COSC2436 Programming Fundamentals III /ITSE2445 Data Structures

The submission opens two days before the due day, and closes right at the due time on due day. One submission is allowed per project. It is students' responsibility to submit timely and correctly. Students MUST start each project when it is first discussed in class in order to complete it on time. All projects are individual projects. Project due days may be changed to better align with lectures.

If one of the following is true, not credits will be given:

- The project is late.
- The algorithm or class design is missing.
- The project has errors.
- There are no comments (Javadoc format required) at all in your code.
- Wrong files are submitted.
- **A project is a copy and modification of another student's project. (Both will receive 0.)**

Files to be submitted through Canvas:

- The UML design (UML class diagram)
- All source codes and driver programs
- All supporting files if any

Software Development Project Rubric: (Design is worth 20%; the rest is worth 80 %.)
Analysis: There will be no credit if the software doesn't meet customer specification at all.

- Does the software meet the exact customer specification?
- Does the software read the exact input data and display the exact output data as they are shown in sample runs?
- Does each class include all corresponding data and functionalities?

Design: There will be no credit if the design is missing.

- Is the design (a UML class diagram) an efficient solution?
- Is the design created correctly?

Code: There will be no credit if there are errors.

- Are there errors in the software?
- Are code conventions and name conventions followed?
- Does the software use the minimum computer resource (computer memory and processing time)?
- Is the software reusable?

Debug:

- Are there bugs in the software?

Documentation: There will be no credit if comments are not included.

- Are there enough comments included in the software?
- Class comments must be included before a class header.
- Method comments must be included before a method header.
- More comments must be included inside each method.
- All comments must be written in Javadoc format.

Project 1 ADT Bag **Due: See calendar**

The ADT Bag is a group of items, much like what you might have with a bag of groceries. Note that the items in the bag are in no particular order and the bag may contain duplicate items. In a software development cycle, specification, design, implementation, test/debug, and documentation are typical activities.

ADT Bag Specification:

Specify operations to

- put an item in the bag *(insert(item)),*
- remove the last item put in the bag *(removeLast()),*
- remove a random item from the bag *(removeRandom()),*
- get an item from the bag(get(index)),
- check how many items are in the bag *(size()),*
- check to see if the bag is full *(isFull()),*
- check to see if the bag is empty*(isEmpty()),*
- and completely empty the bag *(makeEmpty()).*

Exceptions should to be considered when operations are designed.

Java has two types of exceptions: checked exceptions and runtime exceptions.

Checked exceptions are instances of classes that are sub classes of *java.lang.Exception* class. They must be handled locally or explicitly thrown from the method. They are typically used when the method encounters a serious problem. In some cases, the error may be considered serious enough that the program should be terminated.

Runtime exceptions occur when the error is not considered as serious. These types of exceptions can often be prevented by fail-safe programming. For example, it is fairly easy to avoid allowing an array index to go out of range, a situation that causes the runtime exception *ArrayIndexOutOfBoundsException* to be thrown. Runtime exceptions are instances of classes that are subclasses of the *java.lang.RuntimeException* class. *RuntimeException* is a subclass of *java.lang.Exception* that relaxes the requirement forcing the exception to be either handled or explicitly thrown by the method.

In general, some operations of an ADT list can be provided with an index value. If the index value is out of range, an exception should be thrown. Therefore, a subclass of *IndexOutOfBoundException* needs to be defined. In this case, items are in no particular order. Therefore, no operations should be provided with an index value.

Also, an exception is needed when the list storing the items becomes full. A subclass of *java.lang.RuntimeException* should be defined for this erroneous situation. A full ADT bag should throw an exception when a new item is inserted.

ADT Bag Design:

Complete a UML diagram to include all classes that are needed to meet the specifications. An interface class is usually defined to include all operations. A class implementing this interface provides implementation details. Exception classes are define if needed.

ADT Bag Array-based Implementation:

Implement all classes included the design. Javadoc comments should be included during this activity. Use an array to store the items in an ADT Bag list. The element type of the array storing the list should be *Object* type.

ADT Bag Test/Debug:

Complete a driver program to test all operations in the design. In general, an instance of the ADT Bag is created with an empty list, and then, fill the bag list with items read from a file, invoke any operations via the list, and always display the list after changes. Method decomposition must be used to break method main into parts. All testing data must be included in a text file. Here is a template of method *main*:

```
public static void main(String[] args) {
       //Create an instance of the ADT Bag
       ADTBagArrayBased list = new ADTBagArrayBased();

       //Fill a list. A Scanner object must be used to read item information from a file.
       //Items may need to be created before they are stored into the list.
       fillList(list);

       //Invoke other operations
       //Display the list
       …
       displayList(list);
}
```

Note:

An *Item* type can be designed to represent an item in the bag. A *String* object can be used to represent an item in the bag.

ADT Bag Documentation:

Check the entire project, and complete /revise Javadoc comments if needed.