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()

Prima Parte 1

1. Understanding the requirements

Nel primo Homework abbiamo testato il metodo list_intersecton() 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 **testInterserctionEmptyLists()**:
   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 **testIntersectioWithDuplicates()**:
   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_interserction 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);
    }
```

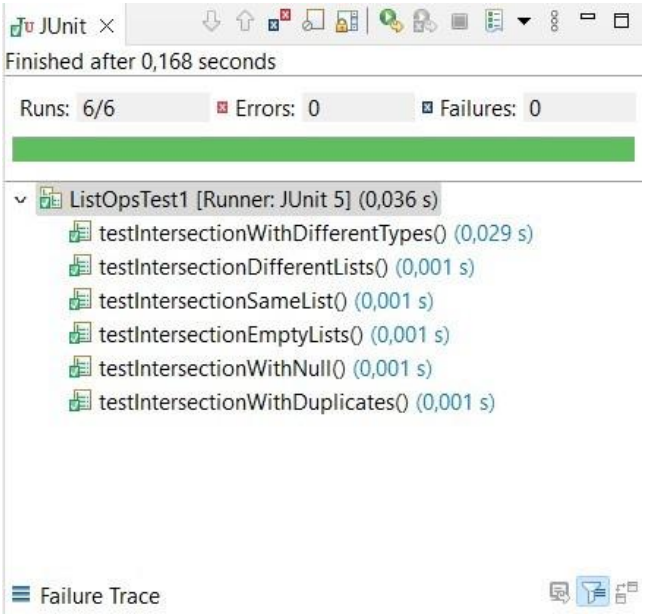
7. Augment the test suite with creativity and experience

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

I primi quattro test controllano il comportamento del metodo quando vengono passati diversi tipi di elenchi in input, inclusi elenchi vuoti, elenchi identici, elenchi diversi ed elenchi con valori null.
Il quinto test invece, controlla il comportamento del metodo quando vengono passati elenchi di tipi diversi.
Per questi test sono state utilizzate tecniche di boundary testing, equivalence partitioning e type testing. Il boundary testing riguarda i test di liste vuote, liste con duplicati e valori null, mentre l'equivalence partitioning riguarda i test di liste che sono identiche o diverse. Infine, il type testing riguarda il test in cui gli elenchi sono di tipi diversi.
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

ListOpsTest1.java

```
1  package ListOps;
2
3  import org.junit.jupiter.api.Assertions;
4  import org.junit.jupiter.api.Test;
5  import java.util.ArrayList;
6  import java.util.Arrays;
7  import java.util.List;
8
9  class ListOpsTest1 {
10
11     //test back-box
12     @Test
13     void testIntersectionEmptyLists() {
14         List<Integer> list1 = new ArrayList<>();
15         List<Integer> list2 = new ArrayList<>();
16         List<Integer> expected = new ArrayList<>();
17
18         List<Integer> result = ListOps.list_intersection(list1, list2);
19
20         Assertions.assertEquals(expected, result);
21     }
22
23     @Test
24     void testIntersectionSameList() {
25         List<Integer> list1 = Arrays.asList(1, 2, 3, 4);
26         List<Integer> list2 = Arrays.asList(1, 2, 3, 4);
27         List<Integer> expected = Arrays.asList(1, 2, 3, 4);
28
29         List<Integer> result = ListOps.list_intersection(list1, list2);
30
31         Assertions.assertEquals(expected, result);
32     }
33
34     @Test
```

Console | Problems | Debug Shell | Coverage

Element	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
ListOps	61,5 %	438	274	712
ListOpsTest2.java	0,0 %	0	270	270
ListOps.java	91,5 %	43	4	47
ListOpsTest1.java	100,0 %	395	0	395
ListOpsTest1	100,0 %	395	0	395

Writable | Smart Insert | 11 : 20 : 220

ListOpsTest1.java

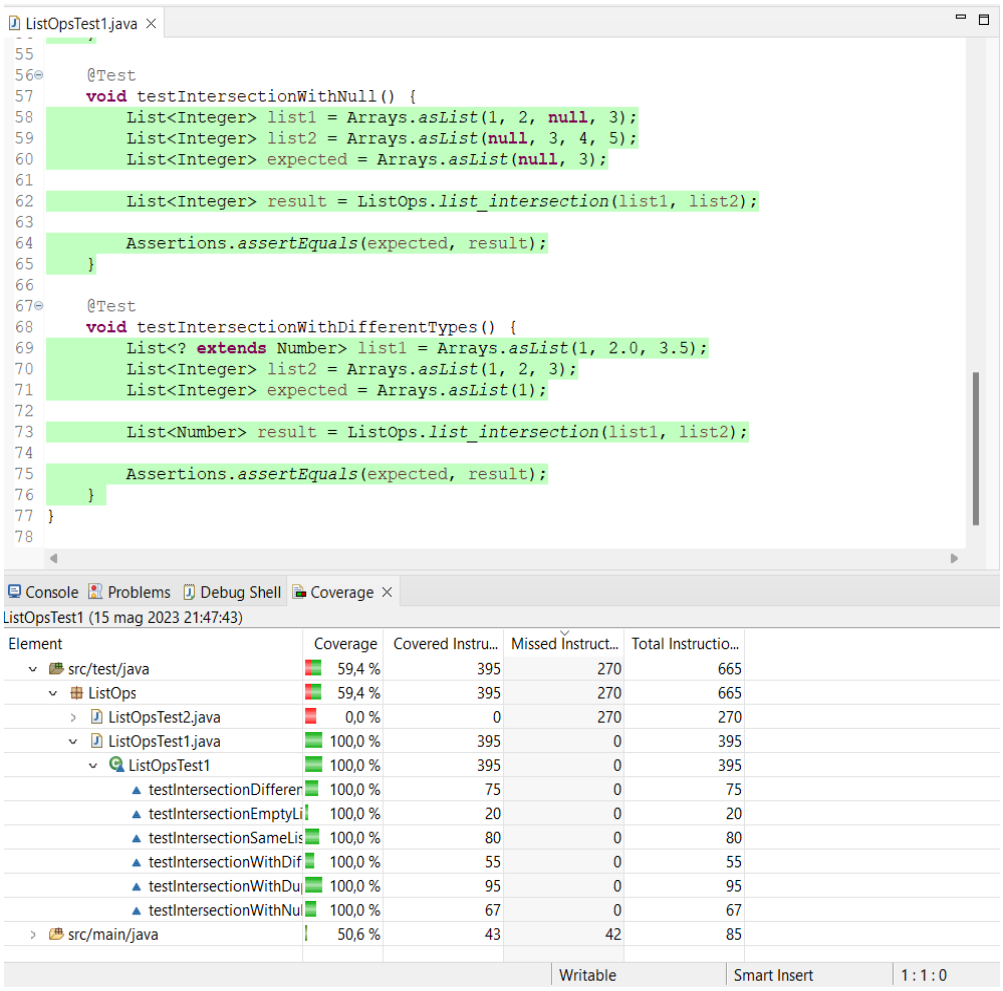
```
33
34     @Test
35     void testIntersectionDifferentLists() {
36         List<Integer> list1 = Arrays.asList(1, 2, 3, 4);
37         List<Integer> list2 = Arrays.asList(2, 3, 4, 5);
38         List<Integer> expected = Arrays.asList(2, 3, 4);
39
40         List<Integer> result = ListOps.list_intersection(list1, list2);
41
42         Assertions.assertEquals(expected, result);
43     }
44
45     @Test
46     void testIntersectionWithDuplicates() {
47         List<Integer> list1 = Arrays.asList(1, 2, 2, 3, 4, 4);
48         List<Integer> list2 = Arrays.asList(2, 2, 3, 3, 4, 5);
49         List<Integer> expected = Arrays.asList(2, 3, 4);
50
51         List<Integer> result = ListOps.list_intersection(list1, list2);
52
53         Assertions.assertEquals(expected, result);
54     }
55
56     @Test
```

Console | Problems | Debug Shell | Coverage

ListOpsTest1 (15 mag 2023 21:47:43)

Element	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
src/test/java	59,4 %	395	270	665
ListOps	59,4 %	395	270	665
ListOpsTest2.java	0,0 %	0	270	270
ListOpsTest1.java	100,0 %	395	0	395
ListOpsTest1	100,0 %	395	0	395
testIntersectionDifferen	100,0 %	75	0	75
testIntersectionEmptyLi	100,0 %	20	0	20
testIntersectionSameLis	100,0 %	80	0	80
testIntersectionWithDif	100,0 %	55	0	55
testIntersectionWithDu	100,0 %	95	0	95
testIntersectionWithNu	100,0 %	67	0	67
src/main/java	50,6 %	43	42	85

Writable | Smart Insert | 1 : 1 : 0



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.

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	/

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));
}
```

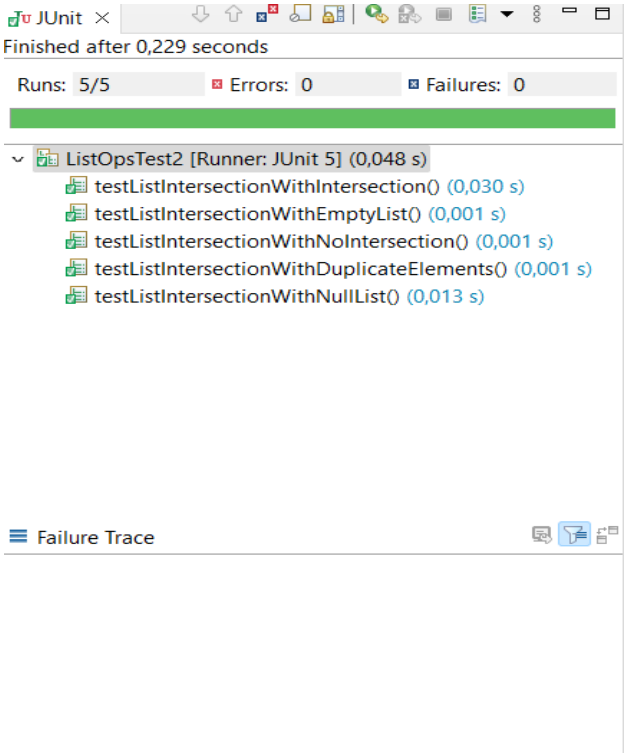
```
@Test
void testListIntersectionWithDuplicateElements() {
	List<Integer> list1 = new ArrayList<>(Arrays.asList(1, 2, 2, 3));
	List<Integer> list2 = new ArrayList<>(Arrays.asList(2, 2, 3, 4));
	List<Integer> result = ListOps.list_intersection(list1, list2);
	assertEquals(2, result.size());
	assertEquals(true,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

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.



9. Code Coverage case test

Nel primo test come si nota è un caso preso in esame con le due liste null, che però come si nota indica che non copre del tutto il codice di testing: Questo non è un errore dovuto dalla non copertura totale del codice, ma è dovuto al fatto che si sta analizzando un test con liste null e lo vede come un test nullo.

ListOpsTest2.java

```
10 import org.junit.jupiter.api.Test;
11
12 class ListOpsTest2 {
13
14     @Test
15      void testListIntersectionWithNullList() {
16           assertThrows(NullPointerException.class, () -> {
17               ListOps.list_intersection(null, Arrays.asList(1, 2, 3));
18           });
19       }
20
21       @Test
22       void testListIntersectionWithEmptyList() {
23           List<Integer> list1 = new ArrayList<>();
24           List<Integer> list2 = new ArrayList<>(Arrays.asList(1, 2, 3));
25           List<Integer> result = ListOps.list_intersection(list1, list2);
26           assertEquals(0, result.size());
27       }
28
29       @Test
30       void testListIntersectionWithNoIntersection() {
31           List<Integer> list1 = new ArrayList<>(Arrays.asList(4, 5, 6));
32           List<Integer> list2 = new ArrayList<>(Arrays.asList(1, 2, 3));
33           List<Integer> result = ListOps.list_intersection(list1, list2);
34           assertEquals(0, result.size());
35       }
36
37       @Test
38       void testListIntersectionWithIntersection() {
```

Console | Problems | Debug Shell | Coverage

Element	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
ListOpsTest2.java	91,9 %	248	22	270
ListOpsTest2	91,9 %	248	22	270
testListIntersectionWith	100,0 %	80	0	80
testListIntersectionWith	100,0 %	36	0	36
testListIntersectionWith	100,0 %	70	0	70
testListIntersectionWith	100,0 %	54	0	54
testListIntersectionWith	100,0 %	5	0	5
Stringhe	0,0 %	0	52	52
src/main/java	50,6 %	43	42	85

ListOpsTest2.java

```
30          void testListIntersectionWithNoIntersection() {
31              List<Integer> list1 = new ArrayList<>(Arrays.asList(4, 5, 6));
32              List<Integer> list2 = new ArrayList<>(Arrays.asList(1, 2, 3));
33              List<Integer> result = ListOps.list_intersection(list1, list2);
34              assertEquals(0, result.size());
35          }
36
37          @Test
38          void testListIntersectionWithIntersection() {
39              List<Integer> list1 = new ArrayList<>(Arrays.asList(1, 2, 3));
40              List<Integer> list2 = new ArrayList<>(Arrays.asList(2, 3, 4));
41              List<Integer> result = ListOps.list_intersection(list1, list2);
42              assertEquals(2, result.size());
43              assertEquals(true, result.contains(2));
44              assertEquals(true, result.contains(3));
45          }
46
47          @Test
48          void testListIntersectionWithDuplicateElements() {
49              List<Integer> list1 = new ArrayList<>(Arrays.asList(1, 2, 2, 3));
50              List<Integer> list2 = new ArrayList<>(Arrays.asList(2, 2, 3, 4));
51              List<Integer> result = ListOps.list_intersection(list1, list2);
52              assertEquals(2, result.size());
53              assertEquals(true, result.contains(2));
54              assertEquals(true, result.contains(3));
55          }
56
57  }
58
```

Console | Problems | Debug Shell | Coverage

Element	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
ListOpsTest2.java	91,9 %	248	22	270
ListOpsTest2	91,9 %	248	22	270
testListIntersectionWith	100,0 %	80	0	80
testListIntersectionWith	100,0 %	36	0	36
testListIntersectionWith	100,0 %	70	0	70
testListIntersectionWith	100,0 %	54	0	54
testListIntersectionWith	100,0 %	5	0	5
Stringhe	0,0 %	0	52	52
src/main/java	50,6 %	43	42	85

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**:
    - Buondary case: la stringa di input è **null**.
      Questo verifica la gestione da parte del programma di un input nullo.
2. **testStringProperties_EmptyString**:
    - Buondary 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.
4. **testStringProperties_OnlyNumbers**:
    - Boundary case: la stringa di input contiene solo caratteri numerici. In questo caso, la stringa di input è **"12345"**.
      Questo verifica il comportamento del programma quando la stringa di input è composta solo da numeri.
5. **testStringProperties_OnlyLetters**:
    - Buondary case: la stringa di input contiene solo caratteri alfabetici. In questo caso, la stringa di input è **"abcd"**.
      Questo verifica il comportamento del programma quando la stringa di input è composta solo da lettere.
6. **testStringProperties_LettersAndNumbers**:
    - Boundary case: la stringa di input contiene sia lettere che numeri. In questo caso, la stringa di input è **"abc123"**.
      Questo verifica il comportamento del programma quando la stringa di input contiene una combinazione di lettere e numeri.
7. **testStringProperties_SpecialCharacters**:
    - Boundary case: la stringa di input contiene caratteri speciali. In questo caso, la stringa di input è **"abc!@#"**.
      Questo verifica il comportamento del programma quando la stringa di input contiene caratteri speciali.

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	COMMNETO
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	/

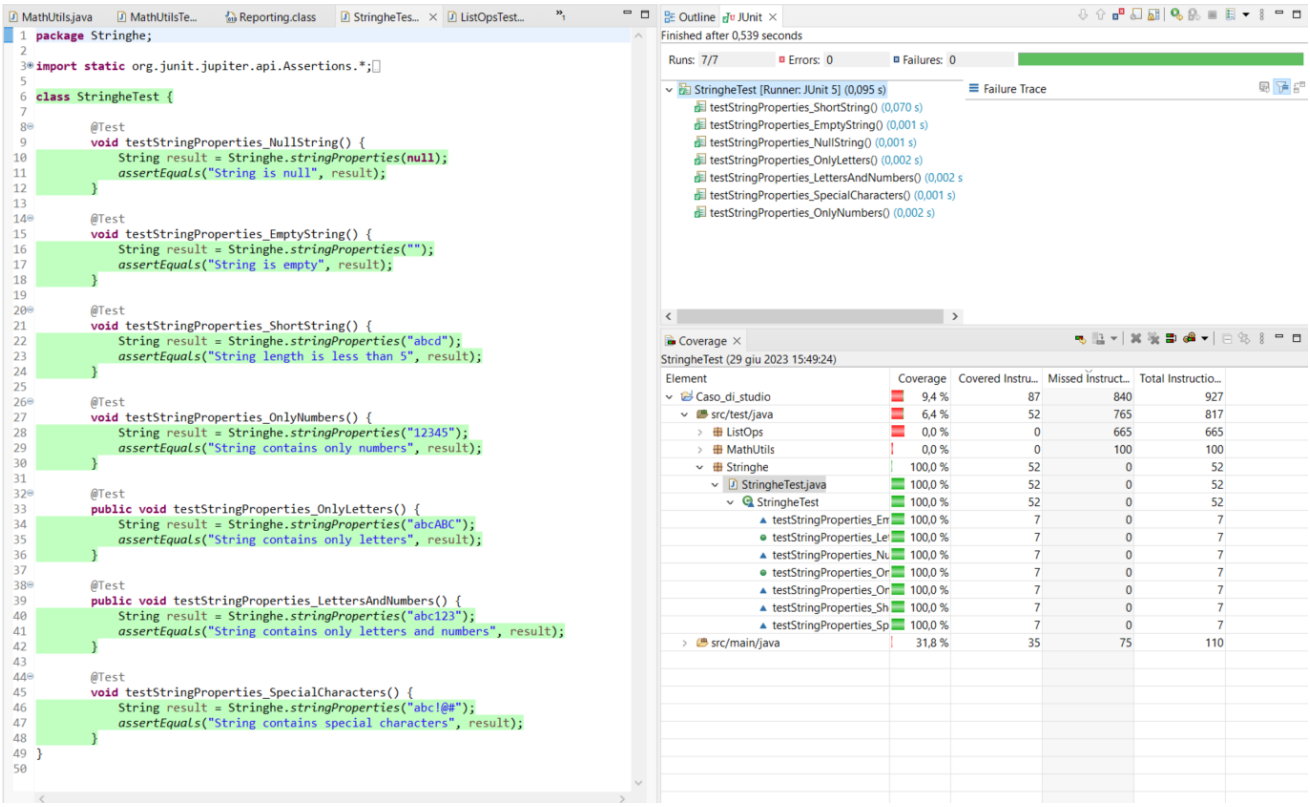
TITOLO DEL TEST	TEST CASE ID	NUMERO DEL TEST	DATA DEL TEST
testStringProperties_LettersAndNumbers	06	01	12-06-2023
DESCRIZIONE DEL TEST	**TEST PROGETTATO DA**	**TEST ESEGUITO DA**	**DATA DI ESECUZIONE**
Verifica il comportamento della funzione "stringProperties" quando viene fornita una stringa composta 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



```java
package Stringhe;

import static org.junit.jupiter.api.Assertions.*;

class StringheTest {

	@Test
	void testStringProperties_NullString() {
		String result = Stringhe.stringProperties(null);
		assertEquals("String is null", result);
	}

	@Test
	void testStringProperties_EmptyString() {
		String result = Stringhe.stringProperties("");
		assertEquals("String is empty", result);
	}

	@Test
	void testStringProperties_ShortString() {
		String result = Stringhe.stringProperties("abcd");
		assertEquals("String length is less than 5", result);
	}

	@Test
	void testStringProperties_OnlyNumbers() {
		String result = Stringhe.stringProperties("12345");
		assertEquals("String contains only numbers", result);
	}

	@Test
	public void testStringProperties_OnlyLetters() {
		String result = Stringhe.stringProperties("abcABC");
		assertEquals("String contains only letters", result);
	}

	@Test
	public void testStringProperties_LettersAndNumbers() {
		String result = Stringhe.stringProperties("abc123");
		assertEquals("String contains only letters and numbers", result);
	}

	@Test
	void testStringProperties_SpecialCharacters() {
		String result = Stringhe.stringProperties("abc!@#");
		assertEquals("String contains special characters", result);
	}
}
```

Finished after 0,539 seconds

Runs: 7/7 Errors: 0 Failures: 0

- StringheTest [Runner: JUnit 5] (0,095 s)
  - testStringProperties_ShortString() (0,070 s)
  - testStringProperties_EmptyString() (0,001 s)
  - testStringProperties_NullString() (0,001 s)
  - testStringProperties_OnlyLetters() (0,002 s)
  - testStringProperties_LettersAndNumbers() (0,002 s
  - testStringProperties_SpecialCharacters() (0,001 s)
  - testStringProperties_OnlyNumbers() (0,002 s)

Failure Trace

Coverage

StringheTest (29 giu 2023 15:49:24)

Element	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
Caso_di_studio	9,4 %	87	840	927
src/test/java	6,4 %	52	765	817
ListOps	0,0 %	0	665	665
MathUtils	0,0 %	0	100	100
Stringhe	100,0 %	52	0	52
StringheTest.java	100,0 %	52	0	52
StringheTest	100,0 %	52	0	52
testStringProperties_Em	100,0 %	7	0	7
testStringProperties_Le	100,0 %	7	0	7
testStringProperties_Nu	100,0 %	7	0	7
testStringProperties_Or	100,0 %	7	0	7
testStringProperties_Or	100,0 %	7	0	7
testStringProperties_Sh	100,0 %	7	0	7
testStringProperties_Sp	100,0 %	7	0	7
src/main/java	31,8 %	35	75	110

Homework 3 - Test case del metodo

Per l'homework 3 abbiamo scelto di testare la funzione factorial della classe MathUtils che calcola il fattoriale di un numero intero positivo. Di seguiti riportiamo la classe scelta, ovvero MathUtils:

```
public class MathUtils {
    /**
     * Calculates the factorial of a given number.
     *
     * @param n the number
     * @return the factorial of the number
     * @throws IllegalArgumentException if the number is negative
     */
    public static int factorial(int n) {
        if (n < 0) {
            throw new IllegalArgumentException("Il numero non può essere
negativo");
        }

        int factorial = 1;
        for (int i = 1; i <= n; i++) {
            factorial *= i;
        }
        return factorial;
    }
}
```

Tests implemented property-based-testing

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

```
@Property
void factorialZero(@ForAll int n) {
      n = 0;
      int factorial = MathUtils.factorial(n);
      Assertions.assertEquals(1, factorial, "Il fattoriale di 0 deve essere 1.");
}
```

1. ***factorialZero*** testa la funzione integrale con un caso limite, cioè con lo 0
viene salvato il risultato della funzione con parametro 0 in una variabile di tipo int, infine viene chiamato un assertEquals per confrontare il risultato atteso, che in questo caso è 1, con il risultato effettivo ritornato dalla funzione. Inoltre, abbiamo anche aggiunto un terzo parametro per visualizzare un messaggio di errore nel caso in cui il confronto non vada a buon fine.

```
@Property
void IncrementingFactorial(@ForAll @IntRange(min = 1, max = 10) int n) {
        long factorialN = MathUtils.factorial(n);
        for (int i = 1; i <= n; i++) {
             long factorialI = MathUtils.factorial(i);
             Assertions.assertTrue(factorialN >= factorialI, "Il fattoriale di un
numero deve essere maggiore o uguale al fattoriale dei numeri precedenti.");
        }
}
```

2. ***incrementingFactorial*** testa la proprietà secondo la quale il fattoriale di un numero intero positivo è sempre maggiore del fattoriale di un qualunque numero intero positivo minore. Come parametri di input da generare abbiamo definito un range di interi che va da 1 a 10. Anche in questo caso, viene salvato il risultato della funzione in una variabile. Successivamente c'è un ciclo for che parte da 1 e termina a n, dove n è il numero generato. Per ogni ciclo si calcola il fattoriale del numero del contatore e viene usato un assertTrue per verificare che il fattoriale di n sia maggiore. Anche in questo caso viene visualizzata una stringa in caso di non successo.

```
@Property
 void factorialPositive(@ForAll @IntRange(min = -15, max = 15) int n) {
      if (n < 0) {
         assertThrows(IllegalArgumentException.class, () ->
MathUtils.factorial(n));
         } else {
                  assertTrue(MathUtils.factorial(n) > 0);
                }
}
```

3. **factorialPositive** testa la proprietà secondo cui il calcolo del fattoriale può essere effettuato solo su un numero intero positivo. In questo metodo abbiamo generato un range di interi che va da -15 a 15. Nel caso in cui il numero generato sia minore di 0 viene lanciata l'eccezione IllegalArgumentException, altrimenti viene calcolato il fattoriale del numero e con un assertTrue si verifica che sia sempre maggiore di 0.

```
@Property
 void pairResult(@ForAll @IntRange(min=2, max=100) int n) {
        assertTrue(MathUtils.factorial(n) % 2 == 0);
 }
```

4. ***pairResult*** verifica la proprietà per cui i fattoriali di tutti i numeri (esclusi 0 e 1) sono pari. Il range di input in questo caso parte da 2 e termina a 100 e nel corpo del metodo c'è un assertTrue che verifica se il resto della divisione del fattoriale di n per 2 sia uguale a 0.

```
@Property
 void multiplesCheck(@ForAll @IntRange(min=1, max=100) int n) {
      int fact = MathUtils.factorial(n);
      boolean flag = true;
        for(int i=1; i == n; i++) {
            if(!(fact % i == 0)) {
                flag = false;
                break;
            }
        }
        assertTrue(flag);
 }
```

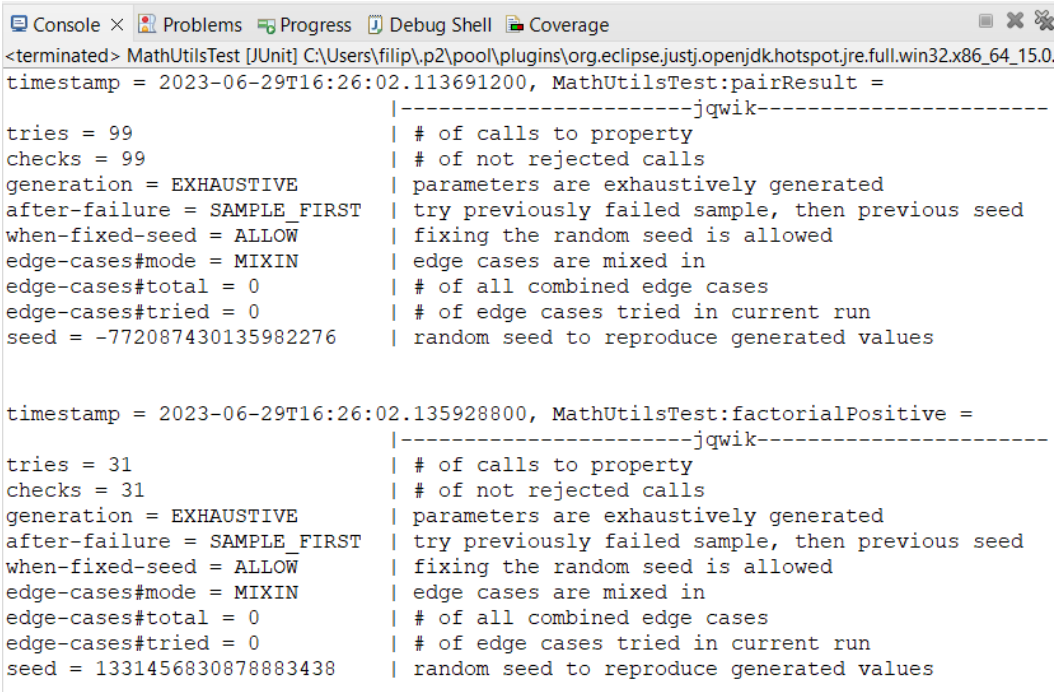
5. ***multipleCheck*** testa che il fattoriale del numero generato n, sia multiplo di tutti i numeri interi positivi minori di n. Il range dei valori di input interi positivi va da 1 a 100. In questo metodo abbiamo usato una variabile booleana per salvare lo stato di ogni confronto. Infatti, subito dopo c'è un ciclo for che parte da 1 e termina a n, con i come valore del contatore. Qui viene controllato che il resto della divisione di n per i sia uguale a 0, mantenendo la variabile booleana nello stato true. In caso contrario, la variabile cambia stato in false e viene interrotto il ciclo for. Infine, il metodo si conclude con un assertTrue che verifica lo stato della variabile booleana.

```
@Property
 void factorialOneTest(@ForAll int n) {
        n = 1;
        int expected = 1;
        int actual = MathUtils.factorial(n);
        assertEquals(expected, actual);
    }
```

6. ***factorialOneTest*** testa la funzione fattoriale quando abbiamo 1 come valore di input. In questo metodo viene calcolato il fattoriale di 1 e viene confrontato, tramite un assertEquals, con il valore atteso, che in questo caso è 1.

Conclusione e valutazione dei property-based-testing

I test basati sulle proprietà sviluppati per la funzione **factorial** in MathUtils utilizzando il property-based testing permettono di verificare comportamenti generali e di individuare eventuali difetti o comportamenti inattesi. Sfruttando la generazione automatica di input casuale, i test coprono una vasta gamma di scenari, contribuendo a migliorare la qualità e l'affidabilità del codice. Inoltre, al lancio dei property-based-testing abbiamo generato il risultato che è visibile nel display in basso e che riportiamo di seguito:



Console × Problems Progress Debug Shell Coverage

<terminated> MathUtilsTest [JUnit] C:\Users\filip\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2

```
timestamp = 2023-06-29T16:26:02.293788800, MathUtilsTest:factorialZero =
                              |-----------------------jqwik-----------------------
tries = 1000                  | # of calls to property
checks = 1000                 | # of not rejected calls
generation = RANDOMIZED       | parameters are randomly generated
after-failure = SAMPLE_FIRST  | try previously failed sample, then previous seed
when-fixed-seed = ALLOW       | fixing the random seed is allowed
edge-cases#mode = MIXIN       | edge cases are mixed in
edge-cases#total = 9          | # of all combined edge cases
edge-cases#tried = 9          | # of edge cases tried in current run
seed = 6318000309076750716    | random seed to reproduce generated values


timestamp = 2023-06-29T16:26:02.309701600, MathUtilsTest:IncrementingFactorial =
                              |-----------------------jqwik-----------------------
tries = 10                    | # of calls to property
checks = 10                   | # of not rejected calls
generation = EXHAUSTIVE       | parameters are exhaustively generated
after-failure = SAMPLE_FIRST  | try previously failed sample, then previous seed
when-fixed-seed = ALLOW       | fixing the random seed is allowed
edge-cases#mode = MIXIN       | edge cases are mixed in
edge-cases#total = 0          | # of all combined edge cases
edge-cases#tried = 0          | # of edge cases tried in current run
seed = -4345672255222548916   | random seed to reproduce generated values
```

Console × Problems Progress Debug Shell Coverage

<terminated> MathUtilsTest [JUnit] C:\Users\filip\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v202

```
seed = -4345672255222548916   | random seed to reproduce generated values


timestamp = 2023-06-29T16:26:02.360600700, MathUtilsTest:factorialOneTest =
                              |-----------------------jqwik-----------------------
tries = 1000                  | # of calls to property
checks = 1000                 | # of not rejected calls
generation = RANDOMIZED       | parameters are randomly generated
after-failure = SAMPLE_FIRST  | try previously failed sample, then previous seed
when-fixed-seed = ALLOW       | fixing the random seed is allowed
edge-cases#mode = MIXIN       | edge cases are mixed in
edge-cases#total = 9          | # of all combined edge cases
edge-cases#tried = 9          | # of edge cases tried in current run
seed = 538078650227135386     | random seed to reproduce generated values


timestamp = 2023-06-29T16:26:02.378708400, MathUtilsTest:multiplesCheck =
                              |-----------------------jqwik-----------------------
tries = 100                   | # of calls to property
checks = 100                  | # of not rejected calls
generation = EXHAUSTIVE       | parameters are exhaustively generated
after-failure = SAMPLE_FIRST  | try previously failed sample, then previous seed
when-fixed-seed = ALLOW       | fixing the random seed is allowed
edge-cases#mode = MIXIN       | edge cases are mixed in
edge-cases#total = 0          | # of all combined edge cases
edge-cases#tried = 0          | # of edge cases tried in current run
seed = -5643016336383317951   | random seed to reproduce generated values
```