

# Waukesha County Technical Institute

## 152-198 Distributed Java

### Class 2 Plan and Assignments

#### Discussion Activities:

- **Due Today:**
  1. Be prepared to discuss the reading assignments from the JEE 7 Tutorial reading assignment. Quiz #1 today.
  2. Be prepared to sign a document indicating that you read and understand the course syllabus.
  3. You should have chosen a research topic from the reading assignment. Be prepared to give that to your instructor.
  4. Prepare to demo your HTML resume. Is it on GitHub? It should be.
  5. Miss any of the work due today? If so, you didn't read the Class Plan for class #1. Remember all class plans contain homework requirements. But you must read those plans!
- **Q&A**
  - Issues with your software?
  - Issues with the reading assignments?
  - Other?
- **Plugins to add to your Netbeans installation**
  - You add plugins by going to the “Tools” menu in Netbeans and selecting “Plugins”. This is also where you check for any updates that are available. For plugins, go to the “Available Plugins” tab and check the following:
    - gitignore.io
    - Show current Git branch in statusbar
    - Serial version UID generator
- **Fundamentals of Data Flow in a Web App**
  - “Request – Response” data flow
    - All web apps, regardless of language or framework, work the same way under the hood.
    - Every time you enter a URL in your browser or submit a web form, or connect to Facebook using a native app on your smartphone, you are making a “**request**” from the client (browser or app) to a web server for some data. A request may or may not provide data of its own (e.g., form data).
    - In a request the client is asking for a web page or server-side program. The identity of this page or program may or may not be visible to you.

- The server will attempt to find the request page or program and return an error code if not found or a success code if found. If found, the server will also process server-side code if necessary and return data. In a traditional web app that data will usually be all the HTML, CSS and JavaScript needed to display a web page to the client (your browser). However, in a native client app like your Facebook app the server will only return data to the client. Either way, this is called the “**response**”.
- In Java, as in most web languages and frameworks, **requests and responses are represented as objects that the server creates on its own**. These objects can then be used by programmers to perform various tasks.
- When a client makes a request (web browser or native app) some kind of URL (Universal Resource Locator) is used to identify the server-side resource needed. Example:

<http://www.wctc.edu/about-wctc/calendar.php>

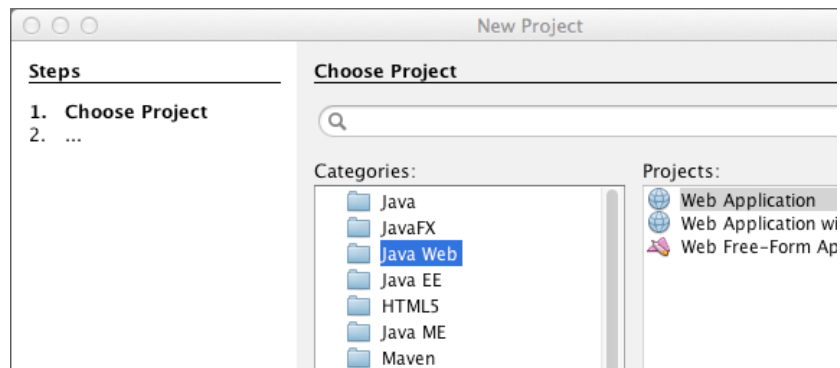
In this example “www.wctc.edu” is the address of the server, and “about-wctc” is a directory on the server, and “calendar.php” is a web page file in that directory stored on the server.

However, if the client was a native app, like a Facebook app on your smartphone, the URL might look very different because you would not be asking for a web page, but rather sending/receiving data. The point is, these are all requests.

- However, not all requests are “HTTP” requests. HTTP (HyperText Transport Protocol) is a communication protocol (a set of rules) for communicating between web browsers and web servers. But it is not the only protocol used on the web. Others, for example, FTP (File transport Protocol) are also used for other purposes (in this case transferring files). So a URL is composed of a protocol, plus a server address, plus other miscellaneous text identifying a server-side resource.
- The HTTP protocol uses special **request methods**, which are NOT Java methods, but rather VERBS that the server understands. Two of the most common HTTP methods are GET and POST.
- A GET request is the content of a URL and is limited to approximately 2000 characters (browser dependent). It is visible in the web browser address box.
- A POST request is the content of a form and has no limitations beyond available memory. It is not visible in the web browser, so it is slightly more secure. However, it is visible, in plain text form, to a sniffer program used by hackers. Later we will deal with this via encryption.
- In addition the request contains **header information** that describes important information about the request. This information can be

accessed and appended by program code on the server. We will see how you can monitor one of these things using Chrome dev tools or by using the Firebug plugin in Firefox.

- A **“response”** is the end goal of the server – to provide data to the client in response to the request. This data may be the code for an entire web page, or it may just be data needed by a web page or native app. The server creates a response object, which programmers can use to perform certain tasks. And the response also contains header information.
- Think of the **header** as information that describes the request or response and the payload of data being provided. Think of the data provided as the **“payload”**
- Demonstration of request-response and header and payload information using Chrome dev tools and Firefox firebug plugin.
- **Review Fundamentals of HTML and Creating Web Projects in Netbeans**
  - ✓ There are 4 types of web projects that you can create in Netbeans. In the New Project Dialog box, under “Categories” you will see:



Of these choices you can:

1. Create a “Java Web” Application
  2. Create a “Java EE” Application
  3. Create a “HTML5” Application, or
  4. Create a “Maven” Web Application
- **For now, we’ll be using the choice #1.** Soon, however, we will be switching to “Maven” and will use that for the rest of the semester. More info to come later.
  - **Your first web application:**

We’ll create a sample project named “FirstWebApp” for this demo in class today:

    - Create a new project (using choice #1 above), and give it the name above without any spaces or other punctuation. Click through the project wizard and make sure you select Glassfish for your server (4.1 or 4.1.1) and “EE 7” for version of the Enterprise Edition of Java. The version is important

because it defines the kind of web app servers you can use and the language features available. If we were using an older server, for example Tomcat 6, we would want EE version 5, because Tomcat 6 does not support EE 7 features.

- After creating a new project in Netbeans, immediately Git enable it by selecting “Team > Git > Initialize Repository”. Then from the File menu select “generate gitignore file” and search for the “netbeans” configuration. It’s important that you always do these two steps immediately after creating a new project. Otherwise it gitignore may not ignore some things. The purpose of gitignore is to not send stuff to GitHub that is not needed. This saves disk space and time to transfer.
- As soon as the wizard is done completing your new project you will see it appear in Netbeans along with a home page in your editor called “index.jsp”. Your home page can have any name you choose, however, “index” is the most common name for a home page. Others are “default”, “home”, “home” and more.
- Web page names must NEVER have spaces or odd symbols in the name. Stick to alphanumeric phrases. Capitalization is up to you but most people use camel case. Also, the extension “.jsp” is not the only one available. Depending on the language there may be other extensions used. For example in Java we will use “.jsp”.
- The **extension is important** because it tells the server the type of resource being requested, and what if anything the server has to do next before sending data back to the client in a response.
- Next, right-click on your project and in the pop-up menu select “Properties”. In the dialog box that follows click the “Libraries” category and verify that you are using JDK 1.8. Further verify that if you have multiple copies of 1.8 that you are using the latest one.
- In the same properties dialog click on the “Run” category and at right set the **“Relative URL”** to “/index.html”. Notice that the **“Context Path”** is set to “/FirstWebApp”. The context path is the name of your web app, which appears just after the server name in a URL. So for example, on your local computer your web app is running at the server address <http://localhost:8080>, where localhost is the default name of your server and “8080” is the port number used to access that server. Port numbers are like apartment numbers. The server address is like the address of the apartment building and the port number is like the address of a particular apartment within that building. In web development the port number is the web server address on that computer. A computer can have more than one web server, hence the need for port numbers.
- The **“Relative URL”** is the name of your web app. Netbeans causes the name of your web app project to be the default name of the web app here. But you can easily change this to something else (NO SPACES). If you forget to set this your app will not display the home page when started. In the future we’ll see how can use a file called “web.xml” to provide this and more features. The relative url shows up as a suffix in the URL:

<http://localhost:8080/FirstWebApp>

Having this is not mandatory. It most useful when there are multiple web apps running on a single server. That way you can target each app by name. But sometimes you will only have only large app running on a server, and in that case the Context Path would be left blank. So in effect there would be no app name. Think of the app name as a sub-directory on the server. You app code will either go into a sub-directory (app name) or at the root (no app name).

- Pause here for a quick review by your instructor about HTML basics. For those who need a more detailed review, or for those who need entry-level training, please go through the tutorial at <http://www.w3schools.com/html/default.asp>

**IMPORTANT: knowledge of HTML is mandatory in this class and you are expected to know it as a prerequisite. Make sure you study this if needed so that you can successfully complete the course.**

- The “**method**” attribute of the HTML form element indicates how the client request is being made and how optional data will be sent to the server. POST is the choice of preference for web forms because there are no data size limits and the data is supplied invisibly to the server for better security. But you could set this to “GET” and it would still work because the data size is small.
- The “**action**” attribute indicates the destination target for the request (the server-side resource). In this case, that resource is the pseudo-name (alias) of the HttpServlet that will be used to process the request. We haven’t created this yet but will soon.
- A **submit** button is used so that when clicked the HTTP protocol will be used automatically to send the form data (make a request) to the resource identified by the action attribute. You could also use a regular <button> tag instead of an <input type=”submit”> tag, but then you would need some client-side JavaScript code to cause a request to happen. With the input tag it’s automatic.
- OK, now let’s create a server-side Java object to process the request and produce a response. Right-click on your project and from the pop-up menu select

New > Other > Web > Servlet

- The servlet wizard will walk you through the creation of this item. Set the class name to “XXXXController”, where “XXXX” describes the purpose of the controller. Set the package to “controller” (always use package names to organize your Java classes into logical groups). Then click

“Next”. There are many options for Configuring Servlet Deployment. For now, we just accept all the defaults. Click “Finish”.

- Your servlet class should appear in your editor. Notice that your Controller extends “HttpServlet”. All servlets have this common parent class, which is an abstract class that provides some common, concrete methods such as doPost(), doGet() and init(), which you will want to override. Netbeans provides an additional method processRequest() which is not part of the HttpServlet class or EE 7, but is provided as a convenience. It is used to process both GET and POST requests. Normally this is done by the inherited doGet() and doPost() methods, but processRequest() can be used conveniently to process both of them in one place. See the doGet() and doPost() methods in your servlet and notice that they call processRequest(). Again, this is not standard, but just provided by Netbeans as a convenience. You do not have to use this if you have other needs.
- A servlet is a Singleton, meaning that there will only be one instance object ever created. How can one instance object be used handle thousands of requests coming from thousands of users? By using threads. The server will create on thread for each request and tie that thread to the servlet. This is very important to understand because multithreading comes with hazards – you do not want to share variables with more than one thread. Your instructor will review that here. So make sure that any class properties you create in your servlet are treated as global constants – not as variables – at least not as writeable variables.
- OK, now let’s write some code to process the client request from our index.html page. In the processRequest() method provided by Netbeans some sample code is provided. This sample code does not do anything with the request info and simply generates a new web page on the fly. This is not what a servlet is typically used for, but it can be. Let’s see what happens if we just run the code as is. Save all changes and run your web project. This will take a while. Netbeans must compile all code, create a “.war” file, deploy that file to the Glassfish server and then start the server and finally load your web app along with the default web browser (change it in Netbeans to Chrome). **Instructor demo to follow.**
- If everything goes according to plan you should eventually see a web page with the message coded in your servlet. Verify this now and let your instructor know if anything failed.
- Great, but this is not what we want. What we want is a message to be dynamically produced by the servlet and returned to the client. To do that we must modify the servlet code.
- **FIRST, you must stop the server.** Do this by clicking on the Glassfish tab in the output window at the bottom of your screen. The Glassfish tab is used to monitor all of the log output from the server which describes what it is doing. This can be useful in debugging later. For now, just click the orange button to stop the server. The button will be in the upper left corner

of the output window. It will be orange if the server is running and gray if it is not running.

- **Why do we need to stop the server to make a change?** Often we do not. If making client-side changes (changes to your web pages) you never have to stop the server. It will do what's called a "hot deploy" after saving your changes. However, when making server-side changes, such as modifying a servlet, this is unreliable. It's always best to stop the server before making any changes to server-side code.
- **Now make your changes**
- The "request.setAttribute" statement creates a key-value pair, storing the error message as a value under the key. This key-value pair is set as an attribute of the request object. The purpose of this is to forward the request object to the destination page where it can be used if needed. This page must be a .jsp **Why ".jsp"?**

Because that page will contain some java code that will be executed on the server BEFORE it's sent back to the client. In other words we will be dynamically changing the html via Java code execution, resulting in a new HTML page whose code will be sent back to the browser in a response. No Java will ever be sent to the browser – only the resulting HTML. Only a JSP page can do that. This would not work if the page extension was ".html". Why? Because ".jsp" tells the server to execute the Java code.

- To create a JSP, right-click on our project and from the pop-up menu select

New > Other > web > JSP

- Give the JSP page a name (the .jsp is added automatically) and save.
- The new JSP page will appear in your editor.

Notice:

The <%@ page ... tag, which indicates that this is a JSP page with certain attributes.

The <% and %> symbols are delimiters that indicate the start-stop points at which real Java code is entered into the JSP page. You are mixing regular HTML with Java code in the same page. HOWEVER, the Java code is processed on the server BEFORE the resulting HTML is sent back to the browser. The Java code never ends up in the web page.

request attributes are retrieved as plain Object data types. These must be cast or changed into the original data types, which in this case are Strings.

We must test for null before casting or converting, else we would get a `NullPointerException`.

Then if not null, we can use the `String` values in our output. Notice we are not using `System.out.println`, but rather `out.println`. Why? Because in this case the `out` object belongs to a servlet. The JSP page is compiled by the server into an `HttpServlet`. Where does the data get printed? Not to the console in this case, but rather to the HTML code stream. This is one way that servlets output data.

- Double check that you have entered all code accurately, save your changes and run the program again. This time you should see a form where you enter our name. Then click the button and see the response output.
- Verify this works and if not let your instructor know immediately if there is a problem.
- This concludes our first, very simple, naïve attempt to create an interactive web application using Java. Now we need to consider some more advanced issues.

- **Introducing the Model-View-Controller (MVC) Design Pattern**

- A **common best-practice** for building web applications in any language is to use the MVC pattern (or variation). The MVC pattern defines a way to separate concerns (responsibilities) into logical groups or modules. This separation decouples code in a way that is beneficial to applications that may have different user interfaces (smart phone, web browser, standalone GUI, etc.), and different server-side logic (Java, PHP, VB.Net, C#.Net, etc.). The idea is that you should be able to change your UI code with affecting your server-side code and vice versus.
- The **“View”** in MVC is the UI code. In our app it was the HTML or JSP page. But it could have been a native smartphone app or standalone VB or Java app.
- The **“Controller”** in MVC is a Java object (or other language) that has one job – to control data flow between client and server (request-response). In a Java web app this will usually be an `HttpServlet`, but it could also be some other Java class, depending on the chosen framework being used.
- The **“Model”** in MVC is one or more classes that are regular Java objects that perform the data processing and data persistence work (database, e.g.). All data processing should be performed here, not in the Controller. And here is where we should use the Service Oriented Architecture (SOA) that you learned in Advanced Java.
- The **data flow in MVC** goes like this:

VIEW  $\leftarrow \rightarrow$  Controller  $\leftarrow \rightarrow$  Model

Notice that the View is decoupled from the Model by the Controller. This



is an important feature. It means that the Model can be modified without affecting the View and vice versa.

- In our examination of the “FirstWebApp” we did not use a Model. Instead we did the data processing in the servlet. This violates the Single Responsibility Principle because the Controller should only control data flow. It should not perform view-related or model-related activity. It should only control flow. What does that mean? It should decide which model object should process a request, and it should decide which view object should be given the response. That’s it!!
- You can have as many controllers as necessary. Generally, you want to follow the Single Responsibility Principle here as well and create different servlet controllers for different control jobs.
- Later we fixed our architecture by using a model object to process the request in our FirstWebApp.
- To create any model object you just create a normal class. Right-click on your project and from the pop-up menu select

New > Other > Java > Java Class

- Name it “XXXXService” and give it a package name of “model”
- You can create other model objects that are not services, but are delegates in the DIP.
- Notice in our “FirstWebApp” example we return the result of the data processing as a String, but we could return any Java object or primitive. The controller needs this as part of the data flow. The controller’s job is to send the data to the appropriate view.
- Now let’s examine our HelloController to see how it sends and receives this data:

Notice the input value in name that came from the View is now being sent to the Model via the HelloService, which in turn delivers a response value back to the Controller. The Controller then forwards that data to the View response page via the RequestDispatcher.

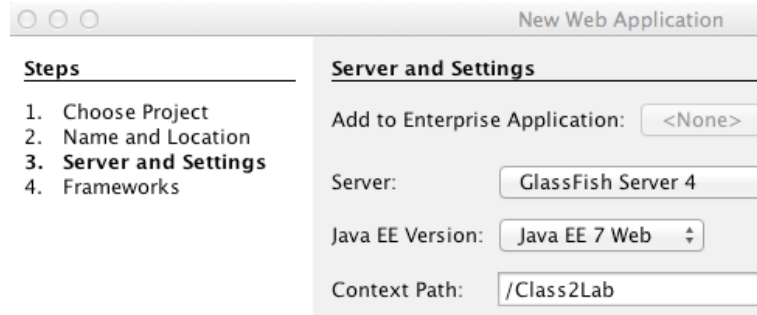
- Run the app again (after stopping and restarting the server) and verify that the app works. Let your instructor know immediately if you have any problems.
- OK, now let’s examine a different sample application provided by your instructor. It’s on Blackboard and is called “BeerAdvice”. Download the zip file, extract and load the project into Netbeans. Then examine the code from View to Controller to Model. Notice how the basic techniques are the same as the app you just built. This is an important thing to understand. You now have a blueprint that you can use to develop other apps, using similar techniques. Use these examples as a study

guide for future work.

- **OK, time to practice what you’ve learned** through a series of lab activities in class and at home if extra time is needed. Please do these one at a time, in order, and **do not continue to the next activity until told to do so.**

✓ **Lab Activity #1** (your instructor will guide you ... wait for instructions):

1. Create a Java Web Application named “Class2Lab”. Make sure you select Glassfish as the server and EE 7 as the version:



2. Note that the “Context Path” can be changed, but defaults to the name of your project. NEVER use spaces in your project name because this Path name is what is used in the URL in your browser to access your site, and spaces are not allowed in URLs.
3. Your web site is running on your computer – not on some remote computer. In a production environment the remote computer would have a domain name, like “wctc.edu” which is the server name. But for your local computer it defaults to “localhost”. Furthermore, in a production environment port 80 is the default and therefore is not needed in the URL, but for development on your local computer, port 8080 is used. This lets you run both a production server and a development server on the same machine. So now the address in your browser (the URL) that you use to access your site will be <http://localhost:8080/Class2Lab>
4. On the next screen in the New Project wizard, do not select any Frameworks. Finally, complete the wizard.
5. Notice that your project appears in the “Projects” tab in Netbeans and your home page, “index.html” has been created and is open in the editor view. Notice further that the HTML5 Doctype is being used. That means you can use all of the new features of HTML5 if you wish, but you can also use older versions of HTML.
6. CAUTION: as you modify or create new files, never use spaces or other punctuation in file or folder names. The web hates that! Best practice is to use camelcase or all lower case.
7. Now, initialize this project for Git.
8. Next, add a “.gitignore” file before doing anything else. A Git Ignore File is a file that begins with a dot “.” And ends with “gitignore”. It can best be created using a Netbeans plugin. You

may or may not have this plugin installed. To see if you do, go to the Tools > Plugins menu in Netbeans and look for it in the “Available Plugins” tab. If you find it there, it is NOT installed. Check the box next to it and install it. Next, use it to create the file. Here’s how: point your mouse cursor at your project name in Netbeans and right-click your mouse on it, which will reveal a popup menu. In that menu, near the bottom, select “Generate gitignore file”. This will reveal a dialog box with a list of templates. Scroll through the list or enter “netbeans” in the filter box. Select that. This will reveal a file in your editor containing some rules. These rules tell Git what NOT to push to GitHub. These are things that do not need to be pushed.

9. After adding your .gitignore file, immediately do a Git > Add, followed by a Git > Commit and enter the message “initial commit”.
10. Create a GitHub project of the same name and push your project to Git using Git > Remote > Push ... (make sure you enter the correct address of your GitHub project and correctly enter your username and password). Report any problems doing this now!!!

✓ **Lab activity #2:**

1. For this activity you will use the same project as in #1 above. Create a package named “activity2” and put all your code there.
2. Your instructor is going to show you how to work with JSPs and Servlets later today, but for now you are going to do some independent research and experimentation.
3. First you need to discover what a “Java Servlet” is. Here are some resources for your research:
  - a. Read this first: <http://www.developer.com/java/an-introduction-to-java-servlets-.html>
  - b. More advanced topics: <https://docs.oracle.com/javaee/7/tutorial/servlets.htm#BNAFD>
4. This is a discovery exercise. You’re supposed to feel like a kid in a candy store experimenting with all the tasty treats that Servlets have to offer. It’s not about being right or wrong, it’s about learning from trial and error.
5. Skills challenge (practice what you’ve learned)
  - a. Create a Servlet named “PageGenerator” and in the New Servlet wizard enter the package name “controller”. To get to the wizard, right-click on your project and select “New > Servlet”. If you don’t see “Servlet” as an option, go to New > Other > web > Servlet. Don’t forget, put your code in the “activity2” package mentioned above.
  - b. On the next screen verify the Servlet Name is “PageGenerator” and change the URL Pattern to “/pager”.

This last change, is of course, optional but demonstrates that you can use any alias you choose for the URL Pattern. DO NOT check the box for using “web.xml”. Now click Finish.

- c. OK you should be looking at your new servlet in the editor view. If you have trouble with this, ask one of your peers for help. Notice the annotations at the top of the class that configure the servlet. We’ll discuss this later.
- d. Netbeans provides a “processRequest” method (this is not part of the servlet API, it’s simply an optional convenience method. Notice that your servlet has a “doGet” and “doPost” method that both call this convenience method. Again, you don’t have to use this method, but it makes things easier. Write your code in this “processRequest” method.
- e. In this method Netbeans has provided some sample “out.println()” statements that generate a web page. Your challenge is to modify this code to output a simple web page that contains a table with three rows and three columns, and a headline above the table. Content can be anything. Watch for typos!!! String concatenation is extremely fragile!
- f. Now run your project to see the results. Enter this URL in your browser:  
<http://localhost:8080/Class2Lab/pager>  
Notice that the word “pager” at the end of the URL is the alias name of your servlet (if you followed the instructions above).
- g. When done (or ask a peer for help). When done, do a Git add/commit and then a push to GitHub.
- h. Time-permitting, experiment with other output in your out.println() statements. You shouldn’t be surprised to know that you can generate any legal code that would normally go into an html page – including CSS and JavaScript. Give it a try. Watch out for String typos and concatenation issues!
- i. The point of this activity is to show you that an HttpServlet can auto-generate the html, css and JavaScript used in a web page. However, this is the least efficient way to create a web page. Use JSPs instead.

✓ **Lab activity #3:**

1. You will use the same project as above and will produce the same results as above, except this time you will use only a single JSP page. This JSP page will be created in the “Web Pages” directory

of your project. If you accidentally create it elsewhere, you can drag n drop it into the correct directory.

2. Create a JSP page named PageGenerator2.jsp (the “.jsp” extension is added automatically by Netbeans). Now program it to output the same results as in the previous lab activity. To do this you can simply add the Java scriptlet delimiters shown previously and write similar code as used in your servlet, or you can use the techniques presented by your instructor.
3. When done (or ask for help) do a Git add/commit/push
4. The purpose of this activity is to show you that a JSP can be programmed to do exactly the same things as a servlet – that’s because JSPs are compiled into servlets. But you can also take advantage of special features that are only available in JSPs (such as mixing html and java code). So if JSPs are really servlets, should we just use them instead? No? Use JSPs as VIEW objects. Use servles as CONROLLERS.

○ **Lab activity #4:**

1. Before starting this activity your instructor will discuss more about servlets and JSPs, and about using “web.xml”. He will also discuss the differences between HTTP methods GET and POST as well as how to use Servlets to forward requests to your view pages. You will need to take notes on this. Your instructor will check to make sure you do!
2. In this activity you are going to use the MVC design pattern to build an app that has a two-page view (both HTML and JSP), a single controller servlet, and a single model class. You will be using the same project as above.
3. Under “Source Packages” create a new package named “model”. In that package create a regular Java class named “WelcomeService”
4. In that service class create a private property for the current date using the Calendar data type. Then create two methods:
5. The first method should be written to determine whether it is morning, afternoon or evening. Hint: you can use the before or after methods in the Calendar class to help you do this. Have it return a String containing “morning”, “afternoon” or “evening”. Come on, it’s not that hard, you just have to think! You made it through Advanced Java, you can do this too!
6. Next, create a method that takes a String value for a person’s name, and constructor a return String that combines that name plus the date result above into a welcome message. Something like this: “Good afternoon, Jim. Welcome!”
7. Next, create an HTML page named “welcome”. Create an appropriate headline and form that takes a person’s name and submits the form to a controller that has the alias “greeter”. Use

what you've learned so far to create the servlet for this alias. In your form don't forget to reference the alias like this:

```
<form id="form1" name="form1" method="POST"
action="greeter">
```

8. Next, create your controller with an appropriate name and the alias mentioned above. In the "processRequest" method remove the boilerplate code provided by Netbeans and replace it with your own. You will need to retrieve the form parameter and pass it to an instance of your model class. The model class will return a String value that you must forward to a result page that is a JSP. In that page you will retrieve that String from the request object and display it on that page. If you don't remember how to do these things, take a look at the sample BeerAdvice app provided by your instructor and use similar techniques. Learn by learning from someone else's code. Again, you are required to do critical thinking to be successful. Bear down and get it done!
9. When done, do a Git add/commit/push. We'll discuss all the lab activities soon, and look at samples of student work.

#### **Lab Activities:**

- Demos and walk-throughs

#### **Textbook Chapters (and other resources) covered:**

- Java EE v1.7 tutorial: <http://docs.oracle.com/javaee/7/tutorial/doc/home.html>
- Java SE API (v1.8): <http://docs.oracle.com/javase/8/docs/api/>
- Java EE API (v1.7): <http://docs.oracle.com/javaee/7/api/>
- Printed Servlet Tutorial: "java-servlet-tutorial.pdf" provided on Blackboard. Note: this uses Eclipse as an IDE. We will not be using eclipse. So the techniques for doing stuff in the tutorial may vary with those of Netbeans.
- Online tutorials for client-side: <http://w2schools.com>
- Netbeans web development tutorials: <https://netbeans.org/kb/trails/java-ee.html>
- Netbeans Git User Guide: <http://netbeans.org/kb/docs/ide/git.html>  
(don't use SSH – we'll be using the modern HTTPS approach)

#### **Preparation Work for Next Class – No points for compliance, but failure to comply costs up to -10 points depending on severity. Do these in order:**

1. Complete lab challenges #2 and #3 in "Class2Lab" project push your solutions to GitHub.
2. Do the research on your selected topic and be prepared to do a 5 minute presentation to the class the next time we meet.