Mathematics for Computers

(Discrete Mathematics)

Setty Operations – Set Theory

Project Report

By

Hesham Medhat & Merit Victor

Set Theory Project Report

CS211 Course, Faculty of Engineering, Alexandria University.

Computer & Systems Engineering Department.

To Professor: Sahar M. Ghanem.

and Teaching Assistants:

- Eng/ Reham Osama.

Index

Introduction

This project was assigned in the CS211: Mathematics for Computers course in the date: Friday, September22nd, 2017.

Due on: Saturday,September 30th, 2017.

As the first project of the course, we were assigned this project as teams of two.

This delivery is by:

Hesham Medhat Mahmoud Ahmed Abou-Mousa

&

Merit Victor Ageeb Saweeres Toussy

The source of this project is available on github on this link:

github.com/hesham-medhat/Setty_Operations/

Overview

This program works to perform operations on sets of strings.

The program asks for the universe at first. Gets input sets, and performs operations on the on user’s desire.

Features

In our implementation, the program can:

• find the complement of a set.

•find the union of two sets.

• find the intersection of two sets.

• find the difference of two sets.

• display the output even if it’s phi (empty set).

• display input sets and stores them for operations.

• record the output sets and let the user use them and perform operations on them.

• Graphical user interface (GUI).

Packages

A. Package: application:

This package contains all the files related to the GUI using JavaFx.

○ Controller class: Main.java.

○ Twofxml files: application.fxml and layout.fxml

○ Styling sheets: application.css.

B. Package: Sets:

This package contains all the files related to the sets implementation and operations.

○ Abstract class: Set.java.

○ First child: Universe.java.

○ Second child: Subset.java.

Their names are self-commenting. Go to “UML Diagram” section for visualization.

C. Package: LinkedLists:

This package contains LinkedList implementation. Go to “Data Structures” section.

Data Structures

As we are learned the course Data Structures-1 in the previous semester, we are using our own implementations of Data Structures in this project.

We are using Singly Linked Lists, of our own implementation.

How we used them is described thoroughly in the respective interfaces in the “Functions” section.

Here is the ILinkedList interface:

```
ILinkedList {
        /**
         * Inserts a specified element at the specified position in the
         * list.
         */
        public void add(int index, Object element);

        /** Inserts the specified element at the end of the list. */
        public void add(Object element);
```

```
/** Returns the element at the specified position in this list.
 */
public Object get(int index);

/**
 * Replaces the element at the specified position in this list
 * with the specified element.
 */
public void set(int index, Object element);

/** Removes all of the elements from this list. */
public void clear();

/** Returns true if this list contains no elements. */
publicbooleanisEmpty();

/** Removes the element at the specified position in this list.
*/
public void remove(int index);
```

```
        /** Returns the number of elements in this list. */
        publicint size();

        /**
         * Returns a view of the portion of this list between the
         * specified
         * fromIndex and toIndex, inclusively.
         */
        publicILinkedListsublist(intfromIndex, inttoIndex);

        /**
         * Returns true if this list contains an element with the same
        *value as thespecified element.
         */
        publicboolean contains(Object o);
}
```

Functions

○ The main idea for the functions is that each subset of the universe has a Boolean array of length equal to the length of the universe. Each element in this array represents a true/false value for whether the mirroring element in the universe exists in this subset or not.
○ Set operations become easier this way where intersections are found by AND-ing these bits/values. Similarly union is found by OR-ing. Complements are found by negating.

○ We have also implemented "difference" which is important in set operations.

○ The complexity of these functions is all in Big-O-of (n). This gives them linear time performance.

○ We shall mention the functions in the "Sets" package and that they do.

Set:

• Set(**final** String[] setInput)

Main constructor when reading input.

param setInput the input array of strings.

• Set(**final**SinglyLinkedList list)

Constructor in case the list is ready.

param list : previously built set.

• Set(**final** Universe universeIn, **finalboolean**[] setBoolIn)

Constructor in case we know the boolean set.

param universeIn : universe of the set.

paramsetBoolIn : boolean array of existence of elements from universe.

• **boolean** isUnique(**final** Object element, **final** SinglyLinkedListsetSLL)

Auxiliary function used for detecting whether the input element is unique or a duplicate before adding it to the SLL.

param element : to be added to the list.

param setSLL : the list.

returns true if it is unique and false otherwise.

• **abstract** Set complement()

Finds the rest of the elements in the universe not existing in the set.

returnscomplement of a set Returns null if the output set is empty.

• **abstract** Set difference(Set other)

Finds the set difference with another set.

param other : input set.

returns the difference. Returns null if the output set is empty.

•**abstract** Set intersection(Set other)

Gets the intersection of this set and another.

param other : input set.

returns intersection set. Returns null if the output set is empty.

• **abstract** Set union(Set other)

Gets the union of this set and another.

param other : input set.

returns union set. Returns null if the output set is empty.

•SinglyLinkedListgetSetList()

Getter for setList.

returnssetList as SLL.

Universe:

This class inherits from “Set” class and implements its abstract methods.

• Universe(**final**SinglyLinkedList list)

Constructor in case the list of elements is ready.

param list : previously built list of elements.

• Universe(**final** String[] setInput)

Constructor that passes the setInput as string array.

paramsetInput : in the form of a string array.

Subset:

This class inherits from “Set” class and implements its abstract methods.

• Subset(**final**SinglyLinkedList list, **finalboolean**[] setBoolIn)

Constructor in case we already have the list built.

param list : of elements in the subset.

• Subset(**final** Universe universeIn, **finalboolean**[] setBoolIn)

Constructor in case the boolean array is ready.

paramuniverseIn : universe

paramsetBoolIn : the readySetBool

• Subset(**final** Universe universeIn, **final** String[] setInput)

Constructor that calls the super "Set" constructor to build the SLL of set.

paramuniverseIn : universe as object.

paramsetInput : set content input as string array.

• **boolean**[] getSetBool()

Getter for setBool.

returnssetBool which acts as a bit map for the existence of the elements in this set in the universe that it belongs to.

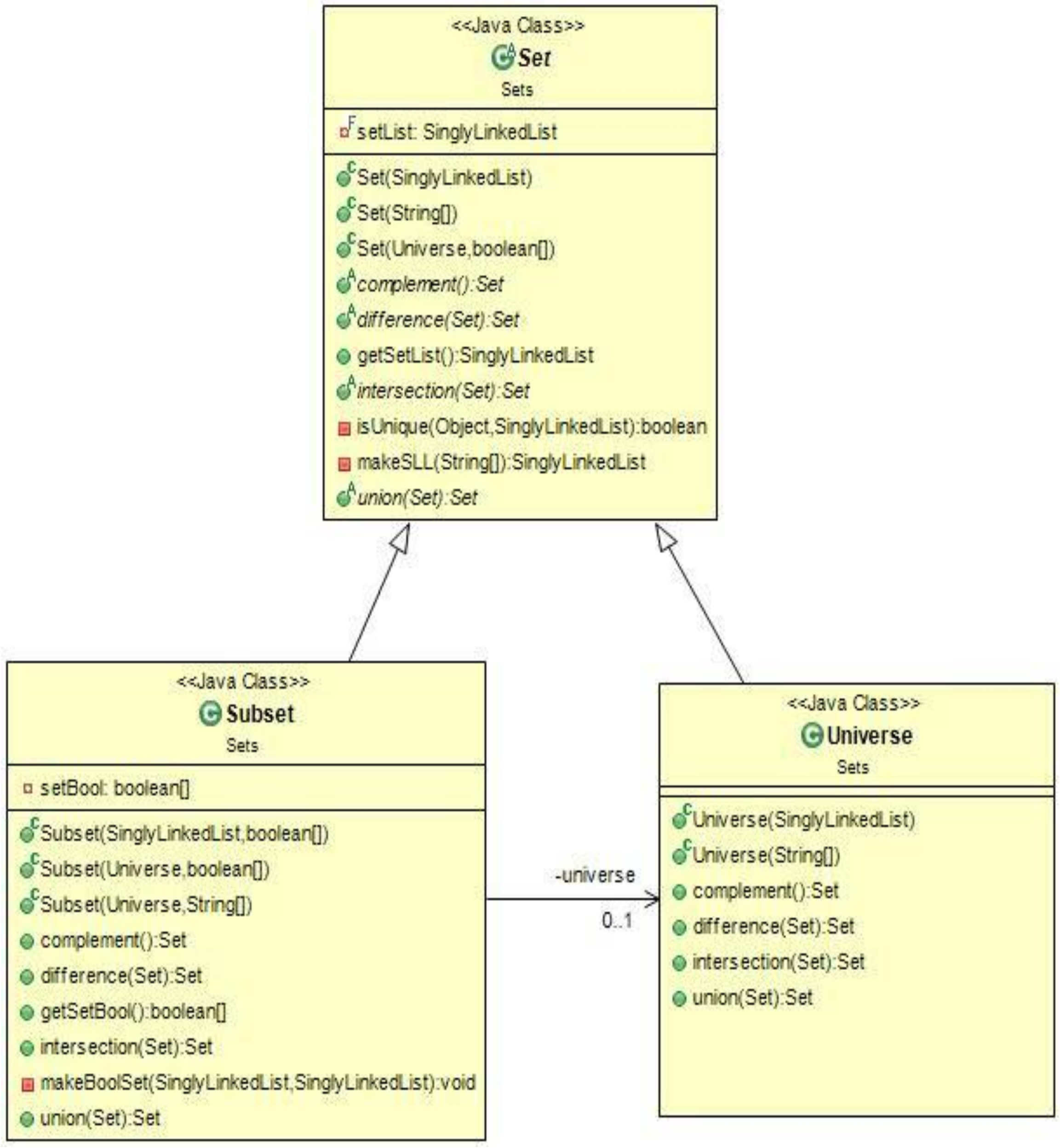
• **void**makeBoolSet(**final**SinglyLinkedList universe, **final**SinglyLinkedList set)

Constructs the setBool to be ready for operations.

param universe : in a SLL form.

param set : in a SLL form.

UML Diagram



<<Java Class>>
Set
Sets
setList: SinglyLinkedList
Set(SinglyLinkedList)
Set(String[])
Set(Universe,boolean[])
complement():Set
difference(Set):Set
getSetList():SinglyLinkedList
intersection(Set):Set
isUnique(Object,SinglyLinkedList):boolean
makeSLL(String[]):SinglyLinkedList
union(Set):Set
<<Java Class>>
Subset
Sets
setBool: boolean[]
Subset(SinglyLinkedList,boolean[])
Subset(Universe,boolean[])
Subset(Universe,String[])
complement():Set
difference(Set):Set
getSetBool():boolean[]
intersection(Set):Set
makeBoolSet(SinglyLinkedList,SinglyLinkedList):void
union(Set):Set
-universe
0..1
<<Java Class>>
Universe
Sets
Universe(SinglyLinkedList)
Universe(String[])
complement():Set
difference(Set):Set
intersection(Set):Set
union(Set):Set

Algorithms (Pseudocode)

- **Operations on Universe:**
  - **Complement:**

```
complement() {
     return null representingempty set (Phi).
}
```

  - **Union:**

```
union(final Set other) {
    return new Universe object with the same
    data.
}
```

  - **Intersection:**

```
intersection(final Set other) {
     check whether the other set is Universe
          if it’s universe
               return new Universe object with
               the same data.
          else if it’s subset
               return new Subset object with
               the same data as “other”.
          End if.
}
```

  - **Difference:**

```
difference(final Set other) {
     check whether the other set is Universe
          if it’s universe
               returnnull representing empty
               set (Phi).
          else if it’s subset
               call the complement function of
               the other set.
          End if.}
```

- **Operations on Subset:**
  - **Complement:**

```
complement() {
     make new array of Booleans.
     Declare Boolean “isUniverse” to make sure
     this is not universe and initialize it to
     true.
     Get the head of the stored universe’s list
     of elements.

     For I = 0 -> this Boolean array of length
          If this element equals false
               Then this isn’t universe
               Set “isUniverse” to false.
               Set the element’s index in the
               new array of Booleans to true.
          End if.
          Get next node in the SLL.
     End for loop.

     Check value of “isUniverse”
          If false
               Return new object of subset with
               the data stored in the array of
               Booleans.
          Else if true
               Return null representing empty
               set (phi).
          End if
}
```

- **Union:**

```
union(final Set other) {
     if this other is universe
          return it.
     Else
          Make new array of Booleans with same
          size as this subset.
          Declare Boolean “isEmpty” and
          initialize it to true.

          For I = 0 -> the Boolean array length
               If OR-ingboth elements of this’
               Boolean array and the other’s
               Boolean array results true
                    Set ”isEmpty”  to false.
                    Set the element’s index to
                    true in the new array of
                    Booleans.
               Else
                    Set the element’s index to
                    false in the new array of
                    Booleans.
               End if
          End for loop.

          Check value of “isEmpty”
               If false
                    Return new subset object
                    with the data stored in the
                    new array of Booleans.
               Else
                    Return null representing
                    empty set (phi).
               End if
     End if
}
```

- **Intersection:**

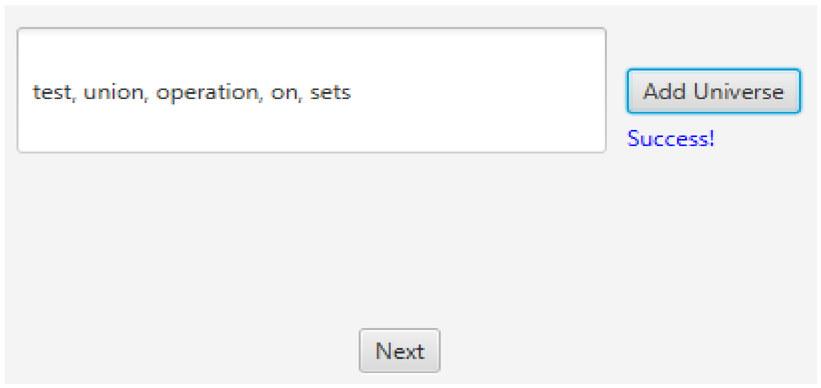
```
intersection(final Set other) {
    if this other is universe
        return this subset.
    Else
        Make new array of Booleans with same
        size as this subset.
        Declare Boolean “intersected” and
        initialize it to false.

        For I = 0 -> the Boolean array length
            If AND-ingboth elements of this’
            Boolean array and the other’s
            Boolean array results true
                Set “intersected” to true.
                Set the element’s index to
                true in the new array of
                Booleans.
            Else
                Set the element’s index to
                false in the new array of
                Booleans.
            End if
        End for loop.

        Check value of “intersected”
            If true
                Return new subset object
                with the data stored in the
                new array of Booleans.
            Else
                Return null representing
                empty set (phi).
            End if
    End if
}
```

- **<u>Difference:</u>**

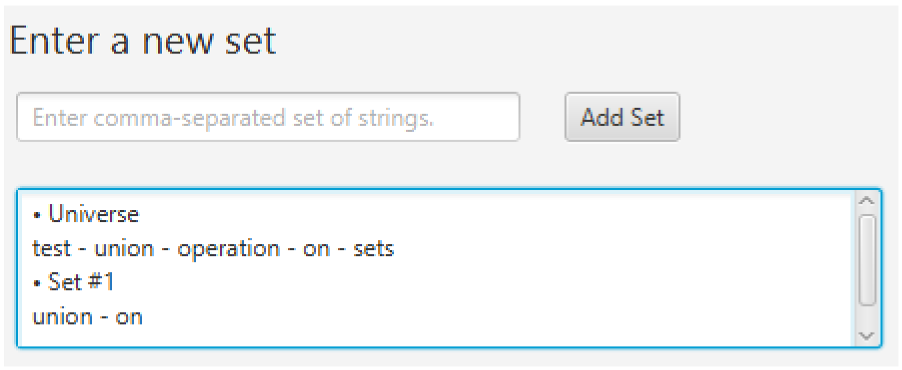
```
difference(final Set other) {
     if this other is universe OR it equals our
     subset
          return null representing empty set
          (phi).
     Else
          Make new array of Booleans with same
          size as this subset.
          Declare Boolean “isEmpty” and
          initialize it to true.

          For I = 0 -> the Boolean array length
               If this set’s element exist and
               the other’s doesn’t
                    Set “isEmpty” to false.
                    Set the element’s index to
                    true in the new array of
                    Booleans.
               Else
                    Set the element’s index to
                    false in the new array of
                    Booleans.
               End if
          End for loop.

          Check value of “intersected”
               If false
                    Return new subset object
                    with the data stored in the
                    new array of Booleans.
               Else
                    Return null representing
                    empty set (phi).
               End if
     End if
}
```

Sample Runs

Test Union operation:

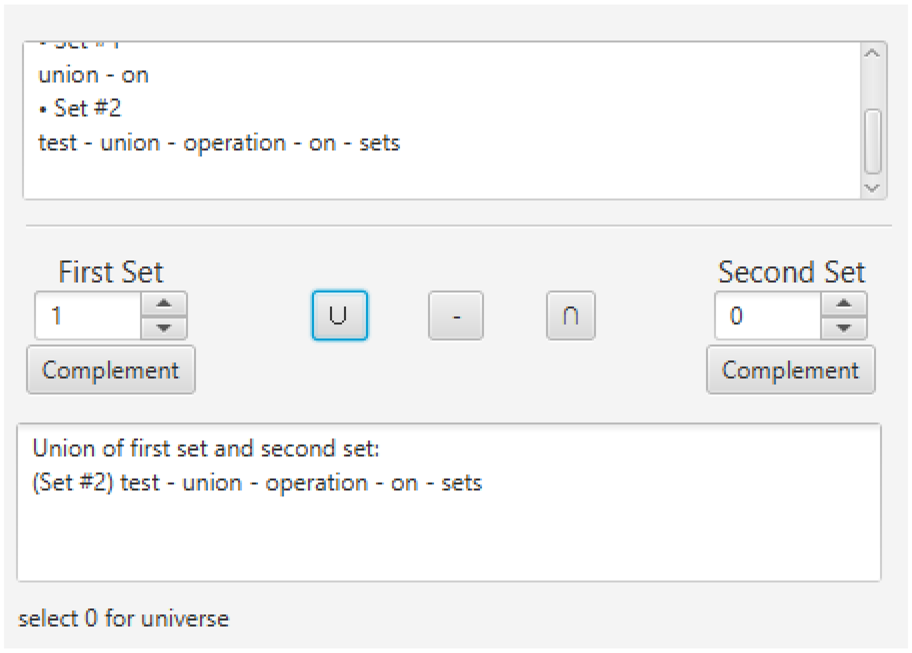

test, union, operation, on, sets

- **User inserts new universe and Next button is enabled.**



• Universe
test - union - operation - on - sets
• Set #1
union - on

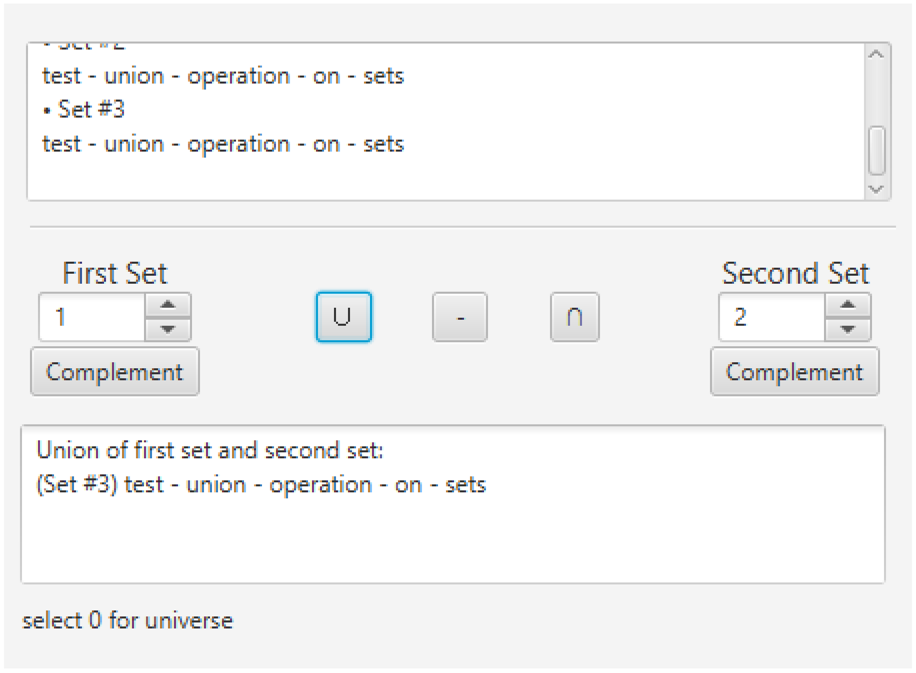
- **User inserts first set.**



union - on
• Set #2
test - union - operation - on - sets

Union of first set and second set:
(Set #2) test - union - operation - on - sets

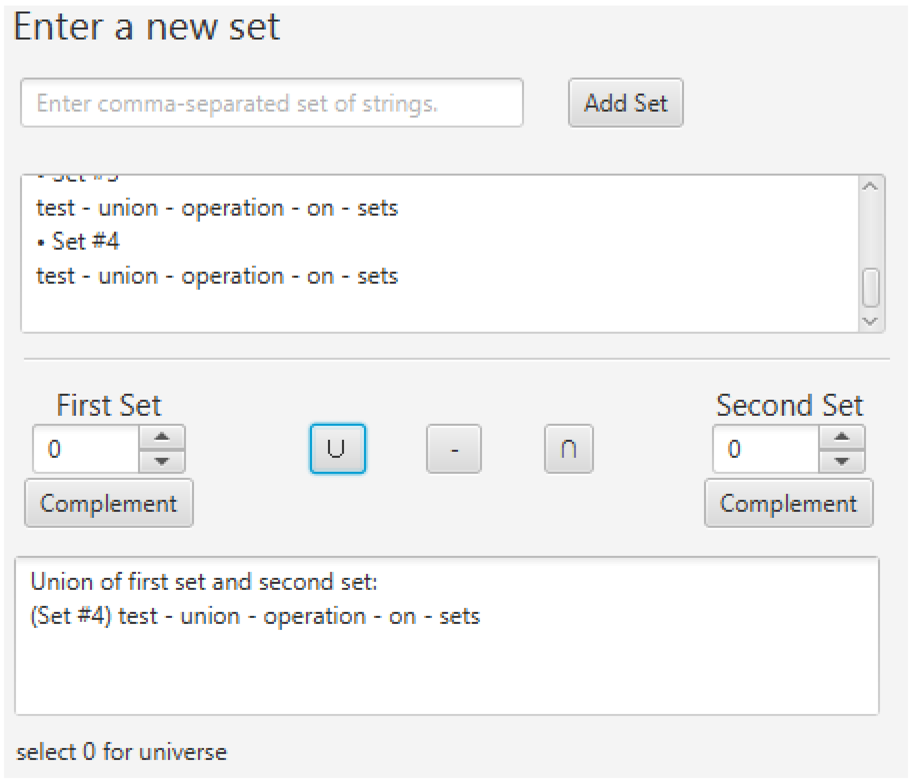
- **Union of subset and universe, the result set is added to the list.**



test - union - operation - on - sets
• Set #3
test - union - operation - on - sets

Union of first set and second set:
(Set #3) test - union - operation - on - sets

- **Union of two subsets – Union operation performed on the result set.**

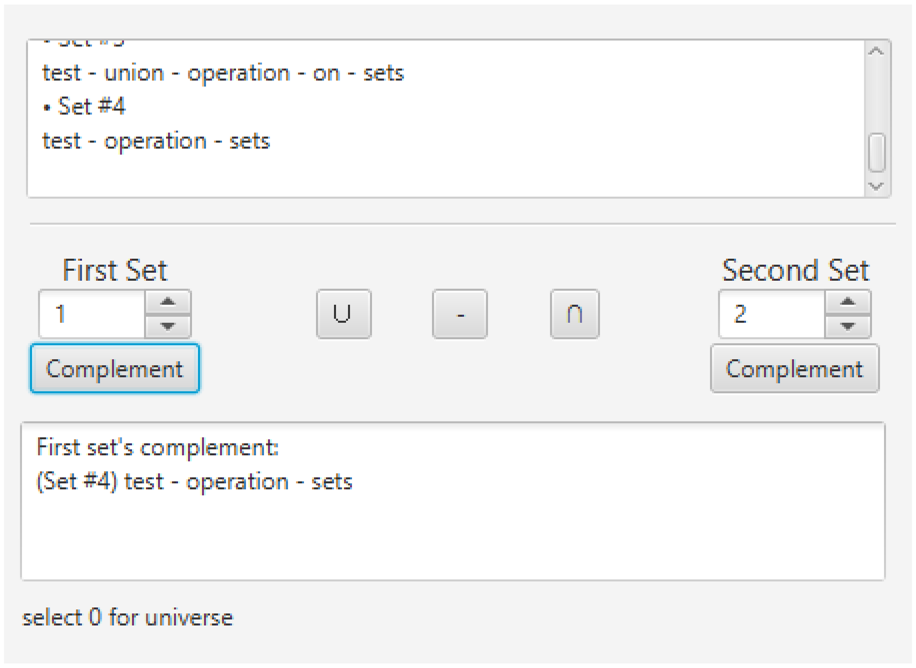


test - union - operation - on - sets
• Set #4
test - union - operation - on - sets

Union of first set and second set:
(Set #4) test - union - operation - on - sets

- **Union of two identical universes.**

Test Complement operation:

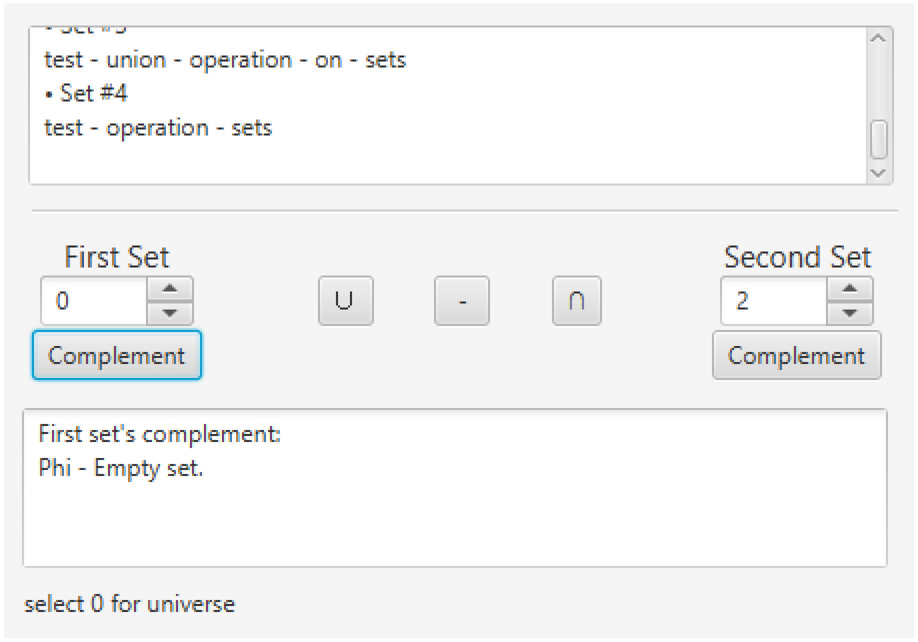


test - union - operation - on - sets
• Set #4
test - operation - sets

First set's complement:
(Set #4) test - operation - sets

select 0 for universe

- **Complement of subset.**

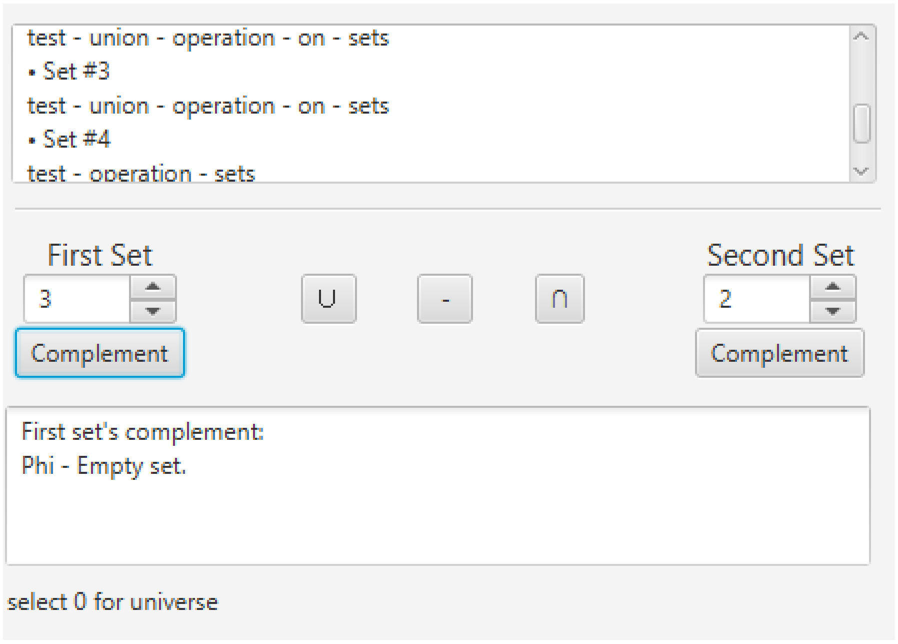


test - union - operation - on - sets
• Set #4
test - operation - sets

First set's complement:
Phi - Empty set.

select 0 for universe

- **Complement of Universe – Empty set isn't added to the list.**

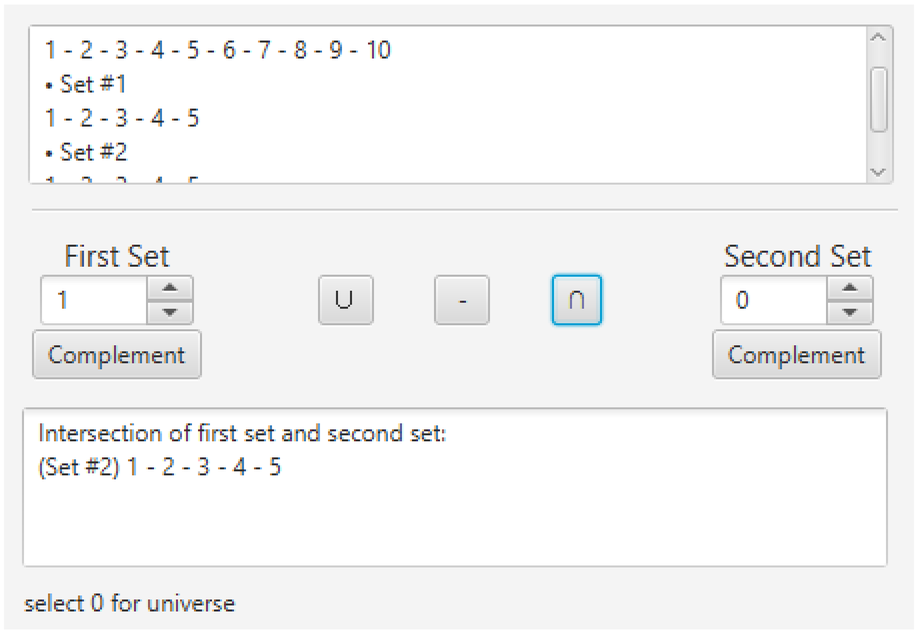

test - union - operation - on - sets
• Set #3
test - union - operation - on - sets
• Set #4
test - operation - sets

First set's complement:
Phi - Empty set.

select 0 for universe

- **Complement of subset includes all elements in the universe.**

Test Intersection operation:

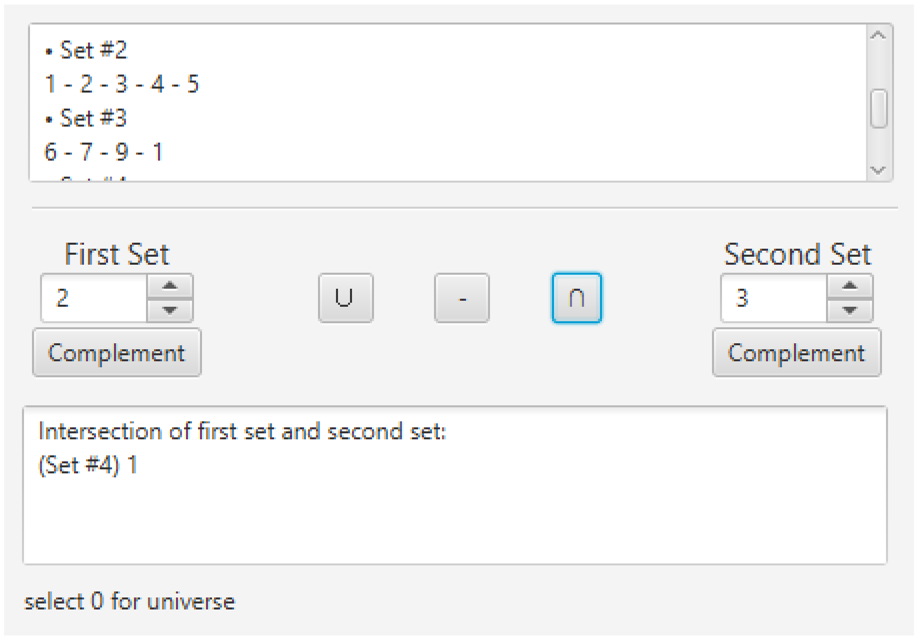

1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10
• Set #1
1 - 2 - 3 - 4 - 5
• Set #2

First Set: 1
Second Set: 0

Intersection of first set and second set:
(Set #2) 1 - 2 - 3 - 4 - 5

select 0 for universe

- **Intersection of subset and universe.**

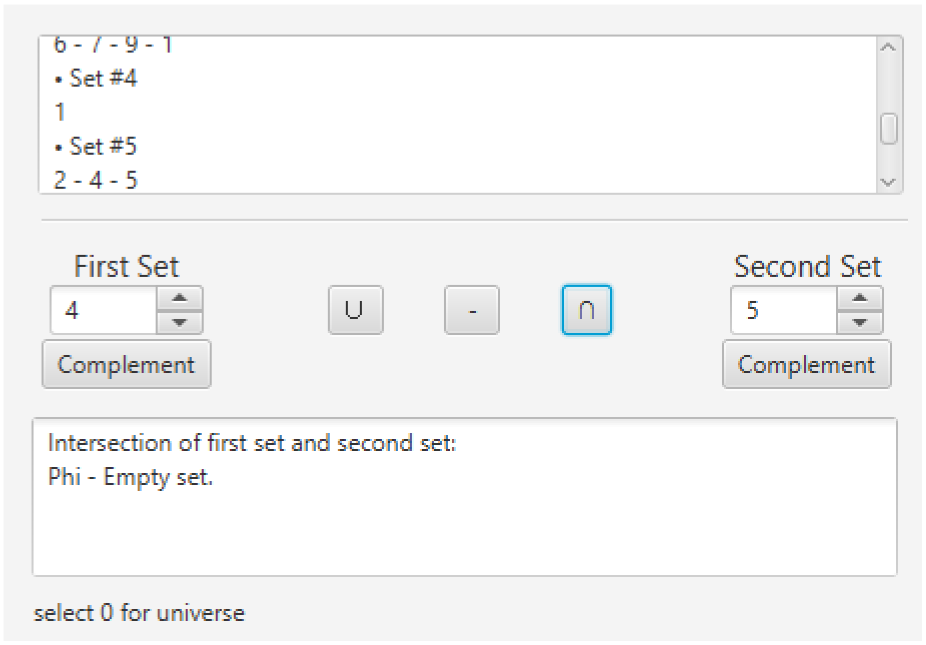

• Set #2
1 - 2 - 3 - 4 - 5
• Set #3
6 - 7 - 9 - 1

First Set: 2
Second Set: 3

Intersection of first set and second set:
(Set #4) 1

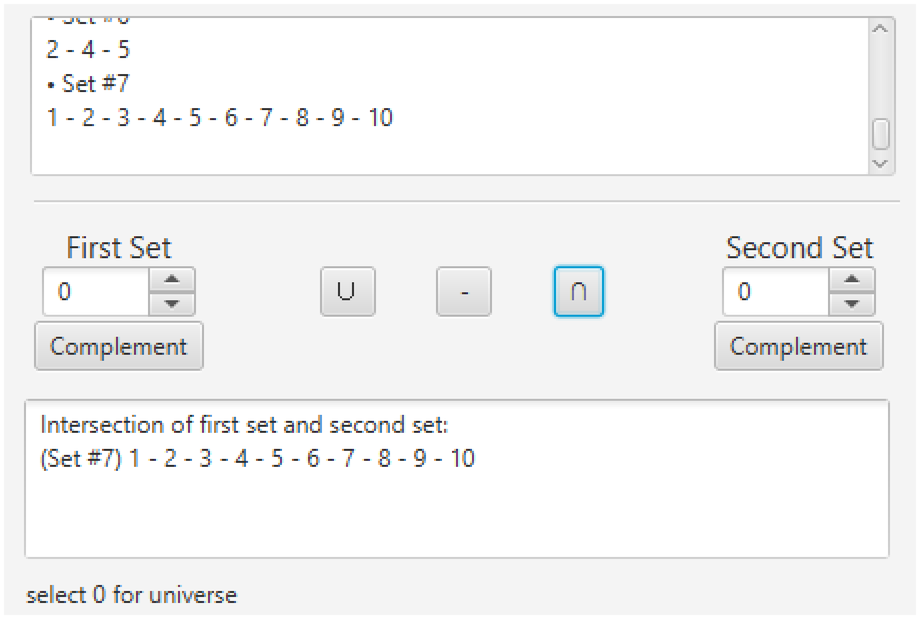
select 0 for universe

- **Intersection of two subsets.**



```
Intersection of first set and second set:
Phi - Empty set.
```

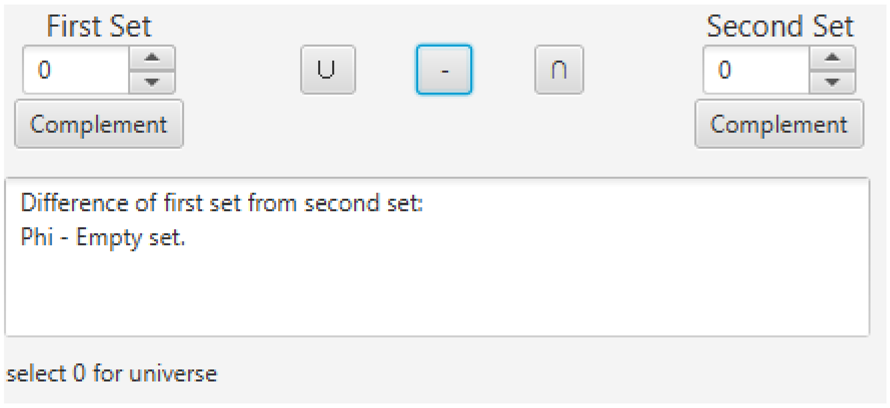
- **Intersection of non-intersected subsets.**



```
Intersection of first set and second set:
(Set #7) 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10
```

- **Intersection of identical two universes.**

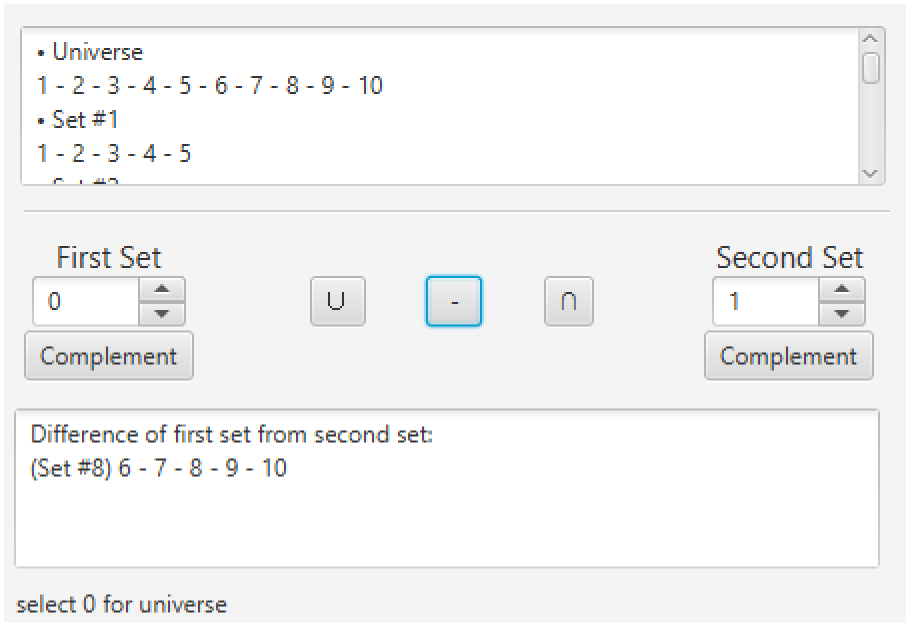
Test Difference operation:



Difference of first set from second set:
Phi - Empty set.

select 0 for universe

- **Difference between two identical universes.**

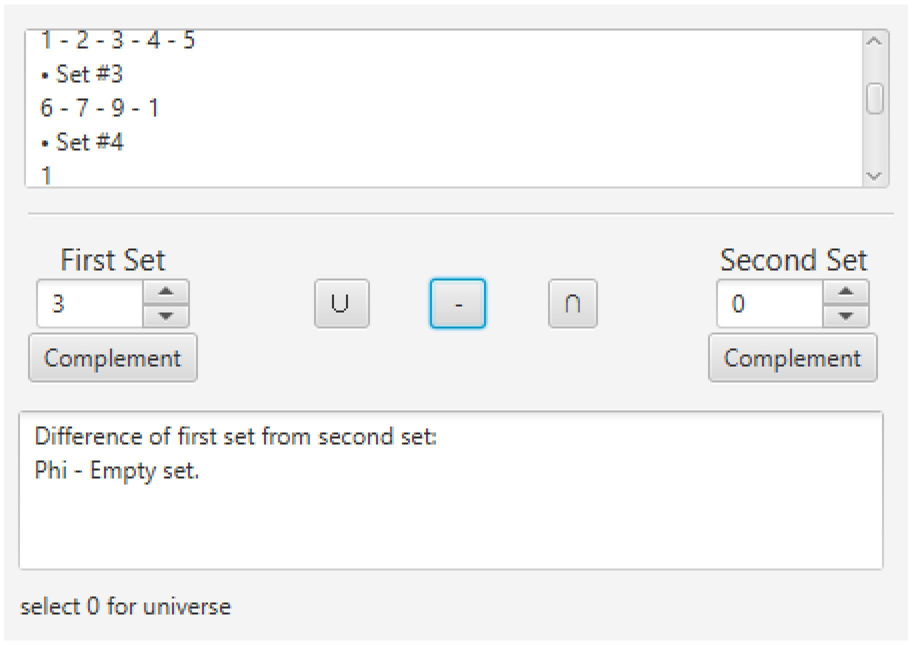


• Universe
1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10
• Set #1
1 - 2 - 3 - 4 - 5

Difference of first set from second set:
(Set #8) 6 - 7 - 8 - 9 - 10

select 0 for universe

- **Difference of universe from subset.**

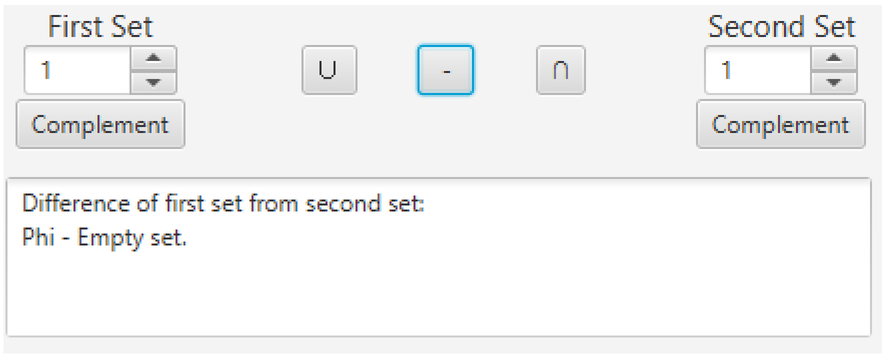


```
1 - 2 - 3 - 4 - 5
• Set #3
6 - 7 - 9 - 1
• Set #4
1
```

```
Difference of first set from second set:
Phi - Empty set.
```

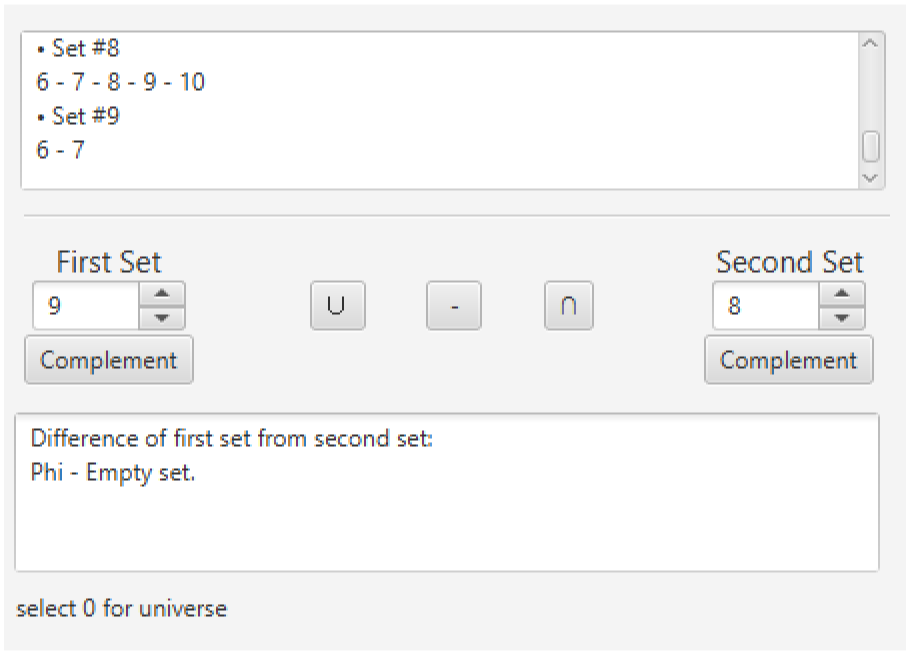
select 0 for universe

- **Difference of subset from universe.**



```
Difference of first set from second set:
Phi - Empty set.
```

- **Difference between two identical subsets.**

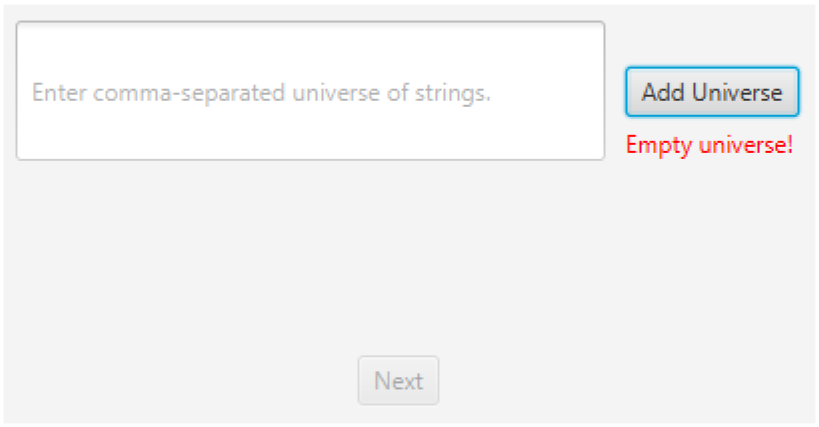

• Set #8
6 - 7 - 8 - 9 - 10
• Set #9
6 - 7

Difference of first set from second set:
Phi - Empty set.

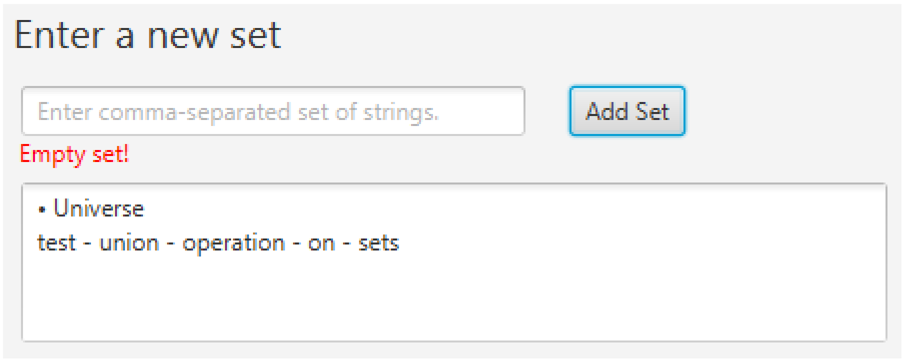
select 0 for universe

- **Difference of subset from another subset which is subset of it too.**

Special Cases:



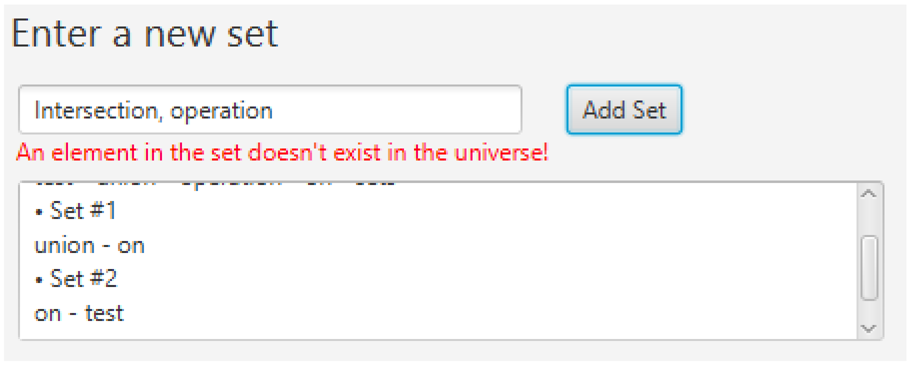
- ***Empty universe.***


Enter a new set

Empty set!

• Universe
test - union - operation - on - sets

- **Empty set.**


Enter a new set

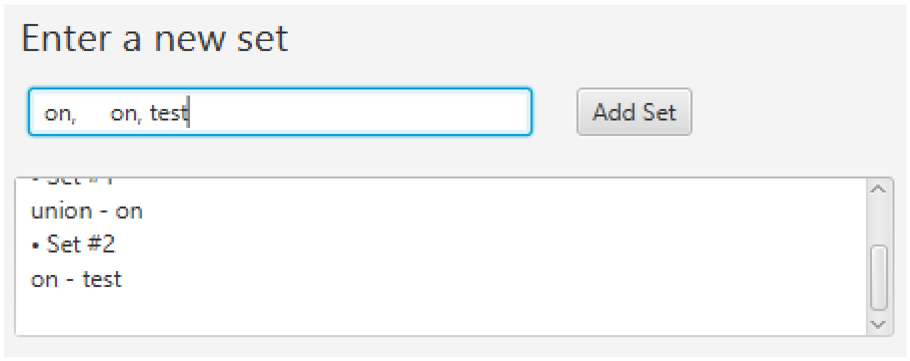
Intersection, operation

Add Set

An element in the set doesn't exist in the universe!

```
• Set #1
union - on
• Set #2
on - test
```

- **Subset outside universe.**

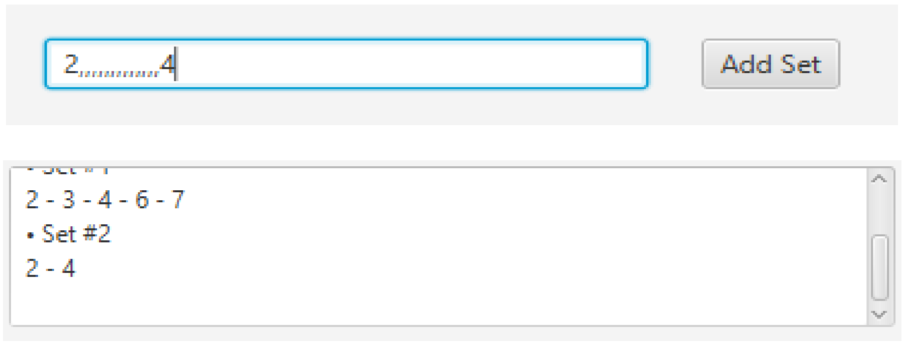

Enter a new set

on,   on, test

Add Set

```
union - on
• Set #2
on - test
```

- **Remove duplicates and whitespaces.**


2,,,,,,,,,,,4

Add Set

```
2 - 3 - 4 - 6 - 7
• Set #2
2 - 4
```

- **Remove empty elements from sets.**

<u>**Assumptions**</u>

- The user inserts any set in **<u>one</u>** line.
- Elements of the set are comma-separated strings.
- The user is allowed to insert only **<u>one</u>** universe.

Thank you.