Project 1

My First XML Parser

137 Points

v. 8.0

due by 2:35 P.M. ET on Monday, 15 October 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 recommend that you begin answering this document's questions immediately thereafter.

Immediate completion of this project's first question is of particular import,
lest there be problems with your registration in this course.

As per the course's syllabus, extensions on projects will not be granted, except in cases of emergency. Technical difficulties, mind you, will not constitute emergencies.

Goals.

The goals of this project are to:

- Provide a foundation for all future XML work, for and beyond this course.
- Challenge you to build an XML parser.
- Demonstrate the importance of grammars to parsing.
- Give you hands-on experience with JAXP 1.3's APIs for SAX 2.0.2 and DOM Level 3.
- Introduce you to an industry-standard parser, Apache's Xerces-J 2.7.1.

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.

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?
Style	Is your code rigorously documented with inline comments and Javadoc doc comments and tags?[1] Is it clear from your comments alone how your code operates? Is your code pretty-printed? Are your data members, methods, and variables aptly named?

Getting Started.

1. (0 points.) As per the course's syllabus, obtain an FAS (Faculty of Arts and Sciences) Computer Account, if you haven't one already, by visiting the URL below and following the on-screen instructions.

   `https://www.fas.harvard.edu/computing/utilities/activate-pin/`

   Not only will this account will provide you with access to FAS's computer facilities, it will also provide you with an email address of the form `username@fas.harvard.edu`, where `username` is your FAS username. Additionally, this account will allow you to access via SFTP and SSH `nice.fas.harvard.edu`, FAS's New Linux Computing Environment, use of which will be required by this and future projects. SFTP and SSH clients are available for various platforms via the course's website.

2. (0 points.) Configure your FAS account for use in this course by connecting via SSH to `nice.fas.harvard.edu` and executing

   ```
   ~cscie259/pub/bin/cscie259setup
   ```

   at the prompt. You will then need to log out for the changes to take effect. Upon logging back in, you can confirm the changes' effect by executing

   ```
   cscie259check
   ```

[1] With regard to Javadoc, we expect descriptions for all methods and appropriate use of the `@param`, `@return`, and `@throws` block tags.

at the prompt. If said command is "not found," you failed to follow these directions correctly. `;-)` For assistance with the process, simply contact the course's staff.

Once your account is configured for CSCI E-259, proceed to execute

```
cscie259web
```

at the prompt, thereafter following the on-screen instructions for selecting a password for Web-accessible directories.

You should not need to run `cscie259setup` or `cscie259web` more than once each this term.

If you find that this course's configuration of your account conflicts with a configuration required by another course you're taking this semester, please contact a member of the course's staff.

3. (0 points.) Finally, per the course's syllabus, subscribe to course's Listserv by following the appropriate link of the course's website.

4. (0 points.) SSH to `nice.fas.harvard.edu` and execute the following command.

```
mkdir ~/cscie259/
```

All of your work, no matter where you develop it, will ultimately need to reside in this directory for submission.

If you intend to do your work on `nice.fas.harvard.edu`, proceed to execute the following sequence of commands as well.[2]

```
cp –r ~cscie259/pub/distribution/projects/project1-8.0/ ~/cscie259/
cd ~/cscie259/
ls
```

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

```
project1-8.0/
```

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 `project1-8.0/` directory from the course's website to your local machine. Or, of course, can you transfer the directory itself via SFTP to your local machine. Refer to this document's Appendix for further directions.

[2] Beware the distinction between `~cscie259` and `~/cscie259`.

Notice, now, that your `project1-8.0/` directory is structured as follows.

```
project1-8.0/
  docs/
    cscie259/
      project1/
        mf/
  samples/
    xml/
  src/
    cscie259/
      project1/
        mf/
```

Not surprisingly, many of these directories contain one or more files. For instance, `project1-8.0/` contains `build.xml`, a configuration file for Ant, the build tool you will use to build Project 1's components.[3] (In fact, during development, this tool will automatically generate a `build/` directory in your `project1-8.0/` directory containing your `.class` files.) In `project1-8.0/docs/`, you will find Javadoc for this project's distribution code. (Incidentally, as discussed later in this document, anytime you modify or add to that code for this project, you can easily update those Javadoc with Ant.) In `project1-8.0/samples/xml/`, you will find a finite supply of XML files that you're welcome to use during any testing of your code. Finally, in `project1-8.0/src/`, you will find this project's distribution code, which has been divided into two packages: `cscie259.project1` and `cscie259.project1.mf`.

Ensure that everything is in order by executing the following command from within your `project1-8.0/` directory.

```
ant compile
```

(Alternatively, you can execute `ant` without any arguments.) By default, `ant` will compile `cscie259.project1.*` and `cscie259.project1.mf.*`, storing the resulting bytecodes in `project1-8.0/build/`. Ensure that compilation was successful by executing

```
java cscie259.project1.AttributeConverter
```

followed by

```
java cscie259.project1.mf.Tester
```

from within your `project1-8.0/` directory. Both commands should yield usage information for the programs; any other results, particularly on machines other than `nice.fas.harvard.edu`, suggest a problem with your setup. Contact a member of the course's staff if you are unable to solve the problem. Compiling the code we wrote, however, should be the easiest part of this project!

[3] You are welcome to add new targets to this file, particularly for testing.

5. (0 points.) Consider utilizing version-control software (*e.g.*, CVS, Microsoft Visual SourceSafe, RCS, Subversion, *etc.*) for this project and all future projects. Neither dogs eating source code nor humans accidentally deleting work constitutes an emergency, so far as extensions are concerned.

Quickies.

Please type your answers to questions 6 through 10 in a file called `questions.html`, `questions.rtf`, `questions.pdf`, or `questions.txt` in your `project1-8.0/` directory. Note that questions 7 and 9 require that you sketch a DOM; ASCII art suffices.

6. (4 points.) Consider the XML fragment below, an excerpt from CSCI E-259's database in 1636.

```
<student type="undergrad">
    <name>John Harvard</name>
    <email>john@harvard.edu</email>
    <phone/>
    <grades>
        <project number="1">137</project>
    </grades>
</student>
```

Unfortunately, phones didn't exist in 1636, so the course hadn't a phone number on file for John. Oddly enough, though, it did have an email address. Anyhow, disregarding ignorable (*i.e.*, meaningless) whitespace, list the SAX events that would be fired were this fragment to be parsed by a SAX parser.

7. (4 points.) Consider now a larger excerpt from CSCI E-259's original database, the root element of which is `students`.

```
<?xml version="1.0"?>
<!-- CSCI E-259 students -->
<students>
    <student type="undergrad">
        <name>John Harvard</name>
        <email>john@harvard.edu</email>
        <phone/>
        <grades>
            <project number="1">137</project>
        </grades>
    </student>
</students>
```

Sketch the DOM represented by this XML, again disregarding ignorable whitespace.

8. (5 points.) For each of the following scenarios, specify whether SAX or DOM is the more appropriate API. Be sure to justify your choice with one or more sentences.

   i. You have been given the task of writing an application that needs to do complex calculations on XML input. These calculations require random access throughout the XML document in order to perform the calculation.

   ii. You are updating an internal database of stock information based on a XML feed of quote ticks from a stock exchange.

   iii. You are reading in a large XML file, performing a calculation on each element, and writing out the result.

   iv. You are reading in a large XML file, sorting the data, and writing out the result.

   v. You need to extract the content of one single element in an XML document quickly each time it is received, and do not need access to the remainder of the document.

9. (8 points.) Consider the list of SAX events implied by the pseudocode below.

```
startDocument();
startElement("lectures");
startElement("lecture", {("number", "1"), ("available", "1")});
startElement("date");
characters("Monday, 17 September 2007");
endElement("date");
startElement("title");
characters("Lecture 1");
endElement("title");
startElement("subtitle");
characters("Introduction");
endElement("subtitle");
startElement("handouts");
startElement("handout");
startElement("name");
characters("Slides");
endElement("name");
startElement("formats");
startElement("format", {("available", "0"),
                        ("type", "PDF"), ("filename", "lecture1.pdf")});
endElement("format");
endElement("formats");
endElement("handout");
endElement("handouts");
endElement("lecture");
endElement("lectures");
endDocument();
```

What XML fragment generated these SAX events upon being parsed? Although these events excluded ignorable (*i.e.*, meaningless) whitespace, do pretty-print your answer.

Then, assuming `lectures` is the root element of some document, sketch the DOM suggested by these SAX events.

10. (6 points.) Give a grammar for a metalanguage (reminiscent of much simplified XML) that only supports elements whose names must be entirely alphabetical and whose content may be zero or more other elements, alphanumeric content, and/or whitespace. The metalanguage's tags, however, cannot contain any whitespace.

    Below is just one of the infinitely many strings that might be generated by such a grammar.

```
<foo>
    <bar>
        <baz/>
        <qux>quux</qux>
    </bar>
</foo>
```

mf.XMLParser.

Please type your answers to questions 12, 14, 16, and 18 in a file called `questions.html`, `questions.rtf`, `questions.pdf`, or `questions.txt` in your `project1-8.0/` directory.

11. (6 points.) Okay, it's time to enhance that simplified XML parser introduced in Lectures 2 and 3!

    Recall that said parser only supported the following grammar, where `Name` could be zero or more characters excluding `'>'`, and `CharData` could be zero or more characters excluding `'<'`.

```
element ::= STag content Etag
content ::= (element | CharData)*
STag    ::= '<' Name '>'
ETag    ::= '<' '/' Name '>'
```

    In other words, the parser did not support attributes or empty elements. And it did not distinguish ignorable whitespace from meaningful whitespace. Moreover, the parser expected the very first character of its input file to be `'<'`; even leading whitespace would produce an error.

    Clearly, that parser had limits. Your job is to make it better.

    First, though, read through the code in `project1-8.0/src/cscie259/project1/mf/`, which you retrieved for question 4. Notice that the files comprise a package called `cscie259.project1.mf`. Also take care to notice which files you MAY and MAY NOT

modify. Again, Javadoc for that code can be found in `project1-8.0/docs/` as well as on the course's website.

Next, go ahead and compile `cscie259.project1.mf.*`, without making any modifications to the code, by executing the following from within your `project1-8.0/` directory.

```
ant compile-Tester
```

Recall that a number of sample XML files can be found in `project1-8.0/samples/xml/`. Proceed to test this parser, as is, by executing the first test routine in the staff's `Tester` program on `1.xml` and `2.xml`. To be clear, execute

```
java cscie259.project1.mf.Tester samples/xml/1.xml 1
```

and

```
java cscie259.project1.mf.Tester samples/xml/2.xml 1
```

from within your `project1-8.0/` directory.

The unmodified parser should have no trouble parsing these files. Now try testing the parser with `3.xml`. Uh oh.

Once you understand the unmodified parser's framework, as well as the staff's test routines, proceed to eliminate this limitation: the parser's expectation that its input file's first character will demark the start of an element. In other words, enhance the parser so that it ignores any whitespace that precedes an input file's root element, where whitespace is any character for which `java.lang.Character.isWhitespace` returns `true`. We suggest you confine your modifications to `XMLParser.java`.

12. (1 point.) Explain, in a sentence or two, how you implemented support for whitespace preceding a document's root element, drawing our attention to code you wrote to solve the problem.

13. (30 points.) Okay, now it's time to eliminate some other shortcomings, particularly the parser's lack of support for attributes. Proceed to enhance your parser so that it supports the following grammar, where `Name` is now one or more letters, numbers, hyphens, periods, and underscores; `CharData` can be zero or more characters excluding `'<'`; `AttValue` can be zero or more characters excluding `'<'` and `'"'`; `S` is any character for which `java.lang.Character.isWhitespace` returns `true`; * denotes zero or more occurrences; and $^+$ denotes one or more occurrences.

```
element   ::= STag content Etag
content   ::= (element | CharData)*
STag      ::= '<' Name (S+ Attribute)* S* '>'
Attribute ::= Name Eq '"' AttValue '"'
Eq        ::= S* '=' S*
ETag      ::= '<' '/' Name S* '>'
```

We suggest you focus your attention on `Attributes.java` and `XMLParser.java`.[4] Though, you are welcome to declare and define other classes as you see fit; be sure they belong to the `cscie259.project1.mf` package.

Once you (think you) have implemented support for attributes in your parser, enhance `XMLSerializer.java` so that the first test routine in the staff's `Tester` program can serialize attributes to `System.out` for testing. Your parser should have no trouble with `4.xml`, `5.xml`, or `6.xml` now.

14. (4 points.) Explain, in a short paragraph, how you went about implementing support for attributes, drawing our attention to code you wrote to solve the problem.

15. (30 points.) Needless to say, you've been doing a great job taking XML in, parsing it, and serializing it right back out. But let's keep that content around in memory a bit longer and construct a DOM by way of the SAX events your parser can generate.

    Specifically, proceed to implement the `DOMBuilder` class, with, of course, support for attributes. Inasmuch as this class extends our `DefaultHandler`, it is designed to build a DOM out of SAX events. As suggested by `DOMBuilder.java`, that DOM will be represented by a reference to a `Document` object whose descendents are instances of the `Attr`, `Element`, and `Text` classes. All of these objects, mind you, are also of type `Node`.

    To be clear, in addition to modifying `DOMBuilder.java`, be sure to alter `Attr.java` and `Element.java` as you see fit.[5] You are welcome to declare and define other classes; be sure they belong to the `cscie259.project1.mf` package.

    Once you (think you) have implemented support for DOM building, tweak `DOMWalker.java` so that it understands your implementation of attributes in your DOM and can, therefore, pass to `startElement` a reference to an `Attributes` object containing an element's collection of attributes (that, in your DOM, were represented as `Attr` objects). You can then test your `DOMBuilder` with the second test routine in the staff's `Tester` program.

16. (4 points.) Explain, in a short paragraph, how you went about implementing support for DOM building, drawing our attention to code you wrote to solve the problem.

17. (10 points.) Okay, let's get rid of one last shortcoming in this parser: its failure to understand empty elements (like `<foo/>` or `<foo bar="baz"/>`). Specifically, enhance your parser so that it supports the grammar below, which is identical to that specified in question 13 except for its additional support for empty elements (embodied in its first two rules).

---

[4] Note that `Attr.java` is intended for DOM-related work later in this project. Although you may be inclined to store `Attr` objects within `Attributes` objects, you may wish to respect the independence of the SAX 2.0.2 and DOM Level 3 APIs, however redundant.

[5] Recall that `Attributes.java` is intended for SAX-related work. Although you may be inclined to store `Attributes` objects within `Element` objects, you may wish to respect the independence of the SAX 2.0.2 and DOM Level 3 APIs, however redundant.

```
element      ::= STag content Etag | EmptyElement
EmptyElement ::= '<' Name (S+ Attribute)* S* '/' '>'
content      ::= (element | CharData)*
STag         ::= '<' Name (S+ Attribute)* S* '>'
Attribute    ::= Name Eq '"' AttValue '"'
Eq           ::= S* '=' S*
ETag         ::= '<' '/' Name S* '>'
```

We suggest you confine your modifications to `XMLParser.java`. Recall that an empty element can be printed as a start tag immediately followed by an end tag, with no characters (even whitespace) in between, so it's not necessary to modify `XMLSerializer.java`.

Your parser should now have no trouble with a file like `7.xml` or `8.xml`.

18. (2 points.) Explain, in a sentence or two, how you went about implementing support for empty elements, drawing our attention to code you wrote to solve the problem.

19. (0 points.) Although your parser may have conquered all of the sample files in `project1-8.0/samples/xml/`, it's probably best to test your code now on a number of other input files, perhaps of your own creation, that do and don't conform to the grammar specified in question 17. After all, though your code may pass the tests we've provided to you, it may not pass or handle nicely the tests we haven't provided to you. `;-)`

20. (2 points.) Update the Javadoc in `project1-8.0/docs/` by executing

    ```
    ant javadoc
    ```

    from within your `project1-8.0/` directory.

    If you wish to view your Javadoc via the Web, you may additionally type

    ```
    ant publish-javadoc
    ```

    from within your `project1-8.0/` directory, thereafter visiting

    ```
    http://www.people.fas.harvard.edu/~username/cscie259/javadoc/project1-8.0/
    ```

    where `username` is your FAS username. This URL should prompt you for your FAS username and the password that you selected for question 2.

21. (0 points.) Phew! It's done. Breathe a sigh of relief and take a short break. Consider having a snack too.

My Second XML Parser.

22. (20 points.) It's now time to put aside your first XML parser and pick up your second: Xerces-J 2.7.1, a fully JAXP 1.3-compliant parser.

    It is always a matter of some debate whether data associated with an element should be expressed as attributes of the element or as child elements with text nodes. Normally, to an application processing an XML file, such details are of little relevance. But let us assume for a moment we have an application that requires that the input document have no attributes. Your job, then, is to write a Java program called `AttributeConverter` that takes as input an input XML file and uses JAXP 1.3's SAX API to convert all attributes to child elements and serialize the result to `System.out`.[6]

    For instance, given

    ```
    <foo><bar baz="qux" quux="quuux"/></foo>
    ```

    as input, your program should produce

    ```
    <?xml version="1.0" encoding="UTF-8"?>
    <foo><bar><baz>qux</baz><quux>quuux</quux></bar></foo>
    ```

    or some pretty-printed equivalent as output, although the XML declaration isn't necessary. However, lest you struggle with the complexities of whitespace, your output needn't be pretty-printed.

    Provided in `project1-8.0/src/cscie259/project1/` is `AttributeConverter.java`, a skeleton for your program. While you are free to implement additional classes as you see fit in the `cscie259.project1` package, your program's `main` method must appear in `cscie259.project1.AttributeConverter`. Although you may be tempted to have `AttributeConverter` implement `org.xml.sax.ContentHandler`, you may find it simpler to have `AttributeConverter` extend `org.xml.sax.helpers.DefaultHandler` or `org.xml.sax.helpers.XMLFilterImpl`, both of which already implement `org.xml.sax.ContentHandler`.

    As suggested by our skeleton, usage of your program must be

    ```
    java cscie259.project1.AttributeConverter filename
    ```

    from within your `project1-8.0/` directory, where `filename` is the name (and path to) of the file whose contents are to be converted; no other command-line arguments should be accepted.

    To facilitate the serialization of your results to `System.out`, you are welcome to utilize one of Apache's `org.apache.xml.serialize.XMLSerializer`; documentation for this class can

[6] In other words, your program must utilize `javax.xml.parsers.SAXParser`.

be found in Xerces-J's API, Javadoc for which is available via the course website's Resources area.

Your code should take care to handle any errors gracefully.[7] Needless to say, crashing on certain input is not "graceful."

For simplicity, your program need not preserve an input file's XML declaration, processing instructions, DTDs, entities, or comments. However, if you are interested in having your program do so nonetheless, you may find `org.xml.sax.ext.LexicalHandler` of interest.

Brownie Points.

Please type your answer to question 24 in a file called `questions.html`, `questions.rtf`, `questions.pdf`, or `questions.txt` in your `project1-8.0/` directory.

23. (0 points.) Notice that `XMLSerializer` provides support for pretty-printing. However, the staff's `Tester` program disables that feature. Why? Well, recall that we defined `CharData` to be zero or more characters excluding only `'<'` and `'>'`. Consider the implication of this definition for the parsing of `2.xml`, which, recall, contains the following.

```
<foo>
    <bar>
        baz
    </bar>
</foo>
```

Though it may not be immediately apparent, `foo` contains three children: the first is a text node containing a newline character, followed by four spaces; the second is a `bar` element; the third is a newline character. Meanwhile, `bar` contains one child: a text node containing a newline character, followed by eight spaces, followed by a `'b'`, followed by an `'a'`, followed by a `'z'`, followed by a newline character, followed by four spaces.

Hence, if we proceed to parse this version of `2.xml` with the staff's `Tester` program, with pretty-printing enabled, we obtain the following output.

[7] If you opt to have `AttributeConverter` extend `org.xml.sax.helpers.DefaultHandler` or `org.xml.sax.helpers.XMLFilterImpl`, note that both already implement `org.xml.sax.ErrorHandler`.

```
<foo>


    <bar>

        baz

    </bar>


</foo>
```

Not quite pretty, is it? The ugliness is the result of `XMLSerializer`'s indenting element and text nodes in accordance with their relative location in the document without removing what was probably "ignorable whitespace" in the original document. In other words, odds are, the whitespace in

```
<foo>
    <bar>
        baz
    </bar>
</foo>
```

is not an integral part of `2.xml`'s content. However, your parser is not designed to remove ignorable whitespace. In fact, it shouldn't. Only validating parsers (*i.e.*, parsers that validate their input against a DTD or an XML schema) have the luxury of removing such whitespace.[8]

However, if you are so inclined, in the interests of pretty-printing, brownie points, and, quite possibly, brownies, enhance your parser so that it neither includes in SAX events ignorable whitespace nor fires SAX events for sequences of characters composed entirely of ignorable whitespace. Let us define "ignorable whitespace" to be any contiguous sequence of characters for which `java.lang.Character.isWhitespace` returns `true` that appears (1) immediately following some element's start tag and immediately before another element's start tag, (2) immediately following some element's start tag and immediately before that element's end tag, (3) immediately following some element's start tag and immediately before some non-whitespace character data, (4) immediately following some non-whitespace data and immediately before some element's end tag, (5) immediately following some element's end tag and immediately before another element's start tag, or (6) immediately following some element's end tag and immediately before another element's end tag. Whitespace characters in attributes' values are not ignorable and are to be left intact.

So enhanced, your parser should interpret `2.xml` as containing but one foo element whose sole child is a bar element whose sole child is a text node containing `'b'` followed by `'a'` followed by `'z'`. With pretty-printing disabled, then, the staff's `Tester` program, with `2.xml` as input, should output the following.

[8] We'll learn about DTDs and XML schemas in Lectures 8, 9, and 10.

```
<foo><bar>baz</bar></foo>
```

And, with pretty-printing enabled, the staff's `Tester` program, with `2.xml` as input, should output the below instead.

```
<foo>
    <bar>
        baz
    </bar>
</foo>
```

We suggest you confine your modifications to `XMLParser.java`.

After making your modifications, be sure to update the Javadoc in `project1-8.0/docs/` by again executing

```
ant javadoc
```

from within your `project1-8.0/` directory. Again, if you wish to view your Javadoc via the Web, you may additionally type

```
ant publish-javadoc
```

from within your `project1-8.0/` directory, thereafter visiting

```
http://www.people.fas.harvard.edu/~username/cscie259/javadoc/project1-8.0/
```

where `username` is your FAS username. This URL should prompt you for your FAS username and the password that you selected for question 2.

24. (0 points.) Explain, in a sentence or two, how you went about ignoring ignorable whitespace, drawing our attention to code you wrote to solve the problem.

Submitting Project 1.

25. (0 points.) If you have not done your work on `nice.fas.harvard.edu`, transfer via SFTP your `project1-8.0/` directory from your local machine to the `cscie259/` directory in your FAS account's home directory. Recall that you created that directory for question 4.

    Next, ensure that the structure of your `project1-8.0/` directory on `nice.fas.harvard.edu` is the following.

```
project1-8.0/
  docs/
    cscie259/
      project1/
        mf/
  samples/
    xml/
  src/
    cscie259/
      project1/
        mf/
```

Your `project1-8.0/` directory should contain `build.xml` along with `questions.html`, `questions.rtf`, `questions.pdf`, or `questions.txt`; `project1-8.0/docs/` should contain updated Javadoc; `project1-8.0/samples/xml/` should contain those XML files it originally contained, along with any others you created during testing; and `project1-8.0/src/` should contain the Java source it originally contained, along with any additional Java source you wrote.

Just to be safe, particularly if you developed offsite, build all of your source one last time by executing the following from within your `project1-8.0/` directory on `nice.fas.harvard.edu`. (Be sure you've updated `build.xml` to handle any files of your own creation, if necessary.)

```
ant compile
```

Both `cscie259.project1.AttributeConverter` and `cscie259.project1.mf.Tester` should be executable from that same directory.
Once everything's in order, clean up your workspace by executing

```
ant clean
```

from within your `project1-8.0/` directory.

Lastly, submit your work electronically by executing the following command from within your `project1-8.0/` directory.

```
cscie259submit project1
```

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 re-submission 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.

26. (1 point.) Congrats, you're done! You deserve a point for that!

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.

JDK 5.0

- On a PC, download Sun's JDK 5.0 Update 12 via the course's website and install it.[9] On a Mac, download Apple's J2SE 5.0 Release 4 via the course's website and install it.
- Define an environment variable called `JAVA_HOME` whose value is the full path to the JDK's directory.
- Prepend `"$JAVA_HOME/bin"` to your `PATH`.
- Define an environment variable called `JAVA_COMPILER` whose value is `"NONE"`.
- Create a directory in `$JAVA_HOME/jre/lib/` called `endorsed`.
- Prepend `"./build"` to your `CLASSPATH`.
- Prepend `"."` to your `CLASSPATH`.

Xalan 2.7.0

- 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.[10]
- Define an environment variable called `XALAN_HOME` whose value is the full path to Xalan's directory.
- Place a copy of `$XALAN_HOME/serializer.jar` in `$JAVA_HOME/jre/lib/endorsed/`.
- Place a copy of `$XALAN_HOME/xalan.jar` in `$JAVA_HOME/jre/lib/endorsed/`.
- Place a copy of `$XALAN_HOME/xercesImpl.jar` in `$JAVA_HOME/jre/lib/endorsed/`.
- Place a copy of `$XALAN_HOME/xml-apis.jar` in `$JAVA_HOME/jre/lib/endorsed/`.
- Define an alias called `EnvironmentCheck` whose value is `"java org.apache.xalan.xslt.EnvironmentCheck"`.[11]
- Define an alias called `xalan` whose value is `"java org.apache.xalan.xslt.Process"`.[12]

---

[9] 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.

[10] You actually don't need Xalan itself for this project, but it ships with Xerces 2.7.1, which you do need. Since you will need Xalan for Projects 2, 3, and 4, though, might as well set up everything now to save time!

[11] 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
```

Ant 1.7.0

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

[12] 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=
```