3/27/2019 Notes

* An abstract data type (ADT) is a model for data types
* A data type is defined by its user of the data, values, operations on data of this type, behavior, etc.
* A data structure is a way of organizing data
* Data structures can implement one or more articular abstract data types
* Interfaces control the structure of subclasses
  + Final keyword in an interface is not required because there are no variables in instances
* FullTimeEmployee class is on Chapter 1 Page 33
* When you create objects, the variable isn’t an Object, it is a reference to an Object. Referenced by addresses
* Using the default constructor for String, the string will be “”, not NULL like most objects
* String will return -1 if the subindex does not exist
* String s = new String();
* String t = new String(“Aloha”);
* System.out.println(t.indexOf(“ha”));
  + 3
* System.out.println(t.indexOf(“a”));
  + 4
* System.out.println(s.indexOf(“ha”));
  + -1
* String w = null;
* This does not make an empty string. This will just save a null object
* System.out.println(w.indexOf(“ha”));
  + Exception in thread “main” NullPointerException
* equals() tests comparison of objects
* The == operator tests for equality of references
  + int and boolean only really work
* String s = new String();
* String t = new String(“Aloha”);
* String w = null;
* String z= new String(“Aloha”);
* s.equals(“”);
  + true
* s == “”
  + false
* t.equals(“Aloha”);
  + true
* t == “Aloha”
  + false
* t.equals(null);
  + false
* t.equals(z);
  + true
* t == z
  + false
* w.equals(null)
  + NullPointerException
* w == null
  + true
* Scanners
  + Scan keyboard input, file input, or a given string value
* Arrays
  + Arrays start from zero lol
* Arguments and Parameters
  + Arguments are the data
    - All arguments are sent by value, even if they are references to Objects
  + Parameters are what is in the header
* Review chapter 0 of the book for dumb basic CS110-111 stuff

3/28/2019

* The point of Data Abstraction is so that the users code should not access the implementation of details of the class used.
* Lots of interfaces
  + No definitions in an interface. Functions end with a semicolon not {}
* Inheritance exists
  + Subclasses and Superclasses – look at cs111 notes
  + Overriding and overloading and stuff

4/1/2019

* JUnit Testing
* Can’t bring the required inputs into your main methods. You need to do Junit testing in a special file
* For example, lets use HourlyEmployee.
  + In a test method, you need to instantiate an HourlyEmployee class with values, then add a ton of assertEquals() where you compare one value to the return of something from the instance
* Testing can reveal the presence of errors but not the absence of errors
* A methods tests should be developed before the method is defined
* In general, methods should be designed to facilitate testing
* Make sure that any boundary conditions are thoroughly tested
* Good testing requires great skepticism
* Testing will help guarantee that your code works well and wont fail during runtime

4/11/19

* Java Collections Framework (JCF)
  + Be able to create and manipulate parameterized collections
  + Compare the ArrayList and LinkedList implementations of the List interface
  + Discuss the differences between a list and a set, and a set and a map
  + Review boxing/unboxing and the enhanced for statement
* A collection is an object that is composed of individual elements
  + For example, an array is a collection
* An array is a random-access storage structure: any element can be accessed immediately
* Drawbacks to an array
  + Size is not dynamic
    - Too big and you waste space
    - Too small and you have to move everything from one array to another
  + Can’t have different types of objects in your array
  + Array accessing is always Big-O linear at least

4/15/2019

* Two different kinds of Collection classes
  + Contiguous
    - ArrayList
    - Stack
    - PriorityQueue
    - ArrayCollection
  + Linked
    - Stack
    - Queue
    - Associative arrays
      * Map
      * Symbol table
      * Dictionary
    - S-Expressions
      * Nested list
      * Tree structure
* ArrayList
  + Can be a list of any type
    - Integer, String, ArrayList<>, etc
  + Generic = parameterized
  + Automatically expands or contracts
  + Number of elements is list.size()
  + Add by
    - list.add(new Double(1234.5));
    - list.add(30, new Double(1234.5));
      * This adds a new double at index 30 instead of just the end
  + Boxing is when an automatic conversion of a primitive to an abstract data type happens
    - int to Integer
  + list.get(int index) returns element at index
* LinkedList
  + When you add an element, you need to include a reference to the next element