CS 445: Data Structures
Fall 2019

Assignment 1

Assigned: Monday, September 16 **Due:** Monday, September 30 11:59 PM

---

1 Motivation

In CS 445, we often discuss the importance of data structure design and implementation to the wide variety of computing applications. Despite decades of study and development of common libraries, organizations must still regularly develop custom data structures to fulfill their applications' specific needs, and as such the field remains hugely relevant to both computer scientists and software engineers.

In this assignment, you will implement two data structures to satisfy their specifications, which are provided in the form of interfaces.

> **Note:** Your goal in this assignment is to write classes that faithfully implement the ADTs described in the interfaces. Sample client code is provided as *one example* of how the classes may be used. However, this sample code does not test all of the capabilities you are asked to implement, and thus your goal is *not* simply to make this sample code work. You are *strongly encouraged* to write your own test client as well, to further test that all operations of your classes work in all the corner cases described in the interfaces.

2 Provided code

First, look over the provided code. You can find this code on Pitt Box in a folder named `cs445-a1-abc123`, where `abc123` is your Pitt username.

The `SetInterface<E>` interface describes a set, a data structure similar in concept to a bag except that it does not allow duplicates. It is a generic interface that declares abstract methods for adding an item, removing an item (specified or unspecified), checking if the set is empty, determining the number of items in the set, and fetching the items from the set into an array. You should *not* modify this interface.

The `SetFullException` class is included to allow potential implementations of `SetInterface` that have a fixed capacity. Your implementation should *not* be fixed capacity, and thus should not throw this exception.

The `GroceryItem` class is a simple way of representing a grocery item and its quantity. This class contains two constructors, a getter and setter for the quantity, a getter for the

description, and some utility methods (for checking equality and converting to string for printing). Note that two `GroceryItem` objects are considered "equal" if they have the same description, even if their quantity is different! You should *not* modify this class.

The `GroceriesInterface` interface describes a collection of `GroceryItem` objects. Its intention is to be used similarly to a shopping list, although it is not a truly a *list*; it is to be backed by the `Set` class, so it is unordered and does not permit duplicates. It declares abstract methods for adding and removing (i.e., increasing or decreasing quantity for) `GroceryItem` objects in the collection. It also has a method for overwriting the quantity on a `GroceryItem`, as opposed to strictly increasing or decreasing it. Finally, it has a method to print all items in the collection.

The `GroceriesExample` class (not in the `cs445.a1` package) shows one example usage of the `Groceries` data structure. It demonstrates a few features, but is not a thorough test.

3 Tasks

3.1 Implement Set, 65 points

Develop the generic class, `Set<E>`, a **dynamic-capacity array-based** implementation of the Set ADT described in `SetInterface<E>`. Include this class in package `cs445.a1`. Read the comments in the interface (and this document!) carefully to ensure you implement it properly; it will be graded using a client that assumes *all* of the functionality described in the `SetInterface`, not just the behavior you need to implement `GroceriesInterface`!

You must include a constructor `public Set(int capacity)` that initializes the array to the specified initial capacity, and a constructor `public Set()` that uses a reasonable default initial capacity. Finally, you should provide a constructor `public Set(E[] entries)` that initializes the set to include each of the provided entries. Note that this constructor must still create its own backing array, and **not** adopt the argument as a data member; it must also skip all duplicates and `null` values in the provided array, to satisfy the data structure's usual assumptions. Whenever the capacity is reached, the array must resize, using the techniques discussed in lecture (i.e., you should *never* throw `SetFullException`).

Method	Points
`Set()`	4
`Set(int)`	4
`Set(E[])`	9
`int getSize()`	4
`boolean isEmpty()`	4
`boolean add(E)`	7
`E remove(E)`	7
`E remove()`	6
`void clear()`	5
`boolean contains(E)`	6
`Object[] toArray()`	9

3.2 Implement Groceries, 35 points

Develop the `Groceries` class, an implementation of the ADT described in `GroceriesInterface`. Include this class in package `cs445.a1`. Read the interface carefully (including comments) to ensure you implement it properly. As with `Set`, it will be graded using a client that expects the full functionality described in its interface. The `Groceries` class can be seen as a client of the `Set` data structure. Use *composition* with your `Set<E>` class to store the items as a data member of type `Set<GroceryItem>`. Include a constructor `public Groceries()` that initializes the set.

Method	Points
`Groceries()`	3
`void addItem(GroceryItem)`	7
`void removeItem(GroceryItem)`	7
`int modifyQuantity(GroceryItem)`	9
`void printAll()`	9

3.3 Testing

`GroceriesExample` is provided as an example client of the `Groceries` class. It *does not* exhaustively test the functionality of both classes you must write. You are responsible for ensuring your implementations work properly in all cases, even those not tested by `GroceriesExample`, and follow the ADTs described in the provided interfaces. Thus, it is highly recommended that you write additional test client code to test all of the corner cases described in the interfaces. For help getting started, re-read the section of the textbook starting at Chapter 2.16.

> **Note:** For functionality that cannot be tested (e.g., methods that crash, cannot be compiled), up to 1/2 points will be awarded by inspection. At this level, turning in code that crashes or does not compile is not acceptable and will not yield success.

4 Submission

Upload your java files in the provided Box directory. If you accidentally overwrite the provided interfaces or `GroceryItem` class, remember to restore them to their original versions. You may not make changes to the functionality in these files.

All programs will be tested on the command line, so if you use an IDE to develop your program, you must export the java files from the IDE and ensure that they compile and run on the command line. Do not submit the IDE's project files. Your TA should be able to download your `cs445-a1-abc123` directory from Box, and compile and run your code as discussed in Lab 1. For instance, `javac cs445/a1/Groceries.java` should compile your `Groceries` class. In addition, `javac GroceriesExample.java` and `java GroceriesExample` must compile and run `GroceriesExample`.

In addition to your code, you may wish to include a `README.txt` file that describes features of your program that are not working as expected, to assist the TA in grading the portions that do work as expected.

Your project is due at 11:59 PM on Monday, September 30. You should upload your progress frequently, even far in advance of this deadline: **No late submissions will be accepted.**