Lab 3: Abstract Data Types

Background

When writing the specifications for an ADT (Abstract Data Type), only answer what the methods will do, and not how they will do it. Once an ADT has been defined, our jobs as the programmer is to create an implementation that follows this contract so that it will function exactly as promised.

There are 2 parts to this lab with 2 different submission requirements and dates.

Problem Description

Consider the following requirements for the DictionaryADT<K,V>: (also known as maps or associative arrays)

- This data type must be able to hold data in (key, value) pairs where every key is unique.
- It must be possible to insert a (key, value) pair into the dictionary.
- It must be possible to remove a pair in the dictionary based on the key.
- It must be possible to update the value of an existing key.
- We also require the ability to lookup a value based on the key.
- An example of a similar ADT can be found in java.util.Map.java.

Instructions

Part 1:

1. This lab can be completed individually or in a group (there is no size limit for the group), write a possible contract (specification) as a Java interface for the DictionaryADT using the Lab3StartingCode project in Lab 3: Abstract Data Types – Part 1.
    - If working in a group, only one submission is required – **all group members' names must be included in the comments of the submission.**
        - All group members will receive the same grade.
        - It is your responsibility to manage the communication, cooperation and contribution of all group members.
2. See the *Marking Criteria* section below for details on how you will be assessed.
    - If there are more than one submission, both individually or across all group members, only the latest submission before the deadline will be accepted.

3. Your contract must include the required functionalities as defined in the problem description as methods stubs with **explicitly stated pre-conditions and post-conditions, with proper Javadoc notations for all parameters, return values and expected exceptions**.
    - The specification should be documented so that the person who receives it could write an implementation without any extra information.
4. Submit your DictionaryADT.java to Brightspace → Lab 3: Abstract Data Types – Part 1 by the posted due date.

Part 2:

5. Download the files from Lab 3: Abstract Data Types – Part 2 and place them in the **correct** folders in your Lab3StartingCode Eclipse project.
    - DictionaryADT.java should be in the “utilities” package.

      Note: This DictionaryADT will replace the one you have done in the previous steps, make sure you rename or save your ADT somewhere else if you wish to keep it!
    - DuplicateKeyException.java should be in the “exceptions” package.
    - DictionaryUnitTest.java should be in the “testDictionary” package in the test folder.
6. Implement the corresponding methods in Dictionary.java using the DictionaryADT from the previous step.
    - Your implementation should only use the ArrayLists to store the key value pairs.
7. Verify the correct functionality of your Dictionary implementation using the set of JUnit tests.
    - The unit tests have been designed against the ADT, i.e. the functionalities specified by the requirements.
    - Practice the Test-Driven Development approach:
        - Choose one method and one test, write just enough code to pass the test!
            - Start with setup() and create() methods so the objects are constructed correctly.
            - You’ll need the ability to insert() before you can remove(), update() or lookup()!
        - Repeat until all tests have passed.

    Note: Do NOT modify the tests provided in any way! Your instructor will test with the default test file.
8. Submit your completed **zipped** exported Eclipse project to Brightspace → Lab 3: Abstract Data Types – Part 2 by the posted due date.
    - If working in a group, only one submission is required – all group members’ names must be included in the comment.

Marking Criteria

Criteria	**Missing** (0%)	**Needs Improvement** (0-50%)	**Good** (51-75%)	**Excellent** (76-100%)	**Marks**
Part 1: ADT	Not submitted	Significant methods and/or documentation are missing	Not all methods and/or documentation included	Most/all methods and documentation are included.	**/10**
Part 2: Implementation	Not submitted	Significant implementation details are missing and/or not functional.	Not all implementation details are included and/or not functional.	Most/All implementation details are included and functional.	**/10**
				Total	**/20**