CSC207H: Software Design
Lecture 8

Wael Aboelsaadat

wael@cs.toronto.edu
http://ccnet.utoronto.ca/20075/csc207h1y/
Office: BA 4261
Office hours: R 5-7

Acknowledgement: These slides are based on material by Prof. Karen Reid

Programming/scripting languages technologies

> Regular Expressions

→ XML

→ Reflection

What is Extensible Markup Language(XML) ?

- NOT a markup language
- Meta-markup Language
- Set of very simple rules
- XML provides a uniform method for describing and exchanging structured data
- Describes structure and semantics, not formatting

The XML Rules….

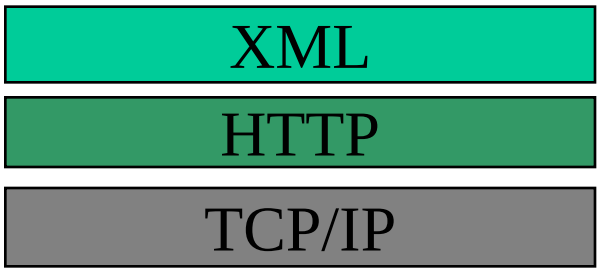
1. Single, unique root element
2. Matching open/close tags
3. Consistent capitalisation
4. Correctly nested elements (no overlapping elements)
5. Attribute values enclosed in quotes
6. No repeating attributes in an element

```
<?xml version="1.0"?>
<company id="4859">
    <name>3Months.com</name>
    <type>Web Development</type>
    <address>
        <street>Wakefield st</street>
        <city>Wellington</city>
        <country>New Zealand</country>
    </address>
</company>
```

Well Formed

Why XML is so powerful ?

- Provides international, vendor independent standard for describing information
- Any information – structured or unstructured



XML example: book meta data

```
<biobliography>
      <book>  <title> Foundations of DBs </title>
                  <author> Abiteboul </author>
                  <author> Hull </author>
                 <author> Vianu </author>
                  <publisher> Addison-Wesley </publisher>
                  ....
      </book>
      <book>   ... <editor> Chomicki </editor>... </book>     ...
</bibliography>
```

XML tags:
content, "semantic"

HTML vs. XML

```
<h1> Bibliography </h1>
<p> <i> Foundations of DBs</i>, Abiteboul, Hull, Vianu
      <br> Addison-Wesley, 1995
<p> <i> Logics for DBs and ISs </i>, Chomicki, Saake, eds.
      <br> Kluwer, 1998
```

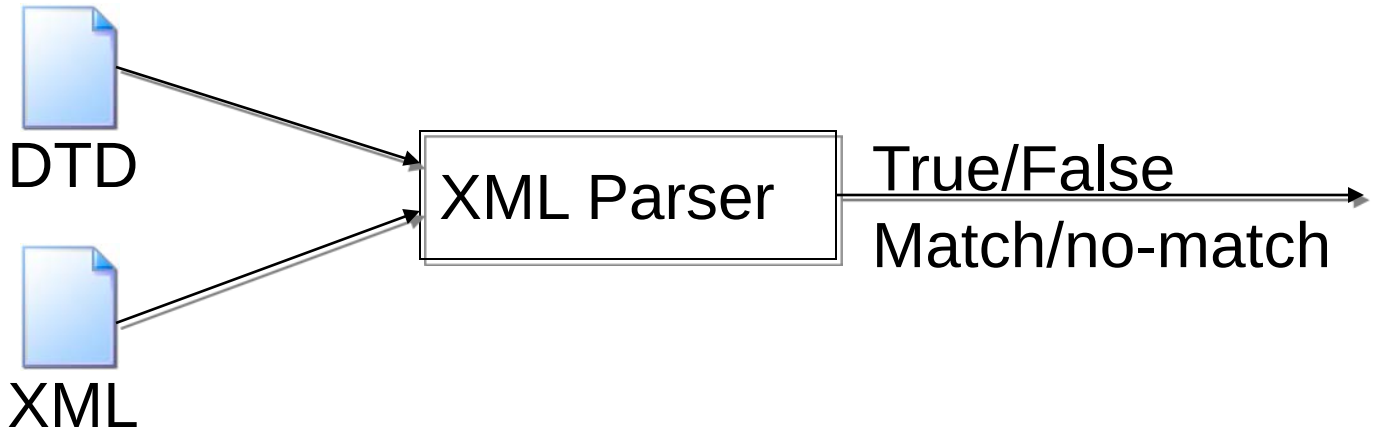
HTML tags:
presentation aspects, generic document structure

```
<biobliography>
    <book>  <title> Foundations of DBs </title>
              <author> Abiteboul </author>
              <author> Hull </author>
              <author> Vianu </author>
              <publisher> Addison-Wesley </publisher>
              ....
    </book>
    <book>   ... <editor> Chomicki </editor>... </book>     ...
</bibliography>
```

XML tags:
content, "semantic"

XML: what makes a valid one?

XML: what makes a valid one?



Document Type Definitions (DTDs)

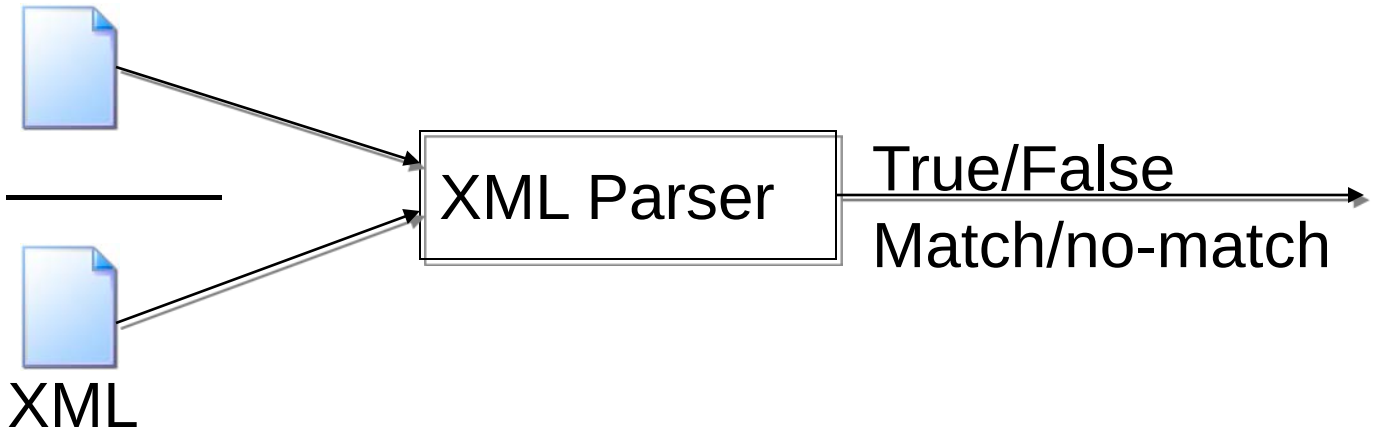
- <!ELEMENT *element-type content-model*>
  - Defines content model of an element type
  - Element-type is the name of the element (or tag)
  - Content-model is a regular expression defining structure of subelements

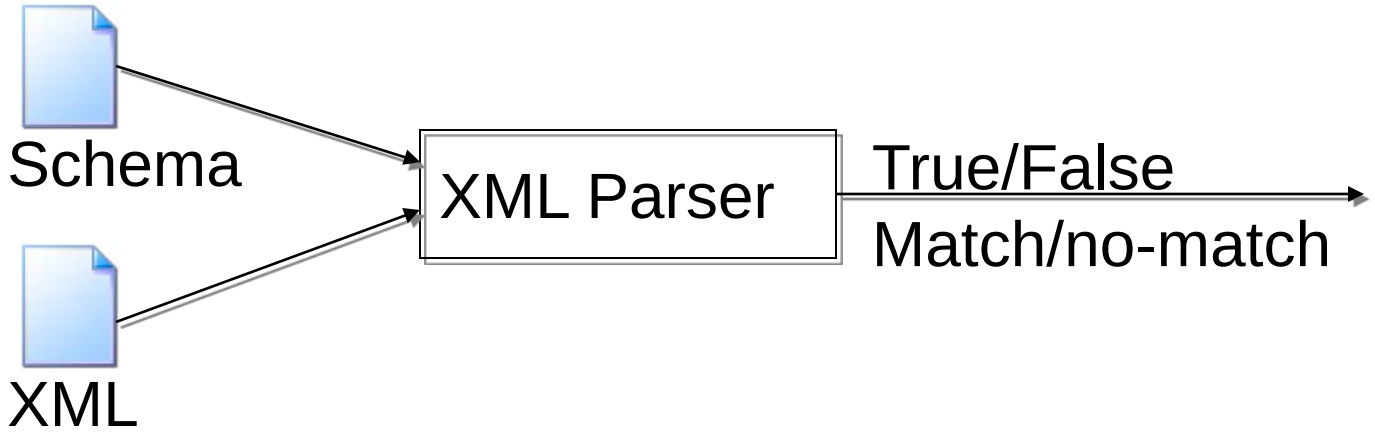
- <!ATTLIST *element-type attribute-name attribute-type*>
  - Defines for elements named *element-type* associated attributes and their types
  - Element-type is the name of the element (or tag)

A Sample DTD

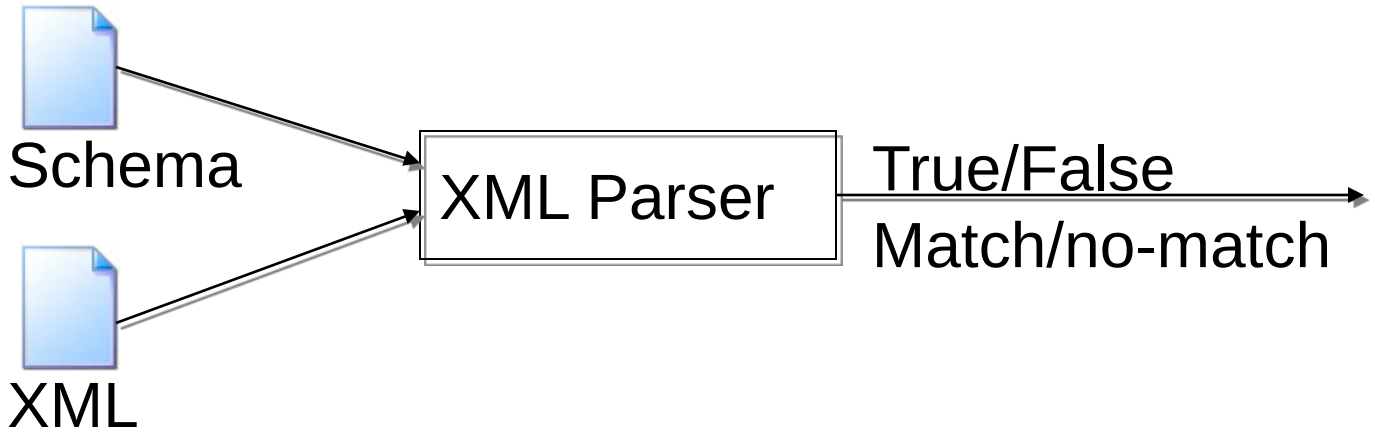
```
<!ELEMENT MYDOC
        (FILENO?,TITLE,AUTHORS?,APPEARED?,ABSTRACT,BODY)>
<!ELEMENT FILENO              (#PCDATA)>
<!ELEMENT AUTHORS             (AUTHOR)*>
...

<!ELEMENT IMAGE     EMPTY>
<!ATTLIST  IMAGE    TYPE    CDATA           #REQUIRED>
...
<!ELEMENT ABSTRACT
        (#PCDATA|ITEMIZE|CREF|REF|EQN|FOOTNOTE|IMAGE|P)*>
<!ELEMENT BODY                (DIV|P|ITEMIZE)*>
...
 <!ELEMENT DIV                (HEADER, (DIV|P|ITEMIZE|IMAGE)*)>
 <!ATTLIST  DIV                ID        CDATA         #REQUIRED
                               DEPTH   CDATA         #REQUIRED
                               R-NO      CDATA       #REQUIRED>
```

XML: what makes a valid one?

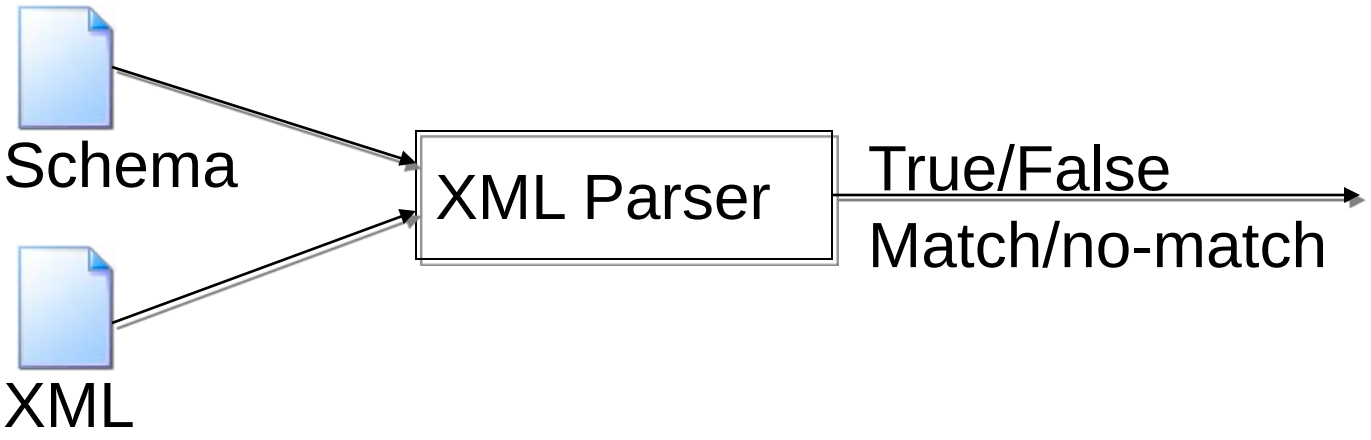




XML: what makes a valid one?



XML: what makes a valid one?



Document Object Model (DOM)

- Cross-language API for representing XML documents as trees
  - Easier to manipulate than strings or streams
  - But may require a lot of memory
- Several implementations in Java
  - This course uses org.jdom

Tree Structure

- The document:
  - <?xml version="1.0" encoding="UTF-8"?>
  - <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  - "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
  - <html>
  - <body>
  - <h1>Title</h1>
  - A word
  - </body>
  - </html>

DOM Rules

- Every document becomes an object of type Document
- This has a single child of type Element
  - The root element of the document
- Its children may be:
  - Other elements
  - Text objects
  - Other things that we won't worry about
- Note: white space is preserved
  - Like the new lines in the previous slide
- But comments are not

Using JDom

```
public static void main(String[] args) {
    try {
      String filename = args[0];
      // Build document tree
      SAXBuilder builder = new SAXBuilder();
      Document doc = builder.build(filename);

      // Show top-level elements (next slide)

    } catch (Exception e) {
      System.err.println(e);
    }
}
```

Build the DOM tree

Iterate over children

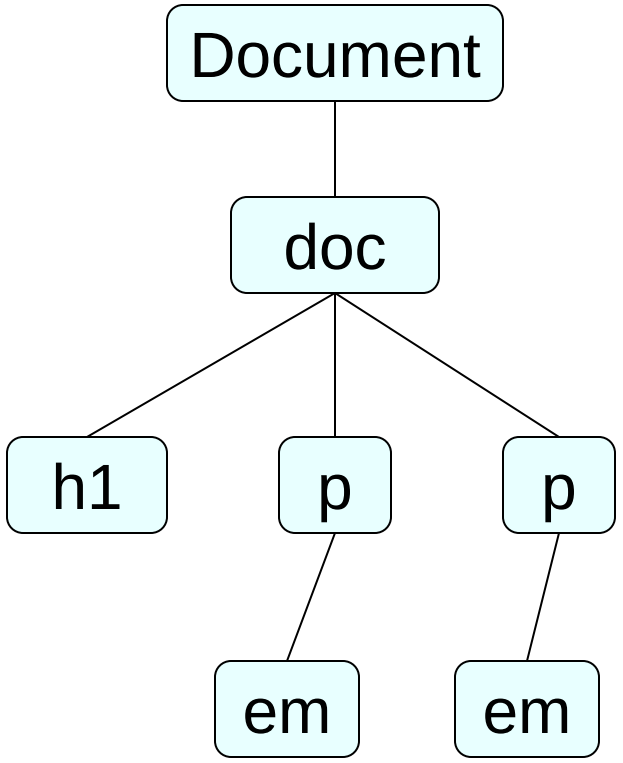
- // Show top-level elements
- Element root = doc.getRootElement();
- for (Element elt : root.getChildren()) {
- System.out.println(elt.getName());
- }

Get root element

Get all children (excluding text)

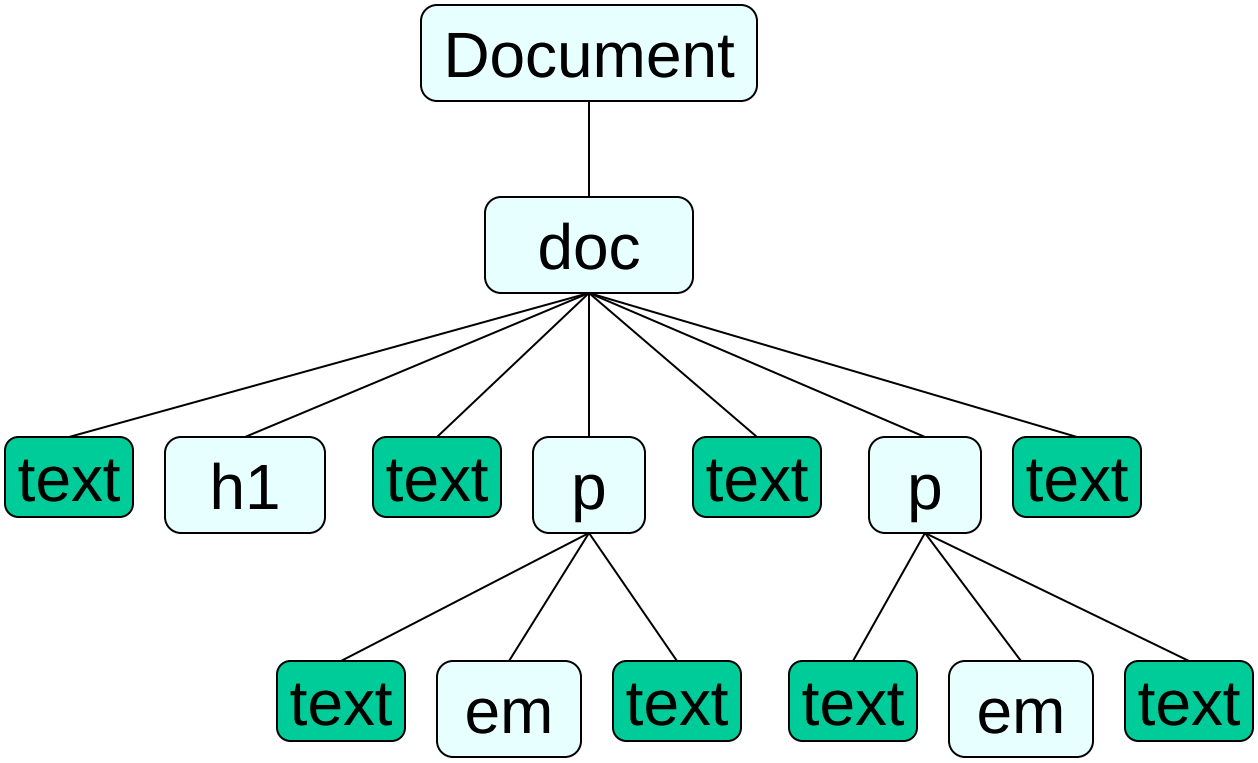
Input and output

- Input
- <?xml version="1.0" ?>
- <doc>
- <h1>First heading</h1>
- First
- paragraph.
- Second
- paragraph.
- </doc>



- Output
- h1
- p
- p

Content


Document
doc
text
h1
text
p
text
p
text
text
em
text
text
em
text

Showing Structure Recursively

public static void descend(Element current, int depth) {

```
  for (int i = 0; i < depth; ++i) {
    System.out.print(" ");
  }
  Element elt = (Element) current;
  System.out.println(elt.getName());
  Iterator ic = elt.getChildren().iterator();
  while (ic.hasNext()) {
    descend((Element) ic.next(), depth+1);
  }
}
```

A DOM Visitor

```
public abstract class DomVisitor {
        public DomVisitor() {}
        public void visit(Element root) {
      fDepth = 0;
      preRoot(root);
      atElement(root);
      recurse(root);
      postRoot(root);
    }
    protected void preRoot(Element root) {}
    protected void postRoot(Element root) {}
    protected void atElement(Element elt) {}
    protected void atText(Text text) {}
```

```
    protected void recurse(Element elt) {
      fDepth += 1;
      for (Object node : elt.getContent()) {
        if (node instanceof Element) {
          Element child = (Element) node;
          atElement(child);
          recurse(child);
        } else if (node instanceof Text) {
          atText((Text) node);
        }
      }
      fDepth -= 1;
    }
  }
```

Building an Attribute Inventory

- Want to find out which attributes appear with which elements in an XML file
- Create a DOM visitor that inspects each element's attributes
- Result is a map in which
  - Keys are element names (e.g. "h1")
  - Values are sets of attribute names (e.g. "align")
- Here we do not record the attribute values
  - Exercise: extend this visitor to inventory them as well

The Inventory Visitor

```
public class Inventory extends DomVisitor {
    public Inventory() {
        fSeen = new HashMap<String,
            Set<String>();
    }
    protected void preRoot(Element root) {
        fSeen.clear();
    }
    protected void atElement(Element elt) {
            …
        }
    protected Map<String, Set<String> fSeen;
}
```

```
protected void atElement(Element elt) {
  String eltName = elt.getName();
  Set<String> seen = fSeen.get(eltName);
  if (seen == null) {
    seen = new HashSet<String, Set<String>();
    fSeen.put(eltName, seen);
  }
  for (Attribute attr = elt.getAttributes()) {
    String attrName = attr.getName();
    seen.add(attrName);
  }
}
```

Input and Output

- <doc>
- <p align="left"
- role="lead">First.
- Second.
- <p align="right"
- font="em">Third.
- </doc>

- doc
- p
  - align
  - role
  - font

Trimming the Tree

- Can add or remove nodes in DOM tree
  - Be careful about deleting items in a list while iterating over that list
- Pattern: delete or move on
  - When an item is deleted, items above it bump down
  - So either delete *or* increment loop index

```
protected void atElement(Element elt) {
  List<Object> content = elt.getContent();
  int i = 0;
  while (i < content.size()) {
    Object node = content.get(i);
    boolean keep = true;
    if (node instanceof Text) {
      Text text = (Text) node;
      if (text.getText().trim().length() == 0) {
        keep = false;
      }
    }
    if (keep) {
      i += 1;
    } else {
      content.remove(i);
    }
  }
}
```

Python

- Like JDOM, Python's DOM library is derived from the W3C standard
  - Uses idiomatic Python instead of trying to be 100% compatible with standard
- In fact, Python has two DOM libraries
  - xml.minidom doesn't have everything
  - But it's fast

Example

```
import sys, xml.dom.minidom

def showTree(node, indent=0):
  print '  ' * indent + node.nodeName
  for child in node.childNodes:
    if child.nodeType == child.ELEMENT_NODE:
      showTree(child, indent+1)

for filename in sys.argv[1:]:
  doc = xml.dom.minidom.parse(filename)
  root = doc.documentElement
  showTree(root)
```

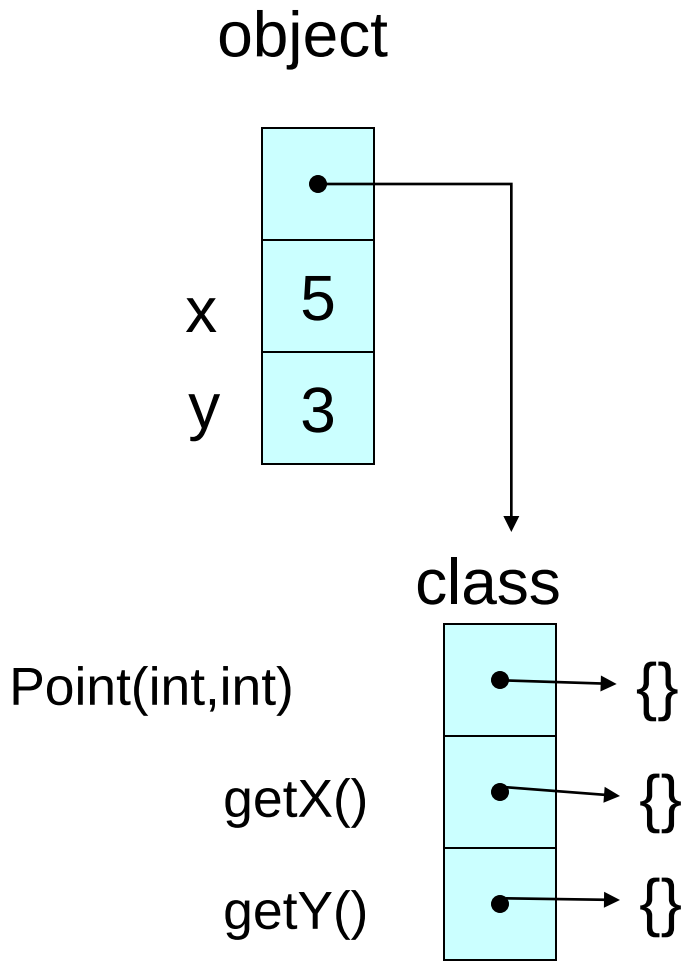
Reflection

Background

- Turing's great insight: programs are just another kind of data
  - Source code is text
  - Manipulate it line by line, or by parsing expressions
- Compiled programs are data, too
  - Integers and strings are bytes in memory that you interpret a certain way
  - Instructions in methods are just bytes too
- No reason why a program can't inspect itself

How objects work

```
class Point {
  public Point(int x, int y) {
    x = 5; Y = 10;
  }
  public int getX() {
          return x;
  }
  public int getY() {
          return y;
    }
  protected int x, y;
}
```



The class Class

- Instances of the class Class store information about classes
  - Class name
  - Inheritance
  - Interfaces implemented
  - Methods, members, etc.

- Can look up instances:
  - By name
  - From an object

http:///java.sun.com/j2se/1.5.0/docs/api/java/lang/Class.html

Showing a type

```
public static void showType(PrintStream out,
                                                String className)
            throws ClassNotFoundException
{
  Class thisClass = Class.forName(className);
  String flavour  = thisClass.isInterface() ? "interface"
                                                : "class";
  out.println(flavour + " " + className);
  Class parentClass = thisClass.getSuperclass();
  if (parentClass != null) {
    out.println(" extends " + parentClass.getName());
  }
  Class[] interfaces = thisClass.getInterfaces();
  for (Class interf : interfaces) {
    out.println("  implements " + interf.getName());
  }
}
```

Output for type example

```
class java.lang.Object

class java.util.HashMap
 extends java.util.AbstractMap
  implements java.util.Map
  implements java.lang.Cloneable
  implements java.io.Serializable

class Point
 extends java.lang.Object
```

Examining class contents

```
public static void showContents(PrintStream out,
                                boolean hideObject,
                                String name)
    throws ClassNotFoundException {

  Class cls = Class.forName(name);
  out.println(name);
  showMembers(out, hideObject, name + " fields",
          cls.getFields());
  showMembers(out, hideObject, name + " constructors",
          cls.getConstructors());
  showMembers(out, hideObject, name + " methods",
          cls.getMethods());
}
```

Examining class contents

```
public static void showMembers(PrintStream out,
                               boolean hideObject,
                    String title,
                               Member[] members) {
  out.println("  " + title);
  for (Member mem : members) {
    if (mem.getDeclaringClass() == Object.class) {
      if (hideObject) {
        continue;
      }
    }
    out.println("\t" + mem);
  }
}
```

- Point *(somewhat edited)*
- Point fields
- Point constructors
- public Point(java.lang.String,int,int)

- public Point(int,int)
- Point methods
- public java.lang.String Point.toString()
- public java.lang.String Point.getName()
- public void Point.setName(java.lang.String)
- public int Point.getX()
- public void Point.setX(int)
- public int Point.getY()
- public void Point.setY(int)

Getting at members

- How to access members of a specific object?
  - Without making raw pointers into memory part of the language
    - (Raw pointers are a rich source of errors in C/C++)

- Introduce a class Field
  - Encapsulates access to a particular field of instances of a class
  - Knows "where the field is" in objects of that class
  - Use its get() and set() methods to inspect and modify the object

Examining fields

```
public static void main(String[] args) {
  PublicPoint p
    = new PublicPoint("center", 3, 3);

  showField(System.out, p, "fName");
  showField(System.out, p, "fX");
  showField(System.out, p, "fY");
  showField(System.out, p, "fZ");
}
```

```
public static void showField(PrintStream out,
        Object obj, String fieldName) {
    try {
        Class cls    = obj.getClass();
        Field field  = cls.getField(fieldName);
        Object value = field.get(obj);
        out.println(fieldName + ": " + value);
    }
    catch (NoSuchFieldException e) {
        System.err.println(e);
    }
    catch (IllegalAccessException e) {
        System.err.println(e);
    }
}
```

Output

- fName: center
- fX: 3
- fY: 3
- java.lang.NoSuchFieldException: fZ

```
public static void showMethods(
                                PrintStream out, Object obj)
        throws NoSuchMethodException,
          IllegalAccessException,
              InvocationTargetException {


  Class cls = obj.getClass();
  out.println(cls.getName());
  for (Method meth : cls.getMethods()) {
    if (meth.getDeclaringClass() == cls) {
      showMethod(out, meth);
    }
  }
}
```

Calling methods

1. Look up a method based on its signature: the name and list of parameter types

2. Specify signature as a comma-separated list of `Class` objects
   - Specifies the types of arguments
   - Special values for types like int and boolean

3. Call the method, passing in parameters and capturing return value

```
Class mysClass = Class.forName("Mystery");
Object o = mysClass.newInstance();

Method m = mysClass.getMethod("euclidean",
  Double.TYPE, Double.TYPE);
double result = (Double) m.invoke(o,
 new Double(5.0), 12.0);

Method m2 = mysClass.getMethod("play",
    Class.forName("java.lang.String"), String.class);

m2.invoke(o, "Che", "Karl");
```

Python: Built-in methods for Reflection

getattr(object, name)

- returns the value of the attribute name

hasattr(object, name)

- returns true if the object has an attribute by the given name

setattr(object, name, value)

- assign value to attribute name

type(object)

- returns the type of the object

Invoking a method given an object

```
class C:
    def __init__(self, val=-1):
        self.x = val
    def foo(self):
        print self.x
if __name__ == "__main__":
    c = C(10)
    f = getattr(c, "foo")
    f()
    f = getattr(c, "x")
    print f
```

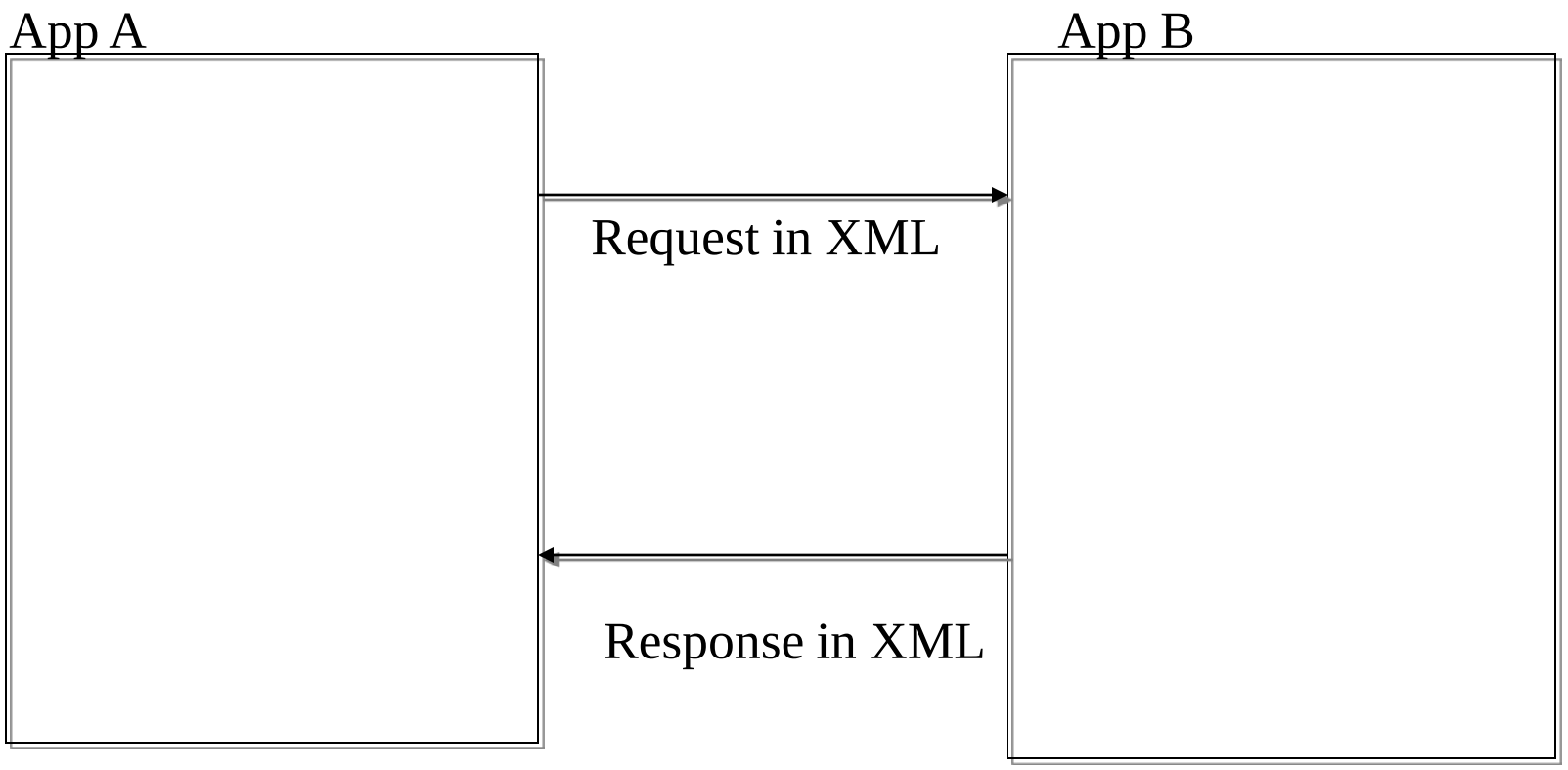
Reflection in Python

- Special attributes in a Python object:
  - >>> c = C(20)
  - >>> c.__class__
  - <class __main__.C at 0x53870>
  - >>> c.__dict__
  - {'x': 20}
  - >>> c.__module__
  - '__main__'
  - >>> C.__name__
  - 'C'
  - >>> c.__class__.__name__
  - 'C'

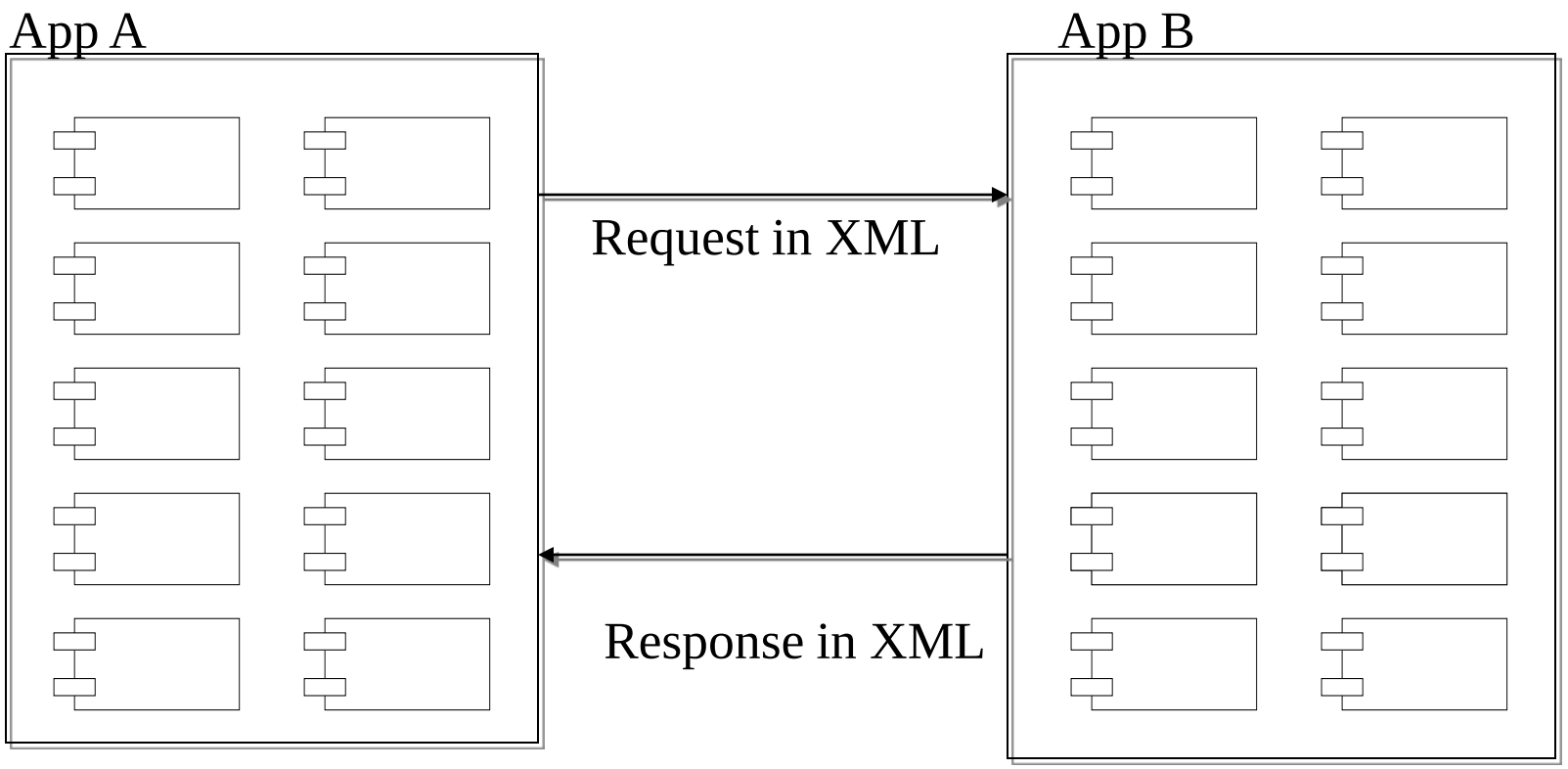
Key points

- There is no magic
  - A class is just a data structure
  - A method is just a data structure, too
- It just happens to contain bytes that look like instructions for the interpreter
- The call stack is another data structure
  - With libraries to give you access to it at runtime
- Many programming tools make use of reflection
  - We'll see one in the next lecture

Impact of XML and Reflection on Software Design



Impact of XML and Reflection on Software Design



e.g. http://www.google.ca/ig?hl=en

Tools in a Software House

- ✓ Programming Languages (e.g. java, C/C++)
- ✓ Scripting Languages (e.g. python)
- • Integrated Development Environment (IDE) App
- • Profiling Tools
- ✓ Version Control App (e.g. cvs)
- ✓ Quality Assurance Framework (e.g. junit)
- ✓ Software Build Management Framework
- • Requirements/Feature Tracking App

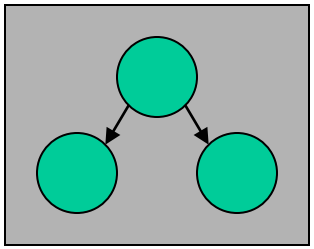
→ Variance Tracking App

- ✓ Architecture Tools

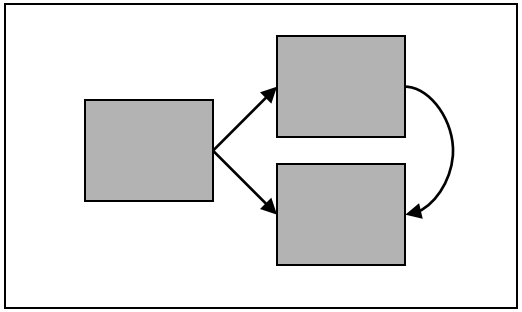
Types of Software Testing

- *Component/unit* testing
  - Individual classes or types

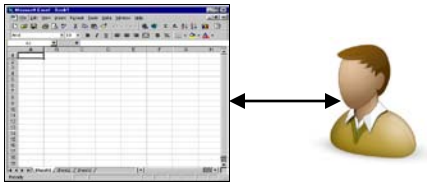
- *Development* testing
  - Also called DIT (development integration testing)
  - Group of related classes or types



- *System* testing
  - Also called SIT (system integration testing)
  - Interaction between classes



- *User Acceptance* testing
  - Also called UAT
  - Interaction between user and program interface



Software Testing Lifecycle

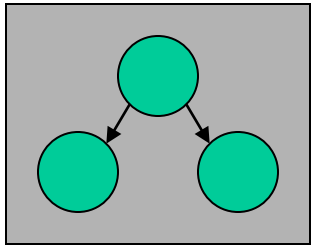
Unit Testing → followed by → DIT → followed by → SIT → followed by → UAT

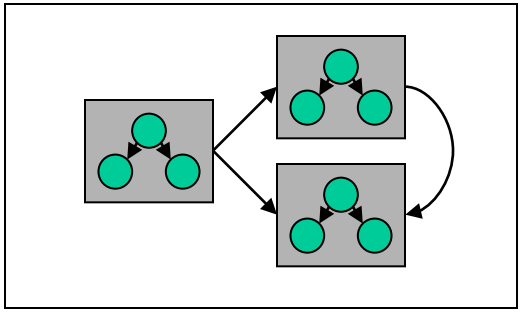
During coding of a component
By developers

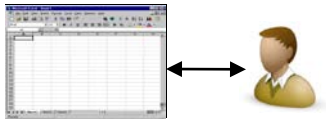
After integration of various components
By developers/QA

After building the whole application
By QA mostly

After alpha release
By QA







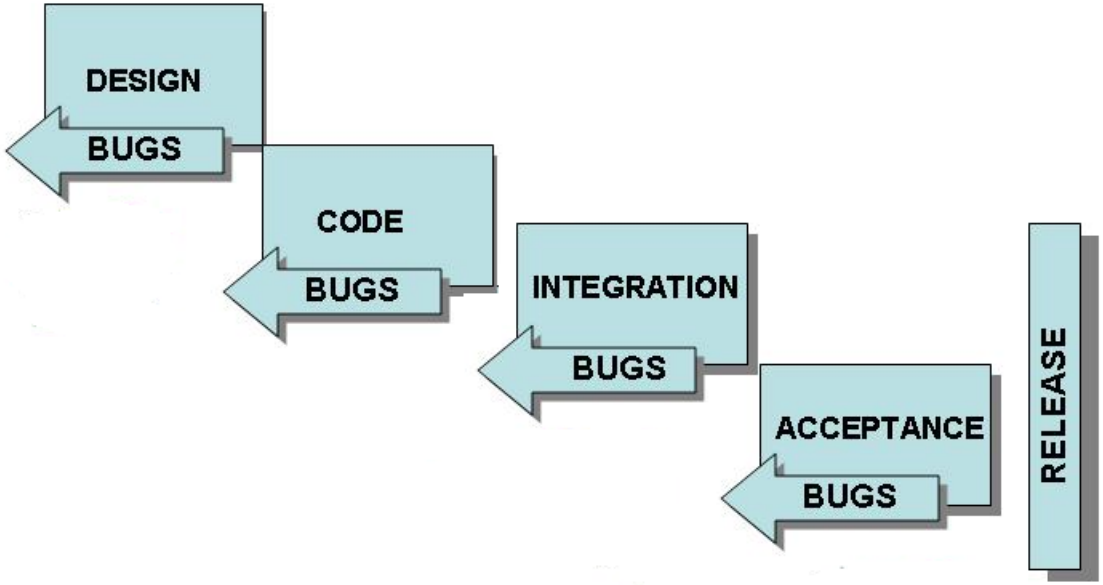
Variance Tracking App

- Variance lifecycle
  - Status: *open*, *assigned*, *in-progress*, *on-hold*, *waiting retest*, *closed*, *rejected*
  - Workflow: *defect found*, *defect verified*, *being fixed*, *ready for retesting*, *defect fixed*, *defect rejected*

- Variance severity
  - *No impact*, *low impact*, *medium impact*, *high impact*, *critical*
  - 1 .. 6

Variance Tracking App

- Variance source



Variance Tracking App

- Variance reporting
  - How to reproduce?
  - Proof (screen capture, audio recording, …)
  - E.g.
    - Step 1: login to website
    - Step 2: click on reset button
    - Step 3: fill name field with value “john”
    - Step 4: click on submit button

Variance Tracking App

- Demo
  - http://ontime3.axosoft.com/UOTCSC207
  - http://www.drproject.org/