Assignment 2: Spring framework and Unit testing
Release Date: September 27, 2017
First Deadline: October 6, 2017 (Friday, 11:59 pm)
Due Date: Midnight October 8, 2017 (Sunday, 11:59 pm)
Total points: 100
(This is an individual assignment)

Objective:
In this assignment you will learn about Spring framework and unit testing.

Introduction:
In this assignment we will build on the previous assignment and calculate the total number of files generated corresponding to meetings that happened for a particular project in a particular year. Similar to assignment 1, you will query the eavesdrop website (http://eavesdrop.openstack.org/meetings) for obtaining the meeting data.

You *do not* need to implement sessions in this assignment.

You will implement a Spring controller that will handle two query parameters:
- project
- year

The project and the year parameters will be used for determining what data should be shown in response to a request.

Setup:
You should run your Spring controller to respond to following URL pattern: /openstackmeetings
You should set the context root of your web app to: /assignment2
With above settings, your web app will be accessible at:
http://localhost:8080/assignment2/openstackmeetings

Here are some sample interactions:

http://localhost:8080/assignment2/openstackmeetings	Welcome to OpenStack meeting statistics calculation page. Please provide project and year as query parameters.
http://localhost:8080/assignment2/openstackmeetings?project=solum&year=2015	Number of meeting files: 4
http://localhost:8080/assignment2/openstackmeetings?project=solum&year=2014	Number of meeting files: 4

The allowed values for the query parameters are as follows:

Query parameter	*Allowed Values*	*Case sensitivity, other restrictions*
project	Names of projects from eavesdrop site	Should not be treated as case sensitive (solum, Solum, soLuM, etc. should be treated as same)

year	Year numbers from eavesdrop site	Only digits are allowed (2016 is correct, twentysixteen is incorrect)

Other constraints on input and output
1. The query parameters themselves are case-sensitive. You are required to support query parameters that are only lower case (“project”, “year”).
2. Output format should be plain text.
3. If an invalid value is provided for project or year parameters you should respond with following error messages as applicable: “Project with <name> not found”. “Invalid year <year-number> for project <project-name>”. If in a single request both the project name and the year are invalid then your response should be “Project with <name> not found” and “Invalid year <year-number> for project <project-name>”.
4. For any additional constraints not specified above your program should behave in a sane manner (should not crash).

Details:

Part 1:

You need to use the *Spring framework* for this assignment. Break down your code into logical layers consisting of a controller layer and a service layer. The controller layer will contain your controller class and the service layer will contain one or more service classes. The controller should perform input validation and output generation. The service classes should implement the logic of querying the eavesdrop website, parsing the data, generating the statistics.

For dependency injection, you are free to use either setter injection or constructor injection. Define the beans and dependency wiring in *servletContext.xml* file.

Part 2:

Implement JUnit and Mockito based unit tests for the controller class and service classes.

You need to provide *at least five unit tests total* between controller and service classes. This means you could have 5 tests for the controller and 0 for the service class, or 4 for controller class and 1 for service class, or 0 for controller and 5 for service class, or some other combination. Note that unit tests that don't test anything will not count. Also, realize that unit tests are going to dependent on your code. If you make your code modular with separate classes and separate methods for different concerns, you will have better chance of writing good unit tests.

Use pom.xml with dependencies for mockito, junit, spring-webmvc.

Grading criteria:

- First version was submitted by October 6th
- Web app works correctly for different inputs subject to various constraints
- Number of unit tests - at least 5
- Unit tests for Controller class
    - Unit test present

    - Tests pass
    - Appropriate mocks created
    - Appropriate expectations set
    - Appropriate asserts and verifys defined
- Unit tests for Service class(es)
    - Same as above

Submission Instructions:
Name your submission folder as “assignment2”
Name your Controller class “OpenStackMeetingsController.java”
Structure of assignment2 directory should be as follows:
assignment2/src/main/java/<Controller class>
assignment2/src/main/java/<Service interface>
assignment2/src/main/java/<Service class>
assignment2/src/main/java/<Any other Java class(es)>
assignment2/src/test/java/<Controller test class>
assignment2/src/test/java/<Service test class>
assignment2/WebContent/WEB-INF/web.xml
assignment2/WebContent/WEB-INF/servletContext.xml
assignment2/pom.xml
assignment2/README.txt

Include following information in README.txt
name: <your name>
eid: <your ut eid>
bitbucketid: <your bitbucket id>
comments: <Comments, if any>

Create a *private* repository on bitbucket (https://bitbucket.org/product/pricing?tab=host-in-the-cloud sign up using the free option if you don't have bitbucket account). Name the repository assignment2 and push/submit all the files that you create for your assignment to this repository.
Grant “read” access to me and Sailesh
- Usernames: devdattakulkarni, svsailesh

We will use the latest commit ID for grading. You don't need to submit anything on canvas.

Useful material:
Class Notes: Servlets-3.pdf Spring_framework.pdf, Unit_testing.pdf
Book Chapters: Chapter 12 from “Java for Web Applications”
Github: https://github.com/devdattakulkarni/ModernWebApps/tree/master/Servlets
Following example projects: spring-email-service, test-load-on-startup, mockito-example, servlet-unit-test

About deadlines:
First deadline: Initial submission which is will be graded for 3 points. You don't need to have assignment implementation completed by this date. This means if your first commit is on or before October 6 11:59 pm, you will receive points out of 100. If your first commit is after that then you will

receive points out of 97. This deadline is to ensure that you complete all the required setup (Eclipse, Tomcat, etc.) early and not postpone it until last minute.

Due date: Deadline by which your assignment needs to be completed.

Late penalty:
5 points for each late day after the due date.

Collaboration policy:
This is an individual assignment. You are allowed to discuss concepts and high-level implementation questions with each other. But you are not allowed to copy or share code with each other or students who might have taken this class before. Final submission should be your own code.