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

$$\textit{Union} \bigcup \text{ and } \textit{intersection} \bigcap.$$

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 tour 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)

```
/*
 * you can not add or test for null elements in this implementation
 */
public class Storage<E> implements StorageI<E> {

    private Storage<E> root;
    private Storage<E> next;
    private int        nElements = 0;

    E payLoad;

    public Storage(){
    }
    public  String getClassName()       {
                return "Storage";
    }
    public  Object[] toArray()  {
        Object[] anArray = new Object[size()];
        Storage<E> runner = root;
        int index = 0;
        while ( runner.payLoad != null )        {
                anArray[index++] = runner.payLoad;
                runner = runner.next;
        }
        return anArray;
    }
    public  boolean addAll(Storage<? extends E> theSetToAdd) {

        boolean rValue = true;
        if ( this == theSetToAdd ) {
                return false;
        }
        Storage<? extends E> runner = theSetToAdd.root;
        while ( runner.payLoad != null )        {
                rValue &= add(runner.payLoad );
                runner = runner.next;
        }
        return rValue;
    }
    public  boolean removeAll(Storage<? extends E> theSetToRemove) {
        boolean rValue = true;
        if ( this == theSetToRemove )   {
```

```
                        return false;
                }
                Storage<? extends E> runner = theSetToRemove.root;
                while ( runner.payLoad != null )        {
                        if ( contains(runner.payLoad) ) {
                                rValue |= null != remove(runner.payLoad);
                                runner = runner.next;
                        }
                }
                return rValue;
        }
        public  boolean contains(E e)       {
                Storage<E> runner = root;
                boolean    result = false;
                if ( e == null )
                        return false;
                if ( root == null )
                        result = false;
                else    {
                        while ( ( runner.payLoad != null ) &&
                                ( ! ( result = runner.payLoad.equals(e) )  )
                              ) {
                                runner = runner.next;
                        }
                }
                return result;
        }
        public int size()   {
                return nElements;
        }

        public boolean add(E e)     {
                boolean rValue = true;
                if ( e == null )
                        return false;
                if ( root == null )     {
                        root            = new Storage<E>();
                        root.payLoad    = e;
                        root.next       = new Storage<E>();
                        next            = root;
                } else {
                        if ( contains(e) )
                                rValue  =  false;
                        else {
                                Storage<E> toAddAtFront = new Storage<E>();
                                toAddAtFront.payLoad = e;
                                toAddAtFront.next = root;
                                root = toAddAtFront;
                                rValue  =  true;
                        }
                }

                if ( rValue )
                        nElements ++;
                return rValue;
        }
        public void    clear()      {
                root = null;
                nElements = 0;
        }
        E element() {
                return root.payLoad;
        }
        public E remove(E o)        {
                if ( o == null )
                        return null;
                Storage<E> index = root;
                E element = null;
                int position = -1;
                if ( ( root == null ) || ( size() == 0 ) )
                        element = null;
                else if ( root.payLoad.equals(o) )      {
                        element = root.payLoad;
                        nElements --;
                        root = root.next;
                } else {
                        index = root;
                        while (     ( index.next != null )
                                &&  ( index.next.payLoad != null )
                                &&  ! ( index.next.payLoad.equals(o) ) ) {
                                index = index.next;
                        }
                        element = index.next.payLoad;
                        index.next = index.next.next;
                        nElements --;
                }
                return element;
        }

        public String toString()    {
                Storage<E> index = root;
                String result    = "# of elements: " + size() + "  ";
                int counter      = 0;
                if ( root == null )
                        return "";
                do      {
                        if ( index == null )
                                result = result + "->null";
                        else
                                result = result + "->" + index.payLoad;
                        index = index.next;
                } while ( index != null );
                return result;
        }
    }

```

Source Code: Src/26_sol/Storage_1.java

```
class TestStorage  {
    public static boolean testAdd()     {
        Storage<String> aStorage = new Storage<String>();
        String theStrings[] = { "a", "b", "c" };
        boolean rValue = true;
        for ( int index = 0; index < theStrings.length; index ++ )
                aStorage.add(theStrings[index]);

        for ( int index = 0; index < theStrings.length; index ++ ) {    // test if
                rValue &= aStorage.remove(theStrings[index]).equals(theStrings[ind
        }

        aStorage.clear();
        rValue &= aStorage.add(theStrings[0]);
        rValue &= aStorage.size() == 1;

        return rValue;
    }
    public static boolean testClear()   {
        Storage<String> aStorage = new Storage<String>();
        String theStrings[] = { "a", "b", "c" };
        boolean rValue = true;

        aStorage.add("a");
        aStorage.clear();
        rValue &= aStorage.size() == 0;
        aStorage.add("a");
        rValue &= aStorage.size() == 1;

        return rValue;
    }
    public static boolean testContains()        {
        Storage<String> aStorage = new Storage<String>();
        String theStrings[] = { "a", "b", "c", "d" };
        boolean rValue = true;

        rValue &= ! aStorage.contains(theStrings[0]);
        aStorage.add(theStrings[0]);
        rValue &= aStorage.contains(theStrings[0]);
        rValue &= ! aStorage.contains(theStrings[1]);

        aStorage.add(theStrings[1]);
        rValue &= aStorage.contains(theStrings[1]);

        aStorage.add(theStrings[2]);
        aStorage.add(theStrings[3]);
        rValue &= aStorage.contains(theStrings[2]);
        rValue &= aStorage.contains(theStrings[3]);

        return rValue;
    }
    public static void addArray(Storage<String> aStorage, String[] theStrings)  {
        for ( int index = 0; index < theStrings.length; index ++ )
```

```
                        aStorage.add(theStrings[index]);
        }
        public static boolean testAddAll()  {
            Storage<String> aStorage = new Storage<String>();
            Storage<String> bStorage = new Storage<String>();
            Storage<String> cStorage = new Storage<String>();
            String theAstrings[] = { "a", "b", "c" };
            String theBstrings[] = { "A", "B", "C" };
            String theCstrings[] = { "AA", "BB", "CC" };
            addArray(bStorage, theBstrings);
            addArray(cStorage, theCstrings);
            boolean rValue = true;
            rValue &= aStorage.addAll(bStorage);
            rValue &= aStorage.size() == theBstrings.length;

            rValue &= aStorage.addAll(cStorage);
            rValue &= aStorage.size() == theBstrings.length + theCstrings.length;

            rValue &= ! aStorage.addAll(aStorage);
            return rValue;
        }
        public static boolean testRemove()  {
            String theStrings[] = { "a", "b", "c" };
            Storage<String> aStorage = new Storage<String>();
            boolean rValue = true;

            rValue &= null ==  aStorage.remove(theStrings[0] );
                    aStorage.add(theStrings[0] );
            rValue &=  aStorage.remove(theStrings[0] ).equals(theStrings[0]);
                    aStorage.add(theStrings[0]);
                    aStorage.add(theStrings[1]);
                    aStorage.remove(theStrings[0]);
            rValue &=  aStorage.remove(theStrings[1] ).equals(theStrings[1]);

            addArray(aStorage, theStrings);
            rValue &=  aStorage.remove(theStrings[1] ).equals(theStrings[1]);

            addArray(aStorage, theStrings);
            rValue &=  aStorage.remove(theStrings[2] ).equals(theStrings[2]);

            return rValue;
        }
        public static boolean testRemoveAll()       {
            boolean rValue = true;
            String theAstrings[] = { "a", "b", "c" };
            String theBstrings[] = { "a", "b", "c" };
            Storage<String> aStorage = new Storage<String>();
            Storage<String> bStorage = new Storage<String>();

            aStorage.add(theAstrings[0] );
            aStorage.add(theAstrings[1] );
            bStorage.add(theAstrings[0] );
            rValue &=  aStorage.removeAll(bStorage);

            return rValue;
        }
        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>());

        }
    }

```

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:

```

public interface StorageI<E>  {

        public boolean add(E e);        // 2
        public E get();
        public void clear();            // 2 3
        public boolean contains(E e);
        public boolean isEmpty();
        public void sort();             // 3
        public int size();              // 2 3
        public String getClassName();

}


```

```
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)

```
class Storage<E> implements StorageI<E> {

    private Storage<E> root = null;
    private Storage<E> next = null;
    private Object[]   getArray = null;
    private int        getIndex = -1;
    private int        nElements = 0;
    private  Object[]  sorted;
    E payLoad;

    public Storage(){
    }
    public int size()   {
        return nElements;
    }

    public boolean add(E e)     {
        if ( root == null )     {
                root            = new Storage<E>();
                root.payLoad    = e;
                root.next       = new Storage<E>();
                next            = root;
        } else {
                next.payLoad    = e;
                next.next       = new Storage<E>();
        }
        next = next.next;
        nElements++;

        return true;
    }
    public void    add(int index, E element)    {
        if ( index > nElements )
                return;
        if ( index == nElements )
                add(element);
        else if ( index == 0 )
                addFirst(element);
        else    {
                nElements++;
                Storage<E> helper = root;
                for ( int counter = 1; counter < index ; counter ++ )   {
                        helper = helper.next;
                }
                Storage<E> tmp = new Storage<E>();

                tmp.next = helper.next;
                tmp.payLoad = element;
                helper.next = tmp;

        }

    }
    public void    addFirst(E e)        {
        Storage<E> newRoot = new Storage<E>();
        newRoot.payLoad = e;
        newRoot.next = root;
        nElements++;
        root = newRoot;
    }
    public void    addLast(E e) {
        add(e);
    }
    public void    clear()      {
        root = null;
        nElements = 0;
    }
    public E       element()    {
        return root.payLoad;
    }
    public E       remove()     {
        if ( root == null )
         return null;
        nElements--;
        E element = root.payLoad;
        root = root.next;
        return element;
    }
    public E       remove(int index)    {
        E element = null;
        if ( root == null ) {
                return null;
        } else if ( index < nElements ) {
                if ( index == 0 )       {
                        element = remove();
                } else  {
                        nElements--;
                        Storage<E> helper = root;
                        for ( int counter = 0; counter < index - 1 ; counter ++ )
                                helper = helper.next;
                        }
                        element = helper.next.payLoad;
                        helper.next = helper.next.next;
// System.out.println("helper.payLoad: " + helper.payLoad);
// System.out.println("element: " + element);
                }
        }
        return element;
    }

    public String toString()    {
        Storage<E> index = root;
        String result    = "# of elements: " + size() + "  ";
        int counter      = 0;
        if ( root == null )
                return "";
        do      {
                if ( index == null )
                        result = result + "->null";
                else
                        result = result + "->" + index.payLoad;
                index = index.next;
        } while ( index != null );
        return result;
    }

    public String getClassName()        {
        return getClass().getCanonicalName();
    }
    Object[]        toArray() {
        if ( root == null )
                return null;

        Object[] theArray = new Object[size()];
        Storage<E> index = root;
        int runner = 0;

        while ( index != null ) {
                if ( index.payLoad != null )
                        theArray[runner++] = index.payLoad;
                index = index.next;
        }
        return theArray;
    }
    private static  void printArray(Object[] theArray) {
        for ( int index = 0; index < theArray.length; index ++ )        {
                System.out.println(index + ":   " + theArray[index]);
        }
    }
    private Object[] bubbleSort(Object[] theArray)      {
        for (int index = 0; index < theArray.length - 1; index++)     {
                for (int walker=0; walker < theArray.length - 1; walker++)  {
                        String leftString  = theArray[walker].toString();
                        String rightString = theArray[walker + 1 ].toString();
                        if ( leftString.compareTo(rightString) > 0 )    {
                                E tmp = (E)theArray[walker];
                                theArray[walker] = theArray[walker + 1];
                                theArray[walker+1] = tmp;
                        }
                }
        }
        return theArray;
    }
    private void fillTheArray(Object[] theArray)        {
        for ( int index = 0; index < theArray.length; index ++ )        {
                if ( theArray[index] != null )
                        add((E)theArray[index]);
        }
    }
    public void sort()  {
        if ( root == null )
                return;
        Object[] theArray = toArray();
        clear();
        theArray = bubbleSort(theArray );
        bubbleSort(theArray );
        fillTheArray(theArray );

    }
    public boolean isEmpty() {
        return size() == 0;
    }
    public boolean contains(E e) {
        Storage<E> index = root;
        boolean rValue = true;
        boolean found  = true;;

        if ( root == null )
                rValue =  false;
        else    {
                do      {
                        found = e.equals(index.payLoad);
                        index = index.next;
                } while ( ! found && ( index != null ) );
        }

        return rValue && found;
    }
    public E get() {
        if  ( root == null )
                return null;
        if ( getIndex == -1 )   {
                getArray = toArray();
                getIndex = 0;
        }
        if ( getIndex < getArray.length )       {
                return (E)getArray[getIndex++];
        } else {
                getIndex = -1;
                return null;
        }
    }
    public static boolean testAddIndex()        {
        Storage<String> aStorage = new Storage<String>();
        String theStrings[] = { "a", "b", "c" };
        boolean rValue = true;
        for ( int index = 0; index < theStrings.length; index ++ )
                aStorage.add(theStrings[index]);
        int size;
```

```

			String theItem = "0";
			aStorage.add(0, theItem);
		rValue &= aStorage.remove() == theItem;
			theItem = "1";
			size = aStorage.size();
			aStorage.add(size, theItem);
		rValue &= aStorage.remove(size) == theItem;
			size = aStorage.size();
			aStorage.add(size + 1, theItem);
		rValue &= aStorage.size() == size;
			theItem = "2";
			aStorage.add(1, theItem);
		rValue &= aStorage.remove(1) == theItem;

		return rValue;
	}
	public static boolean testRemoveIndex()	{
		Storage<String> aStorage = new Storage<String>();
		String theStrings[] = { "a", "b", "c" };
		boolean rValue = true;
		for ( int index = 0; index < theStrings.length; index ++ )
			aStorage.add(theStrings[index]);
		rValue &= aStorage.remove(aStorage.size()) == null;
		rValue &= aStorage.remove(2).equals(theStrings[2]);
		rValue &= aStorage.remove(1).equals(theStrings[1]);
		rValue &= aStorage.remove(0).equals(theStrings[0]);
		rValue &= aStorage.remove(0) == null;

		aStorage.add("c");

		return rValue;
	}
	public static boolean testAdd()	{
		Storage<String> aStorage = new Storage<String>();
		String theStrings[] = { "a", "b", "c" };
		boolean rValue = true;
		for ( int index = 0; index < theStrings.length; index ++ )
			aStorage.add(theStrings[index]);
		for ( int index = 0; index < theStrings.length; index ++ )
			rValue &= aStorage.remove().equals(theStrings[index]);
		rValue &= aStorage.remove() == null;
		aStorage.add("c");

		return rValue;
	}
	public static boolean testClassName()		{
		Storage<String> aStorage = new Storage<String>();

		return "Storage".equals(aStorage.getClassName());
	}
	public static boolean testClear()	{
		Storage<String> aStorage = new Storage<String>();
		boolean rValue = true;
```

```

            rValue &= aStorage.remove() == null;
                    aStorage.add("a");
                    aStorage.clear();
            rValue &= aStorage.size() == 0;

            return rValue;
        }
        public static boolean testContains()        {
            Storage<String> aStorage = new Storage<String>();
            String theStrings[] = { "a", "b", "c" };

            boolean rValue = true;
            rValue &= ! aStorage.contains(theStrings[0]);
            for ( int index = 0; index < theStrings.length; index ++ )
                    aStorage.add(theStrings[index]);
            rValue &= aStorage.contains(theStrings[0]);
            rValue &= ! aStorage.contains("d");

            return rValue;
        }
        public static boolean testToArray() {
            Storage<String> aStorage = new Storage<String>();
            String theStrings[] = { "a", "b", "c" };
            boolean rValue = true;;

            for ( int index = 0; index < theStrings.length; index ++ )
                    aStorage.add(theStrings[index]);
            Object[] anArrry = aStorage.toArray();

            rValue &= anArrry.length == aStorage.size();
            rValue &= anArrry.length == theStrings.length;
            return rValue;
        }
        public static boolean testGet()     {
            Storage<String> aStorage = new Storage<String>();
            String theStrings[] = { "a", "b", "c" };
            boolean rValue = true;;

            rValue &= aStorage.get() == null;

            for ( int index = 0; index < theStrings.length; index ++ )
                    aStorage.add(theStrings[index]);
            for ( int index = 0; index < theStrings.length; index ++ )
                    rValue &= aStorage.get().equals(theStrings[index]);
            rValue &= aStorage.get() == null;

            return rValue;
        }
        public static boolean testSort()    {
            Storage<String> aStorage = new Storage<String>();
            String theStrings[] = { "x", "t", "c" };
            boolean rValue = true;;

            for ( int index = 0; index < theStrings.length; index ++ )
                    aStorage.add(theStrings[index]);

            aStorage.sort();

            for ( int index = 0; index < aStorage.size() - 1; index ++ )    {
                    String leftString  = aStorage.get(); aStorage.remove();
                    String rightString = aStorage.get(); aStorage.remove();
                    rValue &= ( leftString.compareTo(rightString) <= 0 );
            }
            return rValue;
        }
        public static void exampleOfHowToUseIt( Storage<String> aStorage)   {
            aStorage.add("a");
            aStorage.add(0, "0");
            aStorage.add(aStorage.size(), "1");
            aStorage.add(aStorage.size() + 1, "2");
            System.out.println("aStorage: " + aStorage );

        }
        public static void test(Storage<String> aStorage)   {
                    if ( ! testAdd() )
                            System.err.println("testAdd failed");
                    if ( ! testRemoveIndex() )
                            System.err.println("testRemoveIndex failed");
                    if ( ! testAddIndex() )
                            System.err.println("testAddIndex failed");
                    if ( ! testClear() )
                            System.err.println("testClear failed");
                    if ( ! testToArray() )
                            System.err.println("testToArray failed");
                    if ( ! testSort() )
                            System.err.println("testSort failed");
                    if ( ! testGet() )
                            System.err.println("testGet failed");
                    if ( ! testContains() )
                            System.err.println("testContains failed");
                    if ( ! testClassName() )
                            System.err.println("testClassName failed");
        }
        public static void main(String args[] )     {
            test(new Storage<String>());
            // exampleOfHowToUseIt(new Storage<String>());

        }
    }
```

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:

```

    public interface StorageI<E>  {

            public boolean add(E e);        // 2
            public E get();
            public void clear();            // 2 3
            public boolean contains(E e);
            public boolean isEmpty();
            public void sort();             // 3
            public int size();              // 2 3
            public String getClassName();

    }


```

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 differnce 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)

```
class Storage<E> implements StorageI<E> {

    private Storage<E> root = null;
    private Storage<E> next = null;
    private Object[]   getArray = null;
    private int        getIndex = -1;
    private int        nElements = 0;
    private  Object[]  sorted;
    E payLoad;

    public Storage(){
    }
    public int size()   {
        return nElements;
    }

    public boolean add(E e)     {
        if ( root == null )     {
                root            = new Storage<E>();
                root.payLoad    = e;
                root.next       = new Storage<E>();
        } else {
                Storage<E>      insertAfter = findPosition(e);
                Storage<E> next = new Storage<E>();
                next.payLoad    = e;
                if ( insertAfter == null )      {       // new root
                        next.next = root;
                        root = next;
                } else {
                        next.next       = insertAfter.next;
                        insertAfter.next= next;
                }
        }
        nElements++;

        return true;
    }
    private Storage<E> findPosition(E element)  {
        Storage<E>      runner = root;
        Storage<E>      followUp = null;
        boolean         keepGoing = true;
        String          insertString = element.toString();

        do      {
                String payLoadString  = runner.payLoad.toString();
                if ( payLoadString.compareTo(insertString) < 0 )        {
                        followUp = runner;
                        runner   = runner.next;
                        keepGoing = ( runner.payLoad != null );
                } else {
                        keepGoing = false;
                }
        } while ( keepGoing );
        return followUp;
    }
    public void    add(int index, E element)    {
        if ( index > nElements )
                return;
        if ( index == nElements )
                add(element);
        else if ( index == 0 )
                addFirst(element);
        else    {
                nElements++;
                Storage<E> helper = root;
                for ( int counter = 1; counter < index ; counter ++ )   {
                        helper = helper.next;
                }
                Storage<E> tmp = new Storage<E>();

                tmp.next = helper.next;
                tmp.payLoad = element;
                helper.next = tmp;

        }

    }
    public void    addFirst(E e)        {
        Storage<E> newRoot = new Storage<E>();
        newRoot.payLoad = e;
        newRoot.next = root;
        nElements++;
        root = newRoot;
    }
    public void    addLast(E e) {
        add(e);
    }
    public void    clear()      {
        root = null;
        nElements = 0;
    }
    public E       element()    {
        return root.payLoad;
    }
    public E       remove()     {
        if ( root == null )
```

```
                 return null;
                nElements--;
                E element = root.payLoad;
                root = root.next;
                return element;
        }
        public E       remove(int index)    {
                E element = null;
                if ( root == null ) {
                        return null;
                } else if ( index < nElements ) {
                        if ( index == 0 )       {
                                element = remove();
                        } else  {
                                nElements--;
                                Storage<E> helper = root;
                                for ( int counter = 0; counter < index - 1 ; counter ++ )
                                        helper = helper.next;
                                }
                                element = helper.next.payLoad;
                                helper.next = helper.next.next;
    // System.out.println("helper.payLoad: " + helper.payLoad);
    // System.out.println("element: " + element);
                        }
                }
                return element;
        }

        public String toString()    {
                Storage<E> index = root;
                String result    = "# of elements: " + size() + "  ";
                int counter      = 0;
                if ( root == null )
                        return "";
                do      {
                        if ( index == null )
                                result = result + "->null";
                        else
                                result = result + "->" + index.payLoad;
                        index = index.next;
                } while ( index != null );
                return result;
        }

        public String getClassName()        {
                return getClass().getCanonicalName();
        }
        Object[]        toArray() {
                if ( root == null )
                        return null;

                Object[] theArray = new Object[size()];
                Storage<E> index = root;
                int runner = 0;

                while ( index != null ) {
                        if ( index.payLoad != null )
                                theArray[runner++] = index.payLoad;
                        index = index.next;
                }
                return theArray;
        }
        private static  void printArray(Object[] theArray) {
                for ( int index = 0; index < theArray.length; index ++ )        {
                        System.out.println(index + ":   " + theArray[index]);
                }
        }
        private Object[] bubbleSort(Object[] theArray)      {
                for (int index = 0; index < theArray.length - 1; index++)     {
                        for (int walker=0; walker < theArray.length - 1; walker++)  {
                                String leftString  = theArray[walker].toString();
                                String rightString = theArray[walker + 1 ].toString();
                                if ( leftString.compareTo(rightString) > 0 )    {
                                        E tmp = (E)theArray[walker];
                                        theArray[walker] = theArray[walker + 1];
                                        theArray[walker+1] = tmp;
                                }
                        }
                }
                return theArray;
        }
        private void fillTheArray(Object[] theArray)        {
                for ( int index = 0; index < theArray.length; index ++ )        {
                        if ( theArray[index] != null )
                                add((E)theArray[index]);
                }
        }
        public void sort()  {
                if ( root == null )
                        return;
                Object[] theArray = toArray();
                clear();
                theArray = bubbleSort(theArray );
                bubbleSort(theArray );
                fillTheArray(theArray );

        }
        public boolean isEmpty() {
                return size() == 0;
        }
        public boolean contains(E e) {
                Storage<E> index = root;
                boolean rValue = true;
                boolean found  = true;;

                if ( root == null )
                        rValue =  false;
                else    {
                        do      {
                                found = e.equals(index.payLoad);
                                index = index.next;
                        } while ( ! found && ( index != null ) );
                }

                return rValue && found;
        }
        public E get() {
                if  ( root == null )
                        return null;
                if ( getIndex == -1 )   {
                        getArray = toArray();
                        getIndex = 0;
                }
                if ( getIndex < getArray.length )       {
                        return (E)getArray[getIndex++];
                } else {
                        getIndex = -1;
                        return null;
                }
        }
        public static boolean testAddIndex()        {
                Storage<String> aStorage = new Storage<String>();
                String theStrings[] = { "a", "b", "c" };
                boolean rValue = true;
                for ( int index = 0; index < theStrings.length; index ++ )
                        aStorage.add(theStrings[index]);
                int size;
                        String theItem = "0";
                        aStorage.add(0, theItem);
                rValue &= aStorage.remove() == theItem;
                        theItem = "1";
                        size = aStorage.size();
                        aStorage.add(size, theItem);
                rValue &= aStorage.remove(size) == theItem;
                        size = aStorage.size();
                        aStorage.add(size + 1, theItem);
                rValue &= aStorage.size() == size;
                        theItem = "2";
                        aStorage.add(1, theItem);
                rValue &= aStorage.remove(1) == theItem;

                return rValue;
        }
        public static boolean testRemoveIndex()     {
                Storage<String> aStorage = new Storage<String>();
                String theStrings[] = { "a", "b", "c" };
                boolean rValue = true;
                for ( int index = 0; index < theStrings.length; index ++ )
                        aStorage.add(theStrings[index]);
                rValue &= aStorage.remove(aStorage.size()) == null;
                rValue &= aStorage.remove(2).equals(theStrings[2]);
                rValue &= aStorage.remove(1).equals(theStrings[1]);
```

```
                rValue &= aStorage.remove(0).equals(theStrings[0]);
                rValue &= aStorage.remove(0) == null;

                aStorage.add("c");

                return rValue;
        }
        public static boolean testAdd()     {               // weak test
                Storage<String> aStorage = new Storage<String>();
                String theStrings[] = { "t", "x", "c" };
                boolean rValue = true;
                for ( int index = 0; index < theStrings.length; index ++ )      {
                        aStorage.add(theStrings[index]);
                }
                rValue &= aStorage.remove().equals(theStrings[2]);
                rValue &= aStorage.remove().equals(theStrings[0]);
                rValue &= aStorage.remove().equals(theStrings[1]);

                rValue &= aStorage.remove() == null;

                return rValue;
        }
        public static boolean testClassName()       {
                Storage<String> aStorage = new Storage<String>();

                return "Storage".equals(aStorage.getClassName());
        }
        public static boolean testClear()   {
                Storage<String> aStorage = new Storage<String>();
                boolean rValue = true;

                rValue &= aStorage.remove() == null;
                        aStorage.add("a");
                        aStorage.clear();
                rValue &= aStorage.size() == 0;

                return rValue;
        }
        public static boolean testContains()        {
                Storage<String> aStorage = new Storage<String>();
                String theStrings[] = { "a", "b", "c" };

                boolean rValue = true;
                rValue &= ! aStorage.contains(theStrings[0]);
                for ( int index = 0; index < theStrings.length; index ++ )
                        aStorage.add(theStrings[index]);
                rValue &= aStorage.contains(theStrings[0]);
                rValue &= ! aStorage.contains("d");

                return rValue;
        }
        public static boolean testToArray() {
                Storage<String> aStorage = new Storage<String>();
                String theStrings[] = { "a", "b", "c" };
                boolean rValue = true;;

                for ( int index = 0; index < theStrings.length; index ++ )
                        aStorage.add(theStrings[index]);
                Object[] anArrry = aStorage.toArray();

                rValue &= anArrry.length == aStorage.size();
                rValue &= anArrry.length == theStrings.length;
                return rValue;
        }
        public static boolean testGet()     {
                Storage<String> aStorage = new Storage<String>();
                String theStrings[] = { "a", "b", "c" };
                boolean rValue = true;;

                rValue &= aStorage.get() == null;

                for ( int index = 0; index < theStrings.length; index ++ )
                        aStorage.add(theStrings[index]);
                for ( int index = 0; index < theStrings.length; index ++ )
                        rValue &= aStorage.get().equals(theStrings[index]);
                rValue &= aStorage.get() == null;

                return rValue;
        }
        public static boolean testSort()    {
                Storage<String> aStorage = new Storage<String>();
                String theStrings[] = { "x", "t", "c" };
                boolean rValue = true;;

                for ( int index = 0; index < theStrings.length; index ++ )
                        aStorage.add(theStrings[index]);

                aStorage.sort();

                for ( int index = 0; index < aStorage.size() - 1; index ++ )    {
                        String leftString  = aStorage.get(); aStorage.remove();
                        String rightString = aStorage.get(); aStorage.remove();
                        rValue &= ( leftString.compareTo(rightString) <= 0 );
                }
                return rValue;
        }
        public static void exampleOfHowToUseIt( Storage<String> aStorage)   {
                aStorage.add("a");
                aStorage.add(0, "0");
                aStorage.add(aStorage.size(), "1");
                aStorage.add(aStorage.size() + 1, "2");
                System.out.println("aStorage: " + aStorage );

        }
        public static void test(Storage<String> aStorage)   {
                        if ( ! testAdd() )
                                System.err.println("testAdd failed");
        /*
```

```
                        if ( ! testRemoveIndex() )
                                System.err.println("testRemoveIndex failed");
                        if ( ! testAddIndex() )
                                System.err.println("testAddIndex failed");
                        if ( ! testClear() )
                                System.err.println("testClear failed");
                        if ( ! testToArray() )
                                System.err.println("testToArray failed");
                        if ( ! testSort() )
                                System.err.println("testSort failed");
                        if ( ! testGet() )
                                System.err.println("testGet failed");
                        if ( ! testContains() )
                                System.err.println("testContains failed");
                        if ( ! testClassName() )
                                System.err.println("testClassName failed");
        */
            }
            public static void main(String args[] )     {
                test(new Storage<String>());
                // exampleOfHowToUseIt(new Storage<String>());

            }
        }
```

Source Code: Src/26_sol/Storage_3.java

```
        class TestStorage  {
            public static boolean testAdd()     {
                Storage<String> aStorage = new Storage<String>();
                String theStrings[] = { "a", "b", "c" };
                boolean rValue = true;
                for ( int index = 0; index < theStrings.length; index ++ )
                        aStorage.add(theStrings[index]);

                for ( int index = 0; index < theStrings.length; index ++ ) {    // test if
                        rValue &= aStorage.remove(theStrings[index]).equals(theStrings[ind
                }

                aStorage.clear();
                rValue &= aStorage.add(theStrings[0]);
                rValue &= aStorage.size() == 1;

                return rValue;
            }
            public static boolean testClear()   {
                Storage<String> aStorage = new Storage<String>();
                String theStrings[] = { "a", "b", "c" };
                boolean rValue = true;

                aStorage.add("a");
                aStorage.clear();
                rValue &= aStorage.size() == 0;
                aStorage.add("a");
                rValue &= aStorage.size() == 1;

                return rValue;
            }
            public static boolean testContains()        {
                Storage<String> aStorage = new Storage<String>();
                String theStrings[] = { "a", "b", "c", "d" };
                boolean rValue = true;

                rValue &= ! aStorage.contains(theStrings[0]);
                aStorage.add(theStrings[0]);
                rValue &= aStorage.contains(theStrings[0]);
                rValue &= ! aStorage.contains(theStrings[1]);

                aStorage.add(theStrings[1]);
                rValue &= aStorage.contains(theStrings[1]);

                aStorage.add(theStrings[2]);
                aStorage.add(theStrings[3]);
                rValue &= aStorage.contains(theStrings[2]);
                rValue &= aStorage.contains(theStrings[3]);

                return rValue;
            }
            public static void addArray(Storage<String> aStorage, String[] theStrings)  {
                for ( int index = 0; index < theStrings.length; index ++ )
                        aStorage.add(theStrings[index]);
            }
            public static boolean testAddAll()  {
                Storage<String> aStorage = new Storage<String>();
                Storage<String> bStorage = new Storage<String>();
                Storage<String> cStorage = new Storage<String>();
                String theAstrings[] = { "a", "b", "c" };
                String theBstrings[] = { "A", "B", "C" };
                String theCstrings[] = { "AA", "BB", "CC" };
                addArray(bStorage, theBstrings);
                addArray(cStorage, theCstrings);
                boolean rValue = true;
                rValue &= aStorage.addAll(bStorage);
                rValue &= aStorage.size() == theBstrings.length;

                rValue &= aStorage.addAll(cStorage);
                rValue &= aStorage.size() == theBstrings.length + theCstrings.length;

                rValue &= ! aStorage.addAll(aStorage);
                return rValue;
            }
            public static boolean testRemove()  {
                String theStrings[] = { "a", "b", "c" };
                Storage<String> aStorage = new Storage<String>();
                boolean rValue = true;

                rValue &= null ==  aStorage.remove(theStrings[0] );
                        aStorage.add(theStrings[0] );
                rValue &=  aStorage.remove(theStrings[0] ).equals(theStrings[0]);
                        aStorage.add(theStrings[0]);
                        aStorage.add(theStrings[1]);
                        aStorage.remove(theStrings[0]);
                rValue &=  aStorage.remove(theStrings[1] ).equals(theStrings[1]);

                addArray(aStorage, theStrings);
                rValue &=  aStorage.remove(theStrings[1] ).equals(theStrings[1]);

                addArray(aStorage, theStrings);
                rValue &=  aStorage.remove(theStrings[2] ).equals(theStrings[2]);

                return rValue;
            }
            public static boolean testRemoveAll()       {
                boolean rValue = true;
                String theAstrings[] = { "a", "b", "c" };
                String theBstrings[] = { "a", "b", "c" };
                Storage<String> aStorage = new Storage<String>();
                Storage<String> bStorage = new Storage<String>();

                aStorage.add(theAstrings[0] );
                aStorage.add(theAstrings[1] );
                bStorage.add(theAstrings[0] );
                rValue &=  aStorage.removeAll(bStorage);

                return rValue;
            }
            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>());

            }
        }

```

Source Code: Src/26_sol/TestStorage_1.java