

## 1. Homework 6

**Posted:** October/1/2018

**Due:** October/7/2018 24.00

All homework solutions are due October/7/2018 24.00. I recommend to submit at least one version of all homework solutions long before due date.

### 1.1. Homework 6.1 (10 Points)

**Objective:** Modifying an existing algorithm and learning how to use generics

**Grading:**

Correctness: You can lose up to 40% if your solution is not correct

Quality: You can lose up to 80% if your solution is poorly designed

Testing: You can lose up to 50% if your solution is not well tested

Explanation: You can lose up to 100% if your solution if you can not explain your solution during the grading session

**Homework Description:**

You have to implement a storage system that is behaves like a set.

**Explanation:**

A set is collection of objects in which order has no significance. Members of a set are called elements.

The following operations are

*Union  $\cup$  and intersection  $\cap$ .*

defined on sets. It is also possible to ask if  $x$  is a member of a set  $A$ .  $x \in A$  is true, if  $x$  is a member of  $A$  otherwise false.

You have to implement the following methods:

```
boolean      add(E e)      /* add e to the set                      */
boolean      addAll(       /* add all elements of the set                      */
                        /* you have to come of with the correct signature */
                        /* for all elements of type E or subclass shape */
                        /* if possible                               */
                        /* true if one element could have been added */
boolean      removeAll(    /* remove all elements of the set                      */
                        /* you have to come of with the correct signature */
                        /* for all elements of type E or subclass shape */
                        /* if possible                               */
                        /* true if all elements could have been removed */
Object[]      toArray()    /* return all elements of the set in an array          */
boolean      contains(Object o) /* true if o is in the set, false else                */
boolean      remove(Object o) /* remove o, true if o could be removed                */
void         clear()       /* empty the set                                         */
int          size()        /* # of elements in the set                             */
```

**Your Work:**

A constructor is not defined. It is up to you to define at least one constructor.

It might be useful to think about the signatures of *addAll*, and *removeAll* before you implement the class.

It might be useful to think about in which order you should develop the methods and how you will test the methods.

You can not use any existing Java class for this home work.

**Requirements:**

You have to provide a test environment for your work. You have to name your storage class *Storage.java*.

**Example:**

An example of a solution execution:

This is a compilation and execution of the following code:

```
% ls Storage.java TestStorage.java
Storage.java      TestStorage.java
% java TestStorage
aStorage: # of elements: 1  ->a->>null
You can not add yourself to yourself.
aStorage: # of elements: 1  ->a->>null

% cat TestStorage.java | extractPublic
    public static void exampleOfHowToUseIt( Storage<String> aStorage)    {
        aStorage = new Storage<String>();
        Storage<String> bStorage = new Storage<String>();
        aStorage.add("a");
        System.out.println("aStorage: " + aStorage );
        bStorage.add("b");
        if ( ! aStorage.addAll(aStorage) )
            System.out.println("You can not add yourself to yourself.");
        aStorage.addAll(bStorage);
        aStorage.removeAll(bStorage);
        System.out.println("aStorage: " + aStorage );
    }
    public static void test(Storage<String> aStorage)    {
        if ( ! testRemove() )
            System.err.println("testRemove failed");
        if ( ! testRemoveAll() )
            System.err.println("testRemove failed");
        if ( ! testContains() )
            System.err.println("testContains failed");
        if ( ! testAdd() )
            System.err.println("testAdd failed");
        if ( ! testAddAll() )
            System.err.println("testAddAll failed");
        if ( ! testClear() )
            System.err.println("testClear failed");
    }
    public static void main(String args[] )    {
        test(new Storage<String>());
        exampleOfHowToUseIt(new Storage<String>());
    }
}
```

**Submission:**

```
% ssh glados.cs.rit.edu # or use queeg.cs.rit.edu if glados is down
# password
# go to the directory where your solution is ...
% try hpb-grd lab6-1 'All files required'
# you can see if your submission was successful:
# try -q hpb-grd lab6-1
```

### Solution:

(This solution serves as the basis for the discussion in class. Sometimes there will be errors introduced to show common mistakes)

```
1      /*
2      * you can not add or test for null elements in this implementation
3      */
4      public class Storage<E> implements StorageI<E> {
5
6          private Storage<E> root;
7          private Storage<E> next;
8          private int      nElements = 0;
9
10         E payLoad;
11
12         public Storage(){
13         }
14         public String getClassName()      {
15             return "Storage";
16         }
17         public Object[] toArray() {
18             Object[] anArray = new Object[size()];
19             Storage<E> runner = root;
20             int index = 0;
21             while ( runner.payLoad != null )      {
22                 anArray[index++] = runner.payLoad;
23                 runner = runner.next;
24             }
25             return anArray;
26         }
27         public boolean addAll(Storage<? extends E> theSetToAdd) {
28
29             boolean rValue = true;
30             if ( this == theSetToAdd ) {
31                 return false;
32             }
33             Storage<? extends E> runner = theSetToAdd.root;
34             while ( runner.payLoad != null )      {
35                 rValue &= add(runner.payLoad );
36                 runner = runner.next;
37             }
38             return rValue;
39         }
40         public boolean removeAll(Storage<? extends E> theSetToRemove) {
41             boolean rValue = true;
42             if ( this == theSetToRemove )      {
```

```
43         return false;
44     }
45     Storage<? extends E> runner = theSetToRemove.root;
46     while ( runner.payload != null )        {
47         if ( contains(runner.payload) ) {
48             rValue |= null != remove(runner.payload);
49             runner = runner.next;
50         }
51     }
52     return rValue;
53 }
54 public boolean contains(E e)                {
55     Storage<E> runner = root;
56     boolean    result = false;
57     if ( e == null )
58         return false;
59     if ( root == null )
60         result = false;
61     else {
62         while ( ( runner.payload != null ) &&
63             ( ! ( result = runner.payload.equals(e) ) ) )
64             {
65                 runner = runner.next;
66             }
67     }
68     return result;
69 }
70 public int size()    {
71     return nElements;
72 }
73
74 public boolean add(E e)    {
75     boolean rValue = true;
76     if ( e == null )
77         return false;
78     if ( root == null )    {
79         root                = new Storage<E>();
80         root.payload        = e;
81         root.next           = new Storage<E>();
82         next                = root;
83     } else {
84         if ( contains(e) )
85             rValue = false;
86         else {
87             Storage<E> toAddAtFront = new Storage<E>();
88             toAddAtFront.payload = e;
89             toAddAtFront.next = root;
90             root = toAddAtFront;
91             rValue = true;
92         }
93     }
94
95     if ( rValue )
96         nElements ++;
```

```
97         return rValue;
98     }
99     public void clear() {
100         root = null;
101         nElements = 0;
102     }
103     E element() {
104         return root.payload;
105     }
106     public E remove(E o) {
107         if ( o == null )
108             return null;
109         Storage<E> index = root;
110         E element = null;
111         int position = -1;
112         if ( ( root == null ) || ( size() == 0 ) )
113             element = null;
114         else if ( root.payload.equals(o) ) {
115             element = root.payload;
116             nElements --;
117             root = root.next;
118         } else {
119             index = root;
120             while ( ( index.next != null )
121                 && ( index.next.payload != null )
122                 && ! ( index.next.payload.equals(o) ) ) {
123                 index = index.next;
124             }
125             element = index.next.payload;
126             index.next = index.next.next;
127             nElements --;
128         }
129         return element;
130     }
131
132     public String toString() {
133         Storage<E> index = root;
134         String result = "# of elements: " + size() + " ";
135         int counter = 0;
136         if ( root == null )
137             return "";
138         do {
139             if ( index == null )
140                 result = result + "->null";
141             else
142                 result = result + "->" + index.payload;
143             index = index.next;
144         } while ( index != null );
145         return result;
146     }
147 }
148
```

Source Code: Src/26\_sol/Storage\_1.java

```
1  class TestStorage {
2      public static boolean testAdd() {
3          Storage<String> aStorage = new Storage<String>();
4          String theStrings[] = { "a", "b", "c" };
5          boolean rValue = true;
6          for ( int index = 0; index < theStrings.length; index ++ )
7              aStorage.add(theStrings[index]);
8
9          for ( int index = 0; index < theStrings.length; index ++ ) { // test if
10              rValue &= aStorage.remove(theStrings[index]).equals(theStrings[ind
11          }
12
13          aStorage.clear();
14          rValue &= aStorage.add(theStrings[0]);
15          rValue &= aStorage.size() == 1;
16
17          return rValue;
18      }
19      public static boolean testClear() {
20          Storage<String> aStorage = new Storage<String>();
21          String theStrings[] = { "a", "b", "c" };
22          boolean rValue = true;
23
24          aStorage.add("a");
25          aStorage.clear();
26          rValue &= aStorage.size() == 0;
27          aStorage.add("a");
28          rValue &= aStorage.size() == 1;
29
30          return rValue;
31      }
32      public static boolean testContains() {
33          Storage<String> aStorage = new Storage<String>();
34          String theStrings[] = { "a", "b", "c", "d" };
35          boolean rValue = true;
36
37          rValue &= ! aStorage.contains(theStrings[0]);
38          aStorage.add(theStrings[0]);
39          rValue &= aStorage.contains(theStrings[0]);
40          rValue &= ! aStorage.contains(theStrings[1]);
41
42          aStorage.add(theStrings[1]);
43          rValue &= aStorage.contains(theStrings[1]);
44
45          aStorage.add(theStrings[2]);
46          aStorage.add(theStrings[3]);
47          rValue &= aStorage.contains(theStrings[2]);
48          rValue &= aStorage.contains(theStrings[3]);
49
50          return rValue;
51      }
52      public static void addArray(Storage<String> aStorage, String[] theStrings) {
53          for ( int index = 0; index < theStrings.length; index ++ )
```

```
54         aStorage.add(theStrings[index]);
55     }
56     public static boolean testAddAll() {
57         Storage<String> aStorage = new Storage<String>();
58         Storage<String> bStorage = new Storage<String>();
59         Storage<String> cStorage = new Storage<String>();
60         String theAStrings[] = { "a", "b", "c" };
61         String theBStrings[] = { "A", "B", "C" };
62         String theCStrings[] = { "AA", "BB", "CC" };
63         addArray(bStorage, theBStrings);
64         addArray(cStorage, theCStrings);
65         boolean rValue = true;
66         rValue &= aStorage.addAll(bStorage);
67         rValue &= aStorage.size() == theBStrings.length;
68
69         rValue &= aStorage.addAll(cStorage);
70         rValue &= aStorage.size() == theBStrings.length + theCStrings.length;
71
72         rValue &= ! aStorage.addAll(aStorage);
73         return rValue;
74     }
75     public static boolean testRemove() {
76         String theStrings[] = { "a", "b", "c" };
77         Storage<String> aStorage = new Storage<String>();
78         boolean rValue = true;
79
80         rValue &= null == aStorage.remove(theStrings[0] );
81         aStorage.add(theStrings[0] );
82         rValue &= aStorage.remove(theStrings[0] ).equals(theStrings[0]);
83         aStorage.add(theStrings[0]);
84         aStorage.add(theStrings[1]);
85         aStorage.remove(theStrings[0]);
86         rValue &= aStorage.remove(theStrings[1] ).equals(theStrings[1]);
87
88         addArray(aStorage, theStrings);
89         rValue &= aStorage.remove(theStrings[1] ).equals(theStrings[1]);
90
91         addArray(aStorage, theStrings);
92         rValue &= aStorage.remove(theStrings[2] ).equals(theStrings[2]);
93
94         return rValue;
95     }
96     public static boolean testRemoveAll() {
97         boolean rValue = true;
98         String theAStrings[] = { "a", "b", "c" };
99         String theBStrings[] = { "a", "b", "c" };
00         Storage<String> aStorage = new Storage<String>();
01         Storage<String> bStorage = new Storage<String>();
02
03         aStorage.add(theAStrings[0] );
04         aStorage.add(theAStrings[1] );
05         bStorage.add(theAStrings[0] );
06         rValue &= aStorage.removeAll(bStorage);
07     }
```

```
08         return rValue;
09     }
10     public static void exampleOfHowToUseIt( Storage<String> aStorage)    {
11         aStorage = new Storage<String>();
12         Storage<String> bStorage = new Storage<String>();
13         aStorage.add("a");
14         System.out.println("aStorage: " + aStorage );
15         bStorage.add("b");
16         if ( ! aStorage.addAll(aStorage) )
17             System.out.println("You can not add yourself to yourself.");
18         aStorage.addAll(bStorage);
19         aStorage.removeAll(bStorage);
20         System.out.println("aStorage: " + aStorage );
21     }
22 }
23 public static void test(Storage<String> aStorage)    {
24     if ( ! testRemove() )
25         System.err.println("testRemove failed");
26     if ( ! testRemoveAll() )
27         System.err.println("testRemove failed");
28     if ( ! testContains() )
29         System.err.println("testContains failed");
30     if ( ! testAdd() )
31         System.err.println("testAdd failed");
32     if ( ! testAddAll() )
33         System.err.println("testAddAll failed");
34     if ( ! testClear() )
35         System.err.println("testClear failed");
36 }
37 public static void main(String args[] )    {
38     test(new Storage<String>());
39     // exampleOfHowToUseIt(new Storage<String>());
40 }
41 }
42 }
43
```

Source Code: Src/26\_sol/TestStorage\_1.java



## 1.2. Homework 6.2 (10 Points)

**Objective:** Implementing based on interface and specs

**Grading:**

Correctness: You can lose up to 40% if your solution is not correct

Quality: You can lose up to 80% if your solution is poorly designed

Testing: You can lose up to 50% if your solution is not well tested

Explanation: You can lose up to 100% if your solution if you can not explain your solution during the grading session

**Homework Description:**

You have to implement a storage solution based on a interface and a functionality requirement.

**Explanation:**

Given is the following interface:

```
1
2     public interface StorageI<E>  {
3
4         public boolean add(E e);           // 2
5         public E get();
6         public void clear();               // 2 3
7         public boolean contains(E e);
8         public boolean isEmpty();
9         public void sort();                // 3
10        public int size();                 // 2 3
11        public String getClassName();
12
13    }
14
15
```

Source Code: Src/26/StorageI.java

After *sort()* is called, *get()* must return the elements in order, unless a other element is inserted.

**Your Work:**

Implement the storage class and test your implementation. The lines in the interface marked with 2, must perform O(1).

It might be useful to think about in which order you should develop the methods and how you will test the methods.

You can not use any existing Java class for this home work.

**Requirements:**

You have to provide a test environment for your work. You have to name your class *FastAdd.java*.

**Example:**

An example of a solution execution:

**Submission:**

```
% ssh glados.cs.rit.edu # or use queeg.cs.rit.edu if glados is down
# password
# go to the directory where your solution is ...
% try hpb-grd lab6-2 'All files required'
# you can see if your submission was successful:
```

```
# try -q hpb-grd lab6-2
```

**Solution:**

(This solution serves as the basis for the discussion in class. Sometimes there will be errors introduced to show common mistakes)

```
1      class Storage<E> implements StorageI<E> {
2
3          private Storage<E> root = null;
4          private Storage<E> next = null;
5          private Object[]   getArray = null;
6          private int        getIndex = -1;
7          private int        nElements = 0;
8          private Object[]   sorted;
9          E payLoad;
10
11         public Storage(){
12         }
13         public int size() {
14             return nElements;
15         }
16
17         public boolean add(E e) {
18             if ( root == null ) {
19                 root = new Storage<E>();
20                 root.payLoad = e;
21                 root.next = new Storage<E>();
22                 next = root;
23             } else {
24                 next.payLoad = e;
25                 next.next = new Storage<E>();
26             }
27             next = next.next;
28             nElements++;
29
30             return true;
31         }
32         public void add(int index, E element) {
33             if ( index > nElements )
34                 return;
35             if ( index == nElements )
36                 add(element);
37             else if ( index == 0 )
38                 addFirst(element);
39             else {
40                 nElements++;
41                 Storage<E> helper = root;
42                 for ( int counter = 1; counter < index ; counter ++ ) {
43                     helper = helper.next;
44                 }
45                 Storage<E> tmp = new Storage<E>();
46
47                 tmp.next = helper.next;
```

```
48         tmp.payLoad = element;
49         helper.next = tmp;
50
51     }
52
53 }
54 public void    addFirst(E e)        {
55     Storage<E> newRoot = new Storage<E>();
56     newRoot.payLoad = e;
57     newRoot.next = root;
58     nElements++;
59     root = newRoot;
60 }
61 public void    addLast(E e) {
62     add(e);
63 }
64 public void    clear()        {
65     root = null;
66     nElements = 0;
67 }
68 public E       element()      {
69     return root.payLoad;
70 }
71 public E       remove()       {
72     if ( root == null )
73         return null;
74     nElements--;
75     E element = root.payLoad;
76     root = root.next;
77     return element;
78 }
79 public E       remove(int index)    {
80     E element = null;
81     if ( root == null ) {
82         return null;
83     } else if ( index < nElements ) {
84         if ( index == 0 ) {
85             element = remove();
86         } else {
87             nElements--;
88             Storage<E> helper = root;
89             for ( int counter = 0; counter < index - 1 ; counter ++ )
90                 helper = helper.next;
91             }
92             element = helper.next.payLoad;
93             helper.next = helper.next.next;
94 // System.out.println("helper.payLoad: " + helper.payLoad);
95 // System.out.println("element: " + element);
96         }
97     }
98     return element;
99 }
100
101 public String toString()    {
```

```
02         Storage<E> index = root;
03         String result      = "# of elements: " + size() + "  ";
04         int counter        = 0;
05         if ( root == null )
06             return "";
07         do {
08             if ( index == null )
09                 result = result + "->null";
10             else
11                 result = result + "->" + index.payload;
12             index = index.next;
13         } while ( index != null );
14         return result;
15     }
16
17     public String getClassName() {
18         return getClass().getCanonicalName();
19     }
20     Object[]      toArray() {
21         if ( root == null )
22             return null;
23
24         Object[] theArray = new Object[size()];
25         Storage<E> index = root;
26         int runner = 0;
27
28         while ( index != null ) {
29             if ( index.payload != null )
30                 theArray[runner++] = index.payload;
31             index = index.next;
32         }
33         return theArray;
34     }
35     private static void printArray(Object[] theArray) {
36         for ( int index = 0; index < theArray.length; index ++ ) {
37             System.out.println(index + ":  " + theArray[index]);
38         }
39     }
40     private Object[] bubbleSort(Object[] theArray) {
41         for (int index = 0; index < theArray.length - 1; index++) {
42             for (int walker=0; walker < theArray.length - 1; walker++) {
43                 String leftString  = theArray[walker].toString();
44                 String rightString = theArray[walker + 1 ].toString();
45                 if ( leftString.compareTo(rightString) > 0 ) {
46                     E tmp = (E)theArray[walker];
47                     theArray[walker] = theArray[walker + 1];
48                     theArray[walker+1] = tmp;
49                 }
50             }
51         }
52         return theArray;
53     }
54     private void fillTheArray(Object[] theArray) {
55         for ( int index = 0; index < theArray.length; index ++ ) {
```

```
56         if ( theArray[index] != null )
57             add((E)theArray[index]);
58     }
59 }
60 public void sort() {
61     if ( root == null )
62         return;
63     Object[] theArray = toArray();
64     clear();
65     theArray = bubbleSort(theArray );
66     bubbleSort(theArray );
67     fillTheArray(theArray );
68 }
69
70 public boolean isEmpty() {
71     return size() == 0;
72 }
73 public boolean contains(E e) {
74     Storage<E> index = root;
75     boolean rValue = true;
76     boolean found = true;;
77
78     if ( root == null )
79         rValue = false;
80     else {
81         do {
82             found = e.equals(index.payload);
83             index = index.next;
84         } while ( ! found && ( index != null ) );
85     }
86
87     return rValue && found;
88 }
89 public E get() {
90     if ( root == null )
91         return null;
92     if ( getIndex == -1 ) {
93         getArray = toArray();
94         getIndex = 0;
95     }
96     if ( getIndex < getArray.length ) {
97         return (E)getArray[getIndex++];
98     } else {
99         getIndex = -1;
100        return null;
101    }
102 }
103 public static boolean testAddIndex() {
104     Storage<String> aStorage = new Storage<String>();
105     String theStrings[] = { "a", "b", "c" };
106     boolean rValue = true;
107     for ( int index = 0; index < theStrings.length; index ++ )
108         aStorage.add(theStrings[index]);
109     int size;
```

```
10
11         String theItem = "0";
12         aStorage.add(0, theItem);
13     rValue &= aStorage.remove() == theItem;
14         theItem = "1";
15         size = aStorage.size();
16         aStorage.add(size, theItem);
17     rValue &= aStorage.remove(size) == theItem;
18         size = aStorage.size();
19         aStorage.add(size + 1, theItem);
20     rValue &= aStorage.size() == size;
21         theItem = "2";
22         aStorage.add(1, theItem);
23     rValue &= aStorage.remove(1) == theItem;
24
25     return rValue;
26 }
27 public static boolean testRemoveIndex()    {
28     Storage<String> aStorage = new Storage<String>();
29     String theStrings[] = { "a", "b", "c" };
30     boolean rValue = true;
31     for ( int index = 0; index < theStrings.length; index ++ )
32         aStorage.add(theStrings[index]);
33     rValue &= aStorage.remove(aStorage.size()) == null;
34     rValue &= aStorage.remove(2).equals(theStrings[2]);
35     rValue &= aStorage.remove(1).equals(theStrings[1]);
36     rValue &= aStorage.remove(0).equals(theStrings[0]);
37     rValue &= aStorage.remove(0) == null;
38
39     aStorage.add("c");
40
41     return rValue;
42 }
43 public static boolean testAdd()    {
44     Storage<String> aStorage = new Storage<String>();
45     String theStrings[] = { "a", "b", "c" };
46     boolean rValue = true;
47     for ( int index = 0; index < theStrings.length; index ++ )
48         aStorage.add(theStrings[index]);
49     for ( int index = 0; index < theStrings.length; index ++ )
50         rValue &= aStorage.remove().equals(theStrings[index]);
51     rValue &= aStorage.remove() == null;
52     aStorage.add("c");
53
54     return rValue;
55 }
56 public static boolean testClassName()    {
57     Storage<String> aStorage = new Storage<String>();
58
59     return "Storage".equals(aStorage.getClassName());
60 }
61 public static boolean testClear()    {
62     Storage<String> aStorage = new Storage<String>();
63     boolean rValue = true;
```

```
64
65         rValue &= aStorage.remove() == null;
66             aStorage.add("a");
67             aStorage.clear();
68         rValue &= aStorage.size() == 0;
69
70         return rValue;
71     }
72     public static boolean testContains()      {
73         Storage<String> aStorage = new Storage<String>();
74         String theStrings[] = { "a", "b", "c" };
75
76         boolean rValue = true;
77         rValue &= ! aStorage.contains(theStrings[0]);
78         for ( int index = 0; index < theStrings.length; index ++ )
79             aStorage.add(theStrings[index]);
80         rValue &= aStorage.contains(theStrings[0]);
81         rValue &= ! aStorage.contains("d");
82
83         return rValue;
84     }
85     public static boolean testToArray() {
86         Storage<String> aStorage = new Storage<String>();
87         String theStrings[] = { "a", "b", "c" };
88         boolean rValue = true;;
89
90         for ( int index = 0; index < theStrings.length; index ++ )
91             aStorage.add(theStrings[index]);
92         Object[] anArrry = aStorage.toArray();
93
94         rValue &= anArrry.length == aStorage.size();
95         rValue &= anArrry.length == theStrings.length;
96         return rValue;
97     }
98     public static boolean testGet()      {
99         Storage<String> aStorage = new Storage<String>();
100        String theStrings[] = { "a", "b", "c" };
101        boolean rValue = true;;
102
103        rValue &= aStorage.get() == null;
104
105        for ( int index = 0; index < theStrings.length; index ++ )
106            aStorage.add(theStrings[index]);
107        for ( int index = 0; index < theStrings.length; index ++ )
108            rValue &= aStorage.get().equals(theStrings[index]);
109        rValue &= aStorage.get() == null;
110
111        return rValue;
112    }
113    public static boolean testSort()      {
114        Storage<String> aStorage = new Storage<String>();
115        String theStrings[] = { "x", "t", "c" };
116        boolean rValue = true;;
117
```

```
18         for ( int index = 0; index < theStrings.length; index ++ )
19             aStorage.add(theStrings[index]);
20
21         aStorage.sort();
22
23         for ( int index = 0; index < aStorage.size() - 1; index ++ )    {
24             String leftString  = aStorage.get(); aStorage.remove();
25             String rightString = aStorage.get(); aStorage.remove();
26             rValue &= ( leftString.compareTo(rightString) <= 0 );
27         }
28         return rValue;
29     }
30     public static void exampleOfHowToUseIt( Storage<String> aStorage)    {
31         aStorage.add("a");
32         aStorage.add(0, "0");
33         aStorage.add(aStorage.size(), "1");
34         aStorage.add(aStorage.size() + 1, "2");
35         System.out.println("aStorage: " + aStorage );
36
37     }
38     public static void test(Storage<String>aStorage)    {
39         if ( ! testAdd() )
40             System.err.println("testAdd failed");
41         if ( ! testRemoveIndex() )
42             System.err.println("testRemoveIndex failed");
43         if ( ! testAddIndex() )
44             System.err.println("testAddIndex failed");
45         if ( ! testClear() )
46             System.err.println("testClear failed");
47         if ( ! testToArray() )
48             System.err.println("testToArray failed");
49         if ( ! testSort() )
50             System.err.println("testSort failed");
51         if ( ! testGet() )
52             System.err.println("testGet failed");
53         if ( ! testContains() )
54             System.err.println("testContains failed");
55         if ( ! testClassName() )
56             System.err.println("testClassName failed");
57     }
58     public static void main(String args[] )    {
59         test(new Storage<String>());
60         // exampleOfHowToUseIt(new Storage<String>());
61
62     }
63 }
64
65
66
67
68
```

Source Code: Src/26\_sol/Storage\_2.java



### 1.3. Homework 6.3 (10 Points)

**Objective:** Implementing based on interface and specs

**Grading:**

Correctness: You can lose up to 40% if your solution is not correct

Quality: You can lose up to 80% if your solution is poorly designed

Testing: You can lose up to 50% if your solution is not well tested

Explanation: You can lose up to 100% if your solution if you can not explain your solution during the grading session

**Homework Description:**

You have to implement a storage solution based on a interface and a functionality requirement.

**Explanation:**

Given is the following interface:

```
1
2     public interface StorageI<E>  {
3
4         public boolean add(E e);           // 2
5         public E get();
6         public void clear();               // 2 3
7         public boolean contains(E e);
8         public boolean isEmpty();
9         public void sort();                // 3
10        public int size();                 // 2 3
11        public String getClassName();
12
13    }
14
15
```

Source Code: Src/26/StorageI.java

After *sort()* is called, *get()* must return the elements in order, unless a other element is inserted.

**Your Work:**

Implement the storage class and test your implementation. The lines in the interface marked with 3, must perform  $O(1)$ . 3 is the only difference between hw 6.2 and hw 6.3. You will need to develop a different underlying data structure.

It might be useful to think about in which order you should develop the methods and how you will test the methods.

You can not use any existing Java class for this home work.

**Requirements:**

You have to provide a test environment for your work. You have to name your class *FastSort.java*.

**Example:**

An example of a solution execution:

**Submission:**

```
% ssh glados.cs.rit.edu # or use queeg.cs.rit.edu if glados is down
# password
# go to the directory where your solution is ...
% try hpb-grd lab6-3 'All files required'
# you can see if your submission was successful:
# try -q hpb-grd lab6-3
```

### Solution:

(This solution serves as the basis for the discussion in class. Sometimes there will be errors introduced to show common mistakes)

```
1      class Storage<E> implements StorageI<E> {
2
3          private Storage<E> root = null;
4          private Storage<E> next = null;
5          private Object[]   getArray = null;
6          private int        getIndex = -1;
7          private int        nElements = 0;
8          private Object[]   sorted;
9          E payLoad;
10
11         public Storage(){
12             }
13         public int size()    {
14             return nElements;
15         }
16
17         public boolean add(E e)    {
18             if ( root == null )    {
19                 root                = new Storage<E>();
20                 root.payLoad        = e;
21                 root.next           = new Storage<E>();
22             } else {
23                 Storage<E> insertAfter = findPosition(e);
24                 Storage<E> next = new Storage<E>();
25                 next.payLoad      = e;
26                 if ( insertAfter == null )        {           // new root
27                     next.next = root;
28                     root = next;
29                 } else {
30                     next.next      = insertAfter.next;
31                     insertAfter.next= next;
32                 }
33             }
34             nElements++;
35
36             return true;
37         }
38         private Storage<E> findPosition(E element) {
39             Storage<E> runner = root;
40             Storage<E> followUp = null;
41             boolean keepGoing = true;
42             String insertString = element.toString();
```

```
43
44         do      {
45             String payLoadString  = runner.payload.toString();
46             if ( payLoadString.compareTo(insertString) < 0 )      {
47                 followUp = runner;
48                 runner    = runner.next;
49                 keepGoing = ( runner.payload != null );
50             } else {
51                 keepGoing = false;
52             }
53         } while ( keepGoing );
54         return followUp;
55     }
56     public void      add(int index, E element)      {
57         if ( index > nElements )
58             return;
59         if ( index == nElements )
60             add(element);
61         else if ( index == 0 )
62             addFirst(element);
63         else      {
64             nElements++;
65             Storage<E> helper = root;
66             for ( int counter = 1; counter < index ; counter ++ )      {
67                 helper = helper.next;
68             }
69             Storage<E> tmp = new Storage<E>();
70
71             tmp.next = helper.next;
72             tmp.payload = element;
73             helper.next = tmp;
74
75         }
76
77     }
78     public void      addFirst(E e)      {
79         Storage<E> newRoot = new Storage<E>();
80         newRoot.payload = e;
81         newRoot.next = root;
82         nElements++;
83         root = newRoot;
84     }
85     public void      addLast(E e) {
86         add(e);
87     }
88     public void      clear()      {
89         root = null;
90         nElements = 0;
91     }
92     public E          element()      {
93         return root.payload;
94     }
95     public E          remove()      {
96         if ( root == null )
```

```
97         return null;
98         nElements--;
99         E element = root.payload;
100        root = root.next;
101        return element;
102    }
103    public E      remove(int index)      {
104        E element = null;
105        if ( root == null ) {
106            return null;
107        } else if ( index < nElements ) {
108            if ( index == 0 )            {
109                element = remove();
110            } else {
111                nElements--;
112                Storage<E> helper = root;
113                for ( int counter = 0; counter < index - 1 ; counter ++ )
114                    helper = helper.next;
115            }
116            element = helper.next.payload;
117            helper.next = helper.next.next;
118            // System.out.println("helper.payload: " + helper.payload);
119            // System.out.println("element: " + element);
120        }
121    }
122    return element;
123    }
124
125    public String toString()      {
126        Storage<E> index = root;
127        String result      = "# of elements: " + size() + " ";
128        int counter        = 0;
129        if ( root == null )
130            return "";
131        do {
132            if ( index == null )
133                result = result + "->null";
134            else
135                result = result + "->" + index.payload;
136            index = index.next;
137        } while ( index != null );
138        return result;
139    }
140
141    public String getClassName()      {
142        return getClass().getCanonicalName();
143    }
144    Object[]      toArray() {
145        if ( root == null )
146            return null;
147
148        Object[] theArray = new Object[size()];
149        Storage<E> index = root;
150        int runner = 0;
```

```
51
52     while ( index != null ) {
53         if ( index.payload != null )
54             theArray[runner++] = index.payload;
55         index = index.next;
56     }
57     return theArray;
58 }
59 private static void printArray(Object[] theArray) {
60     for ( int index = 0; index < theArray.length; index ++ )      {
61         System.out.println(index + ":  " + theArray[index]);
62     }
63 }
64 private Object[] bubbleSort(Object[] theArray)      {
65     for (int index = 0; index < theArray.length - 1; index++)      {
66         for (int walker=0; walker < theArray.length - 1; walker++) {
67             String leftString  = theArray[walker].toString();
68             String rightString = theArray[walker + 1 ].toString();
69             if ( leftString.compareTo(rightString) > 0 )      {
70                 E tmp = (E)theArray[walker];
71                 theArray[walker] = theArray[walker + 1];
72                 theArray[walker+1] = tmp;
73             }
74         }
75     }
76     return theArray;
77 }
78 private void fillTheArray(Object[] theArray)      {
79     for ( int index = 0; index < theArray.length; index ++ )      {
80         if ( theArray[index] != null )
81             add((E)theArray[index]);
82     }
83 }
84 public void sort() {
85     if ( root == null )
86         return;
87     Object[] theArray = toArray();
88     clear();
89     theArray = bubbleSort(theArray );
90     bubbleSort(theArray );
91     fillTheArray(theArray );
92 }
93
94 public boolean isEmpty() {
95     return size() == 0;
96 }
97 public boolean contains(E e) {
98     Storage<E> index = root;
99     boolean rValue = true;
00     boolean found  = true;;
01
02     if ( root == null )
03         rValue =  false;
04     else      {
```

```
05             do          {
06                 found = e.equals(index.payload);
07                 index = index.next;
08             } while ( ! found && ( index != null ) );
09         }
10
11         return rValue && found;
12     }
13     public E get() {
14         if ( root == null )
15             return null;
16         if ( getIndex == -1 ) {
17             getArray = toArray();
18             getIndex = 0;
19         }
20         if ( getIndex < getArray.length ) {
21             return (E)getArray[getIndex++];
22         } else {
23             getIndex = -1;
24             return null;
25         }
26     }
27     public static boolean testAddIndex() {
28         Storage<String> aStorage = new Storage<String>();
29         String theStrings[] = { "a", "b", "c" };
30         boolean rValue = true;
31         for ( int index = 0; index < theStrings.length; index ++ )
32             aStorage.add(theStrings[index]);
33         int size;
34         String theItem = "0";
35         aStorage.add(0, theItem);
36         rValue &= aStorage.remove() == theItem;
37         theItem = "1";
38         size = aStorage.size();
39         aStorage.add(size, theItem);
40         rValue &= aStorage.remove(size) == theItem;
41         size = aStorage.size();
42         aStorage.add(size + 1, theItem);
43         rValue &= aStorage.size() == size;
44         theItem = "2";
45         aStorage.add(1, theItem);
46         rValue &= aStorage.remove(1) == theItem;
47
48         return rValue;
49     }
50     public static boolean testRemoveIndex() {
51         Storage<String> aStorage = new Storage<String>();
52         String theStrings[] = { "a", "b", "c" };
53         boolean rValue = true;
54         for ( int index = 0; index < theStrings.length; index ++ )
55             aStorage.add(theStrings[index]);
56         rValue &= aStorage.remove(aStorage.size()) == null;
57         rValue &= aStorage.remove(2).equals(theStrings[2]);
58         rValue &= aStorage.remove(1).equals(theStrings[1]);
```

```
59         rValue &= aStorage.remove(0).equals(theStrings[0]);
60         rValue &= aStorage.remove(0) == null;
61
62         aStorage.add("c");
63
64         return rValue;
65     }
66     public static boolean testAdd()          {           // weak test
67         Storage<String> aStorage = new Storage<String>();
68         String theStrings[] = { "t", "x", "c" };
69         boolean rValue = true;
70         for ( int index = 0; index < theStrings.length; index ++ )      {
71             aStorage.add(theStrings[index]);
72         }
73         rValue &= aStorage.remove().equals(theStrings[2]);
74         rValue &= aStorage.remove().equals(theStrings[0]);
75         rValue &= aStorage.remove().equals(theStrings[1]);
76
77         rValue &= aStorage.remove() == null;
78
79         return rValue;
80     }
81     public static boolean testClassName()      {
82         Storage<String> aStorage = new Storage<String>();
83
84         return "Storage".equals(aStorage.getClassName());
85     }
86     public static boolean testClear()    {
87         Storage<String> aStorage = new Storage<String>();
88         boolean rValue = true;
89
90         rValue &= aStorage.remove() == null;
91         aStorage.add("a");
92         aStorage.clear();
93         rValue &= aStorage.size() == 0;
94
95         return rValue;
96     }
97     public static boolean testContains()      {
98         Storage<String> aStorage = new Storage<String>();
99         String theStrings[] = { "a", "b", "c" };
100
101         boolean rValue = true;
102         rValue &= ! aStorage.contains(theStrings[0]);
103         for ( int index = 0; index < theStrings.length; index ++ )
104             aStorage.add(theStrings[index]);
105         rValue &= aStorage.contains(theStrings[0]);
106         rValue &= ! aStorage.contains("d");
107
108         return rValue;
109     }
110     public static boolean testToArray() {
111         Storage<String> aStorage = new Storage<String>();
112         String theStrings[] = { "a", "b", "c" };
```

```
13         boolean rValue = true;;
14
15         for ( int index = 0; index < theStrings.length; index ++ )
16             aStorage.add(theStrings[index]);
17         Object[] anArrry = aStorage.toArray();
18
19         rValue &= anArrry.length == aStorage.size();
20         rValue &= anArrry.length == theStrings.length;
21         return rValue;
22     }
23     public static boolean testGet()      {
24         Storage<String> aStorage = new Storage<String>();
25         String theStrings[] = { "a", "b", "c" };
26         boolean rValue = true;;
27
28         rValue &= aStorage.get() == null;
29
30         for ( int index = 0; index < theStrings.length; index ++ )
31             aStorage.add(theStrings[index]);
32         for ( int index = 0; index < theStrings.length; index ++ )
33             rValue &= aStorage.get().equals(theStrings[index]);
34         rValue &= aStorage.get() == null;
35
36         return rValue;
37     }
38     public static boolean testSort()     {
39         Storage<String> aStorage = new Storage<String>();
40         String theStrings[] = { "x", "t", "c" };
41         boolean rValue = true;;
42
43         for ( int index = 0; index < theStrings.length; index ++ )
44             aStorage.add(theStrings[index]);
45
46         aStorage.sort();
47
48         for ( int index = 0; index < aStorage.size() - 1; index ++ )    {
49             String leftString  = aStorage.get(); aStorage.remove();
50             String rightString = aStorage.get(); aStorage.remove();
51             rValue &= ( leftString.compareTo(rightString) <= 0 );
52         }
53         return rValue;
54     }
55     public static void exampleOfHowToUseIt( Storage<String> aStorage)    {
56         aStorage.add("a");
57         aStorage.add(0, "0");
58         aStorage.add(aStorage.size(), "1");
59         aStorage.add(aStorage.size() + 1, "2");
60         System.out.println("aStorage: " + aStorage );
61
62     }
63     public static void test(Storage<String> aStorage)    {
64         if ( ! testAdd() )
65             System.err.println("testAdd failed");
66     }
/*
```



```
67         if ( ! testRemoveIndex() )
68             System.err.println("testRemoveIndex failed");
69         if ( ! testAddIndex() )
70             System.err.println("testAddIndex failed");
71         if ( ! testClear() )
72             System.err.println("testClear failed");
73         if ( ! testToArray() )
74             System.err.println("testToArray failed");
75         if ( ! testSort() )
76             System.err.println("testSort failed");
77         if ( ! testGet() )
78             System.err.println("testGet failed");
79         if ( ! testContains() )
80             System.err.println("testContains failed");
81         if ( ! testClassName() )
82             System.err.println("testClassName failed");
83     */
84     }
85     public static void main(String args[] )      {
86         test(new Storage<String>());
87         // exampleOfHowToUseIt(new Storage<String>());
88     }
89 }
90
91
92
93
94
95
```

Source Code: Src/26\_sol/Storage\_3.java

```
1     class TestStorage {
2         public static boolean testAdd()      {
3             Storage<String> aStorage = new Storage<String>();
4             String theStrings[] = { "a", "b", "c" };
5             boolean rValue = true;
6             for ( int index = 0; index < theStrings.length; index ++ )
7                 aStorage.add(theStrings[index]);
8
9             for ( int index = 0; index < theStrings.length; index ++ ) {      // test if
10                 rValue &= aStorage.remove(theStrings[index]).equals(theStrings[ind
11             }
12
13             aStorage.clear();
14             rValue &= aStorage.add(theStrings[0]);
15             rValue &= aStorage.size() == 1;
16
17             return rValue;
18         }
19         public static boolean testClear()    {
20             Storage<String> aStorage = new Storage<String>();
21             String theStrings[] = { "a", "b", "c" };
22             boolean rValue = true;
```

```
23
24     aStorage.add("a");
25     aStorage.clear();
26     rValue &= aStorage.size() == 0;
27     aStorage.add("a");
28     rValue &= aStorage.size() == 1;
29
30     return rValue;
31 }
32 public static boolean testContains() {
33     Storage<String> aStorage = new Storage<String>();
34     String theStrings[] = { "a", "b", "c", "d" };
35     boolean rValue = true;
36
37     rValue &= ! aStorage.contains(theStrings[0]);
38     aStorage.add(theStrings[0]);
39     rValue &= aStorage.contains(theStrings[0]);
40     rValue &= ! aStorage.contains(theStrings[1]);
41
42     aStorage.add(theStrings[1]);
43     rValue &= aStorage.contains(theStrings[1]);
44
45     aStorage.add(theStrings[2]);
46     aStorage.add(theStrings[3]);
47     rValue &= aStorage.contains(theStrings[2]);
48     rValue &= aStorage.contains(theStrings[3]);
49
50     return rValue;
51 }
52 public static void addArray(Storage<String> aStorage, String[] theStrings) {
53     for ( int index = 0; index < theStrings.length; index ++ )
54         aStorage.add(theStrings[index]);
55 }
56 public static boolean testAddAll() {
57     Storage<String> aStorage = new Storage<String>();
58     Storage<String> bStorage = new Storage<String>();
59     Storage<String> cStorage = new Storage<String>();
60     String theAstrings[] = { "a", "b", "c" };
61     String theBstrings[] = { "A", "B", "C" };
62     String theCstrings[] = { "AA", "BB", "CC" };
63     addArray(bStorage, theBstrings);
64     addArray(cStorage, theCstrings);
65     boolean rValue = true;
66     rValue &= aStorage.addAll(bStorage);
67     rValue &= aStorage.size() == theBstrings.length;
68
69     rValue &= aStorage.addAll(cStorage);
70     rValue &= aStorage.size() == theBstrings.length + theCstrings.length;
71
72     rValue &= ! aStorage.addAll(aStorage);
73     return rValue;
74 }
75 public static boolean testRemove() {
76     String theStrings[] = { "a", "b", "c" };
```

```
77     Storage<String> aStorage = new Storage<String>();
78     boolean rValue = true;
79
80     rValue &= null == aStorage.remove(theStrings[0] );
81         aStorage.add(theStrings[0] );
82     rValue &= aStorage.remove(theStrings[0] ).equals(theStrings[0]);
83         aStorage.add(theStrings[0]);
84         aStorage.add(theStrings[1]);
85         aStorage.remove(theStrings[0]);
86     rValue &= aStorage.remove(theStrings[1] ).equals(theStrings[1]);
87
88     addArray(aStorage, theStrings);
89     rValue &= aStorage.remove(theStrings[1] ).equals(theStrings[1]);
90
91     addArray(aStorage, theStrings);
92     rValue &= aStorage.remove(theStrings[2] ).equals(theStrings[2]);
93
94     return rValue;
95 }
96 public static boolean testRemoveAll()      {
97     boolean rValue = true;
98     String theAStrings[] = { "a", "b", "c" };
99     String theBStrings[] = { "a", "b", "c" };
00     Storage<String> aStorage = new Storage<String>();
01     Storage<String> bStorage = new Storage<String>();
02
03     aStorage.add(theAStrings[0] );
04     aStorage.add(theAStrings[1] );
05     bStorage.add(theAStrings[0] );
06     rValue &= aStorage.removeAll(bStorage);
07
08     return rValue;
09 }
10 public static void exampleOfHowToUseIt( Storage<String> aStorage)  {
11     aStorage = new Storage<String>();
12     Storage<String> bStorage = new Storage<String>();
13     aStorage.add("a");
14     System.out.println("aStorage: " + aStorage );
15     bStorage.add("b");
16     if ( ! aStorage.addAll(aStorage) )
17         System.out.println("You can not add yourself to yourself.");
18     aStorage.addAll(bStorage);
19     aStorage.removeAll(bStorage);
20     System.out.println("aStorage: " + aStorage );
21
22 }
23 public static void test(Storage<String> aStorage)  {
24     if ( ! testRemove() )
25         System.err.println("testRemove failed");
26     if ( ! testRemoveAll() )
27         System.err.println("testRemove failed");
28     if ( ! testContains() )
29         System.err.println("testContains failed");
30     if ( ! testAdd() )
```

```
31             System.err.println("testAdd failed");
32         if ( ! testAddAll() )
33             System.err.println("testAddAll failed");
34         if ( ! testClear() )
35             System.err.println("testClear failed");
36     }
37     public static void main(String args[] )    {
38         test(new Storage<String>());
39         // exampleOfHowToUseIt(new Storage<String>());
40
41     }
42 }
43
```

Source Code: Src/26\_sol/TestStorage\_1.java

