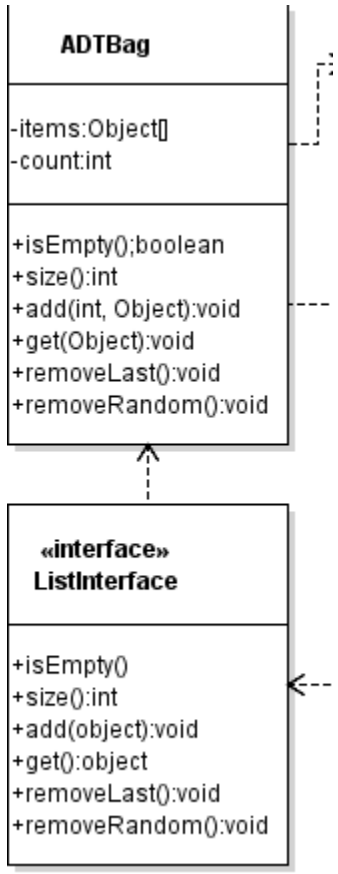
Grade: Great job! 9/10. Please see comments below.

If one of the following is true, you will NOT get credits.

☐The project is late.
☐ The algorithm or class design is missing.
☐The project has errors.
☐ There are no comments (Javadoc format required) at all in your code.
☐Wrong files are submitted.
☐ A project is a copy and modification of another student's project. (Both will receive 0.)

Software Development Project Rubric: Design is worth 20%; the rest is worth 80%.
Analysis
Note: There will be no credit if the software doesn't meet customer specification at all.
Does the software meet the exact customer specification?
Does the software read the exact input data and display the exact output data as they are shown in sample runs?
Does each class include all corresponding data and functionalities?

Design
Note: There will be no credit if the design is missing.
Is the design (a UML class diagram) an efficient solution?
Is the design created correctly?
A class implementing an interface must keep the same signatures for all methods. In the diagram, some signatures are changed.
In the diagram, ListInterface implements ADTBag.



Code (-1)
Note: There will be no credit if there are syntactic errors.
Are there errors in the software?
Are code conventions and name conventions followed?
Does the software use the minimum computer resource (computer memory and processing time)?
Is the software reusable?

ListInterface:

```
package project1;
/**
 * Interface for the ADTBag for the driver.
 *
 * @author Julian Itwar
 * @ver 1.0
 */
public interface ADTBagInterface
{
	/**
	 * Determines if the bag is empty.
	 * @return A boolean value that tells if the bag is empty.
	 */
	public boolean isEmpty();

	/**
	 * Tells the amount of items in the bag.
	 * @return An integer of how many items are in the bag.
	 */
	public int size();

	/**
	 * Determines if the bad is full.
	 * @return A boolean value that tells if the bag is full.
	 */
	public boolean isFull();

	/**
	 * Returns an items.
	 * @param item A reference to the added item.
	 * @throws ListException if list is full.
	 * @throws ArrayIndexOutOfBoundsException  if items is greater than 0, or is full.
	 */
```

An item can be inserted if only if this bag is not full. IndexOutOfBoundsException can be prevented in this case.

```
	public void insertItem(Object item) throws ListException, ~~ArrayIndexOutOfBoundsException~~;

	/**
	 * Removes all items from the bag.
```

```
     */
    public void makeEmpty();
```

When a bag is full, insertion will fail; when a bag is empty, deletion will fail. These are abnormal situations due to the room of this bag. ListException is made for both types of operations.
An item can be removed if only if this bag is not empty. IndexOutOfBoundsException can be prevented in this case. Don't use name ArrayIndexOutOfBounds since the data structure can be changed. Use `ListIndexOutOfBoundsException` instead. There is already a class called ArrayIndexOutOfBounds in programming.

```
    /**
     *
     * Removes an item from the bag.
     * @throws ArrayIndexOutOfBoundsException
     */
    public void removeLast() throws ListException ~~ArrayIndexOutOfBoundsException~~;

    /**
     *
     * Removes a random item from the bag.
     * @throws ArrayIndexOutOfBoundsException
     */
    public void removeRandom() throws ListException ~~ArrayIndexOutOfBoundsException~~;


    /**
     *
     * @param index
     * @return An item at index.
     * @throws ArrayIndexOutOfBoundsException If there is no item to return.
     * @throws ListException If the bag is empty.
     */
    public Object get(int index) throws ~~ArrayIndexOutOfBoundsException~~ ListIndexOutOfBoundsException;
```

ADTBag:

```
    /**
     * Removes an item at random.
     */
    public void removeRandom() throws ArrayIndexOutOfBoundsException
    {
```

```
                        if(this.isEmpty())
                        {
                              throw new ArrayIndexOutOfBoundsException("This list is empty");
                        }
                        else if(this.count == 1)
                        {
                              this.count--;
                        }
                        else
                        {
                              Random rand = new Random();
                              int j = rand.nextInt(this.count);
After each item's address is copied into its previous position, the last item index still contains the address of the last
item. The last item needs to be remembered so that it can be set to null.

                              while(j < this.count)
                              {
                                    items[j - 1] = items[j];
                                    j++;

                              }
                              this.items[this.count – 1] = null;

                              this.count--;
                        }
                  }
```

Debug

Are there bugs in the software?

Documentation

Note: There will be no credit if comments are not included.

Are there enough comments included in the software?

Class comments must be included before a class header.

Method comments must be included before a method header.

More comments must be included inside each method.

All comments must be written in Javadoc format.