Lab – Arrays (Review)

AP Computer Science with Data Structures

Design for this lab should take about 60 minutes. Implementation and testing should take about 90 minutes.

Background: The Locker Problem

Suppose that there are N lockers and M students. Initially, all of the lockers are closed. The first student changes that state of all of the lockers, opening each. The second student begins with the second locker and changes the state of every other locker. The third student begins with the third locker and changes the state of every third locker. Finally, the M^{th} student begins with the M^{th} locker and changes the state of every M^{th} locker. We can now ask a number of questions about the lockers, such as:

1. What is the state of each locker (open or closed)?
2. Which locker had the most state changes?
3. How many times did locker k change state?
4. If student j is absent, what is the state of the lockers? What if there is a list of absent students?
5. If $0.25 is placed into a locker whenever it changes state, which locker has the most money and how much money does it have?

All of these questions can be answered analytically. You, however, are not allowed to use an analytic solution. Instead, you must design and implement a simulation.

Exercise: Simulation Design

The first step is to design the simulation program. You must adhere to the following guidelines:

1. You must have a `main` method and it should only launch the simulation. It must be in a class called `Main`.
2. You must use at least one array.
3. You may not use any other data structure.
4. Your simulation must be capable of answering all 5 questions above without altering the code.

You should consider creating an abstraction (a java class) to represent the lockers, as well as a class to hold the simulation code.

Complete your design, using your notebook to document your design as it progresses. Identify what state variables you will need and what classes you will use. Identify the services each class will provide. Turn in your completed design to Athena2 using the format described in the appendix.

You must also develop a test plan that will ensure that your simulation is correct.

Exercise: Implement your Simulator

Implement and test your simulator. To complete this lab you must:

1. Show your teacher your simulation and demonstrate that it works. You must show tests that follow your test plan and the results of your testing.
2. Turn in all of your code as a single zip file to Athena2. Turn in only the .java files.

Appendix: Design Document Format

Here is the format for a design document that specifies a single object (class). For this project, you will need to document several objects and their interactions. Include a document that describes the entire project.

As you complete the design document, keep in mind that your objective is to create a document that would allow someone else to implement your design without any other help. Your object document will usually be one to two pages in length, depending on the number of services your object makes available. Your overall design document will be longer, depending on the number of objects in your project.

Structure of the Design Document

<Name of Object> Design

Description

Describe what this object does. Tell the reader why they would want to use your object. This is usually a single, succinct paragraph. Spelling and grammar count!

Services

Describe the services (public methods) that this object exposes. Here you need to specify run time requirements, pre and post conditions you have identified in the process of the design, and any other considerations such as state variables affected. Some of the method documentation is likely to come directly from this section.

Internal Data Structures and State

This section describes any internal data structures such as arrays or other objects. You must describe their purpose, how they are initialized and how they are used. Additionally, any state variables needed by your object are described in this section.

Example Design Document

JavaTown LinkedList Design

Description

The `LinkedList` class defines a singly linked list of objects, with each object contained in a `ListNode` object. The `LinkedList` object maintains a pointer to the first element in the linked list. The size of the list is maintained as a state variable so that the size can be queried in O(1) time. All operations, with the exception of initialization and getting the size of the list are implemented recursively.

Services

The following services are available:

`LinkedList()` – constructs a `LinkedList` object and initializes the internal data structures and state. Upon construction, the size of the `LinkedList` is zero. No `ListNOde` objects are created.

`int size()` – returns the size of this list in O(1) time.

`void add(Object)` – adds a new `ListNode` to the beginning of the linked list and sets the data value of the added `ListNode` to the `Object` passed as a parameter in O(1) time. The size of the linked list is incremented by 1.

`Object remove(int index)` – removes the `ListNode` at the position in the linked list specified by the parameter. Note that 0 <= `index` < `size()`. The value contained within the removed `ListNode` is returned and the size of this `LinkedList` is decremented by one. The `remove` operation is O(n) where n is the list size.

`Object get(int index)` – returns the value contained within the `ListNode` at position `index`; 0<= `index` <= `size()`. The `get` service runs in O(n) time where n is the size of the list.

`void set(int index, Object value)` – sets the value in the `ListNode` at position `index`; 0 <= `index` < `size()`. The `set` service runs in O(n) time where n is the size of the list.

Internal Data Structures and State

The `LinkedList` class contains a pointer to a `ListNode` object. The linked list consists of a sequence of `ListNode` objects linked via pointers internal to the `ListNode`.

An `int` state variable is maintained in order to make the `size()` method run in O(1) time.