Assignment 6: Generics - Array List & Linked List

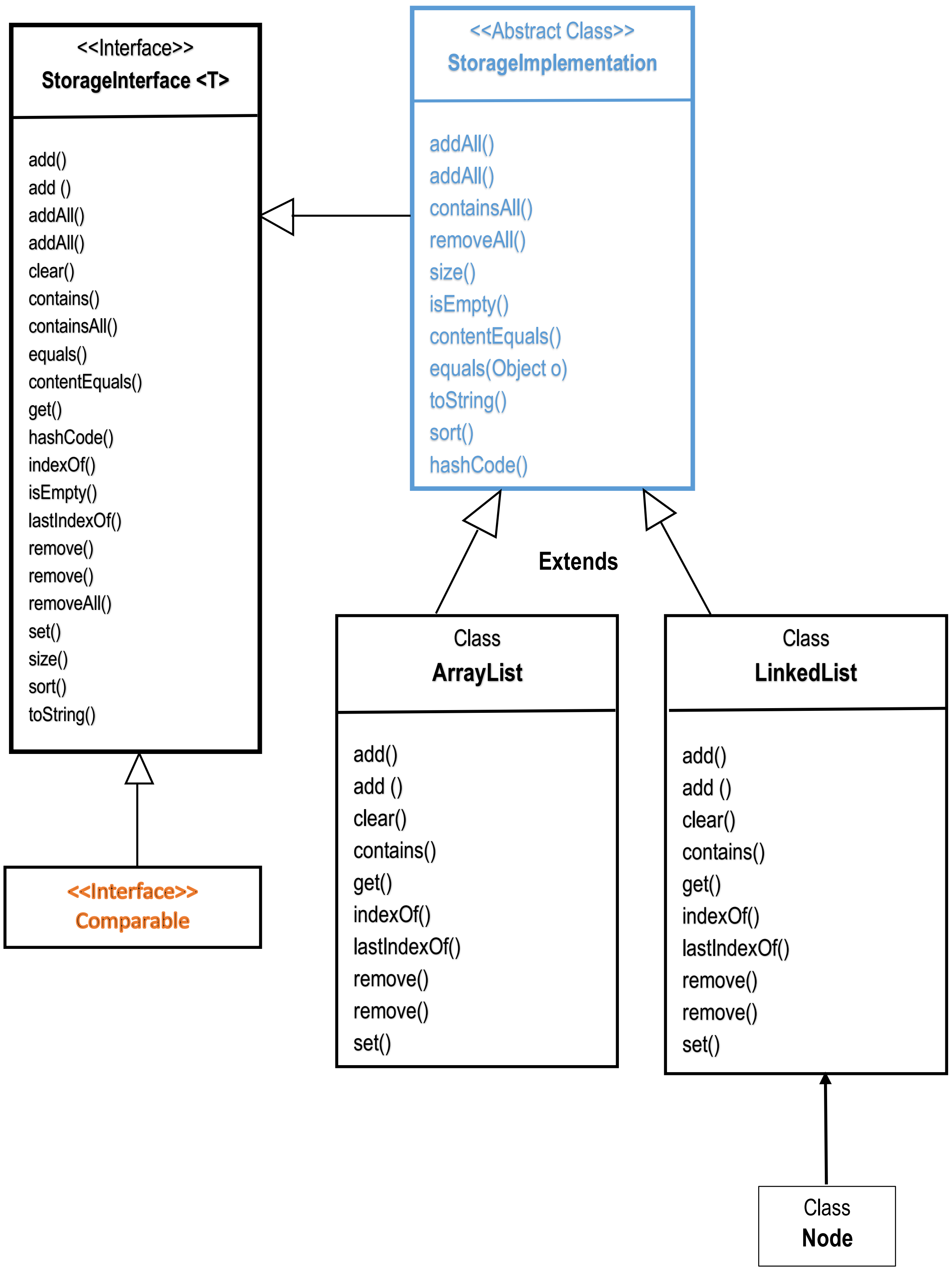
The previous assignment has asked you to implement our own Integer Array List and Linked List, for this week's assignment we had to implement an array list and a linked list which would not only take integers in it but can be dynamic enough to take any other datatype for creation of the lists of that type. Few parts of implementation for the integer array for the previous assignment was not done correctly, the bugs and incorrect application has been corrected. The following gives a brief description of points indicating the fixed elements of the program:

- **Integration**: While implementing both the codes – *IntegerArrayList & IntegerLinkedList* - we ran into problems while merging the logic for the programs I.e. the common methods implemented for both the data structures were not working properly due to the fact that the counter for the linked list was started from 1 and for the arraylist from 0, this problem is solved during this implementation which consequently improved the performance of the common methods like *toString(), contentEquals(), contiansAll()* etc.
- **equals()** – for the previous implementation of both the data structures, our interpretation of equals() was different and which resulted in incorrect implementation n of this method. Previously we were checking for the class names to be equal, for this implementation we have corrected our understanding and implemented the equals methods by checking the memory location of the object which is passed to the method.
- **hashcode()** – the same blunder was done while the initial implementation of the data structures, rather than calculating the sum of the integers which will be the resultant hashcode of the arraylist we had used the *hashcode()* method of the object class. For this implementation we have corrected the method execution and focused on the requirement given to us and have returned the value as the sum of hashcodes all the elements in the lists.
- **removeAll()** – this method was returning a wrong result every time the test cases were ran, we figured this was happening because of the mismatched counters which were used for both the files differently, once we corrected this problem the *removeAll()* method worked as required.

For this implementation there are few methods like the sort where the comparable interfaces' *compareTo()* method has being used to sort the elements in the list. This is done because we can create lists of different data types and comparing different datatypes requires different implementation. The *comaperTo()* method compares object according to their natural ordering, hence if a String list is created the sorting function lexicographically sorts all the elements in the list.

This assignment being an extension of the previous work, we took the opportunity to rectify all the wrong implementations and have tried to perfect all the application of each method taking into consideration the requirement of the problem statement.

Additionally, we have taken assistance from the previous version of test and modifies it by passing different datatypes for checking the correctness of the program



<<Interface>>
StorageInterface <T>
add()
add ()
addAll()
addAll()
clear()
contains()
containsAll()
equals()
contentEquals()
get()
hashCode()
indexOf()
isEmpty()
lastIndexOf()
remove()
remove()
removeAll()
set()
size()
sort()
toString()
<<Abstract Class>>
StorageImplementation
addAll()
addAll()
containsAll()
removeAll()
size()
isEmpty()
contentEquals()
equals(Object o)
toString()
sort()
hashCode()
Extends
Class
ArrayList
add()
add ()
clear()
contains()
get()
indexOf()
lastIndexOf()
remove()
remove()
set()
Class
LinkedList
add()
add ()
clear()
contains()
get()
indexOf()
lastIndexOf()
remove()
remove()
set()
<<Interface>>
Comparable
Class
Node