Java Review:

- **public** - Any class/function may access the method/property.
- **protected** - Only this class and any subclasses may access the method/property.
- **private** - Only this class may access the method/property. It won't even be inherited.
- How to create a new object
- Know syntax of System.arraycopy()
- What does a public boolean method expect as return?
- What about for an int method, double method, void method, String method, E method?
- How to throw an exception.
- Private classes (are usually inner class of another class) are only accessible to the class that wraps around it (outer class)

ADT

- What are ADT?
- What is the difference between an Abstract Data Type and and Abstract Class

Interfaces:

- Interfaces provide a contract that an implementing class should follow
    - Anything that implements an interface must provide all of it methods
- Do you implement the methods inside of the interface?

Abstact Class & Inheritance:

- Does a class that implements an interface inherit all of the interface's methods?
- Does a class that implements an interface provide (implements) all of the interface's methods?
- If class A extends class B, does A inherit all of B's methods?
- If class A extends class B, does A provide all of B's methods?
- If class A extends class B, this means that B is the superclass of A. A is a subclass of B and anything that extends A is also a subclass of B.
- Abstract classes
    - Define methods which can be used by the inheriting subclass.
    - Define abstract methods which the inheriting subclass must implement.
    - Provide a common interface which allows the subclass to be interchanged with all other subclasses.

Invariants:

- Know the definition of invariants

Lists

- variable sized collections of objects in a particular sequence
- parametrized on the type of the objects that are stored in the list
- duplicate and null elements are permitted
- Look at the wordpress site for useful methods that the List interface would use.

ArrayList

- A list made by using array
- Your array's length could be larger than the amount of elements in it.
    - To implement the ArrayList size() method, you need to keep track of how much items are in the list and that is the size of the list.
- Once your list fills up the array, you need to do copy the list contents from the array to a bigger array. Thus adding to the end of the array could be O(n)
- Adding or removing at the beginning or middle of the list is O(n) because you need to shift your elements.
- Accessing an element at a specific index is O(1), Eg: array [1]

LinkedList

- A list that is made up of nodes and the nodes are linked to each other forming the list
- If you list should have 5 elements, you should have 5 nodes
- A node will have a minimum of two elements that make it up. The first is the reference to it data (item, value, data) and a reference to the next node in the list (always called *next*)
- The end of a singly Linked List is called the tail and the tail's next will always be null.
- The start of a Linked List is called the head and the list must store this head to access the list
- Adding or removing the head is O(n)
- Adding or removing the tail is O(1) if you have a tail pointer, otherwise it would be O(n) because you need to traverse your list until you reach the end.
- Adding or removing from the middle of the list is O(n) because you need to traverse the list.
- Accessing an element at an index is O(n) because you need to traverse your list.
- In a circular Linked List, the tail's next is pointing to the head
- In a doubly Linked List, each node has a next and previous reference (pointer)
- In a doubly Linked circular list, the head's previous is the tail and the tail's next is the head.
- Know how to add an item in the beginning, middle and end of your list in terms of code and concept.
- Study the doubly-linked list and circular linked list

Big-O

- Usually look at the for loops to get an idea of the run-time for a method.
- If the for loop stops at a constant number such as i<5, this would be a O(1).
- If the for loop stops at something like i<array.length, this would be O(n).
- If you have a run time where it is $n^2 + n + 3 + n^3$, you take the fastest growth rate (n^3) which would be $O(n^3)$.
- in general, $O(1) < O(\log n) < O(\text{sqrt}(n)) < O(n) < O(n^2) < O(n^3) < O(2^n)$
- this also means that $O(n) < O(n \log n) < O(n\ \text{sqrt}(n)) < O(n^2)$

Iterators:

- Traverses through a collection
- Be able to implement the hasNext() and next() method for the iterator class.

References:

- Every Java variable (that is not of a basic type) is a reference to an object, and may be null
- String a = new String (“a”)
- String b = new String (“a”)
    - == works if both a and b are references to the same object or are null. In this case, a and b are not referring to the same object.
    - equals() works if the contents of object a match the contents of object b. In this case, a and b have the same content.
- String a = new String (“a”), String b = new String (“b”), b = a
    - ==, will now work because b is referencing a.
- If you pass in a variable such as an int, string double, etc into a method. The method will not change the value of the item that was passed into it.
    - Example:
    - int a = 3
    - public void addOne(a)
        - adds one to the parameter passed to it.
    - unless a is a class variable this method does nothing to a, a is still 3.
- If you pass a variable into a method, the only way to change the variable’s value is to change what a is referencing.
    - int a = 3
    - public int addOne(a) is the method
    - a = addOne(a)
- The same if you pass in an object (LinkedList x would be an object) into a method that doesn’t access the object to change it values.
- If your method calls on that object’s method such as
    - public void delete(list)
        - list.remove()
    - This will affect the list that was passed into the method