Summer 2011 CSCB63
Assignment 2

Due Date: **<u>July 14th, 11:59 pm.</u>**

Submission Instructions and Notes

- This assignment must be done in teams of 2. If you can't locate a team member – use the A2 forum (on portal) to look for one.

- Submit your assignment through portal. There is a link in the assignments folder for submission.

- Include in your submission a README with the names, ids, and UTORIDs.

- Late assignments is subject to 25% (absolute value) deducted for every day the assignment is late -- to a maximum of three days.

- There will be a 24 hour blackout prior to the due date - the TA is not going to be answering questions on the designated discussion board (http://portal.utoronto.ca) during that time.

- You are required to document your code.

Description

In this assignment, you are going to practice what you have been learning in the course! You are required to develop 3 generic data structures classes in Java: an extensible hashtable, a 2-3-4 tree (*using the top-down approach*) and a splay tree.

Use the following class & interface definitions;

```
//---------------------------------------------------------------

public class Entry {
      // You can add methods to this class but do not
      // change the class name, attribute names or types.
   public Object  _nkey;

   public Object  _objEntry;
}
```

```
public interface Dictionary {
     // inserts an entry into the dictionary
     public abstract void insert(Entry entry)
                                   throws UoTLibException;
     // deletes the entry whose key is objKey. Returns the entry.
     public abstract void Entry delete(Object objKey)
                                   throws UoTLibException;
     // retrieves the entry whose key is objKey
     public abstract Entry getValue(Object objKey)
                                   throws UoTLibException;
     // retrieves the entry whose key has the minimum value
     public abstract void Entry getMinValue( ) throws UoTLibException;
     // retrieves the entry whose key has the maximum value
     public abstract void Entry getMaxValue( ) throws UoTLibException;

}

public interface OrderedDictionary extends Dictionary {

     // traverses the ordered dictionary inorder. Returns the
     // entries visited in a vector – in the same order visited.
     public abstract Vector<Entry> inOrder( )
                                        throws UoTLibException;
     // traverses the ordered dictionary postorder. Returns the
     // entries visited in a vector – in the same order visited.
     public abstract Vector<Entry> postOrder( )
                                        throws UoTLibException;
     // traverses the ordered dictionary preorder. Returns the
     // entries visited in a vector – in the same order visited.
     public abstract Vector<Entry> preOrder( )
                                        throws UoTLibException;
     // traverses the ordered dictionary level by level. Returns the
     // entries visited in a vector – in the same order visited.
     public abstract Vector<Entry> levelOrder( )
                                        throws UoTLibException;
     // returns a list of entries from start to End. Start must be an
     // entry at higher level than end. Returns null vector if no path
     // is found.
     public abstract Vector<Entry> getPath(Entry entryStart,
                                       Entry  entryEnd )
                                        throws UoTLibException;

}

// Exception for handling errors and exceptions from operations.
public class UoTLibException extends Exception {

     public UoTLibException( String strMessage ){
          super( strMessage );
     }
}
```

Required Directory Structure

```
<your-team-name>
        README
        Makefile
        libs       -- junit jar file + other libs you use goes here.
        classes    -- this is where the Makefile should put the .class
        src
                uotlib
                        datastructure
                                        -- your augmented version of Entry.java
                                        Entry.java

                                        -- extensible hashtable implementation
                                        ExTable.java

                                        -- 2-3-4 tree implementation
                                        TwoThreeFourTree.java

                                        -- Splay tree implementation
                                        SplayTree.java

                        tests
                                // junit test class for ExTable.java
                                ExTableTest.java

                                //junit test class for TwoThreeFourTable.java
                                TwoThreeFourTreeTest.java

                                //junit test class for SplayTree.java
                                SplayTreeTest.java
```

What to submit?

Zip the parent directory <your-team-name> and submit the zipped folder.

Academic Offences

http://www.cs.toronto.edu/~fpitt/documents/plagiarism.html !

Notes

Posted on: Friday, July 8, 2011

This note contains few tips and typo clarification regarding assignment 2.

- There are 2 typos in the Entry class

```
public class Entry {
   public Object  _objKey;    // instead of _nKey
   public Object  _objValue;   // instead of _objEntry
}
```

- Remove the void from the following methods:
public abstract Entry delete(Object objKey)
public abstract Entry getMinValue( ) throws UoTLibException;
public abstract Entry getMaxValue( ) throws UoTLibException;
- As mentioned in class, _objKey can be any primitive but since it is
  of type java.lang.Object that means you will get that primitive encapsulated
  in a Java wrapper class. So, the possible variations are: String, Byte, Integer,
  Float, Double, and Long.
- To find max/min, you need to learn about the Comparable interface:
  http://download.oracle.com/javase/1.4.2/docs/api/java/lang/Comparable.html
- Regarding the hash code, you can get it done in one of the following ways:
 1) Use reflection to find the exact type of an object and have a separate appropriate mechanism
    for each type. To find the type: object.getClass().getName()
    For example;

```
 Integer nKey    = new Integer( 10 );
 String  strType = nKey.getClass( ).getName( ); // will return the String "java.lang.Integer"
 if strType.equals ("java.lang.String" )
 //  input to  hash function that works on strings.
 else if strType.equals( "java.lang.Integer")
       //   ....
```

 2) In java.lang.Object, there is a method that returns a String version of the content stored in the Object.
    For example;

```
 Integer nKey = new Integer( 10 );
 String strValue = nKey.toString( );  // will always return the String "10"
```

   which then you could use as input to hash function that works on strings.
  In either case you must provide an implementation of a hash function and do not rely
  on Java's hash code feature.
- Because 2,3,4 and splay are implementations of the abstract data type
  Ordered Dictionary - they need to implement that interface - while Extensible
  Hashtable is an implementation of Dictionary abstract data type and that
  is why it should implement the Dictionary interface.
- The Entry class is what the user of your data structure inserts and
  retrieves from the data structure. The user of your data structure
  should not know about the internal classes used in your data structure
  and the role of the Entry class - beside encapsulating the attributes:
  key & value - is to shield the implementation details of the data
  structure from client code (i.e. user code). Otherwise, you would have to
  expose the internal Node to the user which is not a good design because
  that will increase coupling between the client code and your data structure.
  Having loosely coupled code is always better because either code
  can change independently without affecting the other.
- Your data structre (e.g. 2-3-4 tree) should have it's own classes. For example,
  a class TwoThreeFourNode that represents a 2-3-4 node with attributes such
  as list of pointers to children nodes, maybe a pointer to adjacent nodes, etc...
  That node would have an attribute called Entry that get assigned the
  user passed entry object of type Entry.
  Your TwoThreeFourTree class will have an attribute - typically called root -
  of type TwoThreeFourNode that points to the root of the tree.
- You can also define other classes that are internal to your data structure.
- Try to keep your Node class simple with only the operations that need to be done
  at the Node level. Higher level operations - such as rotation - should be in the main
  data structure class (i.e. TwoThreeFourTree or SplayTree) or in an auxillary class
  you would create for that purpose.
- You can also add other methods to: TwoThreeFourTree.java , ExTable.java , and SplayTree.java
  but these should be internal methods (protected or private) which you would call from

the public methods you have to implement from the interface.