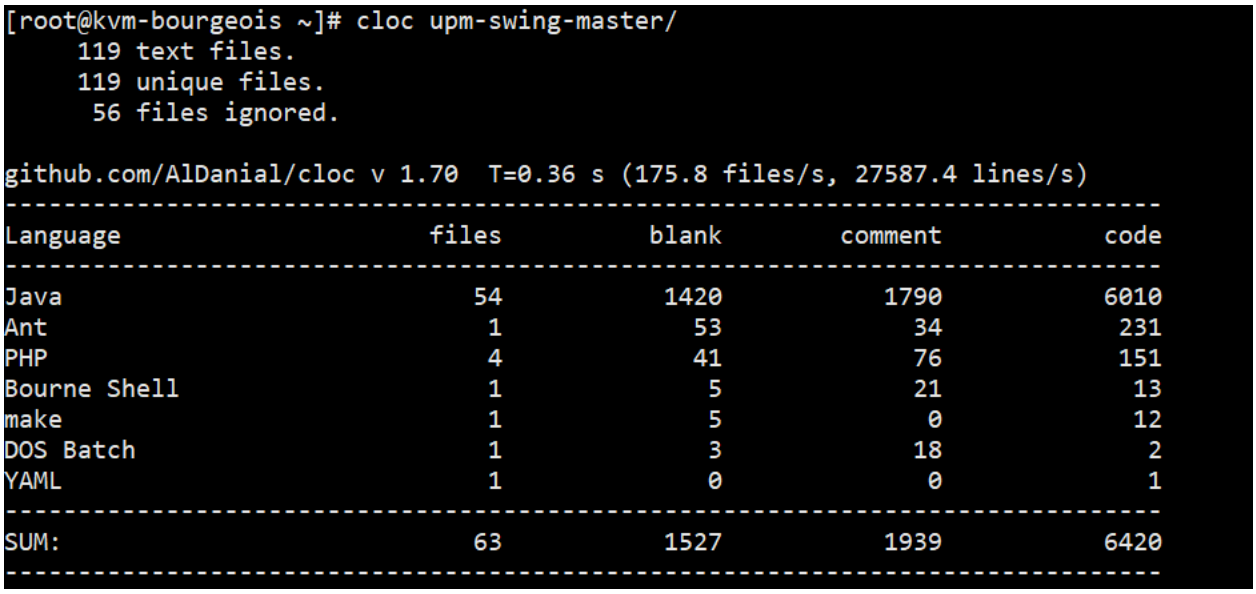
Julian Rechsteiner
Tyler Bourgeois
ESOF 322
Homework 4
Due Thursday October 10th

Homework 4:

Question 1:

a.

1. We downloaded a Universal Password Manager (UMP). We found this system on Sourceforge, which led us to a GitHub repository. The link to this system can be found here: https://github.com/adrian/upm-swing. This open sourced system is written in Java and has cross-platform support.
2. This system manages users' usernames, passwords, URLs, and notes. It is stored in an encrypted database (uses AES for encryption), and it is protected by one master password. The benefit of utilizing this system over other UMPs is that it is cross-platform with any other major OS, has a simplistic design, and has synchronization; this allows the user to update/add a password from a device and use it on another. It also has multilingual support, which broadens the numbers of users capable of using it.
3. The program has 6010 lines of code written in Java. We calculated this using the "cloc" (Count Lines of Code) command on my centos7 Linux server for CSCI 351. This package recursively counts the lines of code in a directory, so we ran the command on the directory containing all the source code. We included a screenshot of the console output below.



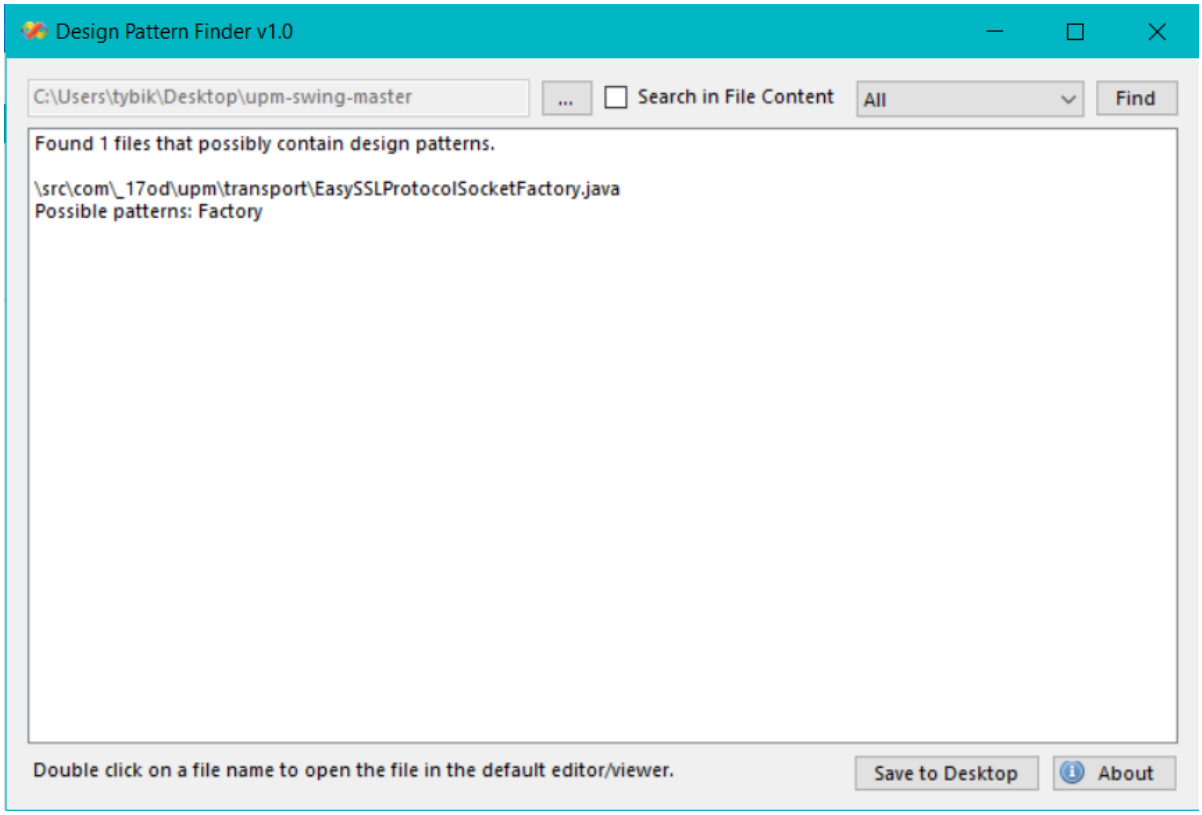
```
[root@kvm-bourgeois ~]# cloc upm-swing-master/
     119 text files.
     119 unique files.
      56 files ignored.

github.com/AlDanial/cloc v 1.70  T=0.36 s (175.8 files/s, 27587.4 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Java                            54           1420           1790           6010
Ant                              1             53             34            231
PHP                              4             41             76            151
Bourne Shell                     1              5             21             13
make                             1              5              0             12
DOS Batch                        1              3             18              2
YAML                             1              0              0              1
-------------------------------------------------------------------------------
SUM:                            63           1527           1939           6420
-------------------------------------------------------------------------------
```

Julian Rechsteiner
Tyler Bourgeois
ESOF 322
Homework 4
Due Thursday October 10th

b.

1. Here is the output of the tool:



2. We believe that the tool scans through the source code and seeks at first either interfaces or abstract classes. Then, the tool looks for their relationships; it checks which interfaces are implemented where and/or which abstract classes are extended. (checks for instance variables, parameters, local variables…) Each design pattern uses an interface or abstract class, and they are each uniquely connected to other concrete classes in some sort of way. The tool looks for these relationships, and then double checks with its source code to find these specific patterns. It may also look at the names of classes, because most classes end with the name of the design pattern used.

3. We were not completely sure if this tool was correct. However, after dissecting the code for some time, the use of a factory pattern here makes sense.
   First of all, according to TutorialsPoint, “the factory pattern is one of the most popular design patterns in Java.” This lines up with the fact that the whole program is written in Java.

Julian Rechsteiner
Tyler Bourgeois
ESOF 322
Homework 4
Due Thursday October 10th

The factory pattern is primarily used to instantiate an object without depicting the logic behind creating this object. This would make sense in this context because you do not want to explicitly show the user how you are creating a socket connection for synchronization, especially when security is your main source of service. We also observe the factory pattern in motion in the "EasySSLProtocolSocketFactory.java" file of the program; it implements an interface named "SecureProtocolSocket-Factory.java", which is an interface in the HTTP Client API created by the Apache Software foundation (more information can be found here: https://hc.apache.org /httpclient3.x/apidocs/org/apache/commons/httpclient/protocol/SecureProtocolSocketFactory.html).

If we look back at the code of the program inside "EasySSLProtocolSocketFactory", it has three different ways of creating a socket (used overloading in this case). Therefore, we can safely say that this file is creating essentially the same socket object, but in three different ways depending on its parameters.

In my opinion, I would use a similar approach to seek design patterns in programs. I would firstly seek for interfaces and abstract classes, and then evaluate their relationships with other concrete classes; this is the main way to find patterns in programs. Then, I would check the name of the class and parse through the name to see if it refers to a pattern. Although not finding a name to pattern using this method is completely fine, it is an easy way to verify if a class is indeed using a design pattern if such name is found during the parsing.

Another way to find a pattern is possibly looking at specific traits of the program for the design pattern it is currently seeking. For example, an adapter pattern is usually a link from legacy code to current code, or used for cross platform integration. We can actually see this in the UMP program we found; there is a file called platform specific, and it changes certain parameters/attributes based on the OS. An adapter pattern would be useful in this case so that different OSs can run the same core methods in a program.

Another example would be the State pattern. A state pattern usually has many states in the program; these states all implement a parent interface, which is generally the generics of the state object. Looking for these design pattern specific patterns, and running a name parsing function for further confirmation would be the best and efficient way of identifying design patterns in a program.

Julian Rechsteiner
Tyler Bourgeois
ESOF 322
Homework 4
Due Thursday October 10th

Question 2:

a. Before uploading any files to my GitHub account, I went ahead and created a specific folder in my local ESOF folder on my computer (all of my school folders are uploaded in the cloud). Once inside the folder I created (in this case I named it "Practice GitHub"), I opened the terminal in that directory and wrote git init. This created a local git repository in that specific directory I was in. In parallel with this, I also created a folder on my GitHub Account called ESOF-322. This is where I am planning on uploading all of my ESOF assignments. Once that is created on my GitHub, I copied the link for an ssh clone; I went back to my terminal and wrote git clone <ssh-clone-link>, where everything in the <> is the link I copied. Once the clone completed I checked to see if the remote repository was downloaded to my computer. From here, I copied all of my assignments into this local repository.
   Once copied, I went back to my terminal and wrote the following lines:
   - Git add . (this is where I staged all of my changes in my local repository)
   - Git commit (this is where I push all of my changes to my local repository)
   - Git push origin master (this is where I push all of my changes to my remote repository)
   - Once I executed all of these lines in the terminal, I double checked my remote repository on GitHub for my pushed changes and they all appeared there.

b. For question b, I will be following the same steps as question a. Technically, GitHub has an upload function where I can simply drag all of my files onto the browser and it would upload. Although that is a faster way, I like using the way I proposed because that is how developers collaborate their code with a version control functionality.
   For this question, I am uploading a python file I created. It is called example.py. I am going to push this file onto my GitHub account.
   - First, I want this file in my ESOF repository so I am going to create this file in that specific directory.
   - Next, since creating a file in a repository is considered a change, I want to stage the change. (You can easily check this by checking the git status with: git status). This is where I make changes but I do not commit anything yet. The terminal line is: "git add ." (I can technically write git add example.py if I wanted to stage a specific file. The period means everything in the current directory).
   - Once staged, I want to commit my changes. This is where I push my changes to my local repository. I will be using the command "git commit" (I also wrote -m "committing", where I created a message for this message called "committing". This is great for collaboration purposes so that other developers (and yourself) know the changes you committed).
   - Finally, I want to upload this to my GitHub account. I previously connected my computer with my GitHub account via ssh so I do not need to do that again; GitHub

Julian Rechsteiner
Tyler Bourgeois
ESOF 322
Homework 4
Due Thursday October 10th

recognizes this computer. I will use the command “git push origin master”. This means that I am pushing my changes to the master branch. Once completed, I can check my GitHub account and I will see example.py on there. The screenshots for this process will be located below.

```
julians-mbp:ESOF-322 myfatduck$ ls
Assignments     README.md       example.py
julians-mbp:ESOF-322 myfatduck$ git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        example.py

nothing added to commit but untracked files present (use "git add" to track)
julians-mbp:ESOF-322 myfatduck$ git add example.py
julians-mbp:ESOF-322 myfatduck$ git commit -m "Adding a file to my repo"
[master 1b30437] Adding a file to my repo
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 example.py
julians-mbp:ESOF-322 myfatduck$ git push origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 286 bytes | 286.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:julianryorex/ESOF-322.git
   cbd8f3e..1b30437  master -> master
julians-mbp:ESOF-322 myfatduck$
```