

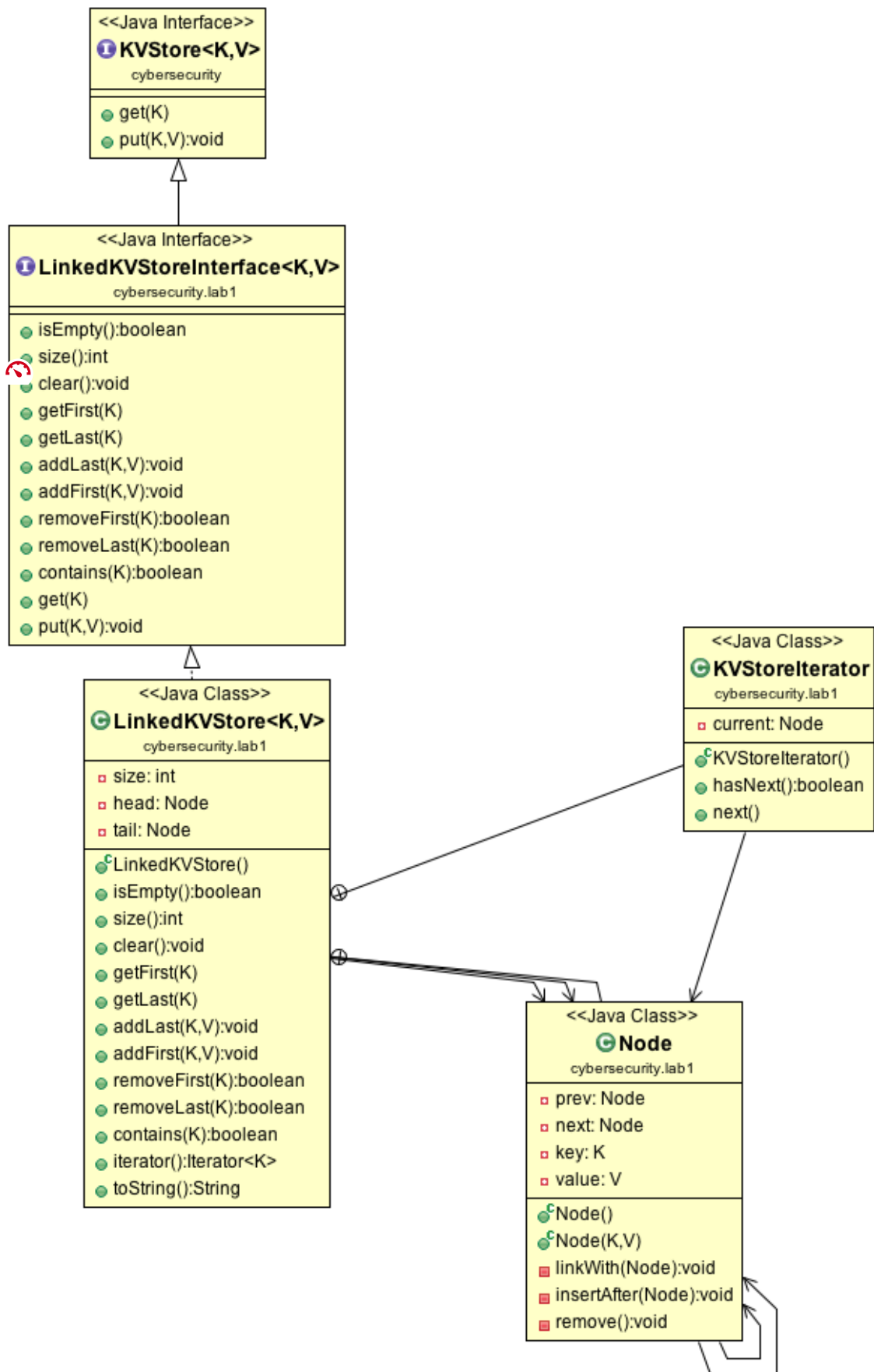
# Lab 11 CyberSecurityLab 01

 Publish **Edit**

## Lab 11 CyberSecurityLab 01

### Learning Objectives

- Get more experience with sentinel nodes, iterators and doubly linked list
- Consider OO design for storing data as key-value pairs
- Explore implications on efficiency and resource management



As we progress in our implementation of data structures we can consider design choices that improve the efficiency of our code and thus the availability of resources on the machine.

Download Skeleton: [Lab11CyberSecurity01.zip](#) 

Study the UML diagram above and the Skeleton code. In `LinkedKVStore.java` you can see how changes to the underlying linked chain are made with the Node methods **`insertAfter(Node curr)`** and **`remove()`**. Notice how many methods use a for loop to iterate through the linked chain, this is a readable variation of how you are accustomed to seeing it implemented with a while loop.

You are given the inner skeletons for the Node and Iterator classes and you need to implement the methods. Once you've implemented the Node constructors, you can follow the videos below to guide you conceptually through the steps for the Node methods. Notice how this Node Class provides functionality to link and unlink nodes from each other so that `LinkedKVStore` member methods don't directly manipulate the Node's field variables. The **`linkWith(Node nextNode)`** is a helper method for both **`insertAfter(Node curr)`** and **`remove()`**. Remember to call **`linkWith()`** and reuse your code! Both **`insertAfter(Node curr)`** and **`remove()`** are responsible for updating the **`size`** field accordingly.

Links this node to parameter node:

Inserts this node after a parameter node:

Removes this node:

Use your pre-lab test to test the methods in `LinkedKVStore.java`. The Iterator's **`next()`** method should throw a `NoSuchElementException` if it is called when there is no next element. Remember this list uses sentinel nodes so the node with the last element in it references tail and not null.

Properly testing those methods will test the inner class methods.

For the implementation of an Iterator used in this lab, the client/test code should get a new iterator any time there is a change to the list. For example: if the client/test code is using an iterator `iter` on a list,

after a call to any of the methods put, get, addLast, addFirst, removeLast or removeFirst, then the code should get a fresh iterator (iter = list.iterator()).

Submit to WebCAT.

**Points** 100

**Submitting** Nothing

Due	For	Available from	Until
-	Everyone	-	-

+ [Rubric](#)