CSC 151 Assignment #3

1. Honor Code

A. For individual assignments: Jane Doe and John Doe will be replaced by your full name(s)
I affirm that I have carried out my academic endeavors with full academic honesty.
[Signed, Manav Bilakhia]

B. Resources/References
Geeksforgeeks
TextBook

2. Java files and outputs

A. Java files
Class (interface): BagInterfave.java

```
package assignment;
/*
 * I affirm that I have carried out the attached academic endeavors with full academic
honesty.
 * Manav Bilakhia (MB)
 */
/**
 * 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
```

Class: Coin.java

```
/*
 * I affirm that I have carried out the attached academic endeavors with full academic
honesty.
 * Manav Bilakhia (MB)
 */
package assignment;

import java.util.Objects;

/**
 *
 * @author Manav Bilakhia
 *
 */
public class Coin {
    /*
     * Instance variables
     */
    private int value;
    private String name;
    private int year;

    /**
     * Default Constructor
     */
    public Coin() {
        int value = 0;
        String name = "";
        int year = 0000;
    }
    /**
     * Parameterized Constructor
     * @param value value of the coin
     * @param year year in which the coin was minted
     */
    public Coin(int value, int year) {
        this.setValue(value);
        this.setName();
        this.setYear(year);
    }
    /**
     * Getter method for value of the coin
     * @return integer value of the coin
     */
    public int getValue() {
        return value;
    }
    /**
     * getter method for coin year
     * @return integer value of coin year
     */
    public int getYear() {
        return year;
    }
    /**
     *getter method
     * @return returns the name of the coin
     */
    public String getName() {
        return name;
    }
    /**
     * setter method sets the name of a coin
     */
    public void setName() {
        String[] coinName = {"PENNY", "NICKEL", "DIME", "QUARTER", "NONAME"};
        if (this.getValue() == 1) {
            this.name = coinName[0];
        }
        else if (this.getValue() == 5) {
            this.name = coinName[1];
        }
        else if (this.getValue() == 10) {
            this.name = coinName[2];
        }
        else if (this.getValue() == 25) {
            this.name = coinName[3];
        }
        else {
            this.name = coinName[4];
        }
    }
    /**
     * setter method
     * @param value sets the value of the coin
     */
    public void setValue(int value) {
        this.value = value;
    }

    public void setYear(int year) {
        this.year = year;
    }
    /**
     * tostring method
     * @return the output in the given format
     */
    @Override
    public String toString() {
        return "[" + this.getValue() + ", " + this.getName() + ", " + this.getYear() +
"]";
    }
    /**
     * override equals method
     * @param obj object to be compared to
     * @return true if this and other object are the same
     */
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Coin other = (Coin) obj;
        if (name == null) {
            if (other.name != null)
                return false;
        } else if (!name.equals(other.name))
            return false;
        if (year != other.year)
            return false;
        if (value != other.value)
            return false;
        return true;
    }
    public static void main(String[] args) {
        Coin c1 = new Coin();
        Coin c2 = new Coin(1, 2002);
        Coin c3 = new Coin(5, 2005);
        Coin c4 = new Coin(10, 1977);
        Coin c5 = new Coin(25, 2001);
        Coin c6 = new Coin(25, 2001);
        System.out.println(c1);
        System.out.println(c2);
        System.out.println(c3);
        System.out.println(c5);
        System.out.println(c5.equals(c6));
        System.out.println(c3.equals(c4));
    }
}
```

class: CoinTest.java (JUnit 5)

```
/*
 * I affirm that I have carried out the attached academic endeavors with full academic
honesty.
 * Manav Bilakhia (MB)
 */
package assignment;

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

class CoinTest {
    Coin c1 = new Coin();
    Coin c2 = new Coin(1, 2002);
    Coin c5 = new Coin(25, 2001);
    Coin c6 = new Coin(25, 2001);
```

```java
    @Test
    void getValue() {
        assertEquals(25,c5.getValue());
    }
    @Test
    void getYear() {
        assertEquals(2001,c5.getYear());
    }
    @Test
    void getName() {
        assertEquals("PENNY",c2.getName());
    }
    @Test
    void equals() {
        assertEquals(true,c5.equals(c6));
        assertEquals(false,c1.equals(c6));
    }
}
```

Class: Item.java

```java
/*
 * I affirm that I have carried out the attached academic endeavors with full academic
honesty.
 * Manav Bilakhia (MB)
 */
package assignment;
/**
 *
 * @author Manav Bilakhia
 *
 */
public class Item {
    /*
     * Instance variables
     */
    private String description;
    private int price;

    /**
     *parameterized constructoe
     * @param description description of the item
     * @param price price of a given item
     */
    public Item(String description, int price)
    {
    this.setDescription(description);
    this.setPrice(price);
    }

    /**
     *getter method
     * @return the description of the item
     */
    public String getDescription() {
        return description;
    }

    /**
     *setter method
     * @param description sets the description of a given method
     */
    public void setDescription(String description) {
        this.description = description;
    }

    /**
     *getter method
     * @return the price of the given the item
     */
    public int getPrice() {
        return price;
    }

    /**
     * setter method
     * @param price sets the price of a given item
     */
    public void setPrice(int price) {
        this.price = price;
    }

    /**
     * to string method
     * @return the output in the given format
     */
    @Override
    public String toString() {
        int dollars = this.price/100;
        int cents = this.price%100;
        String toReturn = this.getDescription() + "\t" +"$"+ dollars + "."+cents;
        return toReturn;
    }
    /**
     * override equals method
     * @param obj object to be compared to
     * @return true if this and other object are the same
     */
    public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Item other = (Item) obj;
    if (description == null) {
        if (other.description != null)
            return false;
    } else if (!description.equals(other.description))
        return false;
    if (price != other.price)
        return false;
    return true;
    }
    public static void main(String[] args) {

        Item i2 = new Item("Shampoo", 2002);
        Item i3 = new Item("Shampoo", 2002);
        Item i4 = new Item("Chicken", 1977);
        System.out.println(i2);
        System.out.println(i3);
        System.out.println(i4);
        System.out.println(i3.equals(i2));
        System.out.println(i2.equals(i4));
    }
} // end Item
```

Class: ItemTest.java (JUnit)

```
/*
 * I affirm that I have carried out the attached academic endeavors with full academic
honesty.
 * Manav Bilakhia (MB)
 */
package assignment;

import org.junit.jupiter.api.Test;

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

class ItemTest {
    Item i2 = new Item("Shampoo", 2002);
    Item i3 = new Item("Shampoo", 2002);
    Item i4 = new Item("Conditioner", 1977);
    @Test
    void getDescription() {
        assertEquals("Conditioner",i4.getDescription());
    }

    @Test
    void getPrice() {
        assertEquals(1977,i4.getPrice());
    }

    @Test
    void testEquals() {
        assertEquals(false, i3.equals(i4));
        assertEquals(true, i3.equals(i2));
    }
}
```

Class: ResizableArrayBag.java

```
/*
 * I affirm that I have carried out the attached academic endeavors with full academic
honesty.
 * Manav Bilakhia (MB)
 */
package assignment;
import java.util.Arrays;
import java.util.StringJoiner;
/**
 * A class that implements a bag of objects by using an array. The bag is never
 * full.
 *
 * @author Frank M. Carrano, Timothy M. Henry
 * @version 5.0
 */
public final class ResizableArrayBag<T> implements BagInterface<T> {
    private T[] bag; // Cannot be final due to doubling
    private int numberOfEntries;
    private boolean integrityOK = false;
    private static final int DEFAULT_CAPACITY = 25; // Initial capacity of bag
    private static final int MAX_CAPACITY = 10000;
    /* Constructors */
    /** Constructor: No parameter. Creates an empty bag whose initial capacity is 25.
```

```java
*/
    public ResizableArrayBag()
    {
        this(DEFAULT_CAPACITY);
    }
    /**
     * Constructor: int parameter. Creates an empty bag having a given initial
capacity.
     *
     * @param initialCapacity The integer capacity desired.
     */
    public ResizableArrayBag(int initialCapacity)
    {
        checkCapacity(initialCapacity);
        T[] tempBag = (T[])new Object[initialCapacity]; // Unchecked cast
        bag = tempBag;
        numberOfEntries = 0;
        integrityOK = true;
    }
    /**
     * Constructor with an array parameter. Creates a bag containing given entries.
     *
     * @param contents An array of objects.
     */
    public ResizableArrayBag(T[] contents) {
        checkCapacity(contents.length);
        bag = Arrays.copyOf(contents, contents.length);
        numberOfEntries = contents.length;
        integrityOK = true;
    } // end constructor
    /**
     * Adds a new entry to this bag.
     *
     * @param newEntry The object to be added as a new entry.
     * @return True.
     */
    public boolean add(T newEntry) {
        checkintegrity();
        if (isArrayFull())
        {
            doubleCapacity();
        } // end if
        bag[numberOfEntries] = newEntry;
        numberOfEntries++;
        return true;
    } // end add
    /**
     * Retrieves all entries that are in this bag.
     *
     * @return A newly allocated array of all the entries in this bag.
     */
    public T[] toArray() {
        checkintegrity();
        T[] result = (T[])new Object[numberOfEntries];
        for (int i = 0; i < numberOfEntries; i++)
        {
            result[i] = bag[i];
        }
        return result;
    } // end toArray
    /**
     * Sees whether this bag is empty.
     *
     * @return True if this bag is empty, or false if not.
     */
    public boolean isEmpty() {
        if (numberOfEntries==0)
            return true;
        return false;
    } // end isEmpty
    /**
     * Gets the current number of entries in this bag.
     *
     * @return The integer number of entries currently in this bag.
     */
    public int getCurrentSize() {
        return numberOfEntries;
    } // end getCurrentSize
    /**
     * 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 ba.
     */
    public int getFrequencyOf(T anEntry) {
        checkintegrity();
        int count = 0;
        for (int i = 0; i < numberOfEntries; i++)
        {
            if (anEntry.equals(bag[i]))
            {
                count++;
            }
        }
        return count;
    } // 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) {
        checkintegrity();
        return getIndexOf(anEntry) >= 0;
    }  // end contains
    /** Removes all entries from this bag. */
    public void clear()
    {
        while (!isEmpty())
            remove();
    } // end clear
    /**
     * Removes one unspecified entry from this bag, if possible.
     *
     * @return Either the removed entry, if the removal was successful, or null.
     */
    public T remove() {
        checkintegrity();
        T toReturn = removeEntry(numberOfEntries - 1);
        return toReturn;
    } // 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) {
        checkintegrity();
        int index = getIndexOf(anEntry);
        T toReturn = removeEntry(index);
        return anEntry.equals(toReturn);
    } // end remove
    // Locates a given entry within the array bag.
    // Returns the index of the entry, if located,
    // or -1 otherwise.
    // Precondition: checkintegrity has been called.
    private int getIndexOf(T anEntry) {
        for (int i = 0; i < numberOfEntries; i++)
        {
            if (anEntry.equals(bag[i]))
                return i;
        }
        return -1;
    } // end getIndexOf
    // Removes and returns the entry at a given index within the array.
    // If no such entry exists, returns null.
    // Precondition: 0 <= givenIndex < numberOfEntries.
    // Precondition: checkintegrity has been called.
    private T removeEntry(int givenIndex)  {
        T toReturn = null;
        if (!isEmpty() && (givenIndex >= 0))
        {
            toReturn = bag[givenIndex];
            int last = numberOfEntries - 1;
            bag[givenIndex] = bag[last];
            bag[last] = null;
            numberOfEntries--;
        } // end if
        return toReturn;
    } // end removeEntry
    // Returns true if the array bag is full, or false if not.
    private boolean isArrayFull()  {
        if (numberOfEntries >= bag.length)
            return true;
        else
            return false;
    } // end isArrayFull
    // Doubles the size of the array bag.
    // Precondition: checkInitialization has been called.
    private void doubleCapacity() {
        int newLength = 2 * bag.length;
        checkCapacity(newLength);
        bag = Arrays.copyOf(bag, newLength);
    } // end doubleCapacity
    // Throws an exception if the client requests a capacity that is too large.
    private void checkCapacity(int capacity) {
        if (capacity > MAX_CAPACITY)
            throw new IllegalStateException("The capacity of the created bag exceeds
allowed maximum capacity");
    } // end checkCapacity
    // Throws an exception if receiving object is not initialized.
    private void checkintegrity() {
        if (!integrityOK)
            throw new SecurityException ("The object of ArrayBag is not initialized");
    } // end checkintegrity
    /**
     * toString joins the bag’s elements with a comma
     * and space then encloses in []
     */
    @Override
    public String toString() {
        StringJoiner joiner = new StringJoiner(", ", "[", "]");
        for (int index = 0; index < numberOfEntries; index++)
            joiner.add(bag[index].toString());
        return joiner.toString();
    }

    public static void main(String[] args) {
        ResizableArrayBag r1 = new ResizableArrayBag();
        System.out.println(r1.isEmpty());
        r1.add("A");
        r1.add("D");
        r1.add("B");
        r1.add("A");
        r1.add("C");
        r1.add("A");
        r1.add("D");
        System.out.println(r1);
        System.out.println(r1.isEmpty());
        System.out.println(r1.getFrequencyOf("A"));
        System.out.println(r1.getFrequencyOf("B"));
        System.out.println(r1.getFrequencyOf("C"));
        System.out.println(r1.getFrequencyOf("D"));
        System.out.println(r1.getFrequencyOf("Z"));
        System.out.println(r1.contains("A"));
        System.out.println(r1.contains("D"));
        System.out.println(r1.contains("Z"));
        System.out.println(r1.remove());
        System.out.println(r1.remove("B"));
        System.out.println(r1.remove("A"));
        System.out.println(r1.remove("C"));
        System.out.println(r1.remove("Z"));
        System.out.println(r1);
        r1.clear();
        System.out.println(r1.isEmpty());
    }
} // end ResizableArrayBag
/*
 * Write the following test in this class or in another driver class
 * Testing isEmpty with an empty bag: isEmpty finds the bag empty: OK.
 *
 * Adding to the bag more strings than its initial capacity. Adding to the bag:
 * A D B A C A D The bag contains 7 string(s), as follows: A D B A C A D Testing
 * isEmpty with a bag that is not empty: isEmpty finds the bag not empty: OK.
 *
 *
 * Testing the method getFrequencyOf: In this bag, the count of A is 3 In this
 * bag, the count of B is 1 In this bag, the count of C is 1 In this bag, the
 * count of D is 2 In this bag, the count of Z is 0
 *
 * Testing the method contains: Does this bag contain A? true Does this bag
 * contain B? true Does this bag contain C? true Does this bag contain D? true
 * Does this bag contain Z? false
 *
 * Removing a string from the bag: remove() returns D The bag contains 6
 * string(s), as follows: A D B A C A
 *
 * Removing "B" from the bag: remove("B") returns true The bag contains 5
 * string(s), as follows: A D A A C
 *
 * Removing "A" from the bag: remove("A") returns true The bag contains 4
 * string(s), as follows: C D A A
 *
 * Removing "C" from the bag: remove("C") returns true The bag contains 3
 * string(s), as follows: A D A
 *
 * Removing "Z" from the bag: remove("Z") returns false The bag contains 3
 * string(s), as follows: A D A
 *
 * Clearing the bag: Testing isEmpty with an empty bag: isEmpty finds the bag
 * empty: OK.
 *
 * The bag contains 0 string(s), as follows:
 */
```

Class: ResizableArrayBagTest.java

```
/*
 * I affirm that I have carried out the attached academic endeavors with full academic
honesty.
 * Manav Bilakhia (MB)
 */
package assignment;

import org.junit.jupiter.api.Test;

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

class ResizableArrayBagTest {
    ResizableArrayBag r1 = new ResizableArrayBag();
    @Test
    void add() {
        r1.add("A");
        r1.add("D");
        r1.add("B");
        r1.add("A");
        r1.add("C");
        r1.add("A");
        r1.add("D");
        String [] expectedArr = {"A","D","B","A","C","A","D"};
        assertArrayEquals(expectedArr, r1.toArray());

    }

    @Test
    void isEmpty()
    {   assertEquals(true, r1.isEmpty());
        r1.add("A");
        r1.add("D");
        assertEquals(false, r1.isEmpty());
    }

    @Test
    void getCurrentSize() {
        assertEquals(0,r1.getCurrentSize());
        r1.add("A");
        r1.add("D");
        assertEquals(2,r1.getCurrentSize());
    }

    @Test
    void getFrequencyOf() {
        r1.add("A");
        r1.add("D");
        r1.add("A");
        assertEquals(2,r1.getFrequencyOf("A"));
        assertEquals(0,r1.getFrequencyOf("Z"));
    }

    @Test
    void contains() {
        r1.add("A");
        r1.add("D");
        r1.add("A");
        assertEquals(true,r1.contains("A"));
        assertEquals(false,r1.contains("z"));
    }

    @Test
    void clear() {
        r1.add("A");
        r1.add("D");
        r1.add("A");
        r1.clear();
        assertEquals(0,r1.getCurrentSize());
    }

    @Test
    void remove() {
        r1.add("A");
        r1.add("D");
        r1.add("A");
        String [] expectedArr1 = {"A","D"};
        r1.remove();
        assertArrayEquals(expectedArr1,r1.toArray());
        r1.remove("A");
        String [] expectedArr2 = {"D"};
        assertArrayEquals(expectedArr2,r1.toArray());
    }
}
```

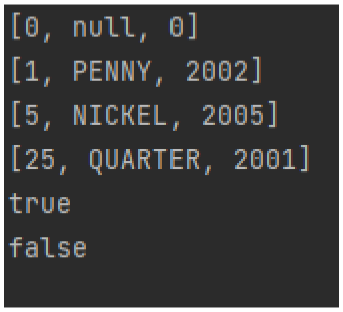
B. Sample output 1

I. Describe your test 1:
Checking to see if correct names are assigned to the correct value of a coin and also seeing if 2 coins are the same

II. Text output 1:

```
[0, null, 0]
[1, PENNY, 2002]
[5, NICKEL, 2005]
[25, QUARTER, 2001]
true
false
```

III. Screenshot 1:



```
[0, null, 0]
[1, PENNY, 2002]
[5, NICKEL, 2005]
[25, QUARTER, 2001]
true
false
```

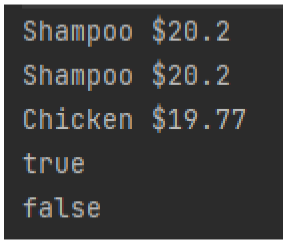
C. Sample output 2

I. Describe your test 2: checking if the price is displayed properly in the item class and if two items are the same.

II. Text output 2:

```
Shampoo     $20.2
Shampoo     $20.2
Chicken     $19.77
true
false
```

III. Screenshot 2:



```
Shampoo $20.2
Shampoo $20.2
Chicken $19.77
true
false
```

D. Sample output 3

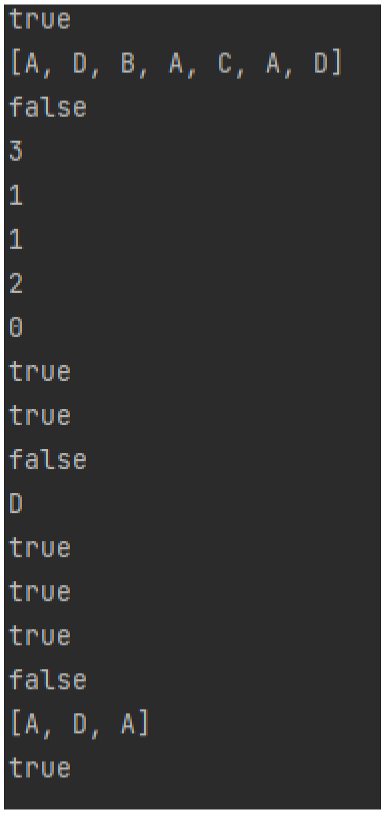
I. Describe your test 3: checking the add, isEmpty, getFrequencyOf, contains and remove function of the code. Please refer to the main method of this class for more information

II. Text output 3:

```
true
[A, D, B, A, C, A, D]
false
3
1
1
2
0
true
true
false
D
true
true
true
```

false
[A, D, A]
true

III. Screenshot 3:



```
true
[A, D, B, A, C, A, D]
false
3
1
1
2
0
true
true
false
D
true
true
true
false
[A, D, A]
true
```