CS144: Web Applications

Project Part 1: Setup Your Environment
Due Friday, January 20, 2017

- **Submission deadline:** Programming work is submitted electronically and must be submitted by Friday at 11:00PM. However, we recognize that there might be last minute difficulty during submission process, so as long as you started your sumission process before 11:00PM, you have until 11:55PM to completely upload your submission. After 11:55PM, you will have to use grace period as follows.

- **Late Policy:** Programming work submitted after the deadline but less than 24 hours late (i.e., by Saturday 11:00PM) will be accepted but penalized 25%, and programming work submitted more than 24 hours but less than 48 hours late (i.e., by Sunday 11:00PM) will be penalized 50%. No programming work will be accepted more than 48 hours late. Since emergencies do arise, each student is allowed a *total* of four unpenalized late days (four periods up to 24 hours each) for programming work, but no single assignment may be more than two days late. See the Assigned Work page for more information.

- **Honor Code reminder:** For more detailed discussion of the Honor Code as it pertains to CS144, please see the Assigned Work page under Honor Code. In summary: You must indicate on all of your submitted work *any assistance* (human or otherwise) that you received. Any assistance received that is not given proper citation will be considered a violation of the Honor Code. In any event, you are responsible for understanding and being able to explain on your own all material that you submit.

- **Reminder:** Projects must be completed individually or by a team of two.

In this course you will be developing an online auction web application using Java, MySQL, Lucene, Apache Tomcat and Apache Axis2. Since it is quite error-prone and tedious to install and configure these software tools correctly, you will be doing all project development using a *virtual machine* (VM) that has all these tools preinstalled. After you follow the steps below, to install the virtual machine for the project, you will have to write a simple Java program.

Part A: Setup Your System

Download and Install VirtualBox Image

VirtualBox is an application that runs on a *host* operating systems, such as Windows or Mac OS X. The host is the actual operating system and physical hardware, such as your laptop. Once VirtualBox is setup, it allows you to run multiple *guest* operating systems inside, which can be any OS capable of running on the x86 or x64 architecture. In other words, it provides the illusion of having multiple "virtual machines (VM)" on a single hardware, each with its own operating system (e.g., Linux guest).

You will need the following files to setup the Linux virtual machine for the class project:

- **VirtualBox Program:** VirtualBox software
- **Virtual Machine Image:** CS144.ova

Follow the step-by-step instructions on this page to install our VM on your computer.

Get Familiar with Unix Command-Line Shell

All of your project development should be done and tested inside our VM, which is based on Ubuntu Linux operating system. For the most part, you will be using a Unix shell through xterm in our VM. If you are not familiar with Unix shell commands, read the Unix Tutorial for Beginners and learn the basic Unix commands.

There are a few important *environment variables* that have been set within the VM.

- **JAVA_HOME**: The `JAVA_HOME` environment variable points to the JDK installation directory, which is `/usr/lib/jvm/default-java` in our VM.

- **CATALINA_HOME**: This variable specifies the location of the Tomcat installation, which is `/usr/share/tomcat7`.

- **CATALINA_BASE**: This variable specifies the base directory of a Tomcat instance, which is `/var/lib/tomcat7`.
- **AXIS2_HOME**: This variable specifies the location of the Apache Axis2 installation, which is `/usr/share/axis2`.
- **EBAY_DATA**: This variable specifies the location of the ebay XML auction data that we will be using for the project. Its value is `/home/cs144/ebay-data`.
- **HOME**: This variable specifies the location of your home directory, which is `/home/cs144`.

All of the above environment variables are set in the `.bashrc` file located at your home directory in the VM and are set up automatically whenever you login. You can refer to the values of these variables in your Unix shell, like `$JAVA_HOME`, `$CATALINA_HOME`, etc. For example, if you issue the following `echo` command in xterm

```
cs144@class-vm:~$ echo $JAVA_HOME
/usr/lib/jvm/default-java
```

you will see the value of `JAVA_HOME`.

Part B: MySQL Warm-Up

In this part, you get yourself familiar with the basic MySQL commands by creating and loading a table and issuing a few queries.

First, read our basic MySQL tutorial to learn how you can interact with MySQL and issue SQL commands. In particular, the tutorial also provides a brief explanation on `CREATE TABLE`, `LOAD DATA` and `SELECT` commands, which will be important to finish this part of the project. If you need to brush up on SQL commands other than the ones explained in the tutorial, you may find the *How Does This RDBMS Thing Work?* section of SQL for Web Nerds helpful.

As we explained in the MySQL tutorial, we have created two databases in MySQL on our VM: `TEST` and `CS144` (note these database names are case sensitive). The `CS144` database is your "production" database, meant for use in the final versions of your code. The `TEST` database is for any experimentation and for use during development and debugging. We have created two MySQL users for the database. For your project work, use the MySQL user `cs144` (no password), which has full access to the `CS144` and `TEST` databases. The MySQL user `root` with password `password` has full unrestricted access to everything and should be used only for special adminstrative operations, like creating new users and databases, etc.

Now take the following steps to create, load, and query a table in MySQL:

1. We have included a comma-separated data file at `~/data/actors.csv` inside VM. Create a table called Actors in the `TEST` database (since this is just a warm-up project, we will be using `TEST`, not `CS144`). The Actors table should have the following schema:

   ```
   Actors(Name:VARCHAR(40), Movie:VARCHAR(80), Year:INTEGER, Role:VARCHAR(40))
   ```

2. Load the `actors.csv` file into the Actors table. Make sure that the double quotes enclosing some of the attributes in the data file are removed when they are loaded.
3. Retrieve some loaded data from the Actors table. In particular, write a query that returns the answer to this question: "Give me the names of all the actors in the movie 'Die Another Day'." Feel free to experiment with other interesting queries.
4. Once you are done, drop the Actors table from MySQL, so that it will not stay in the database for later parts of our project.

Create a MySQL batch script file named `actors.sql` that shows every one of the steps in Part B. Again, the MySQL tutorial had a brief explanation on how to create and run MySQL batch script file. Make sure that your script is executable and has no error, meaning that we should be able to run your script by issuing the following command:

```
cs144@class-vm:~$ mysql TEST < actors.sql
```

Please use `'~/data/actors.csv'` as the location of the data file inside your script (in Unix, the symbol `'~'` indicates the user's home directory). You may assume that there is no Actors table in the `TEST` database when we execute your script. If needed, use the `--` tag to make comments within your SQL script.

Notes on CR/LF issue: If your host OS is Windows, you need to pay particular attention to how each line of a text file (including your script file) ends. By convention, Windows uses a pair of CR (carriage return) and LF (line feed) characters to terminate lines. On the other hand, Unix (including Linux and Mac OS X) uses only a LF character. Therefore, problems arise when you are feeding a text file generated from a Windows program to a Unix tool (such as `mysql`). Since the end of the line of the input file is different from what the tools expect, you may encounter unexpected behavior from these tools. If you encounter any wired error when you run your script, you may want to run the `dos2unix` command in VM on your Windows-generated text file. This command converts CR and LF at the end of each line in the input file to just LF. Type `dos2unix --help` to learn how to use this command.

Part C: Java Programming Warm-Up

If you are new to Java or if it has been a while since your last Java programming, first read A Crash Course from C++ to Java. This excellent tutorial explains the basics of Java, including how you can name, compile, and run your Java program. (It is okay to skip the parts on BlueJ in the tutorial, since we will not be using it.) If you are quite familiar with Java, but you just want to brush up on minor details quickly, you may want to read slides on Java instead. All basic tools needed for Java programming (e.g., `javac` and `java`) are available on our VM.

Now implement a Java program that computes the SHA-1 hash over the content of an input file. SHA-1 is a cryptographic one-way hash function that computes a 160 bit value (or 40-digit hex value) from a sequence of bytes. Your Java program should satisfy the following requirements:

1. Your program should be implemented as a single Java class `ComputeSHA`. Your program should take the input filename as the first command line parameter. For example, a user should be able to execute your program like

   ```
   java ComputeSHA filename.txt
   ```

   where `filename.txt` is the name of the input file.
2. Given the input file, your program should compute the SHA-1 hash value over the entire content of the file and print the computed hash value on screen as follows:

   ```
   17a23c746fed66a6f285c665422deafcf51aca40
   ```

   Your program should print the above hash value if you provide the sample-input.txt file as its input. Please ensure that the output is formatted exactly as above including its case.

   Notes on wget: For downloading a small file to the VM, you can use the command `wget`. For example, the `sample-input.txt` file can be downloaded as follows:

   ```
   cs144@class-vm:~$ wget http://oak.cs.ucla.edu/classes/cs144/projects/project1/sample-input.txt
   ```

In implementing your program, you may find the Java class `java.security.MessageDigest` useful, which provides a number of cryptographic hash functions including SHA-1 and MD5. If you decide to use `MessageDigest` class, make sure you import `java.security.*` packages in your source code. If what we just said sounds Greek to you, again, read A Crash Course from C++ to Java. To learn how you may use MessageDigest class in Java, try a query like "Java MessageDigest example" on Google and look at the top few results. They are likely to contain good example codes that show you how you can use the class. If you want to learn about basic file I/O in Java, see Java File I/O tutorial.

Notes on editors for Java development: You can choose whatever editors you like for Java development. Options include:

- Simple text editors: You may use any text editor in the VM (`vi`, `emacs`, `nano` and `nedit` are available in the VM) to edit text files. Instead, you may use your favorite text editor from your host OS (e.g., notepad or TextEdit) and transfer the edited file to the VM through the shared directory. Remember that the directory that you specified as the shared directory from the host (e.g., `C:\vm-shared` from Windows) is available at `/mnt/hgfs/host_shared` in the VM, which is symbolically linked at `$HOME/shared` as well. Again, be careful with the CR/LF issue if you use the second option.

- Java IDE: IDE (integrated development environment) provides a very powerful and convenient programming environment, with features such as constant compile error checking, automatic compiling, and integrated debugging. While we *do not support* a particular Java IDE, many students have reported that Eclipse was particularly easy to use and powerful.

Your Final Submission

Your project must be submitted electronically before the deadline through our CCLE Course Website. Navigate to **Sections** on left of the page, and click on the **Project 1** submission section. If you submit multiple times, we will grade only the latest submission.

What to Submit

The zip file that you submit must be named `project1.zip`, created using a zip compression utility (like using "`zip -r project1.zip *`" command in the VM). You should submit this single file **project1.zip** that has the following packaging structure.

```
project1.zip
 |
 +- team.txt
 |
 +- actors.sql
 |
 +- ComputeSHA.java
 |
 +- README.txt (Optionally)
```

Each file or directory is as following:

1. **`team.txt`**: A plain-text file (no word or PDF, please) that contains the UID(s) of every member of your team. If you work alone, just write your UID (e.g. 904200000).
   If you work with a partner, write both UIDs separated by a comma (e.g. 904200000, 904200001). **DO NOT put any other content, like your names, in this file!**
2. **actors.sql:** A copy of your MySQL batch script file from Part B of this project.
   - You **must** use '**`~/data/actors.csv`**' as the location of the data file inside your script.
3. **ComputeSHA.java:** The source code of you Java program for SHA-1 hash computation from Part C.
4. (Optionally) **README.txt:** A README file (plaintext) containing anything else you find worth mentioning.

Please ensure that your submission is packaged correctly with all required files. Make sure that each file is correctly named (including its case) and project1.zip contains all files directly, not within a subdirectory. In other words, unzipping project1.zip should produce the files in the same directory as project1.zip. **Significant points may be deducted from your submission if the grader encounters an error due to incorrect packaging, missing files, and failure to follow our exact spec.**

Testing of Your Submission

Grading is a difficult and time-consuming process, and file naming and packaging convention is very important to test your submission without any error. In order to help you ensure the correct packaging of your submission, we have made a "grading script" p1_test available. In essence, the grading script unzips your submission to a temporary directory and executes your files to test whether they are likely to run OK on the grader's machine. Download the grading script and execute it in the VM like:

```
cs144@class-vm:~$ ./p1_test project1.zip
```

(if your project1.zip file is not located in the current directory, you need to add the path to the zip file before project1.zip. You may need to use "chmod +x p1_test" if there is a permission error.)

You **MUST** test your submission using the script before your final submission to minimize the chance of an unexpected error during grading. Again, significant points may be deducted if the grader encounters an error during grading. When evertything runs properly, you will see an output similar to the following from the grading script:

```
Running your actors.sql script...
Name
Brosnan, Pierce
Cleese, John
Echevarria, Emilio
Ho, Thomas
Lee, Will Yun
Madsen, Michael
Makoare, Lawrence
Salmon, Colin
Stephens, Toby
Yune, Rick
Finished running actors.sql
```

```
Compiling ComputeSHA.java...
SUCCESS!
```

Grading Criteria

- System setup (30%) - gets full credit if the submission was successful
- No error in actors.sql (20%) - gets full credit if the script runs without any error
- Correctness of actors.sql (10%) - gets full credit if the query returns the correct results
- ComputeSHA compilation (10%) - gets full credit if the submission compiles without error
- ComputeSHA correctness (30%) - based on several test cases