WIA1002/WIB1002 Data Structure

Lab: ADTs and Bags

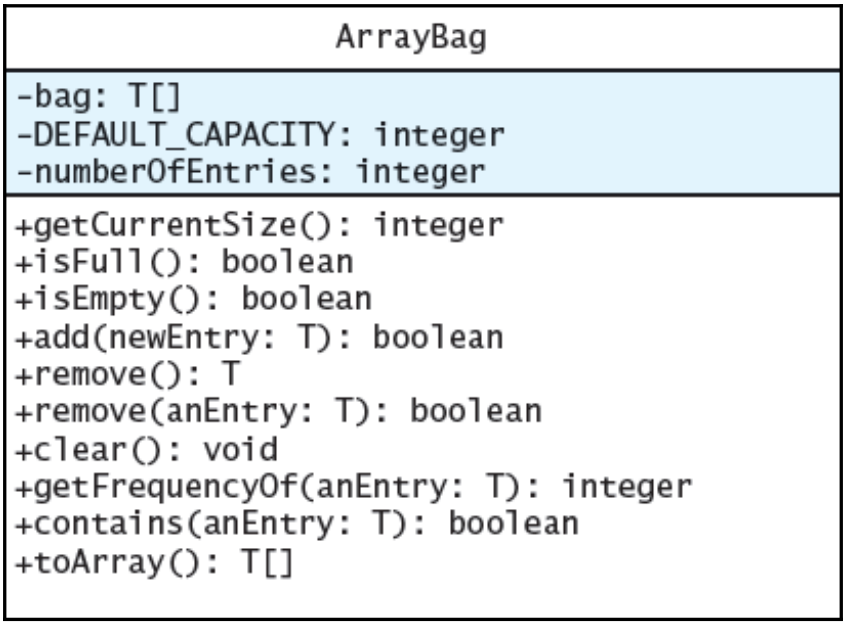
Question 1

Given the interface `BagInterface` below:

```
/**
An interface that describes the operations of a bag of objects.
*/
public interface BagInterface<T>
{
      /** Gets the current number of entries in this bag.
          @return the integer number of entries currently in the bag */
      public int getCurrentSize();

      /** Sees whether this bag is full.
          @return true if the bag is full, or false if not */
      public boolean isFull();

      /** Sees whether this bag is empty.
          @return true if the bag is empty, or false if not */
      public boolean isEmpty();

      /** Adds a new entry to this bag.
          @param newEntry  the object to be added as a new entry
          @return true if the addition is successful, or false if not */
      public boolean add(T newEntry);

      /** Removes one unspecified entry from this bag, if possible.
          @return either the removed entry, if the removal was successful,
                  or null */
      public T remove();

      /** Removes one occurrence of a given entry from this bag.
          @param anEntry  the entry to be removed
          @return true if the removal was successful, or false if not */
      public boolean remove(T anEntry);

      /** Removes all entries from this bag. */
      public void clear();

      /** Counts the number of times a given entry appears in this bag.
          @param anEntry  the entry to be counted
          @return the number of times anEntry appears in the bag */
      public int getFrequencyOf(T anEntry);

      /** Tests whether this bag contains a given entry.
          @param anEntry  the entry to locate
          @return true if this bag contains anEntry, or false otherwise */
      public boolean contains(T anEntry);

      /** Retrieves all entries that are in this bag.
```

```
         @return a newly allocated array of all the entries in the bag */
     public T[] toArray();
} // end BagInterface
```

Write an `ArrayBag` class that implement the `BagInterface`, as shown in the UML diagram below. Set the DEFAULT_CAPACITY to 25.



Question 2

The union of two collections consists of their contents combined into a new collection. **Add a method** `union` to the interface `BagInterface` for the ADT bag that returns as a new bag the union of the bag receiving the call to the method and the bag that is the method's one argument. Include sufficient comments to fully specify the method.

Note that the union of two bags might contain duplicate items. For example, if object x occurs five times in one bag and twice in another, the union of these bags contains x seven times. Specifically, suppose that bag1 and bag2 are Bag objects, where Bag implements BagInterface; bag1 contains the String objects a, b, and c; and bag2 contains the String objects b, b, d, and e. After the statement

```
BagInterface<String> everything = bag1.union(bag2);
```

executes, the bag everything contains the strings a, b, b, b, c, d, and e. Note that union does not affect the contents of bag1 and bag2.

Implement the `union` method in `ArrayBag` class.

Question 3

The intersection of two collections is a new collection of the entries that occur in both collections. That is, it contains the overlapping entries. **Add a method** `intersection` to the interface `BagInterface` for the ADT bag that returns as a new bag the intersection of the bag receiving the call to the method and the bag that is the method's one argument. Include sufficient comments to fully specify the method.

Note that the intersection of two bags might contain duplicate items. For example, if object x occurs five times in one bag and twice in another, the intersection of these bags contains x twice. Specifically, suppose that bag1 and bag2 are Bag objects, where Bag implements BagInterface; bag1 contains the String objects a, b, and c; and bag2 contains the String objects b, b, d, and e. After the statement

```
BagInterface<String> commonItems = bag1.intersection(bag2);
```

executes, the bag commonItems contains only the string b. If b had occurred in bag1 twice, commonItems would have contained two occurrences of b, since bag2 also contains two occurrences of b. Note that intersection does not affect the contents of bag1 and bag2.

Implement the `intersection` method in `ArrayBag` class.

Question 4

The difference of two collections is a new collection of the entries that would be left in one collection after removing those that also occur in the second. **Add a method** `difference` to the interface `BagInterface` for the ADT bag that returns as a new bag the difference of the bag receiving the call to the method and the bag that is the method's one argument. Include sufficient comments to fully specify the method.

Note that the difference of two bags might contain duplicate items. For example, if object x occurs five times in one bag and twice in another, the difference of these bags contains x three times. Specifically, suppose that bag1 and bag2 are Bag objects, where Bag implements BagInterface; bag1 contains the String objects a, b, and c; and bag2 contains the String objects b, b, d, and e. After the statement

```
BagInterface leftOver1 = bag1.difference(bag2);
```

executes, the bag leftOver1 contains the strings a and c. After the statement

```
BagInterface leftOver2 = bag2.difference(bag1);
```

executes, the bag leftOver2 contains the strings b, d, and e. Note that difference does not affect the contents of bag1 and bag2.

Implement the `difference` method in `ArrayBag` class.

Sample Codes for Question 1 to Question 4

BagInterface:

```
/**
   An interface that describes the operations of a bag of objects.
*/
public interface BagInterface<T>
{
      /** Gets the current number of entries in this bag.
             @return  The integer number of entries currently in the bag. */
      public int getCurrentSize();

      /** Sees whether this bag is empty.
             @return  True if the bag is empty, or false if not. */
      public boolean isEmpty();

      /** Adds a new entry to this bag.
          @param newEntry  The object to be added as a new entry.
          @return  True if the addition is successful, or false if not. */
      public boolean add(T newEntry);

      /** Removes one unspecified entry from this bag, if possible.
       @return  Either the removed entry, if the removal.
                was successful, or null. */
      public T remove();

      /** Removes one occurrence of a given entry from this bag.
       @param anEntry  The entry to be removed.
       @return  True if the removal was successful, or false if not. */
      public boolean remove(T anEntry);

      /** Removes all entries from this bag. */
      public void clear();

      /** Counts the number of times a given entry appears in this bag.
             @param anEntry  The entry to be counted.
             @return  The number of times anEntry appears in the bag. */
      public int getFrequencyOf(T anEntry);

      /** Tests whether this bag contains a given entry.
             @param anEntry  The entry to locate.
             @return  True if the bag contains anEntry, or false if not. */
      public boolean contains(T anEntry);

      /** Retrieves all entries that are in this bag.
             @return  A newly allocated array of all the entries in the bag.
                Note: If the bag is empty, the returned array is empty. */
      public T[] toArray();


   /** Creates a new bag that combines the contents of this bag
       and anotherBag. (Question 2)
       @param anotherBag  The bag that is to be added.
```

```
       @return  A combined bag. */
   // public BagInterface<T> union(BagInterface<T> anotherBag);

   /** Creates a new bag that contains those objects that occur
       in both this bag and anotherBag. (Question 3)
       @param anotherBag  The bag that is to be compared.
       @return  A combined bag. */
   // public BagInterface<T> intersection(BagInterface<T> anotherBag);

   /** Creates a new bag of objects that would be left in this bag
       after removing those that also occur in anotherBag. (Question 4)
       @param anotherBag  The bag that is to be removed.
       @return  A combined bag. */
   // public BagInterface<T> difference(BagInterface<T> anotherBag);
} // end BagInterface
```

ArrayBag:

```
/**
   A class ArrayBag that implments BagInterface
*/
public class ArrayBag<T> implements BagInterface<T> {

      private T[] bag;
      private static final int DEFAULT_CAPACITY = 25;
      private int numberOfEntries;

      @SuppressWarnings("unchecked")
      public ArrayBag() {
            this(DEFAULT_CAPACITY);
      }

      @SuppressWarnings("unchecked")
      public ArrayBag(int capacity) {
            bag = (T[]) new Object[capacity];
      }

      /** Gets the current number of entries in this bag.
       @return the integer number of entries currently in the bag */
      public int getCurrentSize() {
            return numberOfEntries;
      } // end getCurrentSize

      /** Sees whether this bag is full.
       @return true if the bag is full, or false if not */
      public boolean isFull() {
            return (numberOfEntries == bag.length);
      } // end isFull

      /** Sees whether this bag is empty.
       @return true if the bag is empty, or false if not */
      public boolean isEmpty() {
            return (numberOfEntries == 0);
      } // end isEmpty
```

```
	/** Adds a new entry to this bag.
	@param newEntry  the object to be added as a new entry
	@return true if the addition is successful, or false if not */
	public boolean add(T newEntry) {
		if (isFull())
			return false;
		else {
			bag[numberOfEntries++] = newEntry;
			return true;
		}
	} // end add

	/** Removes one unspecified entry from this bag, if possible.
	@return either the removed entry, if the removal
	 was successful, or null */
	public T remove() {
		T result = removeEntry(numberOfEntries - 1);
		return result;
	} // end remove


	/** Removes one occurrence of a given entry from this bag.
	@param anEntry  the entry to be removed
	@return true if the removal was successful, or false if not */
	public boolean remove(T anEntry) {
		int index = getIndexOf(anEntry);
		T result = removeEntry(index);
		return anEntry.equals(result);
	} // end remove


	/** Removes and returns the entry at a given index within the array.
	@param givenIndex the index of the entry to be removed
	@return the entry if the removal was successful, or null if not */
	private T removeEntry(int givenIndex)
	{
		T result = null;

		if (!isEmpty() && (givenIndex >= 0)) {
			result = bag[givenIndex];             // entry to remove
			numberOfEntries--;

			// replace entry to remove with last entry
			bag[givenIndex] = bag[numberOfEntries];
			bag[numberOfEntries] = null;  // remove reference to last
entry
		} // end if

		return result;
	} // end removeEntry


	/** Locates a given entry within the array bag.
	@param anEntry  the entry to be found
```

```
	@return the index of the entry, if located, or -1 otherwise*/
	// Returns the index of the entry, if located, or -1 otherwise.
	private int getIndexOf(T anEntry)
	{
		int where = -1;
		boolean found = false;

		for (int index = 0; !found && (index < numberOfEntries); index++)
{
			if (anEntry.equals(bag[index])) {
				found = true;
				where = index;
			} // end if
		} // end for

		// Assertion: If where > -1, anEntry is in the array bag, and it
		// equals bag[where]; otherwise, anEntry is not in the array

		return where;
	} // end getIndexOf


	/** Removes all entries from this bag. */
	public void clear() {
		while(!isEmpty()) remove();
	} // end clear

	/** Counts the number of times a given entry appears in this bag.
	 @param anEntry  the entry to be counted
	 @return the number of times anEntry appears in the bag */
	public int getFrequencyOf(T anEntry) {
		int counter = 0;

		for (int index = 0; index < numberOfEntries; index++) {
			if (anEntry.equals(bag[index])) {
				counter++;
			} // end if
		} // end for

		return counter;
	} // end getFrequencyOf


	/** Tests whether this bag contains a given entry.
	 @param anEntry  the entry to locate
	 @return true if this bag contains anEntry, or false otherwise */
	public boolean contains(T anEntry) {
		boolean found = false;

		for (int index = 0; !found && (index < numberOfEntries); index++)
{
			if (anEntry.equals(bag[index])) {
				found = true;
			} // end if
		} // end for
```

```
            return found;
      } // end contains

      /** Retrieves all entries that are in this bag.
       @return a newly allocated array of all the entries in the bag */
      public T[] toArray() {
            @SuppressWarnings("unchecked")
            T[] result = (T[])new Object[numberOfEntries]; // unchecked cast

            for (int index = 0; index < numberOfEntries; index++)
            {
                  result[index] = bag[index];
            } // end for

            return result;
      } // end toArray


      /** Creates a new bag that combines the contents of this bag and
anotherBag.
      @param anotherBag  the bag that is to be added (Question 2)
      @return a combined bag */
      public BagInterface<T> union(BagInterface<T> anotherBag) {
            int sizeOfUnionBag = anotherBag.getCurrentSize() +
getCurrentSize();
            BagInterface<T> unionBag = new ArrayBag<T>(sizeOfUnionBag);
            ArrayBag<T> otherBag = (ArrayBag<T>)anotherBag;
            int index;
            // add entries from this bag to the new bag
            for (index = 0; index < numberOfEntries; index++)
                  unionBag.add(bag[index]);
            // add entries from the second bag to the new bag
            for (index = 0; index < otherBag.getCurrentSize(); index++)
                  unionBag.add(otherBag.bag[index]);
            return unionBag;
      } // end union


      /** Creates a new bag that contains those objects that occur in both
this bag
      and anotherBag. (Question 3)
      @param anotherBag  the bag that is to be compared
      @return a combined bag */
      public BagInterface<T> intersection(BagInterface<T> anotherBag) {
            BagInterface<T> intersectionBag = new ArrayBag<T>();
            ArrayBag<T> otherBag = (ArrayBag<T>)anotherBag;
            BagInterface<T> copyOfAnotherBag = new ArrayBag<T>();
            int index;
            // copy the second bag
            for (index = 0; index < otherBag.numberOfEntries; index++)
            {
                  copyOfAnotherBag.add(otherBag.bag[index]);
            } // end for
```

```
            // add to intersectionBag each item in this bag that matches an
item in
            // anotherBag; once matched, remove it from the second bag
            for (index = 0; index < getCurrentSize(); index++)
            {
                  if (copyOfAnotherBag.contains(bag[index]))
                  {
                        intersectionBag.add(bag[index]);
                        copyOfAnotherBag.remove(bag[index]);
                  } // end if
            } // end for
            return intersectionBag;
      } // end intersection


      /** Creates a new bag of objects that would be left in this bag
      after removing those that also occur in anotherBag. (Question 4)
      @param anotherBag  the bag that is to be removed
      @return a combined bag */
      public BagInterface<T> difference(BagInterface<T> anotherBag) {
            BagInterface<T> differenceBag = new ArrayBag<T>();
            ArrayBag<T> otherBag = (ArrayBag<T>)anotherBag;
            int index;
            // copy this bag
            for (index = 0; index < numberOfEntries; index++)
            {
                  differenceBag.add(bag[index]);
            } // end for
            // remove the ones that are in anotherBag
            for (index = 0; index < otherBag.getCurrentSize(); index++)
            {
                  if (differenceBag.contains(otherBag.bag[index]))
                  {
                        differenceBag.remove(otherBag.bag[index]);
                  } // end if
            } // end for
            return differenceBag;
      } // end difference
}
```

Question 5

Write a test program `ArrayBagDemo.java` to test the core methods `add`, `getCurrentSize()` and `toArray` of the class ArrayBag, and the three methods that you implemented in Question 2 to 4 (i.e. `union`, `intersection` and `difference`). Initially the main method creates an empty bag by using the default constructor. Since the capacity of the bag is 25, the array should not get full if you add fewer than 25 entries to it. Thus, add should return true after each of these conditions. The test program's descriptive output will indicates that the tested methods are correct.

In your program:

a) Create a private static method named `testAdd` to test the method `Add` of `ArrayBag`:

```
private static void testAdd(BagInterface<String> aBag, String[] content)
```

b) Create a private static method named `displayBag` to test the methods `getCurrentSize()` and `toArray of ArrayBag` while displaying the all the entries of the bag.

```
private static void displayBag(BagInterface<String> aBag)
```

c) In the main method:

- Create two (2) empty `ArrayBag` objects of the type array for a list of strings, named `bag1` and `bag2`.

- Then filling an initially empty bag, `bag1` ad `bag2` with a list of strings using the method `testAdd`:

```
String[] contentsOfBag1 = {"A", "A", "B", "A", "C", "A"};
String[] contentsOfBag2 = {"A", "B", "A", "C", "B", "C", "D", "another
string"};
```

- Display the contents of `bag1` and `bag2` using method `displayBag`.

- Test the method `union` using a new bag named `bag3` that combines the contents of `bag1` and `bag2`. Display the contents of `bag3` using method `displayBag`.

- Test the method `intersection` using a new bag named `bag4` that contains those objects that occur in both `bag1` and `bag2`. Display the contents of `bag4` using method `displayBag`.

- Test the method `difference` using a new bag named `bag5` that create a new bag of objects that would be left in `bag1` after removing those that also occur in `bag2`. Display the contents of `bag5` using method `displayBag`.

An example of the output for the test program is as follows:

```
bag1:
Adding A A B A C A
The bag contains 6 string(s), as follows:
A A B A C A

bag2:
Adding A B A C B C D another string
The bag contains 8 string(s), as follows:
A B A C B C D another string

bag3, test the method union of bag1 and bag2:
The bag contains 14 string(s), as follows:
A A B A C A A B A C B C D another string

bag4, test the method intersection of bag1 and bag2:
The bag contains 4 string(s), as follows:
A A B C

bag5, test the method difference of bag1 and bag2:
The bag contains 2 string(s), as follows:
A A
```

Sample Code for ArrayBagDemo.java (Question 5):

```
public class ArrayBagDemo {

	public static void main(String[] args) {
		/* Create two (2) empty ArrayBag objects of the type array for
		 * a list of strings, named bag1, bag2 */
		BagInterface<String> bag1 = new ArrayBag<>();
		BagInterface<String> bag2 = new ArrayBag<>();

		/* Then filling an initially empty bag, bag1 ad bag2 with
		 * a list of strings using the method testAdd
		 * Display the contents of bag1 and bag2 using method displayBag */

		String[] contentsOfBag1 = {"A", "A", "B", "A", "C", "A"};
		String[] contentsOfBag2 = {"A", "B", "A", "C", "B", "C", "D", "another
string"};

		System.out.println("bag1:");
		testAdd(bag1,contentsOfBag1);
		displayBag(bag1);

		System.out.println("\nbag2:");
		testAdd(bag2,contentsOfBag2);
		displayBag(bag2);

		/*Test the method union using a new bag named bag3
		 * that combines the contents of bag1 and bag2.
		 * Display the contents of bag3 using method displayBag.
		 */
		System.out.println("\nbag3, test the method union +"
				+ "of bag1 and bag2:");
		BagInterface<String> bag3 = bag1.union(bag2);
		displayBag(bag3);
```

```
            System.out.println("\nbag4, test the method intersection +"
                        + "of bag1 and bag2:");
            /*Test the method intersection using a new bag named bag4
             * that contains those objects that occur in both bag1 and bag2.
             * Display the contents of bag4 using method displayBag.
             */
             BagInterface<String> bag4 = bag1.intersection(bag2);
             displayBag(bag4);

             System.out.println("\nbag5, test the method difference "
                        + "+ of bag1 and bag2:");
            /*Test the method difference using a new bag named bag5
             * that create a new bag of objects that would be left in bag1
             * after removing those that also occur in bag2.
             * Display the contents of bag5 using method displayBag.
             */
             BagInterface<String> bag5 = bag1.difference(bag2);
             displayBag(bag5);

      }

      // Tests the method add.
      private static void testAdd(BagInterface<String> aBag, String[] content)
      {
            System.out.print("Adding ");
            for (int index = 0; index < content.length; index++)
            {
                  aBag.add(content[index]);
        System.out.print(content[index] + " ");
            } // end for
       System.out.println();

      } // end testAdd


      // Tests the method toArray while displaying the bag.
      private static void displayBag(BagInterface<String> aBag)
      {
            System.out.println("The bag contains " + aBag.getCurrentSize() +
                               " string(s), as follows:");
            Object[] bagArray = aBag.toArray();
            for (int index = 0; index < bagArray.length; index++)
            {
                  System.out.print(bagArray[index] + " ");
            } // end for

            System.out.println();
      } // end displayBag

}// end ArrayBagDemo
```