**CSCI 3302 Programming Assignment 05 (100 Points)**

**Due: Nov 10, 8:00 AM**

GITHUB Link: [Program 05](https://classroom.github.com/a/9Ruygn5a)

Objectives:

* Demonstrate how to define and implement an abstract data type.
* Demonstrate a basic understanding of object-oriented programming.
* Demonstrate understanding of an ADT List.
* Demonstrate Java programming proficiency for a linked list.
* Demonstrate proficiency with Java generics.

Assignment Assistance:

* This homework assignment is due prior to the date and time specified above.
* This assignment is restricted to individual effort. You may not receive help from any other person except the instructor or the AARC (help from the AARC must be well documented!).
* Any resource used (other than Dr. Becnel or the course text) must be documented in the code (as comments) detailing the source and describing exactly what was learned and how that information was used. Submissions will be severely penalized if copied in part or in whole from any source.
* If you need help, visit your instructor during his posted office hours. If your schedule cannot accommodate any of these times, then email your instructor to schedule a different time.

Problem Description:

1. Over the past weeks, we have been discussing reference-based data structures (i.e., linked lists), and we have seen reference-based implementations of the List ADT, Stack ADT, and Queue ADT. In this assignment, you will write a reference-based implementation of the Dictionary ADT.
   1. A Dictionary ADT (also called an associative array) is an abstract data type composed of a collection of (key, value) pairs, such that each possible key appears at most once in the collection.
   2. Operations associated with this data type allow:
      1. the addition of a (key, value) pair to the collection;
      2. replacement of a (key, value) pair within the collection for a given key;
      3. the removal of a (key, value) pair from the collection; and
      4. the lookup of a value associated with a given key
2. You will implement a class called Dictionary<S, T> that stores (key, value) pairs in a reference-based list.
   1. You will write the Dictionary<S, T> class in a file called Dictionary.java.
   2. Your class should contain a single attribute called head of type Node<S, T> and is a reference to the first Node object in the dictionary (null on empty list).
   3. Your class should implement the IDictionary<S, T> interface found in Appendix A. The provided code should not be modified.
   4. Each (key, value) pair is stored in an object of type Node<S, T>. The Node class is provided in Appendix B of this document. The provided code should not be modified.
   5. The reference-based list is used to store (key, value) pairs does not need to be ordered; however, it should not contain any two (key, value) pairs that have the same key (even if the associated values differ).
   6. Recall that generics allow the client code to specify the types used. The IDictionary<S, T> interface, Node<S, T> class and your implementation of the Dictionary<S, T> class all make use of two generic types, S and T, which means that the type associated with the key attribute does not need to be the same type as that associated with the value attribute. It has probably been a while since you looked at Java generics. Feel free to reference your CSC 2302 textbook to review.
3. To make sure that your implementation is working correctly, you need to create an DictionaryException class as an extension of a RuntimeException. This exception needs to be thrown if any attempt is made to access any (key, value) pair not contained in the dictionary (key is not found). Create a file called DictionaryException.java containing the following code:

**public class** DictionaryException **extends** RuntimeException

{

**public** DictionaryException(**String** s)

{

**super**(s);

}

}

1. Your submission should compile with no compiler errors or warnings. Compiler warnings indicate an incorrect implementation. Your code should also not contain any variables explicitly specified as type Object. Generics are meant to provide the flexibility of Object collections but with the safety of compile-time type checking.
2. Your submission should NOT contain a main method or any extraneous testing code. If you wish to include non-working code for insight into your thought process, make sure to contain it within comment blocks and ensure that submission successfully compiles.
3. For testing and debugging purposes, it is always good to have a toString() method for your classes. Here is some code for one that tells you basic information about the keys and their values.  
   A screen shot of a computer code

   Description automatically generated with low confidence
4. Your program should work in the GitHub codespace (Linux environment) and locally (Windows environment).
5. You may write any private helper methods if needed.

Submission:

* Review the Evaluation below to ensure you have met all the requirements.
* Submit electronic copies of Dictionary.java, IDictionary.java, Node.java, DictionaryException.java to GitHub. Upload a backup copy to D2L. These files should NOT contain a main method or any extraneous testing code. You may include test files in your repository; however, these will not be considered when grading. If you wish to include non-working code for insight into your thought process, make sure to contain it within comment blocks and ensure that submission successfully compiles.

**Evaluation**

|  |  |
| --- | --- |
| **Automatic Deductions:** |  |
| Late/Not Submitted | -100 |
| Code not submitted to GitHub | -30 |
| Code does not run/compile | -50 |
|  |  |
| **Earn Points for the following:** |  |
| Code has comment header with name, section, date | 5 pts |
| Code organization, structure, and indention is appropriate (SHFT + ALT + F in VS Code) | 5 pts |
| Code is well and meaningfully commented. | 5 pts |
| Appropriate variable and method names that follow Java conventions | 5 pts |
| Instructions correctly followed for fields, class, methods | 10 pts |
| isEmpty and clearDictionary | 10 pts |
| add, remove, get | 20 pts each |

**Appendix A: The interface used for this assignment:**

**public interface** IDictionary**<S, T>**

{

**// Determines whether the dictionary is empty.**

**public** **boolean** isEmpty();

**// Adds the provided (key, item) pair to the dictionary.**

**// If the key is already associated with a value, the**

**// old key, value pair are replaced.**

**public** **void** add(S key, T value);

**// Removes the (key, value) pair specified by the given**

**// key from the dictionary. Throws an exception if the**

**// (key, value) pair is not contained in the dictionary.**

**public** **void** remove(S key) throws DictionaryException;

**// Returns the value associated with the given key**

**// from the dictionary. Does not modify the dictionary.**

**// Throws an exception if the (key, value) pair is not**

**// contained in the dictionary.**

**public** **T** getValue(S key) throws DictionaryException;

**// Removes all (key, value) pairs in the dictionary.**

**public** **void** clearDictionary();

}

**Appendix B: The implementation of the Node class used for this assignment:**

**// Node class to store a key of generic type S and**

**// value of generic type T.**

**public class Node<S, T>**

{

**// Attributes**

**private S** key;

**private T** value;

**private Node<S, T>** next;

**// Constructor for a node.**

**// Call setter to assign next.**

**public Node(S newKey, T newValue)**

{

this.key = newKey;

this.value = newValue;

this.next = null;

}

**// Getter for key within the node.**

**public S getKey()**

{

**return** this.key;

}

**// Getter for value within the node.**

**public T getValue()**

{

**return** this.value;

}

**// Getter for the reference to the next node in the**

**// list; if this is the last item, then next is null.**

**public Node<S, T> getNext()**

{

**return** this.next;

}

**// Setter for the reference to the next node.**

**public void setNext(Node<S, T> newNext)**

{

this.next = newNext;

}

}