PSA - Assignment 2

Lab 4 Assignment 2

002615215

Kevin Rodrigues

Q1.

Problem description: The problem tests our knowledge about Abstract Data Type (ADT) and concept of Bags. We need to simulate a battle with the mythical Greek hydra. In this simulation, the hydra starts with just one head of a certain size. Each head has an integer size, and whenever you chop off a head that's larger than 1, two new smaller heads, each one unit smaller, grows in its place. The challenge is to figure out how many chops it takes to get rid of all the heads.

To keep track of the hydra's heads, you use two bags:

1. The first bag holds the current state of the hydra's heads, constantly updating as you chop them off and new ones grow.
2. The second bag counts how many chops you've made, by adding a "chop" each time you slice off a head.

The simulation runs until all the heads are chopped down to size 1 that cannot regrow and we can get rid of it, and the second bag gives you the total number of chops it took to win the battle.

Overflow scenarios, where the bags cannot hold more entries, are also considered in the simulation, which results in early termination if they occur, and we get 'computation ended early'.
This problem helps to transfer learning of ADT and bags in programming and helps build our thinking and logic ability.

Analysis:

Algorithm**:** Data Representation: The simulation uses the ArrayBag class to represent two key aspects of the problem:

- **The state of the hydra heads**: The heads are stored in a bag where each entry represents a head's size.

    - **The number of chops (cuts)**: Another bag is used to record each chop, adding a "chop" string each time a head is removed.

- **Simulation Loop**:
    - While the **head bag** is not empty, perform the following steps:
        - **Remove a head**: Use the remove() method to take a head from the head bag.
        - If the head's size is 1, it is discarded, as no new heads grow from it.
        - If the head's size is **greater than 1**, add two smaller heads (each one size smaller) to the bag using the add() method and record a chop.
    - The simulation continues until the head bag is empty or one of the bags overflows.

- **Termination**:
    - Once the simulation is complete, print the final state of the head bag and the total number of cuts recorded in the work bag.

Difficulties Encountered: Since the simulation involves continuously adding heads and recording chops, there is a possibility that either the head bag or the work bag could overflow. The Boolean value of 'noOverflow' decides whether our computation should end early or not. The logic in simulationStep() and linking it with add() of ArrayBag was bit confusing for me.

Better Solution: Maybe we can use LinkedList as they don't have fixed size, so we can handle large operations as well.

Source Code:

ArrayBag.java

ArrayBag.java × BagTest.java Hydra.java BagInterface.java

```java
package p1.lab4.info6205;

/**
 * A class of bags whose entries are stored in a fixed-size array.
 */

public final class ArrayBag<T> implements BagInterface<T> {

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

    private boolean initialized = false;
    private static final int MAX_CAPACITY = 10000;

    /** Creates an empty bag whose initial capacity is 25. */
    public ArrayBag() {
        this(DEFAULT_CAPACITY);
    } // end default constructor

    /**
     * Creates an empty bag having a given initial capacity.
     * @param desiredCapacity The integer capacity desired.
     */
    public ArrayBag(int desiredCapacity) {
        if (desiredCapacity <= MAX_CAPACITY) {

            // The cast is safe because the new array contains null entries.
            @SuppressWarnings("unchecked")
            T[] tempBag = (T[]) new Object[desiredCapacity]; // Unchecked cast
            bag = tempBag;
            numberOfEntries = 0;
            initialized = true;
        }
        else
            throw new IllegalStateException("Attempt to create a bag " +
```

```java
            throw new IllegalStateException("Attempt to create a bag " +
                                            "whose capacity exceeds " +
                                            "allowed maximum.");
    } // end constructor

    /** 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) {
        checkInitialization();
        boolean result = true;
        if (isArrayFull()) {
            result = false;
        } else { // Assertion: result is true here
            bag[numberOfEntries] = newEntry;
            numberOfEntries++;
        } // end if
        return result;
    } // end add

    public boolean isFull() {
        return numberOfEntries == bag.length;
    }

    /** Throws an exception if this object is not initialized.
     */
    private void checkInitialization()
```

ArrayBag.java × BagTest.java Hydra.java BagInterface.java

```java
    private void checkInitialization()
    {
        if (!initialized)
             throw new SecurityException("ArrayBag object is not initialized properly.");
    }

    /** Retrieves all entries that are in this bag.
     * @return A newly allocated array of all the entries in the bag.
     */
    public T[] toArray() {
        // the cast is safe because the new array contains null entries
        @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

    /** Sees whether this bag is full.
     * @return True if the bag is full, or false if not.
     */
    private boolean isArrayFull() {
        return numberOfEntries >= bag.length;
    } // end isArrayFull

    /** 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

    /** Gets the current number of entries in this bag.
     * @return The integer number of entries currently in the bag.
     */
```

ArrayBag.java × BagTest.java Hydra.java BagInterface.java

```java
     */
    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 the bag.
     */
    public int getFrequencyOf(T anEntry) {
        checkInitialization();
        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 the bag contains anEntry, or false if not.
     */
    public boolean contains(T anEntry) {
        checkInitialization();
        return getIndexOf(anEntry) > -1;
    } // end contains
```

ArrayBag.java × BagTest.java Hydra.java BagInterface.java

```java
    } // 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 if otherwise
     */
    public T remove() {
        checkInitialization();
        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) {
```

ArrayBag.java × BagTest.java Hydra.java BagInterface.java

```java
    public boolean remove(T anEntry) {
        checkInitialization();
        int index = getIndexOf(anEntry);
        T result = removeEntry(index);
        return anEntry.equals(result);
    } // end remove

    /** Removes and returns the entry at a given array index within the array bag.
     * If no such entry exists, returns null.
     * Preconditions: 0 <= givenIndex < numberOfEntries;
     *                checkInitialization has been called.
     */
    private T removeEntry(int givenIndex) {
        T result = null;
        if (!isEmpty() && (givenIndex >= 0)) {
            result = bag[givenIndex];                    // entry to remove
            bag[givenIndex] = bag[numberOfEntries - 1]; // Replace entry with last entry
            bag[numberOfEntries - 1] = null;             // remove last entry
           numberOfEntries--;
         } // end if
        return result;
    } // end removeEntry

    /** Locates a given entry within the array bag.
     * Returns the index of the entry, if located, or -1 otherwise.
     * Precondition: checkInitialization has been called.
     */
    private int getIndexOf(T anEntry) {
        int where = -1;
        boolean stillLooking = true;
        int index = 0;
        while ( stillLooking && (index < numberOfEntries)) {
            if (anEntry.equals(bag[index])) {
                stillLooking = false;
                where = index;
            } // end if
            index++;
        } // 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


    /** Override the toString() method so that we get a more useful display of
     *  the contents in the bag.
     * @return a string representation of the contents of the bag
     */
    public String toString() {
        String result = "Bag[ ";
        for (int index = 0; index < numberOfEntries; index++) {
            result += bag[index] + " ";
        } // end for
        result += "]";
        return result;
    } // end toString

} // end ArrayBag
```

//Hydra.java

ArrayBag.java | BagTest.java | Hydra.java | BagInterface.java

```java
package p1.lab4.info6205;

import java.io.*;

/**
 * Hydra is a program that will simulate the work done for a
 * computational task that can be broken down into smaller subtasks.
 */

public class Hydra {

    public static void main(String args[]) {
        ArrayBag<Integer> headBag = new ArrayBag<>();
        ArrayBag<String> workBag = new ArrayBag<>();

        int startingSize;

        System.out.println("Please enter the size of the initial head.");
        startingSize = getInt("   It should be an integer value greater than or equal to 1.");

        // size of the headBag
        headBag.add(startingSize);

        System.out.println("The head bag is " + headBag);

        boolean noOverflow = true;

        // ADD CODE HERE TO DO THE SIMULATION

        // simulation steps
        while (!headBag.isEmpty() && noOverflow) {

            // Step 6: Call simulationStep in the main method
            noOverflow = simulationStep(headBag, workBag);

            System.out.println();
```

ArrayBag.java | BagTest.java | Hydra.java | BagInterface.java

```java
            System.out.println();
        }

        if (noOverflow) {
            System.out.println("The number of chops required is " + workBag.getCurrentSize());
        } else {
            System.out.println("Computation ended early with a bag overflow");
        }

    }

    /**
     * Take one head from the headBag bag.  If it is a final head, we are done with it.
     * Otherwise put in two heads that are one smaller.
     * Always put a chop into the work bag.
     *
     * @param  headBag   A bag holding the heads yet to be considered.
     * @param  workBag   A bag of chops.
     */

```

```java
    public static boolean simulationStep(ArrayBag<Integer> heads, ArrayBag<String> work) {

        // Size of the current head
        Integer currentHeadSize = heads.remove();

        boolean result = true;

        // If there's no head to process (currentHeadSize is null), we set result to false
        if (currentHeadSize == null) {
            result = false;
            return result;
        }

        // Check if the head is a final head (larger than 1)
        if (currentHeadSize > 1) {
            // Add two smaller heads to the bag
            heads.add(currentHeadSize - 1);
            heads.add(currentHeadSize - 1);
        }

        // Check if either bag is full
        if (heads.isFull() || work.isFull()) {
            result = false;
            return result; // Simulation cannot continue due to overflow
        }

        System.out.println("The head bag is now " + heads);

        // Add a chop to the work bag
        work.add("chop");

        // Display the current state of the work bag
        System.out.println("The work bag is now " + work);


        return result;
    }

    /**
     * Get an integer value.
     * @return     An integer.
     */
    private static int getInt(String rangePrompt) {
        Scanner input;
        int result = 10;        //default value is 10
        try {
            input = new Scanner(System.in);
            System.out.println(rangePrompt);
            result = input.nextInt();
        } catch (NumberFormatException e) {
            System.out.println("Could not convert input to an integer");
            System.out.println(e.getMessage());
            System.out.println("Will use 10 as the default value");
        } catch (Exception e) {
            System.out.println("There was an error with System.in");
            System.out.println(e.getMessage());
            System.out.println("Will use 10 as the default value");
        }
        return result;
    }
}
```

Output:

Input 3:

```
Console ×
<terminated> Hydra [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe  (Oct 2, 2024, 10:15:42 PM – 10:15:45 PM) [pid: 29300]
Please enter the size of the initial head.
   It should be an integer value greater than or equal to 1.
3
The head bag is Bag[ 3 ]
The head bag is now Bag[ 2 2 ]
The work bag is now Bag[ chop ]

The head bag is now Bag[ 2 1 1 ]
The work bag is now Bag[ chop chop ]

The head bag is now Bag[ 2 1 ]
The work bag is now Bag[ chop chop chop ]

The head bag is now Bag[ 2 ]
The work bag is now Bag[ chop chop chop chop ]

The head bag is now Bag[ 1 1 ]
The work bag is now Bag[ chop chop chop chop chop ]

The head bag is now Bag[ 1 ]
The work bag is now Bag[ chop chop chop chop chop chop ]

The head bag is now Bag[ ]
The work bag is now Bag[ chop chop chop chop chop chop chop ]

The number of chops required is 7
```

Input 4:

```
<terminated> Hydra [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe  (Oct 2, 2024, 10:16:38 PM – 10:16:41
Please enter the size of the initial head.
   It should be an integer value greater than or equal to 1.
4
The head bag is Bag[ 4 ]
The head bag is now Bag[ 3 3 ]
The work bag is now Bag[ chop ]

The head bag is now Bag[ 3 2 2 ]
The work bag is now Bag[ chop chop ]

The head bag is now Bag[ 3 2 1 1 ]
The work bag is now Bag[ chop chop chop ]

The head bag is now Bag[ 3 2 1 ]
The work bag is now Bag[ chop chop chop chop ]

The head bag is now Bag[ 3 2 ]
The work bag is now Bag[ chop chop chop chop chop ]

The head bag is now Bag[ 3 1 1 ]
The work bag is now Bag[ chop chop chop chop chop chop ]
```

```
The head bag is now Bag[ 3 1 ]
The work bag is now Bag[ chop chop chop chop chop chop chop ]

The head bag is now Bag[ 3 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop ]

The head bag is now Bag[ 2 2 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop ]

The head bag is now Bag[ 2 1 1 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop chop ]

The head bag is now Bag[ 2 1 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop chop chop ]

The head bag is now Bag[ 2 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop chop chop chop ]

The head bag is now Bag[ 1 1 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop chop chop chop chop ]

The head bag is now Bag[ 1 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop chop chop chop chop chop ]

The head bag is now Bag[ ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop ]

The number of chops required is 15
```

Input 5: Overflow

```
<terminated> Hydra [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe  (Oct 2, 2024, 10:18:01 PM – 10:1
Please enter the size of the initial head.
   It should be an integer value greater than or equal to 1.
5
The head bag is Bag[ 5 ]
The head bag is now Bag[ 4 4 ]
The work bag is now Bag[ chop ]

The head bag is now Bag[ 4 3 3 ]
The work bag is now Bag[ chop chop ]

The head bag is now Bag[ 4 3 2 2 ]
The work bag is now Bag[ chop chop chop ]

The head bag is now Bag[ 4 3 2 1 1 ]
The work bag is now Bag[ chop chop chop chop ]

The head bag is now Bag[ 4 3 2 1 ]
The work bag is now Bag[ chop chop chop chop chop ]

The head bag is now Bag[ 4 3 2 ]
The work bag is now Bag[ chop chop chop chop chop chop ]
```

```
The head bag is now Bag[ 4 3 1 1 ]
The work bag is now Bag[ chop chop chop chop chop chop chop ]

The head bag is now Bag[ 4 3 1 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop ]

The head bag is now Bag[ 4 3 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop ]

The head bag is now Bag[ 4 2 2 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop chop ]

The head bag is now Bag[ 4 2 1 1 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop chop chop ]

The head bag is now Bag[ 4 2 1 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop chop chop chop ]

The head bag is now Bag[ 4 2 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop chop chop chop chop ]
```

```
The head bag is now Bag[ 4 1 1 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop chop chop chop chop chop ]

The head bag is now Bag[ 4 1 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop ]

The head bag is now Bag[ 4 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop ]

The head bag is now Bag[ 3 3 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop ]

The head bag is now Bag[ 3 2 2 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop ]

The head bag is now Bag[ 3 2 1 1 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop ]

The head bag is now Bag[ 3 2 1 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chc

The head bag is now Bag[ 3 2 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chc

The head bag is now Bag[ 3 1 1 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chc
```

```
The head bag is now Bag[ 3 2 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chc

The head bag is now Bag[ 3 1 1 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chc

The head bag is now Bag[ 3 1 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chc

The head bag is now Bag[ 3 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chc

The head bag is now Bag[ 2 2 ]
The work bag is now Bag[ chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chop chc


Computation ended early with a bag overflow
```

Q2:

Problem description: In this problem, we need to manage a stack of books on our desk. We have a pile of heavy books, and you can only remove or add books to the top of the stack. You can't pull books from the middle or bottom of the pile. The task is to design a class that keeps track of the books in the pile. Each book is represented by its title, and you'll implement a set of operations to interact with the pile. These operations include checking if the pile is empty, adding a new book to the top of the pile, peeking at the top book without removing it, and removing the top book. You'll also need to implement a method to clear the entire pile.

We need to create PileInterface, which will define these operations. Then, you'll implement a class called PileOfBooks that uses a resizable array to manage the pile. Once you've built this, you'll test the class PileOfBooksTest using a provided testing program to ensure that each operation works correctly, like adding, removing, and peeking at books. This problem introduces key concepts of stack-based operations using Abstract Data Types (ADT) and resizable arrays.

Analysis:

Algorithm: The Pile of Books is represented using a resizable ArrayList in the PileOfBooks class. This allows dynamic management of the pile of books, with the flexibility to add or remove books from the top of the pile. The PileInterface defines the basic stack-like operations for interacting with the pile, including methods to check if the pile is empty, add a book, remove the top book, view the top book without removing it, and clear the pile.

Methods:

isEmpty(): This method checks whether the pile contains any books. It returns true if the pile is empty and false otherwise

add(T book): This method adds a new book to the top of the pile. Since the pile behaves like a stack, new books are always placed at the end of the ArrayList.

remove(): This method removes the top book from the pile. If the pile is empty, it returns null; otherwise, it removes and returns the last book in the list, maintaining the stack structure.

getTopBook(): This method returns the title of the book at the top of the pile without removing it. If the pile is empty, it returns null.

clear(): This method clears all books from the pile by emptying the ArrayList

The resizable nature of ArrayList ensures that the pile can grow and shrink dynamically as books are added or removed.

Difficulties:

The methods remove() and getTopBook() has to be implemented with some proper thoughts. I had to ensure that we pass the index number i.e. (size of pile – 1), so that appropriate element is removed and retrieved respectively.

Alternative Solution:

Java has it's own stack class through which we can perform push, pop, peek operations. So, we can consider that as well.

Source Code:

PileInterface

PileOfBooksTest.java | PileInterface.java | PileOfBooks.java

```java
package p2.lab4.info6205;

public interface PileInterface<T> {
    boolean isEmpty();
    void add(T book);
    T remove();
    T getTopBook();
    void clear();
}

```

PileOfBooks

PileOfBooksTest.java | PileInterface.java | PileOfBooks.java

```java
package p2.lab4.info6205;

import java.util.ArrayList;

public class PileOfBooks<T> implements PileInterface<T> {
    private ArrayList<T> bookPile;

    public PileOfBooks() {
        bookPile = new ArrayList<>();
    }

    @Override
    public boolean isEmpty() {
        return bookPile.isEmpty();
    }

    @Override
    public void add(T book) {
        bookPile.add(book);
    }

    @Override
    public T remove() {
        if (isEmpty()) {
            return null;
        }
        return bookPile.remove(bookPile.size() - 1);
    }

    @Override
    public T getTopBook() {
        if (isEmpty()) {
            return null;
```

```java
            return null;
        }
        return bookPile.get(bookPile.size() - 1);
    }

    @Override
    public void clear() {
        bookPile.clear();
    }
}
```

Console

PileOfBooksTest

PileOfBooksTest.java × PileInterface.java PileOfBooks.java

```java
package p2.lab4.info6205;

/**
 *  A driver that demonstrates the class PileOfBooks.
 */
public class PileOfBooksTest
{
    public static void main(String[] args)
    {
        System.out.println("Create an empty pile of books: ");
        PileInterface<String> myPile = new PileOfBooks<>();
        System.out.println("isEmpty() returns " + myPile.isEmpty() + "\n");

        System.out.println("Add to pile.\n");
        myPile.add("And Then There Were None");
        myPile.add("The Hobbit");
        myPile.add("The Lord of the Rings");
        myPile.add("The Da Vinci Code");
        myPile.add("The Catcher in the Rye");

        System.out.println("isEmpty() returns " + myPile.isEmpty() + "\n");

        System.out.println("Testing peek and pop:\n");
        while (!myPile.isEmpty())
        {
            String top = myPile.getTopBook();
            System.out.println(top + " is at the top of the pile.");

            top = myPile.remove();
            System.out.println(top + " is removed from the pile.\n");
        }

        System.out.println("The pile should be empty: ");
```

```java
        System.out.print("The pile should be empty: ");
        System.out.println("isEmpty() returns " + myPile.isEmpty() + "\n\n");

        System.out.println("Add to the pile.");
        myPile.add("Anne of Green Gables");
        myPile.add("The Purpose Driven Life");
        myPile.add("The Girl with the Dragon Tattoo");

        System.out.println("\nTesting clear:\n");
        myPile.clear();

        System.out.print("The pile should be empty: ");
        System.out.println("isEmpty() returns " + myPile.isEmpty() + "\n\n");

        System.out.println("myPile.getTopBook() returns " +  myPile.getTopBook());
        System.out.println("myPile.remove()  returns " +  myPile.remove() + "\n");
        System.out.println("\nDone.");
    }
}
```

Output:

Console × PileOfBooksTest.java PileInterface.java PileOfBooks.java

```
<terminated> PileOfBooksTest [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe  (Oct 2, 2024, 11:17:51 PM – 11:17:52 PM) [pid: 25032]
Create an empty pile of books:
isEmpty() returns true

Add to pile.

isEmpty() returns false

Testing peek and pop:

The Catcher in the Rye is at the top of the pile.
The Catcher in the Rye is removed from the pile.

The Da Vinci Code is at the top of the pile.
The Da Vinci Code is removed from the pile.

The Lord of the Rings is at the top of the pile.
The Lord of the Rings is removed from the pile.

The Hobbit is at the top of the pile.
The Hobbit is removed from the pile.

And Then There Were None is at the top of the pile.
And Then There Were None is removed from the pile.

The pile should be empty:
isEmpty() returns true


Add to the pile.

Testing clear:

The pile should be empty: isEmpty() returns true


myPile.getTopBook() returns null
myPile.remove()  returns null


Done.
```

Q3:

Source Code:

MyBagInterface.java

MyBagInterface.java × MyBagTest.java MyBag.java

```java
package p3.lab4.info6205;

import java.util.Iterator;

interface MyBagInterface<T> extends Iterable<T> {

    void add(T item);           // Adds an item to the Bag
    boolean remove(T item);     // Removes one occurrence of the item from the Bag and retur
    boolean isEmpty();          // Returns true if the Bag is empty or false if not
    int size();                 // Returns the number of items in the Bag

    Iterator<T> iterator();                                     // Returns an iterator to it
    MyBagInterface<T> union(MyBagInterface<T> other);           // Returns the union of this
    MyBagInterface<T> intersection(MyBagInterface<T> other);    // Returns the intersection
    MyBagInterface<T> difference(MyBagInterface<T> other);      // Returns the difference be

}
```

MyBag.java

MyBagInterface.java MyBagTest.java MyBag.java

```java
package p3.lab4.info6205;

import java.util.Iterator;

public class MyBag<T> implements MyBagInterface<T> {

     private ArrayList<T> bagContents;

        public MyBag() {
            bagContents = new ArrayList<>();
        }

    @Override
    public void add(T item) {
        // ADD CODE HERE
         bagContents.add(item);
    }

    @Override
    public boolean remove(T item) {
        boolean removed = false;
        Iterator<T> iterator = bagContents.iterator();

        // Iterate through the bag contents
        while (iterator.hasNext()) {
            if (iterator.next().equals(item)) {
                iterator.remove();
                removed = true;
                break;
            }
        }
```

MyBagInterface.java MyBagTest.java MyBag.java

```java
        }
        return removed;
    }

    @Override
    public boolean isEmpty() {
        return bagContents.isEmpty();
    }

    @Override
    public int size() {
        return bagContents.size();
    }
```

MyBagInterface.java MyBagTest.java MyBag.java

```java

    @Override
    public Iterator<T> iterator() {
        return new Iterator<T>() {
            private int currentIndex = 0;

            @Override
            public boolean hasNext() {
                return currentIndex < bagContents.size();
            }

            @Override
            public T next() {
                if (hasNext()) {
                    return bagContents.get(currentIndex++);
                } else {
                    throw new IllegalStateException("No more elements");
                }
            }

            @Override
            public void remove() {
                if (currentIndex == 0) {
                    throw new IllegalStateException("Call next() before remove()");
                }
                bagContents.remove(--currentIndex);
            }
        };
    }
```

MyBagInterface.java MyBagTest.java MyBag.java

```java

    @Override
    public MyBagInterface<T> union(MyBagInterface<T> other) {
        MyBag<T> result = new MyBag<>();  // Create a new bag to hold the union
        result.bagContents.addAll(this.bagContents);  // Add all elements from the current bag

        for (T item : other) {
            result.add(item);  // Add each element from the other bag as well
        }

        return result;
    }

    @Override
    public MyBagInterface<T> intersection(MyBagInterface<T> other) {
        MyBag<T> result = new MyBag<>();  // Create a new bag for the intersection
        ArrayList<T> otherContents = new ArrayList<>();
        for (T item : other) {
            otherContents.add(item);  // Copy items from the other bag into a list
        }
```

MyBagInterface.java × MyBagTest.java MyBag.java ×

```java
        }

        for (T item : this.bagContents) {
            if (otherContents.contains(item)) {
                result.add(item);  // Add to result if the item exists in both bags
                otherContents.remove(item);  // Remove it to prevent duplicates
            }
        }
        return result;
    }

    @Override
    public MyBagInterface<T> difference(MyBagInterface<T> other) {
        MyBag<T> result = new MyBag<>();  // Create a new bag for the difference
        result.bagContents.addAll(this.bagContents);  // Add all elements from the current bag

        for (T item : other) {
            result.remove(item);  // Remove the items that are also in the other bag
        }
        return result;
    }

    public void clear() {
        bagContents.clear();  // This will remove all elements from the internal list
    }

    public String toString() {
        return bagContents.toString();
    }
}
```

MyBagTest.java

MyBagInterface.java | MyBagTest.java | MyBag.java

```java
package p3.lab4.info6205;

/* Implement MyBag concrete class in order to run this test */

public class MyBagTest {
    public static void main(String[] args) {

        MyBagInterface<Integer> bag1 = new MyBag<>();
        MyBagInterface<Integer> bag2 = new MyBag<>();

        // Add elements to bag1
        bag1.add(1);
        bag1.add(2);
        bag1.add(2);
        bag1.add(3);

        // Add elements to bag2
        bag2.add(2);
        bag2.add(3);
        bag2.add(4);

        // Print the initial state of bag1 and bag2 (implement toString() in Bag concrete class)
        System.out.println("Bag 1: " + bag1);  // Bag 1: [1, 2, 2, 3] ORDER DOESNT MATTER
        System.out.println("Bag 2: " + bag2);  // Bag 2: [2, 3, 4] ORDER DOESNT MATTER

        // Perform Union
        MyBagInterface<Integer> unionBag = bag1.union(bag2);
        System.out.println("Union of Bag 1 and Bag 2: " + unionBag);  // Expected: [1, 2, 2, 3, 2, 3, 4]
                                                                      // ORDER DOESNT MATTER, element and frequenci
        // Perform Intersection
        MyBagInterface<Integer> intersectionBag = bag1.intersection(bag2);
```

```java
        System.out.println("Intersection of Bag 1 and Bag 2: " + intersectionBag);  // Expected: [2, 3
                                                                                    // ORDER DOESNT MA
        // Perform Difference
        MyBagInterface<Integer> differenceBag = bag1.difference(bag2);
        System.out.println("Difference of Bag 1 and Bag 2: " + differenceBag);  // Expected: [1, 2]
                                                                                // ORDER DOESNT MATTER
        // Remove an element from bag1
        bag1.remove(2);
        System.out.println("Bag 1 after removing 2: " + bag1);  // Expected: [1, 2, 3]
                                                                // ORDER DOESNT MATTER, element and fr
        // Check the size of bag1
        System.out.println("Bag 1 size: " + bag1.size());  // Expected: 3

        // Check if bag1 is empty
        System.out.println("Bag 1 isEmpty: " + bag1.isEmpty());  // Expected: false

        // Clear bag1 and check if it is empty
        for(int i : bag1)bag1.remove(i);
        System.out.println("Bag 1 isEmpty after clear: " + bag1.isEmpty());  // Expected: true

        // Create a new bag for testing
        MyBagInterface<String> bag3 = new MyBag<>();
        bag3.add("apple");
        bag3.add("banana");
        bag3.add("apple");
```

```java
        // Create a new bag for testing
        MyBagInterface<String> bag3 = new MyBag<>();
        bag3.add("apple");
        bag3.add("banana");
        bag3.add("apple");

        System.out.println("Bag 3: " + bag3);  // Expected: [apple, banana, apple]
                                               // ORDER DOESNT MATTER
        // Remove an element from bag3
        bag3.remove("apple");
        System.out.println("Bag 3 after removing 'apple': " + bag3);  // Expected: [banana, apple]
                                                                      // ORDER DOESNT MATTER
    }
}
```

Output:

```
<terminated> MyBagTest [Java Application] C:\Program Files\Java\jdk-17\bin\ja
Bag 1: [1, 2, 2, 3]
Bag 2: [2, 3, 4]
Union of Bag 1 and Bag 2: [1, 2, 2, 3, 2, 3, 4]
Intersection of Bag 1 and Bag 2: [2, 3]
Difference of Bag 1 and Bag 2: [1, 2]
Bag 1 after removing 2: [1, 2, 3]
Bag 1 size: 3
Bag 1 isEmpty: false
Bag 1 isEmpty after clear: false
Bag 3: [apple, banana, apple]
Bag 3 after removing 'apple': [banana, apple]
```