CS 635 Advanced OO Design and Programming
Spring Semester, 2016
Assignment 4

San Diego State University -- This page last updated 4/17/16

Due May 5

1. In this problem we will use a Chain-of-Responsibility like solution finding substrings in a string with wild characters. We will use the characters "*" and "." for wild characters. The character "." matches any single character. The character "*" matches zero or more characters. So "c.t" with match "cat", "cbt", etc. While "c*t" with match "ct", "cat", "caat", "cbt", "casdert" and many others. Now a Match object will find the first match of a pattern in a string. For example in:

```
match = new Match( "c.t" );
startIndex = match.findFirstIn( "cacacat");
```

startIndex will be 4. Now it is not too hard to implement a pattern matcher. For practice we will use a chain of objects to do the matching. There will be one object in the chain for each character in the pattern. I will use "ca.t" to illustrate. The chain for this pattern will contain at least four objects. The first object (head) in the chain needs to find the first occurrence of the character c in the target string and pass the target string and the location in the string to the next object in chain. If the next object in the chain reports success, then the head of the chain returns the location of the pattern. If the next object reports failure to match, the head object needs to find the second occurrence of c in the string. The head object will try all occurrences of c in the string until a match is found or it is determined that the pattern is not in the string. The second object in the chain behaves slightly differently. It is given a position and the string from the head object. If the next character is an "a" it passes the request to match to the next object in the chain. It then reports to the head object the result returned by the rest of the chain. If the next character is not an "a" it returns failure to the head chain.

You will need to think about the how to implement the objects that handle the wild characters. Also there are several ways to handle the end of the chain.

2. In this problem we will use a visitor on composites. You will read XML description from a file and build a composite tree with nodes of type Document, Text and Header. You will create two separate Visitor classes. The first visitor will print out just the headers as html headers in the order they appear in XML file. The headers are to be separated by a new line character. The second visitor will print out the tree as a single document html document. The headers and text should appear in the document in the same order as they appear in the XML file.

The XML is simple enough that you may wish to parse the XML directly. however you can also use the SAX interface to XML parsers. The Java XML parser and documentation can be found at: http://www.saxproject.org.

The XML

The XML we will use can be described using BNF like syntax as:

```
CS635Document ::= <CS635Document>(CS635Document* |
                        (header [text*] )* | text*)*</CS635Document>
header ::= <header>char+</header>
text ::= <text>char+</text>
```

The proper way to define XML is with a DTD (Document Type Definition). Here is the DTD.

```
<!DOCTYPE CS635Document [
    <!ELEMENT CS635Document ( (CS635Document* | ( header , text*)* | text* )* )>
    <!ELEMENT header (#PCDATA)>
    <!ELEMENT text (#PCDATA)>
]>
```

So an example XML file is:

```
<?xml version="1.0" ?>
<CS635Document>
    <header>This is an example</header>
    <text>Not much here</text>
    <CS635Document>
        <text>Just text here</text>
    </CS635Document>
</CS635Document>
```

If you wish to validate the XML we can add the DTD to get

```
<?xml version="1.0" ?>
<!DOCTYPE CS635Document [
<!ELEMENT CS635Document ( (CS635Document* | ( header , text*)* | text* )* )>
<!ELEMENT header (#PCDATA)>
<!ELEMENT text (#PCDATA)> ]>
<CS635Document>
    <header>This is an example</header>
    <text>Not much here</text>
    <CS635Document>
        <text>Just text here</text>
    </CS635Document>
</CS635Document>
```

SAX

SAX, Simple API for XML, is one way to use an XML parser to parse XML. In SAX you create a class which the parser hands parts of the XML. In Java it implements the interface org.xml.sax**.**ContentHandler. There are a number of methods that the XML parser will call on this class. The important ones for this assignment are:

public void startDocument()

public void endDocument ()

startElement: namespaceURI localName: localName qName: name attributes: attributes

```
public void startElement(java.lang.String uri,
                         java.lang.String localName,
                         java.lang.String qName,
                         Attributes atts)
                  throws SAXException
```

Called when the parser starts to parse a start tag (<header> <text> < CS635Document>. We are only interested in the name of the tag. In our case the localName and qName are the same and will be the name of the tag.

```
public void endElement(java.lang.String uri,
                       java.lang.String localName,
                       java.lang.String qName)
                throws SAXException
```

Called when the parser parses an end tag (</header> </text> </CS635Document>. We are only interested in the name of the tag. In our case the localName and qName are the same and will be the name of the tag.

characters: aString
public void characters(char[] ch, int start, int length) throws SAXException

The parser passes the text between the start and end tags to our code using this method.

Simple Example

I will use the following DTD to define the XML for this example.

```
<!DOCTYPE Sample [
<!ELEMENT Sample (  ( a* , b*)*)>
<!ELEMENT a (#PCDATA)>
<!ELEMENT b (#PCDATA)> ]>
```

This states that the first tag in the document must be:

```
<Sample>
</Sample>
```

The sample tag can contain tags a and b. The tags a and b can occur in any order and appear multiple times. The tags a and b contain only text. So the following are valid

```
<Sample>
    <a>hi</a>
    <b>mom</b>
    <b> how are</b>
    <a> you </a>
</Sample>

<Sample>
    <a>cat</a>
</Sample>
```

Java Example

A short example to show how to call the parser.

```
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.SAXParser;
import java.util.Vector;
import java.io.File;

public class SAXDriverExample extends DefaultHandler
    {
    private Vector root;

    public static void main(String argv[])
        {
        SAXDriverExample handler = new SAXDriverExample();

        // Use the default (non-validating) parser
        SAXParserFactory factory = SAXParserFactory.newInstance();
```

```
        try
            {
            SAXParser saxParser = factory.newSAXParser();
            saxParser.parse( new File("sample"), handler );
            }
        catch (Throwable t)
            {
            t.printStackTrace();
            }
        System.out.println( handler.root());
        }

    public void startDocument()
        {
        root = new Vector();
        }

    public void characters(char[] ch, int start, int length)
        {
        root.addElement( new String( ch, start, length));
        }

    public Vector root()
        {
        return root;
        }
    }
```

The file sample contains:

```
<?xml version="1.0" ?>
<Sample>
    <a>hi</a>
    <b>mom</b>
    <b>how are</b>
    <a>you</a>
</Sample>
```

Grading

	Percent of Grade
Working Code	15%
Unit Tests	15%
Quality of Code	20%
Proper implementation of Patterns	50%

What to Turn in

Turn in hard copy of your code.

Late Policy

An assignment turned in 1-7 days late, will lose 5% of the total value of the assignment per day late. The eight day late the penalty will be 40% of the assignment, the ninth day late the penalty will be 60%, after the ninth day late the penalty will be 90%. Once a solution to an assignment has been posted or discussed in class, the assignment will no longer be accepted. Late penalties are always rounded up to the next integer value.