INFS 740
DB programming for the WEB

Prof. Alex Brodsky
Department of Information and Software Engineering
George Mason University

Java

- Object-oriented programming language.
  - Inheritance, overloading and overriding, dynamic binding.
  - Interfaces, reflection.
- Platform independent.
  - Interpreted.
- Lots of APIs.
  - Standard utilities.
  - 2D and 3D graphics, accessibility, servers, collaboration, telephony, speech, animation, and more.

Benefits of Java

- **Get started quickly**
- **Write less code**
- **Write better code**
- **Develop programs more quickly**
- **Avoid platform dependencies with 100% pure Java**
- **Write once, run anywhere**
- **Distribute software more easily**

HelloWorld.java

```
public class HelloWorld {
    public static void main(String[] args) {
        // Display "Hello World!"
        System.out.println("Hello World!");
    }
}
```

Compilation and Execution

- Compile:
  - Use compiler javac
  - Compile java source file to bytecode program
- Execution
  - Use interpreter java
  - Interprets the bytecode
- Platform independence: bytecode program should be executable on all platforms

Object-Oriented Languages

- Object
  - An object is a software bundle of related variables and methods. Software objects are often used to model real-world objects you find in everyday life.
- Message
  - Software objects interact and communicate with each other using messages.
- Class
  - A class is a blueprint or prototype that defines the variables and the methods common to all objects of a certain kind.

Object-Oriented Languages

- Inheritance
  - A class inherits state and behavior from its superclass. Inheritance provides a powerful and natural mechanism for organizing and structuring software programs.
- Interface
  - An interface is a contract in the form of a collection of method and constant declarations. When a class implements an interface, it promises to implement all of the methods declared in that interface.
    - Example: Set

Variables *Declared as "type name"*

```
public class MaxVariablesDemo {
   public static void main(String args[]) {

      // integers
      byte largestByte = Byte.MAX_VALUE;
      short largestShort = Short.MAX_VALUE;
      int largestInteger = Integer.MAX_VALUE;
      long largestLong = Long.MAX_VALUE;
      // real numbers
      float largestFloat = Float.MAX_VALUE;
      double largestDouble = Double.MAX_VALUE;

      // other primitive types
      char aChar = 'S';
      boolean aBoolean = true;
```

```
// display them all
        System.out.println("The largest byte value is " + largestByte);
        System.out.println("The largest short value is " + largestShort);
        System.out.println("The largest integer value is " + largestInteger);
        System.out.println("The largest long value is " + largestLong);
        System.out.println("The largest float value is " + largestFloat);
        System.out.println("The largest double value is " + largestDouble);
        if (Character.isUpperCase(aChar)) {
            System.out.println("The character " + aChar + " is upper case.");
        } else {
            System.out.println("The character " + aChar + " is lower case.");
        }
        System.out.println("The value of aBoolean is " + aBoolean);
    }
}
```

Handling Objects

```
public class Rectangle {
    public int width = 0;
    public int height = 0;
    public Point origin;

    //Four constructors
    public Rectangle() {
        origin = new Point(0, 0);
    }
    public Rectangle(Point p) {
        origin = p;
    }
```

(continued)

```
public Rectangle(int w, int h) {
        this(new Point(0, 0), w, h);
    }

    public Rectangle(Point p, int w, int h) {
        origin = p;
        width = w;
        height = h;
    }
```

Handling Objects

```
//A method for moving the rectangle
    public void move(int x, int y) {
        origin.x = x;
        origin.y = y;
    }

    //A method for computing the area of the rectangle
    public int area() {
        return width * height;
    }
}
```

Handling Objects

- Create object

```
Rectangle rect_one = new Rectangle(origin_one, 100, 200);
```

- Use object

```
System.out.println("area=" + rect_one.area());
```

- Destroy object
  – When all the references to it are gone (garbage collection).

Language Constructs

- Variables, assignment, operators
- Flow control
  - While loop

    while (*expression*) { *statements* }
  - For loop

    for (*initialization*; *termination*; *increment*) { *statements* }
  - If-else
    - if (*expression*) { *statement(s)* } else {*statement(s)*}

System Utilities and Basic Types

- Output
  - System.out.println (plus some others!)
- Basic types
  - Characters and strings
  - Arrays
  - Numbers
- Interfaces (you can define your own)
  - Sets
  - Maps
  - So on….

Inheritance

- A *subclass* is a class that extends another class. A subclass inherits state and behavior from all of its ancestors. The term "superclass" refers to a class's direct ancestor as well as to all of its ascendant classes.
- Example:
  - Public class Hello extends HttpServlet {…}

XML

- eXtensible Markup Language (XML) is a simple, standard way to describe text data
- Described as “ASCII of the web”
- XML - a W3C standard, lets you create your own tags
- Tags name the concept you are describing

Advantages

- More semantics to the data
- More structures
- Move to the web
- XML can be used to exchange data with other users and programs in a platform independent way

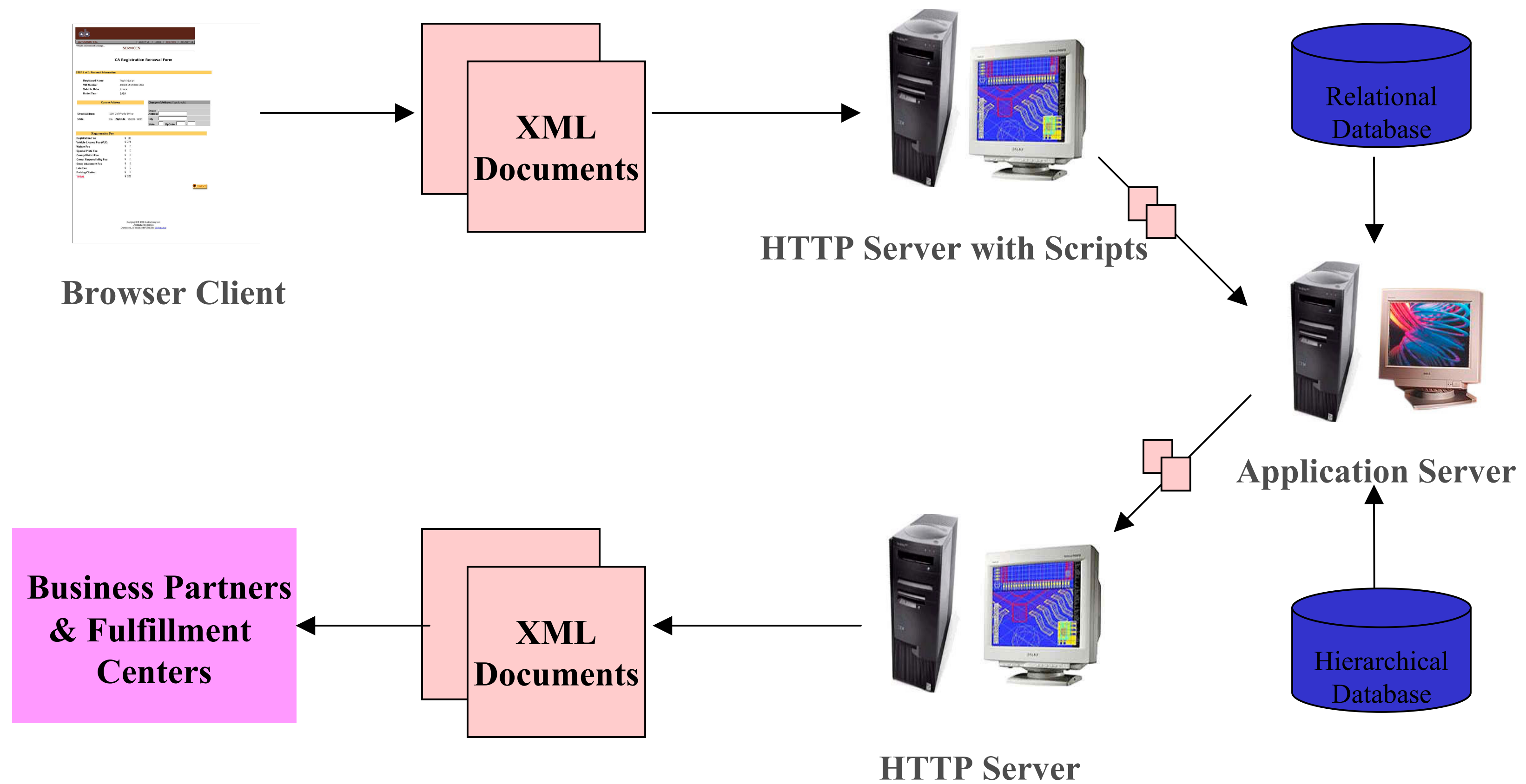
Example

```
<?xml version=”1.0”?>
<vehicle>
        <vin>2HGEJ6675YH519046</vin>
        <make>Honda</make>
        <year>2000</year>
        <model type=”compact”>Civic</model>
        <owner>
                <name>Kate Winslet</name>
                <address>48 My St, Atown, CA</address>
                <type>01</type>
                <dln>18147890</dln>
                <dob>05-11-1974</dob>
        </owner>
</vehicle>
```

HTML vs XML

- HTML tags describe rendering, XML tags describe data
- HTML can at most do browser specific presentation
- In HTML Information extraction is not easy
- XML is written for information exchange

Web Architecture with XML



Well-formed Documents

- Tags are nested correctly
- Well-formed document example

```
<a>1
  <b>2
  </b>
</a>
```

- Not well-formed document example

```
<a>1<b>2</a></b>
```

DTD & Schema

- Describe the structure of "valid" XML documents
- Use a form of context-free grammar.

XML Syntax

- Elements
  - XML tags for markup
- Attributes
  - Tuple information of elements
- Declarations
  - Instructions to XML processor
- Processing Instructions
  - Instructions to external applications

A Piece of XML

```
<seq id="my_seq" name="NUCLEAR RIBONUCLEOPROTEIN">
  <dbxref>
    <database>SWISS-PROT</database>
    <unique_id>P09651</unique_id>
  </dbxref>
  <residues type="aa">
   SKSESPKEPEQLRKLFIGGLSFETTDESLRSHFEQWGTLTDCVVMRDPNTKRSRGFGFV
   TYATVEEVDAAMNARPHKVDGRVVEPKRAVSREDSQRPGAHLTVKKIFVGGIKEDTEEH
   HLRDYFEQYGKIEVIEIMTDRGSGKKRGFAFVTFDDHDSVDKIVIQKYHTVNGHNCEVR
   KALSKQEMASASSSQRGRSGSGNFGGGRGGGFGGNDNFGRGGNFSGRGGFGGSRGGGGY
   GGSGDGYNGFGNDGGYGGGGPGYSGGSRGYGSGGQGYGNQGSGYGGSGSYDSYNNGGGR
   GFGGGSGSNFGGGGSYNDFGNYNNQSSNFGPMKGGNFGGRSSGPYGGGGQYFAKPRNQG
   GYGGSSSSSSYGSGRRF
  </residues>
</seq>
```

An XML DTD

```
<?xml version='1.0' encoding="US-ASCII"?>

<!DOCTYPE seq [
 <!ELEMENT seq (dbxref*, residues?) >
 <!ATTLIST seq id      ID      #REQUIRED
               name    CDATA   #IMPLIED
               length  CDATA   #IMPLIED >

<!ELEMENT residues (#PCDATA)>
 <!ATTLIST residues   type    (dna | rna | aa) #REQUIRED>
]>
```

Elements

- Basic rules
  - Start tag <tag_name> and end tag </tag_name>
  - Tags must be nested
    - `<tag1><tag2>…</tag2></tag1>`
  - Tags may be empty (no enclosed data)
    - `<empty_tag/>`
  - Whitespace in element content usually ignored
    - `<section><p> … </p></section>`
    -
```
<section>
  <p> … </p>
</section>
```

Attributes

- Provides additional information about an element
  - Enclosed by quotes - either `"` or `'`
  - Case-sensitive
  - May be character data or tokenized
    - `value="Blue Peter"` (character data)
    - `value = "blue"` (single token)
    - `value = "red green blue"` (tokens)
  - Values may be enumerated or defaulted (DTD)

Declarations

- Instructions for the XML processor
- Format - <! … > or <! … [<! … >]>
  - Document type - <!DOCTYPE … >
  - Character data - <![CDATA[…]]>
  - Entities - <!ENTITY … >
  - Notation - <!NOTATION … >
  - Element - <!ELEMENT … >
  - Attributes - <!ATTLIST … >
  - <![INCLUDE[…]]> and <![IGNORE[…]]>

Processing Instructions

- Information required by an external application
- Processing Instructions
  - Format - **`<? … ?>`**
  - XML PI - **`<?xml version='1.0’ ?>`**
    - Confusingly, this is called the XML declaration, but is a processing instruction

Comment Declaration

- Comments are not considered part of XML document and should not be published
  - **`<!-- A comment -->`**
- Cannot have additional '**--**' in comment
- Cannot embed inside other declarations

Character Data Declaration

- For occasions when text must contain uninterpreted markup characters
  - **Press <<<ENTER>>>**
  - **<![CDATA[Press <<<ENTER>>>]]>**

Entities

- XML document may be distributed among a number of files
  - Each unit of information is called an **entity**
  - Each entity has a name to identify it
  - Defined using an entity declaration
  - Used by calling an entity reference

Element Declarations

- Used to define new elements and their content
  - **<!ELEMENT name (#PCDATA)>**
    ⇒ **<name> … </name>**
- Empty element has no content
  - **<!ELEMENT name EMPTY>**
    ⇒ **<name/>**
- When children allowed - **any** or model group
  - **<!ELEMENT name ANY>**
  - **<!ELEMENT person (name, e-mail*)>**

Model Groups

- Used to define content of elements
  - `<!ELEMENT person (name, e-mail*)>`
- Used to define hierarchies of elements
  - ```
    <!ELEMENT name    (fname, surname)>
    <!ELEMENT fname   (#PCDATA)>
    <!ELEMENT surname (#PCDATA)>
    <!ELEMENT e-mail  (#PCDATA)>
    ```
- Control organisation of elements
  - Sequence connector ',': **(A, B, C)**
  - Choice connector '|': **(A | B | C)**
    ```

Model Group Quantity Indicators

- Describe constraints on elements in DTD

A?	May occur	[0..1]
A+	Must occur	[1..*]
A*	May occur	[0..*]
A | B	Either A or B	
A, B	A followed by B	

```
(A, B)+
((A,B?) | C+)*
```

Attribute Declarations

- Attributes can be attached to elements
- Declared separately in ATTLIST declaration
  - **`<!ATTLIST tag … >`**
  - Rest of definition specifies
    - attribute name
    - attribute type
    - default value

Attribute Names and Types

- Attribute name
  - **<!ATTLIST tag name type default>**
  - **<!ATTLIST tag first_attr …**
    secon_attr …
    third_attr … >
- Attribute types

```
CDATA        ID
NMTOKEN      IDREF
NMTOKENS     IDREFS
ENTITY       NOTATION
ENTITIES     name group
```

Attribute Types

- CDATA
  - Character data
- NMTOKEN
  - Single token
- NMTOKENS
  - Multiple tokens
- ENTITY
  - Attribute is entity ref
- ENTITIES
  - Multiple entity ref's
- ID
- Unique ID
- IDREF
  - Match to ID
- IDREFS
  - Match to multiple ID's
- NOTATION
  - Describe non-XML data
- Name group
  - Restricted list

Attribute Types

- CDATA
  - `name = "Tom Jones"`
- NMTOKEN
  - `color="red"`
- NMTOKENS
  - `values="12 15 34"`
- ENTITY
  - `photo="MyPic"`
- ENTITIES
  - `photos="pic1 pic2"`
- ID
  - `ID = "P09567"`
- IDREF
  - `IDREF="P09567"`
- IDREFS
  - `IDREFS="A01 A02"`
- NOTATION
  - `FORMAT="TeX"`
- Name group
  - `coord="X"`

Default Attribute Values

- Can specify a default attribute value for when its missing from XML document, or state that value must be entered
  - **#REQUIRED** Must be specified
  - **#IMPLIED** May be specifed
  - **"*default*"** Default value if unspecified
  - **#FIXED** Only one value allowed

```
<!ATTLIST seqlist sepchar NMTOKEN #REQUIRED>
```

Parameter Entities

- Use parameter entities within DTD
  - `<!ENTITY % common "(para|list|table)">`

    ```
    <!ELEMENT chapter ((%common;)*, section*)>
    <!ELEMENT section (%common;)*>
    ```
  - Safest to include parentheses in entity definition and around entity reference

Conditional Sections

- INCLUDE and IGNORE declarations
  - `<![INCLUDE[ … ]]>`
  - `<![IGNORE[ … ]]>`
- Can be used as switch for including and/or excluding declarations
  -
```
<!ENTITY % variant "INCLUDE">

<![%variant;[
  <!ENTITY % Text "(#PCDATA|temp)">
]]>
<!ENTITY % Text "(#PCDATA)">
```

Internal DTD Definition

- Include in the DOCTYPE declaration
  - ```
    <!DOCTYPE MyDoc [
       <!-- DTD appears here -->
       <! … >
       <! … >
    ]>
    <!-- Rest of XML file -->
    ```
    ```

External DTD Definition

- Reference external DTD file as pathname in DOCTYPE declaration
  - ```
    <!DOCTYPE MyDoc
      SYSTEM "./MyDoc.dtd" [
      <!-- Extra declarations -->
      <! … >
      <! … >
    ]>
    <!-- Rest of XML file -->
    ```
  - Document specific declarations kept internally
    ```

External DTD Definition (2)

- Reference external DTD file as URL in DOCTYPE declaration
  - ```
    <!DOCTYPE MyDoc
      SYSTEM ”http://…/MyDoc.dtd" [
      <!-- Extra declarations -->
      <! … >
      <! … >
    ]>
    <!-- Rest of XML file -->
    ```
  - Document specific declarations kept internally
    ```

Designing a DTD

- Not trivial!
  - If an XML DTD needs to be changed - may have serious consequences on other s/w
  - Separation of interface and implementation
- Analogous to database schema design
- Need to consider
  - Granularity
  - Attributes versus Elements
  - Limitations of DTD declarations

Granularity of DTD

- **<PERSON>**
  <NAME>John Smith</NAME>
  </PERSON>

- **<PERSON>**
  <FORENAME>John</FORENAME>
  <SURNAME>Smith</SURNAME>
  </PERSON>

XML Schema

```
<?xml version = "1.0"?>
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
    targetNamespace="http://localhost:8080/wuye"
        xmlns=http://localhost:8080/wuye
            elementFormDefault="qualified">
  <xsd:element name= “product">
  <xsd:complexType>
  <xsd:sequence>
  <xsd:element name="name" type="xsd:string"/>
  <xsd:element name="price" type="xsd:integer"/>
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
  </xsd:schema>
```

Why XML Schema (why not just DTD)?

- More data types
  - Complex data types
- More like a database schema
- Will continue next time….