**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 Professors: Amal ElNahas & Dr. Magdy Abdelazim**

**and Teaching Assistants:**

**• Eng/ Reham Osama.**

**Index**

1. Introduction 4
2. Overview 5
3. Features 5
4. Data Structures 6
5. Functions 9
6. Test Cases 18
7. User Manual 22

**Introduction**

This project was assigned in the CS211: Mathematics for Computers course in the date: Friday, September 22nd, 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 Saweras Toussy*

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

[github.com/hesham-medhat/Setty\_Operations/](https://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.

• graphical user interface (GUI).

**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. \*/

public boolean isEmpty();

/\*\* Removes the element at the specified position in this list. \*/

public void remove (int index);

/\*\* Returns the number of elements in this list. \*/

public int size();

/\*\*

\* Returns a view of the portion of this list between the

\* specified

\* fromIndex and toIndex, inclusively.

\*/

public ILinkedList sublist (int fromIndex, int toIndex);

/\*\*

\* Returns true if this list contains an element with the same

\* value as thespecified element.

\*/

public boolean contains (Object o);

}

**Packages**

**A. Package: application:**

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

○ Controller class: Main.java.

○ Two fxml 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.

**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, **final** **boolean**[] setBoolIn)

Constructor in case we know the boolean set.

**param** universeIn : universe of the set.

**param** setBoolIn : boolean array of existence of elements from universe.

• **boolean** isUnique(**final** Object element, **final** SinglyLinkedList setSLL)

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.

**returns** complement 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.

• SinglyLinkedList getSetList()

Getter for setList.

**returns** setList 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.

**param** setInput : in the form of a string array.

**Subset:**

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

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

Constructor in case we already have the list built.

**param** list : of elements in the subset.

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

Constructor in case the boolean array is ready.

**param** universeIn : universe

**param** setBoolIn : the readySetBool

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

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

**param** universeIn : universe as object.

**param** setInput : set content input as string array.

• **boolean**[] getSetBool()

Getter for setBool.

**returns** setBool 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**

****

**Algorithms (Pseudocode)**

**Sample Runs**

**Test (1)**

*Thank you.*