

# 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 instance of `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`

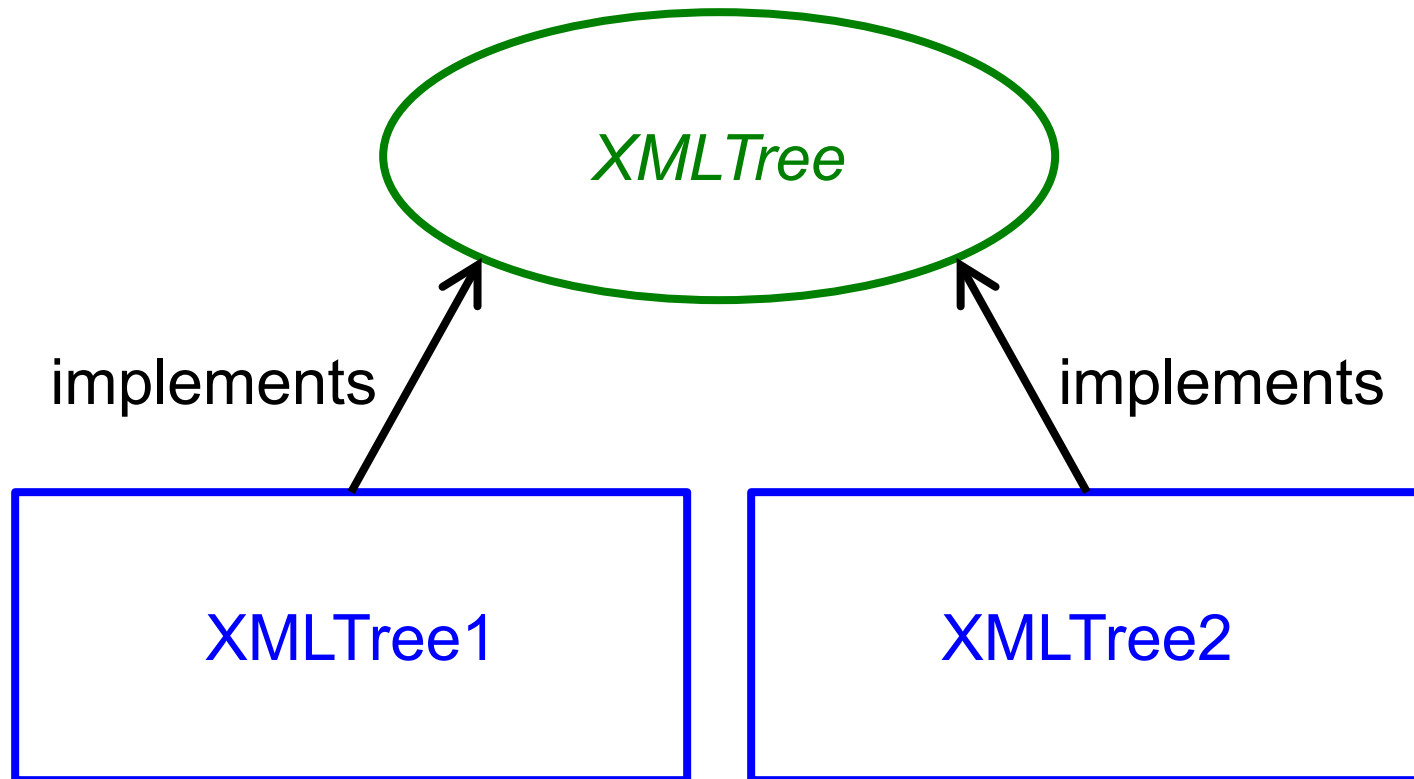
- There are two different **classes** that **implement** the `XMLTree` **interface** contract, and you may use either one: `XMLTree1` or `XMLTree2`
- This choice is made when you initialize a variable of type `XMLTree`, where you must use the name of one of these implementation classes as the name of the constructor

# Implementations of `XMLTree`

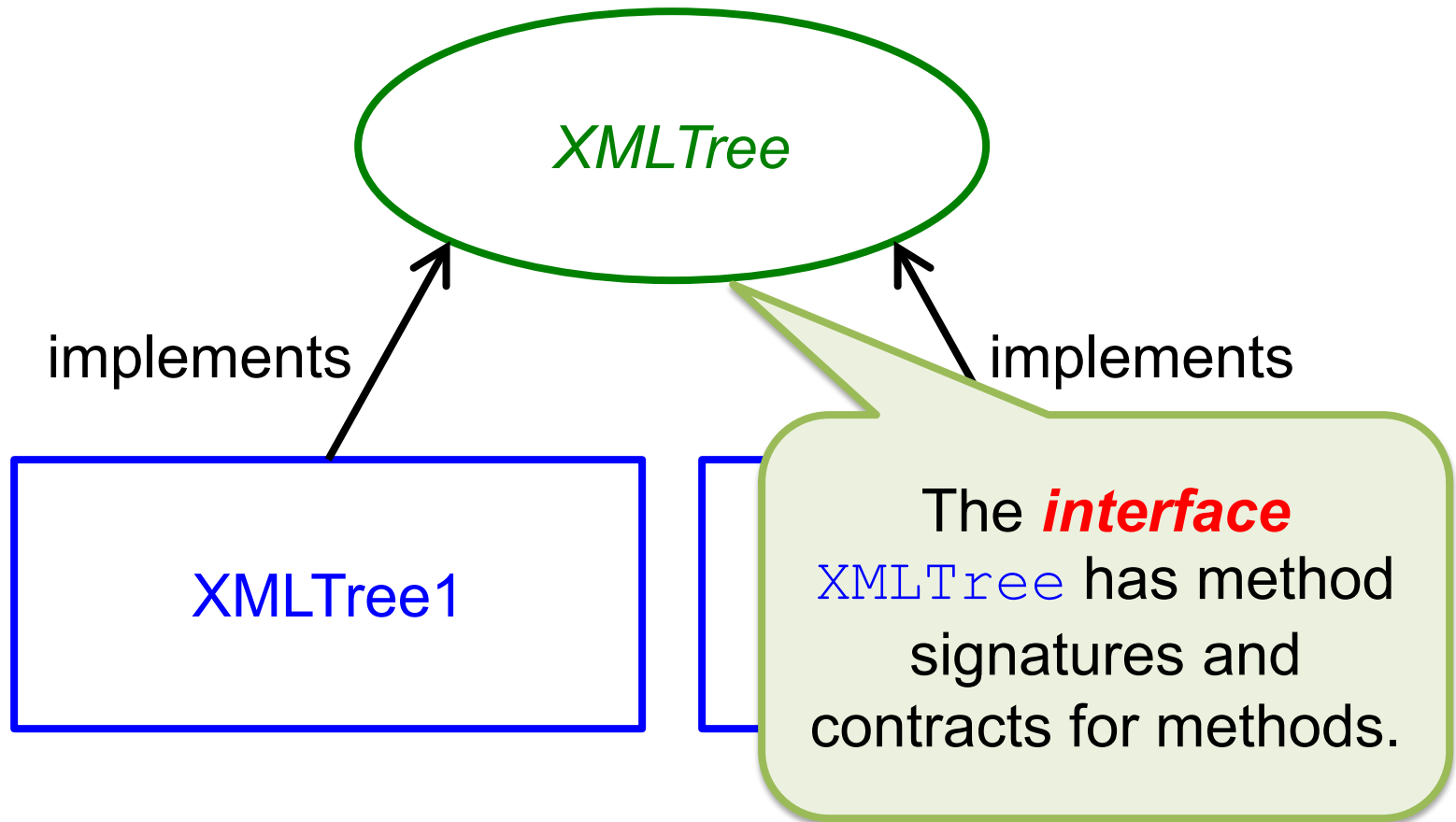
- There are two different **classes** that **implement** the `XMLTree` **interface** contract, and you may use either one: `XMLTree1` or `XMLTree2`
- This choice is made when you initialize a variable of type `XMLTree` to use the name of the implementation constructor

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

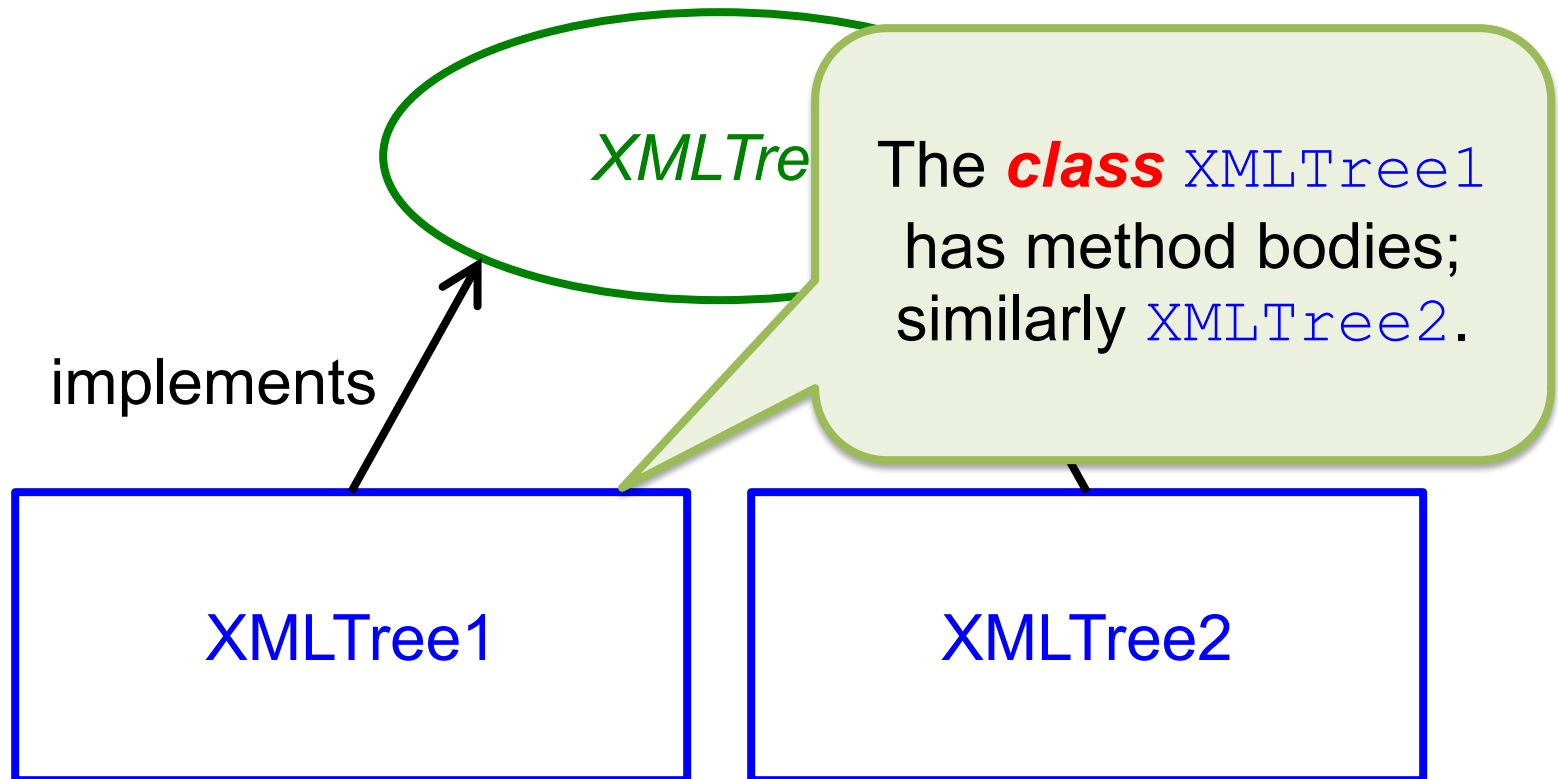
# Interface and Implementing Classes



# Interface and Implementing Classes

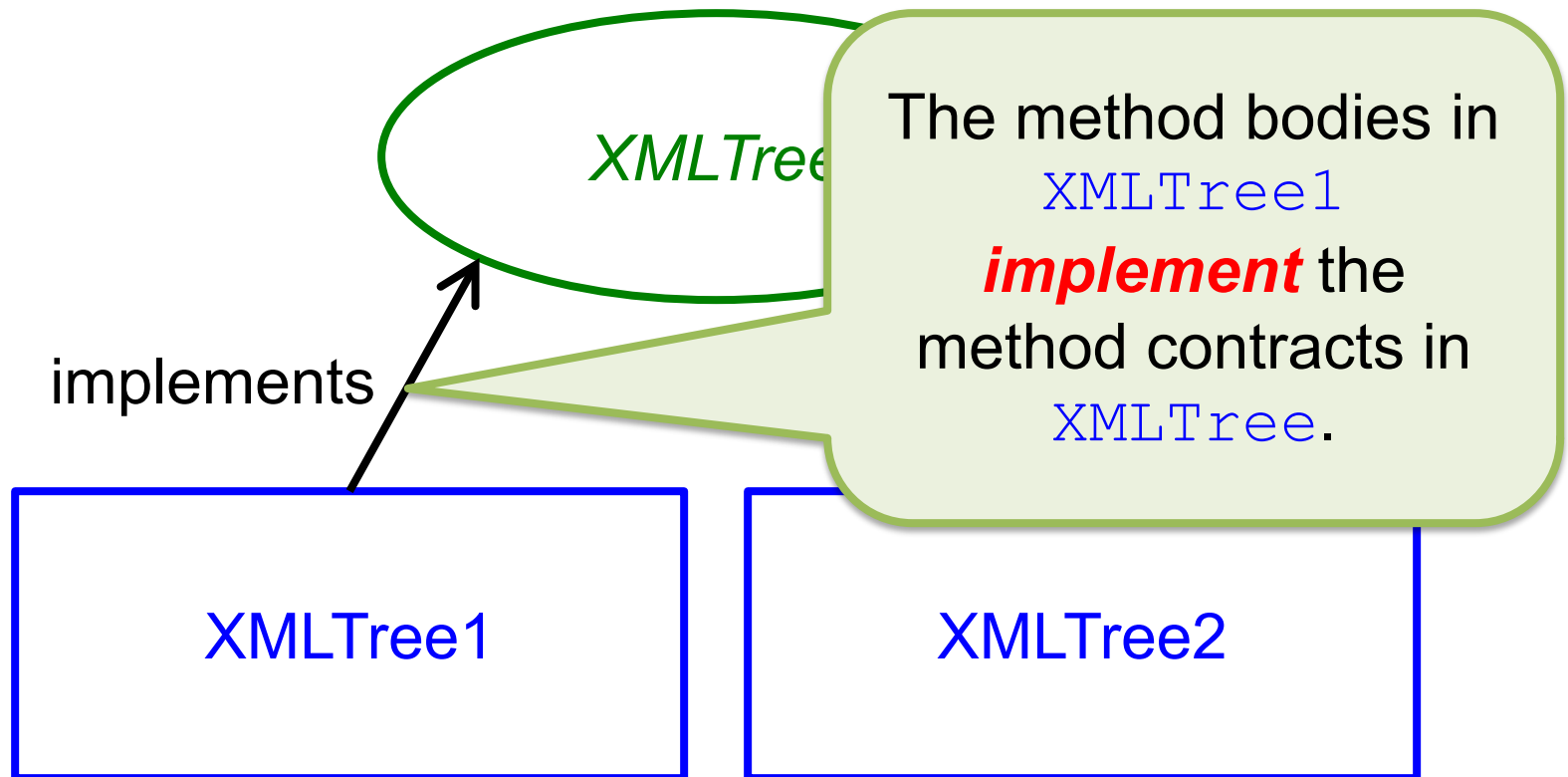


# Interface and Implementing Classes

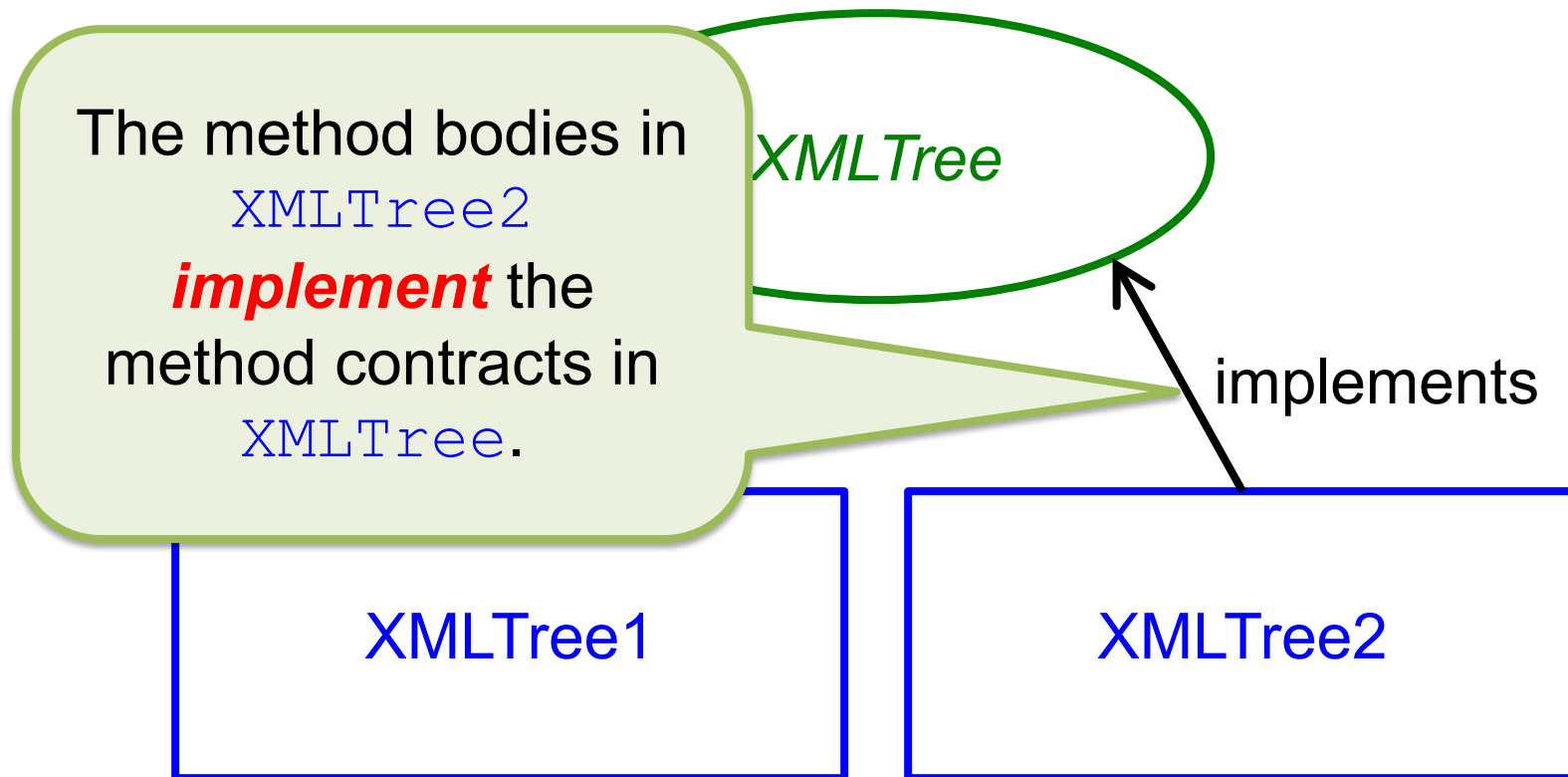




# Interface and Implementing Classes



# Interface and Implementing Classes



# 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

# Example

<b>Code</b>	<b>State</b>
<pre>XMLTree t = new     XMLTree1 ("foo.xml");</pre>	

# Example

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

***State***

```
XMLTree t = new  
XMLTree1("foo.xml");
```

```
t = [tree from  
file "foo.xml"]
```

# label

String label ( )

- Returns the label of the root of **this**.
- Ensures:

*label = [the label of the root of **this** (not including < > for tags)]*



# Example: Label is a Tag

<b>Code</b>	<b>State</b>
	<i>t = [tree for book XML example]</i>
<code>String s = t.label();</code>	

# Label is a Tag

<book>



printISBN → 978-1-118-06331-6

webISBN → 1-118063-31-7

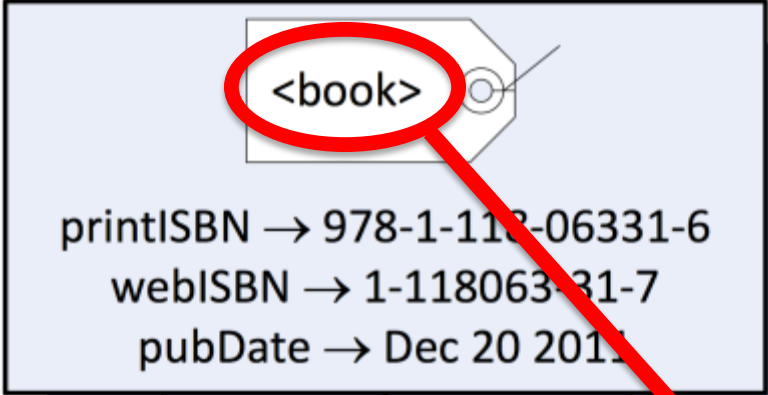
pubDate → Dec 20 2011

**State**

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

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

# Label is a Tag



`<book>`

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

**State**

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

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

```
t = [unchanged]  
s = "book"
```

# Example: Label is Not a Tag

<b>Code</b>	<b>State</b>
	<i>t = [tree rooted at title content in XML example]</i>
<code>String s = t.label();</code>	

# Label is Not a Tag


Java for Everyone:  
Late Objects

**State**

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

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

# label is Not a Tag



Java for Everyone:  
Late Objects

**State**

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

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

```
t = [unchanged]  
s = "Java for  
Everyone: Late  
Objects"
```

# isTag

**boolean** isTag( )

- Returns whether the label of the root of **this** is a tag.
- Ensures:

*isTag = [the label of the root of **this** is a tag]*

# Example: Label is a Tag

<b>Code</b>	<b>State</b>
	<i>t = [tree for book XML example]</i>
<b>boolean</b> b = t.isTag();	



# Label is a Tag

<book>



printISBN → 978-1-118-06331-6

webISBN → 1-118063-31-7

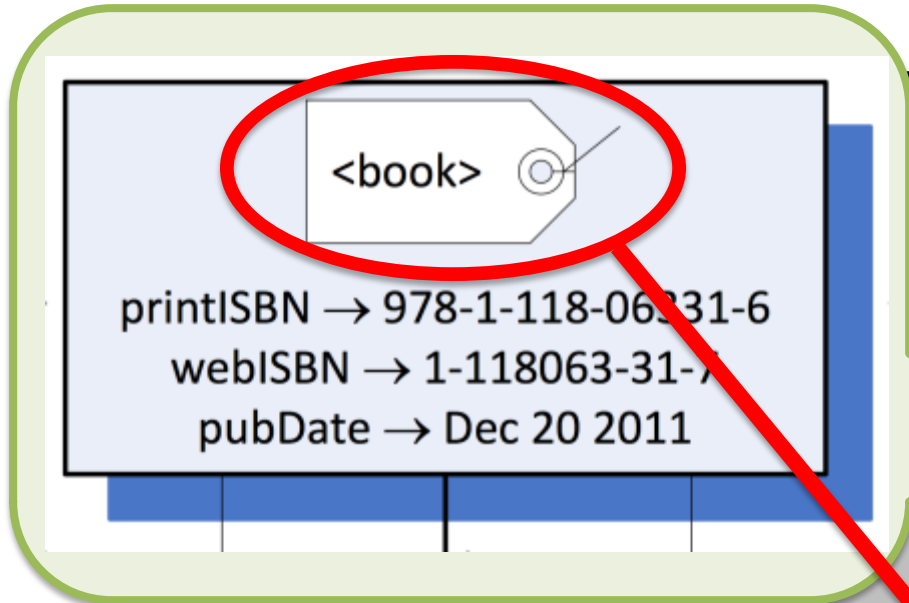
pubDate → Dec 20 2011

**State**

*t = [tree for  
book XML  
example]*

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

# Label is a Tag



## *State*

*t = [tree for  
book XML  
example]*

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

*t = [unchanged]  
b = **true***

# Example: Label is Not a Tag

<b>Code</b>	<b>State</b>
	<i>t = [tree rooted at title content in XML example]</i>
<b>boolean</b> b = t.isTag();	

# isTag is Not a Tag

Java for Everyone:  
Late Objects

***State***

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

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

# isTag() is Not a Tag

Java for Everyone:  
Late Objects

**State**

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

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

```
t = [unchanged]  
b = false
```

# 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

<b>Code</b>	<b>State</b>
	<i>t = [tree for book XML example]</i>
<b>boolean</b> b = t.hasAttribute ("pubDate");	

# Has One

<book>



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.hasAttribute  
("pubDate");
```



# Has One

`<book>` 

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.hasAttribute  
("pubDate");
```

```
t = [unchanged]  
b = true
```

# Example: Has None

<b>Code</b>	<b>State</b>
	<i>t = [tree for book XML example]</i>
<b>boolean</b> b = t.hasAttribute ("fooBar");	

# Has None

<book>



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.hasAttribute  
("fooBar");
```

# Has None

<book>



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.hasAttribute  
        ("fooBar");
```

*t = [unchanged]  
b = **false***

# attributeValue

String attributeValue(String name)

- Returns the value associated with the attribute of the root tag of **this** called `name`.

- Requires:

*[label of root of **this** is a tag and it has an attribute called name]*

- Ensures:

*attributeValue = [value associated with attribute called name of root tag of **this**]*

# Example

<b>Code</b>	<b>State</b>
	<i>t = [tree for book XML example]</i>
<pre>String v =     t.attributeValue     ("pubDate");</pre>	

# Example

<book>



printISBN → 978-1-118-06331-6

webISBN → 1-118063-31-7

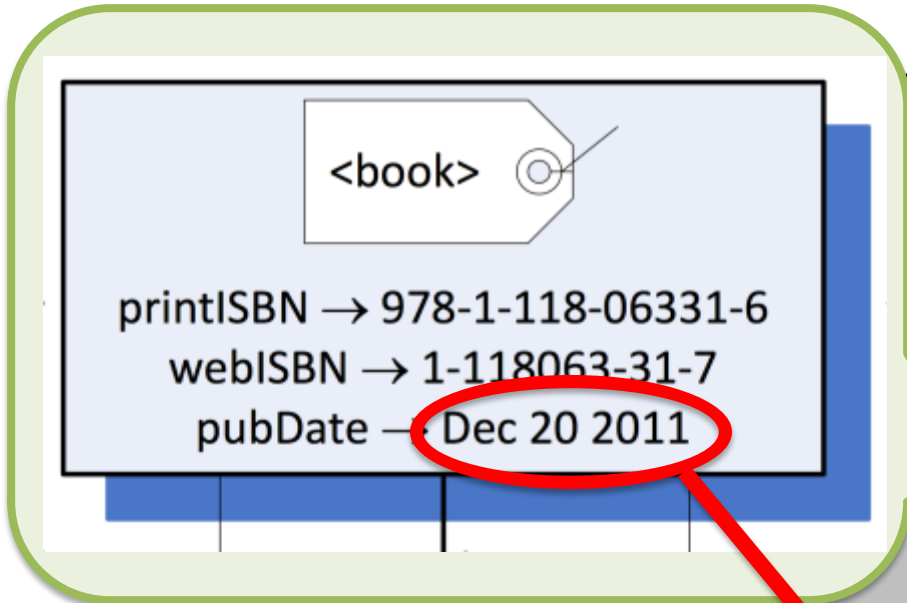
pubDate → Dec 20 2011

**State**

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

```
String v =  
    t.attributeValue  
    ("pubDate");
```

# Example



## State

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

```
String v =  
t.attributeValue  
("pubDate");
```

```
t = [unchanged]  
v = "Dec 20 2011"
```



# numberOfChildren

```
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

<b>Code</b>	<b>State</b>
	<i>t = [tree for book XML example]</i>
<b>int</b> n = t.numberOfChildren();	

# Example

<book>



printISBN → 978-1-118-06331-6

webISBN → 1-118063-31-7

pubDate → Dec 20 2011

**State**

*t = [tree for  
book XML  
example]*

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

# Example

`<book>`

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

## State

*t = [tree for  
book XML  
example]*

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

*t = [unchanged]  
n = 3*

# child

XMLTree child(**int** k)

- Returns the  $k$ -th subtree of the root of **this**.

- Requires:

*[label of root of **this** is a tag and  
 $0 \leq k < \text{number of subtrees of the root of **this}}**$ ]*

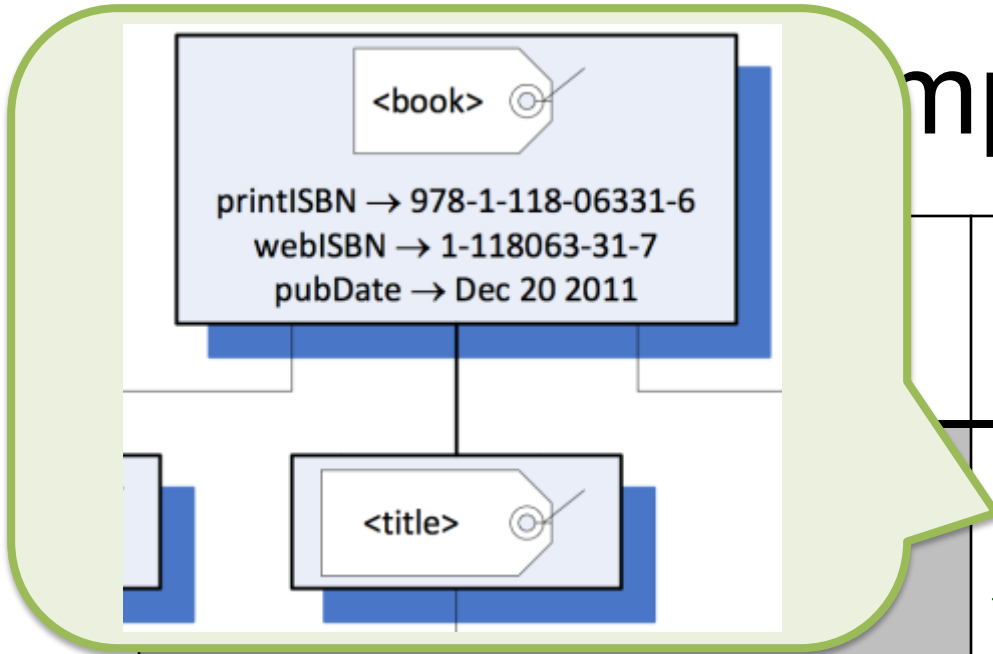
- Ensures:

*child = [the  $k$ -th subtree of the root of **this**]*

# Example

<b>Code</b>	<b>State</b>
	<i>t = [tree for book XML example]</i>
<code>XMLTree st = t.child(1);</code>	

# Example

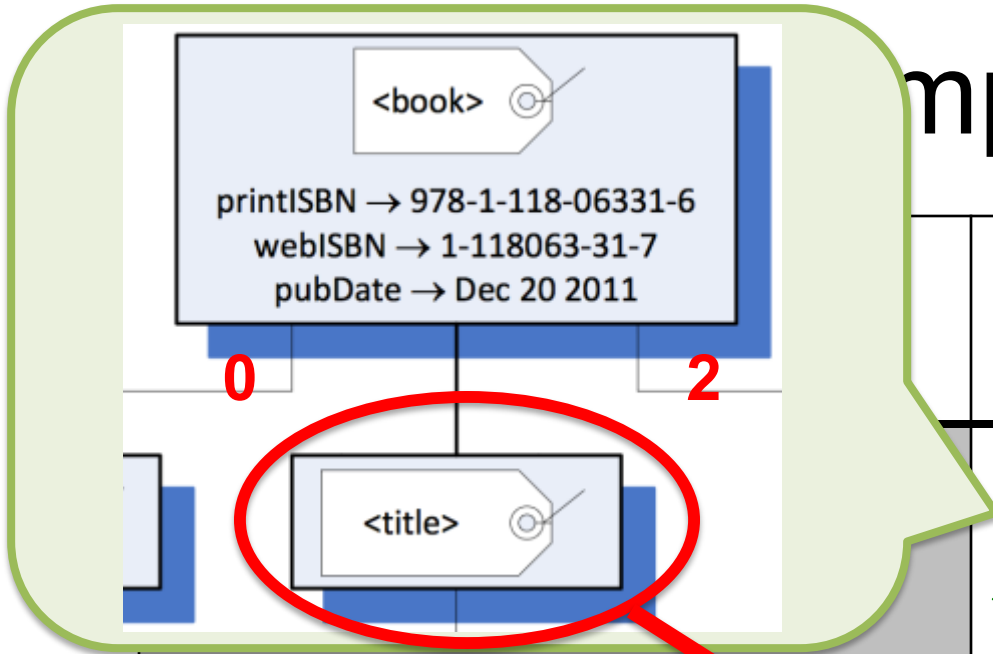


**State**

*t = [tree for  
book XML example]*

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

# Example



## State

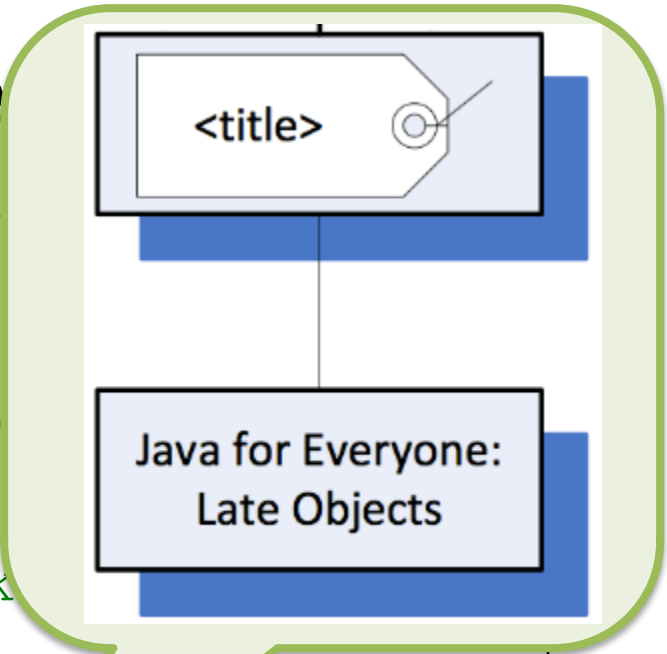
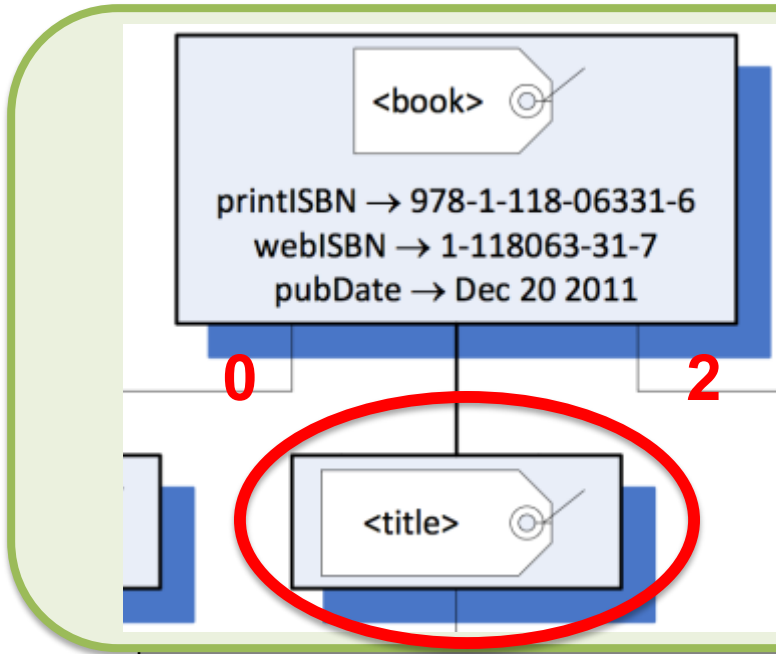
```
t = [tree for  
book XML example]
```

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

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



# Example



```
t =  
book
```

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

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

# 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

- Continuation of previous example: what type and what value?  
`t.child(1).child(0)`

The type is `String`, the return type of the `label` method; the value is `"Java for Everyone: Late Objects"`.

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

```
t.child(1).child(0).label()
```

# 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 that you can iterate on is called **iterable**
- The **collection classes** in Python and the OSU CSE **collections** module provide 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

```
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

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

This is called a ***for-each loop***

# attributeNames

`Iterable<String> attributeNames()`

- Returns an `Iterable<String>` of the attribute names of the root of **this**.

- Requires:

*[label of root of **this** is a tag]*

- Ensures:

*attributeNames = [an `Iterable<String>` of the attribute names of the root of **this**]*



# Example

<b>Code</b>	<b>State</b>
	<i>t = [tree for book XML example]</i>
<i>Iterable&lt;String&gt; it = t.attributeNames();</i>	

# Example

<book>



printISBN → 978-1-118-06331-6

webISBN → 1-118063-31-7

pubDate → Dec 20 2011

**State**

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

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

# Example

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

## **State**

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

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

```
t = [unchanged]  
it = [iterable  
for 3 Strings]
```

# 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`:

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

# display

```
void display()
```

- Displays **this** in a new window.
- Ensures:

*[**this** is displayed in a new window]*

# toString

String toString()

- Returns an XML string representation of **this**.
- Ensures:

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

# toString

String toString()

- Returns an XML string representation of **this**.
- Ensures:

*toString = [a  
representatio*

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`
  - <http://web.cse.ohio-state.edu/software/common/doc/>