XMLTree Methods
Methods for **XMLTree**

- All the methods for **XMLTree** are *instance methods*, i.e., you call them as follows:
  
  ```
  t.methodName(arguments)
  ```

  where `t` is an initialized variable of type **XMLTree**
Methods for **XMLTree**

- All the methods for **XMLTree** are *instance methods*, i.e., you call them as follows:

  ```
  t.methodName(arguments)
  ```

  where `t` is an initialized variable of type `XMLTree`.

  `t` is called the *receiver* of the call; for all instance methods, the corresponding *distinguished formal parameter* implicitly has the name `this`. 
Implementations of **XMLTree**

- There are two different **classes** that **implement** the **XMLTree** **interface** contract, and you may use either one: **XMLTree1** or **XMLTree2**

- This choice is made when you initialize a variable of type **XMLTree**, where you must use the name of one of these implementation classes as the name of the constructor
Implementations of XMLTree

• There are two different classes that implement the XMLTree interface contract, and you may use either one: XMLTree1 or XMLTree2

• This choice is made when you initialize a variable of type XMLTree. You might use the name of one of these implementation classes as the name of the constructor.

The behavior of an XMLTree does not depend on which implementation you choose; this is a key benefit of design-by-contract!
Interface and Implementing Classes

XMLTree

XMLTree1 implements XMLTree

XMLTree2 implements XMLTree
The **interface** `XMLTree` has method signatures and contracts for methods.
Interface and Implementing Classes

The class XMLTree1 has method bodies; similarly XMLTree2.
Interface and Implementing Classes

The method bodies in XMLTree1 implement the method contracts in XMLTree.
The method bodies in **XMLTree2** implement the method contracts in **XMLTree**.
Mathematical Model

• The value of an XMLTree variable is modeled as a tree of nodes whose labels are explained in the previous set of slides.

• Note that this model is described informally, though it could be formalized into mathematical notation (which we will not do here).
Constructors

• There are two *constructors* for each implementation class
• The name of the constructor is the name of the implementation class
• Constructors differ only in their parameters
• For *XMLTree*, we will use only the constructor that takes one *String* parameter, either:
  – The name of an XML file on your computer
  – The URL of an XML file or an XML source on the web
Constructors

• A constructor call has the keyword `new` before the constructor name and is an expression, e.g.:
  ```
  new XMLTree1("foo.xml")
  ```

• The value of this expression is determined by the contract for the constructor
  – In this case, the contract says the value is an `XMLTree` corresponding to the XML document named by the `String` parameter
<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>XMLTree t = new XMLTree1(&quot;foo.xml&quot;);</code></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>State</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
</tr>
</tbody>
</table>
| $\text{XMLTree } t = \text{new XMLTree1("foo.xml");}$ | $t = \text{[tree from file "foo.xml"]}$

See the slides on the **XMLTree** model for a description of the tree that arises from an XML document.
label

String label ( )

• Returns the label of the root of this.
• Ensures:

\[
\text{label} = \text{[the label of the root of this (not including } < > \text{ for tags)]}
\]
Example: Label is a Tag

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>t = [tree for book XML example]</code></td>
<td></td>
</tr>
<tr>
<td><code>String s = t.label();</code></td>
<td></td>
</tr>
</tbody>
</table>
Example: Label is a Tag

State

\[ t = [\text{tree for book XML example}] \]

String \( s = t.\text{label}(); \)
Example: Label is a Tag

```
printISBN → 978-1-112-06331-6
webISBN → 1-118063-31-7
pubDate → Dec 20 2011
```

```
t = [tree for book XML example]
```

```
String s = t.label();
```

```
t = [unchanged]
s = "book"
```
Example: Label is Not a Tag

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>t = [tree rooted at title content in XML example]</code></td>
<td></td>
</tr>
<tr>
<td><code>String s = t.label();</code></td>
<td></td>
</tr>
</tbody>
</table>
Example: Label is Not a Tag

```
Java for Everyone: Late Objects
```

```
String s = t.label();
```

```
t = [tree rooted at title content in XML example]
```
Example: Label is Not a Tag

\[ t = \text{[tree rooted at title content in XML example]} \]

\text{String } s = t\text{.label();}

\text{t = [unchanged]}
\text{s = "Java for Everyone: Late Objects"}
isTag

boolean isTag( )

• Returns whether the label of the root of the node of 
  this is a tag.

• Ensures:

  isTag = [the label of the root of this is a tag]
Example: Label is a Tag

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>t = [tree for book XML example]</code></td>
<td></td>
</tr>
<tr>
<td><code>boolean b = t.isTag();</code></td>
<td></td>
</tr>
</tbody>
</table>
Example: Label is a Tag

printISBN → 978-1-118-06331-6
webISBN → 1-118063-31-7
pubDate → Dec 20 2011

State

t = [tree for book XML example]

boolean b = t.isTag();
### Label is a Tag

<table>
<thead>
<tr>
<th>PrintISBN</th>
<th>978-1-118-06331-6</th>
</tr>
</thead>
<tbody>
<tr>
<td>webISBN</td>
<td>1-118063-31-X</td>
</tr>
<tr>
<td>pubDate</td>
<td>Dec 20 2011</td>
</tr>
</tbody>
</table>

**State**

```java
boolean b = t.isTag();
```

```java
{unchanged}
```
Example: Label is Not a Tag

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean ( b = t.isTag(); )</td>
<td>( t = [\text{tree rooted at title content in XML example}] )</td>
</tr>
</tbody>
</table>
**Java for Everyone:** Late Objects

<table>
<thead>
<tr>
<th>Example: Label is Not a Tag</th>
</tr>
</thead>
</table>

```
t = [tree rooted at title content in XML example]

boolean b = t.isTag();
```
<table>
<thead>
<tr>
<th>Java for Everyone: Late Objects</th>
</tr>
</thead>
</table>

**Example: Label is Not a Tag**

<table>
<thead>
<tr>
<th><strong>State</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><em>t = [tree rooted at title content in XML example]</em></td>
</tr>
</tbody>
</table>

```java
boolean b = t.isTag();
```

<table>
<thead>
<tr>
<th><strong>State</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><em>t = [unchanged]</em></td>
</tr>
<tr>
<td><em>b = false</em></td>
</tr>
</tbody>
</table>
hasAttribute

**boolean hasAttribute(String name)**

- Returns whether the root tag of `this` has an attribute called `name`.
- Requires:
  
  ```
  [label of root of `this` is a tag]
  ```
- Ensures:

  ```
  hasAttribute = [label of root of `this` has an attribute called `name`]
  ```
### Example: Has One

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>t = [tree for book XML example];</code></td>
<td><code>boolean b = t.hasAttribute(&quot;pubDate&quot;);</code></td>
</tr>
</tbody>
</table>
Example: Has One

<table>
<thead>
<tr>
<th>Print ISBN</th>
<th>978-1-118-06331-6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web ISBN</td>
<td>1-118063-31-7</td>
</tr>
<tr>
<td>Pub Date</td>
<td>Dec 20 2011</td>
</tr>
</tbody>
</table>

$t = \text{[tree for book XML example]}$

```java
boolean b = 
    t.hasAttribute("pubDate");
```
### Has One

<table>
<thead>
<tr>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t = [\text{tree for book XML example}]$</td>
</tr>
</tbody>
</table>

```java
boolean b = 
    t.hasAttribute("pubDate");
```

```
t = [unchanged]
b = true
```
## Example: Has None

<table>
<thead>
<tr>
<th><strong>Code</strong></th>
<th><strong>State</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( t = \text{[tree for book XML example]} )</td>
</tr>
<tr>
<td>boolean ( b = )</td>
<td></td>
</tr>
<tr>
<td>( t \text{.hasAttribute(&quot;fooBar&quot;);} )</td>
<td></td>
</tr>
</tbody>
</table>
Example: Has None

```java
boolean b = t.hasAttribute("fooBar");
```

State:

```
t = [tree for book XML example]
```
Example: Has None

```java
boolean b =
t.hasAttribute("fooBar");
```

<table>
<thead>
<tr>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>t = [tree for book XML example]</code></td>
</tr>
<tr>
<td><code>t = [unchanged]</code></td>
</tr>
<tr>
<td><code>b = false</code></td>
</tr>
</tbody>
</table>
attributeValue

String attributeValue(String name)

• Returns the value associated with the attribute of the root tag of \textit{this} called \texttt{name}.

• Requires:

\hspace{1cm} \texttt{[label of root of this is a tag and it has an attribute called name]} 

• Ensures:

\hspace{1cm} \texttt{attributeValue} = \texttt{[value associated with attribute called name of root tag of this]}
### Example

<table>
<thead>
<tr>
<th><strong>Code</strong></th>
<th><strong>State</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>[\text{t} = \text{[tree for book XML example]}]</td>
<td></td>
</tr>
<tr>
<td>\text{String v = t.attributeValue(&quot;pubDate&quot;);}</td>
<td></td>
</tr>
</tbody>
</table>
String v =
t.getAttributeValue("pubDate");
Example

```java
String v = t.getAttributeValue("pubDate");
```

State

<table>
<thead>
<tr>
<th>t = [tree for book XML example]</th>
</tr>
</thead>
<tbody>
<tr>
<td>String v = &quot;Dec 20 2011&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>t = [unchanged]</th>
</tr>
</thead>
<tbody>
<tr>
<td>v = &quot;Dec 20 2011&quot;</td>
</tr>
</tbody>
</table>
**numberOfChildren**

```plaintext
int numberOfChildren()

• Returns the number of subtrees of the root of this.

• Requires:
  [label of root of this is a tag]

• Ensures:
  numberOfChildren = [the number of subtrees of the root of this]
```
**Example**

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
| \[
t = [\text{tree for book XML example}]
\] | \[
t = [\text{tree for book XML example}]
\] |
| \[
\text{int } n =
\text{t.numberOfChildren();}
\] | \[
\text{int } n =
\text{t.numberOfChildren();}
\] |
**Example**

```java
int n = t.numberOfChildren();
```

**State**

- printISBN → 978-1-118-06331-6
- webISBN → 1-118063-31-7
- pubDate → Dec 20 2011

```xml
<book>
  printISBN → 978-1-118-06331-6
  webISBN → 1-118063-31-7
  pubDate → Dec 20 2011
</book>
```
Example

```java
int n = t.numberOfChildren();
```

State

```
t = [tree for book XML example]
```

```
t = [unchanged]
n = 3
```
XMLTree child(int k)

• Returns the $k$-th subtree of the root of \textit{this}.

• Requires:
  
  \begin{align*}
  \text{[label of root of \textit{this} is a tag and } & \quad 0 \leq k < \text{ number of subtrees of the root of \textit{this}]}
  \end{align*}

• Ensures:

  \begin{align*}
  \text{child} = \text{[the } k\text{-th subtree of the root of \textit{this}]}
  \end{align*}
### Example

<table>
<thead>
<tr>
<th><strong>Code</strong></th>
<th><strong>State</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>XMLTree st =</td>
<td>$t = [\text{tree for book XML example}]$</td>
</tr>
<tr>
<td>t.child(1);</td>
<td></td>
</tr>
</tbody>
</table>
XMLTree st = t.child(1);
Example

```
Example
```

```
XMLTree st =
t.child(1);
```

```
t = [tree for book XML example]
```

```
XMLTree st =
t.child(1);
```

```
t = [unchanged]
```  
```
st = [tree rooted at title tag]
```
Example

```
XMLTree st =
t. child(1);
```

$t = [\text{unchanged}]$

$st = [\text{tree rooted at title tag}]$

Java for Everyone: Late Objects

- printISBN → 978-1-118-06331-6
- webISBN → 1-118063-31-7
- pubDate → Dec 20 2011
Complex Expressions

• Continuing code from the previous example, this expression has the same type and value as `st.child(0):`:
  `t.child(1).child(0)`

• And this expression has what type and what value?
  `t.child(1).child(0).label()`
Complex Expressions

• Continuing from the previous example, this expression has the same type and value as `st.child(0)`:
  ```java
t.child(1).child(0)
  ```
• And this expression has what type and value?
  ```java
t.child(1).child(0).label()
  ```
The type is **String**, the return type of the `label` method; the value is **"Java for Everyone: Late Objects"**.
An Aside: Iterators and Iterables

• An *iterator* lets you easily “visit” all members of a “collection” of things (without changing them while visiting them)

• A “collection” of things you can iterate on is called *iterable*

• The *collection classes* of the Java library and the OSU CSE components library have methods to give you an iterator for the corresponding collection and thus are iterable
An Aside: Iterators and Iterables

• An **iterator** lets you easily “visit” all members of a “collection” of things (without changing them while visiting them)

• A “collection” of things you can iterate on is called **iterable**

• The **collection classes** of the Java library and the OSU CSE components library have methods to give you an iterator for the corresponding collection and thus are iterable

For now, we’ll not further elaborate what is meant by a “collection”; it’s what you probably think it is.
Example Code With Iterable

• Suppose `dictionary` is some iterable collection of, say, `Strings`

• This code “does something” with each `String` in the collection

```java
for (String word : dictionary) {
    // do something with word
}
```
Example Code With Iterable

• Suppose `dictionary` is some iterable collection of, say, `Strings`

• This code “does something” with each `String` in the collection

```java
for (String word : dictionary) {
    // do something with word
}
```

This is called a `for-each loop`
attributeNames

Iterables<String> attributeNames()

• Returns an Iterable<String> of the attribute names of the root of this.
• Requires:
  \[\text{label of root of this is a tag}\]
• Ensures:
  \[\text{attributeNames = [an Iterable<String> of the attribute names of the root of this]}\]
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Iterable&lt;String&gt; it = t.attributeNames();</code></td>
<td><code>t = [tree for book XML example]</code></td>
</tr>
</tbody>
</table>
Example

```
t = [tree for book XML example]

Iterable<String> it =
   t.attributeNames();
```
### Example

```java
// Example
Iterable<String> it = t.attributeNames();
t = [tree for book XML example]

<table>
<thead>
<tr>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>t = [unchanged]</td>
</tr>
<tr>
<td>it = [iterable for 3 Strings]</td>
</tr>
</tbody>
</table>
```

State table:
- PrintISBN: 978-1-118-06331-6
- PubDate: Dec 20 2011

Diagram:
- `<book>` tag
- Attributes: printISBN, webISBN, pubDate
- Values: 978-1-118-06331-6, 1-118063-31-7, Dec 20 2011
Iterating Over Attribute Names

• To iterate over the attributes of the root of an XMLTree there is no need to declare an Iterable.

• This code “does something” with each attribute name (String) of the root of XMLTree t:

```java
for (String name : t.attributeNames()) {
    // do something with attribute name
}
```
void display() 

• Displays \textit{this} in a new window.

• Ensures:

\[\textit{this} \text{ is displayed in a new window}\]
String toString()?

• Returns an XML string representation of this.
• Ensures:

    toString = [an XML string representation of this]
**toString**

```java
String toString()
```

- Returns an XML string representation of **this**.
- Ensures:
  
  ```java
  toString = [a representation of this]
  ```

Equivalent to the content of an XML file that, if identified in the constructor for `XMLTree`, would result in the `XMLTree this`. 
An Immutable Type

• Observation: no method changes the value of an `XMLTree` variable!
  – Once an `XMLTree` variable is initialized by assigning it a value (e.g., the result of a constructor call), its value cannot be changed except by assigning something else to it

• This kind of type is called `immutable`
  – More details later…
Resources

• OSU CSE Components API: XMLTree