CSC 151 Assignment 3

Objectives:

- Implementing Bags with Arrays
- Using Generics
- Using Interfaces

Definition

In this assignment you will implement various operations on array bags and use various types to create and call the methods of the array bags.

The interfaces and the classes in this assignment are partially or completely provided in your textbook. Please complete the partial implementations and submit all files listed in the submission instructions.

Interface BagInterface

- Use the code provided at the end of the document.
- It will be in **package assignment** (it is required otherwise **Gradescope** cannot run the tests)

Class Coin

- Use the partial code provided at the end of the document.
- It will be in **package assignment** (it is required otherwise **Gradescope** cannot run the tests)
- It has 3 private instance variables: int value, String name and int year
- It has 2 constructors: One with no parameter and assigns 0 for the value, 0000 for the year and appropriate name depending on the value. The other constructor has 2 parameters and assigns the instance variables similar to the parameterless constructor. You may use chaining.
- Provide set/get methods
- Provide a setName method that assigns name PENNY, NICKEL, DIME, QUARTER, NONAME for the values of 1, 5, 10, 25 and others.
- Provide a toString method that returns comma separated value, name, year enclosed in [ ]. Put a space after commas. Example: `[1, PENNY, 2010]`
- Provide equals method. (value, name, year should be equal)
- Test your class in the main method.
- Test your class by JUnit

Class Item

- Use the partial code provided at the end of the document.
- It will be in **package assignment** (it is required otherwise **Gradescope** cannot run the tests)
- It has 2 private instance variables: String description and int price
- It has 1 constructor: The constructor has 2 parameters and assigns them to description and price.

- Provide set/get methods
- Provide a toString method that returns description followed by a tab followed by a $, followed by price/100, followed by a ., followed by, price % 100. (A price value of 2010 means 20 dollars and 10 cents). Example: `Cheese	$20.10`
- Provide equals method. (description, price should be equal)
- Test your class in the main method.
- Test your class by JUnit

Class ResizableArrayBag

- Use the partial code provided at the end of the document.
- It will be in **package assignment** (it is required otherwise **Gradescope** cannot run the tests)
- See the details below written as comments and complete the partial code as instructed.
- Test your class in the main method.
- Test your class by JUnit

General Submission Instructions

- You may have groups for the assignments **if it is allowed.** If you have a group state it **explicitly in the honor code** by writing all group members' names. Please name the pdf files accordingly. **However, all group members should submit individually.**
- All submissions require a package called **assignment** and your source codes (.java files) will be under it.
- You should submit
    - Your java files to **Gradescope** as required and
    - Your java files and sample outputs for various scenarios as a **single pdf** file named as **YourFullName(s)_AssignmentN.pdf** with the **honor code** at the top as a comment to Nexus under the appropriate link. Replace N in AssignmentN with the appropriate value. (e.g., if I submit Assignment1 I will submit a pdf file whose name is **ZeynepOrhan_Assignment1.pdf.** If I have a group whose members are me and Jane Doe, then the file name will be **JaneDoeZeynepOrhanAssignment1.pdf** and the honor code will be modified). Use the template .docx file attached and modify it accordingly.
- **If you use any resource to get help, please state it in the honor code. Otherwise, it will be an honor code violation!!!**
- **Gradescope provides a feature to check code similarity by comparing all submissions. If the percentage of your code's similarity is above a threshold, then this will be an honor code violation!!!**

- Make sure that you have no compiler errors.
- When you have compiler error free codes and submit appropriately to Gradescope, your code will be tested automatically and graded. You do not need to know the test cases, but you will see the test results. Informative messages will be provided for success and failure cases.
- You can resubmit your files any number of times if you have failing tests until the due date.
- Your final submission or the one you selected will be graded after the due date of the assignment.
- Please start as early as possible so that you may have time to come and ask if you have any problems.
- Read the late submission policy in the syllabus.
- Please be aware that the correctness, obeying the OOP design and submission rules are equally important
- Gradescope testing will generally be 100% and sometimes manual grading will be done. Please be aware that **your grade can be reduced if you do not follow the instructions and apply good programming practices**.
- Use the starter code if provided.
- Do not use any predefined Collection classes of Java unless stated otherwise.

Assignment specific instructions

- General submission instructions apply.
- **Group work: NOT ALLOWED!!!**
- **Due date:** January 23, 2021 until 11:50 PM
- **Nexus:** Submit the code files (**BagInterface.java, Coin.java, Item.java, ResizableArrayBag.java**) as a single pdf file named as **YourFullName_Assignment3.pdf** with the honor code at the top as a comment under Assignment 3 link.
- **Gradescope:** Submit the java files (**BagInterface.java, Coin.java, Item.java, ResizableArrayBag.java**) under the Assignment 3
- **Grading:**
    - Gradescope: Yes
    - Manual: No
- **Starter code:** Use the code given below

```java
package assignment;
/*
 * Honor code
 */
/**
 * 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
```

```java
package assignment;

/**
 *
 * @author
 *
 */
public class Coin {
        /*
         * Instance variables
         */



        /**
         * Constructors
         */




      /**
         * set/get methods
         */







      /**
         * toString method
         */




        /**
         * equals method
         */



     /**
         * main method to test
         */



}
```

```java
package assignment;

/**
 *
 * @author
 */
public class Item {
        /*
         * Instance variables
         */


        /**
         * Constructor
         */
```

```
      /**
         * set/get methods
         */




      /**
         * toString method
         */



        /**
         * equals method
         */



     /**
         * main method to test
         */



} // end Item
```

```
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. */



        /**
         * Constructor: int parameter. Creates an empty bag having a given initial capacity.
         *
         * @param initialCapacity The integer capacity desired.
         */


        /**
         * Constructor with an array parameter. Creates a bag containing given entries.
         *
         * @param contents An array of objects.
         */
        public ResizableArrayBag(T[] contents) {

        } // end constructor

        /**
```

```java
    * 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) {


   } // 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() {


   } // end toArray

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


   } // end isEmpty

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


   } // 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) {


   } // 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) {


   } // end contains

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


   } // 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() {


   } // 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) {

        } // 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) {

        } // 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) {

        } // end removeEntry

        // Returns true if the array bag is full, or false if not.
        private boolean isArrayFull() {


        } // end isArrayFull

        // Doubles the size of the array bag.
        // Precondition: checkInitialization has been called.
        private void doubleCapacity() {

        } // end doubleCapacity

        // Throws an exception if the client requests a capacity that is too large.
        private void checkCapacity(int capacity) {

        } // end checkCapacity

        // Throws an exception if receiving object is not initialized.
        private void checkintegrity() {

        } // 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();

        }

} // 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:
 */
```