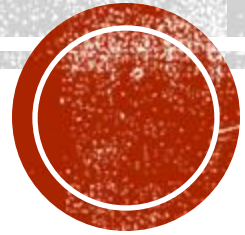
CHAPTER 3.2



Pattern matching with regular expressions.

Parsing structured data (HTML, XML and JSON.)

Acknowledged:

https://courses.cs.duke.edu/fall14/compsci316/lectures/15-xml-notes.pdf

https://www.cs.cornell.edu/courses/cs2112/2018fa/labs/lab07/Slides.pdf

REGULAR EXPRESSIONS

- Server-side programming language- Perl is used for text manipulation.
- Regular expressions are built into the syntax of Perl.
- Beginning with Java 1.4, Java has a regular expression package, java.util.regex
- Java's regular expressions are almost identical to those of Perl.
- Regular expressions are used in the automatic generation of Web pages.

REGULAR EXPRESSIONS

Regular expressions are not easy to use at first,

- It's a bunch of punctuation, not words.
- The individual pieces are not hard, but it takes practice to learn to put them together correctly.
- Regular expressions form a miniature programming language.
- It's a different kind of programming language than Java, and requires you to learn new thought patterns.
- In Java you can't just use a regular expression; you have to first create Patterns and Matchers.

REGULAR EXPRESSIONS IN JAVA

First compile the pattern

- import java.util.regex.*;
- Pattern p = Pattern.compile("[a-z]+");

Next create a matcher for a specific piece of text by sending a message to your pattern

- Matcher m = p.matcher("Now is the time");

REGULAR EXPRESSIONS IN JAVA

- Pattern and Matcher are both in java.util.regex
- Neither Pattern nor Matcher has a public constructor; you create these by using methods in the Pattern class
- The matcher contains information about both the pattern to use and the text to which it will be applied

REGULAR EXPRESSIONS IN JAVA

matcher m,

- m.matches() returns true if the pattern matches the entire text string, and false otherwise
- m.lookingAt() returns true if the pattern matches at the beginning of the text string, and false otherwise
- m.find() returns true if the pattern matches any part of the text string, and false otherwise

REGULAR EXPRESSIONS IN JAVA

- After a successful match, m.start() will return the index of the first character matched
- After a successful match, m.end() will return the index of the last character matched, plus one
- If no match was attempted, or if the match was unsuccessful, m.start() and m.end() will throw an IllegalStateException

EXAMPLE

```
import java.util.regex.*;

public class RegexTest {
   public static void main(String args[]) {
      String pattern = "[a-z]+";
      String text = "Now is the time";
      Pattern p = Pattern.compile(pattern);
      Matcher m = p.matcher(text);
      while (m.find()) {
         System.out.print(text.substring(m.start(),
                                  m.end()) + "*");
      }
   }
}
```

PATTERNS

- abc exactly this sequence of three letters
- [abc] any one of the letters a, b, or c
- [^abc] any character except one of the letters a, b, or c (immediately within an open bracket, ^ means "not," but anywhere else it just means the character ^)
- [a-z] any one character from a through z, inclusive
- [a-zA-Z0-9] any one letter or digit
- If one pattern is followed by another, the two patterns must match consecutively
  - For example, [A-Za-z]+[0-9] will match one or more letters immediately followed by one digit
- The vertical bar, |, is used to separate alternatives
  - For example, the pattern abc|xyz will match either abc or xyz

PREDEFINED CHARACTER CLASSES

- . any one character except a line terminator
- \d a digit: [0-9]
- \D a non-digit: [^0-9]
- \s a whitespace character: [\t\n\x0B\f\r]
- \S a non-whitespace character: [^\s]
- \w a word character: [a-zA-Z_0-9]
- \W a non-word character: [^\w]

GREEDY QUANTIFIERS

Assume X represents some pattern

- X? optional, X occurs once or not at all
- X* X occurs zero or more times
- X+ X occurs one or more times
- X{n} X occurs exactly n times
- X{n,} X occurs n or more times
- X{n,m} X occurs at least n but not more than m times

CAPTURING GROUPS

- If m is a matcher that has just performed a successful match, then m.group(n) returns the String matched by capturing group n. This could be an empty string This will be null if the pattern as a whole matched but this particular group didn't match anything.
- m.group() returns the String matched by the entire pattern (same as m.group(0)). This could be an empty string
- If m didn't match (or wasn't tried), then these methods will throw an IllegalStateException

PARSING STRUCTURED DATA

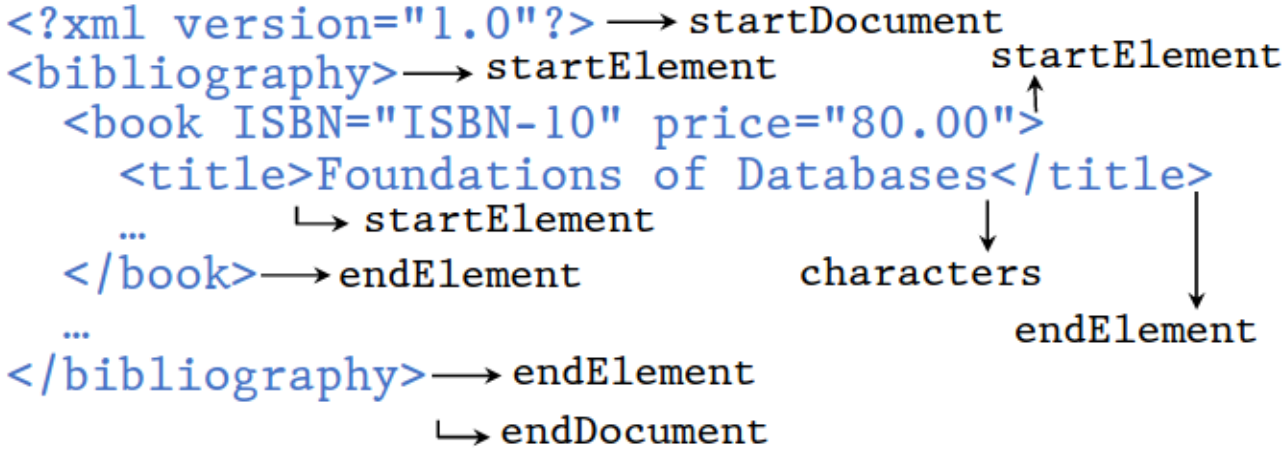
XML

• SAX (Simple API for XML): Started out as a Java API, but now exists for other languages too
• DOM (Document Object Model): Language-neutral API with implementations in Java, C++, python, etc.

SAX EVENTS

Most frequently used events:

- StartDocument
- EndDocument
- StartElement
- EndElement
- characters



DOM

XML is parsed by a parser and converted into an in-memory DOM tree.

DOM API allows an application to,

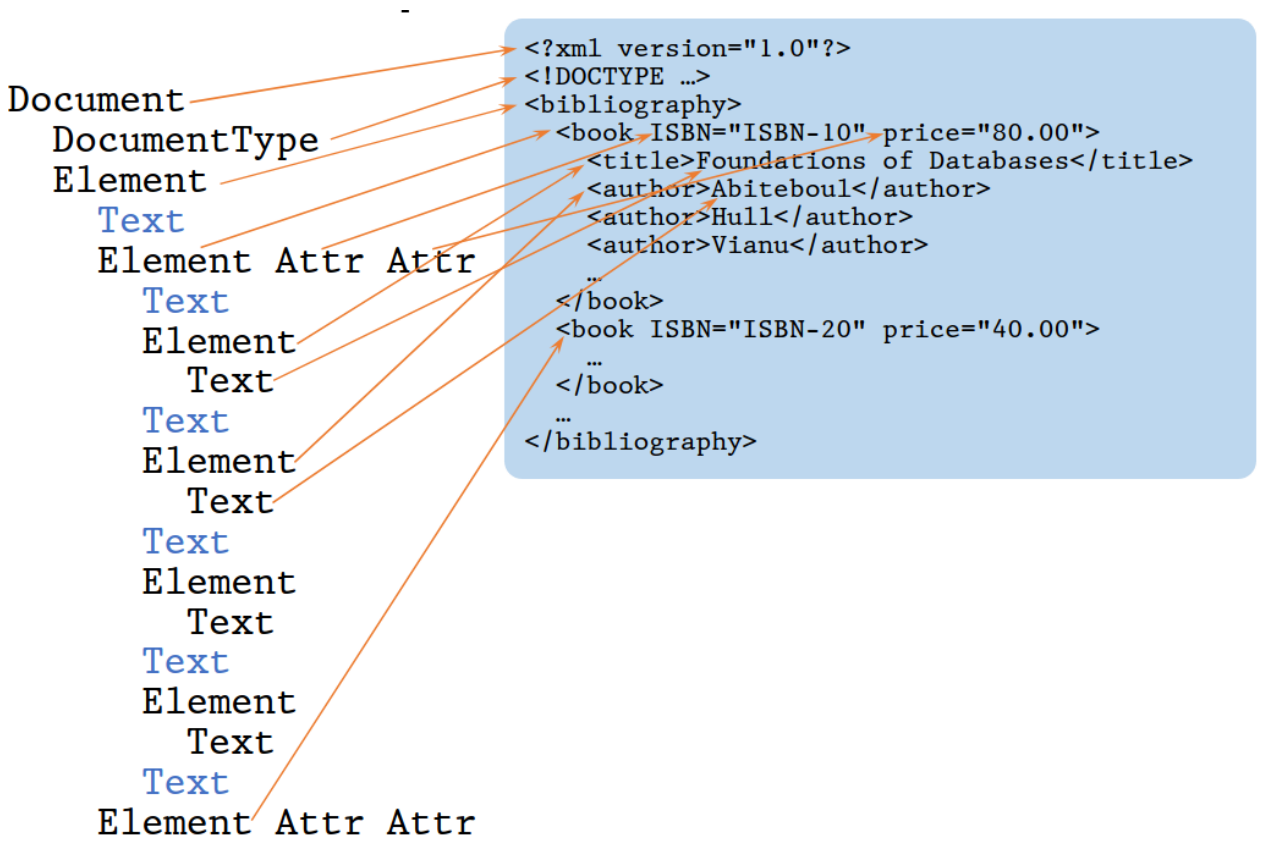
- Construct a DOM tree from an XML document
- Traverse and read a DOM tree
- Construct a new, empty DOM tree from scratch
- Modify an existing DOM tree
- Copy subtrees from one DOM tree to another etc.

DOM TREE

Most frequently used types of Node's:

- Document: root of the DOM tree
- Not the sames as the root element of XML
- DocumentType: corresponds to the DOCTYPE declaration in an XML document
- Element: corresponds to an XML element
- Attr: corresponds to an attribute of an XML element
- Text: corresponds to chunk of text

DOM TREE


Document
DocumentType
Element
Text
Element Attr Attr
Text
Element
Text
Text
Element
Text
Text
Element
Text
Text
Element
Text
Text
Element Attr Attr
<?xml version="1.0"?>
<!DOCTYPE …>
<bibliography>
<book ISBN="ISBN-10" price="80.00">
<title>Foundations of Databases</title>
<author>Abiteboul</author>
<author>Hull</author>
<author>Vianu</author>
…
</book>
<book ISBN="ISBN-20" price="40.00">
…
</book>
…
</bibliography>