Massachusetts Institute of Technology

16.410-13 Principles of Autonomy and Decision Making

Problem Set #1

Objective

The primary objective of this problem set is to begin to exercise your skill at designing and implementing Java programs. This includes developing comfort at programming in Java and using JUnit to test your code.

Submission Instructions

Please submit your source codes [src] electronically. However, do NOT submit a collection of files. You must submit a single ZIP file containing all of the source files. Make sure to comment your codes sufficiently so that the grader can easily understand your code. No partial credit can be given if the code cannot be understood.

Submit a single PDF [pdf] file containing all additional material such as execution traces and analysis. You must also submit a hard copy of the PDF file.

At the end of each problem, a list specifies all items that must be included in your submission. Each item that has [pdf] written next to it must be part of the PDF document and not a separate file. As you complete your assignment, please include in your PDF document at the end of each answer how long you spent on it.

Background

In the following, JINS denotes "Java in a Nutshell," by David Flanagan.

Start by reading the intro, Chapter One, of JINS. Next, run the chapter's factorial example using the Java Developers Kit (JDK) and the Eclipse Integrated Development Environment (IDE), as described in the class handout, "Java Jump Start." This will give you a sense of the Java debugging and development cycle.

Next, to develop your understanding of the Java language, work through the Sun Java tutorial, located at http://java.sun.com/docs/books/tutorial/. In addition, you may want to read Chapter 3 of JINS to further develop your understanding of object oriented programming in JAVA, and Chapter 2 of JINS to develop your understanding of expressions in the language.

You will save significant time implementing Problems 1-2 by using the Collections package, which is part of the java.util package. This is described in JINS pages 224-238. In addition, you will likely find the Sun online API documentation useful for this (http://java.sun.com/javase/6/docs/api/).

You can choose to develop and debug your code using command-line commands, through JDK, or using the window-based Eclipse IDE. Both can be found on the lab machines. Many of you will want to load these systems on your own machines. You can download Java Runtime Environment (JRE) and JDK (JDK 6 Update 2) from http://java.sun.com/javase/downloads/index.jsp. You can download Eclipse (Eclipse IDE for Java Developers) from http://www.eclipse.org/downloads/.

JDK is described in Chapter 8 of JINS. Of particular note is the command "javac," which compiles a java source file (.java) to byte code file (.class), the command "java," which interprets (runs) a byte code file, and the command "jdb," which is used to debug a java program. Eclipse provides an online tutorial and online documentation, by starting up Eclipse, and clicking Help in the toolbar.

A Special Note about JUnit and Automatic Grading

Please take care to read the instructions carefully. If you are asked to implement a class called "`Foo`," it is important that your class be called "`Foo`". If you do not do this, then our automated test code will fail to find your class and you will get no credit for your effort. Similarly, method signatures must match those specified in the problem set.

Automated testing is performed using JUunit. You can download Junit from http://www.junit.org/.

There are lots of valuable information on www.junit.org. Check their "Getting Started" and "Documentation" tabs. Some simple examples of how to use JUnit are included. JUnit is fun to play with.

Problem 1

In this problem you will begin exercising your skill at simple Java programming, debugging (via JDK or Eclipse) and testing (using junit). You should begin by learning about the java.util.LinkedList class and the interface java.util.List (the API documentation mentioned above or JINS will be useful for this).

Implement a class called FunWithLinkedLists<T> that does the following:

1. Implements a reverse method that takes a LinkedList and returns a new LinkedList that has the same elements as the old LinkedList, but in the reverse order. Your method signature must be:

```
    public LinkedList<T> reverse(LinkedList<T> list) {
    };
```

2. Implement an append method that takes two LinkedList arguments and returns a new LinkedList which has all the elements of the first LinkedList followed by all of the elements of the second LinkedList. The method signature must be:

```
    public  LinkedList<T>  append(LinkedList<T>  list_a,  LinkedList<T>
list_b) {
    };
```

You may use the following as a template:

```
/**
 * Implement a class called FunWithLinkedLists that is capable of
 * reversing a linked list or appending two linked list of the same
 * type.
 *
 * @author        <First> <Last>
 */
import java.util.LinkedList;
import java.util.ListIterator;

public class FunWithLinkedLists<T> {
    /**
     * Takes a linked list and returns a new linked list that has the
     * same elements as the old linked list, but in the reverse order.
     *
     * @param     list        A linked list containing elements of type
     *                        <T>.
     * @return                A linked list with the same list in the
     *                        reversed order.
     */
    public LinkedList<T> reverse(LinkedList<T> list) {
    };

    /**
     * Takes two linked lists and returns a new linked list which has
     * all the elements of the first linked list followed by all of
     * the elements of the second linked list.
     *
     * @param     list_a      First linked list containing elements of
     *                        type <T>.
     * @param     list_b      Second linked list containing elements of
     *                        type <T>.
     * @return                A linked list with <code>list_a</code>
     *                        appended to <code>list_b</code>.
     */
    public LinkedList<T> append(LinkedList<T> list_a, LinkedList<T>
list_b) {
      };
}
```

You must include in your submission:

1. *[src] The source code of your FunWithLinkedLists class.*
2. *[pdf] The amount of time you spent on this problem.*

Problem 2

Build a JUnit test suite for the new methods, reverse and append, that you developed in Steps 1 and 2. Demonstrate that your new methods work correctly by showing the output from JUnit.

You must include in your submission:

1. *[src] The source code of your JUnit test class.*
2. *[pdf] Execution trace of the test.*
3. *[pdf] The amount of time you spent on this problem.*