Recitation II: Linked List

Tags	
Date	@February 2, 2024
Files & media	DS recitation 2 annotated.pdf

- References and pass-by-value
    - When we define a class by `Dog d`, this is actually a pointer. To initialize a class, we use the keyword `new`. By `Dog d = new Dog()`, this is assign the address of the newly instantiated class to the pointer `d`.
    - When we pass `d` into a function call, say `setName(Dog dog)`, the *address* is passed to become `dog`. Because the nature of address, if we change the attribute of `dog`, we change the attributes and the class.
    - For more specific clarification, see this blog post.
- Review for Singly Linked List
    - Implementation
    - General guideline for solving linked list problem
        1. draw a graph
        2. figure out which edges need to be modified
        3. think of corner case
- Equivalence testing for linked list (§3.5.2)
    - Because classes are actually references, when comparing two classes in java, it will only be equal if they are the same class. However, this is usually not very useful. We want actual “semantic comparing”. For instance, if two strings have the same characters, I should expect the comparing to tell me they are the same. In order to do this, we need to implement the method “equals(Class o)”.
    - For a linked list, a very natural way to define the semantic comparison is by the element of the linked list. There are some criteria that guarantee

that two linked lists are not semantically equal, for instance, different lengths for linked lists.

- Shallow and deep equal
    - If elements in linked lists are classes, then again, by the nature of reference, comparing classes by “==” will only return true if they have the same address, that is, same object. Sometimes, this behavior is expected: we want the element in the linked list point to the same object. Then this is called “shallow comparison”.
    - In other cases, if the comparison target has a nested structure, we want to actually compare the element in the “most fundamental” (or leaf node) layer. This is called “deep comparison”, where it should handle both when the element is a primitive or an object. (see more in §3.5.1)