Labs: Fast Track to XML, XSLT, and Java (Eclipse)

Version: 20211118

Presentation Slides:
For INSTRUCTOR Use Only

Release Level

- This manual contains instructions for creating and running the labs using the following platforms:
  - **Java 11**
  - **Eclipse Enterprise Java Developers Edition**
    - Tested with versions: 2021-06 (Java 11)
- All labs have been tested on a recent Windows operating system

STOP

Lab

Lab 1.1: Setting up the Environment

- **Purpose**: To familiarize you with the lab environment
  - Become familiar with the lab structure and Eclipse
  - Start up Eclipse, and creating/using a Eclipse project
- You will also get a brief introduction to Eclipse's capabilities
  - To learn enough to be able to work comfortably with Eclipse
  - It is not an in-depth coverage
  - We'll start Eclipse, make a simple project, and work with XML files
- **Builds on previous labs**: None
- **Approximate Time**: 30-40 minutes

Extract the Lab Setup Zip File

Lab

- To set up the labs, you'll need the course setup zip file *
  - It has a name like: ***LabSetup_XML-XSLT-Java_Eclipse_20211118.zip***
- Our base working directory for this part of the course will be **C:\StudentWork\XMLIntro**
  - This directory will be created when we extract the Setup zip
  - It includes a directory structure and files (e.g., Java files, XML files, other files) that will be needed in the labs
  - All instructions assume that this zip file is extracted to C:\. **If you choose a different directory, please adjust accordingly**

Tasks to Perform

- **Unzip** the lab setup file to **C:\**
  - This will create the directory structure, described in the next slide, containing files that you will need for doing the labs

General Instructions

Lab

- **Lab Directory Structure**: Your labs will be in the directory: **StudentWork\XMLIntro\workspace**
- The root lab directory where you will do your work for this lab is: **C:\StudentWork\XMLIntro\workspace\Lab01.1**
  - This directory already exists in your workspace – you'll do your work in this directory
  - Generally, the files you work on for a lab will be under the root directory (and instructions are given relative to this directory)
- Detailed instructions are included in this lab
  - They include complete instructions for working in the **Eclipse** environment, as well as details about the lab requirements
- Subsequent labs require you to do the same thing as this lab to build/run, so they include fewer detailed instructions

The Eclipse Development Environment

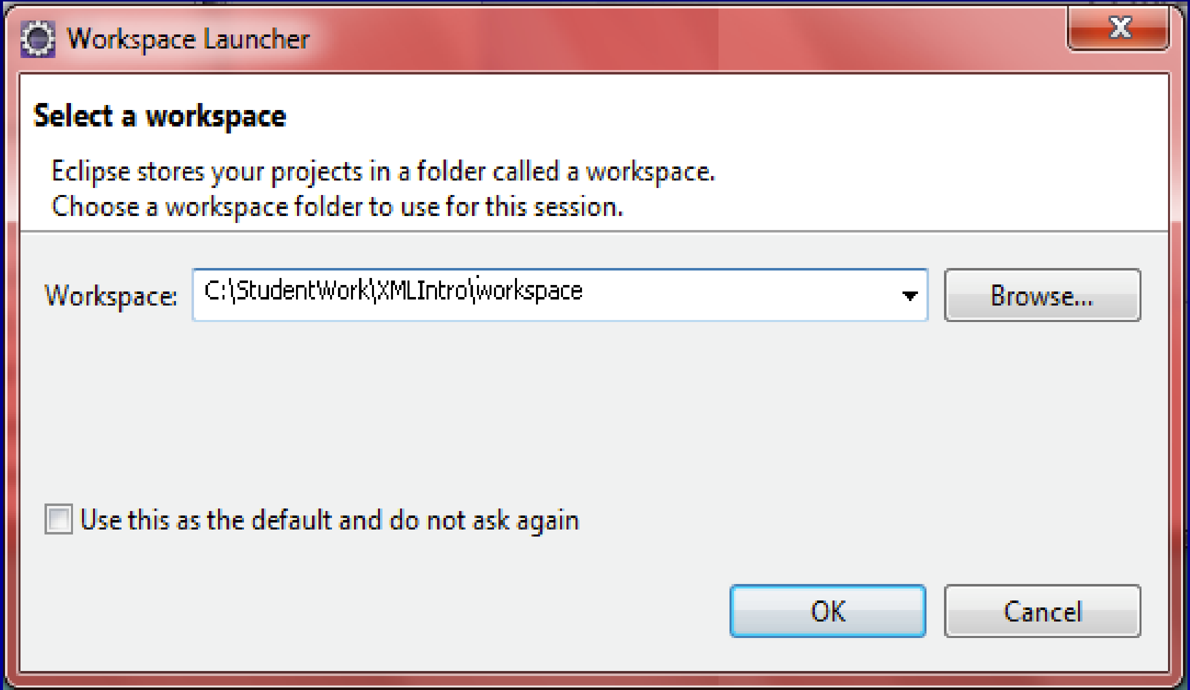
Lab

- **Eclipse**: Open source platform for building integrated development environments (IDEs)
  - Used mainly for Java development
  - Can be flexibly extended via plugins to add capabilities
  - **http://www.eclipse.org** is the main website
- This lab includes **detailed instructions** on using Eclipse
  - Starting it, creating and configuring projects, etc.
- Other labs include **fewer Eclipse details** - they may just say build/run as previously
  - Just use the same procedures to build/run as in this lab
  - Refer back to these lab instructions as needed

Launch Eclipse

Tasks to Perform

- Launch eclipse, go to **c:\eclipse** and run **eclipse.exe**
  - A dialog box appears prompting for workbench location (below left)
  - Set the workbench location to **C:\StudentWork\XMLIntro\workspace**
  - If a different default Workbench location is set, change it
  - Click **OK**
  - Close the **Welcome** screen: Click the X on the tab (below right)



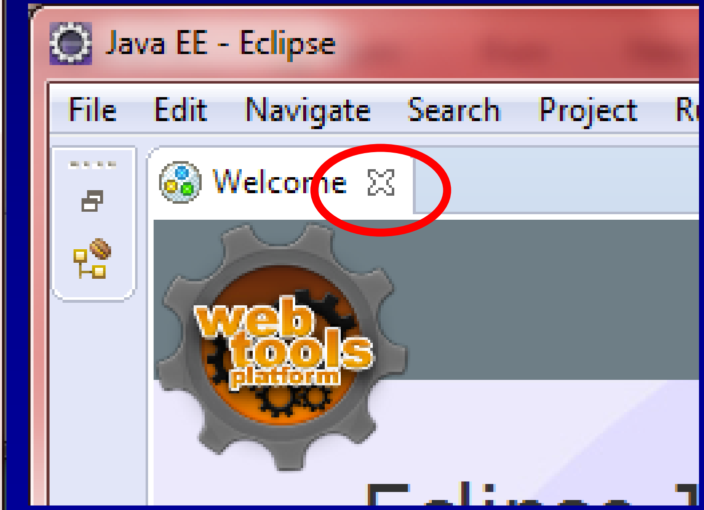
Workspace Launcher

Select a workspace

Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\StudentWork\XMLIntro\workspace

□ Use this as the default and do not ask again



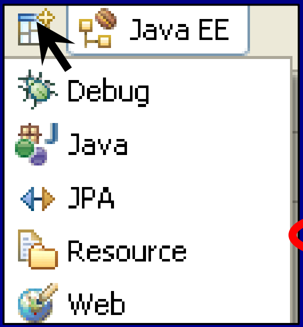
Java EE - Eclipse

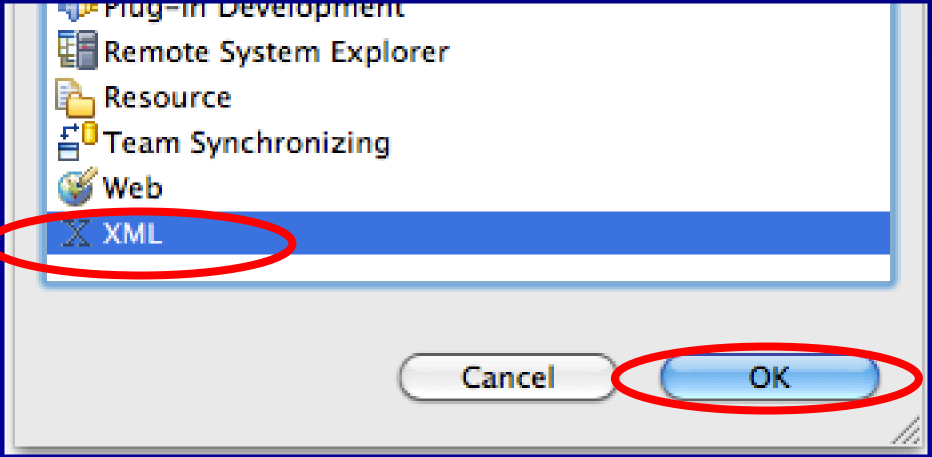
Workbench and XML Perspective

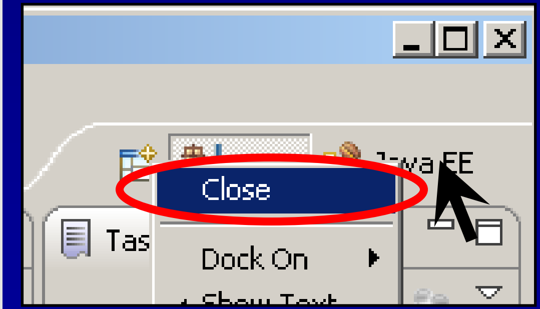
Lab

Tasks to Perform

- You'll likely be in a Java EE perspective *
- If in a **Java EE** perspective, **open an XML one** by clicking the Perspective icon at the top right of the Workbench (below left), and select XML (below center)
  - Close the Java EE perspective by right clicking its icon, and selecting close (below right)
  - If you were in an XML perspective, then just remain in it





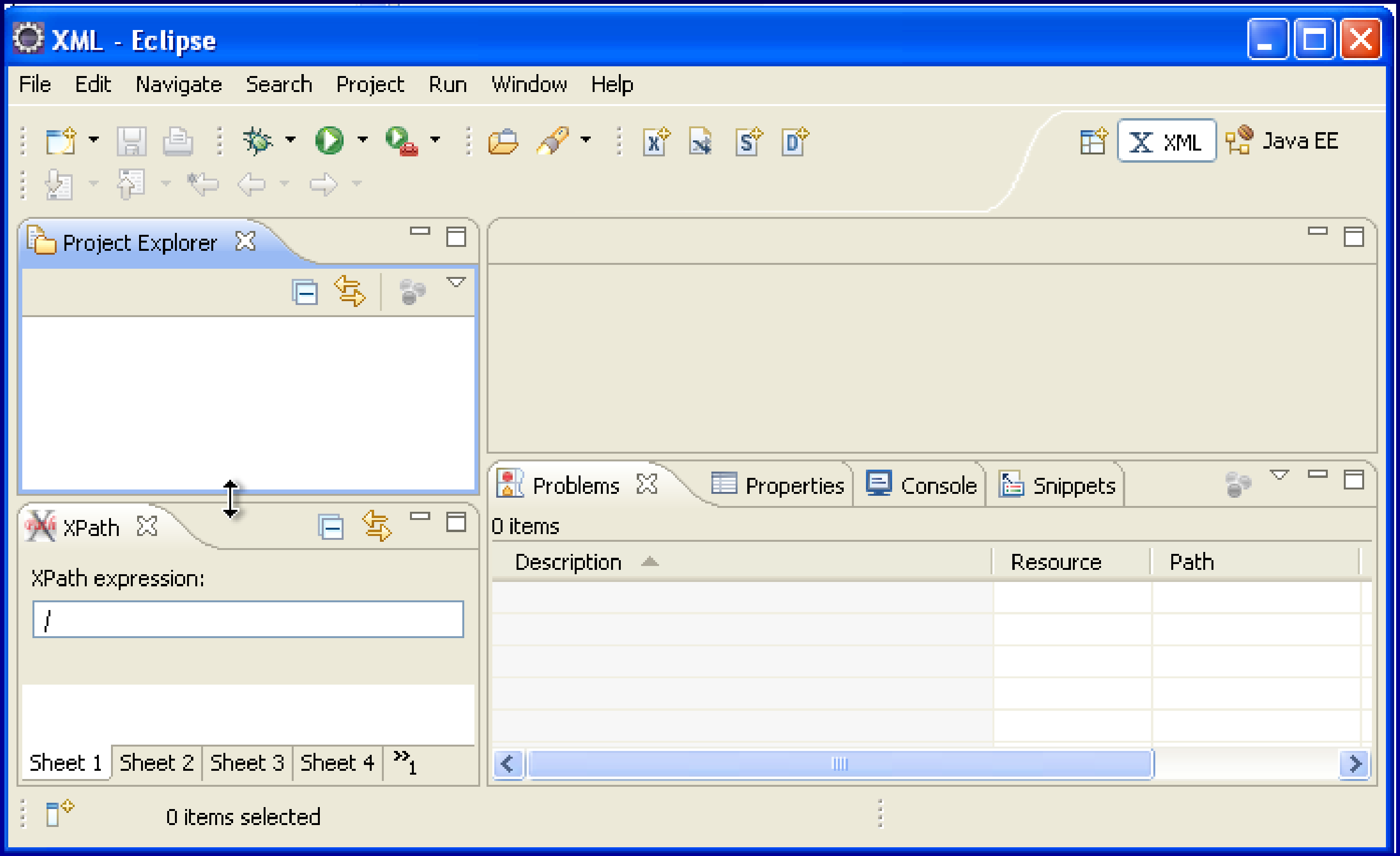


Unclutter the Workbench

Lab

Tasks to Perform

- Let's unclutter the Perspective by closing some views
  - Close the Outline view (click on the X)
  - You can save this as the default if you want (see note)



XPath expression:

/

0 items

0 items selected

Create a Project for our Lab

Lab

Tasks to Perform

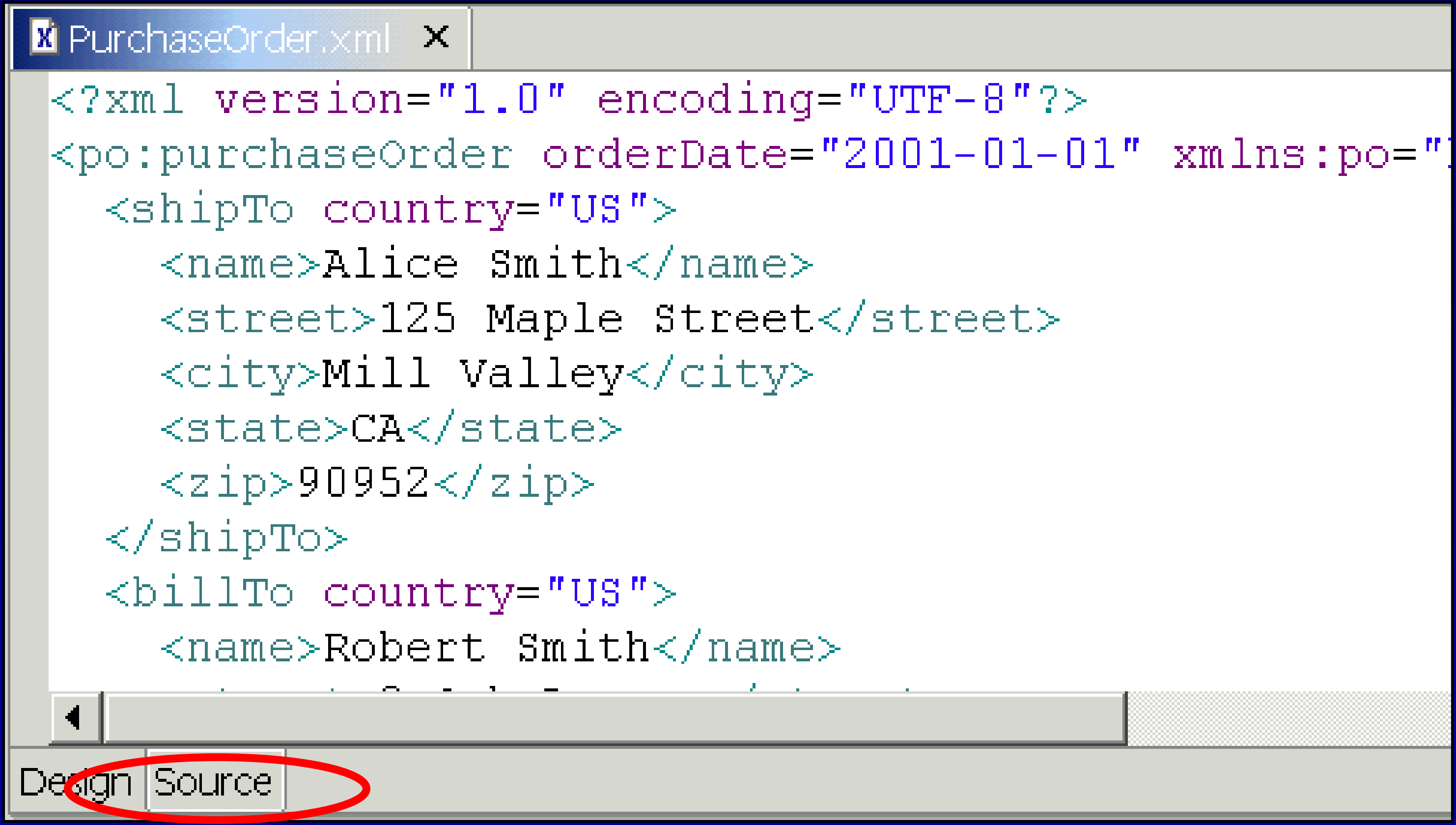
- Create a **Project**
  - To create a new Project, use the menu item: **File | New | Project | General| Project** (see notes)
  - Call the project **Lab01.1** - Eclipse will then automatically set the project directory to *Lab01.1* (it contains lab files)
  - Click **Finish**

- Create a new XML file within the project you just created
  - There are multiple ways to do this – we mention one way here
  - Right click on the Lab01.1 project icon in **Project Explorer** and select **New | XML File** to create a new XML file
  - Call the file *order.xml*, click **Next**, and in the next dialog, choose **Create XML file from a template** *
  - Click **Finish** – this will create and open the XML file

Editors

Lab

- There is a source editor like this one for a .xml file.
  - This is seen in the **Source** tab of the editor
  - We're showing sample XML in it (it's not in the lab)

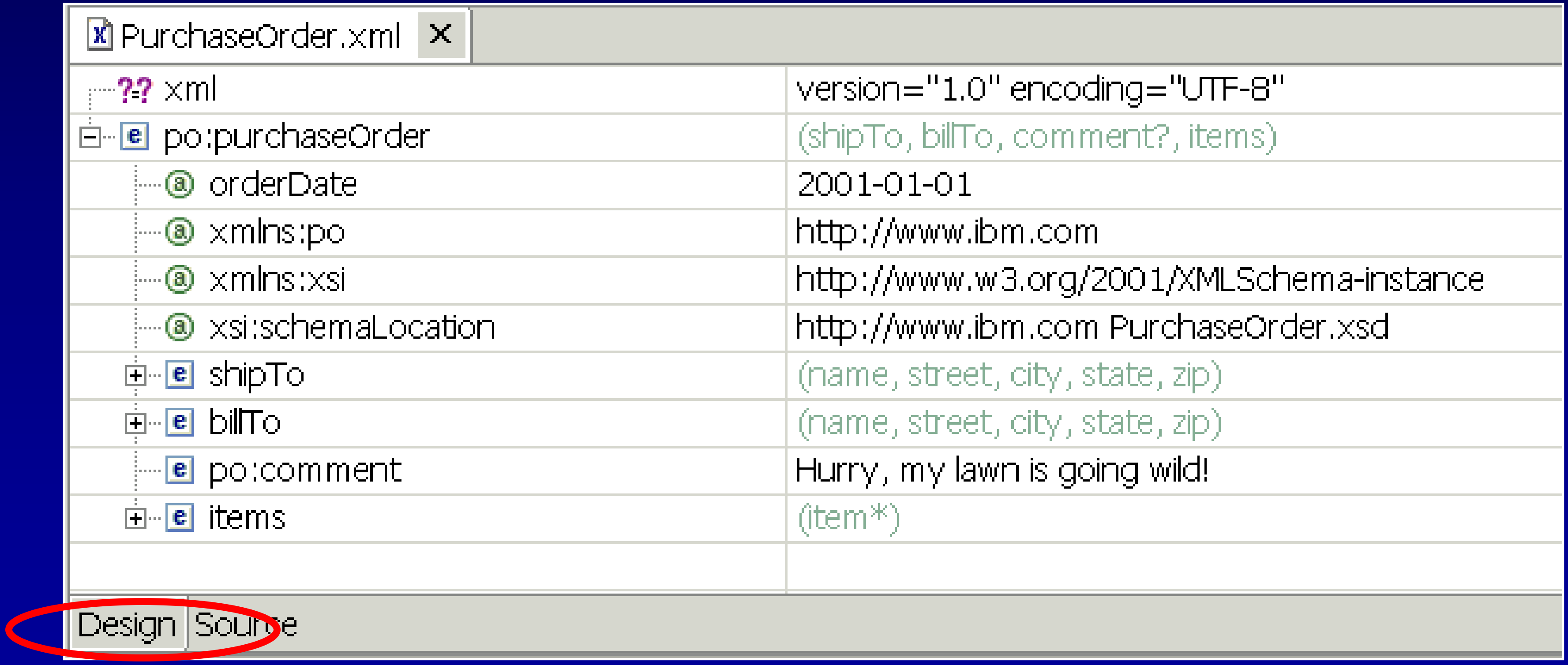


```
<?xml version="1.0" encoding="UTF-8"?>
<po:purchaseOrder orderDate="2001-01-01" xmlns:po="
  <shipTo country="US">
    <name>Alice Smith</name>
    <street>125 Maple Street</street>
    <city>Mill Valley</city>
    <state>CA</state>
    <zip>90952</zip>
  </shipTo>
  <billTo country="US">
    <name>Robert Smith</name>
```

Editors

Lab

- There are design editors like this one for an xml file and for many other types of files. (JSP, HTML etc.)
  – This is seen in the **Design** tab of the editor



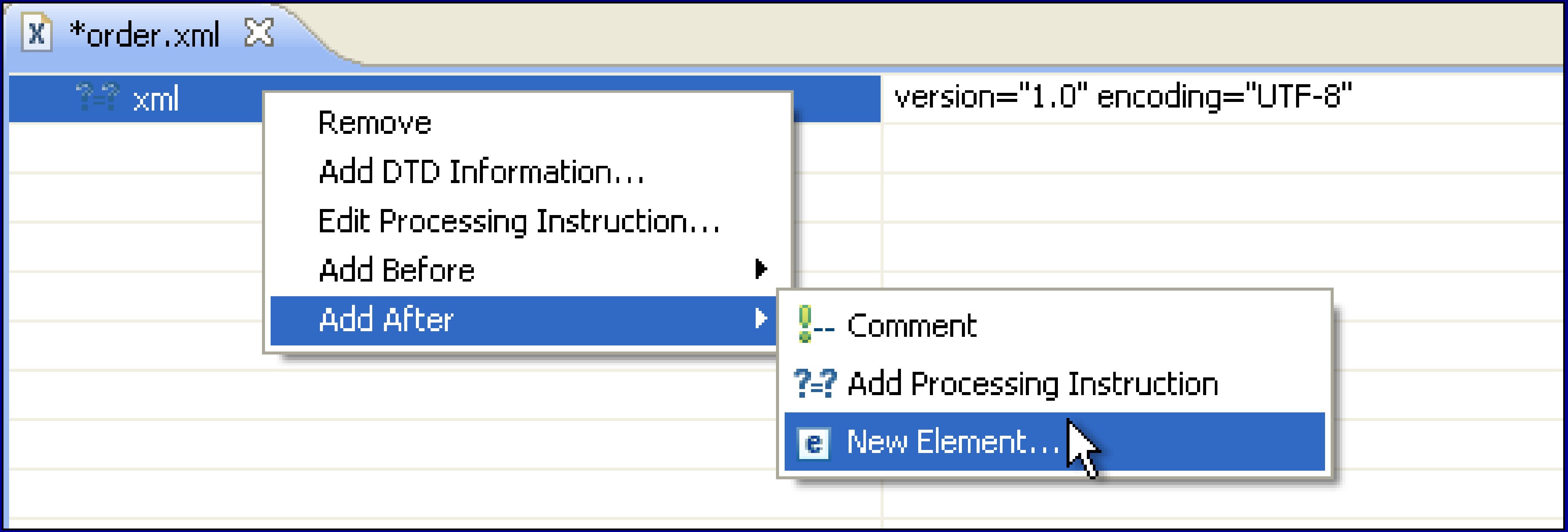
?? xml	version="1.0" encoding="UTF-8"
po:purchaseOrder	(shipTo, billTo, comment?, items)
orderDate	2001-01-01
xmlns:po	http://www.ibm.com
xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation	http://www.ibm.com PurchaseOrder.xsd
shipTo	(name, street, city, state, zip)
billTo	(name, street, city, state, zip)
po:comment	Hurry, my lawn is going wild!
items	(item*)

Add in an Element

Lab

Tasks to Perform

- Add in an "order" element using the design view
  - Right click on the XML declaration, choose **Add After -> New Element**
  - Call the element **order**
  - Look at the document in the design and source views
  - You can also type the element directly in the source view


*order.xml
xml
version="1.0" encoding="UTF-8"
Remove
Add DTD Information...
Edit Processing Instruction...
Add Before
Add After
Comment
Add Processing Instruction
New Element...

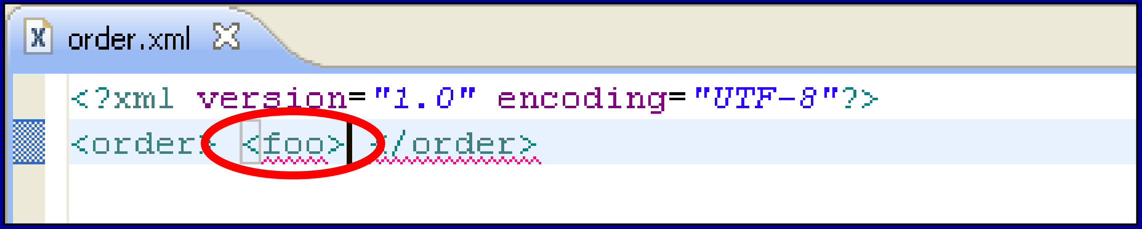
Errors and The Task View

Lab

- Let's add an error in the XML file
  - We'll then validate the file, and see that Eclipse can find XML errors

Tasks to Perform

- View the file in **Source** view, and add a **<foo>** tag with no matching end tag, as shown below
  - **You'll have to remove the end tag manually**, because Eclipse automatically adds it in when you create a new element
  - This is not valid XML (we'll talk about what that means later)
  - **Save** the file – it will show errors in the Problems view
    - When you save an XML file, Eclipse will validate it



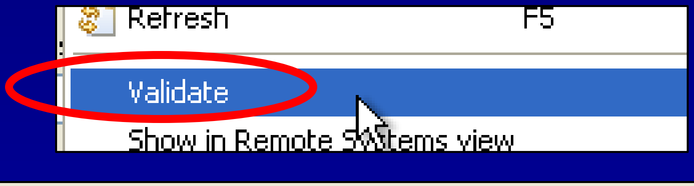
```
<?xml version="1.0" encoding="UTF-8"?>
<order> <foo> </order>
```

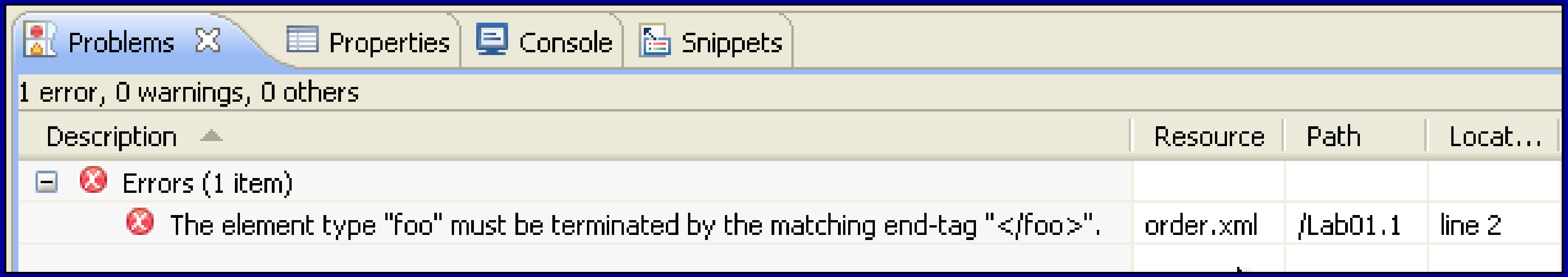
Manually Validating a file

Lab

Tasks to Perform

- Validation will check if the file is good (well formed) XML
- **Validate** the file by right clicking on it in Project Explorer, and selecting Validate (see image below)
  - This will check the document for well formedness
  - The error should show up in the Problems view (see bottom image)
- The rest of this lab describes the structure of Eclipse, for those that haven't used it





1 error, 0 warnings, 0 others

Description	Resource	Path	Locat...
⊟ ⊗ Errors (1 item)			
⊗ The element type "foo" must be terminated by the matching end-tag "</foo>".	order.xml	/Lab01.1	line 2

Important Notes for Using Eclipse

Lab

- Any lab that has a **new lab directory** will require you to create **a new Eclipse project**
  - Sometimes several labs are done one directory, in which case you will use the same project for all of them

- If you copy/paste files, paste them **within Eclipse**
  - Generally you'll be copying files from the one of the *LabSetup* subfolders
  - Then pasting them into the project you are working on
  - We'll give detailed instructions the first time we do this

- For anyone not familiar with Eclipse, the next few slides give a (very) brief overview of how Eclipse is structured
  - There is **nothing you need to do** in those slides – they are for information purposes only

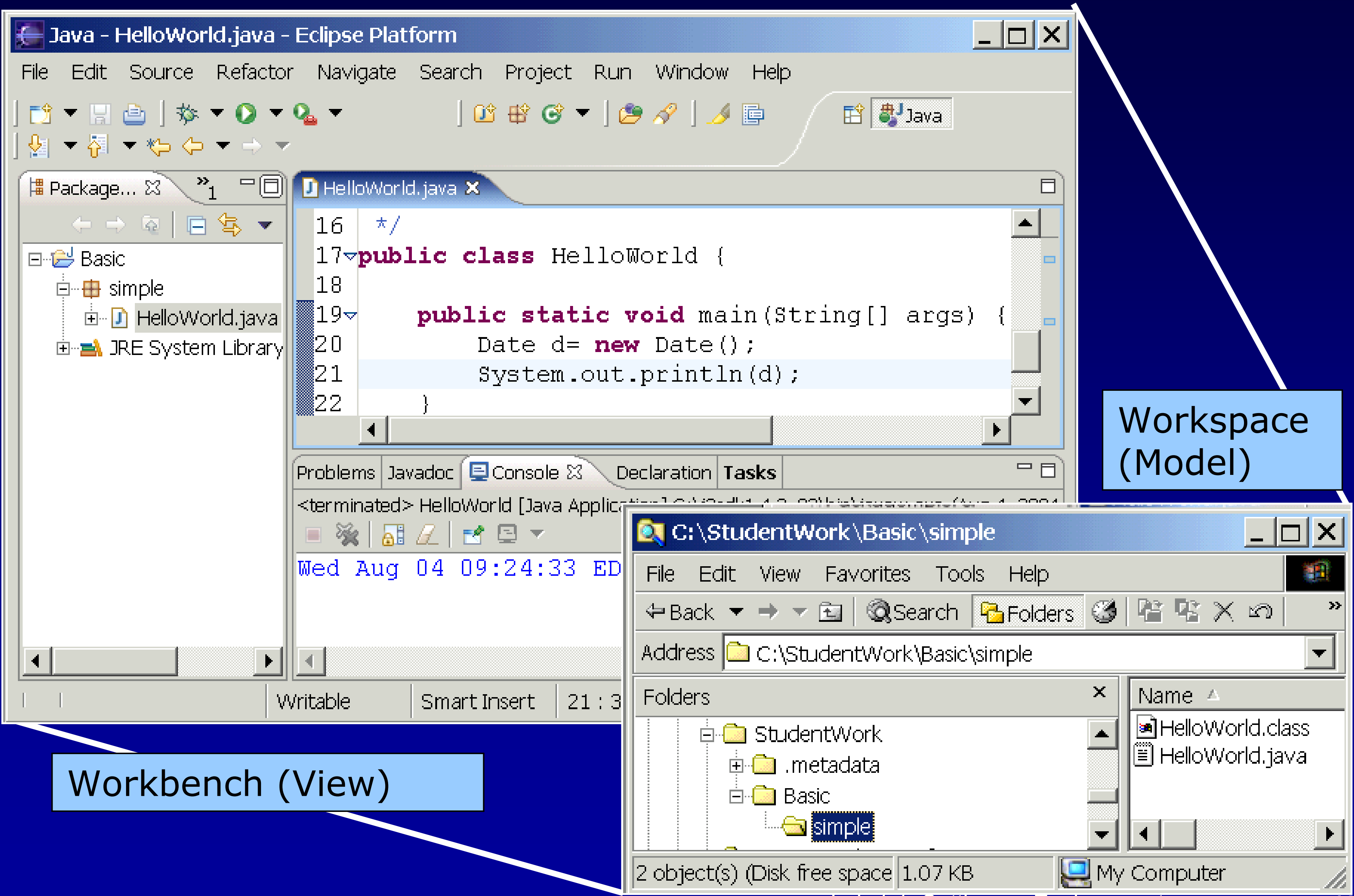
The Eclipse Paradigm

Lab

- Eclipse products have two fundamental layers
  - The **Workspace** – files, packages, projects, resource connections, configuration properties
  - The **Workbench** – editors, views, and perspectives
- The Workbench sits on top of the Workspace
  - Provides views to access/manipulate resources
  - **Editor** – A component that allows a developer to interact with and modify the contents of a file.
  - **View** – A component that exposes meta-data about the currently selected resource.
  - **Perspective** – A grouping of related editors and views that are relevant to a particular task and/or role.
- You can have multiple perspectives open to provide access to different aspects of the underlying resources

Workbench and Workspace

Lab



Java - HelloWorld.java - Eclipse Platform

```
 */
public class HelloWorld {

    public static void main(String[] args) {
        Date d= new Date();
        System.out.println(d);
    }
```

<terminated> HelloWorld [Java Applic

```
Wed Aug 04 09:24:33 ED
```

C:\StudentWork\Basic\simple

Address C:\StudentWork\Basic\simple

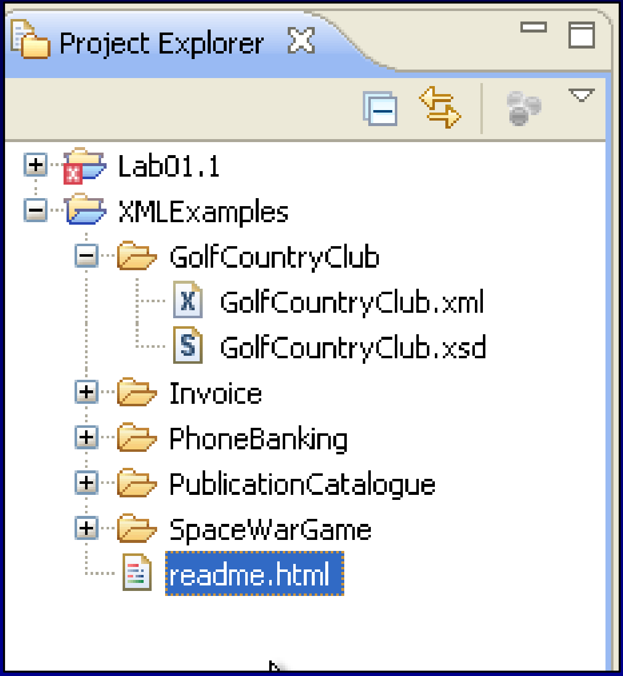
Workspace (Model)

Workbench (View)

Project Explorer View

Lab

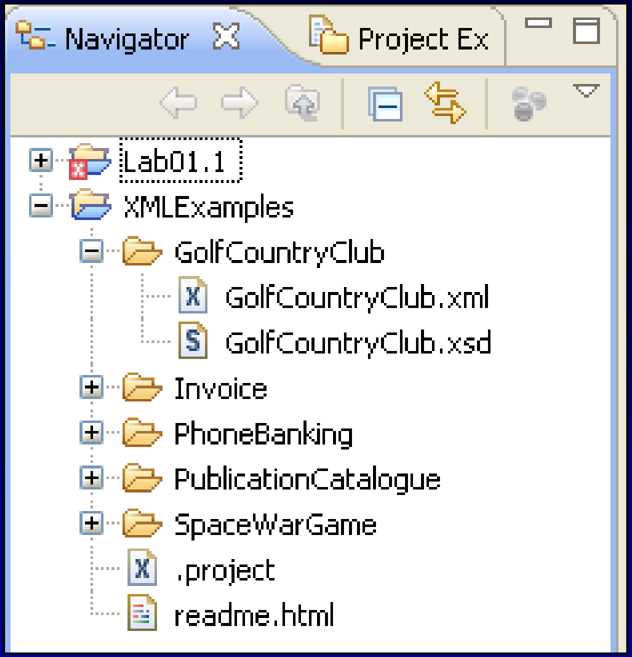
- Shows the resources in a Project
- What is shown may change depending on the type of project
  - In a simple project, there are generally projects, files and folders shown
  - May not show all the files that are in the file system (for example, the .project file which is used by Eclipse to organize the project)



Navigator View

Lab

- Similar to file system view
  - There are three kinds of resources described below



- **Files**
  - Correspond to files on the file system
- **Folders**
  - Like folders on the file system
- **Projects**
  - Used to organize all your resources and for version control.
  - Creating a new project assigns a physical location for it on the file system.
  - A third-party SCM (Source Control Manager) may be used to properly share project files amongst developers.

STOP

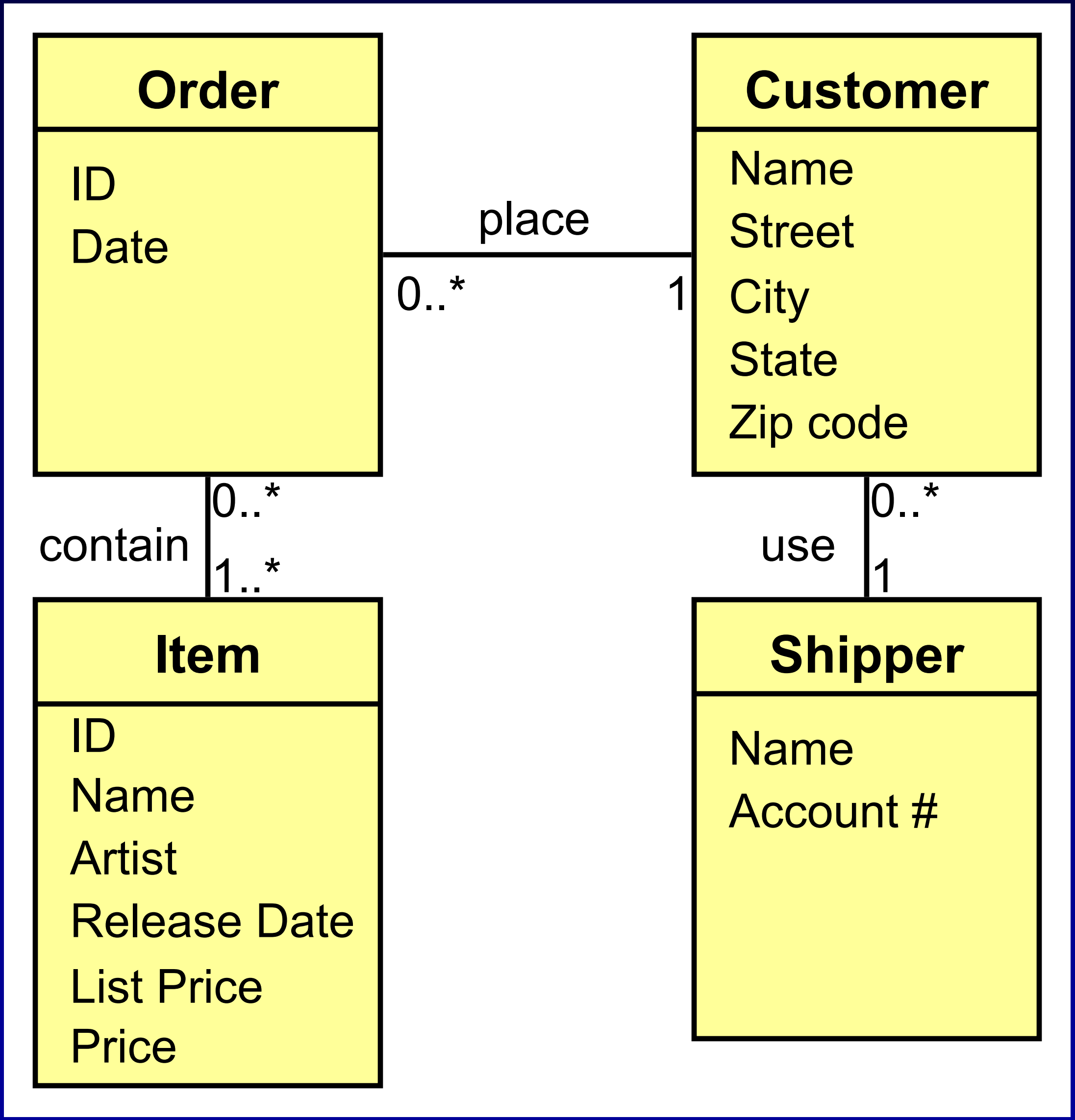
Lab

Lab 2.1: Representing Data as XML

- **Purpose**: In this lab, we will create a JavaTunes purchase order XML document from an order form
  - Continue working in your **Lab01.1** project
- **Objectives**: Learn more about creating XML documents
  - You will create a valid XML document based on the data model of the JavaTunes purchase order seen in the slides
- **Builds on previous labs**: Lab01.1
- **Approximate Time**: 20-30 minutes

Purchase Order - Data Model

Lab



This is a UML class diagram showing the data model for the JavaTunes purchase order

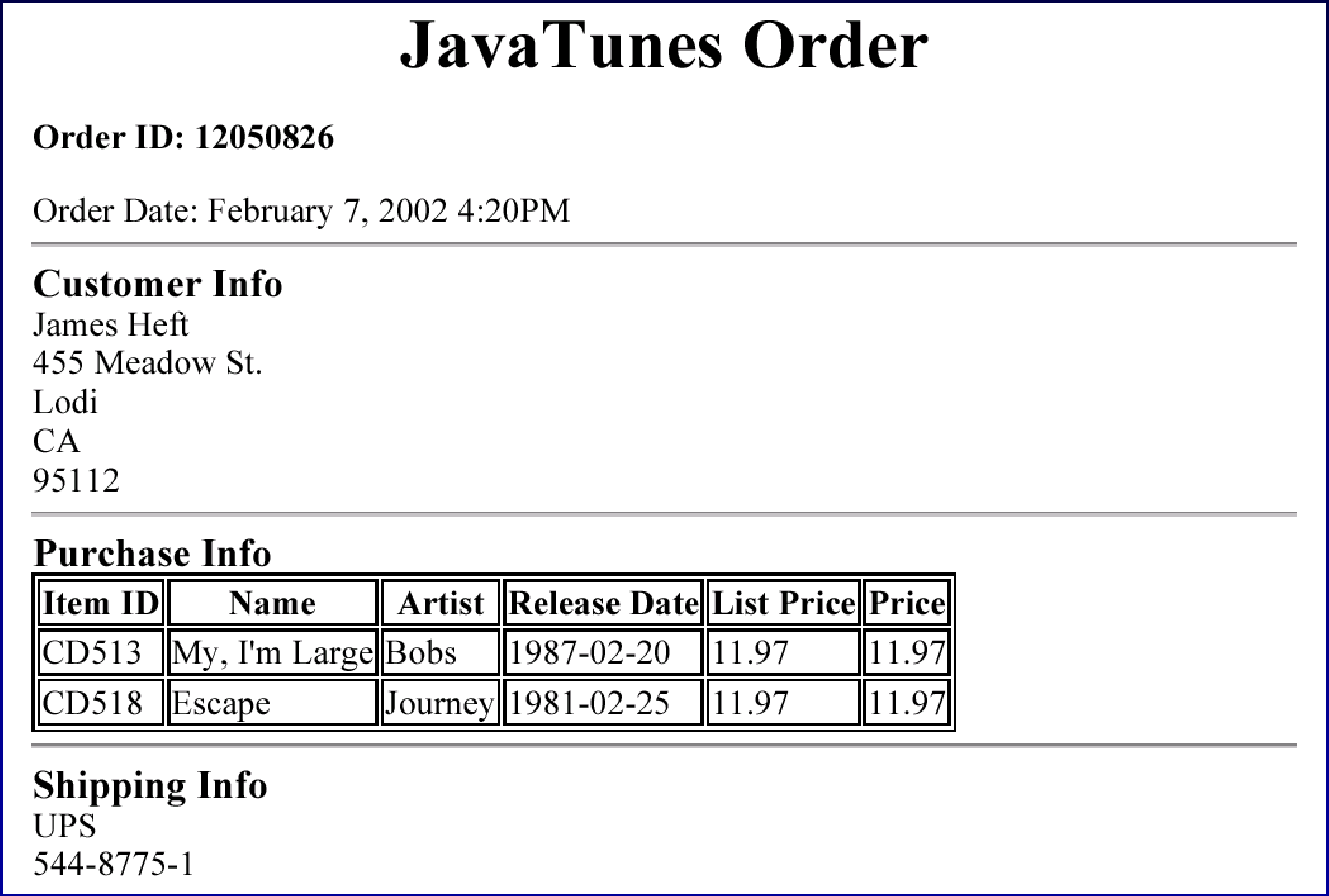
Purchase Order - Sample Document

Lab

```
<!-- JavaTunes order XML document -->

<order ID='67183625' dateTime='2001-10-03 09:50'>
  <customer>
    <name>Leanne Ross</name>
    <street>1475 Cedar Avenue</street>
    <city>Fargo</city>
    <state>ND</state>
    <zipcode>58103</zipcode>
    <shipper name='FedEx' accountNum='893-192'/>
  </customer>
  <item ID='CD509'>
    <name>Surfacing</name>
    <artist>Sarah McLachlin</artist>
    <releaseDate>1997-12-04</releaseDate>
    <listPrice>17.97</listPrice>
    <price>13.99</price>
  </item>
</order>
```

Purchase Order - Data

Lab



JavaTunes Order

Order ID: 12050826

Order Date: February 7, 2002 4:20PM

Customer Info

James Heft
455 Meadow St.
Lodi
CA
95112

Purchase Info

Item ID	Name	Artist	Release Date	List Price	Price
CD513	My, I'm Large	Bobs	1987-02-20	11.97	11.97
CD518	Escape	Journey	1981-02-25	11.97	11.97

Shipping Info

UPS
544-8775-1

Testing for Well-formedness

Lab

Tasks to Perform

- Modify the *order.xml* document from the previous lab so it contains purchase order information as well-formed XML
  - Use the structure of the example JavaTunes purchase order XML document earlier in this section (also shown in these lab instructions)
  - You can right click on the ***order.html*** file in the project, and select ***Open With | Web Browser*** to have a nice visual view of the data *
- **Validate** *order.xml* when you're done
  - Right click on *order.xml*, and select **Validate**
  - If you have any errors, they should show up in the Problems view
  - Correct any errors that you find
- Once you have a valid XML document with the data, you're done

STOP

Lab

Lab 2.2: Adding a PI

- **Purpose**: In this lab, we will add a processing instruction that transforms our XML into HTML
  - We'll also experiment with some of the other XML syntax
  - Continue working in your **Lab01.1** project
- **Objectives**: Work with Processing Instructions, and with other XML syntax
- **Builds on previous labs**: Lab02.1
- **Approximate Time**: 20-30 minutes

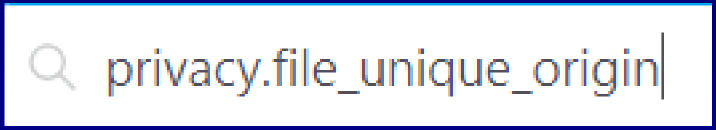
Enabling Local XSLT Files in Firefox

Lab

Tasks to Perform

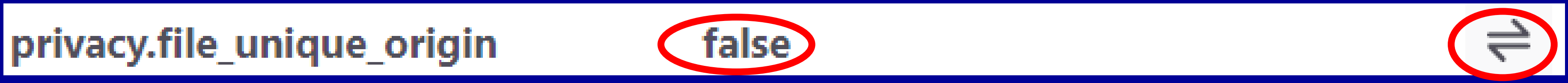
- Most browsers disallow XSLT transforms for local files loading a local stylesheet
  - The FireFox browser makes it easy to enable, so we'll do that now

- Open FireFox, type "about:config" into the URL field, Return
  - Search for the preference **`privacy.file_unique_origin`**



privacy.file_unique_origin

  - Toggle the value so it's false, as shown below
  - That's it, you're ready to do the transform



privacy.file_unique_origin false

Adding a PI to Our Document

Lab

Tasks to Perform

- Add an `xml-stylesheet` PI to *order.xml*
  - Have it specify ***customer.xsl*** as the stylesheet (this is an XSLT stylesheet already supplied in your lab directory)
  - The `xml-stylesheet` PI must be in the prolog - See the example in this section
  - The PI directs an XML-enabled browser to display the document according to what's in *customer.xsl*
- **Load** *order.xml* into the FireFox browser -- what do you see?
  - Open the supplied *customer.html* file in a browser – they should be similar (This file is in the lab directory)
  - After you see what it does, comment out the PI

Adding PIs and Comments

Lab

Tasks to Perform

- Create a PI that directs the application to page the shipper
  - Put it somewhere in the document body, e.g., right after the `shipper` element -- we’re not going to use it, so you can make it up -- example:

    `<?pager UPS 544-8775-1?>`

- If you wish, add some comments to the document
  - Comments can go in the prolog and the body

- OPTIONAL - change some of the purchase order data to use markup characters and then escape those characters
  - For example, `<artist>Seals & Crofts</artist>`

STOP

Lab

Lab 3.1: Namespaces

- **Purpose**: In this lab, we will add namespace definitions to an XML document
  - We'll declare the namespaces in two different ways, and compare them
- **Objectives**: Work and become more familiar with namespaces
- **Builds on previous labs**: None
  - The root lab directory where you will do your work for this lab is:

  C:\StudentWork\XMLIntro\workspace\Lab03.1
- **Approximate Time**: 30-40 minutes

Namespaces

Lab

Tasks to Perform

- Create a **Project** (**File | New | Project | General| Project**)
  - Call the project **Lab03.1**
- You will work on the file ***orderns.xml***, which is already in the lab directory, and is shown on the next two slides
- Verify your solution **after each step** by validating it and making sure that there are no errors
  - You can also load your *orderns.xml* file into a browser, to see how it formats namespace definitions

orderns.xml

Lab

```
<?xml version='1.0'?>

<!-- JavaTunes order XML document -->

<order ID='03230413' dateTime='2002-03-24 01:20'>
  <customer>
    <name title='Ms.'>
      <firstName>Rachel</firstName>
      <lastName>Jacobs</lastName>
    </name>
    <street>1408 Fell St.</street>
    <city>Oneida</city>
    <state>NY</state>
    <zipcode>14180</zipcode>
    <shipper name='FedEx' accountNum='77-63-2478'/>
  </customer>
  ...
```

orderns.xml

Lab

```
  ...
  <item ID='CD514'>
    <name>So</name>
    <artist>Peter Gabriel</artist>
    <releaseDate>1986-10-03</releaseDate>
    <listPrice>17.97</listPrice>
    <price>13.99</price>
  </item>
  <item ID='CD506'>
    <name>Seal</name>
    <artist>Seal</artist>
    <releaseDate>1991-08-18</releaseDate>
    <listPrice>17.97</listPrice>
    <price>14.99</price>
  </item>
</order>
```

Part A - Namespaces and Prefix Bindings

Lab

Tasks to Perform

- Define a namespace prefix for **items** so that its scope is confined to **only** each `item` element (see notes)
  - Place each `item` element and each one's child elements in the namespace -- do not put `item`'s `ID` attribute in the namespace
- Define a namespace prefix for **customers** so that its scope is **only** the `customer` element (see notes)
  - Place the `customer` element and all of its descendant elements in the namespace -- do not put `name`'s `title` attribute in the namespace
  - Do not put the `shipper` element's attributes in the namespace
- After validating *orderns.xml*, look at it in design view -
  - Open the nodes for the order, customer, and item elements, and notice where the namespace declarations lie
- Copy *orderns.xml* to ***ordernsA.xml*** (within Eclipse *) for use later

Part B - Namespace Scope

Lab

Tasks to Perform

- Define the namespace prefix for items so that it includes both `item` elements, but only define the namespace in one place
  - What are your choices for where this namespace definition can go?
- Where else can you define the namespace prefix for customers?
  - Define it there
- After making these changes, validate your document again
  - Open *ordernsA.xml* also, and compare it to your current *orderns.xml* looking at both of them in design view
  - It's easy to see in this view where the namespaces are declared, and how the contained elements are effected by the declarations

STOP

Lab

Lab 3.2: Default Namespaces

- **Purpose**: In this lab, we will use default namespaces in an XML document
  - We'll work with our order document, as well as with a new document, ***purchase-requestns.xml***, which is also in the lab dir
- **Objectives**: Work with default namespaces
- **Builds on previous labs**: Lab 3.1
  - Continue working in your Lab03.1 project
- **Approximate Time**: 30-40 minutes

purchase-requestns.xml

Lab

```
<?xml version='1.0'?>

<!-- JavaTunes purchase request XML document -->

<purchase-request>
  <purchase>
    <amount currency='USD'>1016.84</amount>
    <dateTime>2002-01-04 14:21</dateTime>
  </purchase>
  <merchant>
    <merchant-name>JavaTunes</merchant-name>
    <business-number>190973</business-number>
  </merchant>
  <credit-card type='Visa'>
    <name-on-card>Bob Smith</name-on-card>
    <card-number>1987987399918277</card-number>
    <exp-date>01/04</exp-date>
  </credit-card>
</purchase-request>
```

Default Namespaces

Lab

Tasks to Perform

- Use a namespace for purchase requests (see notes)
  - Make it the default namespace for the entire document
  - Where do you make this namespace definition?
- In the scope of the `credit-card` element, reset the default namespace to be the one for credit cards (see notes)
  - This element and all of its child elements should belong to this “JavaTunes credit card” namespace
  - The `type` attribute should not be in any namespace

Default Namespaces

Lab

Tasks to Perform

- Use a namespace for currencies (see notes)
  - Use the URI *http://www.monetary.org*
  - Put the `amount` element's **`currency`** attribute in this namespace
- Validate your document, and view it in Design View
  - Notice the namespaces
  - You can also load the document into a browser to view it
- Use a default namespace for orders in the *orderns.xml* document from the last lab (see notes)
  - All the elements in the order should be in this one namespace (see the notes for an explanation as to why this is okay)
  - As before, none of the attributes should be in the namespace

STOP

Lab

Lab 4.1: Simple Schema

- **Purpose**: In this lab, we will create a very simple XML Schema and validate an instance document against that schema
- **Objectives**: Become familiar with XML Schema basics
- **Builds on previous labs**: None
  - The root lab directory where you will do your work for this lab is:

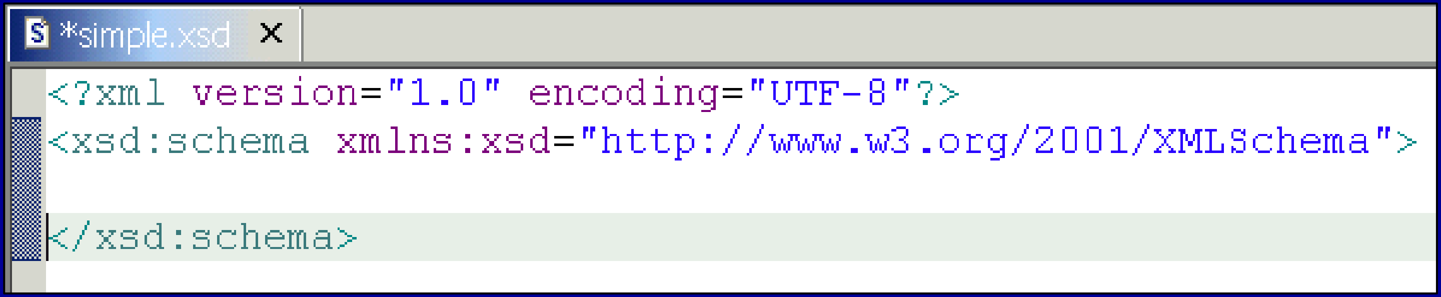
    C:\StudentWork\XMLIntro\workspace\Lab04.1
- **Approximate Time**: 20-30 minutes

Create XML and Schema Files

Lab

Tasks to Perform

- Create a **Project** (**File | New | Project | General| Project**)
  - Call the project **Lab04.1**
- In *your project*, create a file named ***simple.xsd*** which has the schema for documents of type `simple`
  - Create a new Schema doc via **File | New | XML Schema**
  - Next, **modify the namespace declaration** for the XML schema namespace to use the **xsd prefix** - Instead of the default prefix
  - Also **remove the namespace declarations** for the target and "simple" namespaces - We'll add in namespaces later

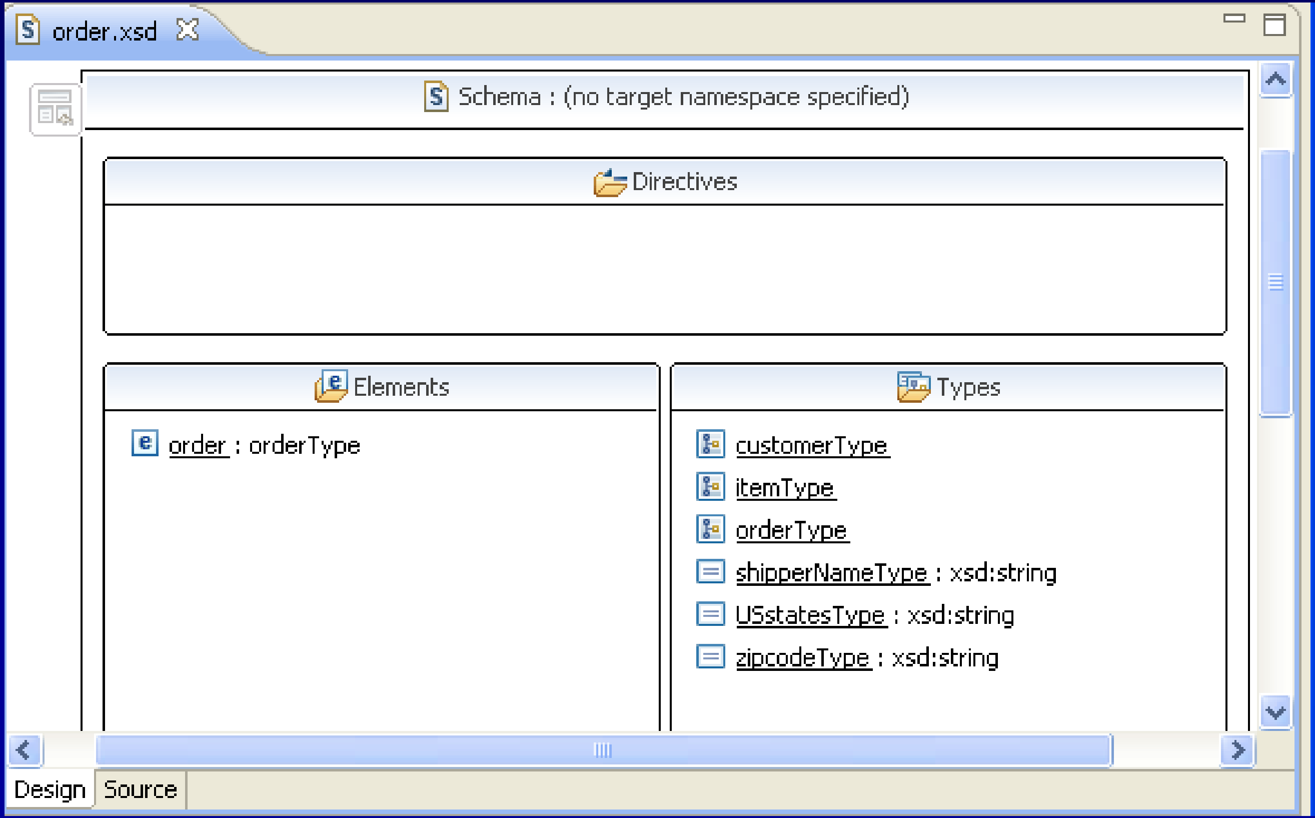


```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

</xsd:schema>
```

Design Schema View

Lab

- Shows the structure of the schema
  - We illustrate with an example order schema below
  - You can edit the schema here, or in the source (they'll stay in sync)
  - Double clicking zooms the editor in on the clicked element, right clicking allows you to modify elements



order : orderType

customerType
itemType
orderType
shipperNameType : xsd:string
USstatesType : xsd:string
zipcodeType : xsd:string

Create XML and Schema Files

Lab

Tasks to Perform

- Add a single element to the schema

  ```
  <xsd:element name='simple' type='xsd:string'/>
  ```

  - You can use design or source view – as you prefer
  - Changes in one are reflected in changes in the other

- **Create an XML document** based on *simple.xsd* - it's easy to use Eclipse to create an XML file based on a schema
  - Right click on the project and select: **New | XML File**
  - Name the file ***simple.xml*** in the first dialog, **Next**, in second dialog, select "**Create file using a DTD or XML Schema File**"
  - **Next**, and select simple.xsd from the Lab04.1 project
  - **Next**, **Finish**
  - Eclipse will create the file with most of the content you need

Validate the XML Document

Lab

Tasks to Perform

- Review ***simple.xml*** which has our very simple XML document in it (example shown at bottom)
  - Note the schema location element *
- Validate your XML file
  - Since you declare a schema file, it will validate against the schema
- If the document is valid, you will get no error messages
- If the document is invalid, you will see an error message(s)

```
<?xml version='1.0'?

<simple
 xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
 xsi:noNamespaceSchemaLocation='simple.xsd'>
  Hey, I'm an XML Document
</simple>
```

Use Different Types

Lab

Tasks to Perform

- Experiment with some of the other simple datatypes
  - `xsd:integer`, `xsd:float` , `xsd:date` are common ones to use
  - The easiest way is to use them one at a time, changing the XML and schema documents each time
- Make sure you try some content that should be invalid
  - For example, use a schema type of xsd:integer, and leave the content as the original text string
  - Or use an element with a name different from <simple>
  - You should get an error(s)

STOP

Lab

Lab 4.2: More Complex Schemas

- **Purpose**: In this lab, we will create a schema that has complex content
  - We will create an XML Schema for a simplified JavaTunes order
- **Objectives**: Work with a more complex schema
- **Builds on previous labs**: Lab 4.1
  - Continue working in your Lab04.1 project
- **Approximate Time**: 30-40 minutes

Element Definitions in a Schema

Lab

- Here are the rules for the content models for our simplified order document
  - An **`order`** has a `customer` (we will ignore `items` for now)
  - A **`customer`** has a `name`, a `street`, an `apt`, a `city`, a `state`, a `zipcode`, and a `shipper`
  - **`shipper`** is an empty element
  - All other elements have character content
- Ignore the attributes for now
  - We will define them in the next lab

Simple Order XML Document

Lab

```
<?xml version='1.0'?>

<!-- JavaTunes order XML document (simplified) -->

<order>
  <customer>
    <name>Susan Phillips</name>
    <street>763 Rodeo Circle</street>
    <apt>1A</apt>
    <city>San Francisco</city>
    <state>CA</state>
    <zipcode>94109</zipcode>
    <shipper/>
  </customer>
</order>
```

Element Definitions in a Schema

Lab

Tasks to Perform

- Create a schema in file ***order.xsd***, with the following types:
  - **`orderType`** - a sequence of one element, `customer`
  - **`customerType`** - a sequence of the 7 `customer` child elements
  - Use an anonymous type for the **`shipper`** element (see notes)
- Then define the document element `order`
- Test your schema by validating the *simpleorder.xml* file
  - We supply this in the project (without the schema location)
  - **NOTE** - be sure to add the schema location attribute to the `order` element, referring to the schema in *order.xsd* (see notes)

Adding a Choice

Lab

Tasks to Perform

- Enhance the content model for `customerType` to support both US and Canadian customers -- allow a choice of either:
  - `state` and `zipcode`

    OR
  - `prov` and `pcode`
  - This is a choice of two sequences
- Again, experiment with the document, checking its validity

STOP

Lab

Lab 4.3: A Complete Order Schema

- **Purpose**: In this lab, we will learn to use occurrence constraints and attributes, and use them to create a complete order schema
  - We will create a complete Schema for a JavaTunes order
- **Objectives**: Work with occurrence constraints and attributes
- **Builds on previous labs**: Lab 4.2
  - Continue working in your Lab04.1 project
- **Approximate Time**: 40-50 minutes

A Complete Order Schema

Lab

- **Purpose** - learn to use occurrence constraints and attributes, and use them to create a complete order schema
- In this lab, we will create a complete schema for a JavaTunes order
- The JavaTunes order schema is *order.xsd*, the one you created in the previous lab
- The JavaTunes order document is *order.xml*, which you used in earlier labs, and which we supply in the lab dir
  - **NOTE** - be sure to add the schema location attribute to the `order` element, referring to the schema in *order.xsd* (see notes)

Element Content Models

Lab

- Here are the rules for the element content models:
  - An **`order`** has a `customer` and **1 or more** `items`
  - A **`customer`** has a `name`, a `street`, an **optional** `apt`, a `city`, a **choice** of `state` and `zipcode` or `prov` and `pcode`, and a `shipper`
  - **`shipper`** is an empty element
  - An **`item`** has a `name`, **1 or more** `artists`, a `releaseDate`, a `listPrice`, and a `price`
  - `releaseDate` has `xsd:date` content
  - `listPrice` and `price` have decimal content; `price` has a **default value** of `9.99`
  - All other elements have character content

Attributes

Lab

- Here are the attribute definitions: (see notes for type formats)
- **`order`**
  - **`ID`** is type `xsd:ID` and is required
  - **`dateTime`** is type `xsd:dateTime` and is required
- **`shipper`**
  - **`name`** is type `xsd:NMTOKEN` and is defaulted to `USMail`
  - **`accountNum`** is type `xsd:string` and is optional with no default
- **`item`**
  - **`ID`** is type `xsd:ID` and is required
  - **`type`** is type `xsd:NMTOKEN` and is defaulted to `CD`

Create the Schema

Lab

Tasks to Perform

- Based on the content model in the previous slides, extend your *order.xsd* schema to conform to it
  - We supply a sample order XML document, *order.xml*, that you can validate against
  - Finish the schema, then try validating *order.xml* against it
  - If your schema is correct, then you should not get any errors.
  - See notes for some of the attribute data

Testing

Lab

Tasks to Perform

- Experiment with the order document and check its validity
  - Give an item two artists -- is the document valid?
  - This order’s customer has no apartment -- is that valid?
  - Remove a required attribute -- is the document valid?
  - Remove an optional attribute -- is the document valid?
  - Change the shipper name to `UPS Ground` -- is that value okay?

- Are your default values being used?
  - Is the `type` attribute of `item` defaulting to CD?
  - Is the `name` attribute of `shipper` defaulting to `USMail`?
  - Is the `price` child element of `item` defaulting to `9.99`? Remember that it must appear as `<price/>` to take the default value

STOP

Lab

[Optional] Lab 4.4: Schema Namespace Support

- **Purpose**: In this lab, we will add namespace support to our schema and provide context-sensitive element definitions for the name elements
- **Objectives**: Work with Schema and namespaces
- **Builds on previous labs**: Lab 4.3
  - Continue working in your Lab04.1 project
- **Approximate Time**: 40-50 minutes

Namespace Support - Optional

Lab

Tasks to Perform

- Make a copy of *order.xsd* and name it ***orderns.xsd***
  - Work in *orderns.xsd* for this lab – this way we'll preserve the non-namespace version
- Add support for the JavaTunes **order** namespace to *orderns.xsd*
  - You can use the example just shown as your model
  - Recall that in our earlier namespace lab, we used a single order namespace – look at ***orderns.xml*** in this project to see the namespace declaration
  - Recall also that you'll need to use an `elementFormDefault` attribute (see notes)

Context-Sensitive `names` - Optional

Lab

Tasks to Perform

- Continuing in *orderns.xml*, create a complex type for the **`customer name`** element
  - You can use the example shown a few slides back as your model
  - No change is necessary to the `item name` element definition
- Add the necessary attributes to the supplied *orderns.xml* to refer to the schema
  - You will use **`xsi:schemaLocation`** instead of `xsi:noNamespaceSchemaLocation`
  - And recall that the value of `xsi:schemaLocation` is a **pair**: ***namespace-URI location-of-schema*** (separated by a space)
  - You can use the example just shown as your model
- Validate the *orderns.xml* document
  - This document uses the order namespace

STOP

Lab

[Optional] Lab 4.5: Advanced Topics

- **Purpose**: In this lab, we will use derivation of simple types to refine some of the datatypes in our JavaTunes order schema
- **Builds on previous labs**: Lab 4.4
  - Continue working in your Lab04.1 project
- **Approximate Time**: 40-50 minutes

Derived Types

Lab

Tasks to Perform

- Create simple types to provide the following refinements:
  - Restrict the **`shipper name`** attribute to allow the following values: `USMail`, `FedEx`, `UPS`
  - Create a type for zipcodes (see notes)
  - Restrict the **`state`** element content to actual state abbreviations
  - Don’t actually do this for 50 states
  - Pick your 3 favorite states and use them

Testing Our Derived Types

Lab

- Modify your schema appropriately to use these new types
  - Simply refer to them by name in the appropriate `type` attributes, e.g.,
    `<element name='zipcode' type='zipcodeType'/>`
- Test your schema by changing some of the data in an order
  - Try invalid values for the `shipper name` attribute
  - Try 5-digit and zip-plus4 values for the `zipcode` element
  - Try invalid values, as well
  - Try invalid values for the `state` element
- You can do this work in *order.xsd* or *orderns.xsd*

STOP

Lab

[Optional] Lab 5.1 : Generating XML

- **Purpose**: In this lab, we will create a set of Java classes that can generate XML for an order document
- **Objectives**: Explore a simple way to generate XML documents
  - You will also set up the lab working environment
- **Builds on previous labs**: None
  - The root lab directory where you will do your work for this lab is:

    C:\StudentWork\XMLIntro\workspace\Lab05.1
    - There is starter code in the folder
- **Approximate Time**: 30-40 minutes

Generating XML

Lab

- In this lab, you will write a **`toXML()`** method and generate an XML document from a set of application objects

- The JavaTunes application classes have been provided -- `Order`, `Customer`, `Shipper`, and **`Item`**
  - The classes for this lab are all in a package, genXML, and so will appear in a ***genXML*** folder below the project directory

- **But you need to complete `Item`'s `toXML()` method**
  - First though, we'll set up our lab environment for these labs

Setup

Lab

- Our base working directory for this part of the course will be **C:\StudentWork\XMLIntro**
  - This directory was created when we extracted the Setup zip (assuming extraction to to C:\. **If not, please adjust accordingly**)
  - Labs will be in the directory: **StudentWork\XMLIntro\workspace**
  - The root lab directory where you will do your work for this lab is:

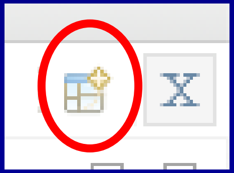
C:\StudentWork\XMLIntro\workspace\Lab05.1

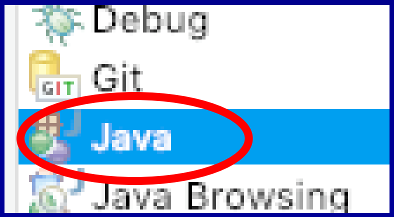
Workbench and Java Perspective

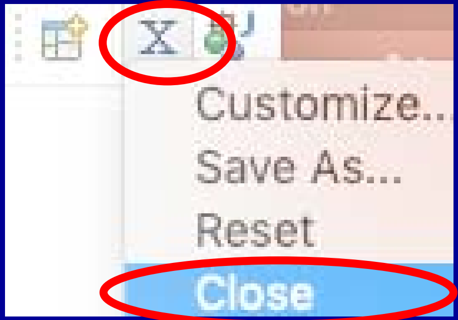
Lab

Tasks to Perform

- You'll likely be in a XML Perspective from the earlier labs
  - If already in a Java perspective, then just remain in it
- **To open a Java perepective**, click the Perspective icon at the top right of the Workbench, and select XML (below left, and center)
  - In the next dialog, select the Java Perspective
  - Close the XML perspective by right clicking its icon, and selecting close (below right)





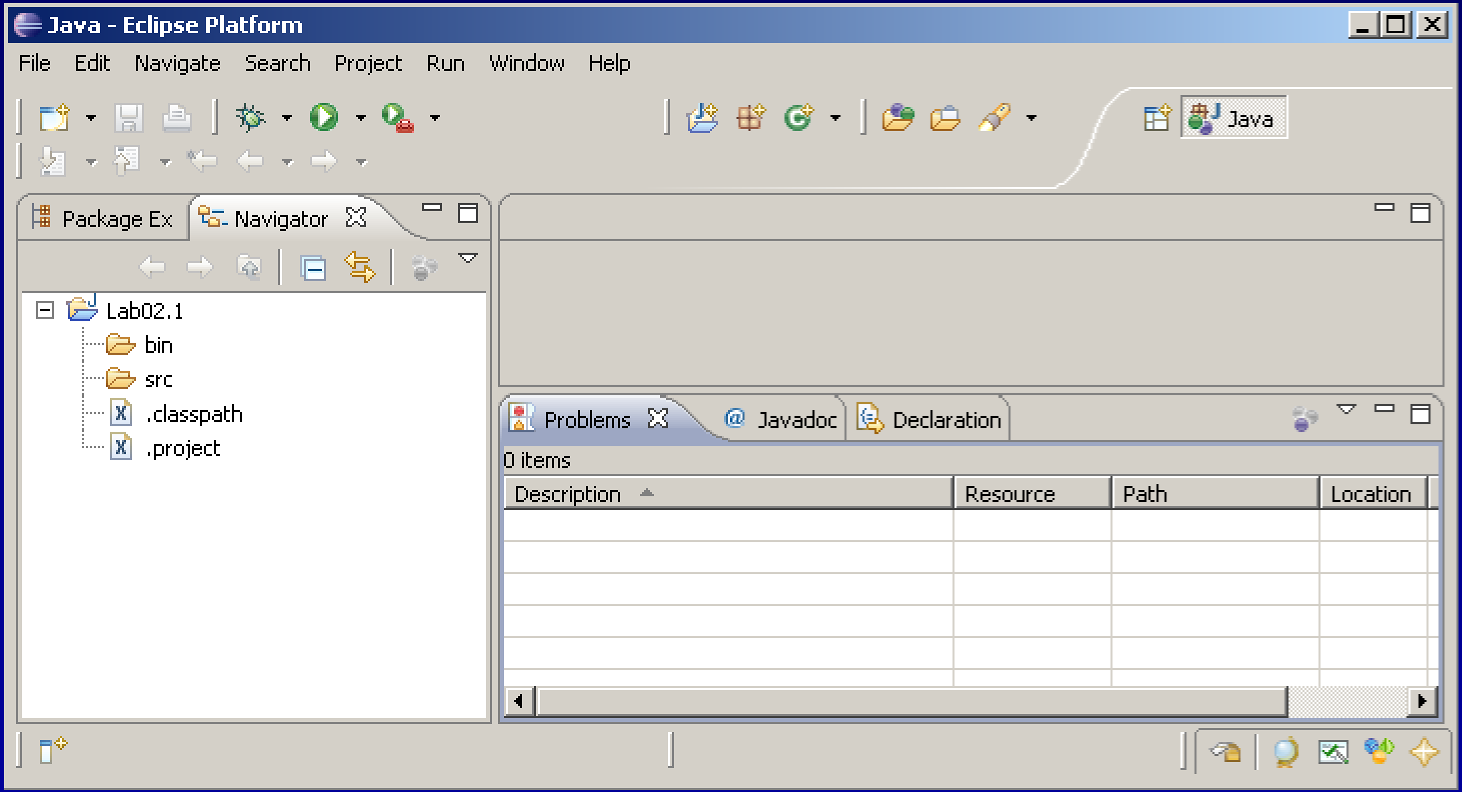


Unclutter the Workbench

Lab

Tasks to Perform

- Let's unclutter the Perspective by closing some views
  - Close the Task List, Outline and Hierarchy views (click on the X)
  - Open the Navigator View (**Window | Show View | Navigator**)
  - You can save this as the default if you want



0 items

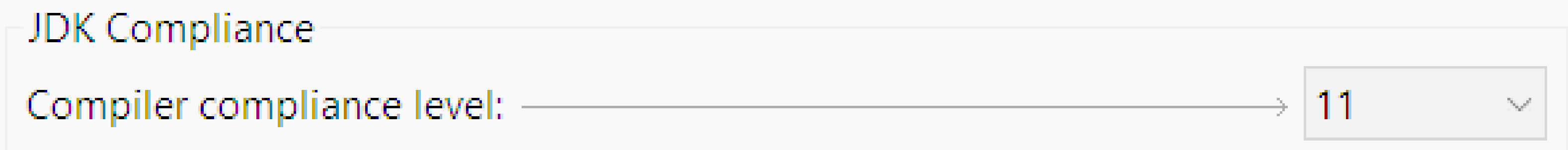
Description	Resource	Path	Location

Set Up Java 11 Usage

Lab

Tasks to Perform

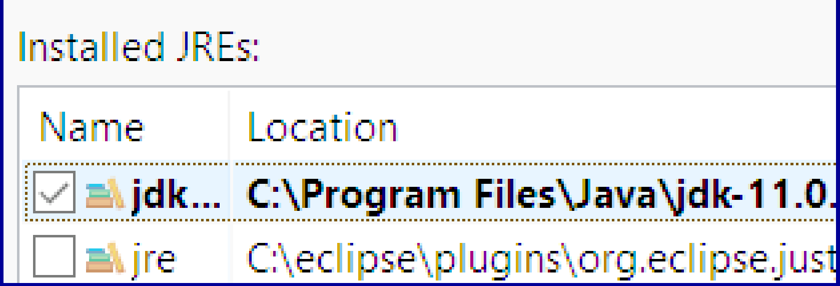
- Eclipse ships with its own JDK (and version varies between releases)
  - We'll now configure Java projects to always use Java 11 (1)
- Go to menu **Window > Preferences > Java > Compiler**
  - Set **Compiler compliance level** to **11**



JDK Compliance

Compiler compliance level: 11

- Go to **Preferences > Java > Installed JREs**
  - If your Java 11 compiler is not there, add it (2)
  - Check the Java 11 Java install to make it the default
  - Click **Apply and Close**



Installed JREs:

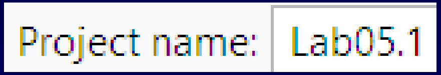
Name	Location
☑ jdk...	C:\Program Files\Java\jdk-11.0.
□ jre	C:\eclipse\plugins\org.eclipse.just

Create a Project for our Application

Lab

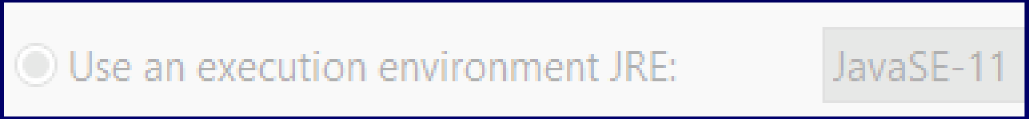
Tasks to Perform

- Create a **Java Project** (**File** | **New** | **Project** | **Java** | **Java Project**)
  - Call the project **Lab05.1**



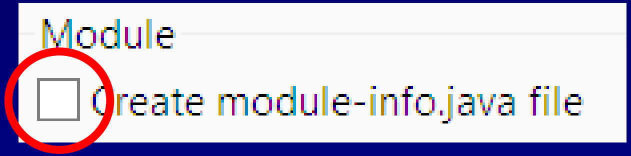
Project name: Lab05.1

  - Eclipse will automatically set the project directory to *Lab05.1*
  - Should use Java 11 also



○ Use an execution environment JRE: JavaSE-11

  - **Uncheck** "Create module-info.java file"
  - Click **Finish**



Module

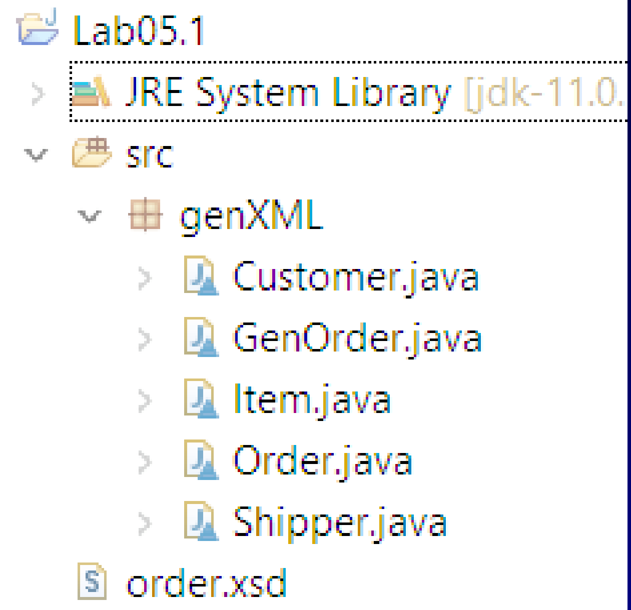
□ Create module-info.java file

- We supply the required Java classes in the `genXML` package for you
  - As well as any XML files that you'll need
  - You'll just need to finish up `Item`, and then run a supplied program to generate the XML

Generating XML

Lab

- You should have this now in the Package Explorer (on the right):
- In this lab, you will modify **`Item.java`** to add the **`toXML()`** method and generate an XML document from a set of application objects
- The JavaTunes application classes have been provided -- `Order`, `Customer`, `Shipper`, and **`Item`**
  - **But you will need to complete `Item`'s `toXML()` method**
- The classes for this lab are all in the `genXML` package,
  - You can see them at right in the ***genXML*** package folder below the project directory



```
Lab05.1
  > JRE System Library [jdk-11.0.
  v src
    v genXML
      > Customer.java
      > GenOrder.java
      > Item.java
      > Order.java
      > Shipper.java
    order.xsd
```

Kick off the Process

Lab

- The provided **Order** class will be the "controller" for the whole process and its `toXML()` method starts things off
  - Since `order` is the document element, it makes perfect sense to start the process in this object
  - **NOTE** that this method has already been coded for you

```
public String toXML() {
  StringBuffer buffer = new StringBuffer();
  buffer.append("<?xml version='1.0'?>");
  buffer.append("<order ID=...>");
  // ...
}
```

Tasks to Perform

- **Open *Item.java*, and finish the `toXML()` method of `Item`**
  - The supplied implementation just returns null
  - You must modify it to return the appropriate XML elements

Kick off the Process

Lab

Tasks to Perform

- Run the provided **`GenOrder`** class, which creates the JavaTunes objects, builds an order, then generates the XML
- From within Eclipse: right-click on *GenOrder.java* and select **Run As | Java Application**
  - This will create the output in the file ***genorder.xml*** under the root project directory
  - If you want to change the name of the output file, you can do so by running it as: : **`java GenOrder`** ***`output-file`***
  - To pass an argument with Eclipse, see the notes
- **Refresh your project**, so the new file is recognized in Eclipse
  - Open the generated XML file, and review it in both design and source views (what do you notice in the source view – is this OK?)
  - Right click on it and **validate** it – it includes a schema reference to the *order.xsd* file that is also in the root of the project

Examine the Results

Lab

- Some things to think about:
  - What if an object contains data with a **<** or **&** character?
  - How would you handle this?
  - What if an object contains data that is invalid with respect to the schema, e.g., a `Customer` object contains an `m_state` value of `abc`?
  - How would you handle this? What are your options?

STOP

Lab

Lab 5.2: Creating Parsers

- **Purpose**: In this lab, we will write classes that instantiate SAX and DOM parsers
  - We'll use the previous examples as a guide,
- **Objectives**: Work with JAXP parsers
- **Builds on previous labs**: None
  - The root lab directory where you will do your work for this lab is:

    C:\StudentWork\XMLIntro\workspace\Lab05.2
- **Approximate Time**: 30-40 minutes

Instantiating Parsers

Lab

Tasks to Perform

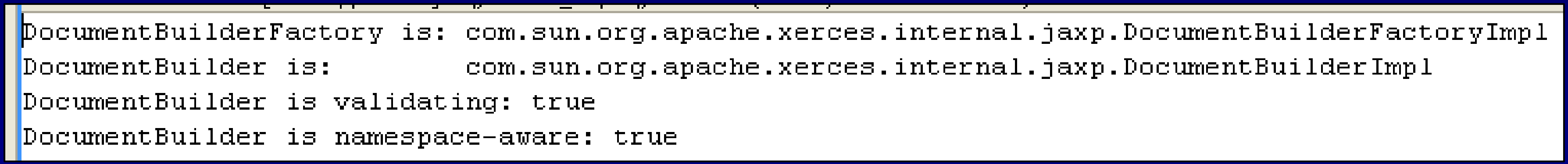
- Create a **Java Project** called **Lab05.2**
  - The directory for this project already exists from the setup
  - Remember to **uncheck** "Create module-info.java file"
- Write a **`SAXTest`** class with a main method that instantiates a "Schema-validating, namespace-aware SAX parser"
  - Use the JAXP default parser
  - Put the class in package `com.javatunes.sax`
- Write a **`DOMTest`** class with a main method that instantiates a "Schema-validating, namespace-aware DOM parser"
  - Use the JAXP default parser
  - Put the class in package `com.javatunes.dom`
- Include code displaying the factory and parser (see notes)
  - We're not parsing yet – just creating parsers

Run your programs

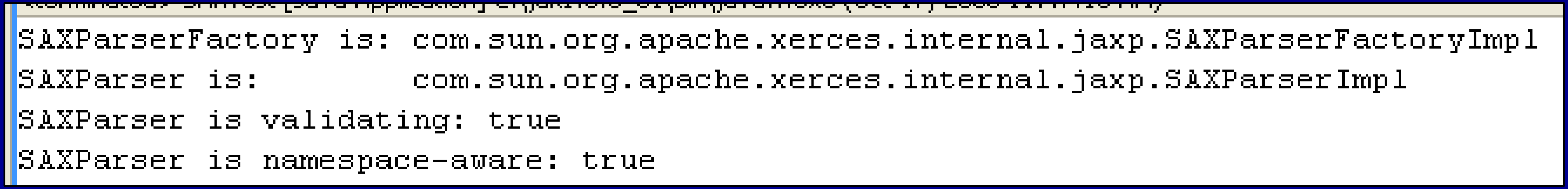
Lab

Tasks to Perform

- Run both of these programs and see how they work
  - Right click on the program, select Run As | Java Application
  - You should see something like that below (DOMTest on top)
  - That's it – you've created parsers – next we'll actually parse



```
DocumentBuilderFactory is: com.sun.org.apache.xerces.internal.jaxp.DocumentBuilderFactoryImpl
DocumentBuilder is:        com.sun.org.apache.xerces.internal.jaxp.DocumentBuilderImpl
DocumentBuilder is validating: true
DocumentBuilder is namespace-aware: true
```



```
SAXParserFactory is: com.sun.org.apache.xerces.internal.jaxp.SAXParserFactoryImpl
SAXParser is:        com.sun.org.apache.xerces.internal.jaxp.SAXParserImpl
SAXParser is validating: true
SAXParser is namespace-aware: true
```

STOP

Lab

Lab 6.1: Your First Parse

- **Purpose**: In this lab, we will write some simple handlers for the SAX parser
- **Objectives**: Work with SAX parsing
- **Builds on previous labs**: Lab 5.2
  - Continue working in your Lab05.2 project
- **Approximate Time**: 30-40 minutes

Your First Parse

Lab

Tasks to Perform

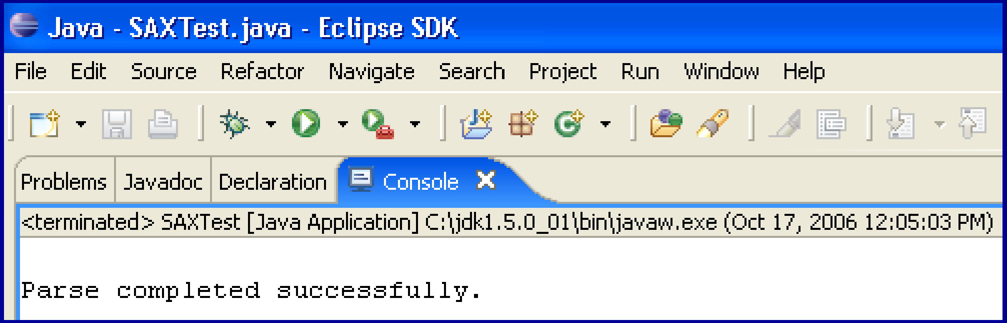
- In your project, create a **`com.javatunes.sax.OrderHandler`** class
  - Subclass `DefaultHandler`, as in the manual examples
  - We'll start overriding the parse event callback methods in the next lab
- Enhance **`SAXTest`** to set up your handler and an input source and parse some files -- use the example in the manual slides as your model
  - Also modify `SAXTest` to take a command line argument which should be the name of the file to parse
- The files to parse, and an associated schema are already in the project directory – they include:
  - ***test.xsd***: A schema file that we will be validating against
  - ***test.xml***, ***testWarning.xml***, ***testError.xml***, and ***testFatalError.xml***: XML files that use test.xsd, and have various levels of problems in them (as described by their names)

Running

Lab

Tasks to Perform

- Run `SAXTest` to process the files ***test.xml***, ***testWarning.xml***, ***testError.xml***, and ***testFatalError.xml***
  - The format is: **`java SAXTest`** ***`filename_argument`***
  - For Eclipse, you need to fill in this argument in the run configuration
  - Right-click *SAXTest.java*, select **Run As | Run. Configurations…** and select the SAXTest configuration
  - In the **Arguments** tab, enter "*test.xm*l" into the **Program arguments**
  - Click the **Run** button at the bottom
- You'll need to change the argument to run against each file


Java - SAXTest.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console

```
<terminated> SAXTest [Java Application] C:\jdk1.5.0_01\bin\javaw.exe (Oct 17, 2006 12:05:03 PM)

Parse completed successfully.
```

Reporting the Parse Results

Lab

Tasks to Perform

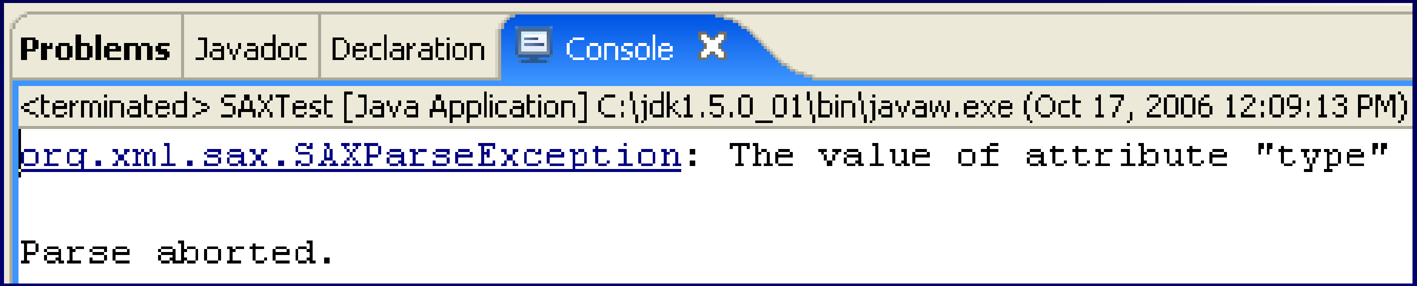
- Create an exception handler for SAXException -- print a message indicating that the parse was aborted
  - If no exceptions are thrown, the parse was successful
- You can catch ParserConfigurationException and IOException individually or just catch Exception

```
try {
  // set up factory and parser, handler and XML source
  parser.parse(...);                   // go parser go
  System.out.println("\nParse completed successfully.");
}
catch (SAXException e) {
  System.out.println(e);
  System.out.println("\nParse aborted.");
}
catch (Exception e) {  // catches the other exceptions
  System.out.println(e);
}
```

What You Will See (and Not See)

Lab

- All the files should parse successfully except for ***testFatalError.xml***
  - Which should abort, as shown below



```
<terminated> SAXTest [Java Application] C:\jdk1.5.0_01\bin\javaw.exe (Oct 17, 2006 12:09:13 PM)
org.xml.sax.SAXParseException: The value of attribute "type"

Parse aborted.
```

- **You will not see any document content**
  - **Our handler is receiving parse events** from the parser, but it doesn't do anything with them yet -- it will soon
  - Remember that the methods inherited from `DefaultHandler` are empty, i.e., `{ /* no op */ }`
- Examine the four XML documents with a text editor
  - Can you figure out why we are getting this behavior?
  - We'll discuss the different types of SAX parse errors next

STOP

Lab

Lab 6.2: Sax Error Handling

- **Purpose**: In this lab, we will write error handlers for the SAX parser
- **Objectives**: Work with SAX Error Handling
- **Builds on previous labs**: Lab 6.1
  - Continue working in your Lab05.2 project
- **Approximate Time**: 20-30 minutes

SAX Error Handling

Lab

Tasks to Perform

- Part A - in `OrderHandler`, write the error handling methods provided on the next slide
  - These override the methods inherited from `DefaultHandler`
  - Process the four XML test files from the last lab and note the results
- Part B - change the code in the **`error()`** method to abort the parse on the occurrence of a validity error -- process the files
  - How do you tell the parser to abort the parse?
- Part C - turn off validation in the parser -- process the files
  - Which `ErrorHandler` method gets invoked on validity errors?
  - How does turning off validation affect the invocation of that method?
  - **NOTE** - turn validation back on when finished and change the `error()` method back so that it does not abort on validity errors

SAX Error Handling

Lab

```
public void warning(SAXParseException e)
throws SAXException {
  System.out.println("\n---- Warning at line " +
    e.getLineNumber());
  System.out.println(e.getMessage());
}

public void error(SAXParseException e)
throws SAXException {
  System.out.println("\n++++ Error at line " +
    e.getLineNumber());
  System.out.println(e.getMessage());
}

public void fatalError(SAXParseException e)
throws SAXException {
  System.out.println("\n**** Fatal error at line " +
    e.getLineNumber());
  System.out.println(e.getMessage());
}
```

STOP

Lab

Lab 6.3: Handling Parse Events

- **Purpose**: In this lab, we will start parsing the contents of the document
- **Objectives**: Work with SAX parse events
- **Builds on previous labs**: Lab 6.2
  - Continue working in your Lab05.2 project
- **Approximate Time**: 20-30 minutes

Handling Parse Events

Lab

Tasks to Perform

- Write the code necessary to store the document locator in an instance variable when the parser calls **`setDocumentLocator()`**
  - There is an example of how to do this where we discussed the document locator
- Write **`startDocument()`** and **`endDocument()`** methods
  - Print a message indicating the parse event that is occurring
  - Use the document locator to included the line number of each event in the output
  - A suggested implementation is shown in the notes below

Reporting Elements

Lab

- Write **startElement()** and **endElement()** methods

```
public void startElement(String nsURI, String localName,
                         String qName, Attributes atts)
throws SAXException
{
  System.out.print("<" + qName);
  for (int i = 0; i < atts.getLength(); i++)
  {
    System.out.print(" " + atts.getQName(i) +
                     "=" + atts.getValue(i));
  }
  System.out.println(">");
}
```

```
public void endElement(String nsURI, String localName,
                       String qName)
throws SAXException
{
  System.out.println("</" + qName + ">");
}
```

Testing

Lab

- Prepare to test out your new handler methods on an order
  - Work with ***order.xml*** file, which has a schema location attribute for *order.xsd*, and ***orderNoSchema.xml***, which has no schema associated
  - All these files are already in your lab directory – they should be familiar to you

Tasks to Perform

- Process both of these files with `SAXTest` -- do the elements and attributes appear?
  - You can pass the filename arg in the run configuration as before
  - What do you notice when you process *orderNoSchema.xml*? Turn validation off -- what changed? (see notes)
  - **NOTE** - you will not see element content yet; that's next
  - **NOTE** - turn validation back on when finished

STOP

Lab

Lab 6.4: Getting Content Out of Elements

- **Purpose**: In this lab, we will get content from the elements in the document
- **Objectives**: Extract content from elements using SAX
- **Builds on previous labs**: Lab 6.3
  - Continue working in your Lab05.2 project
- **Approximate Time**: 20-30 minutes

Getting Content out of Elements

Lab

Tasks to Perform

- Write a `characters()` method (see below) and test it
  - With validation turned on, process *order.xml* -- does the element content appear? Is there any extra whitespace in the output? Explain
  - Turn validation off and process *order.xml* -- what do you notice regarding extra whitespace? Explain
  - Keep validation off and process *orderNoSchema.xml* -- what do you notice regarding extra whitespace? Explain
  - NOTE - turn validation back on when finished

```
public void characters(char[] data, int start, int length)
throws SAXException
{
  String content = new String(data, start, length);
  System.out.println(content);
}
```

Handling Whitespace

Lab

- Whitespace is significant and sent to `characters()` when:
  - There is no schema (DTD or XML Schema), because the parser knows nothing about the elements' content models
  - Using XML Schema

Tasks to Perform

- Enhance `characters()` to ignore the whitespace that we consider to be insignificant
  - Use `String`'s **`trim()`** method -- see `String`'s Javadoc for details
  - In addition to using `trim()`, use an `if` statement to print content only if the trimmed `String`'s length is nonzero
- Repeat the tests on the previous slide
  - Do you notice any difference?

Processing Instructions - Optional

Lab

Tasks to Perform

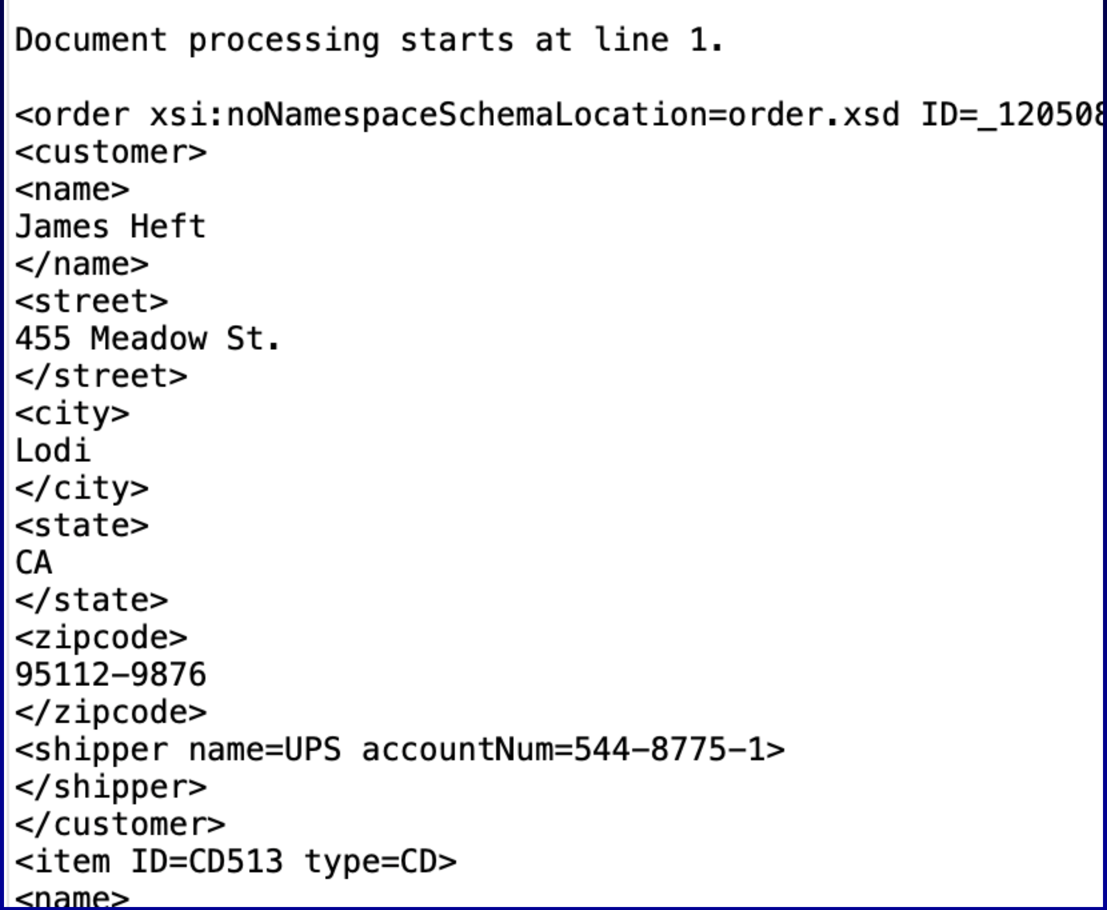
- Write a **processingInstruction()** method
- Create a file ***notepad.xml*** (see notes) with PI to test it out
  - The PI in it will cause our handler to invoke *notepad*

```
public void processingInstruction(String target,
                                  String data)
throws SAXException
{
  try
  {
    Runtime.getRuntime().exec(target + " " + data);
  }
  catch (java.io.IOException e)
  {
    System.out.println("Unable to invoke " + target);
  }
}
```

Results

Lab

- With SAXTest parsing *order.xml* with validation turned OFF:



```
Document processing starts at line 1.

<order xsi:noNamespaceSchemaLocation=order.xsd ID=_120508
<customer>
<name>
James Heft
</name>
<street>
455 Meadow St.
</street>
<city>
Lodi
</city>
<state>
CA
</state>
<zipcode>
95112-9876
</zipcode>
<shipper name=UPS accountNum=544-8775-1>
</shipper>
</customer>
<item ID=CD513 type=CD>
<name>
```

STOP

Lab

Lab 6.5: State Dependent Processing

- **Purpose**: In this lab, we will get content from the elements in the document
- **Objectives**: Extract content from elements using SAX
- **Builds on previous labs**: Lab 6.4
  - Continue working in your Lab05.2 project
- **Approximate Time**: 20-30 minutes

State-Dependent Processing

Lab

- We will now process only desired content -- we will output the content as before
  - We will provide the target element name on the command line

    `java SAXTest` ***`file target-element`***

Tasks to Perform

- Add appropriate state variables and a constructor for initialization -- use the example as your model
  - Also, **provide a default (no-argument) constructor** (used later)
    - This constructor doesn't have to do anything, i.e., { }
- In **`startElement()`**:
  - Turn the `boolean` **on if** called with the target element name
  - **If** in the target element, print out the start-tag and any attributes, as before

State-Dependent Processing

Lab

Tasks to Perform

- In **`endElement()`**:
  - **If** in the target element, print out the end-tag, as before
  - Turn the `boolean` **off if** called with the target element name
  - OPTIONAL - terminate parse early - see notes below
- In **`characters()`**:
  - **If** in the target element, print out the content, as before
- In `SAXTest`'s `main()` method:
  - Be sure to instantiate `OrderHandler` with the command line argument for the target element name
  - If `SAXTest` is invoked via

    `java SAXTest` ***`file`*** ***`target-element`***

    this would be: `new OrderHandler(`**`args[1]`**`)`

State-Dependent Processing

Lab

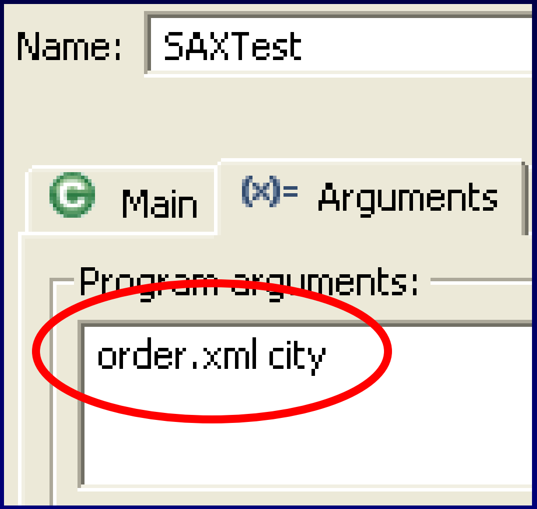
Tasks to Perform

- Test it

  ```
  java SAXTest order.xml target-element
  ```

  should show:

  - *`target-element`*'s start-tag and attributes, as before
  - *`target-element`*'s content, as before
  - *`target-element`*'s end-tag, as before
  - **And nothing else**
- Use Eclipse: **Run As | "Run Configurations ..." | "Arguments" tab | [Run]**
  - If *`target-element`* has child elements, you should see each one's start-tag and attributes, content, and end-tag, as well
- Try it with different target elements
  - **`customer`** shows all customer information, `item` shows items, etc.
  - **`artist`** shows all the artists that created the items in the order
- This is the type of thing you for which you will use SAX



Name: SAXTest

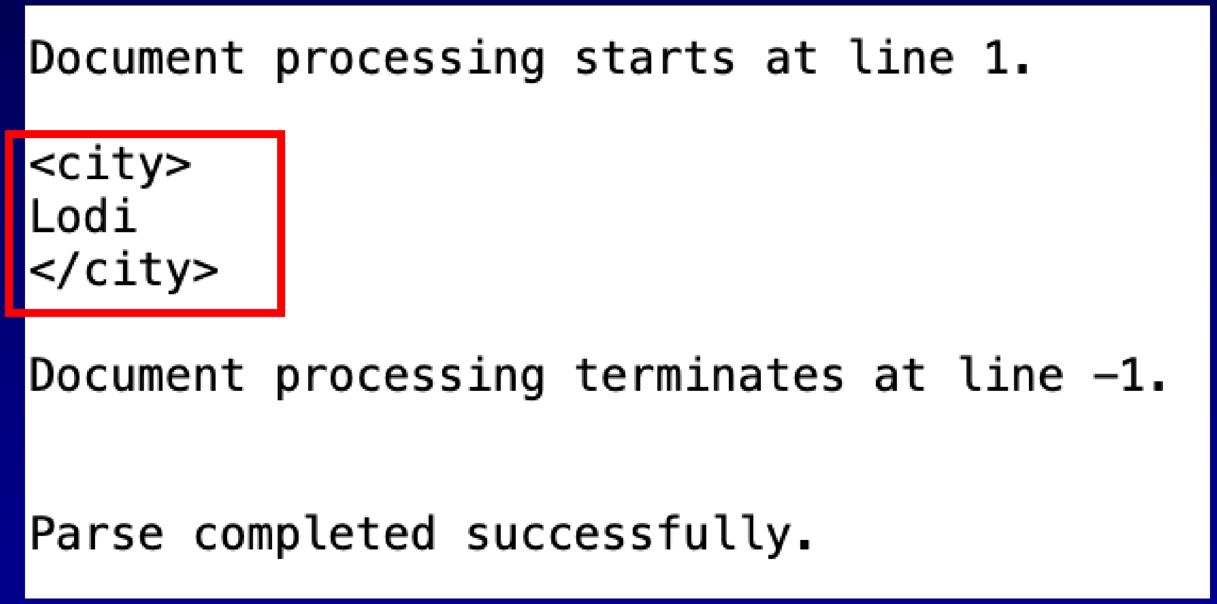
Program arguments:

order.xml city

Results

Lab

- The console output for this parse:



```
Document processing starts at line 1.

<city>
Lodi
</city>

Document processing terminates at line -1.


Parse completed successfully.
```

STOP

Lab

Lab 7.1: Getting a DOM Tree From a Parser

- **Purpose**: In this lab, we will configure our DOM parser, and get a DOM tree from the parser
- **Objectives**: Work with DOM parsing
- **Builds on previous labs**: Lab 5.2
  - Continue working in your Lab05.2 project
- **Approximate Time**: 25-35 minutes

Getting a DOM Tree from a Parser

Lab

Tasks to Perform

- In `DOMTest`, configure the factory to produce a parser that is:
  - Validating and namespace-aware
  - Ignoring insignificant whitespace
  - **Not** ignoring comments
  - Expanding entity references
  - Coalescing CDATA sections

- Remember to set these properties on the factory **before** getting the parser from it

Getting a DOM Tree from a Parser

Lab

Tasks to Perform

- Add code to `DOMTest` to get a `Document` from an `InputSource`
  - You can use the example as your model
  - You can use an instance of `OrderHandler` (from the SAX labs) as your `ErrorHandler` (see notes)
  - Remember that the parser returns a **`Document`** object this time
- Use Eclipse to run `DOMTest`: right-click on DOMTest.java, select **Run As | "Run Configurations..."**
  - Select the DomTest configuration
  - In the "Arguments" tab, just have "***order.xml***" nothing else
- Next try it for ***orderNoSchema.xml***
  - *order.xml* should parse successfully
  - *orderNoSchema.xml* should cause your error handler to be invoked
- **NOTE** - you will not see any content yet; that’s next

Error Handling Behavior

Lab

Tasks to Perform

- You can also use `DOMTest` to process our four test files
  - *test.xml*, *testWarning.xml*, *testError.xml*, and *testFatalError.xml*
  - Your error handler should be invoked as in the SAX labs
- Observe the behavior of the parser if no error handler has been registered
  - Comment out the code that registers your error handler with the parser and process *order.xml*, *orderNoSchema.xml*, and the four test files
  - Note the results

STOP

Lab

Lab 7.2: Getting Node Data

- **Purpose**: In this lab, we will configure our DOM parser, and get a DOM tree from the parser
- **Objectives**: Work with DOM parsing
- **Builds on previous labs**: Lab 7.1
  - – Continue working in your Lab05.2 project
- **Approximate Time**: 30-40 minutes

Getting Node Data

Lab

Tasks to Perform

- Create a new class **`com.javatunes.dom.DOMUtilities`** **(File | New | Class)**
  - Implement the following utility method to read and print node data

```
public static void printNode(Node node)
{
  // first print the node name and node type
  System.out.print(node.getNodeName() +
    " type=" +     node.getNodeType());
  // then print the node's attributes (if any)
  NamedNodeMap atts = node.getAttributes();
  if (atts != null) {// only element nodes have attributes
    for (int i = 0; i < atts.getLength(); i++)
    {
      Node attrib = atts.item(i);
      System.out.print(" " + attrib.getNodeName() + "=" +
                             attrib.getNodeValue());
    }
  } // continued ...
```

Getting Node Data

Lab

```
  /* ... continued
  now print the node value */
  String nodeValue = node.getNodeValue();
  if (nodeValue != null)  // not all nodes have a value
  {
    System.out.println(" value=" + nodeValue);
  }
  else
  {
    System.out.println();
  }
} // end printNode()
```

Navigating the DOM Tree

Lab

Tasks to Perform

- Implement the following code to recursively walk the tree
  - Add this method to DOMUtilities

```
public static void walkTree(Node node)
{
  // print current node
  printNode(node);

  // get this node's children
  NodeList children = node.getChildNodes();

  // go through children, calling this method recursively
  // until at end of node list
  for (int i = 0; i < children.getLength(); i++)
  {
    Node child = children.item(i);
    walkTree(child);
  }
}
```

Start off the Process

Lab

Tasks to Perform

- Implement the code below in the `main()` method of `DOMTest`, after the `Document` has been obtained
- **NOTE** that the `Document` object represents the ***root of the DOM tree*** -- the root of the tree is **not** the root element
  - It is "above" the root element, and its children include the items in the prolog as well as the root element
- Process *order.xml* and *orderNoSchema.xml* -- note the output

```
// parse the input source (done already)
doc = parser.parse(source);
System.out.println("\nParse completed successfully.");

// walk the DOM tree, starting at the root of the tree
DOMUtilities.walkTree(doc);
```

Notes on the Output

Lab

- You will see familiar things like element names and the values of text nodes
  - Element nodes are type 1; text nodes are type 3 and contain the document content
  - **NOTE - Xerces may not be eliminating insignificant whitespace**
- Note also some of the other types of nodes, particularly:
  - The document node (type 9)
  - Comment nodes (type 8)
  - If the document has PIs, are they showing up as nodes (type 7)?
  - Note which node types have an explicit name and which have a value

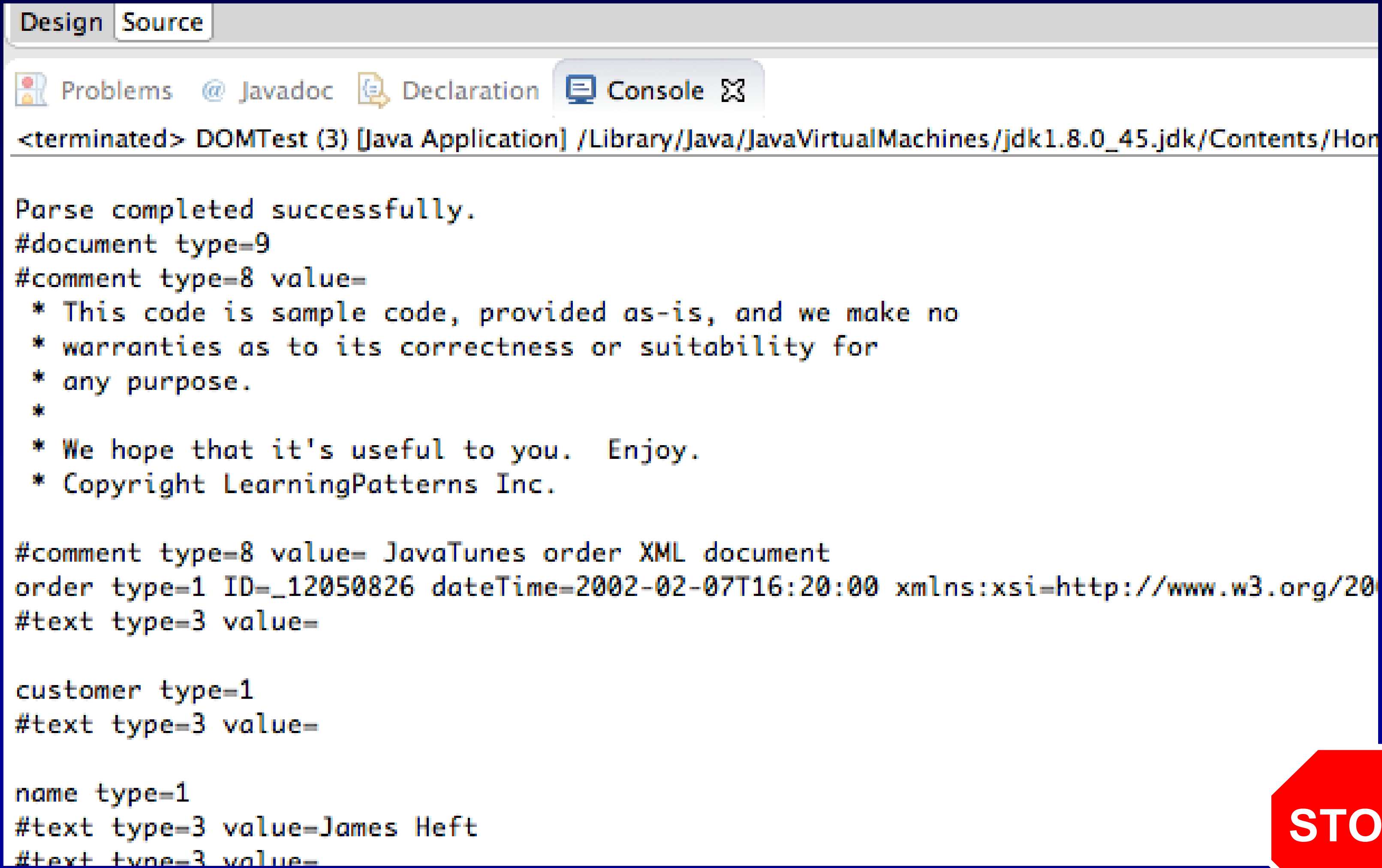
Tasks to Perform

- Turn validation off and reprocess the files -- note the output
  - Is the schema being read? Turn validation back on when finished

Results

Lab

- Parsing *order.xml*:



```
<terminated> DOMTest (3) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Hon

Parse completed successfully.
#document type=9
#comment type=8 value=
 * This code is sample code, provided as-is, and we make no
 * warranties as to its correctness or suitability for
 * any purpose.
 *
 * We hope that it's useful to you.  Enjoy.
 * Copyright LearningPatterns Inc.

#comment type=8 value= JavaTunes order XML document
order type=1 ID=_12050826 dateTime=2002-02-07T16:20:00 xmlns:xsi=http://www.w3.org/20
#text type=3 value=

customer type=1
#text type=3 value=

name type=1
#text type=3 value=James Heft
#text type=3 value=
```

STOP

Lab

Lab 7.3: Modifying the DOM Tree

- **Purpose**: In this lab, we will work with the DOM tree, deleting various elements and then printing out the tree
- **Objectives**: Learn to delete elements from the DOM tree
- **Builds on previous labs**: Lab 7.2
  - Continue working in your Lab05.2 project
- **Approximate Time**: 30-40 minutes

Modifying the DOM Tree

Lab

Tasks to Perform

- In DOMUtilities, implement the following method, which prunes the tree of named elements
  - Notice how we handle the “liveness” of the node collection

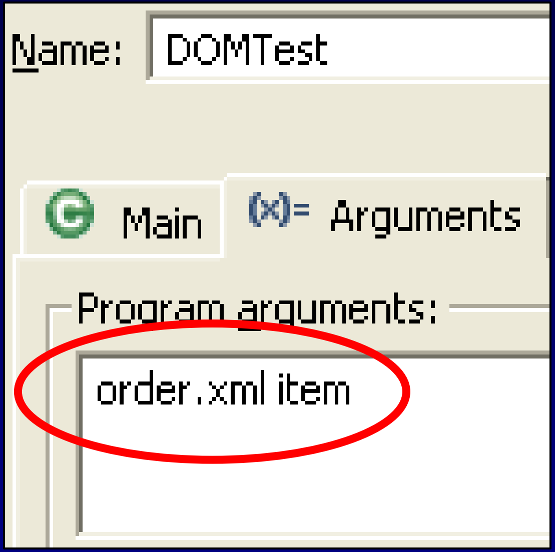
```
public static void deleteElements(Document doc,
                                  String tagname) {
  // query (entire) tree for the named element nodes
  NodeList list = doc.getElementsByTagName(tagname);

  // iterate through, from length-1 to 0 ("backwards")
  // collection automatically shrinks when node is deleted
  for (int i = list.getLength() - 1; i >= 0; i--)  {
    // get node to be deleted (tbd) and its parent
    Node tbd = list.item(i);  // process right-to-left
    Node parent = tbd.getParentNode();
    parent.removeChild(tbd);
  }
}
```

Testing

Lab

Tasks to Perform

- Test your tree modification code from `DOMTest`'s `main()` method
- After the call to `DOMUtilities.walkTree()`, invoke **`DOMUtilities.deleteElements()`**
  - Pass in a reference to the `Document` and an element name
- Invoke `DOMUtilities.walkTree()` again
  - Have the specified element(s) been deleted?
- Use a command line argument for the element name, e.g., to delete the **`item`** elements[1] you would run it like this:
  - Eclipse: right-click *DOMTest.java*, **Run As | "Run Configurations..." | "Arguments" tab |**
  - Provide program arguments: **"order.xml item"**



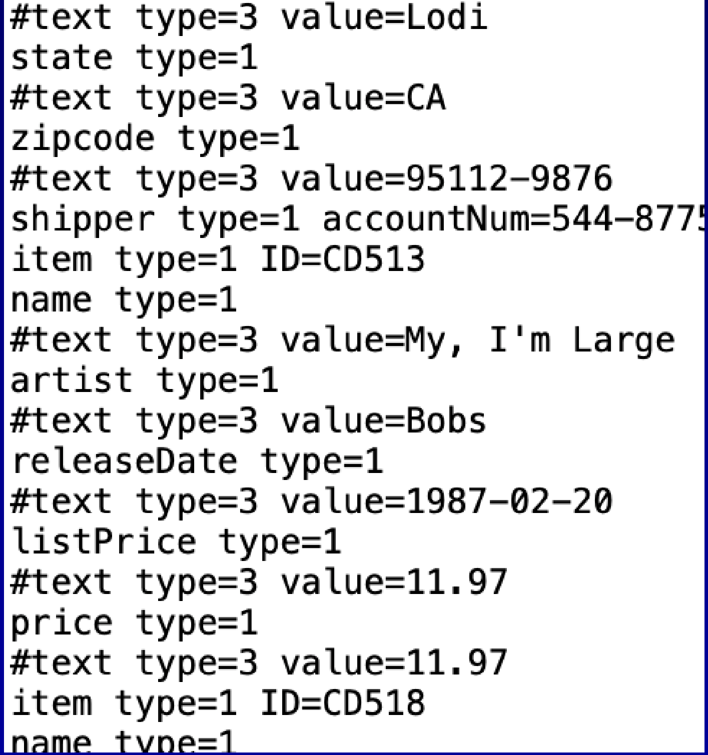
Name: DOMTest

Program arguments: order.xml item

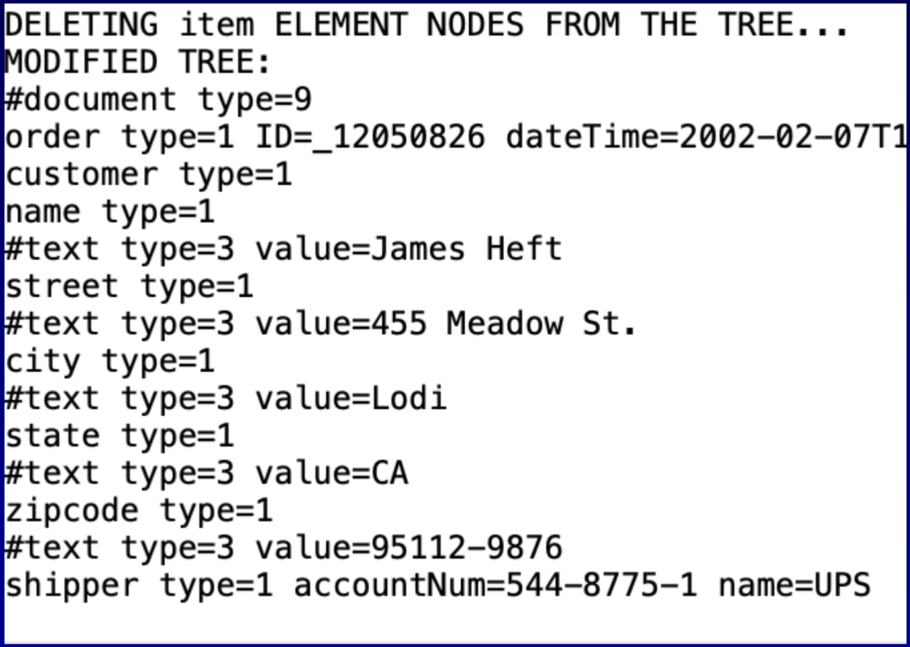
Results

Lab

- Original tree left below, pruned tree right below
- Where did the "item" elements go?



```
#text type=3 value=Lodi
state type=1
#text type=3 value=CA
zipcode type=1
#text type=3 value=95112-9876
shipper type=1 accountNum=544-8775
item type=1 ID=CD513
name type=1
#text type=3 value=My, I'm Large
artist type=1
#text type=3 value=Bobs
releaseDate type=1
#text type=3 value=1987-02-20
listPrice type=1
#text type=3 value=11.97
price type=1
#text type=3 value=11.97
item type=1 ID=CD518
name type=1
```



```
DELETING item ELEMENT NODES FROM THE TREE...
MODIFIED TREE:
#document type=9
order type=1 ID=_12050826 dateTime=2002-02-07T1
customer type=1
name type=1
#text type=3 value=James Heft
street type=1
#text type=3 value=455 Meadow St.
city type=1
#text type=3 value=Lodi
state type=1
#text type=3 value=CA
zipcode type=1
#text type=3 value=95112-9876
shipper type=1 accountNum=544-8775-1 name=UPS
```

Stripping Whitespace - Optional

Lab

Tasks to Perform

- In DOMUtilities, implement the following method, which removes insignificant whitespace nodes from a NodeList

```
public static void stripWhitespaceNodes(NodeList list)
{
  // decrementing i from length-1 to 0 prevents shifting
  for (int i = list.getLength() - 1; i >= 0; i--)
  {
    Node node = list.item(i);

    // if text node and trimmed length is 0 -> whitespace
    if (node.getNodeType() == Node.TEXT_NODE &&
        node.getNodeValue().trim().length() == 0)
    {
      // "step up" to parent node to remove current node
      node.getParentNode().removeChild(node);
    }
  }
}
```

Stripping Whitespace - Optional

Lab

Tasks to Perform

- Call `stripWhitespaceNodes()` from `walkTree()` before recursively processing the current node's children

```
public static void walkTree(Node node) {
  printNode(node);
  NodeList children = node.getChildNodes();

  // remove whitespace nodes from the ("live") node list
  stripWhitespaceNodes(children);

  // go through remaining children, calling recursively
  for (int i = 0; i < children.getLength(); i++) {
    Node child = children.item(i);
    walkTree(child);
  }
}
```

- Does it work?

STOP

Lab

Lab 7.4: Transforming a DOM Tree to XML

- **Purpose**: In this lab, we will transform the DOM tree back to XML and print it out
- **Objectives**: Learn to convert a DOM tree to XML
- **Builds on previous labs**: Lab 7.3
  - Continue working in your Lab05.2 project
- **Approximate Time**: 20-30 minutes

Transforming a DOM Tree to XML

Lab

Tasks to Perform

- In DOMUtilities, implement the following method, which transforms a DOM tree into XML

```
public static void writeXML(Document doc, String file) {
  try {
    // create factory and (the identity) transformer
    TransformerFactory factory =
      TransformerFactory.newInstance();
    Transformer xformer = factory.newTransformer();

    // do transform
    // pass in DOM tree as source, empty file as result
    xformer.transform(new DOMSource(doc),
                      new StreamResult(new File(file)));
  }
  catch (TransformerException e) {
    System.out.println(e);
  }
}
```

Testing

Lab

Tasks to Perform

- Invoke this method from the `main()` method of `DOMTest`, **after** the call to `DOMUtilities.deleteElements()`
  - Use a filename of order-result.xml for your output
- Run your program, then refresh the project to recognize the new file
  - Open and view the output file in Eclipse
  - Are the specified element(s) gone?

DOM Tree to SAX Events - Optional

Lab

Tasks to Perform

- In DOMUtilities, implement the following method, which transforms a DOM tree into a series of SAX parse events

```
public static void writeSAX(Document doc, String target) {
  try {
    // create factory and (the identity) transformer
    TransformerFactory factory =
      TransformerFactory.newInstance();
    Transformer xformer = factory.newTransformer();

    // do transform
    // pass in DOM tree as source, SAX handler as result
    xformer.transform(new DOMSource(doc),
      new SAXResult(new OrderHandler(target)));
  }
  catch (TransformerException e) {
    System.out.println(e);
  }
}
```

STOP

Lab

Lab 8.1: Trying Out StAX

Try a StAX Parser

Lab

- **Purpose** – Try out a StAX parser
- Your working directory for this lab will be ***workspace\Lab08.1***
  - This is a new working directory
  - You will **not need to write anything for this** – just try out the parser
- We've written a program, **`com.javatunes.stax.StAXTest`**, that parses XML documents using the Java implementation of the StAX event-based parser
  - StAXTest parses an XML document passed in on the command line
  - Each time it **pulls** an event from parsing the document, it prints out the details of the information at the current cursor position
  - It then prompts you to ask if you want to continue the parse
  - This allows you to see what kind of events a pull parser receives, and the information that is available, **and to control the program execution**
- You can look at the program code if you're interested in it

Run the StAX Parser

Lab

Tasks to Perform

- Create a new **Java Project** called Lab08.1
  - Remember to **uncheck** "Create module-info.java file"
  - Click **Finish**

- This project contains the *StAXTest.java* program, as well as an XML file to parse (*order.xm*l)
  - Open *StAXTest.java* and review it
  - Right click on *StAXTest.java*, and select **Run as | Java application**
  - You'll need to type in the console window (click in it to gain focus) to continue running the program – it prompts you for input to continue
  - Notice how the program "pulls" elements from the XML file as long as you keep typing "y" at the command line

STOP

Lab

Lab 9.1 - Working with Node Trees

- **Purpose**: In this lab, we will work with some XPath node trees and nodes
- **Objectives**: Become familiar with XPath node trees
- **Builds on previous labs**: None
- **Approximate Time**: 20-30 minutes

Node Trees, Names, String-values

Lab

- Part One:
  - Below is a JavaTunes credit card XML document
  - Draw an XPath node tree for this XML document
  - Compute the name and string-value of each node

```
<?xml version='1.0'?>
<!-- JavaTunes credit-card XML document -->
<credit-card type='VISA'>
  <customer-name>Marvin Gardiner</customer-name>
  <card-number>4987877419837781</card-number>
  <credit-limit currency='USD'>5000</credit-limit>
  <expiry>12/05</expiry>
</credit-card>
```

Draw Node Tree

Lab

More Names and String-values

Lab

- Part Two:
  - For the JavaTunes order document shown previously, compute the name and string-value of:
  - First `item` element
  - `shipper` element
  - Document element’s `ID` attribute
  - First text node descendant of `customer` element

STOP

Lab

Lab 9.2 - Location Paths

- **Purpose**: In this lab, we will work with a slightly more complex document, and use XPath location paths on it
- **Objectives**: Continue working with XPath node trees, and work with location paths
- **Builds on previous labs**: None
- **Approximate Time**: 20-30 minutes

Location Paths

Lab

- Part One - draw a node tree for this document - optional

```xml
<?xml version='1.0'?>
<purchase-request>
  <purchase>
    <amount>10.00</amount>
    <currency>USD</currency>
    <timedate>2003-01-18T14:21:00</timedate>
  </purchase>
  <merchant>
    <merchant-name>JavaTunes</merchant-name>
    <business-number>987676257625</business-number>
  </merchant>
  <credit-card>
    <type>Visa</type>
    <name-on-card>Bob Smith</name-on-card>
    <card-number>1987987399918277</card-number>
    <expdate>01/04</expdate>
  </credit-card>
</purchase-request>
```

Draw Node Tree

Lab

Location Paths

Lab

- Part Two - identify the resulting node-set for each of the following XPath expressions

```
/purchase-request/*

//*

/*/*/amount

/*/card-number

/*/*/type/..

//credit-card//text()
```

Location Paths

Lab

- Part Three - for the purchase request document, write two different location paths for each of the following -- use absolute location paths (see notes)
  - Parent of the **`type`** element
  - **`amount`** element
  - **All** children of the **`credit-card`** element (be careful on this one)
  - **Element** grandchildren of the **document element**
  - Text node descendants of the **`purchase`** element

STOP

Lab

Lab 9.3 - Predicates, Functions, Operators

- **Purpose**: In this lab, we will work with the JavaTunes order document, and use more complex XPath location paths
  - We will also set up the lab environment
- **Objectives**: Work with location paths that include predicates, functions, and operators
- **Builds on previous labs**: None
- **Approximate Time**: 20-30 minutes

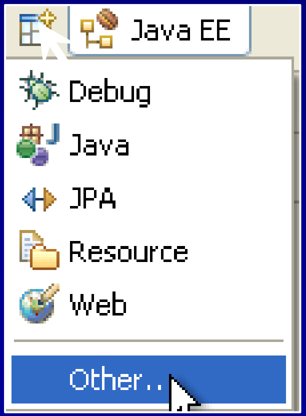
XML Perspective

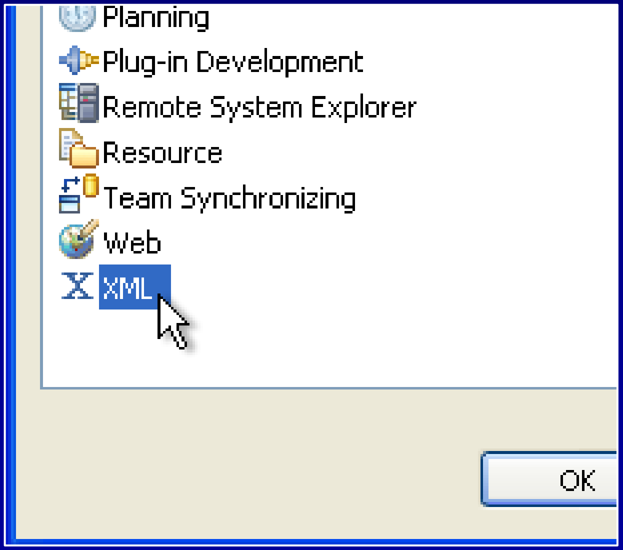
Lab

Tasks to Perform

- If you're in an XML perspective, then just remain in it
- If not in an **XML** perspective, **open an XML one** by clicking the Perspective icon at the top right of the Workbench, and select Other, then XML (as shown below left)

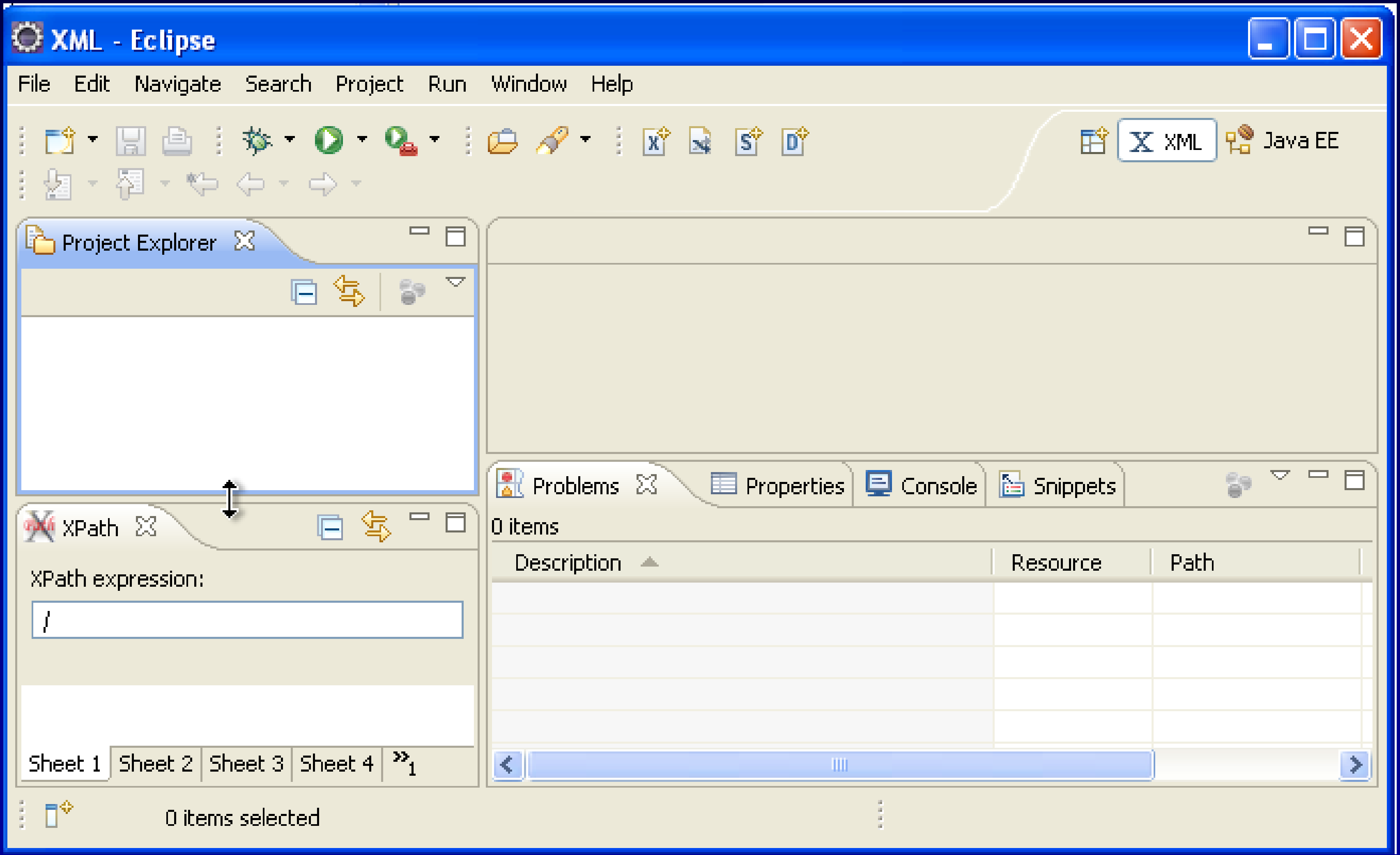
– Select the XML perspective from the chooser





The XML Perspective

Lab



- Note: We've closed the Outline view to unclutter the perspective

Create a Project for our Lab

Lab

Tasks to Perform

- Create a **Project**
  - To create a new Project, use the menu item: **File | New | Project | General| Project**
  - Call the project **Lab09.3**
  - Eclipse will then automatically set the project directory to *Lab09.3*
    - Where we have starter files for this lab
  - Click **Finish**

- Open the existing *order.xml* file that is in the project
  - You can validate it by right clicking on it and selecting validate

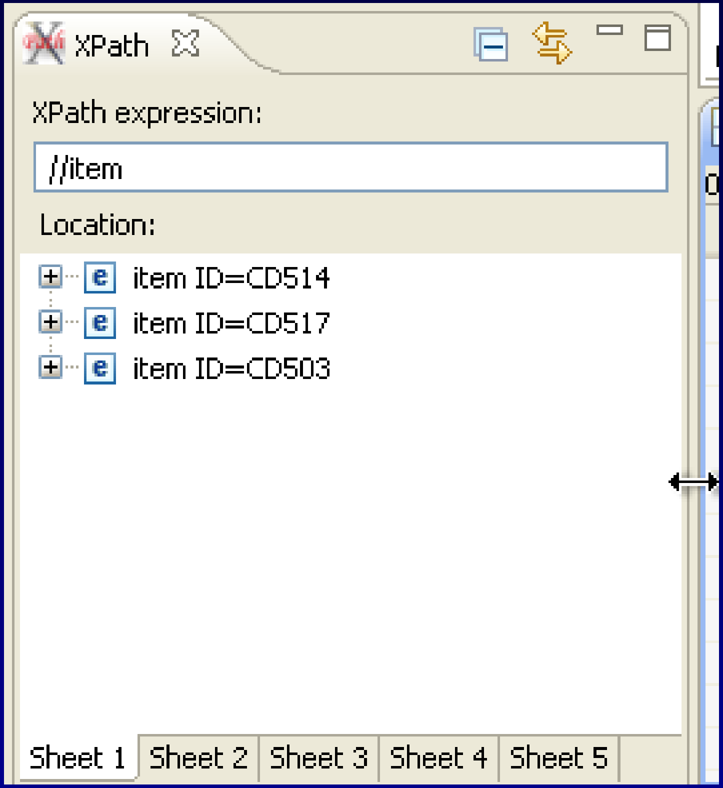
- You can **evaluate XPath expressions** in the XPath Expression view that normally appears in the lower left corner
  - See next slide for an example

XPath Expression View

Lab

Tasks to Perform

- In the XPath expression view, type the expression //item
  - This selects all the item nodes in the document
  - Note the result in the Location pane (shown at right)
- You can use this to work with and test the XPath expressions that you'll be figuring out in this lab
  - It has some limitations
  - It only shows the nodes selected – not their values
  - It won't work with functions like sum() or count()



XPath expression:

//item

Location:

item ID=CD514

item ID=CD517

item ID=CD503

Predicates, Functions, Operators

Lab

Tasks to Perform

- Using a JavaTunes order XML document (see next slides), write an XPath expression to select:
  1. The **value** (text content) of the state in which the customer lives
  2. The **value** (text content) of the second item's artist
  3. The item whose ID is CD503
  4. The items by an artist whose name begins with Peter
  5. The number of items in the order with a list price of at least $15
  6. The total cost of the order (items are sold at price, not list price)
  7. The average cost of the items in the order (items are sold at price, not list price) -- there is no `average()` function in XPath
  - Use absolute location paths in the expressions that are paths
- **NOTE** that not all of these expressions are used in predicates
  - e.g. to select the number of customers in the order you just use:

```
count(/order/customer)
```

JavaTunes Order XML Document

Lab

```
<?xml version='1.0'?>

<!-- JavaTunes order XML document -->

<order ID='_01170302' dateTime='2002-03-20T05:02:00'
 xmlns:xsi='...' xsi:noNamespaceSchemaLocation='...'>
  <customer>
    <name>Susan Phillips</name>
    <street>763 Rodeo Circle</street>
    <city>San Francisco</city>
    <state>CA</state>
    <zipcode>94109</zipcode>
    <shipper name='UPS' accountNum='343-9080-1'/>
  </customer>
  ...
```

JavaTunes Order XML Document

Lab

```
  ...
  <item ID='CD514'>
    <name>So</name>
    <artist>Peter Gabriel</artist>
    <releaseDate>1986-10-03</releaseDate>
    <listPrice>17.97</listPrice>
    <price>13.99</price>
  </item>
  <item ID='CD517'>
    <name>1984</name>
    <artist>Van Halen</artist>
    <releaseDate>1984-08-19</releaseDate>
    <listPrice>11.97</listPrice>
    <price>11.97</price>
  </item>
  <item ID='CD503'>
    <name>Trouble is...</name>
    <artist>Kenny Wayne Shepherd Band</artist>
    <releaseDate>1997-08-08</releaseDate>
    <listPrice>17.97</listPrice>
    <price>14.99</price>
  </item>
</order>
```

STOP

Lab

Lab 10.1 – Default Template Rules

- **Purpose**: In this lab, you will create an empty stylesheet which uses the default template rules
  - You'll run this stylesheet on your *order.xml* document
- **Objectives**: Understand the default template rules
  - Run a transformation in a stylesheet on an XML document
- **Builds on previous labs**: Lab 9.3
  - Continue working in your **Lab09.3** project
- **Approximate Time**: 20-30 minutes

Default Template Rules

Lab

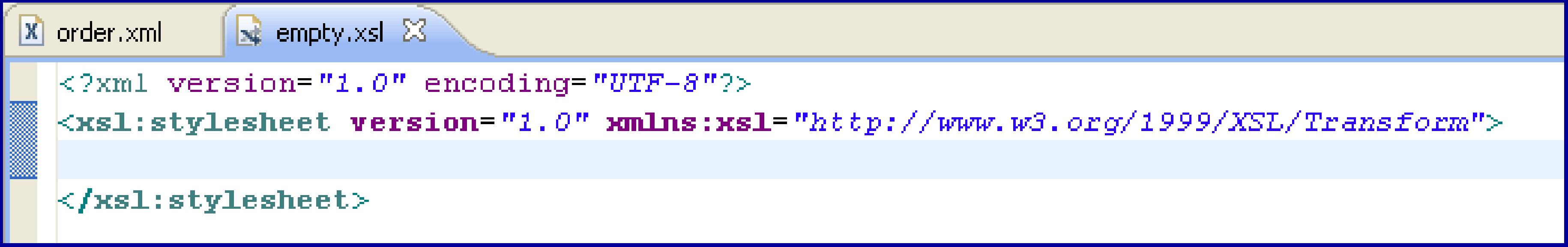
- In this lab, you will see the default template rules in action
  - The processor will visit all element and text nodes
  - The text nodes’ string-values will be output
  - The processor will also visit all PI and comment nodes
  - Their string-values will **not** be output -- why not?
  - The processor will **not** visit attribute nodes -- why not?
- You will write the “empty” stylesheet in a file called ***empty.xsl***

Create the StyleSheet

Lab

Tasks to Perform

- Right click your project (In Package Explorer view) and select: ***New* | *XSL***
  - Name the file ***empty.xsl***, Click **Next**
  - In the next dialog, make sure the **Use template** checkbox is selected, select **Basic stylesheet – XSLT 1.0**, and click **Finish**
  - *empty.xsl* should open for editing
    - Follow the same procedure in future labs to create any stylesheets
  - In empty.xsl, **delete the xsl:template element** that Eclipse creates in the stylesheet, so it is truly the empty stylesheet, as shown at bottom



```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

</xsl:stylesheet>
```

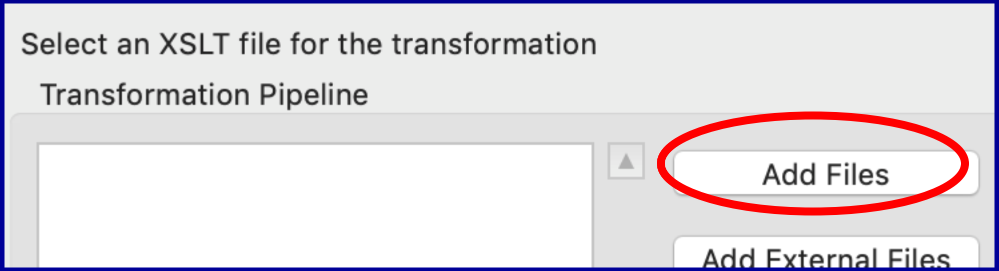
Run the Transformation

Lab

- Eclipse has the built in capability to use an XSLT transform engine
  – It's very easy to invoke a transform on an XML file
  - The output is another file that is created in your project

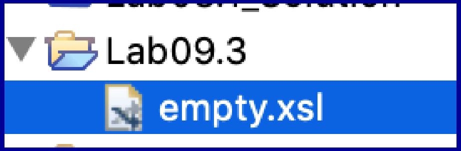
Tasks to Perform

- Run a JavaTunes order through this stylesheet as follows
  - Right click on ***order.xml*** and select **Run As | XSL Transformation**
  - In the dialog that comes up, click **Add Files**, browse to *empty.xsl* and click **OK**
  - You should see a file named **order.out.xml** in your project
  - It should also be open in an editor



Select an XSLT file for the transformation

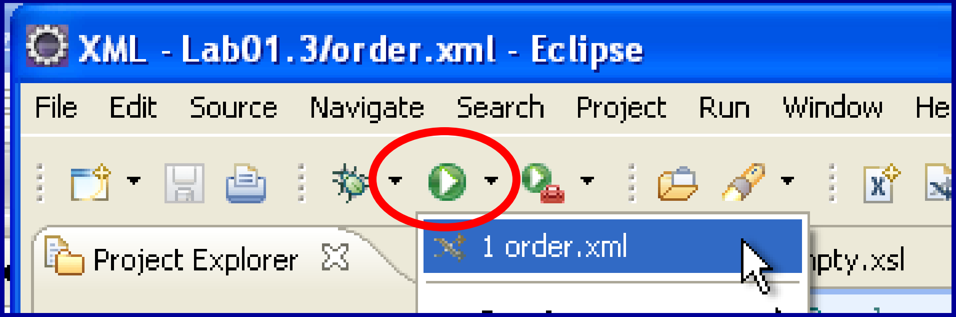
Transformation Pipeline



What You Will See

Lab

- You should see all the text nodes, along with some whitespace
  - You will also see an XML declaration
  - Why are the comments missing?
  - Why is the PI missing?
  - Why are the attributes missing?
- To run the transformation again, you can just click the run icon, and select ***order.xml***


XML - Lab01.3/order.xml - Eclipse
File Edit Source Navigate Search Project Run Window He
Project Explorer
1 order.xml
pty.xsl

Why is This the Output?

Lab

- Can you understand why the output is what it is? Pretend you are the XSLT processor:

1. Start at / and look for a matching template -- remember that your “empty” stylesheet has the three default template rules!

2. Follow the instructions in that template -- when processing child nodes, the processor takes each child node in turn and looks for a matching template ... if it finds one, it follows the instructions in it

– This process continues recursively

STOP

Lab

Lab 10.2 – Writing Templates

- **Purpose**: In this lab, you will override some of the default template rules
  - We wish to see attributes nodes and PI and comment nodes
  - You'll run this stylesheet on your *order.xml* document
- **Objectives**: Gain more understanding of XSLT processing
- **Builds on previous labs**: Lab 10.1
  - Continue working in your **Lab09.3** project
- **Approximate Time**: 20-30 minutes

Writing Templates

Lab

Tasks to Perform

- Create a stylesheet in a file called ***override.xsl***
  - Right click on the Lab09.3 project and select **New | XSL**
  - In your *XML_XSLT* project, as you did with empty.xsl

- Add attribute nodes to the transformation output
  - You need to explicitly direct the processor to visit them, as shown below

```
<xsl:template match='/ | *'>

  <!-- process child nodes and attribute nodes -->
  <xsl:apply-templates select='node() | @*'/>

</xsl:template>
```

Overriding the Default Templates

Lab

- To see PI and comment nodes, you need a template that outputs them
  - The default template rule for PI and comment nodes suppresses them from the output

Tasks to Perform

- Override this default template rule to output the string-values of PI and comment nodes

```
<xsl:template match='processing-instruction()|comment()'>

  <!-- copy string-value of node to output -->
  <xsl:value-of select='.'/>

</xsl:template>
```

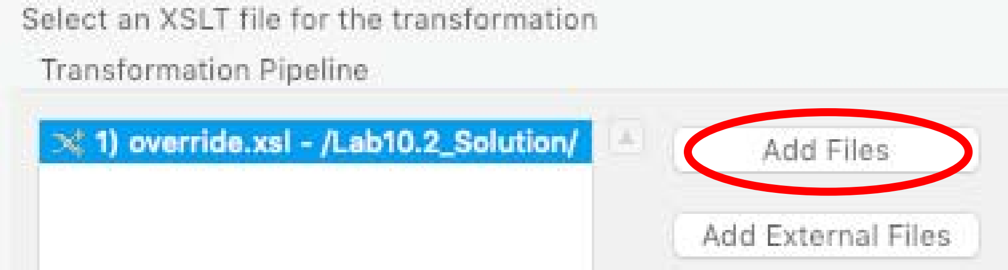
Doing the Transforms

Lab

- In the lab directory, we supply a number of XML documents
  - In this lab, you'll work with **order.xml**, ***orderNoSchema.xml*** and ***orderDTD.xml***

Tasks to Perform

- Transform each of the above documents with ***override.xsl***
  - For each file to transform, you'll need to right click on the XML file, and select **Run As | XSL Transformation**
  - In the dialog box that comes up, select **Add Files**, browse to *override.xsl*, and click **OK**



Select an XSLT file for the transformation

Transformation Pipeline

1) override.xsl - /Lab10.2_Solution/

What You Will See

Lab

- Do you now see the values of the attributes?
  Do you now see the values of the comments and the PI?

- Compare *order.out.xml*, *orderNoSchema.out*,.xml and *orderDTD.out.xml*
  - Do you see any differences? What are they?

- Does the processor use the XML Schema? The DTD?
  - Does an `item`'s `type` attribute default to CD?
  - *orderDTD.xml* contains an entity reference, `discount-price` -- does the processor perform entity replacement for it?

  - See the notes below for details

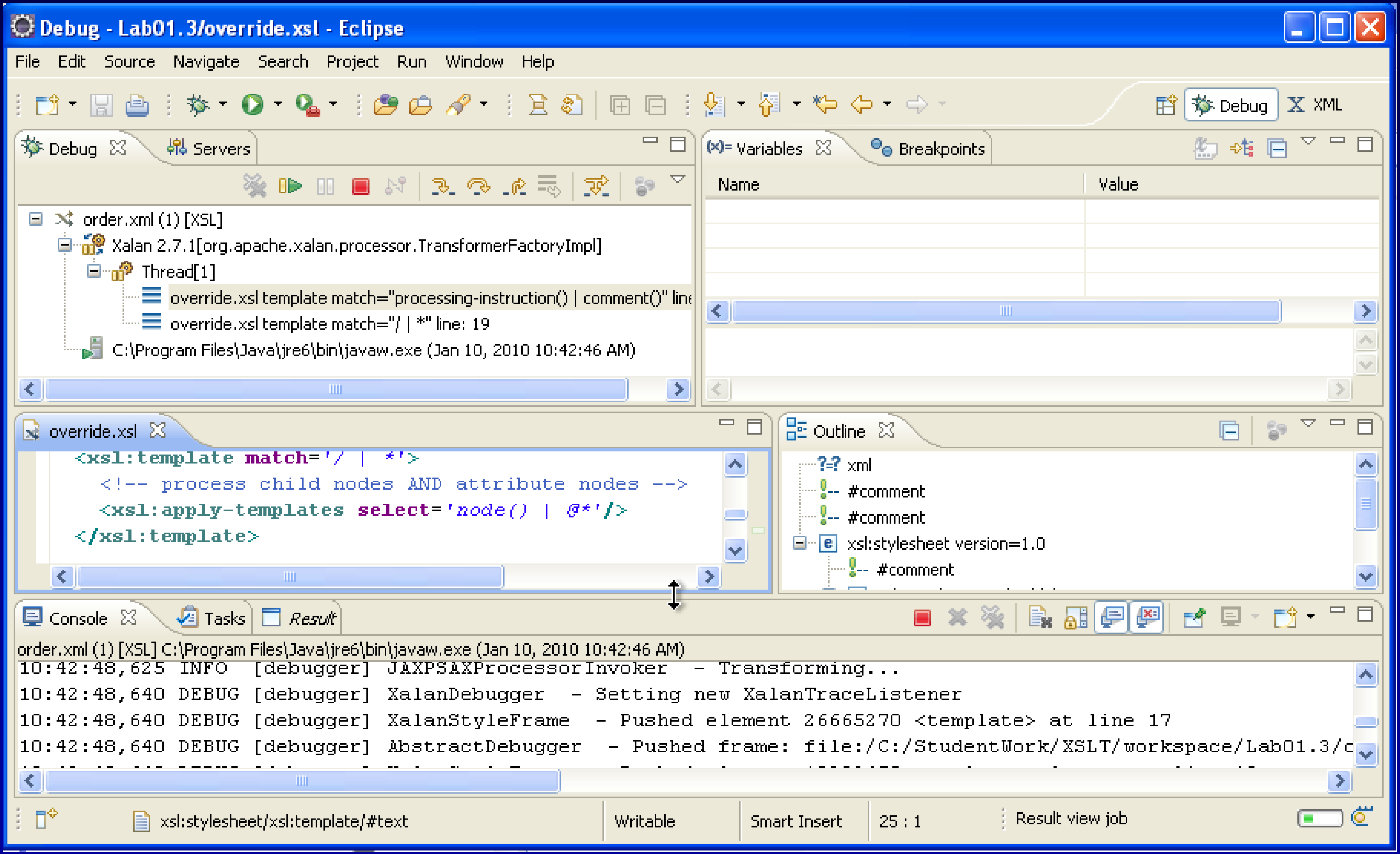
[Optional] The XSLT Debugger

Lab

- One of the very nice things in Eclipse is the XSLT debugger
  - You can do graphical debugging of an XSLT transform, just as you would with something like a Java program
- To use the XSLT debugger
  - Make sure *override.xsl* is open in an editor window
  - Right click on ***order.xml***, and select **Debug As | XSL Transformation**
  - **Add Files**, Select *override.xml* as the XSLT file and click **OK**
- Open a debug perspective (Using Perspective icon in the upper right of the Workbench)
- This will put you in the **Debug perspective**
  - See next slide

[Optional] Debug Perspective with XSL

Lab



Debug - Lab01.3/override.xsl - Eclipse

```
order.xml (1) [XSL]
  Xalan 2.7.1[org.apache.xalan.processor.TransformerFactoryImpl]
    Thread[1]
      override.xsl template match="processing-instruction() | comment()" line
      override.xsl template match="/ | *" line: 19
  C:\Program Files\Java\jre6\bin\javaw.exe (Jan 10, 2010 10:42:46 AM)
```

```
<xsl:template match='/ | *'>
  <!-- process child nodes AND attribute nodes -->
  <xsl:apply-templates select='node() | @*'/>
</xsl:template>
```

```
?-? xml
!-- #comment
!-- #comment
e xsl:stylesheet version=1.0
  !-- #comment
```

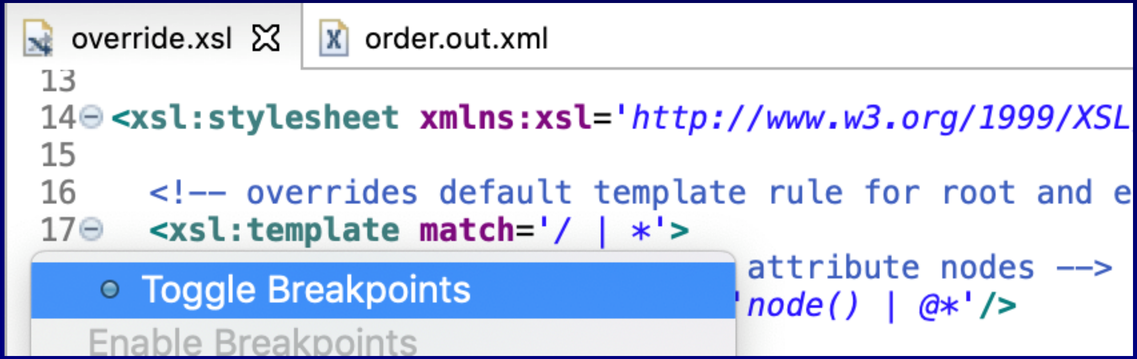
```
order.xml (1) [XSL] C:\Program Files\Java\jre6\bin\javaw.exe (Jan 10, 2010 10:42:46 AM)
10:42:48,625 INFO  [debugger] JAXPSAXProcessorInvoker  - Transforming...
10:42:48,640 DEBUG [debugger] XalanDebugger  - Setting new XalanTraceListener
10:42:48,640 DEBUG [debugger] XalanStyleFrame  - Pushed element 26665270 <template> at line 17
10:42:48,640 DEBUG [debugger] AbstractDebugger  - Pushed frame: file:/C:/StudentWork/XSLT/workspace/Lab01.3/c
```

xsl:stylesheet/xsl:template/#text | Writable | Smart Insert | 25 : 1 | Result view job

[Optional] Adding Breakpoints

Lab

- Right click on a line in the stylesheet to toggle a breakpoint on
  - Do this in a couple of places - once in each template rule you're written

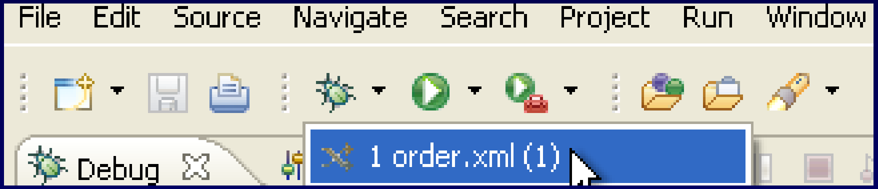


```
<xsl:stylesheet xmlns:xsl='http://www.w3.org/1999/XSL

   <!-- overrides default template rule for root and e
   <xsl:template match='/ | *'>
                                     attribute nodes -->
                                    'node() | @*'/>
```

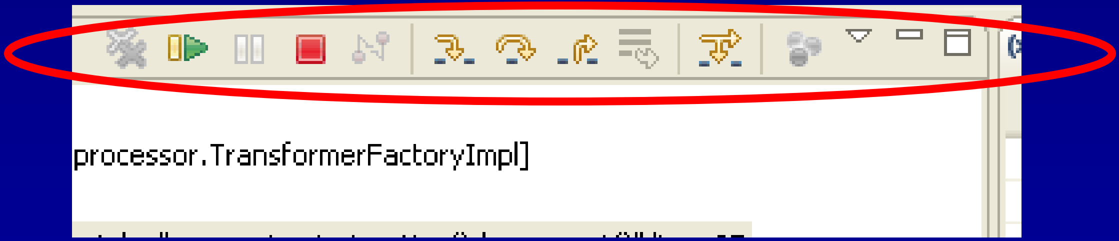
[Optional] Running and Stepping

Lab

- You can click the debug icon to run again (until a breakpoint)



- You can step forward, backward, resume, and so on with the icons in the debug view
  - These are standard Eclipse debugger buttons

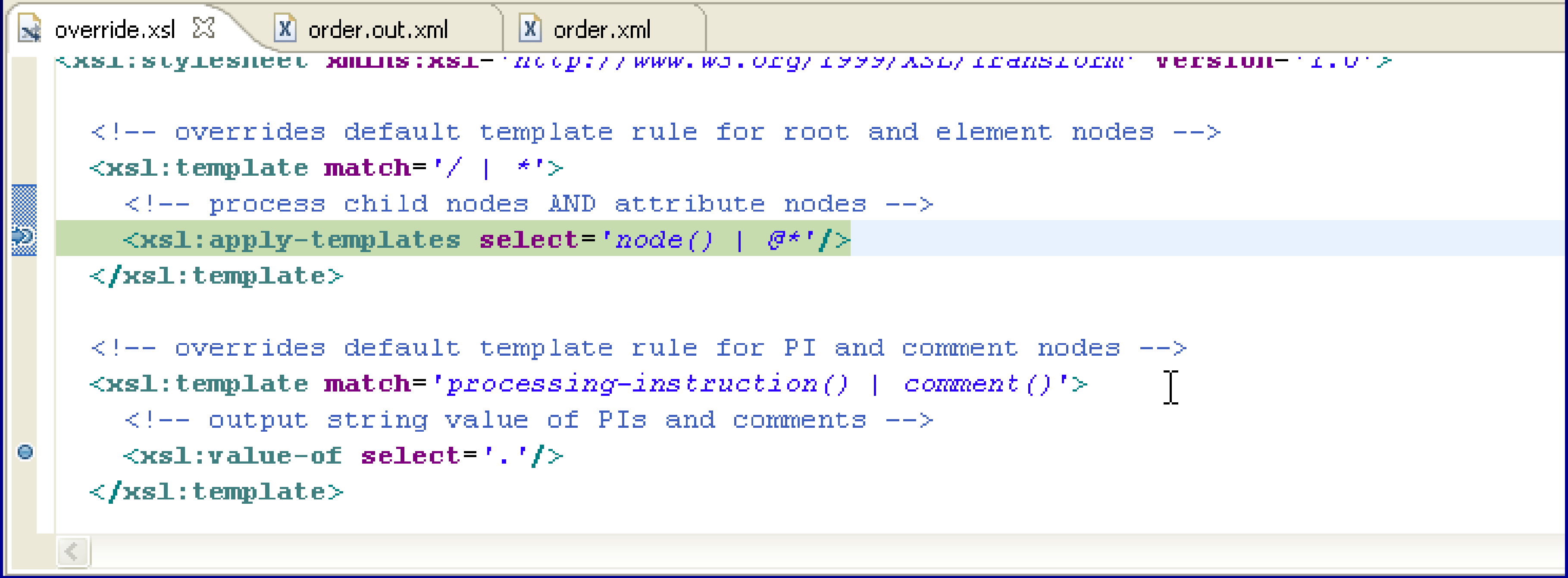


processor.TransformerFactoryImpl]

[Optional] Seeing What's Happening

Lab

- You can easily view the current location in the stylesheet

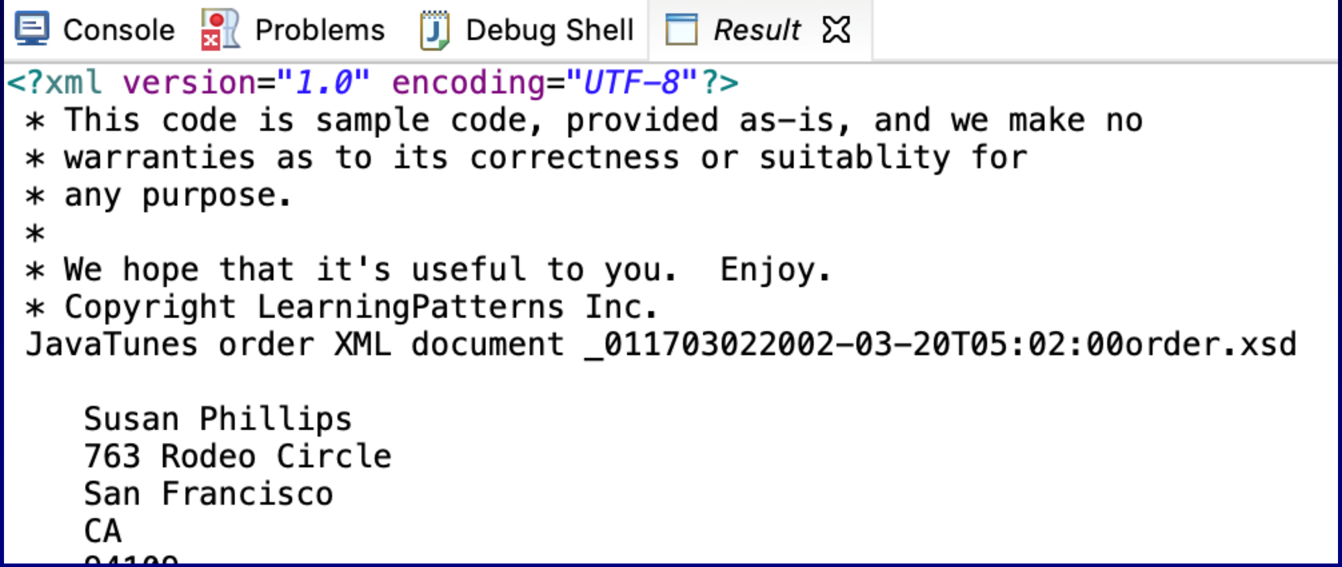


```
<xsl:stylesheet xmlns:xsl='http://www.w3.org/1999/XSL/Transform' version='1.0'>

  <!-- overrides default template rule for root and element nodes -->
  <xsl:template match='/ | *'>
    <!-- process child nodes AND attribute nodes -->
    <xsl:apply-templates select='node() | @*'/>
  </xsl:template>

  <!-- overrides default template rule for PI and comment nodes -->
  <xsl:template match='processing-instruction() | comment()'>
    <!-- output string value of PIs and comments -->
    <xsl:value-of select='.'/>
  </xsl:template>
```

[Optional] Seeing What's Happening

Lab

- You can view the current transformed output in the **Result** view



```
<?xml version="1.0" encoding="UTF-8"?>
 * This code is sample code, provided as-is, and we make no
 * warranties as to its correctness or suitablity for
 * any purpose.
 *
 * We hope that it's useful to you.  Enjoy.
 * Copyright LearningPatterns Inc.
 JavaTunes order XML document _011703022002-03-20T05:02:00order.xsd

    Susan Phillips
    763 Rodeo Circle
    San Francisco
    CA
    94109
```

- It's a nice tool to have
  - But - like XSLT itself, there's still quite a bit of complexity
  - A debugger can help, but understanding what's going on during a transformation will still take some time
- Spend a little time with the debugger to get a feel for it, and for how a transform occurs

STOP

Lab

Lab 10.3 – Pruning the Source Tree

- **Purpose**: In this lab, you will generate customized XML output from a JavaTunes order
  - You will write separate stylesheets that select for the customers and for the items
- **Objectives**: Learn how to select parts of an XML document
- **Builds on previous labs**: Lab 10.2
  - Continue working in your **Lab09.3** project
- **Approximate Time**: 20-30 minutes

Pruning the Source Tree

Lab

Tasks to Perform

- Part A - create a stylesheet in a file called ***customer.xsl***
  - Create this in your XML_XSLT project
  - See earlier labs for details if needed
- In *customer.xsl*, write a template that copies **only** the **`customer`** subtree to the result document
  - This will be the only template in your stylesheet
- **Transform *order.xml*** with this stylesheet
  - We've already seen how to run a transform
  - Look back at previous labs if you need to refresh your memory

Pruning the Source Tree

Lab

Tasks to Perform

- The resulting XML file should be a well formed XML document that contains only the customer subtree
  - Review the result document to see that it is correct (Design view is useful to see the subtree)
- Now write another template in *customer.xml* that copies **only** the first comment in the order to the result document
  - This is the one with copyright information
  - Copy the comment **node, not** its string value
  - View the result document to check your work
- Optional - perform both tasks in one template
  - You may want to do this in another stylesheet to keep things neat - e.g.*customer-optional.xsl*

Custom Copying

Lab

Tasks to Perform

- Part B - create a stylesheet in a file called ***items.xsl***
  - See previous lab instructions if needed
- In *items.xsl*, write a template that copies only the data listed below from the **`item`** elements to the result document:
  - The **`ID`** attribute
  - The **`name`**, **`artist`**, and **`releaseDate`** child elements
  - Each `item` element in the result document should look like this:

```
<item ID="CD502">
  <name>Dream of the Blue Turtles</name>
  <artist>Sting</artist>
  <releaseDate>1985-02-05</releaseDate>
</item>
```

Custom Copying

Lab

Tasks to Perform

- To make the result document well-formed, we need to wrap the set of resulting `item` elements in a document element
  - Write another template that outputs a ***literal result element*** -- this will be the document element
  - The copied items need to appear between its start-tag and end-tag
  - You will need to do this in a template that gets matched **only once**

```
<xsl:template match='...'>
  <items>
    <xsl:apply-templates .../>
  </items>
</xsl:template>
```

- Transform the order
  - Look at the results to check your work

STOP

Lab

Lab 10.4 – Creating an Order Summary

- **Purpose**: In this lab, you will create an order summary document from a JavaTunes order
  - You will use XSLT to summarize different portions of the document
- **Objectives**: Learn more complex uses of XSLT
- **Builds on previous labs**: Lab 2.3
  - Continue working in your **Lab09.3** project
- **Approximate Time**: 30-40 minutes

Creating an Order Summary

Lab

- We wish to summarize the following aspects of an order:
  - **Total cost** of the order
  - **Sales region** the customer lives in
  - **Shipper** used for that sales region
  - **Artists** of the items that were purchased

- This well help JavaTunes management understand our customers' buying habits
  - Where are our customers?
  - What shippers should we negotiate better rates with?
  - What artists are our customers most interested in?

Example of an Order Summary

Lab

```
<?xml version='1.0' encoding='UTF-8'?>

<order-summary total-cost='29.98'
               sales-region='WY'
               shipper='USMail'
               ID='_47632423'
               xmlns:xsi='...'
               xsi:noNamespaceSchemaLocation='...'>
  <artist>Jonny Lang</artist>
  <artist>Tori Amos</artist>
</order-summary>
```

- Notice the extensive use of attributes
  - Summary values are usually scalars, so attributes work quite well here
- And we provide a schema location attribute, for validation
  - We've supplied ***order-summary.xsd*** in your project that you can use to validate the result document

Getting Started

Lab

Tasks to Perform

- Create a stylesheet in a file called ***order-summary.xsl***
- Use **`xsl:output`** to specify:
  - That we want XML output format
  - That output elements should be indented
- Write stylesheet elements to output an `order-summary` element
  - Since this is our document element, we need to output this in a template that will be matched **only once**
  - We need to use attribute value templates to fill in the attribute values
  - `ID` is the original order ID -- the rest are computed values
- See the next slide

Writing the Templates

Lab

- The template will output something like that below
- Some of the values, like the one for the total-cost attribute, are computed values
  - See the notes for guidelines on how to do this

```
<xsl:template match='...'>
  <order-summary total-cost='{...}'
                 sales-region='{...}'
                 shipper='{...}'
                 ID='{...}'
                 xmlns:xsi='...'
                 xsi:noNamespaceSchemaLocation='...'>

    <!-- the artist elements get inserted here -->

  </order-summary>
<xsl:template>
```

Writing the Templates

Lab

Tasks to Perform

- Next add stylesheet elements to insert the `artist` elements between `<order-summary>` and `</order-summary>`
  - You can use `<xsl:apply-templates ...>` and write another template that matches `artist` elements
  - This second template should copy the `artist` elements into the result

    OR
  - You can perform the copy in the first template
- You need to decide between `xsl:copy` and `xsl:copy-of`
  - Remember that `xsl:copy` performs a shallow copy and only applies to the context node

Doing the Transform

Lab

Tasks to Perform

- **Transform *order.xml*** using this new stylesheet
  - Look at the results to check your work
- Validate the result document
  - Right click on it and select **Validate**

Optional

- Look at *order-summary.xml.out* (it should be open in Eclipse)
  - Look at the source view
  - Were the elements indented?
  - Set `indent='no'` in `xsl:output` and do the transform again -- what do you notice?
- Insert a comment into the order summary

```
<!-- order-summary XML document -->
```

STOP

Lab

Lab 10.5 – Transforming to HTML

- **Purpose**: In this lab, you will create an HTML page from a JavaTunes order
  - The principles are much the same, but you will generate HTML elements, rather than XML
- **Objectives**: Transform XML into HTML
- **Builds on previous labs**: Lab 9.3
  - Continue working in your **Lab09.3** project
- **Approximate Time**: 50-60 minutes

Transforming an Order to HTML

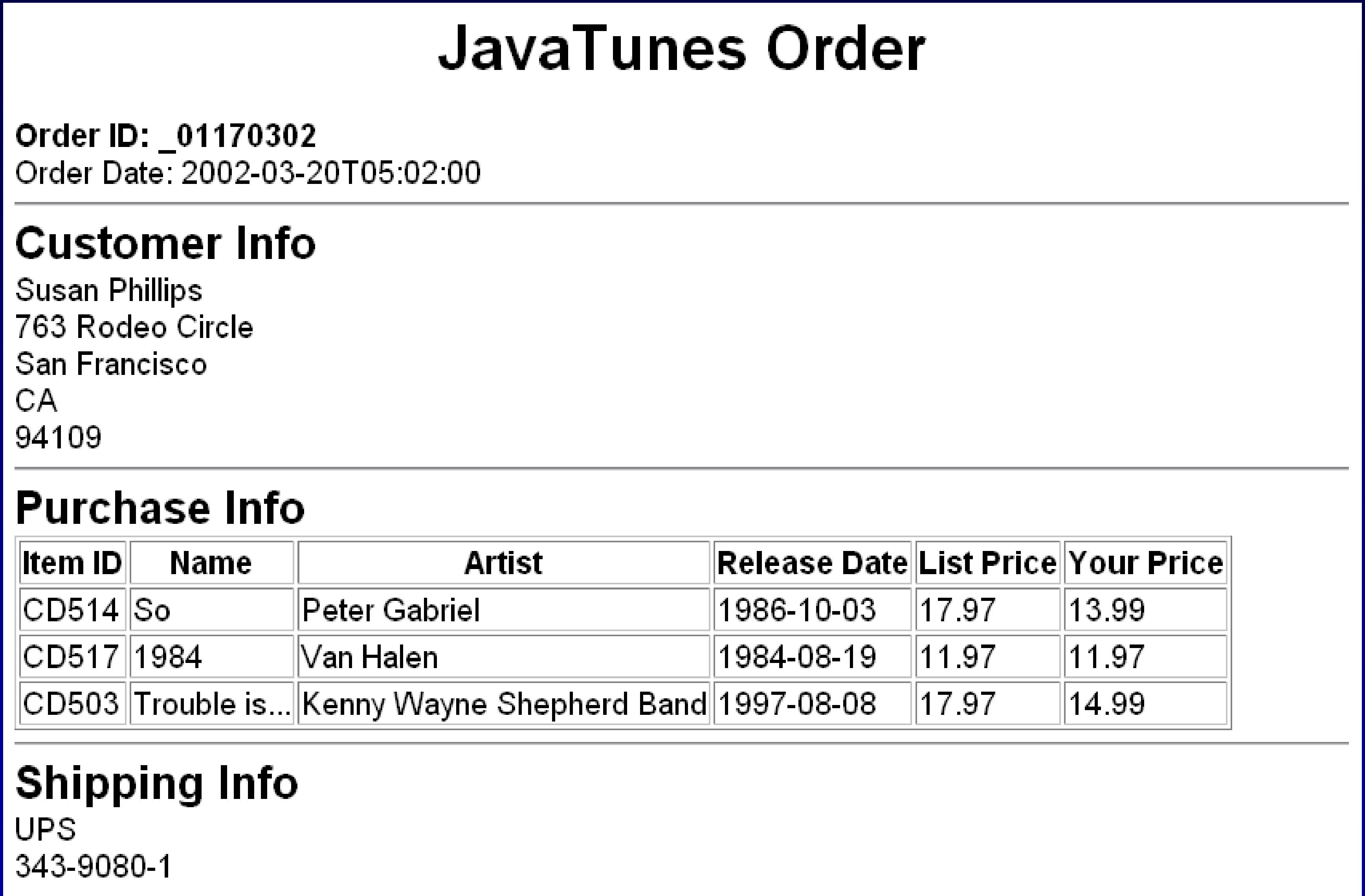
Lab

Tasks to Perform

- Create a stylesheet in a file called ***order-html.xsl***
  - Use the screen shot on the next slide as your output goal
  - We want our HTML output to look similar to this
- Use **`xsl:output`** to specify:
  - That we want HTML output format
  - That output elements should be indented
- Recommendation: **do this lab iteratively**
  - Do some of the work, do the transform, view the output in a browser
  - This helps you see the cause-and-effect of your templates
- Recommendation: **type carefully**
  - Typos, well-formedness errors, etc., in your stylesheet will cause the processor to fail and it may not provide meaningful information in the error text

The Desired Output

Lab



JavaTunes Order

Order ID: _01170302
Order Date: 2002-03-20T05:02:00

Customer Info

Susan Phillips
763 Rodeo Circle
San Francisco
CA
94109

Purchase Info

Item ID	Name	Artist	Release Date	List Price	Your Price
CD514	So	Peter Gabriel	1986-10-03	17.97	13.99
CD517	1984	Van Halen	1984-08-19	11.97	11.97
CD503	Trouble is...	Kenny Wayne Shepherd Band	1997-08-08	17.97	14.99

Shipping Info

UPS
343-9080-1

Checklist of Templates (1 of 3)

Lab

Tasks to Perform

- Write the following templates (listed by match pattern)
  - Their major tasks are outlined in a second level bullet
    - Where to look for completed code in the manual is given for each task in a third level bullet
- Remember to do these one at a time, and test each one !!
- `/`
  - `html`, `head`, `body`, apply templates to `order`
    - This template is given in its entirety on the slide titled *Fundamental Approach - Example*

Checklist of Templates (2 of 3)

Lab

- **`order`**
  - `h1`, Order ID, Order Date, apply templates to `customer`
    - This portion of the template is given on the slide titled *Fundamental Approach - Example*
  - Purchase Info heading, HTML table framework
  - The table framework is given on the slide titled *HTML Tables - Example - Table Framework*
    - This will apply templates to `item`
  - apply templates to `shipper`
- **`customer`**
  - Customer Info heading, extract customer data

Checklist of Templates (3 of 3)

Lab

- `item`
  - `tr`, apply templates to `ID` attribute and child elements
    - This template is given in its entirety on the slide titled *HTML Tables - Example - Table Rows*
- `item/@id | item/*`
  - `td`, extract item data
    - This template is given in its entirety on the slide titled *HTML Tables - Example - Table Data*
- `shipper`
  - Shipping Info heading, extract shipper data

Some HTML Tidbits

Lab

- The line break in HTML is `<br/>`
  - A line break with an empty line underneath is `<p/>`

- The horizontal line is `<hr/>`

- To create the Info headings
  - You can use heading elements such as `h1`, `h2`, `h3`, etc. (`h1` is biggest)

    ```
    <h2 align='left'>Purchase Info</h2>
    ```
  - It will automatically be bold and padded with a blank line (which you may not want)

  - Or you can just use a larger font and make it bold yourself

    ```
    <font size='+2'><b>Purchase Info</b></font>
    ```

Do the Transform

Lab

Tasks to Perform

- Transform *order.xml* with your new template
  - Look at the result when it opens in the editor
- Review the output document (order.out.html)
  - Open the html doc in a Web browser (you can right click and select **Open with | Web Browser**)
  - Check that it looks correct in the browser

STOP

Lab

[Optional] Lab 10.6 – Conditional Processing

- **Purpose**: In this lab, you will enhance our HTML output from the previous lab
  - The enhancements will improve the quality of the generated HTML
- **Objectives**: Use conditional processing
- **Builds on previous labs**: Lab 10.5
  - Continue working in your **Lab09.3** project
- **Approximate Time**: 30-40 minutes

Adding Conditional Processing

Lab

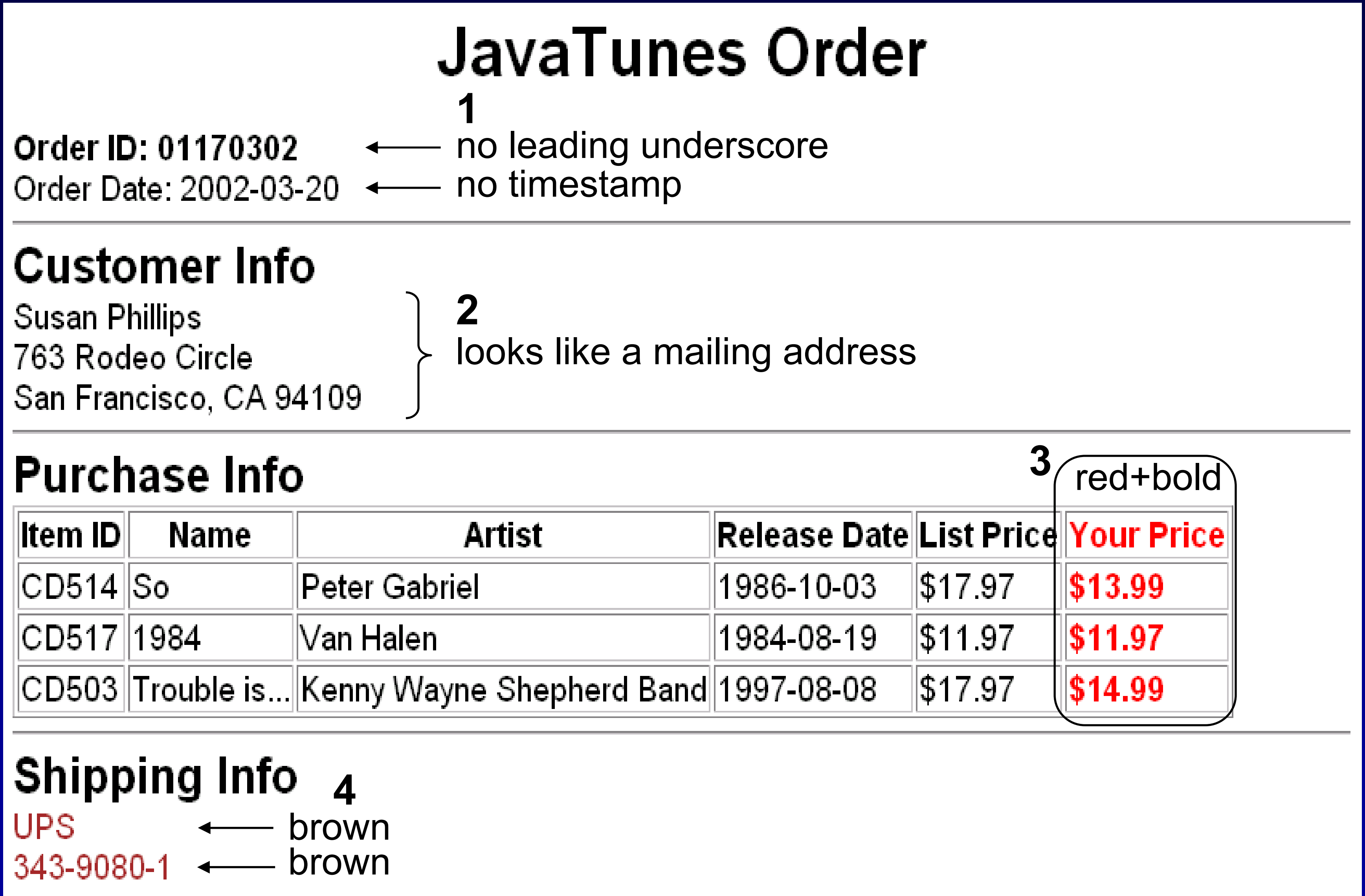
- In this lab, we will enhance our HTML output from the previous lab

- These are the enhancements:
  1. Strip the leading underscore from the order ID
     Strip the timestamp from the order date
  2. Make the customer address look like a mailing address
  3. List price and price should have a $ in front of the value
     Make the price values red+bold (as well as the heading)
  4. If the shipper is FedEx, make the name blue and the account number orange
     If the shipper is UPS, make the name and account number brown

- Use the screen shots on the next two slides as your output goal

The Desired Output - UPS

Lab



JavaTunes Order

Order ID: 01170302 ← 1 no leading underscore

Order Date: 2002-03-20 ← no timestamp

Customer Info

Susan Phillips
763 Rodeo Circle
San Francisco, CA 94109

2 looks like a mailing address

Purchase Info

3 red+bold

Item ID	Name	Artist	Release Date	List Price	Your Price
CD514	So	Peter Gabriel	1986-10-03	$17.97	**$13.99**
CD517	1984	Van Halen	1984-08-19	$11.97	**$11.97**
CD503	Trouble is...	Kenny Wayne Shepherd Band	1997-08-08	$17.97	**$14.99**

Shipping Info

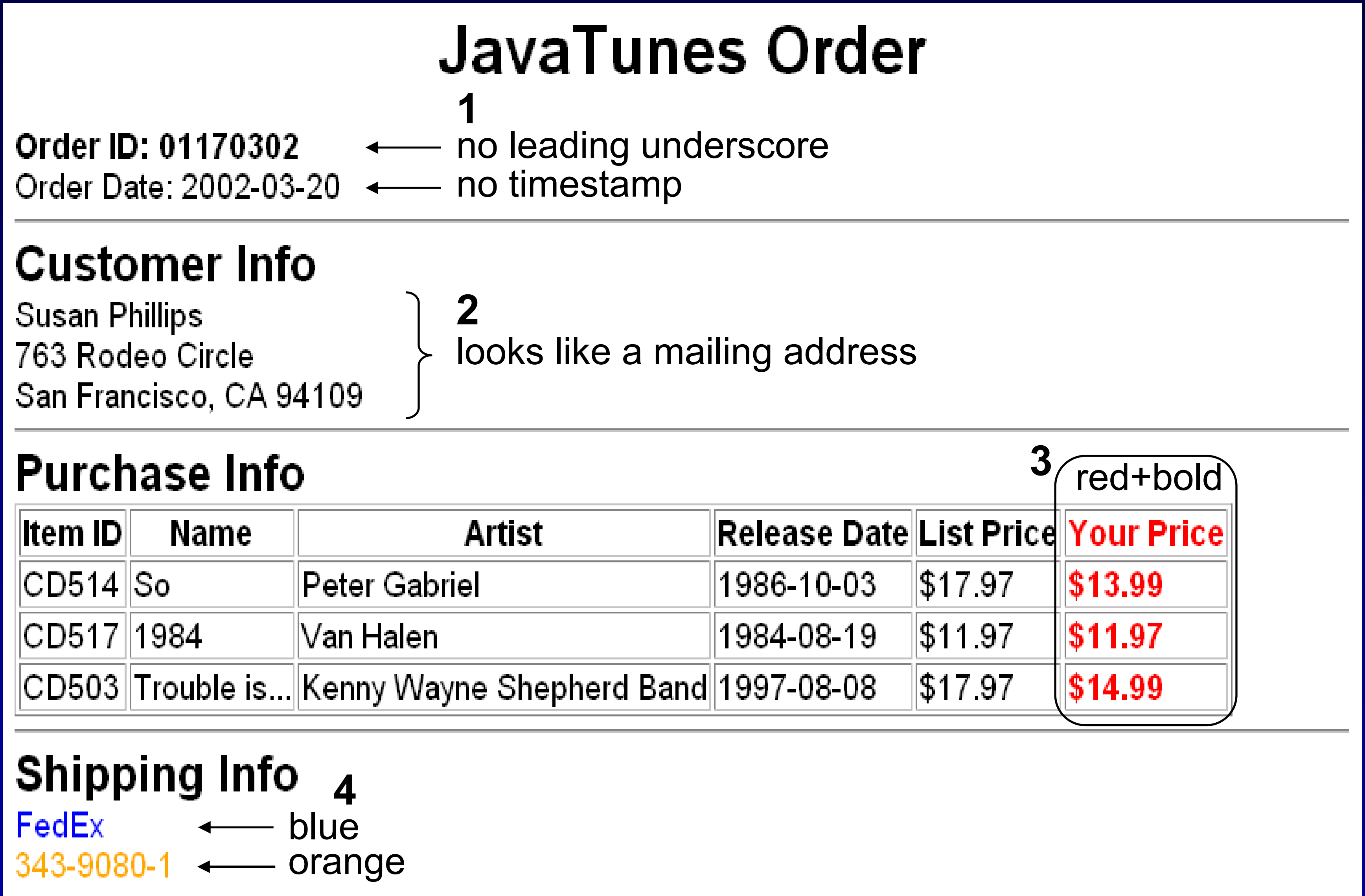
4

UPS ← brown

343-9080-1 ← brown

The Desired Output - FedEx

Lab



JavaTunes Order

Order ID: 01170302
Order Date: 2002-03-20

1
no leading underscore
no timestamp

Customer Info

Susan Phillips
763 Rodeo Circle
San Francisco, CA 94109

2
looks like a mailing address

Purchase Info

3
red+bold

Item ID	Name	Artist	Release Date	List Price	Your Price
CD514	So	Peter Gabriel	1986-10-03	$17.97	**$13.99**
CD517	1984	Van Halen	1984-08-19	$11.97	**$11.97**
CD503	Trouble is...	Kenny Wayne Shepherd Band	1997-08-08	$17.97	**$14.99**

Shipping Info

FedEx
343-9080-1

4
blue
orange

Enhancing the Templates

Lab

Tasks to Perform

- Make a backup copy of *order-html.xsl* from the previous lab
  - We want to have a clean working solution in case you get into trouble on this one
  - You can right click on the file and copy then paste it in the project

1. Strip the leading underscore from the order ID
   - In the `xsl:value-of` element that extracts this value, use the XPath **`substring-after()`** function

   ```
   <xsl:value-of select='substring-after(@ID, "_")'/>
   ```

   - This returns the string following the first occurrence of _

- Strip the timestamp from the order date
  - Use the XPath **`substring-before()`** function

  ```
  <xsl:value-of
   select='substring-before(@dateTime, "T")'/>
  ```

Enhancing the Templates

Lab

Tasks to Perform

2. Make the customer address look like a mailing address
   - Just remove the applicable `<br/>` elements and add the comma between the city and state
   - To get a space between the state and the zipcode, you can use the XPath **`string()`** function
     - The XSLT processor does not recognize `&nbsp;` (see notes)
   - Insert the following between your `xsl:value-of` elements for `state` and `zipcode`:

```
<xsl:value-of select='string(" ")'/>
```

Enhancing the Templates

Lab

Tasks to Perform

3. Add a $ in front of the values for list price and price
   Make the price values red+bold (as well as the heading)
   - The heading is easy -- use `<font color='red'>` for that heading

   - The $ sign and red+bold is going to take some logic processing
   - In the **`item/@ID | item/*`** template, we are using a simple `<xsl:value-of select='.'/>` to extract the item data, regardless of which value it is (`artist`, `price`, etc.)

   - These are the actions we have to take:
   - If the node is **`price`**, add the $ and make the value red+bold
   - If the node is **`listPrice`**, just add the $
   - Otherwise, just extract the value (as before)
   - This calls for `xsl:choose ... xsl:when ... xsl:otherwise`

Enhancing the Templates

Lab

```
<xsl:template match='item/@ID | item/*'>
  <td>
  <xsl:choose>
    <!-- see if name of the node is price -->
    <xsl:when test='name(.)="price"'>
      <font color='red'>
        <b>$<xsl:value-of select='.'/></b>
      </font>
    </xsl:when>
    <!-- see if name of the node is listPrice -->
    <xsl:when test='name(.)="listPrice"'>
      $<xsl:value-of select='.'/>
    </xsl:when>
    <xsl:otherwise>
      <!-- simply extract the value as before -->
      <xsl:value-of select='.'/>
    </xsl:otherwise>
  </xsl:choose>
  </td>
</xsl:template>
```

Enhancing the Templates

Lab

Tasks to Perform

4. Use the shipper's corporate colors for the shipper data
   - Brown if the shipper is UPS
   - Blue and orange if the shipper is FedEx

   - In the **`shipper`** template, we need to determine the value of the **`name`** attribute

   - These are the actions we then have to take:
   - If the value of `name` is **`UPS`**, make both name and account number brown
   - If the value of `name` is **`FedEx`**, make name blue and account number orange
   - Otherwise, just extract the values (as before)

Enhancing the Templates

Lab

```
<xsl:template match='shipper'>
  <font size='+2'><b>Shipping Info</b></font><br/>
  <xsl:choose>
    <!-- see if value of name attribute is FedEx -->
    <xsl:when test='@name="FedEx"'>
      <font color='blue'>
        <xsl:value-of select='@name'/>
      </font>
      <br/>
      <font color='orange'>
        <xsl:value-of select='@accountNum'/>
      </font>
    </xsl:when>
    <!-- likewise for UPS - test='@name="UPS"' -->
    <xsl:otherwise>
      <!-- simply extract the values as before -->
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Doing the Transform

Lab

Tasks to Perform

- Transform the order as before, and view the output (*order.out.html*)
  - Look at the result in a Web browser to check your work
  - Are your enhancements in place?
- Change the value of the shipper name in *order.xml* to `FedEx` and do the transform again -- do the colors change?
  - Change the value to `USMail` -- is the shipping information in plain black text?

STOP

Lab

[Optional] Lab 10.7 – Transforming in Browsers and Java

- **Purpose**: In this lab, you will see how to use XSLT in other environments
- **Builds on previous labs**: Lab 10.6
- **Approximate Time**: 20-30 minutes

Part A: Using a Browser’s XSLT Engine

Lab

Tasks to Perform

- NOTE: Only works in a few browsers now – see notes
- Continue working in your **Lab09.3** project for this part
- Make a copy of *order.xml* and name the copy ***order-pi.xml***
- Insert an **`xml-stylesheet`** PI into *order-pi.xml*

```
<?xml-stylesheet
  href='order-html.xsl'
  type='text/xsl'?>
```

- Load *order-pi.xml* into an XSLT-enabled browser (FireFox)
  - Do you see the transform being performed in the browser?

Part B – Use JAXP to Transform

Tasks to Perform

- You'll create a new project for this part
- Create a Java Project (**File | New | Project | Java Project**)
  - Call the project **Lab10.7**
  - Remember to **uncheck** "Create module-info.java file"
  - Click **Finish**
- This project contains the XSLT.java program seen earlier, as well as all the XML files needed (*order.xml, order.xsd, order-html.xsl*)
  - Open XSTL.java and review it
  - Right click on *XSLT.java*, and select **Run as | Java application**
  - Refresh the project to pick up the output file (order-result.html)
  - Right click on this file, and select **Open with | Web browser**
  - You'll see the generated HTML
  - You've used XSLT via a Java program

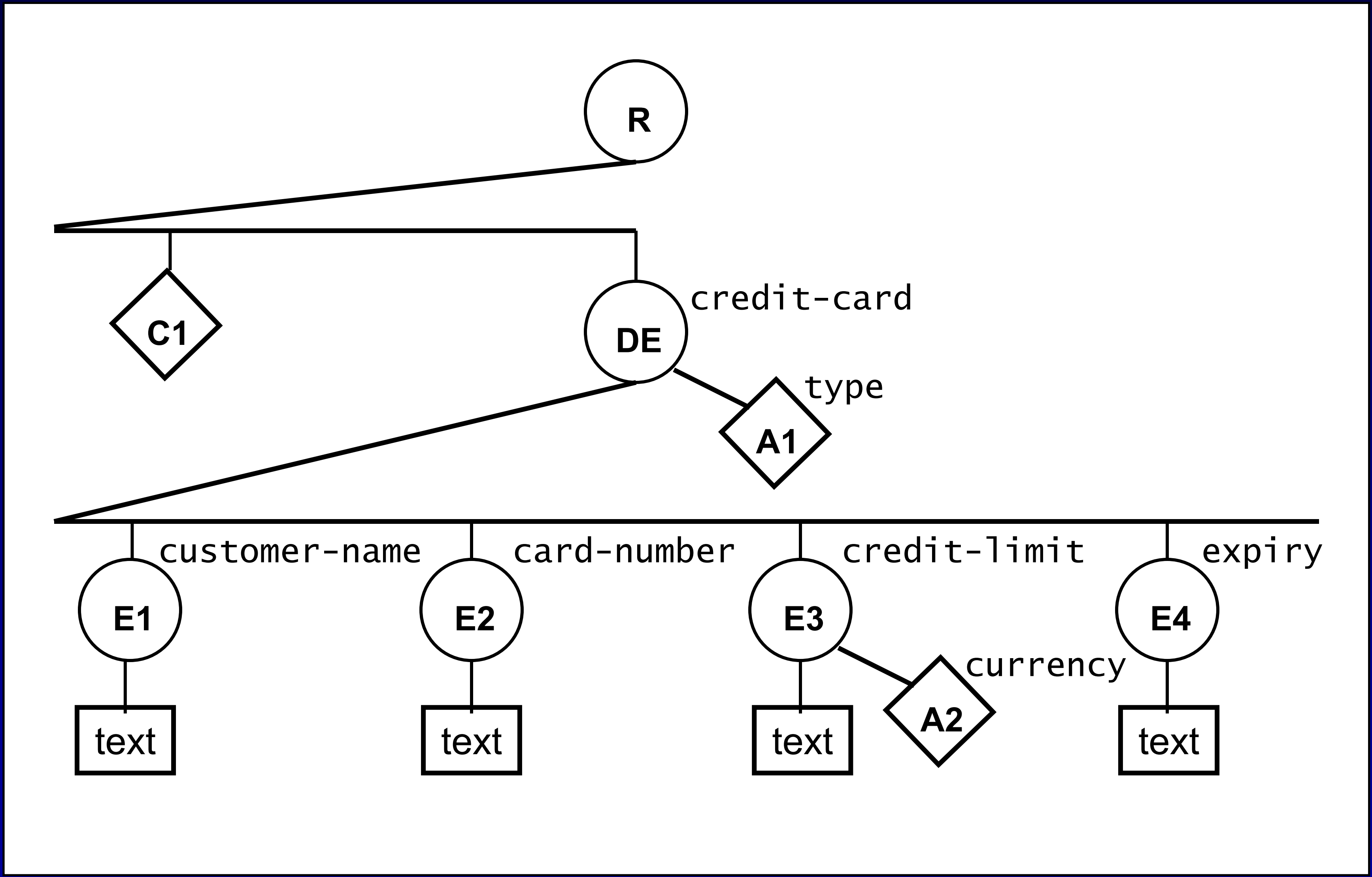
XPath Lab Solutions

Lab 9.1 - Part One: Credit Card XML Document

- Part One:
  - Below is a JavaTunes credit card XML document
  - Draw an XPath node tree for this XML document
  - Compute the name and string-value of each node

```xml
<?xml version='1.0'?>
<!-- JavaTunes credit-card XML document -->
<credit-card type='VISA'>
  <customer-name>Marvin Gardiner</customer-name>
  <card-number>4987877419837781</card-number>
  <credit-limit currency='USD'>5000</credit-limit>
  <expiry>12/05</expiry>
</credit-card>
```

Lab 9.1 - Part One: Node Tree



Lab 9.1 - Part One: Names and String-values

- Root node
  - name: **-none-**
  - string-value: **Marvin Gardiner4987877419837781500012/05**
- `credit-card` element node (document element)
  - name: **credit-card**
  - string-value: **Marvin Gardiner4987877419837781500012/05**
- `customer-name` element node
  - name: **customer-name**
  - string-value: **Marvin Gardiner**
- `card-number` element node
  - name: **card-number**
  - string-value: **4987877419837781**

Lab 9.1 - Part One: Names and String-values

- `credit-limit` element node
  - name: **credit-limit**
  - string-value: **5000**
- `expiry` element node
  - name: **expiry**
  - string-value: **12/05**
- `type` attribute node
  - name: **type**
  - string-value: **VISA**
- `currency` attribute node
  - name: **currency**
  - string-value: **USD**

Lab 9.1 - Part One: Names and String-values

- Comment node
  - name: **-none-**
  - string-value: **JavaTunes credit-card XML document**
- All text nodes
  - name: **-none-**
  - string-value: **-text contents of node-**

Lab 9.1 - Part Two: JavaTunes Order

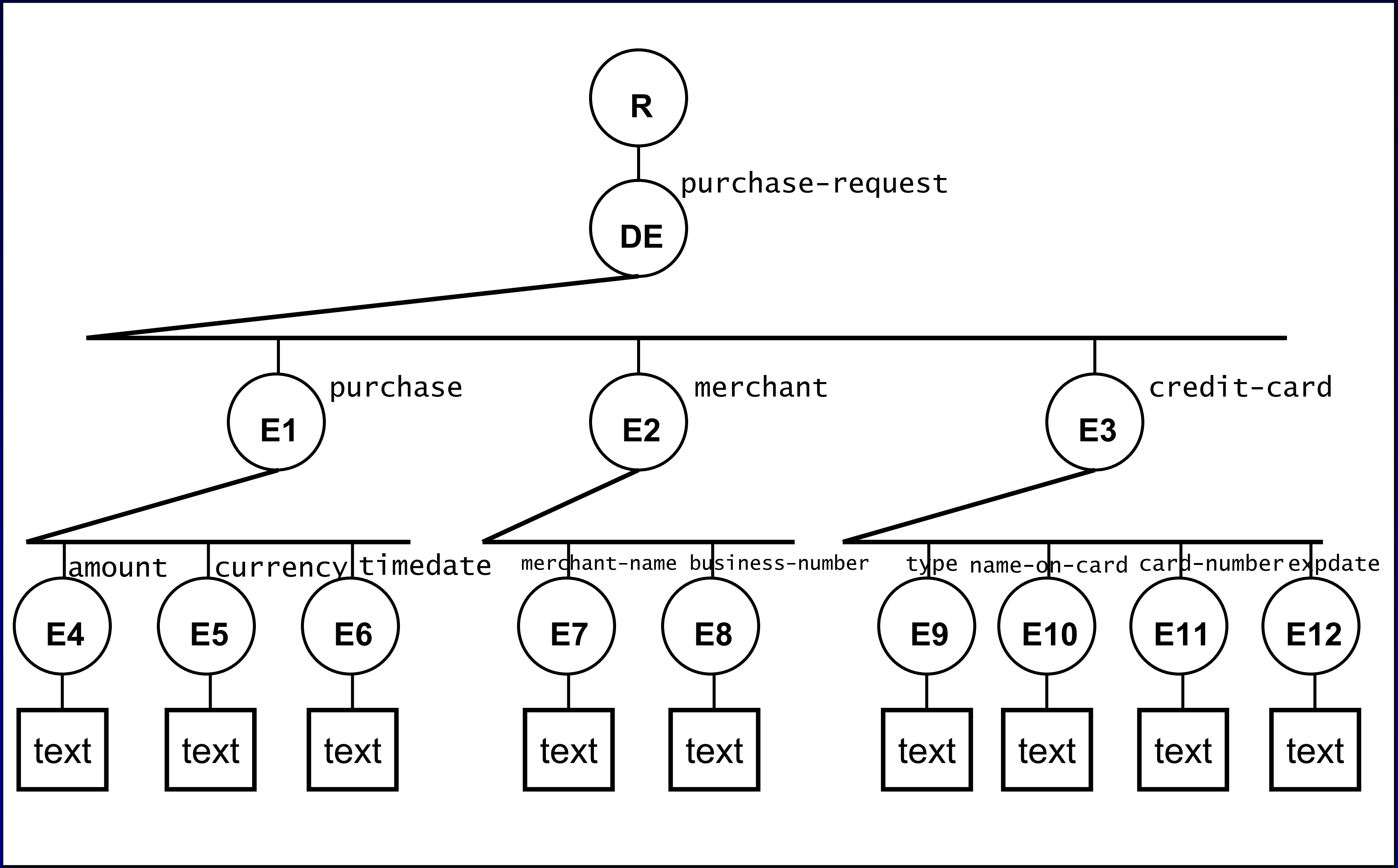
- First `item` element
  - name: **item**
  - string-value: **SoPeterGabriel1986-10-0317.9713.99**
- `shipper` element
  - name: **shipper**
  - string-value: **-null-** (empty element)
- Document element's (`order`) `ID` attribute
  - name: **ID**
  - string-value: **_01170302**
- First text node descendant of `customer` element
  - name: **-none-**
  - string-value: **Susan Phillips**

Lab 9.2 - Part One: XML Document

- Part One - draw a node tree for this document - optional

```
<?xml version='1.0'?>
<purchase-request>
  <purchase>
    <amount>10.00</amount>
    <currency>USD</currency>
    <timedate>2003-01-18T14:21:00</timedate>
  </purchase>
  <merchant>
    <merchant-name>JavaTunes</merchant-name>
    <business-number>987676257625</business-number>
  </merchant>
  <credit-card>
    <type>Visa</type>
    <name-on-card>Bob Smith</name-on-card>
    <card-number>1987987399918277</card-number>
    <expdate>01/04</expdate>
  </credit-card>
</purchase-request>
```

Lab 9.2 - Part One: Node Tree

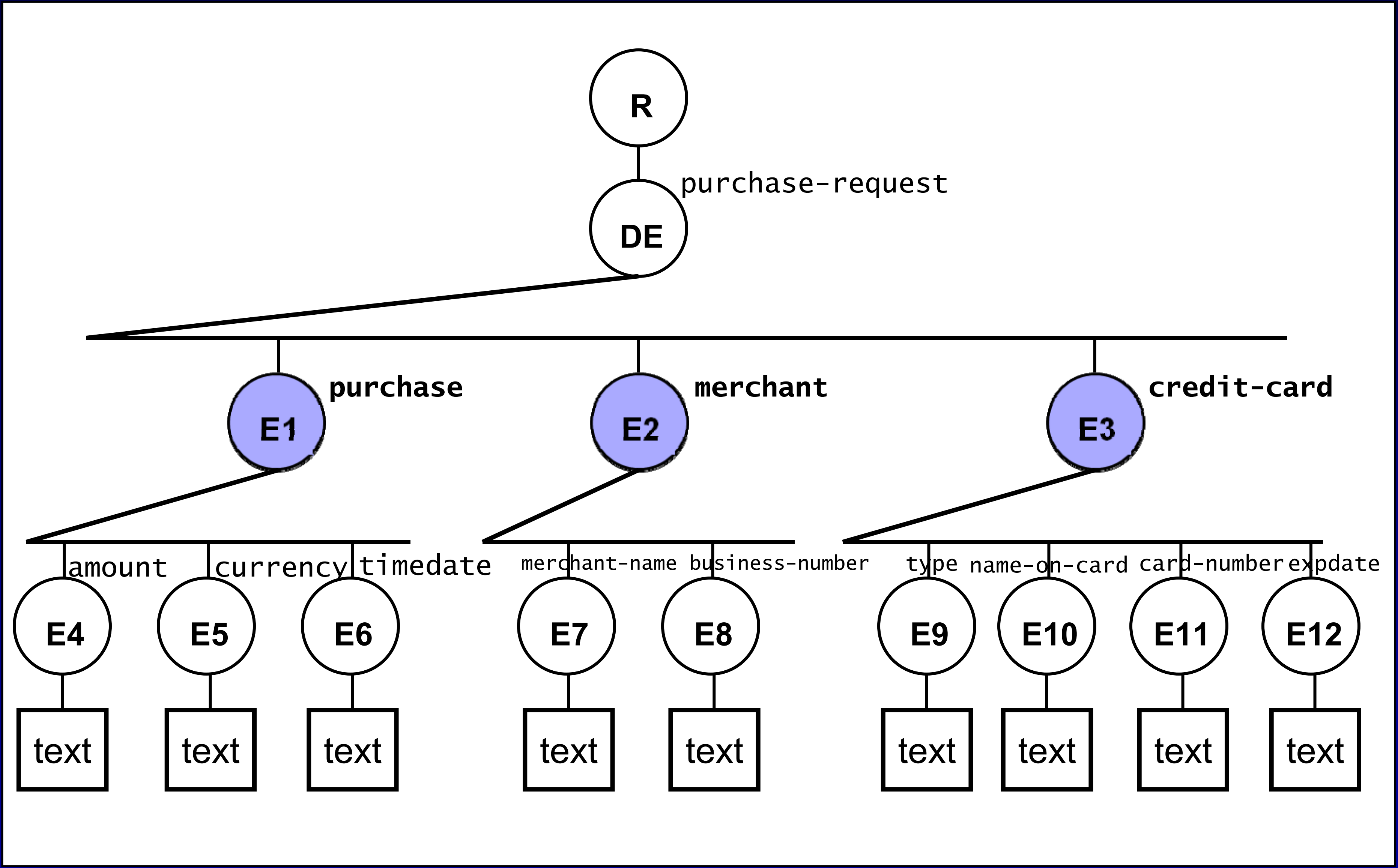


Lab 9.2 - Part Two: Location Paths

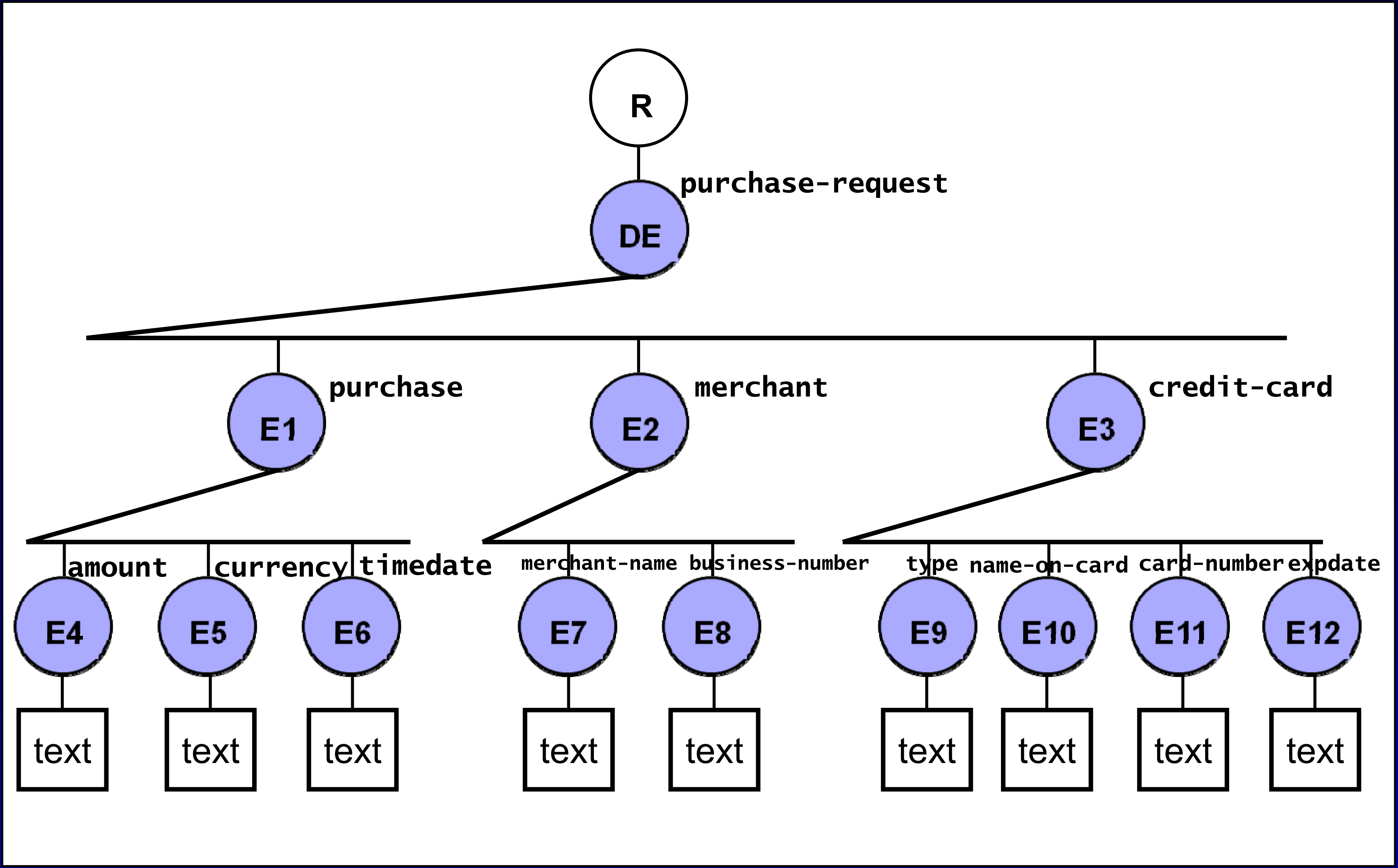
- Part Two - identify the resulting node-set for each of the following XPath expressions

```
/purchase-request/*

//*

/*/*/amount

/*/card-number

/*/*/type/..

//credit-card//text()
```

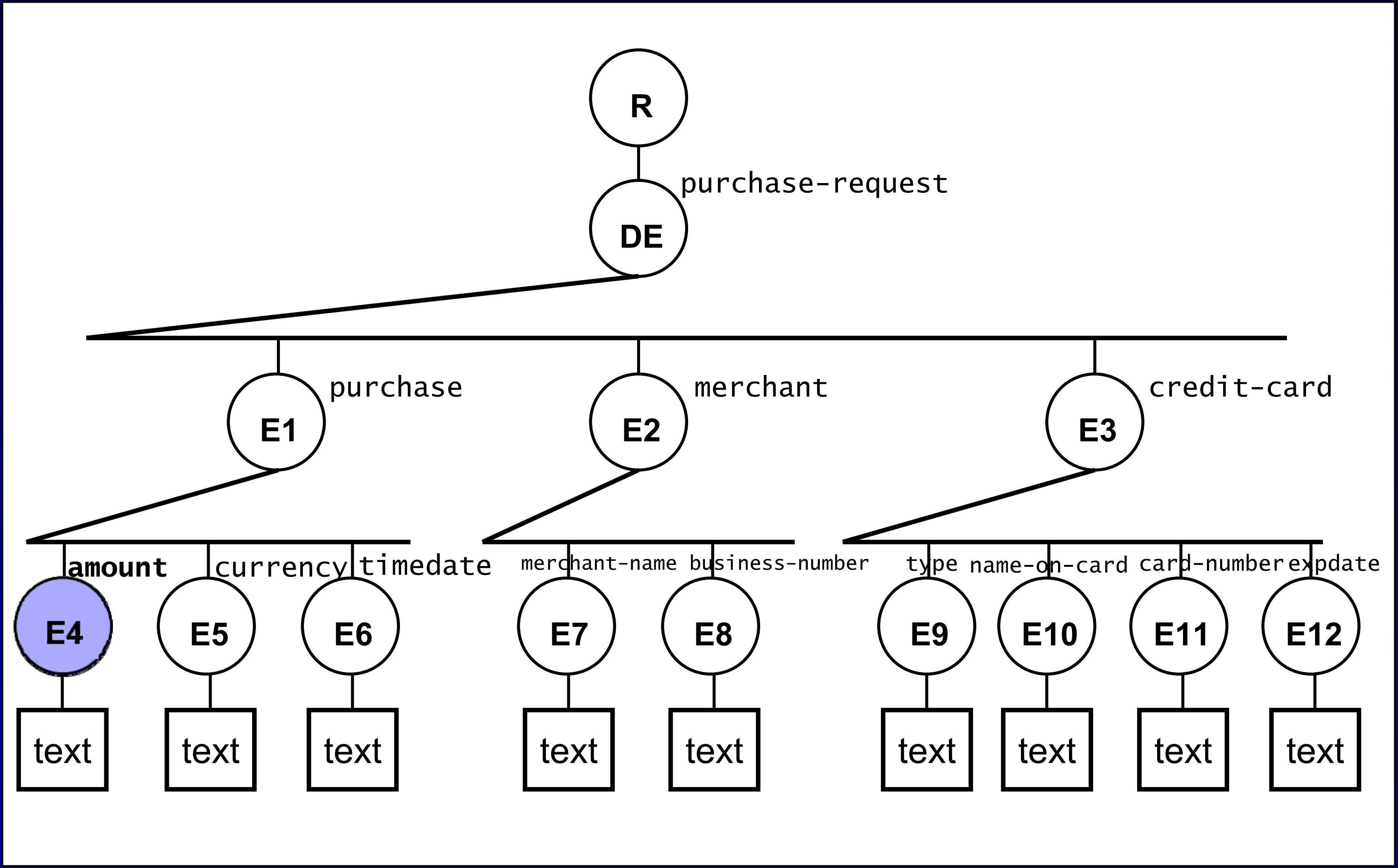
Lab 9.2 - Part Two: /purchase-request/*



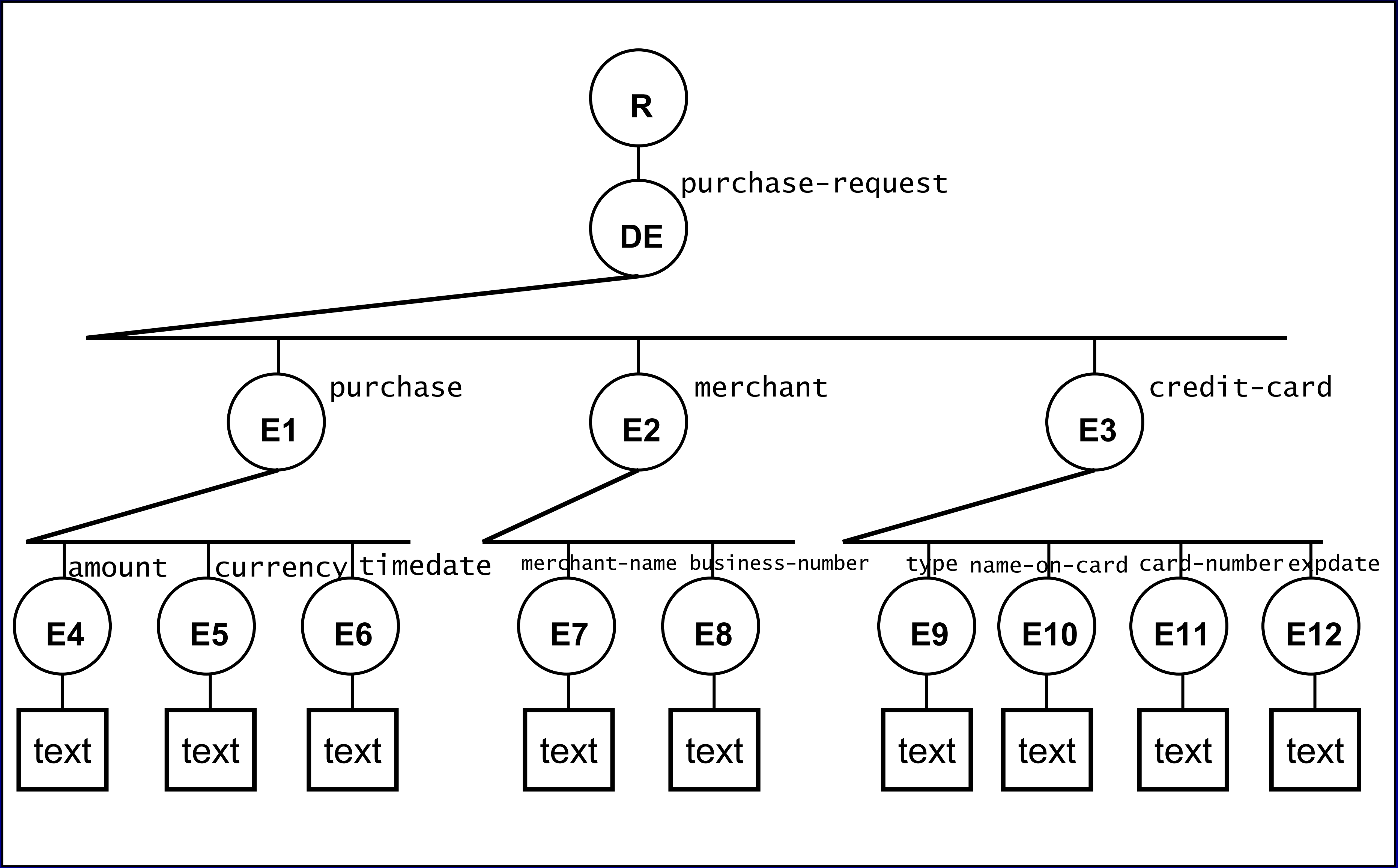
Lab 9.2 - Part Two: //*



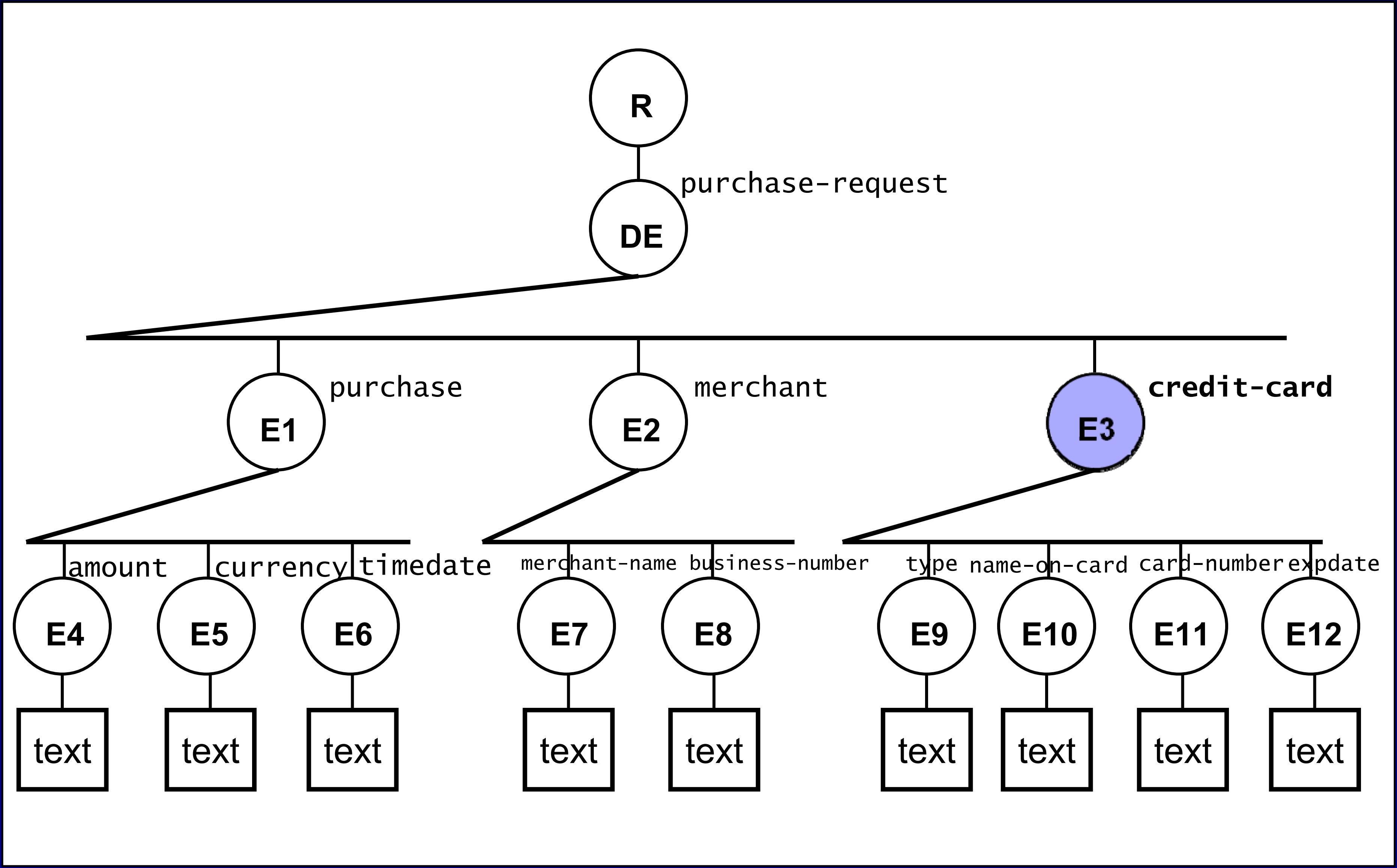
Lab 9.2 - Part Two: /*/*/amount



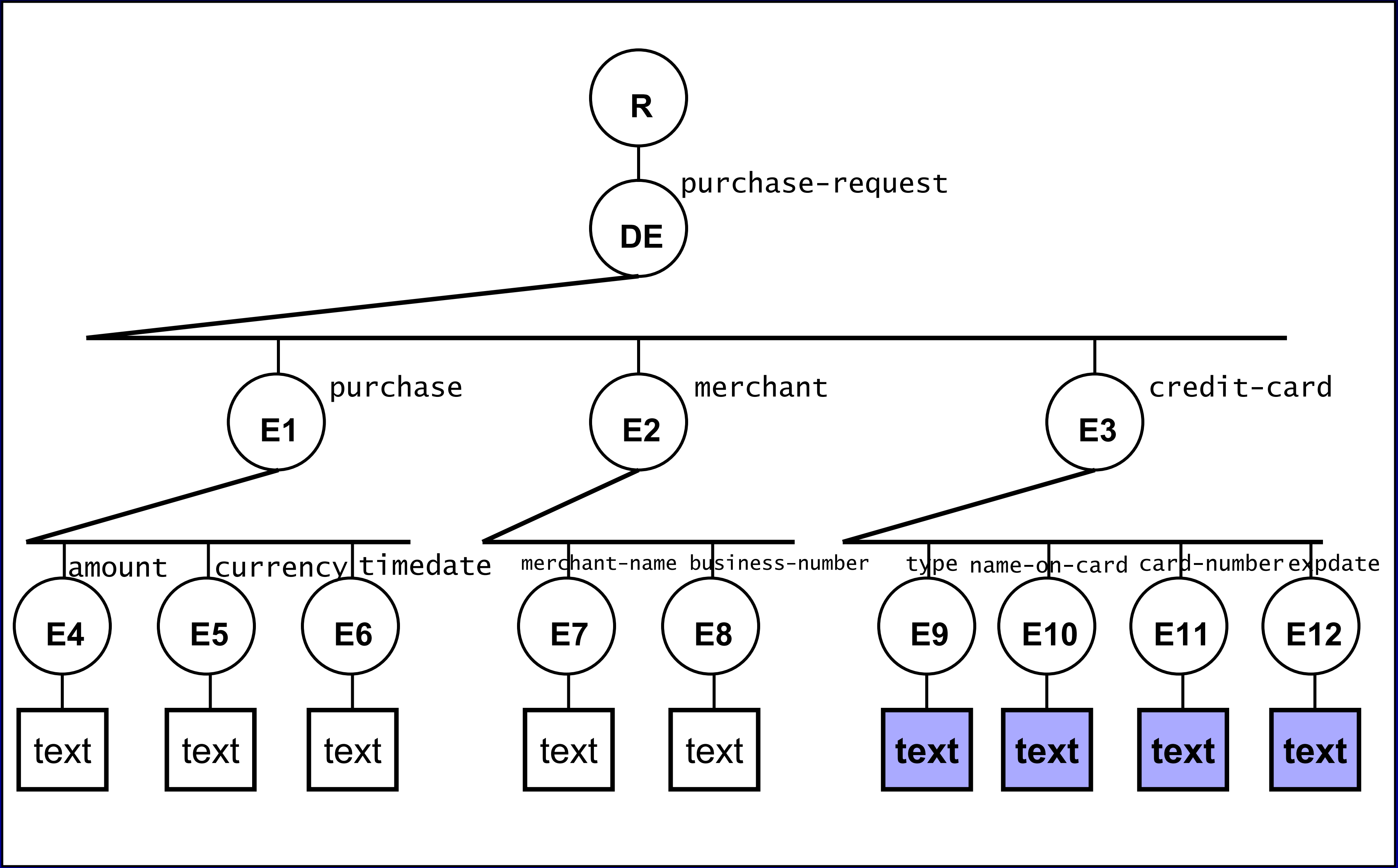
Lab 9.2 - Part Two: /*/card-number



Lab 9.2 - Part Two: /*/*/type/..



Lab 9.2 - Part Two: //credit-card//text()



Lab 9.2 - Part Three

- Part Three - for the purchase request document, write two different location paths for each of the following -- use absolute location paths (see notes)
  - Parent of the **`type`** element
  - **`amount`** element
  - **All** children of the **`credit-card`** element (be careful on this one)
  - **Element** grandchildren of the **document element**
  - Text node descendants of the **`purchase`** element

Lab 9.2 - Part Three

- Parent of the `type` element

  `//type/..` **`/purchase-request/*/type/..`**

- `amount` element

  `//amount` **`/purchase-request/purchase/amount`**

- All children of the `credit-card` element

  `//credit-card/node()`

  `/purchase-request/credit-card/node()`

- Element grandchildren of the document element

  `/*/*/*` **`/purchase-request/*/*`**

- Text node descendants of the `purchase` element

  `//purchase//text()`

  `/purchase-request/purchase//text()`

Lab 9.3 - Predicates, Functions, Operators

- Using a JavaTunes order XML document (see next slides), write an XPath expression to select:
  1. The **value** (text content) of the state in which the customer lives
  2. The **value** (text content) of the second item's artist
  3. The item whose ID is CD503
  4. The items by an artist whose name begins with Peter
  5. The number of items in the order with a list price of at least $15
  6. The total cost of the order (items are sold at price, not list price)
  7. The average cost of the items in the order (items are sold at price, not list price) -- there is no `average()` function in XPath
  - Use absolute location paths in the expressions that are paths
- **NOTE** that not all of these expressions are used in predicates
  - For example, the expression that selects the number of customers in the order is simply `count(/order/customer)`

Lab 9.3 - JavaTunes Order XML Document

```
<?xml version='1.0'?>

<!-- JavaTunes order XML document -->

<order ID='_01170302' dateTime='2002-03-20T05:02:00'
 xmlns:xsi='...' xsi:noNamespaceSchemaLocation='...'>
  <customer>
    <name>Susan Phillips</name>
    <street>763 Rodeo Circle</street>
    <city>San Francisco</city>
    <state>CA</state>
    <zipcode>94109</zipcode>
    <shipper name='UPS' accountNum='343-9080-1'/>
  </customer>
  ...
```

Lab 9.3 - JavaTunes Order XML Document

```
   ...
   <item ID='CD514'>
     <name>So</name>
     <artist>Peter Gabriel</artist>
     <releaseDate>1986-10-03</releaseDate>
     <listPrice>17.97</listPrice>
     <price>13.99</price>
   </item>
   <item ID='CD517'>
     <name>1984</name>
     <artist>Van Halen</artist>
     <releaseDate>1984-08-19</releaseDate>
     <listPrice>11.97</listPrice>
     <price>11.97</price>
   </item>
   <item ID='CD503'>
     <name>Trouble is...</name>
     <artist>Kenny Wayne Shepherd Band</artist>
     <releaseDate>1997-08-08</releaseDate>
     <listPrice>17.97</listPrice>
     <price>14.99</price>
   </item>
</order>
```

Lab 9.3 - Predicates, Functions, Operators

1. `/order/customer/state/text()`
2. `/order/item[2]/artist/text()`
3. `/order/item[@ID='CD503']`

4.
   ```
   /order/item[starts-with(artist, 'Peter')]
   /order/item[starts-with(artist/text(),
                              'Peter')]
   ```
5. `count(/order/item[listPrice>=15])`
6. `sum(/order/item/price)`
7. `sum(/order/item/price) div count(/order/item)`