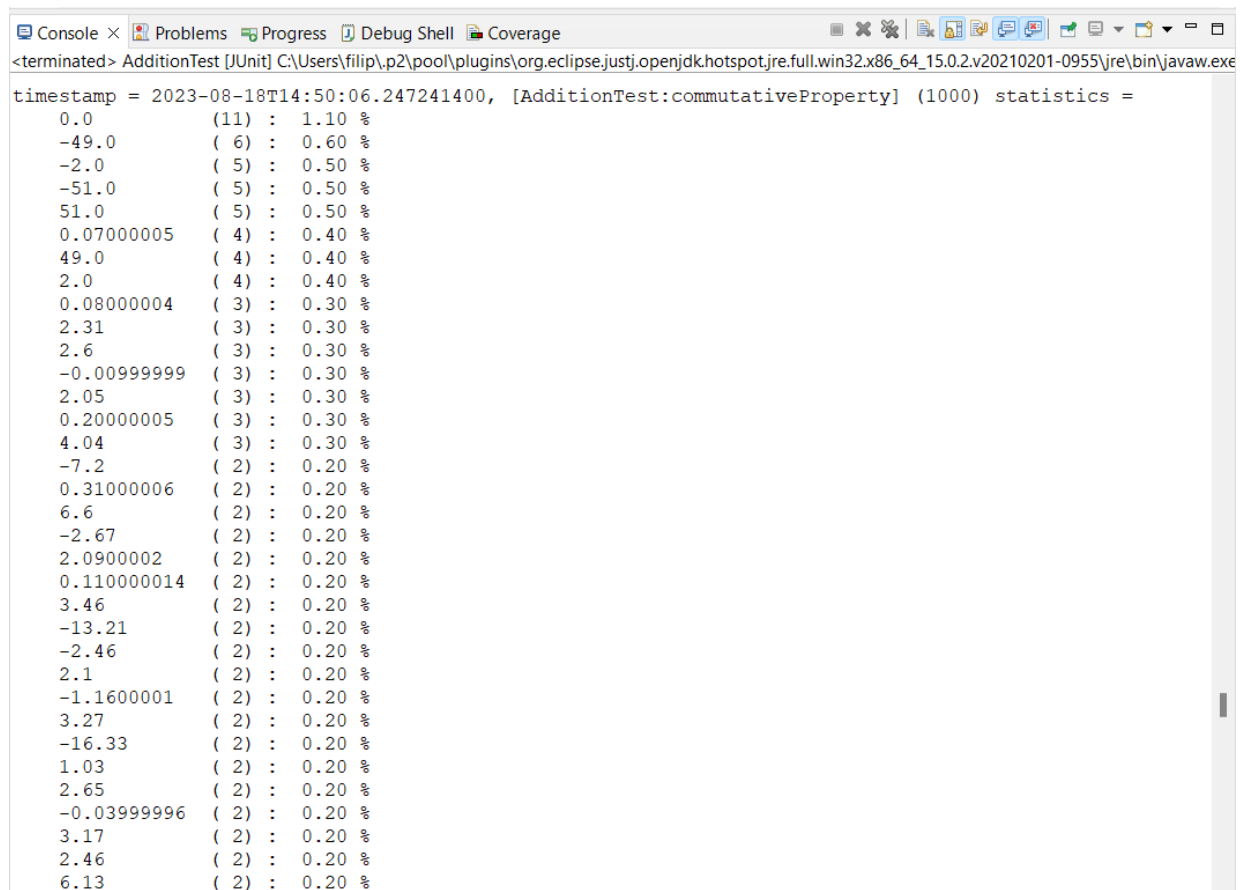
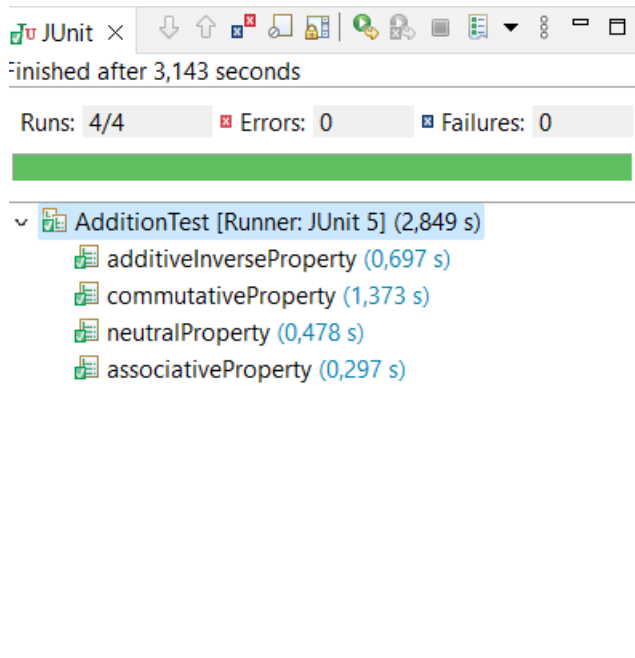
@property
@Report(Reporting.GENERATED)
void additiveInverseProperty(@ForAll("values") float n1) {
    float n_i = n1 * -1;
    float function = Addition.addition(n1, n_i);
    Assertions.assertEquals(function, 0);
    Statistics.collect(function);
}

```

Il test additiveInverseProperty ha lo scopo di verificare che l'implementazione dell'addizione rispetti la proprietà dell'inverso additivo. L'inverso additivo di un numero è quel numero moltiplicato per -1, la cui somma con il numero originale deve dare zero.



Risultati Ottenuti e statistiche dei property-based-testing



```
Console x Problems Progress Debug Shell Coverage
<terminated> AdditionTest [JUnit] C:\Users\filip\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe

timestamp = 2023-08-18T14:47:43.315785500, [AdditionTest:associativeProperty] (1000) statistics =
-1.0      (32) : 3.20 %
-50.0     (26) : 2.60 %
1.0       (26) : 2.60 %
50.0      (14) : 1.40 %
-52.0     (13) : 1.30 %
-48.0     (10) : 1.00 %
-3.0      ( 9) : 0.90 %
48.0      ( 8) : 0.80 %
-101.0    ( 6) : 0.60 %
-99.0     ( 6) : 0.60 %
3.0       ( 4) : 0.40 %
52.0      ( 4) : 0.40 %
15.81     ( 2) : 0.20 %
0.4699998 ( 2) : 0.20 %
-0.73     ( 2) : 0.20 %
-2.06     ( 2) : 0.20 %
-21.83    ( 2) : 0.20 %
1.54      ( 2) : 0.20 %
99.0      ( 2) : 0.20 %
11.179999 ( 2) : 0.20 %
6.3999996 ( 2) : 0.20 %
-3.04     ( 2) : 0.20 %
12.75     ( 2) : 0.20 %
0.99      ( 2) : 0.20 %
-5.3      ( 2) : 0.20 %
37.97     ( 2) : 0.20 %
-1.2      ( 2) : 0.20 %
-1.04     ( 2) : 0.20 %
9.32      ( 2) : 0.20 %
-6.67     ( 2) : 0.20 %
-1.19     ( 2) : 0.20 %
4.76      ( 2) : 0.20 %
-20.27    ( 2) : 0.20 %
-42.99    ( 2) : 0.20 %
-2.16     ( 1) : 0.10 %
```

```
Console x Problems Progress Debug Shell Coverage
<terminated> AdditionTest [JUnit] C:\Users\filip\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe

timestamp = 2023-08-18T14:45:12.681733, [AdditionTest:neutralProperty] (1000) statistics =
-1.0      (39) : 3.90 %
1.0       (33) : 3.30 %
50.0      (20) : 2.00 %
-50.0     (16) : 1.60 %
-1.11     ( 8) : 0.80 %
-1.14     ( 8) : 0.80 %
-1.08     ( 7) : 0.70 %
-1.02     ( 7) : 0.70 %
-1.03     ( 6) : 0.60 %
1.18      ( 5) : 0.50 %
-1.2      ( 5) : 0.50 %
-1.16     ( 5) : 0.50 %
1.17      ( 5) : 0.50 %
1.13      ( 5) : 0.50 %
-1.44     ( 5) : 0.50 %
1.1       ( 5) : 0.50 %
1.12      ( 5) : 0.50 %
2.3       ( 4) : 0.40 %
1.19      ( 4) : 0.40 %
-1.38     ( 4) : 0.40 %
1.37      ( 4) : 0.40 %
1.02      ( 4) : 0.40 %
1.43      ( 4) : 0.40 %
-2.43     ( 4) : 0.40 %
-1.1      ( 4) : 0.40 %
-1.06     ( 4) : 0.40 %
1.14      ( 4) : 0.40 %
-1.24     ( 4) : 0.40 %
1.05      ( 4) : 0.40 %
1.16      ( 4) : 0.40 %
-2.33     ( 4) : 0.40 %
1.08      ( 4) : 0.40 %
1.07      ( 4) : 0.40 %
1.82      ( 3) : 0.30 %
```



```
Console x Problems Progress Debug Shell Coverage
<terminated> AdditionTest [JUnit] C:\Users\filip\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe

timestamp = 2023-08-18T14:53:31.228501700, generated =
  arg0: 50.0

timestamp = 2023-08-18T14:53:31.228501700, generated =
  arg0: 1.57

timestamp = 2023-08-18T14:53:31.229498200, generated =
  arg0: -1.05

timestamp = 2023-08-18T14:53:31.229498200, generated =
  arg0: 1.45

timestamp = 2023-08-18T14:53:31.229498200, generated =
  arg0: 1.0

timestamp = 2023-08-18T14:53:31.230500500, generated =
  arg0: -2.65

timestamp = 2023-08-18T14:53:31.230500500, generated =
  arg0: 1.17

timestamp = 2023-08-18T14:53:31.231497300, generated =
  arg0: -4.57

timestamp = 2023-08-18T14:53:31.257633700, [AdditionTest:additiveInverseProperty] (1000) statistics =
  0.0 (1000) : 100 %
```

Code Coverage

```
1 package Addition;
2
3
4 public class Addition{
5
6     private Addition(){
7
8     }
9     public static float addition(float n1, float n2) {
10         return n1 + n2;
11     }
12 }
13
14
15
```

