CS362-004

Final Project:
The final project is designed to check for your cumulative understanding.

Part-A: Due date **is Sunday, July 30th** at 23:59 pm

Part-B: Due date **is Thursday, August 17th at** 23:59 pm (PST).

Extra Credit Total Points: 5 points. Due date is **Friday, August 18th at** 23:59 pm (PST)

Note:

- Use the **URLValidatorCorrect** folder for **Part-A** and use the **URLValidatorInCorrect** folder for **Part-B**.
- Submit **one copy** per group to the following locations: Canvas (pdfs) and the GitHub class repository (code).
- Add a comment in **Canvas** and give the URL of the student fork who submitted your project in the GitHub (under final project).
- Do not forget to write your group memebrs in the pdf files!
- The code and documentation for the Apache Commons Validator is available in this link **http://commons.apache.org/proper/commons-validator/**

Preliminaries and Warm up

Get the latest files from the class repository by running

```
git remote add upstream "https://github.com/aburasa/CS362-004-U2017"
git checkout master            # make sure we're on the master branch
git fetch upstream             # pull any information about changes in upstream
git merge upstream/master -m "Sync" # merge new files
```

You should now have new files in /FinalProject/

Copy finalproject folder into your projects/youonid/FinalProject/

The directory "/FinalProject/" contains 2 different folders **URLValidatorCorrect** and **URLValidatorInCorrect**. We will be using the **URLValidatorCorrect** folder for **Part-A** and **URLValidatorInCorrect** for **Part-B**.

Part-A : Review of Existing Test Case(s)

For Part-A, you will be provided a correctly working version of URL Validator. Explain testIsValid Function of UrlValidator test code. It is available under **URLValidatorCorrect** folder. Give how many total number of the URLs it is testing. Also, explain how it is building all the URLs. Give an example of valid URL being tested and an invalid URL being tested by testIsValid() method. UrlValidator code

is a direct copy paste of apache commons URL validator code. The test file/code is also direct copy paste of apache commons test code. Do you think that a real world test (URL Validator's testIsValid() test in this case) is very different than the unit tests that we wrote (in terms of concepts & complexity)? Explain in few lines. Submit a file called **ProjectPartA.pdf** in Canvas with your write-up. (**20 Points**)

Part-B: Testing URL Validator

You are provided a buggy version of URLValidator. You need to find out as many bugs as you can in this bad URLValidator. In the **Part-A**, I have provided the current test framework that Apache commons team had to test URLValidator. We need to assume that all those tests don't exist. Your team is a testing company and client comes to you with URLValidator implementation and asks for your help to make it bug-free. You need to just concentrate on **isValid method**, the one that is tested in testIsValid() method. Your task is to find out all the bugs, find out failure causes and provide bug reports. Though for this project, as long as you find out **three bugs** that should be sufficient. You don't need to fix any of the bugs. Developers will do it.

You can use any methodology that you learn during the class to test it. To stay consistent let us do it this way:

First, just do manual testing. Call the valid method of URLValidator with different possible valid/invalid inputs and see if you find a failure. (**10 points**)

Second, come up with good input partitioning. Try to provide a varying set of inputs that partition the overall input set well. Did you find any failures? You can call valid method once or more for each partition. (**10 points**)

Third, do programming based testing. Write few unit test cases. You can have some sort of loop in your unit test and test different URL with each instance of the loop. Something very similar to testIsValid() but your own logic and idea. Even a single test will be sufficient if you write it like testIsValid() method. Did you find any failures? (**20 points**)

Submit a report called **ProjectPartB.pdf** in Canvas contains the following Sections: (**40 points**)

- **Methodology Testing** (**15 points**)
  - Clearly, mention your methodology of testing.
  - For manual testing, provide some of your (not all) urls.
  - For partitioning, mention your partitions with reasons and some of the urls that represent each partition.
  - In the report mention the name of your test.
- **Bug Report** (**10 points**)
  - Write bug report for each of the bugs you found. You need to write about the following:
    1. What is the failure?
    2. How did you find it and describe the test case that detected it?
    3. What is the cause of that failure? Explain what part of the code is causing it?
- **Debugging** (**10 points**)
  - When you find out any failure, debug using Eclipse/IntelliJ debugger or any other tool and try to localize its cause. Provide at what line/lines in what file the failure manifested itself. Did you use any of Agan's principle in debugging URLValidator?

- Provide your debugging details for each bug.

- **Team Work** (**5 points**)
  - Write about how did you work in the team? How did you divide your work? How did you collaborate?

Extra credit (5 points) Due date is Friday**, August 18th** at 23:59 pm (PST)

Extra credit part is individual.

Submission guideline for extra credit:

1. Add a folder Extra Credit under your onid. Submit your work there.

2. Send an email to **aburasa@oregonstate.edu** mentioning that you submitted extra credit work.

Do one of the following:

1. Write a random tester for **URL Validator** valid() method.
2. Implement Tarantula using gcov, and describe the process of using it to localize a bug in Dominion. Write this up as tarantula.pdf, and include any relevant code or files.
3. Use delta debugging tools (downloaded from Zeller's website) to minimize a failing test case for Dominion. Describe the process and what you did in deltadebug.pdf, and include any relevant code (modifications of the python scripts) or files. One way to go about this is to take your work from Assignment 4 and modify it so that it produces a standalone C file containing calls to play the randomly generated Dominion game.

Submission instructions:

- **Canvas**
  - **ProjectPartA.pdf** **(**due date is Sunday, July 30th at 23:59 pm) (**20 Points**)
  - **ProjectPartB.pdf** **(**due date is Thursday, August 17th at 23:59 pm) (**80 Points**)

- **The class github repository**
  - Submit your unit tests/random tests on github under **projects/your-onid/URLValidatorInCorrect/** folder. **(**due date is Thursday, June 17th at 23:59 pm)

  - Create a new **branch** of your repository called "**youronid-finalproject**" contains your final submission. This branch must be created before the due date to receive credit.

Add a comment in **Canvas and give the URL of the student fork who submitted your project in the GitHub(under final project).