APS 105 – Computer Fundamentals

Assignment #9: Linked List

Fall 1999

(To be completed *before* your lab period, week of November 8)

Objective: To understand linked lists.

You are to implement an abstract data type for an ordered set using linked lists. Each set will hold a series of character values (of type `char`), with the characters kept in increasing lexicographical order. Furthermore, each member of the set will be unique, i.e. duplicates should never be added or included in a set.

The interface for the class is defined by the annotated java code in the following section. You **must** start with the file `/share/copy/aps105/Interface.java` as the basis for your assignment. You should complete and test one method in this code at a time. To help you do this testing, each method has been given some *dummy code* which does nothing but allow you to compile the class without completing all the methods. You are allowed to add additional `private` methods to the class `Ordered` if you need to do so, but you may not remove or change any of the methods in the class interface. You will not need to add any additional instance or class variables.

1 Ordered Set Interface

The interface to the set ADT is shown below. The dummy code to help your testing has been omitted for clarity – however, it is included in the `Interface.java` code which you copy from `/share/copy/aps105`.

```
// A simple class for linked list elements
class Element
{
  public Element next;
  public char    value;

  Element (char c, Element e)
  {
    value = c;
    next  = e;
  }
}
```



```
// Class to represent an ordered set
class Ordered
{
  private Element list;

  // Constructor for an empty set
  Ordered ()
  {
  }

  // Constructor for a singleton set
  Ordered (char c)
  {
  }

  // Constructor for a populated set
  // - Creates a set with all the characters in str as elements
  // - Assumes that elements of str are already in order
  // - Assumes that all elements of str are unique
  // - Each set character in str may be separated by zero or more
  //   spaces, which you must ignore (spaces are not part of the set).
  Ordered (String str)
  {
  }

  // Converts a set into a string
  // - there must be eactly one space (i.e. ' ') between each set element
  //   in the resulting string
  // - be sure to keep the string characters in the same order as the set
  public String toString ()
  {
  }

  // Add c to set, putting it in the correct location
  // - mutates the original set by adding the new element
  public void add (char c)
  {
  }

  // Is the set empty?
  public boolean isEmpty ()
  {
  }
```

```
  // Is the character a member of the set
  // - returns true if c in set, false otherwise
  public boolean isMember (char c)
  {
  }

  // Return a new copy of the set
  public Ordered copy ()
  {
  }

  // Return a new set describing the union of two sets
  public Ordered union (Ordered set)
  {
  }

  // Return a new set describing the intersection of two sets
  public Ordered intersection (Ordered set)
  {
  }

  // Return a new set describing the subtraction of <set> from this set
  // i.e., return (this_set minus (this_set intersect set) )
  public Ordered difference (Ordered set)
  {
  }

  // Return a reference to the first item in the linked list
  public Element getList ()
  {
      return list;
  }

} // End of class Ordered
```


2 The Main Program

Unlike other labs, you will not be required to submit a program which actually uses `Ordered`. You should create a `main` method in order to test your implementation before submitting it, but any code you implement for this purpose will not be marked. However you are still responsible for ensuring that your entire submission satisfies style guidelines. As part of the marking process your version of `Ordered` will be recompiled and then executed with a testing class that will not have seen. The mark you recieve for the implementation component of this lab will be based on whether your abstract data type provides the desired behaviour. If your class does not compile, or if the constructors do not work, you will get 0 since your class is not usable. If you only complete part of the lab, it is important that you leave the *dummy code* in place so the TA can still compile and test the working parts of your code.

You will find it convenient to use the method `toString` when testing since it is automatically called by `println` and + (string concatenation) when given a reference to an object. The following simple `main` method, which successfully compiles and executes, illustrates this.

```
public static void main (String[] args)
{
  // Space between elements in string is optional
  Ordered o1 = new Ordered ("a b c d e");
  Ordered o2 = new Ordered ("acegi");

  // the Ordered method 'toString' is automatically
  // called in following printlns

  System.out.println ("Union of: " + o1);
    // calls o1.toString() before printing

  System.out.println ("     and: " + o2);
    // calls o2.toString() before printing

  System.out.println ("      is: " + o1.union (o2));
    // makes a new object, the union of o1 and o2,
    // and calls toString() for the new object
}
```

The output of this program is shown below.

```
Union of: a b c d e
     and: a c e g i
      is: a b c d e g i
```

3 Submitting your Assignment

Your ADT is going to be electronically submitted and marked. For this reason, it is very important that you have not modified the interface to the ADT as given in `Interface.java`, and that each of the methods performs the correct function. It is alright if you have added additional methods, but you must not have removed or modified the ones given.

To prepare your program for submission, you should make sure the file containing your program is named `Ordered.java`. Everything should be in this one file, including the description of the `Element` class. If you define any `main()` methods, these will be ignored. You should not use any `Stdin` methods in your program.

First test that your file `Ordered.java` is self-contained by making a new directory. Copy it and `InterfaceTest.ja` into that directory. Compile those two programs and run the test. If you have any compiling errors, your `Ordered.java` is incomplete and will not be markable.

Once you are sure that `Ordered.java` is self-contained, you may submit it electronically. Do this by running the following command:

- `submitaps105f 1 Ordered.java` if your lab is on Monday.
- `submitaps105f 2 Ordered.java` if your lab is on Tuesday.
- `submitaps105f 3 Ordered.java` if your lab is on Wednesday.

You may run the submit command more than once – only the latest version will be marked. Your deadline to submit is midnight at the end of your lab day.

You may also check that your program has been submitted by placing the the `-l` (the letter 'l') option before your lab day/number, for example:

```
skule.ecf% submitaps105f -l 1
total 10
-rw-r-----    1 tester   aps105f     4662 Nov  7 21:43 Ordered.java
```

You may compare the size of the file submitted (the 5th column, 4662 bytes) against size of the file in your directory by doing an `ls -l` in your directory.

If your program has only a few working methods, they will be tested and you will receive partial credit. If none of your methods work, or if you fail to submit anything, you will receive a mark of zero.