Project 4

Scamazon.com

140 Points

due by 12:30 P.M. ET on Wednesday, 25 April 2007

It is recommended that you read the entirety of this document,
well in advance of this project's deadline, before answering any of its questions.

It is further recommended that you begin answering this document's questions immediately thereafter. After all, the sooner you finish, the more time you'll have for your final project!

And it is also recommended that you request, per problem 11 herein,
an "Access Key ID" for Amazon's Web Services immediately,
lest it take time for your request to be processed.

Goals.

The goals of this project are to:

- Challenge you to build your own e-commerce engine and fulfillment partner.
- Give you hands-on experience with DTD, SOAP, web services, and XML Schema, as well as additional experience with JavaServer Pages 2.1, Java Servlet 2.5, TrAX, XSL-FO, and XSLT.
- Introduce you to Apache's Axis 1.4 and to FOP 0.93's API.
- Synthesize Lectures 1 through 11 and equip you for your final project and future XML endeavors!

Grading Metric.

Each question is worth the number of points specified parenthetically in line with it.

Your responses to questions requiring exposition will be graded on the basis of their clarity and correctness. Your responses to questions requiring code will be graded on the following bases.[1]

Basis	Considerations
Correctness	Does your code work in accordance with the question's guidelines?
Design	Does your code make sense, given the question's framework? Is your code written logically, clearly, and succinctly? Is your code efficient? Is your code divided into logical units (*e.g.*, multiple methods and templates)?
Style	Is your code rigorously documented with inline comments and Javadoc doc comments and tags?[2] Is it clear from your comments alone how your code operates? Is your code pretty-printed? Are your attributes, data members, elements, methods, templates, and variables aptly named? Do your files contain lines of code that have been commented out?[3]

Academic Honesty.

Again for this project, you will be running your own instance of a webserver on `nice.fas.harvard.edu` which does not require that any files or directories being served be world-executable or world-readable.

If, however, you opt to publish world-readable files on `www.people.fas.harvard.edu` for some reason, it is expected that you will ensure the privacy of your work by password-protecting any Web-accessible directories and naming files in such a (random) way as to render their viewing on `nice.fas.harvard.edu` unlikely.

Recall that, for Project 1, you configured your account with a `~/public_html/` directory and a password-protected `~/public_html/cscie259/` directory.

Needless to say, attempting to view the work of another student, even if published in a world-accessible directory, is considered academic dishonesty and will be handled accordingly.

[1] By code, we mean DTD, Java, JSP, XHTML, XML, XML Schema, XPath, XSL-FO, and XSLT.
[2] With regard to Javadoc, we expect descriptions for all methods and appropriate use of the `@param`, `@return`, and `@throws` block tags.
[3] They shouldn't.

Getting Started.

1. (0 points.) No point for you this time!

2. (0 points.) If you intend to do your work on `nice.fas.harvard.edu`, SSH to that machine and execute the following sequence of commands.[4,5]

```
cp -r ~cscie259/pub/distribution/projects/project4-7.0/ ~/cscie259/
cd ~/cscie259/
ls
```

   You should see that you have the following in your current working directory

```
project1-7.0/       project2-7.0/     project3-7.0/     project4-7.0/
```

   If, on the other hand, you do not intend to do your work on `nice.fas.harvard.edu`, you may proceed to download a `gzip`-compressed tarball or a ZIP file containing this `project3-7.0/` directory from the course's website to your local machine. Or, of course, you can transfer the directory itself via SFTP to your local machine. Refer to this document's Appendix for further directions.

   Notice, now, that your `project4-7.0/` directory is structured as follows.

[4] These commands assume that you already created (for Projects 1, 2, and 3) a directory called `cscie259/` in your FAS account's home directory.
[5] Beware the distinction between `~cscie259` and `~/cscie259`.

```
project4-7.0/
  conf/
  src/
    com/
      amazon/
        soap/
          AWSECommerceService/
    cscie259/
      project4/
        scamazon/
        warehouse/
  temp/
  webapps/
    ROOT/
      docs/
        com/
          amazon/
            soap/
              AWSECommerceService/
        cscie259/
          project4/
            scamazon/
            warehouse/
      WEB-INF/
    scamazon/
      dtd/
      images/
      WEB-INF/
        lib/
      xml/
        cache/
      xsd/
      xsl/
    warehouse/
      dtd/
      images/
      WEB-INF/
      xml/
      xsd/
      xsl/
```

Ensure that your account or machine is in order by executing the following command from within any directory.[6]

```
EnvironmentCheck
```

Confirm that your account is configured to use Ant 1.7.0, Xerces-J 2.7.1, and Xalan-J 2.7.0. Next, execute the following command from within your `project4-7.0/` directory.[7,8]

[6] Note that, on `nice.fas.harvard.edu`, `` `EnvironmentCheck` `` is an alias for `` `java org.apache.xalan.xslt.EnvironmentCheck` ``.

[7] Alternatively, you can execute `ant` without any arguments.

[8] You can ignore any notes about unchecked or unsafe operations in the source files destined for `project4-7.0/webapps/scamazon/WEB-INF/classes/`.

```
ant compile
```

By default, `ant` will compile (among other things) `cscie259.project4.scamazon.*`, storing the resulting bytecodes in `project4-7.0/webapps/scamazon/WEB-INF/classes/`, and `cscie259.project4.warehouse.*`, storing the resulting bytecodes in `project4-7.0/webapps/warehouse/WEB-INF/classes/`.

As always, take a look at `project4-7.0/build.xml` so that you know what `ant` can do for you during this project.

Then, assuming you've connected via SSH to `nice.fas.harvard.edu`, take notice of the particular host to which you're connected by examining your prompt (which should be of the form `"$USER@$ICEBOX (%~):"`, per Project 1's configuration of your account) or by executing the following command.[9]

```
echo $ICEBOX
```

Next, proceed to edit `project4-7.0/conf/server.xml` with your favorite text editor. Per the file's comments, locate the `Server` element's `port` attribute and assign it an integral value between 1024 and 65535, inclusive; then, locate the `Connector` element's `port` attribute and assign it an integral value between 1024 and 65535, inclusive, as well, taking care not to reuse the value you chose for the `Server` element's `port` attribute. After saving and closing the file, execute the following command from within `project4-7.0/`.

```
tomcat
```

You should see output resembling the below, where `$ICEBOX` is the host to which you're connected, `m` is your choice of ports for the `Server` element's `port` attribute, and `n` is your choice of ports for the `Connector` element's `port` attribute.

[9] Though we referred to it as one system prior to Project 3, `nice.fas.harvard.edu` actually refers to a cluster of machines, all of which mount via NFS your home directory; you are connected to one essentially at random.

```
Using CATALINA_BASE:   .
Using CATALINA_HOME:   /home/c/s/cscie259/pub/local/jvm/apache-tomcat-6.0.10
Using CATALINA_TMPDIR: ./temp
Using JRE_HOME:       /home/c/s/cscie259/pub/local/i386/jdk1.5.0_11
Mar 27, 2007 12:11:09 AM org.apache.coyote.http11.Http11Protocol init
INFO: Initializing Coyote HTTP/1.1 on http-n
Mar 27, 2007 12:11:09 AM org.apache.catalina.startup.Catalina load
INFO: Initialization processed in 712 ms
Mar 27, 2007 12:11:09 AM org.apache.catalina.core.StandardService start
INFO: Starting service Catalina
Mar 27, 2007 12:11:09 AM org.apache.catalina.core.StandardEngine start
INFO: Starting Servlet Engine: Apache Tomcat/6.0.10
Mar 27, 2007 12:11:11 AM org.apache.coyote.http11.Http11Protocol start
INFO: Starting Coyote HTTP/1.1 on http-n
Mar 27, 2007 12:11:12 AM org.apache.catalina.startup.Catalina start
INFO: Server startup in 2452 ms
```

At this point, an instance of Tomcat is running under your userid on port `n` of the host to which you're connected. (Tomcat should now have control of your terminal; you should not be returned to your shell's prompt.) Confirm as much by visiting the URL below, taking care to substitute `$ICEBOX` and `n` with their appropriate values, per your `$ICEBOX` and `Connector`'s `port`.[10,11]

```
http://$ICEBOX.fas.harvard.edu:n/
```

You should see a directory listing for your container's root directory (namely, `project4-7.0/webapps/ROOT/`), the contents of which include `docs/` (the project's initial Javadoc), `happyaxis.jsp` (a JSP that ships with Axis), and `happyenv.jsp` (a JSP that the staff whipped up). Confirm that the listing's footer reports version 6.0.10 of Tomcat.

Confirm also that Axis is functioning properly by clicking on `happyaxis.jsp`: all "Needed Components" and "Optional Components" should be present. That is, just above a horizontal rule, the page should report: "*The core axis libraries are present. The optional components are present.*" Then, scroll down a bit and confirm that "Servlet version" is 2.5, that `java.version` is `1.5.0_11`, that the `local.host` system property is present and its value is your host's fully qualified domain name or IP address,[12] and that the remaining properties appear as expected.

Next, go back to your root directory's listing and click on `happyenv.jsp`. Confirm that Tomcat is configured to use JDK 5.0 Update 11, Xerces-J 2.7.1, and Xalan-J 2.7.0 and that the remaining properties appear as expected.

Proceed to terminate Tomcat by hitting `Ctrl-C` at its console (*i.e.*, your terminal).[13]

If your account or system ain't so happy, feel free to contact `cscie259@lists.dce.harvard.edu` or the staff for assistance.

---

[10] If you opt to develop on your own machine, you should instead visit `http://localhost:n/` or `http://127.0.0.1:n/` in lieu of this and all future references to `http://$ICEBOX.fas.harvard.edu:n/`.

[11] Although you needn't type `"http://"` to visit most websites with most browsers these days, be sure to do so in this case.

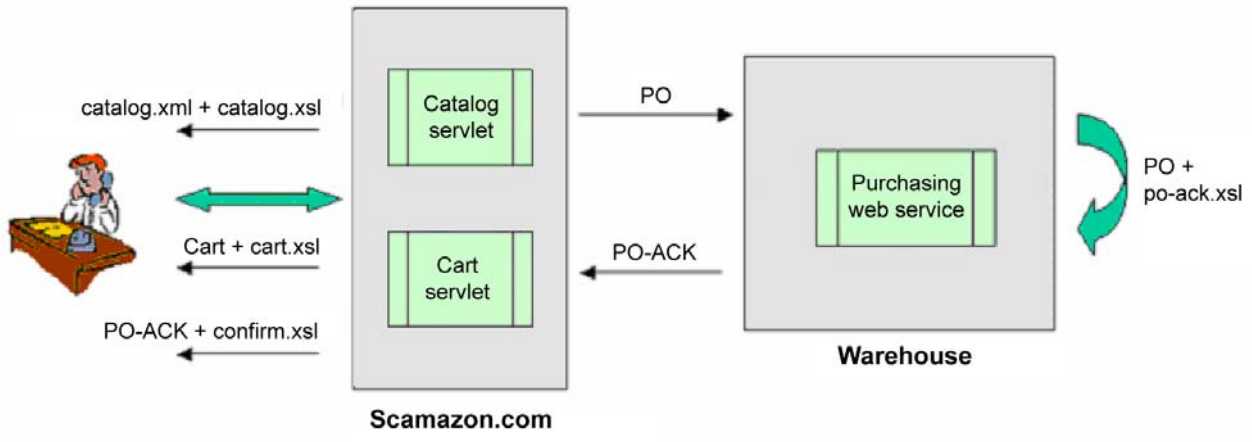
[12] Recall that this property should have been set by way of your `CATALINA_OPTS` environment variable.

[13] Now and hereafter, you can ignore any exceptions thrown by Axis's `SOAPMonitorService`.

Earth's Smallest Selection.

3. (0 points.) Your mission for this project, if you choose to accept it,[14] is to build an e-commerce engine whose orders will be transmitted to and (not actually) fulfilled by a local warehouse. In fact, upon completion, your project will be the place (not) to shop! Visitors will be able to browse your project's dynamically generated aisle(s), add items to your project's shopping cart, and then check out, their order (never to be) fulfilled by your project's warehouse, a web service!

   Quite the scam, eh?
   Lest you be overwhelmed by thoughts of supply-chain management, inventory management, bottom dollars, and the like, rest assured that we've again provided you with a framework. Per Lecture 9, this project's architecture resembles the below.[15]



   Before trying to wrap your mind around this architecture, take a look at `project4-7.0/webapps/scamazon/xml/catalog.xml`; the file details Scamazon.com's offerings. Next, take a look at `project4-7.0/webapps/warehouse/xml/inventory.xml`; the file specifies how many of each item is currently in stock at the warehouse.

   Now, as the figure above suggests, Scamazon.com currently embodies two servlets: a `Catalog` servlet that will allow a visitor to browse the site's catalog and a `Cart` servlet that will allow that visitor to add items to his or her shopping cart and submit an order. Upon submission of an order, Scamazon.com will pass that order (in the form of an XML element called `PO`) along to the local warehouse, which is to be implemented as a web service called `Purchasing`. The warehouse will check its inventory and respond to the order (in the form of an XML element called `PO-ACK`), acknowledging which items are en route (er, would be, if this weren't a scam) and which are backordered. (For simplicity, the warehouse will operate out of the same instance of Tomcat. However, Scamazon.com will still communicate with the web service via SOAP; in fact, the former won't really know the latter is running on the same host.)

[14] On second thought, you must accept it.
[15] The man on the phone at the desk, however, may not resemble you.

The `Cart` servlet will then inform the user of the order's completion and allow the user to obtain a receipt in the form of a PDF. Make sense?

Okay, let's take a look at the code in `project4-7.0/src/cscie259/project4/scamazon/`.

Start by examining Scamazon.com's `Catalog` servlet by way of `Catalog.java`; notice how it aspires to apply `project4-7.0/webapps/scamazon/xsl/catalog.xsl` to `project4-7.0/webapps/scamazon/xml/catalog.xml` in order to generate your project's aisle(s). Then take a look at the `Cart` servlet by way of `Cart.java`; eventually, this servlet will handle all operations involving the user's shopping cart. Finally, give `Proxy.java` a once-over: it contains a proxy for (*i.e.*, a method with which to invoke) the warehouse's `Purchasing` service.

Let's now examine the code in `project4-7.0/src/cscie259/project4/warehouse/`. Wow, not all that much there, eh? Although `Purchasing.java` appears only to define a class called `Purchasing`, that class will be "magically" transformed into a web service by Axis, a SOAP engine we've integrated into Tomcat.

Did you notice, by the way, that `Catalog.java` and `Purchasing.java` are already capable of validating XML inputs, either by way of DTD or XML Schema?

If you prefer Javadoc over this stroll down directory lane, feel free to view `project4-7.0/webapps/ROOT/docs/` with your favorite browser. We've placed the project's initial Javadoc in `project4-7.0/webapps/ROOT/` so that you can view our (and, soon, your) Javadoc with your own instance of Tomcat![16]

Now take a look at this project's configuration. Look over `web.xml` in `project4-7.0/webapps/scamazon/WEB-INF/` to see how Scamazon.com is configured. Then look over the same in `project4-7.0/webapps/warehouse/WEB-INF/` to see how the warehouse is configured; also take a look at that directory's `server-config.wsdd`. You certainly needn't understand the entirety of those files' syntax; but try to get a sense of what each file is doing for you.

Alright, on your mark, get set—

[16] During the development of this project, you can update your Javadoc by executing `` `ant javadoc` `` from within your `project4-6.0/` directory.

Go!

4. (5 points.) Go ahead and spawn Tomcat again from within your `project4-7.0/` directory. This time, though, point your browser at the URL below.

   ```
   http://$ICEBOX.fas.harvard.edu:n/scamazon/
   ```

   You should be immediately redirected to the URL below.

   ```
   http://$ICEBOX.fas.harvard.edu:n/scamazon/servlet/catalog
   ```

   Not the sexiest of online catalogs yet, is it?

   Go ahead and kill Tomcat. Then execute the following command (at the same terminal in which you've been running Tomcat), the effect of which will be to enable validation by DTD.[17]

   ```
   setenv CATALINA_OPTS "$D_LOCAL_HOST $D_VALIDATION_DTD"
   ```

   Spawn Tomcat again from within your `project4-7.0/` directory. Then re-visit the URL below.[18]

   ```
   http://$ICEBOX.fas.harvard.edu:n/scamazon/
   ```

   Notice that a blank page is returned this time (generated automatically by Tomcat). Moreover, notice Tomcat's console: it should have reported that no grammar was found. That is, `catalog.xml` failed validation because it hasn't even a DTD associated with it (yet)!

   Well this is no good. Go ahead and define a DTD for `catalog.xml` in `project4-7.0/webapps/scamazon/dtd/catalog.dtd`. Rest assured that more than one solution may be possible. But, in the interests of defending your application against bad data, try to make your DTD as restrictive as possible. When ready to test your design, make `catalog.dtd` an external subset for `catalog.xml` by adding the following just below the latter's XML declaration.

   ```
   <!DOCTYPE items SYSTEM "webapps/scamazon/dtd/catalog.dtd">
   ```

   Perfect your DTD by refreshing your view of Scamazon.com's (minimalist) catalog and watching Tomcat's console, tweaking the DTD as necessary.

   You may not, however, alter the structure of `catalog.xml` beyond adding the aforementioned `DOCTYPE` element.

---

[17] Windows users should instead execute the below.

```
set CATALINA_OPTS=%D_LOCAL_HOST% %D_VALIDATION_DTD%
```

[18] Be sure to reload the page; beware caching browsers.

5. (5 points.) Given the shortcomings of DTD, your `catalog.dtd` probably doesn't embody the ideal definition of Scamazon.com's catalog.[19] Let's take another stab at the problem, this time with the more powerful, the more flexible, the more, er, complicated XML Schema!

   First, kill Tomcat yet again and remove from `catalog.xml` the `DOCTYPE` declaration you just inserted. Then execute the following command (at the same terminal in which you've been running Tomcat), the effect of which will be to enable validation by XML Schema.

   ```
   setenv CATALINA_OPTS "$D_LOCAL_HOST $D_VALIDATION_XSD"
   ```

   Spawn Tomcat yet again from within your `project4-7.0/` directory. Then re-visit the URL below.

   ```
   http://$ICEBOX.fas.harvard.edu:n/scamazon/
   ```

   Again a blank page should be returned. And, again, Tomcat's console has something to say: it should have reported an inability to find the declaration of `items`. Well, of course: `catalog.xml` failed validation because it hasn't an XML schema associated with it!

   In `project4-7.0/webapps/scamazon/xsd/catalog.xsd`, define an XML schema for `catalog.xml`. Be less concerned about imposing order and more concerned about enforcing data types and constraining values. Who cares, after all, if an item's `name` element precedes its `category` element?[20] When ready to test your design, change `catalog.xml`'s root element to the below.

   ```
   <items xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="webapps/scamazon/xsd/catalog.xsd">
   ```

   As before, perfect your schema by refreshing your view of Scamazon.com's `Catalog` servlet and watching Tomcat's console, tweaking the schema as necessary.

6. (0 points.) Once your DTD and schema are working, you may, for the sake of marginally better performance, disable validation by killing Tomcat (at the same terminal in which you've been running Tomcat) and then restarting it, after executing the following command.

   ```
   setenv CATALINA_OPTS "$D_LOCAL_HOST $D_VALIDATION_OFF"
   ```

   After all, `catalog.xml` is static, so there's no need to validate it repeatedly.

[19] No matter how hard you tried!
[20] Well, some applications might. But hopefully not yours. :-)

Checking In.

7. (30 points.) Let's get those virtual aisles up and running.

   Complete the implementation of Scamazon.com's online catalog by editing `catalog.xsl` and `Catalog.java` as you see fit. For now, you needn't worry about any interaction with the site's shopping cart. (However, you're welcome to develop the catalog in lockstep with the `Cart` servlet, whose requirements are discussed later in this document.) Although the design of this catalog is largely up to you, we do ask that you adhere to the following guidelines.

   i. The catalog must somehow display the values of every `label` attribute and every text node in `catalog.xml`; it's not necessary, however, to display the values of any `key` attributes. Rather than display images' file names, however, you should display the images themselves, all of which can be found in `project4-7.0/webapps/scamazon/images/`.[21] Though you probably thought on `catalog.xml`'s structure during development of your DTD and schema, be sure you understand it fully before attempting to display the file's contents.[22]

      It's fine to display the catalog's entire contents in one (large) page (*i.e.*, "aisle"), but you're welcome to partition the catalog into multiple pages. In fact, it'd be neat if you allowed the user to browse the catalog by category; but such is not required.

   ii. Needless to say, your site's aisle(s) should be dynamically generated by some transformation of `catalog.xml` and not defined statically in XHTML files.

   Although works of art are not required, they are certainly encouraged!

Checking Out.

8. (30 points.) Let's grab a shopping cart now and take it out for a spin.

   Implement, in `Cart.java`, some kind of shopping cart and modify your virtual aisle(s) so that a user can somehow add one or more items to that cart via hyperlinks (that reference the `Cart` servlet and include parameters) or via XHTML forms (which can be posted to the `Cart` servlet). Technically speaking, a shopping cart is little more than some mechanism for remembering (for the duration of the current session) which items a user would like to buy.

   Although the design of this shopping cart is largely up to you, we ask that you adhere to the following guidelines.

[21] Feel free to edit or resize these images or even replace them with nicer ones.
[22] If, after significant thought, you're still puzzled by some aspect of `catalog.xml`, feel free to contact the staff or fellow students.

i. The `Cart` servlet must allow a user to add items to and remove items from his or her shopping cart. Moreover, it must be possible somehow to update the quantities of items in the shopping cart (either by submitting specific values or, perhaps more simply, by adding or removing items).

ii. It must be possible to view the contents of one's shopping cart. Not only should the shopping cart display the quantity of each item in the cart, it must also display each item's `category`, `name`, `description`, `descriptor`, and `price`. The shopping cart must also display the total cost of the items therein. (If you'd like to be a geek and compute shipping and tax, feel free. Heck, you can even accept coupons.) All of this information must be assembled by the `Cart` servlet into a `String` of XML (structured however you see fit), which must then be passed to `cart.xsl` (in `project4-7.0/webapps/scamazon/xsl/`) for transformation into XHTML. So that we know the structure of your `String` of XML, create in `project4-7.0/webapps/scamazon/dtd/cart.dtd` a DTD or in `project4-7.0/webapps/scamazon/xsd/cart.xsd` an XML schema. (You needn't create both.) Though your DTD or schema must be correct, you needn't configure your `Cart` servlet validate your XML; after all, since the XML is for Scamazon.com's internal use, it's presumably always valid.

iii. While browsing Scamazon.com's aisle(s), it must be possible to view the contents of one's shopping cart, perhaps via some hyperlink.

iv. While viewing one's shopping cart, it must be possible to return to Scamazon.com's aisle(s), perhaps via some hyperlink.

v. When viewing one's shopping cart, it must be possible to check out. That is, it must be possible to submit the requisite contents of that shopping cart in the form of a `PO` element (*i.e.*, a `String` of XML whose root element is `PO`) to the warehouse's `Purchasing` service. That `String` must be submitted to the web service by way of the `Proxy` class's `processPO(·,·,·)` proxy method.[23] Upon receipt of an acknowledgement from the web service in the form of a `PO-ACK` element (*i.e.*, a `String` of XML whose root element is `PO-ACK`), the `Cart` servlet should display a confirmation of the user's order (again displaying each purchased item's `category`, `name`, `description`, `descriptor`, and `price`). However, the `Cart` servlet should take care to inform the user of how many of each item will actually be shipped (*i.e.*, whether some item was backordered as a result of insufficient availability at the warehouse). This confirmation should be in the form of a webpage, generated dynamically by `confirm.xsl` (in `project4-7.0/webapps/scamazon/xsl`); as before, the `Cart`

[23] Know that you can avoid hardcoding the URL of the warehouse's `Purchasing` service into your code. In fact, you can obtain dynamically the service's hostname by executing the below.

```
System.getProperty("local.host");
```

And you can obtain the service's port number by executing (from within some servlet) the below.

```
request.getServerPort();
```

servlet must apply this stylesheet to a `String` of XML, structured however you see fit. So that we know the structure of your `String` of XML, create in `project4-7.0/webapps/scamazon/dtd/confirm.dtd` a DTD or in `project4-7.0/webapps/scamazon/xsd/confirm.xsd` an XML schema. (You needn't create both.) Also, so that we know the structure of your `PO` element, create in `project4-7.0/webapps/scamazon/dtd/po.dtd` a DTD or in `project4-7.0/webapps/scamazon/xsd/po.xsd` an XML schema. (You needn't create both.) Though your designs must be correct, you needn't validate your XML.

vi. Upon checkout and confirmation thereof, it must be possible for a user to obtain a receipt for his or her order as a PDF, perhaps by clicking a hyperlink. That receipt should contain at least as much information as is required for the XHTML-based confirmation screen. How you implement receipts is up to you, but look to the URL below for guidance on integrating FOP into servlets.

`http://xmlgraphics.apache.org/fop/0.93/servlets.html`

Think carefully about how much information your shopping cart has to remember about each item. Per this document's forthcoming discussion of the warehouse's `Purchasing` service, your `Cart` servlet need submit to that service, upon the user's request, only the values of items' `key` attributes. After all, recall the structure of `inventory.xml`. However, your shopping cart's going to need to display more than just the values of `key` attributes. It's probably more efficient to tuck a good amount of data away in a `Session` object or, if it's system-wide (*i.e.*, not user-specific) data, in a class variable, than it is to store only the values of `key` attributes in a `Session` object and constantly re-parse `catalog.xml`. In fact, you may find it helpful to define in `Cart.java` an inner class or two, instantiations of which can be tossed into a `Session` object for safekeeping. But, again, such implementation details are up to you.

The (Non-)Fulfillment Partner.

9. (30 points.) Let's make sure that warehouse is able to (pretend to) fulfill those orders that the `Cart` servlet wants to submit in the form of `PO` elements.

Notice that `Purchasing.java` is already designed to apply `project4-7.0/webapps/warehouse/xsl/po-ack.xsl` to whatever `String` of XML is passed into its `processPO(·)` method. Bearing in mind the structure you chose for the `Cart` servlet's dynamically generated `PO` element, augment `po-ack.xsl` so that it transforms a `PO` element into a `PO-ACK` element, whose structure is up to you. However, as you know from this document's discussion of the `Cart` servlet, that `PO-ACK` element had better contain enough information so that the `Cart` servlet can confirm (or deny) a user's order effectively. Of course, `po-ack.xsl` should respect the actual quantities of items in stock, as defined in `project4-7.0/webapps/warehouse/xml/inventory.xml`, ultimately reporting which items are in stock and which are backordered. It is not necessary to update `inventory.xml`

dynamically based on `PO`s processed; you may assume that the quantities specified therein are eternal.

You're welcome to modify `Purchasing.java` as you see fit.

So that we know the structure of your `PO-ACK` element, create in `project4-7.0/webapps/warehouse/dtd/po-ack.dtd` a DTD or in `project4-7.0/webapps/warehouse/xsd/po-ack.xsd` an XML schema. (You needn't create both.) Though your DTD or schema must be correct, you needn't configure your service to validate your XML; nor need you configure your service to validate the `PO` elements that it receives from Scamazon.com's `Cart` servlet.

10. (0 points.) Notice again that the `Purchasing` class makes no reference to its being a web service. Such is the beauty of a SOAP toolkit like Axis: it can transform Java classes into web services "magically"—that is, without your having to write a skeleton.

    For simplicity, this project makes use of a handwritten proxy method with which your `Cart` servlet can access the web service.[24] However, we could have generated a stub by first generating (or handwriting) the `Purchasing` service's WSDL, thereafter processing it with a SOAP toolkit. In fact, visit the URL below to see the WSDL that Axis generates for your `Purchasing` class.

    ```
    http://$ICEBOX.fas.harvard.edu:n/warehouse/services/Purchasing?wsdl
    ```

    Let's generate a Java stub from your `Purchasing` service's WSDL. Go ahead and execute the following command (which, unfortunately, wraps onto a second line).[25,26]

    ```
    WSDL2Java
    "http://$ICEBOX.fas.harvard.edu:n/warehouse/services/Purchasing?wsdl"
    ```

    Check out the stub code that Axis emitted into `./edu/harvard/fas/$ICEBOX/warehouse/services/Purchasing/`!

    Now let's generate a Java skeleton from your `Purchasing` service's WSDL. Go ahead and execute the following command (which also, unfortunately, wraps onto a second line).[27]

    ```
    WSDL2Java --server-side --skeletonDeploy true
    "http://$ICEBOX.fas.harvard.edu:n/warehouse/services/Purchasing?wsdl"
    ```

[24] Trust us, we thought through the logistics of having you generate skeletons and stubs and decided you it wasn't worth the additional complexity; launching your servlets and service within the confines of a single application server would no longer be as simple as executing `` `ant` `` followed by `` `tomcat` ``.
[25] Note that, on `nice.fas.harvard.edu`, `` `WSDL2Java` `` is an alias for
`` `java org.apache.axis.wsdl.WSDL2Java` ``.
[26] You can ignore any log4j warnings.
[27] You can ignore any log4j warnings.

Check out the skeleton code Axis has added to `./edu/harvard/fas/$ICEBOX/warehouse/services/Purchasing/`.

You're welcome to try out this stub and skeleton code. But, if you do, be sure to reconfigure your project to use the provided proxy class prior to submission; do not submit a project that utilizes a dynamically generated stub or skeleton.

For additional details on dynamic generation of stubs and skeletons, follow the link to the User's Guide for Axis at `http://ws.apache.org/axis/`.

~~Scamazon.com~~. Amazon.com

11. (30 points.) It's time to add one last feature to your infrastructure. It turns out that your application's namesake, Amazon.com, offers access to its wares via Amazon Web Services (AWS)! Your final challenge, then, is to incorporate AWS's E-Commerce Service (ECS) 4.0 into Scamazon.com using SOAP! Perhaps you'd like to allow visitors to Scamazon.com to search Amazon.com's books by author, keyword, or title (so that they have somewhere to turn for their purchase once they realize your site's a scam). Or perhaps you'd like to allow visitors to browse Amazon.com's DVDs along with Scamazon.com's own "products." The possibilities are endless! Well, not quite, but it may feel that way when you peruse AWS's documentation.

    In fact, surf on over to the URL below.

    `http://aws.amazon.com/`

    Poke around the site for a bit, particularly the leftmost links entitled "Amazon E-Commerce Service" and "FAQs." Proceed to "Create an Account" in order to obtain a "Access Key ID." (After creating an account, you'll receive an email containing a link; follow that link to see your Access Key ID.) You needn't download anything from Amazon, as we've included all that you need in this project's distribution. But do look over the SDK, particularly its API Reference, available at the URL below.

    `http://docs.amazonwebservices.com/AWSEcommerceService/2005-03-23/`

    Know that Amazon allows you to access ECS not only via SOAP but also via REST (XML over HTTP) and XSLT.

    Also take a peek at ECS's WSDL (for developers within the United States) at the URL below.

    `http://webservices.amazon.com/AWSECommerceService/US/AWSECommerceService.wsdl`

    Unfortunately, it's probably not immediately apparent from these resources alone how to employ the kit. So we'll get you started.

Notice that, in `project4-7.0/src/`, we've provided source code for `com.amazon.soap.AWSECommerceService.*`. We generated this code by executing, within `project4-7.0/src/`, the following command (which, unfortunately, wraps onto three lines).[28]

```
java org.apache.axis.wsdl.WSDL2Java -v -W -p
com.amazon.soap.AWSECommerceService
http://webservices.amazon.com/AWSECommerceService/US/AWSECommerceService.wsdl
```

Throughout this project, Ant has been compiling the 171 source files that compose `com.amazon.soap.AWSECommerceService.*`, storing the resulting bytecodes in `project4-7.0/webapps/scamazon/WEB-INF/classes/`.

It's time to put those bytecodes to use. First, take a look at `project4-7.0/src/cscie259/project4/scamazon/ECS.java` and `project4-7.0/webapps/scamazon/ecs.jsp`. Then, go ahead and paste in your Access Key ID for AWS into the former, per the file's comments, and surf on over to the URL below for a demonstration of ECS.

```
http://$ICEBOX.fas.harvard.edu:n/scamazon/servlet/ecs
```

Needless to say, this servlet and JSP demonstrate ECS by querying Amazon.com for a "Large" set of details on the course's recommended books. Note that this demonstration happens to make use of Apache's Standard Taglib 1.1.2, an implementation of the JSP Standard Tag Library (JSTL), jars for which have been provided in `project4-7.0/webapps/scamazon/WEB-INF/lib/`. With JSTL are we able to confine this demonstration's logic to a servlet and its aesthetics to a JSP. You're welcome, but not required, to make use of JSTL in your own work; additional detail is available at the URL below.

```
http://java.sun.com/products/jsp/jstl/reference/docs/
```

For the curious, SOAP messages corresponding to this demonstration's request and response can be found in `project4-7.0/webapps/scamazon/xml/cache/`.

How you proceed to incorporate ECS into Scamazon.com is up to you, so long as you interact with it via SOAP, as we have done with our demonstration. (Though we offer this demonstration as a JSP, you might wish to incorporate ECS into one or more of your servlets.) But allow us to advise that simply incorporating into Scamazon.com some strangers' opinions on the course's recommended texts isn't terribly interesting (and certainly not worth 30 points).

Consider this exposure to ECS an opportunity to discover for yourself how to incorporate a real-world web service into code of your own. Unfortunately, ECS's documentation isn't

[28] Know that, after generating this code, we commented out the (unnecessarily generated) explicit constructor in `com/amazon/soap/AWSECommerceService/ItemAttributes.java`, as it has more parameters than Java allows.

terrific, but the API Reference, coupled with the Javadoc we've provided in `project4-7.0/webapps/ROOT/docs/`, should provide sufficient direction. But be sure to look to `project4-7.0/src/cscie259/project4/scamazon/ECS.java` and, optionally, `project4-7.0/webapps/scamazon/ecs.jsp` for hints on usage.

Discussion on the course's listserv of ECS's features and usage is both permissible and encouraged!

README.

12. (10 points.) In `project4-7.0/README.txt`, provide the teaching staff with an overview of your implementation of Scamazon.com and its warehouse.[29] Specifically, summarize how you implemented the project's catalog, shopping cart, and fulfillment partner, highlighting interesting implementation details. Also summarize how you incorporated ECS into your application. Your discussion should be at least three paragraphs and should well prepare the staff for a closer reading of your source code.

Submitting Project 4.

13. (0 points.) If you have not done your work on `nice.fas.harvard.edu`, transfer via SFTP your `project4-7.0/` directory from your local machine to the `cscie259/` directory in your FAS account's home directory, taking care to upload any text files in ASCII mode.

    Particularly if you developed on another machine, ensure that your application builds and executes properly on `nice.fas.harvard.edu`; misconfigured submissions may be ineligible for full credit.

    Once everything's in order, clean up your workspace by executing

    ```
    ant clean
    ```

    from within your `project4-7.0/` directory.[30]

    Next, ensure that the structure of your `project4-7.0/` directory on `nice.fas.harvard.edu` is the following.

[29] The MIME type of this file, so to speak, should remain `text/plain`.
[30] Know that this command deletes not only your bytecodes but also Tomcat's `logs`, runtime directories, and temporary files.

```
project4-7.0/
  conf/
  src/
    com/
      amazon/
        soap/
          AWSECommerceService/
    cscie259/
      project4/
        scamazon/
        warehouse/
  temp/
  webapps/
    ROOT/
      docs/
        com/
          amazon/
            soap/
              AWSECommerceService/
        cscie259/
          project4/
            scamazon/
            warehouse/
      WEB-INF/
    scamazon/
      dtd/
      images/
      WEB-INF/
        lib/
      xml/
        cache/
      xsd/
      xsl/
    warehouse/
      dtd/
      images/
      WEB-INF/
      xml/
      xsd/
      xsl/
```

Moreover, ensure that each directory contains exactly those files required by this specification; do not submit files which are unnecessary for an evaluation of your work, such as outdated source files, copies of original files, unnecessary output files, *etc.*

However, to facilitate your work's evaluation, please do not remove your subscription ID for AWS from your source code.

Finally, submit your work electronically by executing the following command from within your `project4-7.0/` directory.

```
cscie259submit project4
```

Thereafter, follow any on-screen instructions until you receive visual confirmation of your project's successful submission. You may re-submit as many times as you'd like; each resubmission will overwrite any previous submission. But take care not to re-submit after the project's deadline, as only your latest submission's timestamp is retained.

14. (0 points.) On to the final project![31]

[31] Note to self: use fewer footnotes.

Appendix

So long as your project ultimately compiles and executes on `nice.fas.harvard.edu`, you are welcome to develop it on your own computer. We leave it to you to translate the commands in this document to your own operating system's syntax (*e.g.*, forward slashes to backslashes, `$VAR` to `%VAR%`, *etc.*). This appendix explains how to configure your computer like `nice.fas.harvard.edu`.

You are encouraged to discuss and troubleshoot these steps with classmates via the listserv. Some of these steps also appeared in the specifications for Project 1 (①), Project 2 (②), and/or Project 3 (③).

JDK 5.0 Update 11

① Download JDK 5.0 Update 11 via the course's website and install it.[32]
① Define an environment variable called `JAVA_HOME` whose value is the full path to the JDK's directory.
① Prepend `"$JAVA_HOME/bin"` to your `PATH`.
① Define an environment variable called `JAVA_COMPILER` whose value is `"NONE"`.
③ Create a directory in `$JAVA_HOME/jre/lib/` called `endorsed`.
① Prepend `"./build"` to your `CLASSPATH`.
① Prepend `"."` to your `CLASSPATH`.

Xalan 2.7.0

① Download Xalan 2.7.0 (*i.e.*, `xalan-j_2_7_0-bin.{tar.gz,zip}`) via the course's website and extract it to its own directory.
① Define an environment variable called `XALAN_HOME` whose value is the full path to Xalan's directory.
③ Remove `"$XALAN_HOME/serializer.jar"` from your `CLASSPATH`.
③ Place a copy of `$XALAN_HOME/serializer.jar` in `$JAVA_HOME/jre/lib/endorsed/`.
③ Remove `"$XALAN_HOME/xalan.jar"` from your `CLASSPATH`.
③ Place a copy of `$XALAN_HOME/xalan.jar` in `$JAVA_HOME/jre/lib/endorsed/`.
③ Remove `"$XALAN_HOME/xercesImpl.jar"` from your `CLASSPATH`.
③ Place a copy of `$XALAN_HOME/xercesImpl.jar` in `$JAVA_HOME/jre/lib/endorsed/`.
③ Remove `"$XALAN_HOME/xml-apis.jar"` from your `CLASSPATH`.
③ Place a copy of `$XALAN_HOME/xml-apis.jar` in `$JAVA_HOME/jre/lib/endorsed/`.

[32] Sun's installer for Windows installs both a JDK and a JRE, the latter of which effectively has higher priority in the operating system's `PATH`; Windows users may wish to uninstall this JRE via Add or Remove Programs in their Control Panel.

② Define an alias called `EnvironmentCheck` whose value is `"java org.apache.xalan.xslt.EnvironmentCheck"`.[33]
② Define an alias called `xalan` whose value is `"java org.apache.xalan.xslt.Process"`.[34]

Ant 1.7.0

① Download Ant 1.7.0 via the course's website and extract it to its own directory.
① Define an environment variable called `ANT_HOME` whose value is the full path to Ant's directory.
① Add `"$ANT_HOME/lib/ant.jar"` to your `CLASSPATH`.
① Add `"$ANT_HOME/lib/ant-launcher.jar"` to your `CLASSPATH`.
① Append `"$ANT_HOME/bin"` to your `PATH`.

Tomcat 6.0.10

③ Download Tomcat 6.0.10 (*i.e.*, `apache-tomcat-6.0.10.{tar.gz,zip}`) via the course's website and extract it to its own directory.
③ Define an environment variable called `CATALINA_HOME` whose value is the full path to Tomcat's directory.
③ Define an environment variable called `CATALINA_BASE` whose value is `"."`.
③ Add `"$CATALINA_HOME/bin/bootstrap.jar"` to your `CLASSPATH`.
③ Add `"$CATALINA_HOME/lib/servlet-api.jar"` to your `CLASSPATH`.
③ Create a directory in `$CATALINA_HOME/` called `endorsed`.
③ Place a copy of `$XALAN_HOME/serializer.jar` in `$CATALINA_HOME/endorsed/`.
③ Place a copy of `$XALAN_HOME/xalan.jar` in `$CATALINA_HOME/endorsed/`.
③ Place a copy of `$XALAN_HOME/xercesImpl.jar` in `$CATALINA_HOME/endorsed/`.
③ Place a copy of `$XALAN_HOME/xml-apis.jar` in `$CATALINA_HOME/endorsed/`.
③ Define an alias called `tomcat` whose value is `"$CATALINA_HOME/bin/catalina.sh run"`.[35]

---

[33] On Windows, you can create a batch file (in any directory in your `PATH`) called `ENVIRONMENTCHECK.BAT` containing the following two lines.

```
@echo off
java org.apache.xalan.xslt.EnvironmentCheck
```

[34] On Windows, you can create a batch file (in any directory in your `PATH`) called `XALAN.BAT` containing the following seven lines.

```
@echo off
:rep
shift
set args=%args% %0
if not !%1==! goto rep
java org.apache.xalan.xslt.Process %args%
set args=
```

[35] On Windows, you can create a batch file (in any directory in your `PATH`) called `TOMCAT.BAT` containing the following two lines.

- Define an environment variable called `D_LOCAL_HOST` whose value is `"-Dlocal.host=localhost"` or `"-Dlocal.host=127.0.0.1"`.
- Define an environment variable called `D_VALIDATION_OFF` whose value is `" "` (*i.e.*, a single space).
- Define an environment variable called `D_VALIDATION_DTD` whose value is `"-Denable.validation=dtd"`.
- Define an environment variable called `D_VALIDATION_XSD` whose value is `"-Denable.validation=xsd"`.
- Define an environment variable called `CATALINA_OPTS` whose value is `"$D_LOCAL_HOST $D_VALIDATION_OFF"`.

FOP 0.93

② Download FOP 0.93 (*i.e.*, `fop-0.93-bin-jdk1.4.{tar.gz,zip}`) via the course's website and extract it to its own directory.
② Define an environment variable called `FOP_HOME` whose value is the full path to FOP's directory.
② Add `"$FOP_HOME"` to your `PATH`.
- Add `"$FOP_HOME/build/fop.jar"` to your `CLASSPATH`.
- Add `"$FOP_HOME/lib/avalon-framework-4.2.0.jar"` to your `CLASSPATH`.
- Add `"$FOP_HOME/lib/batik-all-1.6.jar"` to your `CLASSPATH`.
- Add `"$FOP_HOME/lib/commons-io-1.1.jar"` to your `CLASSPATH`.
- Add `"$FOP_HOME/lib/xmlgraphics-commons-1.1.jar"` to your `CLASSPATH`.
- Place a copy of `$FOP_HOME/build/fop.jar` in `$CATALINA_HOME/lib/`.
- Place a copy of `$FOP_HOME/lib/avalon-framework-4.2.0.jar` in `$CATALINA_HOME/lib/`.
- Place a copy of `$FOP_HOME/lib/batik-all-1.6.jar` in `$CATALINA_HOME/lib/`.
- Place a copy of `$FOP_HOME/lib/commons-io-1.1.jar` in `$CATALINA_HOME/lib/`.
- Place a copy of `$FOP_HOME/lib/xmlgraphics-commons-1.1.jar` in `$CATALINA_HOME/lib/`.

Axis 1.4

- Download Axis 1.4 (*i.e.*, `axis-bin-1_4.{tar.gz,zip}`) via the course's website and extract it to its own directory.
- Define an environment variable called `AXIS_HOME` whose value is the full path to Axis's directory.
- Add `"$AXIS_HOME/lib/axis.jar"` to your `CLASSPATH`.
- Add `"$AXIS_HOME/lib/commons-discovery-0.2.jar"` to your `CLASSPATH`.
- Add `"$AXIS_HOME/lib/commons-logging-1.0.4.jar"` to your `CLASSPATH`.
- Add `"$AXIS_HOME/lib/jaxrpc.jar"` to your `CLASSPATH`.

```
@echo off
"%CATALINA_HOME%"\BIN\CATALINA.BAT run
```

- Add "$AXIS_HOME/lib/log4j-1.2.8.jar" to your CLASSPATH.
- Add "$AXIS_HOME/lib/saaj.jar" to your CLASSPATH.
- Add "$AXIS_HOME/lib/wsdl4j-1.5.1.jar" to your CLASSPATH.
- Place a copy of $AXIS_HOME/lib/axis.jar in $CATALINA_HOME/lib/.
- Place a copy of $AXIS_HOME/lib/commons-discovery-0.2.jar in $CATALINA_HOME/lib/.
- Place a copy of $AXIS_HOME/lib/commons-logging-1.0.4.jar in $CATALINA_HOME/lib/.
- Place a copy of $AXIS_HOME/lib/jaxrpc.jar in $CATALINA_HOME/lib/.
- Place a copy of $AXIS_HOME/lib/log4j-1.2.8.jar in $CATALINA_HOME/lib/.
- Place a copy of $AXIS_HOME/lib/saaj.jar in $CATALINA_HOME/lib/.

JavaMail 1.4

- Download JavaMail 1.4 (*i.e.*, javamail-1_4.zip) via the course's website and extract it to its own directory.
- Define an environment variable called JAVAMAIL_HOME whose value is the full path to JavaMail's directory.
- Add "$JAVAMAIL_HOME/mail.jar" to your CLASSPATH.
- Place a copy of $JAVAMAIL_HOME/mail.jar in $CATALINA_HOME/lib/.

XML Security 1.4

- Download XML Security 1.4 (*i.e.*, xml-security-bin-1_4_0.zip) via the course's website and extract it to its own directory.
- Define an environment variable called XMLSEC_HOME whose value is the full path to XML Security's directory.
- Add "$XMLSEC_HOME/libs/xmlsec-1.4.0.jar" to your CLASSPATH.
- Place a copy of $XMLSEC_HOME/libs/xmlsec-1.4.0.jar in $CATALINA_HOME/lib/.

JavaBeans Activation Framework 1.1

- Download JavaBeans Activation Framework (JAF) 1.1 (*i.e.*, jaf-1_1-fr.zip) via the course's website and extract it to its own directory.
- Define an environment variable called JAF_HOME whose value is the full path to JAF's directory.
- Add "$JAF_HOME/activation.jar" to your CLASSPATH.
- Place a copy of $JAF_HOME/activation.jar in $CATALINA_HOME/lib/.