15-121: Introduction to Data Structures
Final Exam – Spring 2010 APPENDIX

Listing 1: The BST Class.

```
import java.util.*;

public class BST {
   private Node root;

   public BST() { root = null; }

   public void insert(Comparable data){
      root = insert(root, data);
   }
   private Node insert(Node p, Comparable toInsert){
      if (p == null) return new Node(toInsert);
      if (toInsert.compareTo(p.data) == 0) return p;
      if (toInsert.compareTo(p.data) < 0)
         p.left = insert(p.left, toInsert);
      else
         p.right = insert(p.right, toInsert);
      return p;
   }

   public boolean search(Comparable toSearch){
      return search(root, toSearch);
   }
   private boolean search(Node p, Comparable toSearch){
      if (p == null) return false;
      else if (toSearch.compareTo(p.data) == 0) return true;
      else if (toSearch.compareTo(p.data) < 0)
         return search(p.left, toSearch);
      else
         return search(p.right, toSearch);
   }

   private static class Node        /* The inner Node class */
   {
      private Comparable data;
      private Node left, right;

      public Node(Comparable data){
         left = null; right = null; this.data = data;
      }
      public Node(Comparable data, Node l, Node r){
         left = l; right = r; this.data = data;
      }
      public String toString() { return data.toString(); }
   }
}
```

Listing 2: The Set Interface (implemented by HashSet and TreeSet).

```
public interface Set<AnyType> {
   /* If obj is not present in the set, adds obj and returns
      true.  Otherwise, returns false. */
   public boolean add(AnyType obj);

   /* Adds all of the elements in the specified collection to
      this set if they're not already present. */
   public boolean addAll(Collection<E> c);

   /* Returns true if the set contains obj. */
   public boolean contains(Object obj);

   /* Returns an iterator for iterating over the values in the set. */
   public Iterator<AnyType> iterator();

   /* If obj is present in the set, removes obj and returns true.
      Otherwise, returns false. */
   public boolean remove(Object obj);

   /* Returns the number of elements in the set. */
   public int size();
}
```

Listing 3: The Map Interface (implemented by HashMap and TreeMap).

```
public interface Map<K, V> {
   /* Returns true if the map contains this key.*/
   public boolean containsKey(Object key);

   /* Returns true if this value is associated with some key.*/
   public boolean containsValue(Object value);

   /* Returns the value associated with this key, or null if
      there is no associated value. */
   public V get(Object key);

   /* Returns a set of the keys contained in this map. */
   public Set<K> keySet();

   /* Associates this key with this value.  Returns the value formerly
      associated with this key, or null if this key was not present.
   public V put(K key, V value);

   /* Removes and returns the value associated with this key.
      Returns null if there is no associated value. */
   public V remove(Object key);

   /* Returns the number of key-value mappings in the map. */
   public int size();

   /* Returns a collection of the values contained in the map. */
   public Collection<V> values();
}
```

Listing 4: The LinkedList Class API.

`class LinkedList<AnyType>`

- `boolean add(AnyType obj)` // appends `obj` to end of list
- `void add(int index, AnyType obj)` // inserts `obj` at position `index`
- `AnyType get(int index)` // returns the element at position `index`
- `AnyType remove(int index)` // removes and returns the element at `index`
- `int size()` // returns the number of elements in the list

Listing 5: The ExamStack Class API.

`class ExamStack<AnyType>`

- `boolean isEmpty()` // tests if the stack is empty
- `AnyType pop()` // removes and returns the top item
- `AnyType peek()` // returns the top item (without removing it)
- `void push(AnyType value)` // inserts an item onto the top of the stack
- `int size()` // returns the number of items in the stack

Listing 6: The ExamQueue Interface.

```
public interface ExamQueue<AnyType> {
   /* Tests if the queue is empty */
   public boolean isEmpty();

   /* Adds a value to the back of the queue. */
   public void enqueue(AnyType value);

   /* Returns the first element in the queue. */
   public AnyType getFront();

   /* Returns and removes the front element of the queue. */
   public AnyType dequeue();

   /* Returns the number of items in this queue. */
   public int size();
}
```

Listing 7: The Comparable Interface (implemented by Integer, String, etc).

```
public interface Comparable<AnyType> {
   /* Returns a value < 0 if this is less than other.
      Returns a value = 0 if this is equal to other.
      Returns a value > 0 if this is greater than other. */
   public int compareTo(AnyType other);
}
```

Listing 8: The Comparator Interface.

```
public interface Comparator<AnyType> {
   /* Returns a value < 0 if the first object is less than the second.
      Returns a value = 0 if the first object is equal to the second.
      Returns a value > 0 if the first object is greater than the second.
    */
   public int compare(AnyType first, AnyType second);
}
```

Listing 9: The Iterator Interface.

```
public interface Iterator<AnyType> {
   /* Returns true if the iteration has more elements. */
   public boolean hasNext();

   /* Returns the next element in the iteration. */
   public AnyType next();

   /* Removes the last element that was returned by next. */
   public void remove();
}
```

Listing 10: Selected Methods of the String Class API.

```
public final class String implements Comparable<String> {
   /* Returns the char value at the specified index. */
   public char charAt(int index);

   /* Compares two strings lexicographically. */
   public int compareTo(String anotherString);

   /* Compares this string to the specified object for equality. */
   public boolean equals(Object anObject);

   /* Returns the index within this string of the first occurrence of
   the specified character. */
   public int indexOf(char ch);

   /* Returns the length of this string. */
   public int length();

   /* Returns a new string resulting from replacing all occurrences of
   oldChar in this string with newChar. */
   public String replace(char oldChar, char newChar);

   /* Returns a new string that is a substring of this string starting
   at beginIndex. */
   public String substring(int beginIndex);

   /* Returns a new string that is a substring of this string starting
   at beginIndex and ending at endIndex-1. */
   public String substring(int beginIndex, int endIndex);
}
```