

# A3: Web Services Tutorial (Q2, Q3, and Q4)

August 1, 2007

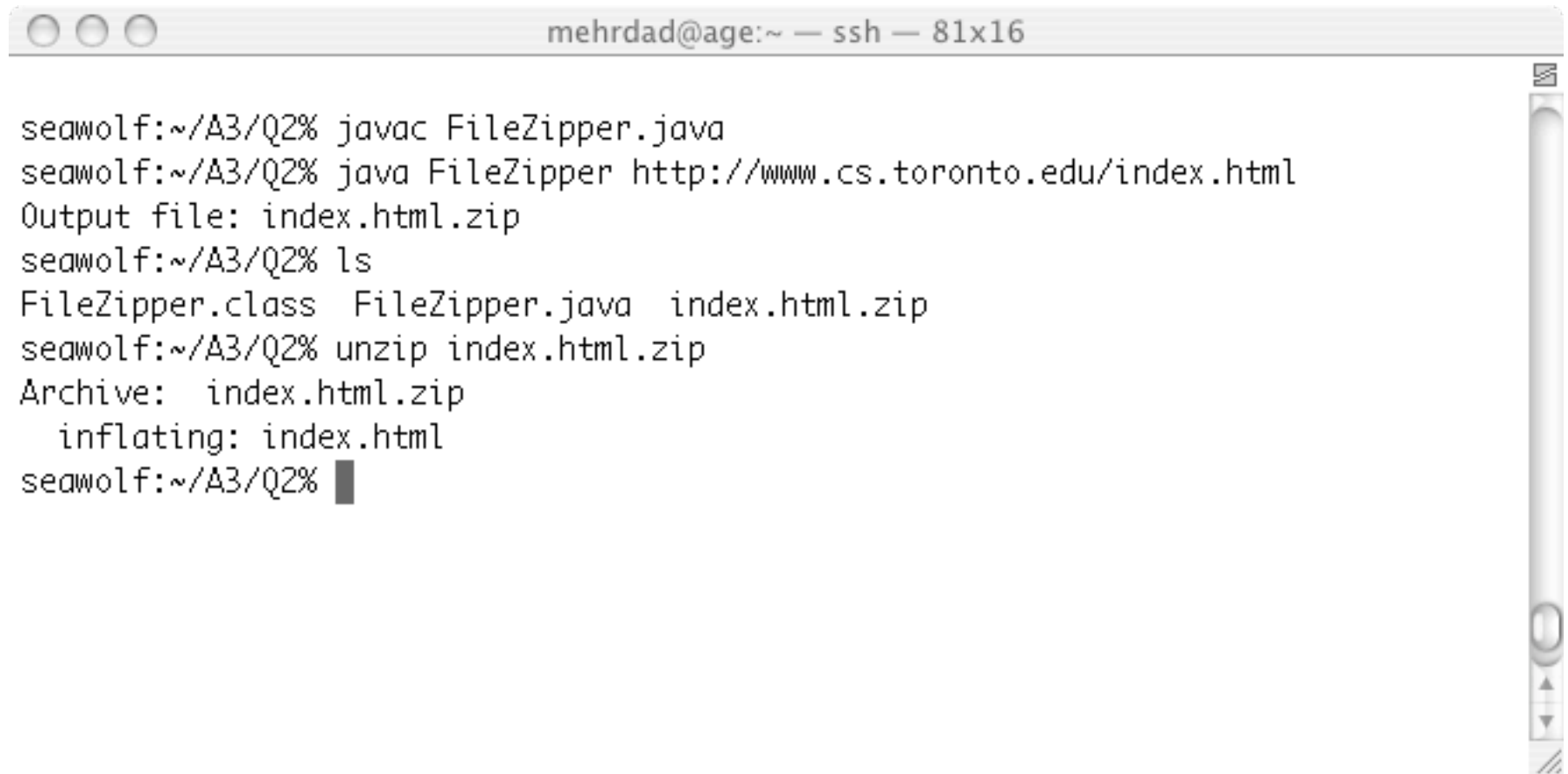
# First thing to do

- Go through the step-by-step guide provided at <http://www.cdf.toronto.edu/~radu/csc309/guide/axis.html>
- This tells you how to install and run a simple calculator web service.

# Question 2

- This is easy to do. You need to use the `java.util.zip` package to develop a Zip utility.
- Assume that the input URL is a reference to a single file (not a directory or anything else)
- Method you need to write:
  - **`String getDocument(String urlString)`**
  - `urlString`: url to the desired file;
  - `output`: name of the zipped file. The actual (zipped) file must be stored locally on the harddisk.

# Question 2 (example)

A screenshot of a terminal window with a title bar that reads "mehrddad@age:~ — ssh — 81x16". The terminal content shows a sequence of commands and their outputs: "javac FileZipper.java", "java FileZipper http://www.cs.toronto.edu/index.html" (outputting "index.html.zip"), "ls" (outputting "FileZipper.class FileZipper.java index.html.zip"), and "unzip index.html.zip" (outputting "Archive: index.html.zip" and "inflating: index.html"). The prompt "seawolf:~/A3/Q2%" is visible at the start of each line, and a cursor is shown at the end of the last line.

```
mehrddad@age:~ — ssh — 81x16
seawolf:~/A3/Q2% javac FileZipper.java
seawolf:~/A3/Q2% java FileZipper http://www.cs.toronto.edu/index.html
Output file: index.html.zip
seawolf:~/A3/Q2% ls
FileZipper.class FileZipper.java index.html.zip
seawolf:~/A3/Q2% unzip index.html.zip
Archive: index.html.zip
  inflating: index.html
seawolf:~/A3/Q2% █
```

# Classpath for Q's 3, 4

- Make sure that the following libraries are on your classpath:
  - <tomcat-home>/lib/axis.jar
  - <tomcat-home>/lib/saaj.jar
  - <tomcat-home>/lib/commons-discovery-0.2.jar
  - <tomcat-home>/lib/commons-logging-1.0.4.jar
  - <tomcat-home>/lib/jaxrpc.jar
  - <tomcat-home>/lib/log4j-1.2.8.jar
  - <tomcat-home>/lib/wsdl4j-1.5.1.jar
  - **<tomcat-home>/lib/activation.jar**
  - **<tomcat-home>/lib/mail.jar**
    - **(you must download mail.jar separately)**
- The calculator example didn't use attachments and hence didn't need mail.jar.
- mail.jar depends on activation.jar!
- Why mail? You need to work with MIME attachments in your assignment!

# Question 3

- **Generating the wsdl file and the client stub code**
  - All you need to have got right at this point is the signature of the getDocument method. What actually goes in the method is unimportant for A3.
  - **Part (a):** `java org.apache.axis.wsdl.Java2WSDL -o filezipper.wsdl -l "http://localhost:yourport/axis/services/FileZipper" -n "urn:FileZipper" FileZipper`
    - The above generates the wsdl file for you!
  - **Part (b):** `java org.apache.axis.wsdl.WSDL2Java filezipper.wsdl`
    - The above generates the client stub code for you!

# Question 3 (cnt'd)

- **Changing the stub code:**

- In the FileZipper\_pkg package, you should have the following files:
  - FileZipper.java, FileZipperServiceLocator.java, FileZipperService.java, FileZipperSoapBindingStub.java
- For convenience, add the following declaration to FileZipper.java:
  - `public java.lang.Object[] getAttachments();`
  - This exposes the getAttachments method in org.apache.axis.client.Stub and helps us avoid unnecessary type-casts later in Q4.

# Question 3 (cnt'd)

- **Avoid hard-coded ports!**

- Make FileZipperServiceLocator.java read the FileZipper\_address variable from a property file.
- The marker should need to set only one property to get your work up and running.

# Question 4

- **Writing the client side:** Very similar to the CalcClient code. What is different is that you need to also read the attachment supplied by the server
- **Writing the server side:** Need to update FileZipper so that it sends the resulting zipped file as an attachment.

# Question 4

- **Extracting attachments (client side):**

```
import org.apache.axis.attachments.AttachmentPart;
...
FileZipperServiceLocator locator = new FileZipperServiceLocator();
FileZipper service = locator.getFileZipper();

// the url below is just an example. Your code should read the url from args[0]
String outputFileName = service.getDocument("http://www.cs.toronto.edu/dcs/index.html");

Object[] attachments = service.getAttachments();

if (attachments.length < 1) {
    System.out.println("No attachments were found in the response.");
    return;
}

// We're interested in the first attachment only
AttachmentPart result = (AttachmentPart) attachments[0];

//Get an input stream over the result
InputStream in = (InputStream) result.getContent();

// Now dump the above input stream to a file. The name of this file should be set to
the outputFileName variable.
// It's up to you to write the code for this part!
```

# Question 4

- **Sending attachments (server side)**

- Now it's time to update the FileZipper class you wrote in Q2.

```
import javax.xml.soap.AttachmentPart;
import org.apache.axis.MessageContext;
import org.apache.axis.Message;
import org.apache.axis.MessageContext;
...
// Do the same things as in Q2.
...
//Then, create an attachment to be sent along with the response. This is done as follows:
// 1) get the soap message context
MessageContext msgContext= MessageContext.getCurrentContext();
// 2) get the response object
Message response = msgContext.getResponseMessage();
// 3) create an attachment and set its mime type
AttachmentPart a = response.createAttachmentPart();
a.setContent(new FileInputStream(zippedFileNameOnServer), "application/zip");
response.addAttachmentPart(a);

// We can remove the file from the server if we don't need it anymore.
File toDelete = new File(zippedFileNameOnServer);
toDelete.delete();

// Finally, return from the getDocument method. The attachment will be automatically sent.
return outFileName;
```

# Deploying your web service

- **This is exactly as described at**  
<http://www.cdf.toronto.edu/~radu/csc309/guide/axis.html>
- Grab a copy of deploy.wsdd for the Calculator example
- Adapt this deployment descriptor to your FileZipper service
  - This is really easy!
- Make sure tomcat is up and running.
- Run the following command to deploy your service:
  - `java org.apache.axis.client.AdminClient -pyourport deploy.wsdd`
- Now, you should be able to see your deployed service when you list the web services in your axis webapp.

# Concurrency

- What if two people simultaneously request the same URL to be zipped?
- Or the URLs are different but the files to be zipped have the same names?
- **You need to deal with concurrency in your assignment and explain your approach in your readme file.**

# What to hand in for Qs 2, 3, and 4.

- These must be bundled as a gzipped tarball (see the assignment handout).
- Question 2: A single java file named FileZipper.java. Your class must provide a main method so that the marker can easily test your implementation.
- Question 3: 1) The generated wsdl file, 2) your (updated) stub code.
- **see next page**

# What to hand in for Qs 2, 3, and 4.

- Question 4: 1) A file named ZipClient.java that implements the client. This should work with the client stub code in Q3. 2) The updated FileZipper class (that responds with attachments). 3) Your deployment descriptor (deploy.wsdd).
- Organize the files (and directories) for these three questions under Q2, Q3, and Q4.
- It is OK to use helper classes to provide better separation of concerns, but this is not required.