

## Assignment #4

### Description

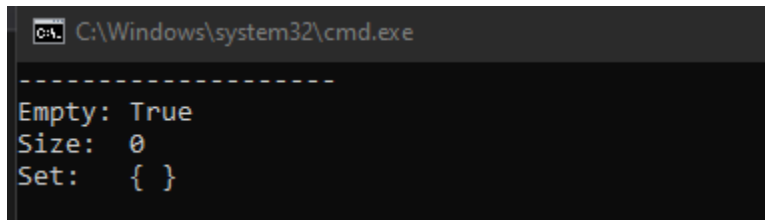
For this assignment we are asked to create generic class `Set<T>` and subclass `SortedSet<T>`. In `Set<T>`, which requires for its members of type `T` to be `Enumerable`, we should have default and explicit value constructors (this one should receive `IEnumerable` item in order to iterate through it). Also we should have the following properties: `Count` – to show number of items in set, `IsEmpty` – to show if set is empty or not, and the following methods: `Add()` – to add item to set, `Remove()` – remove item from the set, `Contains()` – check if item is in set, `Filter()` – to filter sets by using delegate and calling functions for set, also overloaded operator `+` to create union of two sets. For `SortedSet<T>`, I sort set using ordinary `Sort()` for lists as I have sets represented by `List` to easily manipulate on them. `SortedSet<T>` requires for its members of type `T` to be comparable. I overrode `Add()` and `Remove()` functions for this class. Other functions and properties are inherited from `Set` class.

In `Set` class, `Set` constructor simply iterates through the `Enumerable` set passed in it and adds each item to the list. In `Add()` function I first check if set contains the item to be added, if contains, do not add and return false, otherwise add and return true. I check item in `Remove()` function similarly and decide remove it or not. In `Filter()` function I created temporary set, also I have object of `F<T>` which calls a `filterFunction`. I call this function for items in my real set and store filtered items (for which the function returns true) into temporary set. Finally, the function returns the temporary set. For the union of sets, all elements from left hand side set are stored in temporary set, then I take elements from right hand side set and if item is not in union set, I add it, so union of sets contains single copy of each item.

In `main()` function, I have sets of integers, strings and also little `Person` class to test my classes. Also have two filter functions – `filterByEven` for creating set of even integers, and `filterByLetterA` for creating set of strings starting with letter A.

## Screenshots

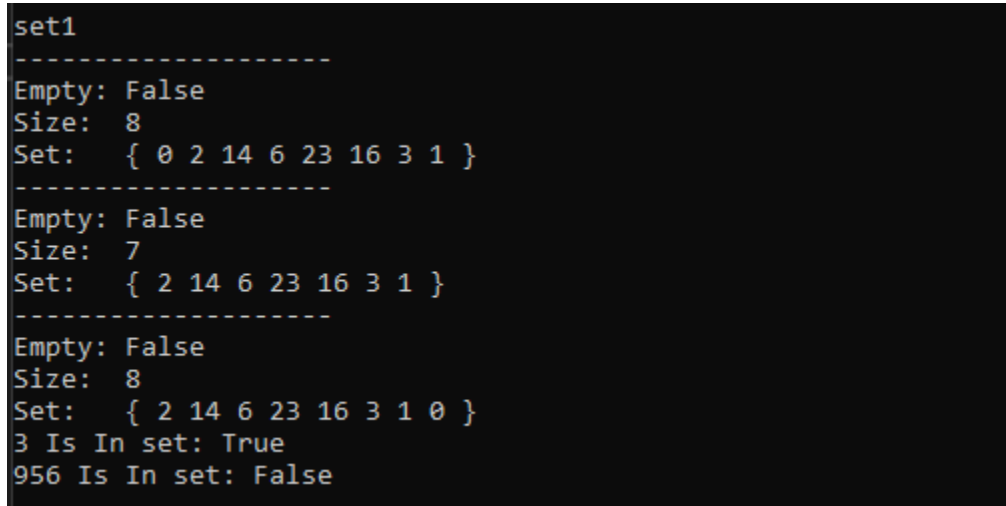
1) Empty set



```
C:\Windows\system32\cmd.exe

-----
Empty: True
Size: 0
Set:  { }
```

2) Set of integers. In second case, 0 is removed, in third case 0 is added.



```
set1
-----
Empty: False
Size: 8
Set:  { 0 2 14 6 23 16 3 1 }
-----
Empty: False
Size: 7
Set:  { 2 14 6 23 16 3 1 }
-----
Empty: False
Size: 8
Set:  { 2 14 6 23 16 3 1 0 }
3 Is In set: True
956 Is In set: False
```

3) Union of set1 and set 2

```
C:\Windows\system32\cmd.exe

set1
-----
Empty: False
Size: 8
Set: { 0 2 14 6 23 16 3 1 }
-----
Empty: False
Size: 7
Set: { 2 14 6 23 16 3 1 }
-----
Empty: False
Size: 8
Set: { 2 14 6 23 16 3 1 0 }
3 Is In set: True
956 Is In set: False

set2
-----
Empty: False
Size: 6
Set: { 4 3 10 6 5 2 }

Union of set1, set2
-----
Empty: False
Size: 11
Set: { 2 4 10 5 14 6 23 16 3 1 0 }
```

4) Set of strings

```
set3
-----
Empty: False
Size: 5
Set: { D B A M G }
J Is In set: False
D Is In set: True
```

- 5) Sorted set1. In second case removed 7. In third case added 25 and 19.  
Sorted set3.  
Union of these sets.

```
C:\Windows\system32\cmd.exe

sorted1
-----
Empty: False
Size: 6
Set: { 4 5 6 7 8 9 }
-----
Empty: False
Size: 5
Set: { 4 5 6 8 9 }
-----
Empty: False
Size: 7
Set: { 4 5 6 8 9 19 25 }

sorted3
-----
Empty: False
Size: 7
Set: { 0 4 5 7 18 26 56 }

Union of sorted1 and sorted3
-----
Empty: False
Size: 12
Set: { 0 4 5 6 7 8 9 18 19 25 26 56 }
```

- 6) Sorted set of strings. In second case added "Dachi".

```
sorted2
-----
Empty: False
Size: 5
Set: { Anna Giorgi Khatia Luka Nika }
-----
Empty: False
Size: 6
Set: { Anna Dachi Giorgi Khatia Luka Nika }
```

- 7) 1 – set of Person objects  
2 – sorted set of Person objects

```
setOfP
-----
Empty: False
Size: 3
Set: { Nini Anna Mariam }

sortedSetOfP
-----
Empty: False
Size: 3
Set: { Anna Mariam Nini }
```

- 8) 1 – filtered set of even numbers  
2 – filtered set of string names starting with 'A'

```
Filtered1 - even numbers
-----
Empty: False
Size: 4
Set: { 4 10 6 2 }

Filtered2 - starts with 'A'
-----
Empty: False
Size: 1
Set: { Anna }
```

## Code

### Set.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

/// <summary>
/// Ana Maghradze
/// 823356346
/// </summary>
namespace COMPE361
{
    // delegate for Filter()
    public delegate bool F<T>(T e);

    class Set<T> : IEnumerable<T>
    {
        // using list for to create sets
        public List<T> mySet = new List<T>();

        //This property returns the number of elements in the set.
        //This property should be readable but not writable.
        private int Count => mySet.Count;
        //This property returns true if there are no items in the set.
        //This property should be readable but not writable.
        private bool IsEmpty => mySet.Count == 0;

        // default constructor
        public Set(){ }

        //constructor that fills the set with all the elements in some enumerable
        collection.
        public Set(IEnumerable<T> e)
        {
            foreach(T item in e)
            {
                mySet.Add(item);
            }
        }
        // This method returns true if the input element is in the set.
        public bool Contains(T item)
        {
            return mySet.Contains(item);
        }
        //This method adds the input element to the set.
        //It returns true if the element is added to the set,
        //and false if the element is already present in the set.
        public virtual bool Add(T item)
        {

```

```
        if (mySet.Contains(item))
        {
            return false; // if item is already in set
        }
        mySet.Add(item); // if not in set, add item to list
        return true;
    }
    // This method removes the input element from the set.
    // It returns true if the element is removed to the set,
    // and false if the element was not in the set to begin with.
    public virtual bool Remove(T item)
    {
        if (!mySet.Contains(item))
        {
            return false; // if item is not in set
        }
        mySet.Remove(item);
        return true;
    }
    //This method takes a delegate of type bool F<T>(T elt) and returns
    //all the elements in the set for which this function returns true.
    public Set<T> Filter(F<T> filterFunction)
    {
        F<T> d1 = new F<T>(filterFunction); // instance of delegate
        Set<T> tempSet = new Set<T>(); // temporary set for filtered items
        foreach (T item in mySet)
        {
            if (d1(item))
            {
                tempSet.Add(item); // add elements to tempset
            }
        }
        return tempSet; // return filtered set
    }

    // This operator implements set union: it should return a new
    // set that contains any item contained in either the lhs or the rhs set.
    public static Set<T> operator +(Set<T> lhs, Set<T> rhs)
    {
        Set<T> union = new Set<T>();
        foreach(T i in lhs)
        {
            union.Add(i); // add lhs items
            foreach(T j in rhs)
            {
                if (!lhs.Contains(j))
                {
                    union.Add(j); // if not already in set, add rhs items
                }
            }
        }
        return union;
    }

    // IEnumerable implementation
    public IEnumerator<T> GetEnumerator()
    {
        return mySet.GetEnumerator();
    }
}
```

COMPE 361  
Ana Maghradze

```
    }

    IEnumerator<T> IEnumerable<T>.GetEnumerator()
    {
        return GetEnumerator();
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        throw new NotImplementedException();
    }

    // display results for objects
    public void DisplaySet()
    {
        Console.WriteLine("-----");
        Console.WriteLine("Empty: " + IsEmpty);
        Console.WriteLine("Size: " + Count);
        Console.Write("Set:  { ");
        foreach (var i in this)
        {
            Console.Write(i + " ");
        }
        Console.WriteLine("\n");
    }
}
```

## SortedSet.cs

```
using System;
using System.Collections.Generic;

/// <summary>
/// Ana Maghradze
/// 823356346
/// </summary>
namespace COMPE361
{
    class SortedSet<T> : Set<T> where T : IComparable<T>
    {
        // default constructor
        public SortedSet() { }

        // explicit value constructor
        public SortedSet(IEnumerable<T> e) : base(e) { }

        // override Add for SortedSet
        public override bool Add(T item)
        {
            if (mySet.Contains(item))
```



```
        {
            return false; // if item is already in set
        }
        mySet.Add(item);
        mySet.Sort(); // sort set after item is added
        return true;
    }
    // override Add For SortedSet
    public override bool Remove(T item)
    {
        if (!mySet.Contains(item))
        {
            return false;
        }
        mySet.Remove(item);
        return true;
    }

    // + operator for unions
    public static SortedSet<T> operator +(SortedSet<T> lhs, SortedSet<T> rhs)
    {
        SortedSet<T> union = new SortedSet<T>();
        foreach (T i in lhs)
        {
            union.Add(i);
            foreach (T j in rhs)
            {
                if (!lhs.Contains(j))
                {
                    union.Add(j);
                }
            }
        }
        return union;
    }

    public int CompareTo(Object obj)
    {
        SortedSet<T> other = obj as SortedSet<T>;
        return this.CompareTo(other);
        throw new NotImplementedException();
    }
}
}
```

## Driver

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

/// <summary>
/// Ana Maghradze
/// 823356346
/// </summary>
namespace COMPE361
{
    class Program
    {
        // return even numbers
        public static bool FilterByEven(int a)
        {
            return a % 2 == 0;
        }
        // return objects starting with A
        public static bool FilterByLetterA(string a)
        {
            return a.StartsWith("A");
        }

        static void Main(string[] args)
        {
            // objects of Set class with type int
            Set<int> set1 = new Set<int>() { 0, 2, 14, 6, 23, 16, 3, 1 };
            Set<int> set2 = new Set<int>() { 4, 3, 3, 10, 3, 6, 5, 2 };
            // obj of Set class with type string
            Set<string> set3 = new Set<string>() { "D", "B", "A", "M", "G" };
            // obj of SortedSet class with type int
            SortedSet<int> sorted1 = new SortedSet<int>() { 5, 7, 4, 8, 6, 9 };
            // obj of SortedSet class with type string
            SortedSet<string> sorted2 = new SortedSet<string>() { "Luka", "Khatia",
"Nika", "Anna", "Giorgi" };

            // create objects of class Person
            Person p1 = new Person("Nini");
            Person p2 = new Person("Anna");
            Person p3 = new Person("Mariam");
            // object of Set class with type Person
            Set<Person> setOfP = new Set<Person>() { p1, p2, p3 };
            // object of SortedSet class with type Person
            SortedSet<Person> sortedSetOfP = new SortedSet<Person>() { p1, p2, p3 };

            // set0 - empty set
            Set<int> set0 = new Set<int>() { };
            set0.DisplaySet();
            // set1
            Console.WriteLine("\nset1");
        }
    }
}
```

```
set1.DisplaySet(); // display set
set1.Remove(12); // 12 is not in set so nothing will be removed
set1.Remove(0); // 0 is removed from the set
set1.DisplaySet(); // display modified set
set1.Add(0); // add 0 to the set
set1.DisplaySet(); // display modified set
int a = 3; // check if item is in set
Console.WriteLine($"{a} Is In set: {set1.Contains(a)}");
int b = 956; // check if item is in set
Console.WriteLine($"{b} Is In set: {set1.Contains(b)}");
// set2
Console.WriteLine("\n\nset2");
set2.DisplaySet();
// union of non-sorted sets
Console.WriteLine("\n\nUnion of set1, set2\n");
Set<int> union = new Set<int>();
union = set1 + set2; // union of set1 and set2
union.DisplaySet();
// set3
Console.WriteLine("\n\nset3");
set3.DisplaySet();
string item = "J"; // check if item is in set
string item1 = "D"; // check if item is in set
Console.WriteLine($"{item} Is In set: {set3.Contains(item)}");
Console.WriteLine($"{item1} Is In set: {set3.Contains(item1)}");
// sorted1
Console.WriteLine("\n\nsorted1");
sorted1.DisplaySet();
sorted1.Remove(7); // remove 7 from sorted set
sorted1.DisplaySet(); // display modified set
sorted1.Add(25); // Add 25 to sorted set
sorted1.Add(19); // Add 25 to sorted set
sorted1.DisplaySet(); // display modified set

// sorted3
SortedSet<int> sorted3 = new SortedSet<int>() { 0, 18, 5, 56, 4, 26, 7 };
Console.WriteLine("\n\nsorted3");
sorted3.DisplaySet();
// union of sorted sets
Console.WriteLine("\n\nUnion of sorted1 and sorted3");
SortedSet<int> sortedUnion = new SortedSet<int>();
sortedUnion = sorted1 + sorted3; // union of sorted1 and sorted3
sortedUnion.DisplaySet();

// sorted2 - type of string
Console.WriteLine("\n\nsorted2");
sorted2.DisplaySet();
sorted2.Add("Dachi");
sorted2.Remove("Mariam");
sorted2.DisplaySet();

// setOfP
Console.WriteLine("\n\nsetOfP");
setOfP.DisplaySet();

// sortedSetOfP
Console.WriteLine("\n\nsortedSetOfP");
sortedSetOfP.DisplaySet();
```

```
        // Call Filter for FilterByEven
        Console.WriteLine("\n\nFiltered1 - even numbers");
        var filtered1 = set2.Filter(FilterByEven);
        filtered1.DisplaySet();
        // Call Filter for FilterByLetterA
        Console.WriteLine("\n\nFiltered2 - starts with 'A'");
        var filtered2 = sorted2.Filter(FilterByLetterA);
        filtered2.DisplaySet();
        Console.WriteLine();
    }
}

// test class
class Person : IComparable<Person>
{
    public string Name { get; set; }

    public Person(string name) { Name = name; }

    public override string ToString()
    {
        return Name.ToString();
    }

    public int CompareTo(Person y)
    {
        return this != y ? Name.ToString().CompareTo(y.Name) : 0;
    }
}
}
```