Software Architecture

Week 5

Quick reminder

How to install Git

On Mac OS

The easiest way to install Git on a Mac is via the stand-alone installer:

1. Download the latest Git for Mac installer.
2. Follow the prompts to install Git.
3. Open a terminal and verify the installation was successful by typing `git --version`:

```
$ git --version
git version 2.9.2
```

4. Configure your Git username and email using the following commands, replacing Emma's name with your own. These details will be associated with any commits that you create:

```
$ git config --global user.name "Emma Paris"
$ git config --global user.email "eparis@atlassian.com"
```

5. (Optional) To make Git remember your username and password when working with HTTPS repositories, configure the git-credential-osxkeychain helper.

Install Git with Homebrew

If you have installed Homebrew to manage packages on OS X, you can follow these instructions to install Git:

1. Open your terminal and install Git using Homebrew:

```
$ brew install git
```

2. Verify the installation was successful by typing which `git --version`:

```
$ git --version
git version 2.9.2
```

3. Configure your Git username and email using the following commands, replacing Emma's name with your own. These details will be associated with any commits that you create:

```
$ git config --global user.name "Emma Paris"
$ git config --global user.email "eparis@atlassian.com"
```

4. (Optional) To make Git remember your username and password when working with HTTPS repositories, install the git-credential-osxkeychain helper.

Install Git on Windows

1. Download the latest Git for Windows installer.
2. When you've successfully started the installer, you should see the Git Setup wizard screen. Follow the Next and Finish prompts to complete the installation. The default options are pretty sensible for most users.
3. Open a Command Prompt (or Git Bash if during installation you elected not to use Git from the Windows Command Prompt).
4. Run the following commands to configure your Git username and email using the following commands, replacing Emma's name with your own. These details will be associated with any commits that you create:

```
$ git config --global user.name "Emma Paris"
$ git config --global user.email "eparis@atlassian.com"
```

5. Optional: Install the Git credential helper on Windows

Bitbucket supports pushing and pulling over HTTP to your remote Git repositories on Bitbucket. Every time you interact with the remote repository, you must supply a username/password combination. You can store these credentials, instead of supplying the combination every time, with the Git Credential Manager for Windows.

Setting up the tools

Download the projects

1. Clone the project repository (https://github.com/doplab/soar-tp.git)
2. In your terminal, move to the newly created folder ("Enter the name of the folder")
3. Create a new project on your Gitlab account
4. Link the project to your newly created repository by doing: `git remote add origin <link-to-your-repo>`
5. Make your changes, then push to your repository: `git push -u origin master`

` **Recurrent errors** *Java Fatal Error: Unable to find package java.lang in classpath or bootclasspath*: This issue happen when Netbeans cannot find your JDK or it is

improperly configured. To solve it, you can right-click on your project > properties > Libraries > Select a Java Platform

`

Using Source Tree

1. Click **File** > **Clone / New**
2. Enter the URL below to **Source Path / URL** and choose a **Destination Path** and **Clone** the project. https://github.com/doplab/soar-tp.git

Installing Payara Server

1. Open Netbeans
2. Go to Tools > Plugins > Search "*Payara*" > Check the following plugins:

- Payara Server - the main plugin, and contains the server plugin features
- Payara Common - common shared code between server and micro plugins
- Payara EE Common - API and SPI code for both plugins
- Payara Tooling - UI related source for both plugins

3. Click install once all the relevant plugins are selected, and follow the installation wizard.
4. Click on **Finish** and **Restart** Netbeans.
5. Go to Services tab on Netbeans (If you don't see **Services** tab, click on **Windows** > **Reset Windows**)
6. Right click on **Servers**
7. Click on **Add server**
8. From the server list, choose **Payara Server**
9. Choose an **Installation Location** (and make sure there is no space in the installation path)
   Please note, if you remove the Payara Server folder later, you won't be able to use it anymore.
10. Choose **Local Domain**
11. Below **Choose Payara Server 5.184**, you will see **Download** option, click it and wait for NetBeans to download and install Payara Server. (this will take some time)
12. After it is done, click **Next**
13. Leave the **Domain** as it is (i.e. "domain1"), type a user name and password (if you want to)
14. **Finish**

Starting Payara Server

1. Go to **Services** tab

2. Expand Servers, there you should have **Payara Server**
3. Right-click on Payara Server and **Start**

Opening a project on NetBeans (and running EJB projects)

1. **File** >> **Open Project**
2. Navigate to your project
3. Right-click on the project >> **Run**
4. When you run an EJB project for the first time you will be prompted to select a deployment server. Select **Payara Server** and **Remember in Current IDE Session** or **Remember Permanently**.
5. NetBeans will run the application and redirect you to **localhost:8080/<project_name>**

Modularity and Unit Testing

Requirements

1. Netbeans IDE
2. JUnit
3. TestNG

Running the Unit Test

The goal of this exercise is to run a unit test to see how the methods behave. Our sample project is a **Caesar cipher** project with two main functions: Encoding and decoding a String.

Opening the project on NetBeans IDE

1. **File** > **Open Project**
2. Navigate to `week5>SimpleUnitTests`
3. Run the unit test by right-clicking on your project > **Test**

Exercise 1 - CaesarNGTest.java

You will observe that only 50% of the unit tests are executed successfully.

Implement the encode() method

In this session, we will implement the **encode()** method by returning the Cipher of the message. The **encode()** method should look like this:

```
public String encode(String message) {
    return cipher(message);
}
```

Now, if you re-run the tests, you will observe that 100% of the unit tests are executed successfully.

Write additional tests

In this exercise, you have to write 4 additional tests:

1. A set of **two tests** with k = 0 for the encode() + decode() methods
2. a set of **two tests** for testing that the result is identical for key = 3 and key = 29, again for the encode() + decode() methods; After running the tests, you will observe that they successfully pass even if
   both **encode()** and **decode()** methods return "not implemented yet".

Exercise 2 - ReadTextFileTest.java

In ReadTextFile.java, a simple text reader is implemented.

In ReadTextFileTest.java, you will see that all test cases pass successfully. The purpose of this exercise is to show you some good practices of using;

1. @BeforeClass annotation, is executed before all test methods (@Test) of the class. For instance; you can put initialization code here. Here, we use this annotation to read from a file and initialize the nameList.
2. @AfterClass annotationn, is executed after all test methods (@Test) of the class. For instance, you can write cleanup code here. We use this annotation to clear the nameList and close the file we were working on.
3. Exception testing - @Test(expectedExceptions = .class) In order to test an exeption, you must specify which exception is expected when the test method is executed.
4. Timeout testing - @Test(timeOut = 100) Sometimes a test might run too long than it is expected. In such cases, you can limit the run time of a test by setting a timeOut value. If the test takes longer than a defined timeOut value, it will fail. (timeOut value is in milliseconds)

EJB Exercises

Requirements

1. Netbeans IDE
2. Payara Server

Running an example with session beans

The goal of this exercise is to show how Session bean works. During this session, we will highlight the differences between stateless **session bean**, **stateful session bean** and **singleton bean**.

Opening the project on NetBeans IDE

1. **File** > **Open Project**
2. Navigate to `week5>tutorial-examples-master`

Run the examples

The sample project includes many modules implementing the different types of session beans. We will focus on the **converter example** which shows a simple **stateless session bean** and the **counter example** which shows a simple **singleton bean**

Please note that everytime you click on a module, NetBeans will open the module as a different project. Be patient.

1. Open **Java EE Tutorials Examples >> Modules >> ejb >> converter**

A simple **stateless session bean** is implemented in the converter application.

2. Open **Java EE Tutorials Examples >> Modules >> ejb >> counter**

A simple **singleton bean** is implemented in the counter application.

Write your own application

1. After running the example, the next step is to write our own application with a **stateless session bean** doing the caesar conversion. For this exercise, we will use the **converter app** as template.
2. Write an ejb-based application with a **stateful session bean** doing the caesar conversion, using the **converter app** as template.
3. **Singleton Session Bean** - For this exercise, you're expected to complete the **CaesarSingletonBean.java**. You are given the lists of already encoded/decoded pairs. Make sure you have all the accessors and mutators needed.