CSC Programming Assignment #1 (PRA1)

We affirm that we have carried out our academic endeavors with full academic honesty.

- James Heffernan
- Manav Bilakhia
- Saeed AlSuwaidi
- Eric Zhao

Java files

BagInterface.java

```
package assignment;

/*
 * Honor code
 * James Heffernan, Manav Bilakhia, Saeed AlSuwaidi, Eric Zhao
 */

/**
 * An interface that describes the operations of a bag of objects.
 *
 * @author Frank M. Carrano
 * @author Timothy M. Henry
 * @version 5.0
 */
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, if possible.
     *
     * @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 find.
     * @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();
} // end BagInterface
```

DoublyLinkedBag.java

```
package assignment;

/*
 * Honor code
 * James Heffernan, Manav Bilakhia, Saeed AlSuwaidi, Eric Zhao
 */

import java.util.Objects;
import java.util.StringJoiner;

/**
 * DoublyLinkedBag class: A class of bags whose entries are stored in a chain of
doubly linked nodes.
 * The bag is never full.
 *
```

```
 * @author Frank M. Carrano
 * @version 5.0
 */
public class DoublyLinkedBag<T> implements BagInterface<T> {
    private DoublyLinkedNode firstNode; // Reference to first node
    private int numberOfEntries;

    /**
     * Parameterless constructor that initializes this Bag.
     */
    public DoublyLinkedBag() {
        firstNode = null;
        numberOfEntries = 0;
    }

    /**
     * add: 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) {
        DoublyLinkedNode newNode = new DoublyLinkedNode(newEntry);
        newNode.next = firstNode;
        if (firstNode != null) {
            firstNode.setPrevious(newNode);
        }
        firstNode = newNode;
        numberOfEntries++;
        return true;
    }

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

        int index = 0;
        DoublyLinkedNode currentNode = firstNode;
        while ((index < numberOfEntries) && (currentNode != null)) {
            result[index] = currentNode.data;
            index++;
            currentNode = currentNode.next;
        }

        return result;
    }
```

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

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

/**
 * getFrequencyOf: 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 this bag.
 */
public int getFrequencyOf(T anEntry) {
    int frequency = 0;
    int loopCounter = 0;
    DoublyLinkedNode currentNode = firstNode;

    while ((loopCounter < numberOfEntries) && (currentNode != null)) {
        if (anEntry.equals(currentNode.data))
            frequency++;
        loopCounter++;
        currentNode = currentNode.next;
    }
    return frequency;
}

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

    while (!found && (currentNode != null)) {
        if (anEntry.equals(currentNode.data))
            found = true;
        else
```

```
            currentNode = currentNode.next;
        }
        return found;
    }

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

    /**
     * remove: 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 = null;
        if (firstNode != null) {
            result = firstNode.data;
            firstNode = firstNode.next;
            if (firstNode != null) {
                firstNode.previous = null;
            }
            numberOfEntries--;
        }
        return result;
    }

    private DoublyLinkedNode getReferenceTo(T anEntry) {
        boolean found = false;
        DoublyLinkedNode currentNode = firstNode;

        while (!found && (currentNode != null)) {
            if (anEntry.equals(currentNode.data))
                found = true;
            else
                currentNode = currentNode.next;
        }
        return currentNode;
    }

    /**
     * remove: Removes one occurrence of a given entry from this bag, if possible.
     *
     * @param anEntry The entry to be removed.
     * @return True if the removal was successful, or false otherwise.
     */
    public boolean remove(T anEntry) {
```

```
        DoublyLinkedNode removing = getReferenceTo(anEntry);
        if (removing != null) {
            DoublyLinkedNode next = removing.next;
            DoublyLinkedNode prev = removing.previous;
            if (next != null) {
                next.previous = prev;
            }
            if (prev != null) {
                prev.next = next;
            }
            numberOfEntries--;
            return true;
        }
        return false;
    }

    /**
     * union: Creates a new bag that combines the contents of this bag and
anotherBag.
     *
     * @param anotherBag The bag that is to be added.
     * @return A combined bag.
     */
    public DoublyLinkedBag<T> union(DoublyLinkedBag anotherBag) {
        int loopCounter = 0;
        int loopCounter2 = 0;
        DoublyLinkedNode otherCurrentNode = anotherBag.firstNode;
        DoublyLinkedNode thisCurrentNode = firstNode;
        DoublyLinkedBag<T> Torba = new DoublyLinkedBag();

        while ((loopCounter < anotherBag.numberOfEntries) && (otherCurrentNode !=
null)) {
            Torba.add(otherCurrentNode.data);
            loopCounter++;
            otherCurrentNode = otherCurrentNode.next;
        }
        while ((loopCounter2 < numberOfEntries) && (thisCurrentNode != null)) {
            Torba.add(thisCurrentNode.data);
            loopCounter2++;
            thisCurrentNode = thisCurrentNode.next;
        }
        return Torba;
    }

    /**
     * intersection: Creates a new bag that contains those objects that occur in
both this bag and
     * anotherBag.
     *
     * @param anotherBag The bag that is to be compared.
     * @return The intersection of the two bags.
     */
```

```
    public DoublyLinkedBag<T> intersection(DoublyLinkedBag anotherBag) {
        DoublyLinkedBag<T> returnBag = new DoublyLinkedBag();
        int loopCounter = 0;
        DoublyLinkedNode thisCurrentNode = firstNode;
        while ((loopCounter < numberOfEntries) && (thisCurrentNode != null)) {
            int loopCounter2 = 0;
            DoublyLinkedNode otherCurrentNode = anotherBag.firstNode;

            while ((loopCounter2 < anotherBag.numberOfEntries) && (otherCurrentNode
!= null) && thisCurrentNode != null) {
                if (thisCurrentNode.data.equals(otherCurrentNode.data))
                    returnBag.add(thisCurrentNode.data);
                loopCounter2++;
                otherCurrentNode = otherCurrentNode.next;

                loopCounter++;
                thisCurrentNode = thisCurrentNode.next;
            }
        }
        return returnBag;
    }


    /**
     * difference: Creates a new bag of objects that would be left in this bag after
removing
     * those that also occur in anotherBag.
     *
     * @param anotherBag The bag that is to be removed.
     * @return The difference of the two bags.
     */
    public DoublyLinkedBag<T> difference(DoublyLinkedBag anotherBag) {
        DoublyLinkedBag<T> bag = new DoublyLinkedBag();

        for (Object item : anotherBag.toArray()) {
            boolean found = false;
            for (Object item2 : this.toArray()) {
                if (item.equals(item2)) {
                    found = true;
                }
            }

            if (!found) bag.add((T) item);
        }

        for (Object item : this.toArray()) {
            boolean found = false;
            for (Object item2 : anotherBag.toArray()) {
                if (item.equals(item2)) {
                    found = true;
                }
            }
```

```
            if (!found) bag.add((T) item);
        }

        return bag;
    }



    /**
     * toString: Convert this bag to a String for displaying.
     * each item will be comma separated and a space after comma enclosed in [ and ]
     * if we have a b c d in the bag a is the most recent one and will be converted
as
     * [a, b, c, d]. StringJoiner is a good option to use.
     */
    public String toString() {
        StringJoiner joinItems = new StringJoiner(", ", "[", "]");
        // passing comma(, ) as delimiter and brackets [] as prefix and suffix
respectively
        DoublyLinkedNode thisNode = firstNode;
        for (int i = 0; i < numberOfEntries; i++) {
            joinItems.add(thisNode.data.toString());
            thisNode = thisNode.next;
        }
        return joinItems.toString();
    }

    // private inner class DoublyLinkedNode:
    // A class of nodes for a chain of doubly linked nodes.
    private class DoublyLinkedNode {

        private T data; // Entry in bag
        private DoublyLinkedNode next; // Link to next node
        private DoublyLinkedNode previous; // Link to previous node


        // private constructor of class DoublyLinkedNode with a data parameter
        private DoublyLinkedNode(T dataPortion) {
            this(dataPortion, null, null);
        }

        // private constructor of class DoublyLinkedNode with a data, nextNode and
previousNode parameters
        private DoublyLinkedNode(T dataPortion, DoublyLinkedNode nextNode,
DoublyLinkedNode prevNode) {
            data = dataPortion;
            next = nextNode;
            previous = prevNode;
        }

        // get and set methods for DoublyLinkedNode class
```

```
        public void setData(T data) {
            this.data = data;
        }

        public T getData() {
            return data;
        }

        public void setNext(DoublyLinkedNode next) {
            this.next = next;
        }

        public DoublyLinkedNode getNext() {
            return next;
        }

        public void setPrevious(DoublyLinkedNode previous) {
            this.previous = previous;
        }

        public DoublyLinkedNode getPrevious() {
            return previous;
        }
    }

    public static void main(String[] args) {
        DoublyLinkedBag<String> bag = new DoublyLinkedBag();
        bag.add("a");
        bag.add("b");
        bag.add("c");
        bag.add("d");

        System.out.println(bag);
        System.out.println("Bag Size: " + bag.getCurrentSize());
        System.out.println("Contains a: " + bag.contains("a"));
        System.out.println("Frequency of c: " + bag.getFrequencyOf("c"));

        System.out.println("Remove random element" + bag.remove());
        System.out.println("Bag Size after remove: " + bag.getCurrentSize());
        System.out.println("Remove a: " + bag.remove("a"));
        System.out.println("Bag Size after remove: " + bag.getCurrentSize());
        System.out.println("Bag after remove: " + bag);

        DoublyLinkedBag<String> bag2 = new DoublyLinkedBag();
        bag2.add("a");
        bag2.add("b");
        bag2.add("c");

        System.out.println("Bag2: " + bag2);
        System.out.println("Union of bag and bag2: " + bag.union(bag2));
        System.out.println("Intersection of bag and bag2: " +
bag.intersection(bag2));
```

```
        System.out.println("Difference of bag and bag2: " + bag.difference(bag2));
    }
} // end DoublyLinkedBag
```

SpellCheckerDoubly.java

```
package assignment;

/*
 * Honor code
 * James Heffernan, Manav Bilakhia, Saeed AlSuwaidi, Eric Zhao
 */

import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;

/**
 * SpellCheckerDoubly class: A spelling checker class.
 *
 * @author Frank M. Carrano
 * @author zorhan modified
 * @version 5.0
 */
public class SpellCheckerDoubly {
    BagInterface<String> correctWords; // Storage for the correct words

    /**
     * Parameterless constructor that initializes this Bag.
     * correctWords should be a new DoublyLinkedBag of String
     */
    public SpellCheckerDoubly() {
        this.correctWords = new DoublyLinkedBag();
    }

    /**
     * setDictionary: Initializes the dictionary of correctly spelled words from a
file.
     *
     * @param fileName A File name as a String that represents a text file of
correctly
     *                 spelled words, one per line.
     * @return True if the dictionary is initialized, if not returns false.
     */
    public boolean setDictionary(String fileName) {
        File file = new File(fileName);
        Scanner sc;
        try {
            sc = new Scanner(file);
        } catch (FileNotFoundException e) {
            return false;
```

```
        }

        while (sc.hasNextLine()) {
            correctWords.add(sc.nextLine());
        }
        return true;
    }

    /**
     * setDictionary: Initializes the dictionary of correctly spelled words from a
list.
     *
     * @param correctWordsList An ArrayList list of String that contains correctly
spelled words.
     */
    public void setDictionary(ArrayList<String> correctWordsList) {
        for (String correctWords : correctWordsList) {
            this.correctWords.add(correctWords);
        }
    }

    /**
     * setDocument: Initializes the document to be checked from a file.
     *
     * @param fileName A File name as a String that represents a text file of the
document
     *                 whose words will be checked, one per line.
     * @return DoublyLinkedBag(BagInterface) of document's words.
     */
    public BagInterface<String> setDocument(String fileName) throws
FileNotFoundException {
        File file = new File(fileName);
        Scanner sc = new Scanner(file);
        DoublyLinkedBag<String> bag = new DoublyLinkedBag();
        while (sc.hasNextLine()) {
            bag.add(sc.nextLine());
        }

        return bag;
    }

    /**
     * setDocument: Initialises the document to be checked from a list
     *
     * @param wordList An ArrayList of String that contains words to be checked.
     * @return DoublyLinkedBag(BagInterface) of document's words.
     */
    public BagInterface<String> setDocument(ArrayList<String> wordList) {
        DoublyLinkedBag<String> bag = new DoublyLinkedBag();
        for (String word : wordList) {
            bag.add(word);
        }
```

```
        return bag;
    }

    private boolean checkSpelling(Object[] dictList, String aWord) {
        for (Object correctWord : dictList) {
            if (aWord.equals(correctWord))
                return true;
        }
        return false;
    }

    /**
     * checkSpelling: Checks the spelling of a given single word as String.
     *
     * @param aWord A string whose spelling is to be checked.
     * @return True if the given word is spelled correctly, otherwise returns false.
     */
    public boolean checkSpelling(String aWord) {
        return checkSpelling(this.correctWords.toArray(), aWord);
    }

    private void checkSpellingBag(Object[] dictList, Object[] wordList,
BagInterface<String> correct, BagInterface<String> incorrect) {
        for (Object o : wordList) {
            String word = (String) o;
            if (checkSpelling(dictList, word)) {
                correct.add(word);
            } else {
                incorrect.add(word);
            }
        }
    }

    /**
     * checkBagSpelling: Checks the spelling of a given bag of words
     *
     * @param wordBag   A bag of words whose spelling is to be checked.
     * @param correct   A bag of the words in wordBag whose spellings are correct
     * @param incorrect A bag of the words in wordBag whose spellings are incorrect
     */
    public void checkSpellingBag(BagInterface<String> wordBag, BagInterface<String>
correct, BagInterface<String> incorrect) {
        checkSpellingBag(this.correctWords.toArray(), wordBag.toArray(), correct,
incorrect);
    }

    /**
     * checkSpellingFromLists: Checks the spelling of a given bag of words
     * from the given list of words list and dictionary list.
     *
     * @param dictList  A list of dictionary words
```

```
     * @param wordList  A list of words to be checked
     * @param correct   A bag of the words in wordBag whose spellings are correct
     * @param incorrect A bag of the words in wordBag whose spellings are incorrect
     */
    public void checkSpellingFromLists(ArrayList<String> dictList, ArrayList<String>
wordList, BagInterface<String> correct, BagInterface<String> incorrect) {
        Collections.reverse(dictList);
        Collections.reverse(wordList);
        checkSpellingBag(dictList.toArray(), wordList.toArray(), correct,
incorrect);
    }

    /**
     * checkSpellingFromFile: Checks the spelling of a given bag of words by
     * creating the words list and dictionary from the files.
     *
     * @param dictFile  A file name of dictionary words
     * @param wordFile  A file name of words to be checked
     * @param correct   A bag of the words in wordBag whose spelling are correct
     * @param incorrect A bag of the words in wordBag whose spelling are incorrect
     */
    public void checkSpellingFromFile(String dictFile, String wordFile,
BagInterface<String> correct, BagInterface<String> incorrect) {
        try {
            ArrayList<String> dictList = new ArrayList();
            File fileD = new File(dictFile);
            Scanner dsc = new Scanner(fileD);
            while (dsc.hasNextLine()) {
                dictList.add(dsc.nextLine());
            }

            ArrayList<String> wordList = new ArrayList();
            File fileW = new File(wordFile);
            Scanner wsc = new Scanner(fileW);
            while (wsc.hasNextLine()) {
                wordList.add(wsc.nextLine());
            }

            checkSpellingFromLists(dictList, wordList, correct, incorrect);
        } catch (FileNotFoundException e) {

        }
    }

    public static void main(String[] args) {
        ArrayList<String> wordList = new ArrayList<>();
        wordList.add("cow");
        wordList.add("dog");
        wordList.add("vcat");
        wordList.add("piga");
        wordList.add("elk");
        ;
```

```
        ArrayList<String> dict = new ArrayList<>();
        dict.add("cow");
        dict.add("dog");
        dict.add("cat");
        dict.add("pig");
        dict.add("elk");


        SpellCheckerDoubly spD = new SpellCheckerDoubly();
        spD.setDictionary(dict);
        System.out.println("Check spelling of misspelled word: " +
spD.checkSpelling("emuashf"));
        System.out.println("Check spelling of correctly spelled word: " +
spD.checkSpelling("cat"));

        System.out.println("-----------------------------------------------------");

        BagInterface<String> correct = new DoublyLinkedBag<>();
        BagInterface<String> incorrect = new DoublyLinkedBag<>();
        spD.checkSpellingFromLists(dict, wordList, correct, incorrect);
        System.out.println("All words to be checked " + wordList);
        System.out.println("All dictionary words " + dict);
        System.out.println("Correctly spelled" + correct);
        System.out.println("Incorrectly spelled" + incorrect);

        System.out.println(" -----------------------------------------------------");

        SpellCheckerDoubly spD2 = new SpellCheckerDoubly();
        BagInterface<String> correct2 = new DoublyLinkedBag<>();
        BagInterface<String> incorrect2 = new DoublyLinkedBag<>();
        spD2.checkSpellingFromFile("src/assignment/dictionary.txt",
"src/assignment/document.txt", correct2,
                incorrect2);
        System.out.println("Correctly spelled" + correct2);
        System.out.println("Incorrectly spelled" + incorrect2);
    }
}
```

JUnit tests

DoublyLinkedBagTest.java

```
package assignment;

/*
 * Honor code
 * James Heffernan, Manav Bilakhia, Saeed AlSuwaidi, Eric Zhao
 */

import static org.junit.jupiter.api.Assertions.*;
```

```
class DoublyLinkedBagTest {

    // Test the add method
    @org.junit.jupiter.api.Test
    void add() {
        DoublyLinkedBag<String> bag = new DoublyLinkedBag<>();
        bag.add("a");
        bag.add("b");
        bag.add("c");

        assertEquals(3, bag.getCurrentSize());
    }

    // Test the toArray method
    @org.junit.jupiter.api.Test
    void testToArray() {
        DoublyLinkedBag<String> bag = new DoublyLinkedBag<>();
        bag.add("a");
        bag.add("b");
        bag.add("c");

        Object[] bagArray = bag.toArray();
        assertEquals(3, bagArray.length);
    }

    // Test the isEmpty method
    @org.junit.jupiter.api.Test
    void isEmpty() {
        DoublyLinkedBag<String> bag = new DoublyLinkedBag<>();
        assertTrue(bag.isEmpty());
    }

    // Test the getCurrentSize method
    @org.junit.jupiter.api.Test
    void getCurrentSize() {
        DoublyLinkedBag<String> bag = new DoublyLinkedBag<>();
        bag.add("a");
        bag.add("b");
        bag.add("c");

        assertEquals(3, bag.getCurrentSize());
    }

    // Test the getFrequencyOf method
    @org.junit.jupiter.api.Test
    void getFrequencyOf() {
        DoublyLinkedBag<String> bag = new DoublyLinkedBag<>();
        bag.add("a");
        bag.add("b");
        bag.add("c");
```

```
    assertEquals(1, bag.getFrequencyOf("a"));
    assertEquals(1, bag.getFrequencyOf("b"));
    assertEquals(1, bag.getFrequencyOf("c"));
}

// Test the contains method
@org.junit.jupiter.api.Test
void contains() {
    DoublyLinkedBag<String> bag = new DoublyLinkedBag<>();
    bag.add("a");
    bag.add("b");
    bag.add("c");

    assertTrue(bag.contains("a"));
    assertTrue(bag.contains("b"));
    assertTrue(bag.contains("c"));
}

// Test the clear method
@org.junit.jupiter.api.Test
void clear() {
    DoublyLinkedBag<String> bag = new DoublyLinkedBag<>();
    bag.add("a");
    bag.add("b");
    bag.add("c");

    bag.clear();
    assertEquals(0, bag.getCurrentSize());
}

@org.junit.jupiter.api.Test
void remove() {
    DoublyLinkedBag<String> bag = new DoublyLinkedBag<>();
    bag.add("a");
    bag.add("b");
    bag.add("c");

    bag.remove("a");
    assertEquals(2, bag.getCurrentSize());
}


@org.junit.jupiter.api.Test
void testRemove() {
    DoublyLinkedBag<String> bag = new DoublyLinkedBag<>();
    bag.add("a");

    bag.remove();
    assertTrue(bag.isEmpty());
}

@org.junit.jupiter.api.Test
```

```
void union() {
    DoublyLinkedBag<String> bag = new DoublyLinkedBag<>();
    bag.add("a");
    bag.add("b");
    bag.add("c");

    DoublyLinkedBag<String> bag2 = new DoublyLinkedBag<>();
    bag2.add("a");
    bag2.add("b");
    bag2.add("d");

    DoublyLinkedBag<String> union = bag.union(bag2);
    assertEquals(2, union.getFrequencyOf("a"));
    assertEquals(2, union.getFrequencyOf("b"));
    assertEquals(1, union.getFrequencyOf("c"));
    assertEquals(1, union.getFrequencyOf("d"));
}

@org.junit.jupiter.api.Test
void intersection() {
    DoublyLinkedBag<String> bag = new DoublyLinkedBag<>();
    bag.add("a");
    bag.add("b");
    bag.add("c");

    DoublyLinkedBag<String> bag2 = new DoublyLinkedBag<>();
    bag2.add("a");
    bag2.add("b");
    bag2.add("d");

    DoublyLinkedBag<String> intersection = bag.intersection(bag2);
    assertEquals(1, intersection.getFrequencyOf("a"));
    assertEquals(1, intersection.getFrequencyOf("b"));
    assertEquals(2, intersection.getCurrentSize());
}

@org.junit.jupiter.api.Test
void difference() {
    DoublyLinkedBag<String> bag = new DoublyLinkedBag<>();
    bag.add("a");
    bag.add("b");
    bag.add("c");

    DoublyLinkedBag<String> bag2 = new DoublyLinkedBag<>();
    bag2.add("a");
    bag2.add("b");
    bag2.add("d");


    DoublyLinkedBag<String> difference = bag.difference(bag2);
    assertEquals(1, difference.getFrequencyOf("c"));
    assertEquals(1, difference.getFrequencyOf("d"));
```

```
        assertEquals(2, difference.getCurrentSize());
    }
}
```

SpellCheckerDoublyTest.java

```
package assignment;

/*
 * Honor code
 * James Heffernan, Manav Bilakhia, Saeed AlSuwaidi, Eric Zhao
 */

import org.junit.jupiter.api.Test;

import java.io.FileNotFoundException;
import java.util.ArrayList;

import static org.junit.jupiter.api.Assertions.*;

class SpellCheckerDoublyTest {

    @Test
    void checkSpelling() throws FileNotFoundException {
        SpellCheckerDoubly spellCheckerDoubly = new SpellCheckerDoubly();
        // Create list of words
        ArrayList<String> dictionary = new ArrayList<>();
        dictionary.add("hello");
        dictionary.add("world");

        spellCheckerDoubly.setDictionary(dictionary);

        assertTrue(spellCheckerDoubly.checkSpelling( "hello"));
        assertFalse(spellCheckerDoubly.checkSpelling( "word"));
    }

    @Test
    void checkSpellingBag() throws FileNotFoundException {
        SpellCheckerDoubly spellCheckerDoubly = new SpellCheckerDoubly();
        // Create list of words
        DoublyLinkedBag<String> words = new DoublyLinkedBag<>();
        words.add("hello");
        words.add("word");

        // Dictionary list
        ArrayList<String> dictionary = new ArrayList<>();
        dictionary.add("hello");
        dictionary.add("world");

        // Check spelling
        spellCheckerDoubly.setDictionary(dictionary);
```

```
        DoublyLinkedBag<String> correct = new DoublyLinkedBag<>();
        DoublyLinkedBag<String> incorrect = new DoublyLinkedBag<>();

        spellCheckerDoubly.checkSpellingBag(words, correct, incorrect);

        assertEquals(1, correct.getFrequencyOf("hello"));
        assertEquals(correct.getCurrentSize(), 1);

        assertEquals(1, incorrect.getFrequencyOf("word"));
        assertEquals(incorrect.getCurrentSize(), 1);


    }

    @Test
    void checkSpellingFromLists() {
        SpellCheckerDoubly spellCheckerDoubly = new SpellCheckerDoubly();
        // Create list of words
        ArrayList<String> words = new ArrayList<>();
        words.add("hello");
        words.add("word");

        // Dictionary list
        ArrayList<String> dictionary = new ArrayList<>();
        dictionary.add("hello");
        dictionary.add("world");

        // Check spelling
        spellCheckerDoubly.setDictionary(dictionary);

        DoublyLinkedBag<String> correct = new DoublyLinkedBag<>();
        DoublyLinkedBag<String> incorrect = new DoublyLinkedBag<>();

        spellCheckerDoubly.checkSpellingFromLists(dictionary, words, correct,
incorrect);

        assertEquals(1, correct.getFrequencyOf("hello"));
        assertEquals(correct.getCurrentSize(), 1);

        assertEquals(1, incorrect.getFrequencyOf("word"));
        assertEquals(incorrect.getCurrentSize(), 1);
    }

    @Test
    void checkSpellingFromFile() {
        SpellCheckerDoubly spellCheckerDoubly = new SpellCheckerDoubly();

        DoublyLinkedBag<String> correct = new DoublyLinkedBag<>();
        DoublyLinkedBag<String> incorrect = new DoublyLinkedBag<>();

        spellCheckerDoubly.checkSpellingFromFile("src/assignment/dictionary.txt",
```

```
"src/assignment/document.txt", correct, incorrect);

        assertEquals(1, correct.getFrequencyOf("hello"));
        assertEquals(correct.getCurrentSize(), 1);

        assertEquals(1, incorrect.getFrequencyOf("word"));
        assertEquals(incorrect.getCurrentSize(), 1);
    }
}
```

Sample output 1

Describe your test 1

This test tests all of doubly linked bag

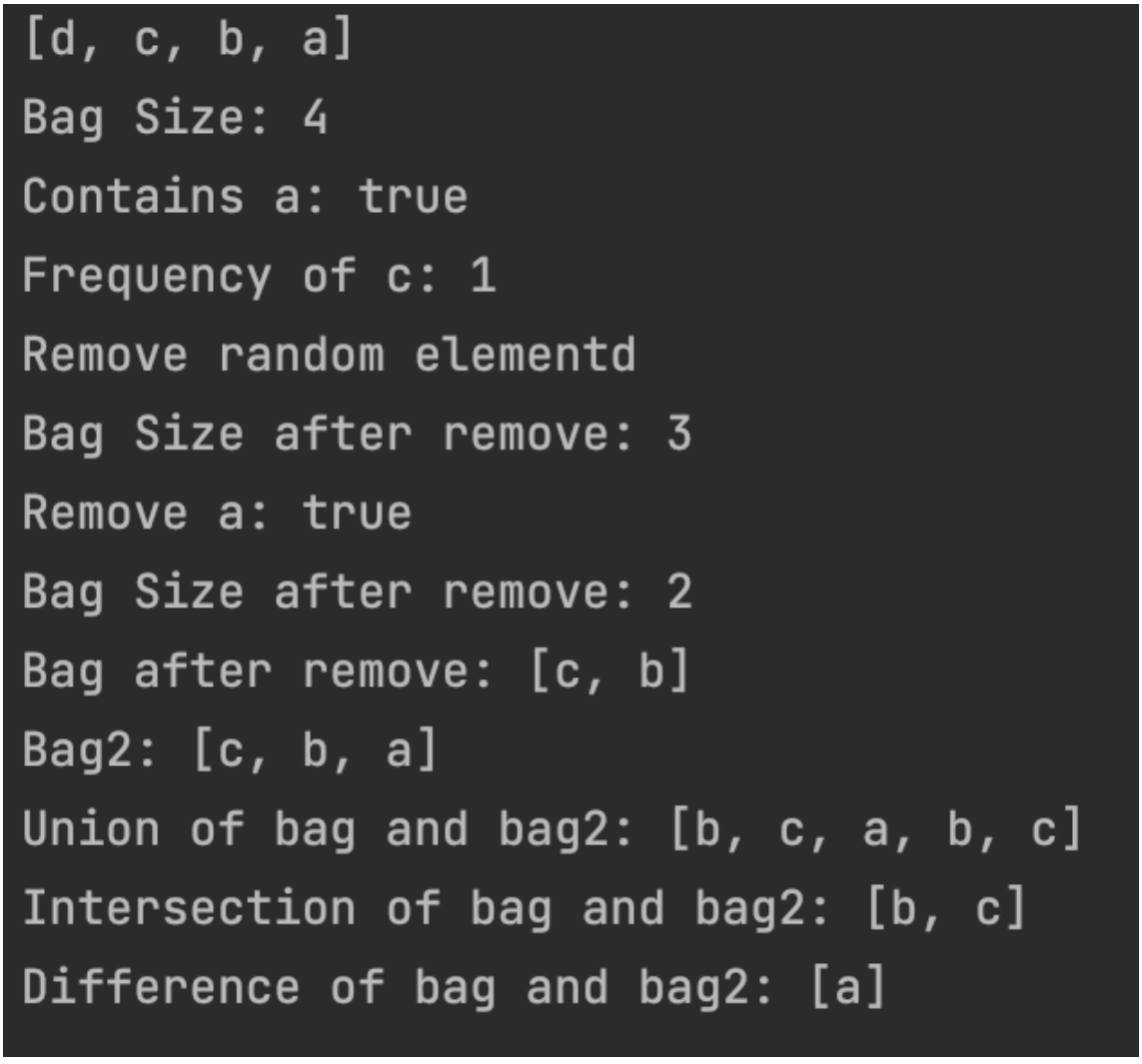
```
DoublyLinkedBag<String> bag = new DoublyLinkedBag();
bag.add("a");
bag.add("b");
bag.add("c");
bag.add("d");

System.out.println(bag);
System.out.println("Bag Size: " + bag.getCurrentSize());
System.out.println("Contains a: " + bag.contains("a"));
System.out.println("Frequency of c: " + bag.getFrequencyOf("c"));

System.out.println("Remove random element" + bag.remove());
System.out.println("Bag Size after remove: " + bag.getCurrentSize());
System.out.println("Remove a: " + bag.remove("a"));
System.out.println("Bag Size after remove: " + bag.getCurrentSize());
System.out.println("Bag after remove: " + bag);

DoublyLinkedBag<String> bag2 = new DoublyLinkedBag();
bag2.add("a");
bag2.add("b");
bag2.add("c");

System.out.println("Bag2: " + bag2);
System.out.println("Union of bag and bag2: " + bag.union(bag2));
System.out.println("Intersection of bag and bag2: " + bag.intersection(bag2));
System.out.println("Difference of bag and bag2: " + bag.difference(bag2));
```

Text output 1

```
[d, c, b, a]
Bag Size: 4
Contains a: true
Frequency of c: 1
Remove random elementd
```

```
Bag Size after remove: 3
Remove a: true
Bag Size after remove: 2
Bag after remove: [c, b]
Bag2: [c, b, a]
Union of bag and bag2: [b, c, a, b, c]
Intersection of bag and bag2: [b, c]
Difference of bag and bag2: [a]
```

Screenshot 1



Sample output 2

Describe your test 2

This test tests the entirety of spell checker doubly

```
ArrayList<String> wordList = new ArrayList<>();
wordList.add("cow");
wordList.add("dog");
wordList.add("vcat");
wordList.add("piga");
wordList.add("elk");
;
ArrayList<String> dict = new ArrayList<>();
dict.add("cow");
dict.add("dog");
dict.add("cat");
dict.add("pig");
dict.add("elk");

SpellCheckerDoubly spD = new SpellCheckerDoubly();
spD.setDictionary(dict);
System.out.println("Check spelling of misspelled word: " +
spD.checkSpelling("emuashf"));
System.out.println("Check spelling of correctly spelled word: " +
spD.checkSpelling("cat"));

System.out.println("-----------------------------------------------------");

BagInterface<String> correct = new DoublyLinkedBag<>();
BagInterface<String> incorrect = new DoublyLinkedBag<>();
spD.checkSpellingFromLists(dict, wordList, correct, incorrect);
System.out.println("All words to be checked " + wordList);
System.out.println("All dictionary words " + dict);
System.out.println("Correctly spelled" + correct);
System.out.println("Incorrectly spelled" + incorrect);

System.out.println(" -----------------------------------------------------");

SpellCheckerDoubly spD2 = new SpellCheckerDoubly();
BagInterface<String> correct2 = new DoublyLinkedBag<>();
BagInterface<String> incorrect2 = new DoublyLinkedBag<>();
spD2.checkSpellingFromFile("src/assignment/dictionary.txt",
"src/assignment/document.txt", correct2,
incorrect2);
System.out.println("Correctly spelled" + correct2);
System.out.println("Incorrectly spelled" + incorrect2);
```

Text output 2

```
Check spelling of misspelled word: false
Check spelling of correctly spelled word: true
-----------------------------------------------------
All words to be checked [elk, piga, vcat, dog, cow]
All dictionary words [elk, pig, cat, dog, cow]
Correctly spelled[cow, dog, elk]
```

```
Incorrectly spelled[vcat, piga]
 ---------------------------------------------------
Correctly spelled[hello]
Incorrectly spelled[word]
```

Screenshot 2