Programming Assignment #4

125 Points

Store Checkout System

For this assignment, you will create a portion of a retail store checkout system. For this assignment you have both data structures and an application to develop. The application provides fast lookup for product information. A clerk (or customer) inputs a product stock number and the system returns the record for that item. The system is designed to provide facilities to add, remove and look up items, along with certain other utility functions.

- Your project will use the **`DictionaryADT`** interface (provided) to support your application program. The dictionary takes key-value pairs.
- Each key must be distinct; no duplicates are allowed. There may be duplicate values.

Due Date/Time:

Your project is due on Friday, May 4th at the 3:30 p.m. IMPORTANT, for this assignment there is no physical hard copy to turn in. **Do not submit any printouts for this assignment**. Put your files in your `handin/prog4/` folder to submit your assignment.

The May 4th deadline is final, and **no late programs will be accepted** as we are at the end of the semester.

The Project:

This program consists of the ProductLookup application program, which allows the user to interact with the dictionary. You will implement the `DictionaryADT` interface in three ways:

- A Hashtable
- A BinarySearchTree
- A Red/Black Tree

Your project will consist of the following files:

`ProductLookup.java`	The application program (methods below).
`StockItem.java`	The items that are stored in the dictionary.
`DictionaryADT.java`	The Dictionary interface. (Provided)
`Hashtable.java`	A hash table implementation of the DictionaryADT interface. Using chaining.
`BinarySearchTree.java`	The BST implementation of the DictionaryADT.
`BalancedTree.java`	A red/black tree implementation of the DictionaryADT, which uses the Java API class `java.util.TreeMap`.

You will write all of the above classes with the exception that the two interfaces are provided.

You may not make any modifications the Dictionary ADT interface; the instructor's copy of this file will be used to grade your project.

All of the above classes **except** `ProductLookup.java` and `StockItem.java` must be in a package named `data_structures`.

ProductLookup.java

```
import data_structures.*;
import java.util.Iterator;

public class ProductLookup {

    // Constructor.  There is no argument-less constructor, or default size
    public ProductLookup(int maxSize)

    // Adds a new StockItem to the dictionary
    public void addItem(String SKU, StockItem item)

    // Returns the StockItem associated with the given SKU, if it is
    // in the ProductLookup, null if it is not.
    public StockItem getItem(String SKU)

    // Returns the retail price associated with the given SKU value.
    // -.01 if the item is not in the dictionary
    public float getRetail(String SKU)

    // Returns the cost price associated with the given SKU value.
    // -.01 if the item is not in the dictionary
    public float getCost(String SKU)

    // Returns the description of the item, null if not in the dictionary.
    public String getDescription(String SKU)

    // Deletes the StockItem associated with the SKU if it is
    // in the ProductLookup.  Returns true if it was found and
    // deleted, otherwise false.
    public boolean deleteItem(String SKU)

    // Prints a directory of all StockItems with their associated
    // price, in sorted order (ordered by SKU).
    public void printAll()

    // Prints a directory of all StockItems from the given vendor,
    // in sorted order (ordered by SKU).
    public void print(String vendor)
```

```
    // An iterator of the SKU keys.
    public Iterator<String> keys()

    // An iterator of the StockItem values.
    public Iterator<StockItem> values()
}
```

StockItem.java

```
import java.util.Iterator;
import data_structures.*;

public class StockItem implements Comparable<StockItem> {
    String SKU;
    String description;
    String vendor;
    float cost;
    float retail;

    // Constructor.  Creates a new StockItem instance.
    public StockItem(String SKU, String description, String vendor,
                     float cost, float retail)

    // Follows the specifications of the Comparable Interface.
    // The SKU is always used for comparisons, in dictionary order.
    public int compareTo(StockItem n)

    // Returns an int representing the hashCode of the SKU.
    public int hashCode()

    // standard get methods
    public string getDescription()

    public String getVendor()

    public float getCost()

    public float getRetail()

    // All fields in one line, in order
    public String toString()
}
```

DictionaryADT

```
/*  DictionaryADT.java
    Dictionary interface.
*/

package data_structures;

import java.util.Iterator;
import java.util.NoSuchElementException;

public interface DictionaryADT<K extends Comparable<K>,V> {

      // Returns true if the dictionary has an object identified by
      // key in it, otherwise false.
      public boolean contains(K key);

      // Adds the given key/value pair to the dictionary.  Returns
      // false if the dictionary is full, or if the key is a duplicate.
      // Returns true if addition succeeded.
      public boolean add(K key, V value);

      // Deletes the key/value pair identified by the key parameter.
      // Returns true if the key/value pair was found and removed,
      // otherwise false.
      public boolean delete(K key);

      // Returns the value associated with the parameter key.  Returns
      // null if the key is not found or the dictionary is empty.
      public V getValue(K key);

      // Returns the key associated with the parameter value.  Returns
      // null if the value is not found in the dictionary.  If more
      // than one key exists that matches the given value, returns the
      // first one found.
      public K getKey(V value);

      // Returns the number of key/value pairs currently stored
      // in the dictionary
      public int size();

      // Returns true if the dictionary is at max capacity
      public boolean isFull();

      // Returns true if the dictionary is empty
      public boolean isEmpty();

      // Returns the Dictionary object to an empty state.
      public void clear();

      // Returns an Iterator of the keys in the dictionary, in ascending
```

```
    // sorted order.  The iterator must be fail-fast.
    public Iterator<K> keys();

    // Returns an Iterator of the values in the dictionary.  The
    // order of the values must match the order of the keys.
    // The iterator must be fail-fast.
    public Iterator<V> values();
}
```

Additional Details:

- Your project will consist of exactly the files/classes named in this assignment. You may not have any additional classes or public methods. (Inner classes and private methods are OK).
- The ProductLookup class must have a variable of type DictionaryADT<K,V> which will be instantiated using your Hashtable class. You must insure that your ProductLookup class works with all three implementations, **but turn it in hardcoded to use the Hashtable class**. Your ProductLookup class, and all three implementations will be tested with driver programs not available to you.
- For the BinarySearchTree and Hashtable implementations, you must write your own code; you may import only `java.util.Iterator`, `java.util.NoSuchElementException`, and `java.util.ConcurrentModificationException`;
- Your DictionaryADT implementation methods should be as efficient as possible.
- Your iterators must be fail-fast.
- For the red/black tree implementation, you should use the Java API class `java.util.TreeMap`. For this implementation only, you may use any classes in the Java API.
- **Each source code file should begin with your name and class account number (commented out, of course). This is a gradable issue.**
- Your class names and methods and method signatures must match the specifications exactly.
- IMPORTANT!!! As the project is due the last week of classes, there will be NO opportunity for resubmission. **Projects that fail to compile or run on will receive very little or no credit**.

Project Submission:

You will NOT submit any printout of the source code files you have written, as you will not be getting them back.

All your files must be placed in your `handin/prog4/` subdirectory for your project to be graded. Some of your source code files will have been in a package. You must not recreate the package directory structure in your `handin/` subdirectory. When turning your files in for grading, they all go in the same place, the `handin/prog4/` subdirectory. Please be sure to follow all specifications carefully. As the end of the semester is near, **no resubmissions can be accepted.**

Late Program:

As we are at the end of the semester, late programs cannot and will not be accepted.

Cheating Policy

There is a zero tolerance policy on cheating in this course. You are expected to complete all programming assignments on your own. Collaboration with other students in the course is not permitted. You may discuss ideas or solutions in general terms with other students, but you must not exchange code. During the grading process I will examine your code carefully. Anyone caught cheating on a programming assignment (or on an exam) will receive an "F" in the course, and a referral to Judicial Procedures.