Imperial College of Science, Technology and Medicine	Department of Computing
Computing Science (CS) / Software Engineering (SE)	BEng and MEng
Examinations Part I	Integrated Laboratory Course

Laboratory work is a continuously assessed part of the examinations

and is a required part of the degree assessment.

Laboratory work must be handed in for marking by the due date.

Late submissions may not be marked.

Exercise: 16	Working: Individual
Title: Java Database	
Issue date: 3rd March 2008	Due date: 10th March 2008
System: Linux	Language: Java

Aims

- To gain further experience of *abstract data types* (ADTs) in Java.
- To familiarise yourselves further with Java 1.6's API and Generic Types.
- To illustrate OO concepts such as polymorphism and highlight the advantages of programming with interfaces.
- To practice the design principles taught in the Abstract Data Type course.

The Problem

- Your task is to *implement* two abstract data types (**ADT**s) for use in a database for cat management and *compare* their efficiencies. The database stores information, using an ADT, about the number of cats living in a cattery; in particular, the name of each cat together with the number of tins of cat food each cat eats per week. The cats database should be ordered alphabetically by name. For the purposes of this exercise you can assume that a cat can be uniquely identified by its name.
- You are provided with a main program class **CatAdmin** that provides a menu driven interface to the database.
- **CatAdmin.java** provides a simple menu-driven facility for performing operations on the database. **Database.java** provides implementation methods for those operations through the Dictionary interface. The database commands provided are (where NAME is a String and TINS is an integer):

```
add NAME TINS  // Add a cat to the database
               // If the cat is already present
               // change the number of tins to TINS

lookup NAME    // Print the number of tins eaten by NAME

clear          // Remove all entries

count          // Print the total number of cats and tins

delete NAME    // Remove the cat NAME from the database

print          // Print out the database

help           // Print the help menu

quit           // Quit the database


switch         // Toggle the database type used between
               // BinarySearchTree to OrderedLinkedList
               // Note the BinarySearchTree is used initially

debug          // Print the debug menu

load NUMBER    // Reload the database with NUMBER cats
               // Where NUMBER is an non-negative integer

benchmark      // Benchmark the current implementation
```

- You are also provided with the class **Database** that contains an encapsulated **Dictionary** together with the methods called by **CatAdmin** needed to modify it.
- The **Database** class contains an reference to a **Dictionary** and calls the methods provided by the **Dictionary** class to modify the **Dictionary**.
- The **Dictionary** is an **interface** that contains the headers for the methods that you need to write. This class is generic but you should call it using a **String** and **Integer** as parameters representing the name of a cat and the number of tins of cat food it eats each week. Also provided is the **DictionaryEntry interface** which contains methods that allow the program to get the key and value associated with each entry (or cat name and number of tins eaten for the purposes of this lab exercise).
- You are required to write four implementation classes for the provided interfaces **Dictionary.java** and **DictionaryEntry.java**. You should implement these interfaces using a Binary Search Tree in **BinarySearchTree.java** and **BinarySearchTreeEntry.java**, and an Ordered Linked List in **OrderedLinkedList.java** and **OrderedLinkedListEntry.java**. Each of these classes should reside in the **database** package.

- Once you have written these implementations you should test the relative efficiency of your implementations using the benchmarking tools provided.
- Your **BinarySearchTree** and **OrderedLinkedList** classes should provide the operations needed to process the database. These include an **Iterator** so that the user can step through the database to print out the database and count the total number of tins needed by the cattery each week.

The interface classes that you need to implement

- The **Dictionary** interface which provides the top level methods for operations on the database is given below. You should note that this interface uses generics.

```
public interface Dictionary<K extends Comparable<K>, V>
extends Iterable<DictionaryEntry<K, V>> {

public int size();
/*
 * Returns the number of DictionaryEntry objects stored in the dictionary.
 */

public boolean isEmpty();
/*
 * Tests if this Dictionary contains no DictionaryEntry objects.
 */

public V get(K key);
/*
 * Returns the value associated with the named key in the dictionary. There
 * can be at most one value associated with each key.
 */

public void put(K key, V value);
/*
 * Creates a DictionaryEntry object associating the named key with the
 * specified value. New Entry objects are inserted in to the structure such
 * that they preserve the ordering on keys in the ADT. If the dictionary
 * previously contained an Entry for the key, its value is updated.
 */

public void remove(K key);
/*
 * Removes the DictionaryEntry with the named key from the dictionary,
 * preserving the ordering of Entries on the Dictionary.
 */

public void clear();
/*
 * Removes all entries from the dictionary.
 */
```

```
}
```

- You should note that although these access methods contain no specific information about the data structure implementing the dictionary, some constraints are present. The Dictionary should be instantiated with two kinds of objects, representing key and value respectively. Instances of the key-type, `K`, must be `java.lang.Comparable` [1] objects - this is reflected in the interface declaration. You do not need to implement the **Comparable** interface in the **OrderedLinkedListEntry** and **BinarySearchTreeEntry** classes by providing the code for **compareTo** as this method is already available for the **String** class.

- Each concrete implementation of this interface is required to have an `iterator()` method that returns a `java.util.Iterator` [2] object. Your implementation of the iterators should traverse the ADT returning `DictionaryEntry` objects in ascending order - by key. You should implement the Iterator as an *inner class* to the classes **BinarySearchTree** and **OrderedLinkedList** as the iterator needs to access the data structure in the outer class.

- The **DictionaryEntry** interface which provides the methods for operations on a database entry is given below. You should note that this interface uses generics.

```
public interface DictionaryEntry<K, V> {

public K getKey();
/*
 * Access method for the key corresponding to this Entry.
 */

public V getValue();
/*
 * Access method for the associated value corresponding to this Entry.
 */
}
```

- The interface, `DictionaryEntry`, encapsulates the individual records of the Dictionary exposing *only* access methods for the DictionaryEntry's key and value.

Submit by Monday 10th March 2008

What to Do

- Copy the test harness and main program files **CatAdmin.java**, **Database.java**, **Dictionary.java** and **DictionaryEntry.java** in the package **database**; **Benchmark.java** and **Counter.java** in the package **database.benchmark**; and the file **benchmark.jar** in the top level directory, using the command **exercise 16**.

- This should give you the files you need to write your program in the appropriate directory structure.

[1] http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Comparable.html
[2] http://java.sun.com/j2se/1.5.0/docs/api/java/util/Iterator.html

- Write **BinarySearchTree.java** and **BinarySearchTreeEntry.java** in the **database** package to implement the interface **Dictionary.java** as a binary search tree.
- Write **OrderedLinkedList.java** and **OrderedLinkedListEntry.java** in the **database** package to implement the interface **Dictionary.java** as a an ordered linked list.
- Launch the test harness with the command:

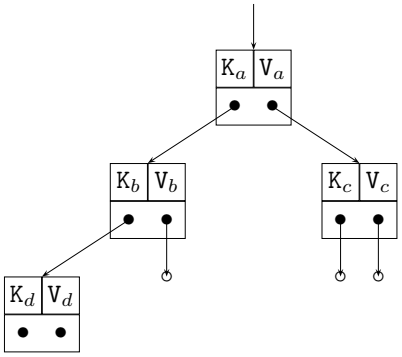
```
java -javaagent:benchmark.jar database.CatAdmin
```

- **Test your code thoroughly** by running the test harness and trying out a wide variety of user command sequences. Use the `debug` menu command from the Cat Administrator menu to access additional menu-items that may help with testing.
- For each ADT find the complexity of the lookup ('get') operation using the Cat Administrator's `load` and `benchmark` commands. Include your findings as a comment at the top of each implementation class.

The BinarySearchTree and OrderedLinkedList classes

- You will not be able to compile the Cat Administrator tool, **CatAdmin.java**, until you have implemented *both* ADTs. We therefore recommended that you create stub classes with dummy methods (see the `database.Dictionary` and `java.lang.Iterable`[3] interfaces) to allow incremental development.
- The node representations for each ADT should implement the `database.DictionaryEntry` interface.
- Java provides a generic `Iterator<T>` interface for traversing elements in collections (where all elements in the collection are traversed only once!). Note that the method **printDatabase** uses an Interator to to access the contents of the database in ascending order. A concrete implementation of an iterator, `Iterator<DictionaryEntry<K,V>>` provides implementations of the following three methods:
  - `boolean hasNext()`: tests if the iteration has more elements, yet to be traversed.
  - `DictionaryEntry<K,V> next()`: returns the next `DictionaryEntry<K,V>` element in the iteration; for this lab, the elements should be returned in ascending order by `K`.
  - `void remove()`: removes the previously returned element from the underlying data structure. (You should *not* implement this method, rather you should provide a stub so that your program compiles.)
- The binary search tree iterator should perform an in-order traversal. Given that the binary search tree is ordered, the iterator's `next()` method should return the nodes of the binary search tree in the "left, top, right" order for example: (K_d,V_d), (K_b,V_b), (K_a,V_a), (K_c,V_c) in the diagram. Pseudo-code for this algorithm is given below.

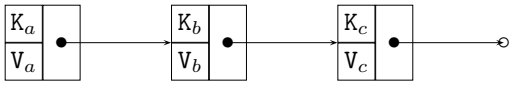
[3] http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Iterable.html



globals:
```
Stack workList;
Node current = root of tree;
```

code:
```
while(current!=null){
    push current node on to workList
    current = current.left
    }
if(workList not empty){
    RESULT = pop node off the workList;
    current = RESULT.right
    }
return RESULT
```

- The ordered linked list iterator should perform a linear traversal. The `next()` method should return the nodes of the OrderedLinkedList in the order: (K_a,V_a), (K_b,V_b), (K_c,V_c). Pseudo-code for this algorithm is given below.



globals:
```
Node current = head of list
```

code:
```
if(current!=null){
    RESULT = current
    current = current.next
    }
return RESULT
```

Benchmarking

- The benchmarking tool hooks on to your implementation classes and counts, specifically, the number of times the method `DictionaryEntry.getKey(..)` is invoked during the execution of the `Dictionary.get(..)` method. For correct results, you **must** ensure your implementations access a DictionaryEntry's key *only* via the `getKey(..)` method.

- Additional commands for the Cat Administrator tool are available through the `debug` menu. For each ADT implementation, plot a graph of the (average number of lookups) vs. (size of database) for varying database sizes. You may find the following helpful.

  1. The `switch` command can be used to toggle the 'active' ADT implementation used by the Database: BinarySearchTree or OrderedLinkedList.
  2. The `load` command can be used to populate the database with a specified number of entries. e.g. the command `load 10000` will populate the database with 10,000 entries.
  3. The `benchmark` command will calculate the average number of lookups to retrieve an arbitrary DictionaryEntry from your implementation of the active ADT.
  4. Using your dataset (or otherwise), deduce the complexity of the lookup operation for each of the ADTs.

- (Optional) Were the average number of lookups for the binary search tree what was expected? If not, explain the discrepancy and amend your implementation based on your findings. You can include your answers to this as a comment at the top of your **BinarySearchTree.java** file.

Submission

- Submit **OrderedLinkedList.java**, **OrderedLinkedListEntry.java**, and **BinarySearchTree.java** and **BinarySearchTreeEntry.java** in the usual way using the **CATE** system.

Assessment

`OrderedLinkedList`	`1.0`
`OrderedLinkedListEntry`	`0.5`
`Iterator for OrderedLinkedList`	`0.5`
`BinarySearchTree`	`1.5`
`BinarySearchTreeEntry`	`0.5`
`Iterator for BinarySearchTree`	`1.0`
`Design, style, readability`	`5.0`
`Total`	`10.0`