Assignment 3

This assignment is due Thursday, January 31 at 5:00pm. You must work with a partner on this assignment.

The analysis document for this assignment takes significantly more time than previous assignments. Plan to finish the code portion early, so you have enough time for the analysis.

The Java Collection interface

This assignment will give you practice creating a generic data structure and implementing an interface using the Java API specification. It also gives you an introduction to binary search. Before you proceed, familiarize yourself with the Java Collection interface.

Part 1:
ArrayCollection

Your task is to implement the generically parameterized Collection interface using an array as the base storage for items.

Start by downloading ArrayCollection.java. Create a new package for it called `assignment3`. ArrayCollection.java contains all the method headers for the `Collection` interface, as well as a private class `ArrayCollectionIterator`, with all the method headers for implementing the `Iterator` interface. ArrayCollection should not allow for duplicate items to be added.

Part 2:
Binary Search

Once you've finished your ArrayCollection, add SearchUtil.java to your `assignment3` package. Fill in the missing implementation of `binarySearch`. This is a generic static method using a comparator to efficiently search for an item in a sorted generic `ArrayList`, to be used in combination with your `ArrayCollection`'s `toSortedList` method. Since binarySearch is a static method, invoke it by first specifying the class name, as in `SearchUtil.binarySearch(...)`

ArrayCollection Specification

Your `ArrayCollection` must implement all methods of the `Collection` interface as specified in the Java API, unless other specific instructions are listed here. You do not need to override Object's `equals` or `hashCode` methods.

The storage for items is already created for you, called `data`, and starts with an initial size of 10. As items are added to the collection, this capacity may be insufficient. If an add is attempted when the capacity is full, you must increase the capacity of `data` to accommodate additional items.

`grow` -

Doubles the capacity of the the data array by creating a new array and copying the data over, then assigning data to the new array. Do not worry about the array growing too large. Assume the machine has infinite memory, and let Java handle the crash if it actually gets too large. To create a new `E[]` array, use code similar to the line that allocates `data` in the constructor.

`add` -

returns `false` if the item being added is already in the collection, otherwise, adds the item to the first empty spot in the `data` array and returns `true`. This function must call `grow` if the data array is full.

`addAll` -

use either a for-each loop or an Iterator (all Collections have an `iterator()` method) to iterate through the collection being added. Add only the items that don't already exist in your `ArrayCollection`. If any items were added, return `true`, otherwise return `false`.

`clear` -

removes all items from the collection. The result of this operation is that the size of the collection is 0. Optionally, you can set each item that was in the collection to `null`.

`contains` -

returns `true` if the specified object is in the collection, as determined by the object's `.equals` method, returns `false` otherwise.

`containsAll` -

returns `true` if this `ArrayCollection` contains all items in the argument collection, returns `false` otherwise.

`isEmpty` -

returns `true` if this `ArrayCollection` contains no items, `false` otherwise

`iterator` -

returns a new `ArrayCollectionIterator.` See the `ArrayCollectionIterator` class section below.

`remove` -

if the specified object is not in the collection, returns `false`, otherwise, removes the item from the collection and returns `true`. If an item is removed, you must coalesce the data array by moving all items after the deleted item back one space to fill the empty spot.

`removeAll` -

if any item in the input collection is also in this collection, removes it from this collection. Returns `true` if any items were removed, `false` otherwise.

`retainAll` -

The result of this function call is that this collection contains only the items that were in both this collection and the input collection. In other words, removes any item from this collection that is not also in the specified collection. Returns `true` if any items were removed, `false` otherwise.

I recommend completing your `ArrayCollectionIterator` first, and using `next` and `remove` to iterate through your collection while potentially removing items. This can get tricky if you use a for-loop, since the loop bounds will change as items are removed.

`size` -

returns the number of items held in this collection (not the capacity)

`toArray()` -

returns a new `Object` array containing all items in this collection. The returned array must be new, and it must be the exact size of the number of items in the collection (don't just return `data`).

`toArray(T[] arg0)` -

Don't worry about implementing this function. Our test code won't use it.

`toSortedList` -

Returns an `ArrayList` sorted (using selection sort) according to the order specified by the input comparator. The `sort` method from Assignment 2 implements a selection sort. You can copy it, but it will require minor modifications.

`ArrayCollectionIterator` class

This class implements a very basic iterator for `ArrayCollection`. It should iterate through the data array in sequential order, starting at index 0. In other words, items are returned by the iterator's `next` method in the same order that they were added to the collection.
See the Java Iterator API documentation

Note: Iterator behavior is undefined if the underlying collection changes during iteration, except through calls to the iterator's `remove` method, so don't worry about anyone changing the collection behind the scenes in your iterator methods.

`hasNext` -
returns `true` if there are any more items in the collection to iterate through, `false` otherwise. In other words, returns `false` if a call to `next` would throw an exception.

`next` -
must throw a new `NoSuchElementException` if there are no more items to

iterate through, otherwise, returns the next item in the collection

`remove` -

this function is a bit tricky. It removes the *last* item that was returned by `next`. It can therefore only be called once per call to `next`. If `next` has not been called since the last call to `remove`, or if it hasn't been called at all, throws a new `IllegalStateException`. Your iterator will need to keep track of some state indicating whether or not it is currently legal to call `remove`. If it does remove an item, think about what must change in the iterator so that the next call to `next` returns the correct item.
Since this method shares the same name as one of the `ArrayCollection` methods, you can explicitly call `ArrayCollection`'s `remove` method with the following: `ArrayCollection.this.remove`(...)

Commenting

The provided ArrayCollection file has minimal comments. You must fill in comments for each method and any unclear code. This is specified in the comment at the top of the ArrayCollection class.

Testing

This is the first assignment in which we don't provide any tests for you. You must create your own class `TestArrayCollection.java`, which contains a `main` method and tests your `ArrayCollection`, as well as your `SearchUtil.binarySearch` method. As always, your tests should be thorough and convincingly sufficient to ensure that your data structure and algorithms are correct. We will not test inserting `null` in to your `ArrayCollection`.

Analysis

The analysis document for this assignment will take a significant portion of the total time. Make sure you start on it early. Hand in your code and analysis document here by Thursday, January 31 at 5:00pm.

Handing in

Use the button at the bottom of this page to hand in your source code. Hand in a .zip file with your ArrayCollection.java, SearchUtil.java, and TestArrayCollection.java (only one partner hand in the code). Hand in your analysis document as a separate file. Both of you must hand in your own analysis document.

Due date:	Thursday, 31 January 2013, 5:05 PM

Submission feedback

Poonam Ekhelikar
Saturday, 9 February 2013, 11:48 PM

Grade: 61.00 / 100.00

```
NoSuchElementException
NoSuchElementException
TEST FAILED: non-empty iterate
   next() returned wrong element
TEST FAILED: non-empty iterate except.
   exception not thrown on iterator.remove() before iterator.next()
TEST FAILED: non-empty iterate except.
   next() returned wrong element
TEST FAILED: non-empty toSortedList
   Contents incorrect after method
TEST FAILED: Add duplicate string 0
   Returned true
TEST FAILED: Add duplicate string 4
   Returned true
TEST FAILED: Add duplicate string 10
   Returned true
TEST FAILED: containsAll of empty
   Returned false
TEST FAILED: empty containsAll of non-empty
   Threw java.lang.ArrayIndexOutOfBoundsException: 10
TEST FAILED: removeAll of empty
   Returned true
TEST FAILED: empty removeAll of non-empty
   Returned true
TEST FAILED: empty retainAll of non-empty
   Threw java.lang.NullPointerException
```

```
TEST FAILED: addAll from empty
   Returned true
TEST FAILED: retainAll on identical collection
   Threw java.lang.NullPointerException
NoSuchElementException
NoSuchElementException
NoSuchElementException
IllegalStateExeption
TEST FAILED: iterate except. after removeAll
   exception not thrown on iterator.remove() before iterator.next()
NoSuchElementException
NoSuchElementException
TEST FAILED: iterate except. after removeAll
   exception not thrown on iterator.next() when there is no next
NoSuchElementException
NoSuchElementException
NoSuchElementException
IllegalStateExeption
TEST FAILED: iterate except. after clear
   exception not thrown on iterator.remove() before iterator.next()
NoSuchElementException
NoSuchElementException
TEST FAILED: iterate except. after clear
   exception not thrown on iterator.next() when there is no next
TEST FAILED: non-empty toSortedList
   Contents incorrect after method
TEST FAILED: toSortedList after addAll
   Contents incorrect after method
TEST FAILED: retainAll of empty
   Returned false
TEST FAILED: empty addAll from empty
   Returned true
TEST FAILED: empty containsAll of empty
   Returned false
TEST FAILED: empty removeAll of empty
   Returned true
TEST FAILED: Empty remove element 0
   Threw java.lang.NegativeArraySizeException
TEST FAILED: Empty remove element 1
   Threw java.lang.NegativeArraySizeException
```

```
TEST FAILED: Empty remove element 2
   Threw java.lang.NegativeArraySizeException
TEST FAILED: Empty remove element 3
   Threw java.lang.NegativeArraySizeException
TEST FAILED: Empty remove element 4
   Threw java.lang.NegativeArraySizeException
TEST FAILED: Empty remove element 5
   Threw java.lang.NegativeArraySizeException
TEST FAILED: Empty remove element 6
   Threw java.lang.NegativeArraySizeException
TEST FAILED: Empty remove element 7
   Threw java.lang.NegativeArraySizeException
TEST FAILED: Empty remove element 8
   Threw java.lang.NegativeArraySizeException
TEST FAILED: Empty remove element 9
   Threw java.lang.NegativeArraySizeException
TEST FAILED: Empty remove element 10
   Threw java.lang.NegativeArraySizeException
TEST FAILED: iterate after remove
   next() returned wrong element
TEST FAILED: iterate except. after remove
   exception not thrown on iterator.remove() before iterator.next()
TEST FAILED: iterate except. after remove
   next() returned wrong element
TEST FAILED: toSortedList after remove
   Contents incorrect after method
TEST FAILED: retainAll superset
   Threw java.lang.NullPointerException
TEST FAILED: addAll subset
   Returned true
TEST FAILED: removeAll non-intersecting 1
   Returned true
TEST FAILED: removeAll non-intersecting 2
   Returned true
TEST FAILED: retainAll non-intersecting
   Threw java.lang.NullPointerException
TEST FAILED: retainAll non-intersecting
   Threw java.lang.NullPointerException
TEST FAILED: iterator remove first element
   next() returned wrong element
```

```
TEST FAILED: iterator remove except. last element
   exception not thrown on iterator.remove() before iterator.next()
TEST FAILED: iterator remove except. last element
   next() returned wrong element
TEST FAILED: iterator remove except. middle element
   exception not thrown on iterator.remove() before iterator.next()
TEST FAILED: iterator remove except. middle element
   next() returned wrong element
TEST FAILED: iterator remove several elements
   next() returned wrong element
TEST FAILED: toSortedList average case 1
   Contents incorrect after method
TEST FAILED: toSortedList average case 2
   Contents incorrect after method
TEST FAILED: toSortedList large number of elements
   Contents incorrect after method

              SUMMARY
==============================================
add:       17/20 tests
      3.4/4 points
addAll:      10/13 tests
      3.08/4 points
clear:      3/3 tests
      2/2 points
contains:   28/28 tests
      4/4 points
containsAll:   2/5 tests
      1.6/4 points
isEmpty:   10/10 tests
      4/4 points
remove:      6/17 tests
      2.47/7 points
removeAll:   3/8 tests
      1.5/4 points
retainAll:   0/6 tests
      0/8 points
size:      19/19 tests
      2/2 points
toArray:   4/4 tests
```

4/4 points
iterator: 3/5 tests
2.4/4 points
iter. remove: 0/4 tests
0/4 points
iter. except.: 0/14 tests
0/6 points
toSortedList: 1/8 tests
0.88/7 points
binarySearch: 26/26 tests
7/7 points

Subtotal: 132/190 tests
38.32/75 points
Student Tests: 3/5
Analysis Doc: 19/20
Final Score: 61/100

TA Comments: Q9. Best case O(1)

AnalysisCs2420 #3.pdf

Submission

AnalysisCs2420 #3.pdf
Assignment03Final.zip

No further submissions are allowed.