CS310 Fall 2015

Data Structures

Programming Assignment #4

150 Points

A Phone Book

For this assignment, you will create a Phone Book utility. The Phone Book stores Phone Numbers and the Name associated with each number in a Dictionary. The Phone Book is designed to provide facilities to add, remove and look up Phone Numbers, 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. The **phone number** is the key and the **name** is the value.

Due Date/Time:

Your project is due on Tuesday, December 8th at the beginning of class. **The due date is final, and no late programs will be accepted as we are at the end of the semester.** 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 `handin/prog4/` to submit your assignment.

The Project:

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

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

Your project will consist of the following files:

- `PhoneBook.java` The application program (methods below).
- `PhoneNumber.java` A telephone number instance
- `DictionaryADT.java` The Dictionary interface. (Provided)
- `Hashtable.java` A hash table implementation of the DictionaryADT interface. Using chaining.
- `LinearListADT.java` The linear list ADT.
- `LinearList.java` The linear list implementation from project #2.
- `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 will use the LinearList class from your programming assignment #2 to build your Hashtable.

You may not make any modifications to either of the two interfaces; the instructor's copy of this file will be used to grade your project.

All of the above classes except `PhoneBook.java` and `PhoneNumber.java` must be in a package named `data_structures`.

PhoneBook.java

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

public class PhoneBook {


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

    // Reads PhoneBook data from a text file and loads the data into
    // the PhoneBook.  Data is in the form "key=value" where a phoneNumber
    // is the key and a name in the format "Last, First" is the value.
    public void load(String filename)

    // Returns the name associated with the given PhoneNumber, if it is
    // in the PhoneBook, null if it is not.
    public String numberLookup(PhoneNumber number)

    // Returns the PhoneNumber associated with the given name value.
    // There may be duplicate values, return the first one found.
    // Return null if the name is not in the PhoneBook.
    public PhoneNumber nameLookup(String name)

    // Adds a new PhoneNumber = name pair to the PhoneBook.  All
    // names should be in the form "Last, First".
    // Duplicate entries are *not* allowed.  Return true if the
    // insertion succeeds otherwise false (PhoneBook is full or
    // the new record is a duplicate).  Does not change the datafile on disk.
    public boolean addEntry(PhoneNumber number, String name)

    // Deletes the record associated with the PhoneNumber if it is
    // in the PhoneBook.  Returns true if the number was found and
    // its record deleted, otherwise false.  Does not change the datafile on disk.
    public boolean deleteEntry(PhoneNumber number)

    // Prints a directory of all PhoneNumbers with their associated
    // names, in sorted order (ordered by PhoneNumber).
    public void printAll()

    // Prints all records with the given Area Code in ordered
```

```
    // sorted by PhoneNumber.
    public void printByAreaCode(String code)

    // Prints all of the names in the directory, in sorted order (by name,
    // not by number).  There may be duplicates as these are the values.
    public void printNames()
}
```

PhoneNumber.java

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

public class PhoneNumber implements Comparable<PhoneNumber> {
    String areaCode, prefix, number;
    String phoneNumber;

    // Constructor.  Creates a new PhoneNumber instance.  The parameter
    // is a phone number in the form xxx-xxx-xxxx, which is area code -
    // prefix - number.  The phone number must be validated, and an
    // IllegalArgumentException thrown if it is invalid.
    public PhoneNumber(String n)

    // Follows the specifications of the Comparable Interface.
    public int compareTo(PhoneNumber n)

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

    // Private method to validate the Phone Number.  Should be called
    // from the constructor.
    private void verify(String n)

    // Returns the area code of the Phone Number.
    public String getAreaCode()

    // Returns the prefix of the Phone Number.
    public String getPrefix()

    // Returns the the last four digits of the number.
    public String getNumber()

    // Returns the Phone Number.
    public String toString()
}
```

DictionaryADT

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

```
package data_structures;

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


public interface DictionaryADT<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 PhoneBook class must have a variable of type DictionaryADT<K,V> which will be instantiated using your Hashtable class. You must insure that your PhoneBook class works with all three implementations, but turn it in hardcoded to use the Hashtable class. Your PhoneBook 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 comments, including your name and class account number.
- 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 rohan will receive no credit**.

Project Submission:

All of your files must be placed in your `handin/prog4` subdirectory for your project to be graded. **You must not create a `data_structures` folder within your `handin/prog4` 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 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.