COM498 Algorithms & Data Structures

Assignment 1 – Part 2

PRACTICE

You have been provided with the file *Bag.java*, which defines a **Bag** data type as an un-ordered collection of generic elements stored in an array. The following methods on **Bag** objects have been implemented.

```
public int getCurrentSize();
/* the  number of elements currently contained in the bag

@return - (int) number of elements
*/

public boolean isEmpty();
/* test for an empty bag

@return - (boolean) true if the number of elements is zero, false otherwise
*/

public boolean addNewEntry(T newEntry);
/* test for spare capacity in the bag and if it exists, add the new element
and increment the number of elements in the bag

@param (T) newEntry - the item to be added to the bag
@return - (boolean) true if the item is added, false otherwise
*/

public T remove();
/* remove any element from the bag, if one is available, and decrement the number
of elements.  Return the remove element or null if none is available

@return - (T) the element removed from the bag, or null
*/

public boolean remove(T anEntry);
/* remove the specified element from the bag, if it is present, and decrement
the number of elements

@return - (boolean) true if the element is available, false otherwise
*/

public void clear();
/* empty the bag - set the number of elements to zero
*/

public int getFrequencyOf(T anEntry);
/* count how many times a given element appears in the bag

@param (T) anEntry - the element to look for in the bag
@return - (int) the number of times that the given element appears
*/

public boolean contains(T anEntry);
/* check for the presence of a specified element in the bag

@param (T) anEntry - the item to look for in the bag
@return - (boolean) true if the bag contains the element, false otherwise
*/

public T[] toArray();
/* find all elements in the bag

@return - (T[]) an array of size number of elements, containing all contents of the bag
*/
```

You have been asked to produce the new object definition **BagSet**, which is a variation of **Bag** in which any object can only appear at most once, as well as a collection of methods to apply to **BagSet** objects.

Tasks

1. Create a new Java Project in IntelliJ called **Assignment1** and copy the files *Bag.java*, *BagSet.java* and *BagSetTest.java* into its **src** folder.

2. Modify the definition of the new class **BagSet** so that it is defined as a **sub-class** of **Bag**.

3. Write a new **`addNewEntry()`** method for **BagSet** that overrides the inherited behaviour and prevents any value being added to the **BagSet** more than once.

4. Provide the code for the following new methods for **BagSet** (the skeletons of each of these methods have been provided for you and you should discard any code currently in the method body). Note that these methods should be **non-destructive** – the state of any **BagSet** object is unaffected by the method.

    a. **`union(anotherBagSet)`**

    which returns a **BagSet** containing only those elements that appear both in the object **BagSet** and the parameter **`anotherBagSet`**.

    NOTE: the **`union()`** method has been provided for you in *BagSet.java* as an example of some of the techniques that you might need in order to implement the other methods

    b. **`intersection(anotherBagSet)`**

    which returns a **BagSet** containing only those elements that appear both in the object **BagSet** and the parameter **`anotherBagSet`**.

    c. **`difference(anotherBagSet)`**

    which returns a **BagSet** containing those objects that appear in the object **BagSet** but not in the parameter **`anotherBagSet`**.

    d. **`equals(anotherBagSet)`**

    which returns **true** if the object **BagSet** contains the same collection of elements as **`anotherBagSet`** and false otherwise. Remember that the order of elements in a Bag is insignificant.

5. Run the **`main()`** method in the file *BagSetTest.java*, which conducts tests on your new methods. If any of these tests fail, you should spend any remaining time de-bugging your code and re-running the tests.

6. Take a screenshot of your final attempt at running **BagSetTest** and upload this, along with your source code from *BagSet.java* to the link provided on Blackboard.

Marking Scheme

60% of the marks for Assignment 1 are available from this exercise. Marks will be allocated as follows:

Element	Marks
Modification of **`BagSet`** definition	12
Implementation of **`addNewEntry()`** method	12
Implementation of **`union()`** method	0
Implementation of **`intersection()`** method	12
Implementation of **`difference()`** method	12
Implementation of **`equals()`** method	12

Target Output

If your **`BagSet`** class has been properly implemented, running the **`main()`** method in the **`BagSetTest`** class should generate the following output.

```
A = Bag[ A B C ]
B = Bag[ C D ]

A union B = Bag[ A B C D ]
Union test - true

A intersection B = Bag[ C ]
Intersection test - true

A difference B = Bag[ A B ]
Difference test - true

A == B is false
B = Bag[ C B A ]
A == B is true

Process finished with exit code 0
```

Submission

Please submit a screenshot of your final attempt at running **`BagSetTest`** and your file *BagSet.java* to the **Assessment 1 Part 2 - Practice** link in the **Assessment** section on Blackboard.