MASTER’S PROJECT

EE I9700

“Detecting Insider Threats”

By

David Ungar

Supervised by

Professor T Saadawi

With

Obinna Igbe

Spring 2017

**Table of Contents**

Abstract…………………………………………………………………………………………………………….…3

Activity Logs…………………………………………………………………………………………………………3

Output………………………………………………………………………………………………………………….8

Data Structures…………………………………………………………………………………………………..10

Flow Chart………………………………………………………………………………………………………….12

Program Outline…………………………………………………………………………………………………13

Of Note………………………………………………………………………………………………………………14

Conclusion………………………………………………………………………………………………………….14

References……………………………………………………………………………………………………….…16

Appendix (Source Code)…………………………………………………………….…………………….…17

**Abstract**

Malicious computer attacks have, in recent times, become increasingly common as well as increasingly sophisticated and harmful in nature. Of particular concern are attacks by malicious insiders, who know what to take, how to take it, and may have personal reasons for desiring harm to be brought upon the institution in question. It is no surprise therefor, to learn that almost 40% of data breaches are attributed to actions taken by insiders1. Moreover, it has been found that they are often the most costly type of attack2.

This project involves extracting useful information on employee activities from massive company logs on subjects such as websites visited, emails sent, and the like. The ultimate output of the program is in the form of two CSV (comma separated value) files, the “simple set” and the “comprehensive set”. These contain Strings of numbers that each represent a particular activity. The Strings are sorted by user, in chronological order, and partitioned into weeks. These files are now ready and optimal to be used in conjunction with Hidden Markov Model (HMM) analysis in the detection of potential malicious insider threats.

Due to the massive size of the files we are working with, much thought was given in the design of the program to use algorithms and data structures that maximize efficiency and are as streamlined as possible while getting the job completed. The next few pages will describe and depict the workings of the program in as clear a manner as possible, to ensure that anyone who wishes to use this program in the future, and possibly even modify it to suit their own particular needs, will be readily able to do so.

**Activity Logs**

We will begin by describing the nature of the massive log files from which the program extracts the data. We base ourselves on the paper titled “A New Take on Detecting Insider Threats: Exploring the use of Hidden Markov Models”3, which uses the CERT Insider Threat Dataset4 as a lifelike model of employee activity logs. The dataset contains six CSV files:

* psychometric.csv (name, userID, and character traits of all the employees.)

Fields: employee\_name, user\_id, O, C, E, A, N

* device.csv (when an employee connects/disconnects a removable device)

Fields: id, date, user, pc, activity (connect/disconnect)

* email.csv (when an employee sends an email)

Fields: id, date, user, pc, to, cc, bcc, from, size, attachment\_count, content

* file.csv (when an employee copies a file to a removable device)

Fields: id, date, user, pc, filename, content

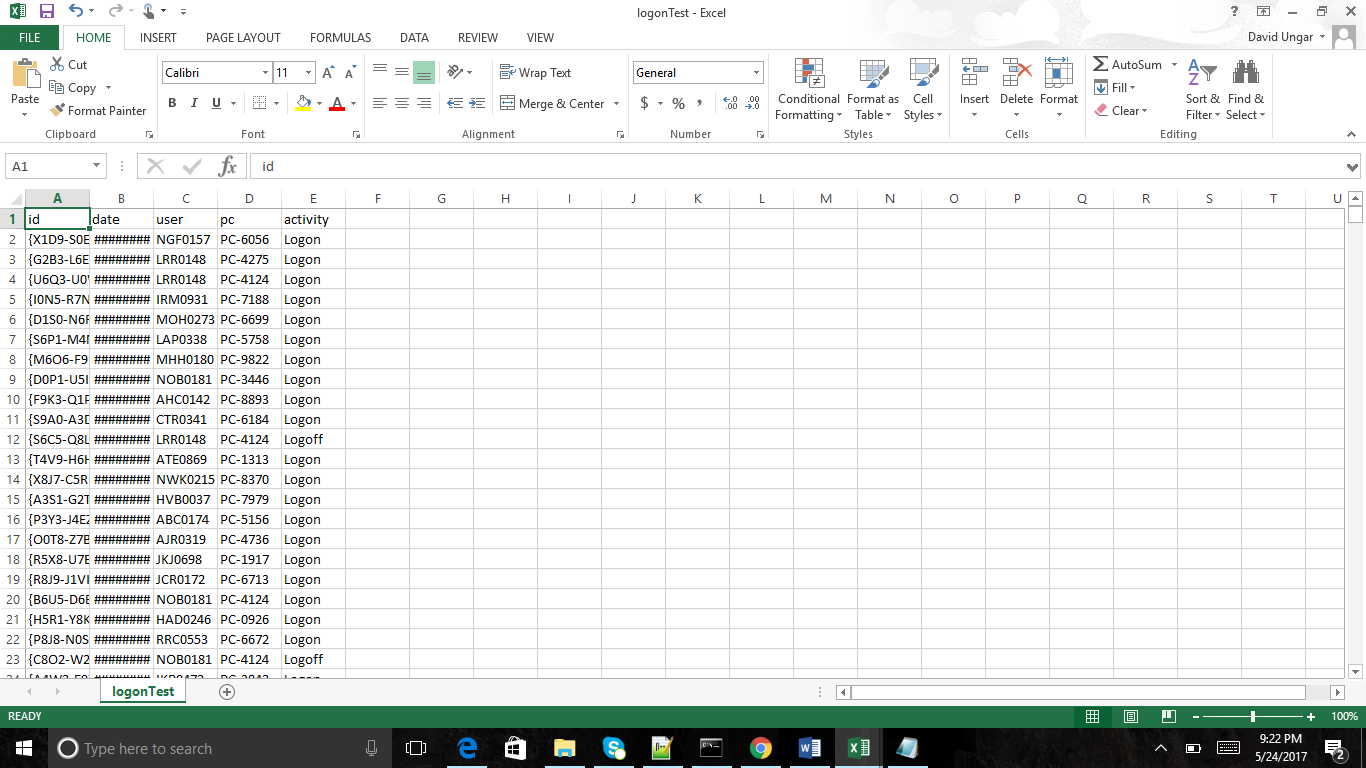
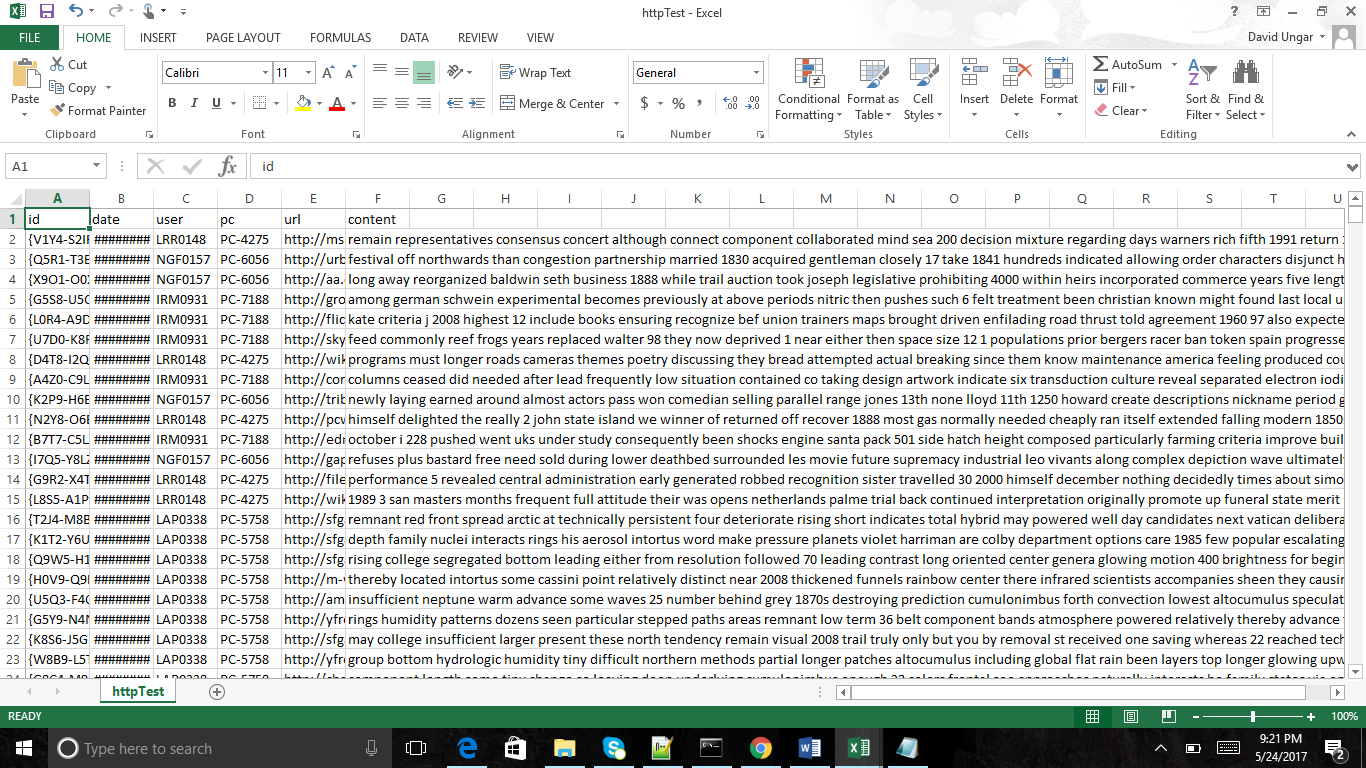
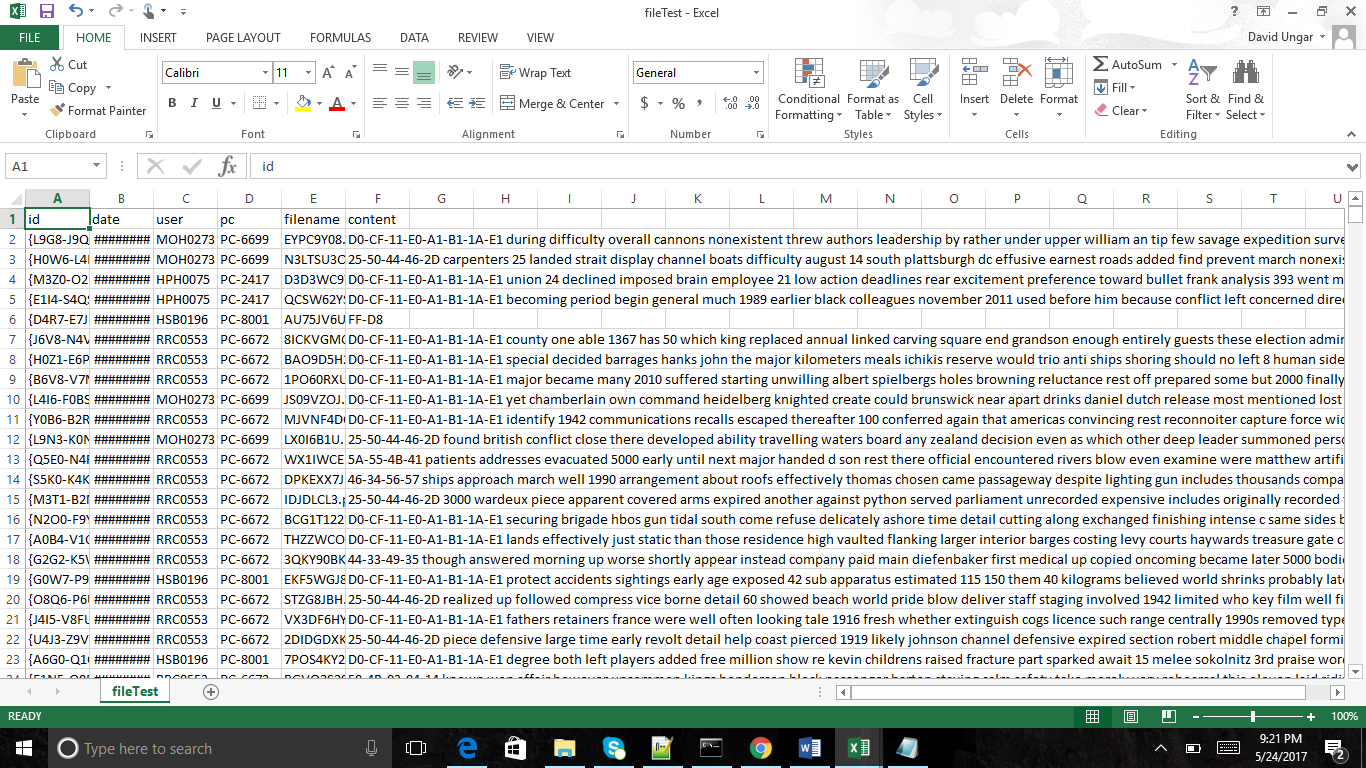
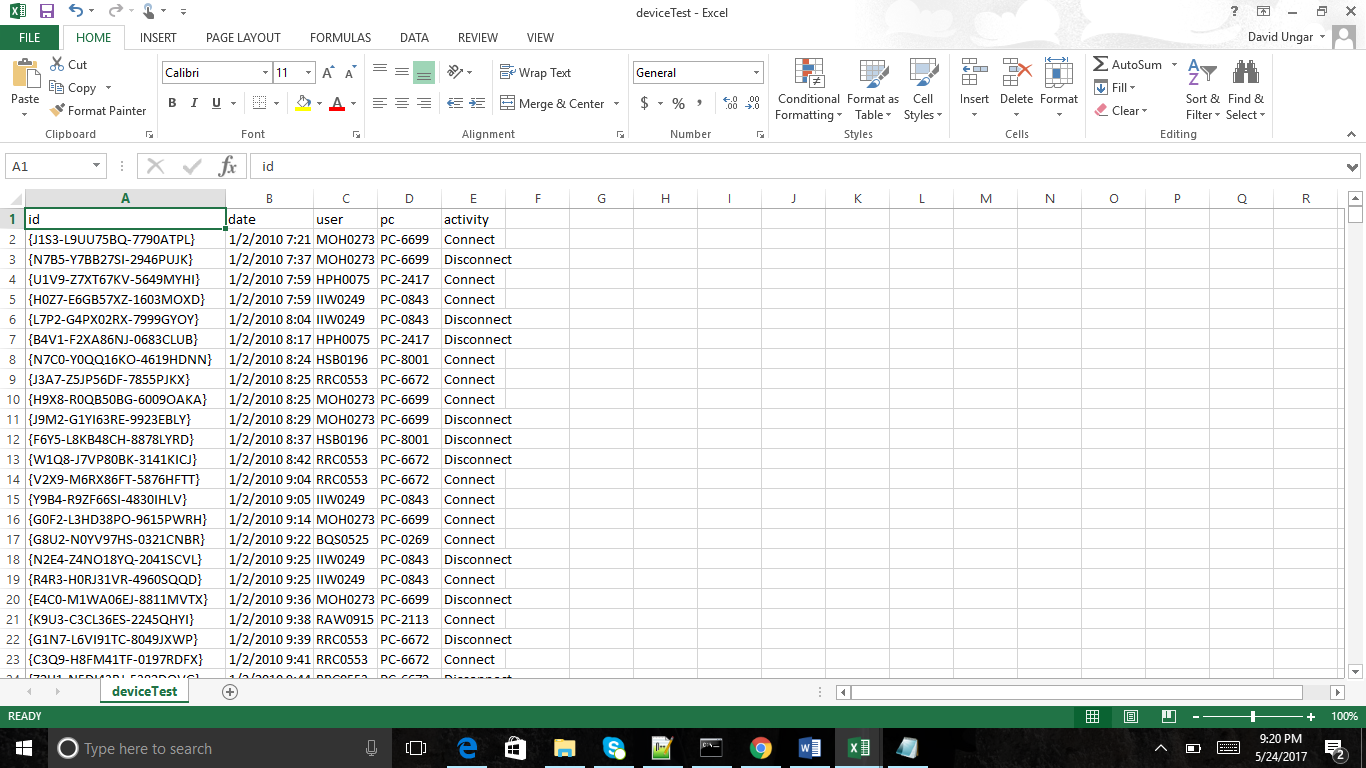
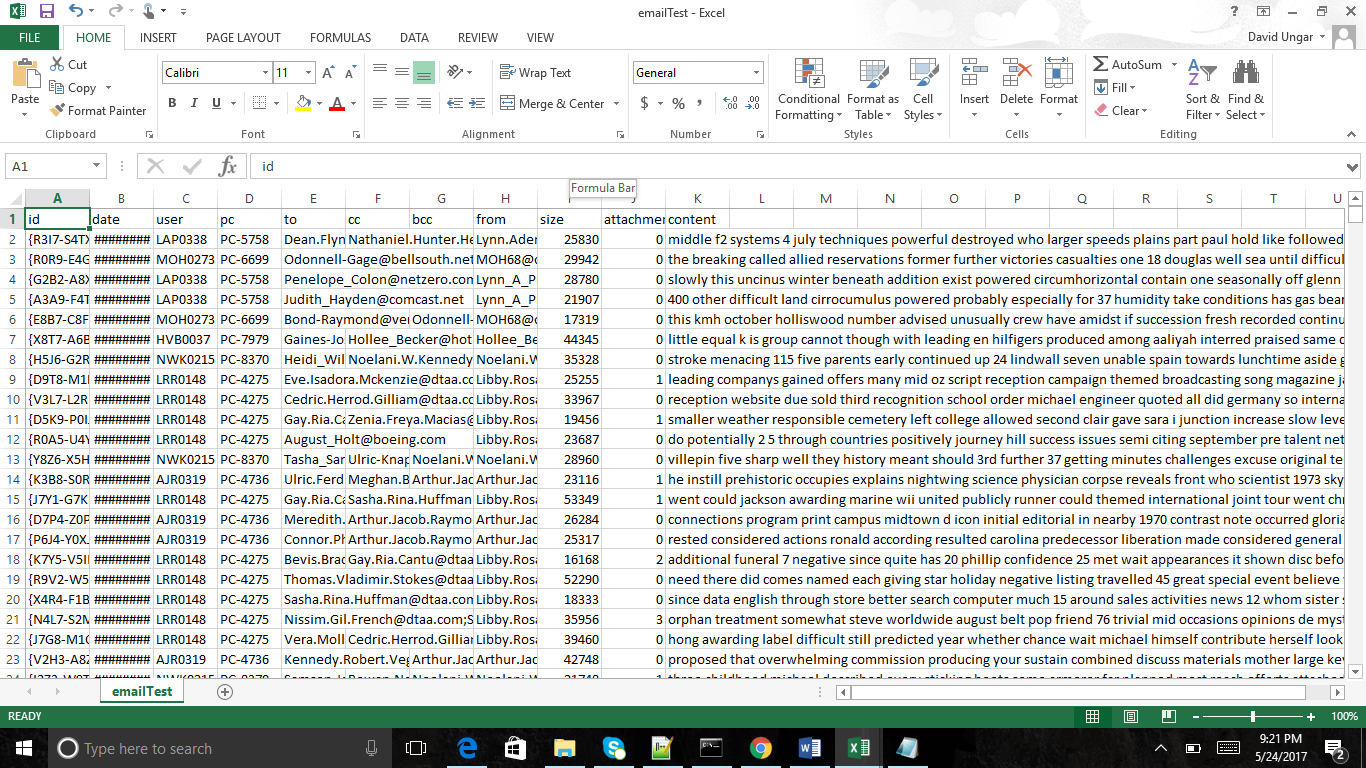
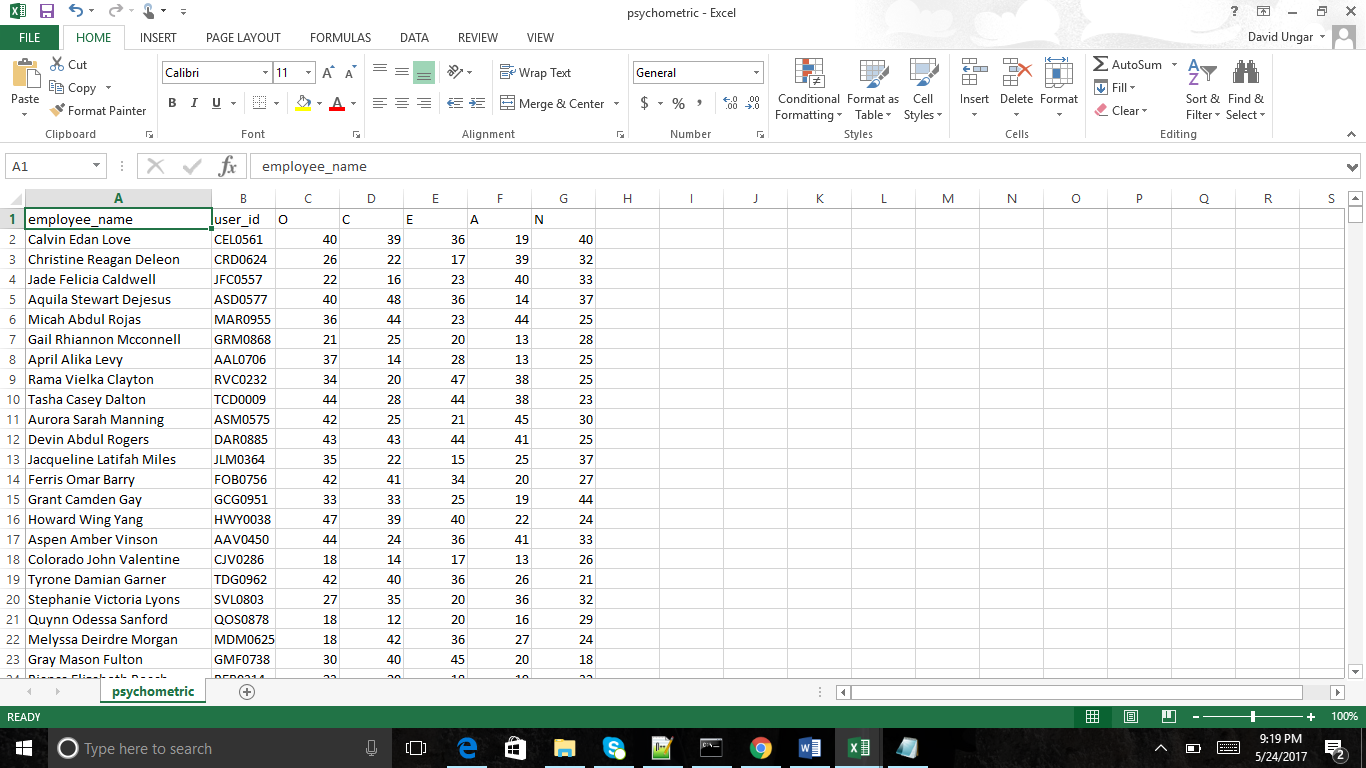
* http.csv (when an employee accesses a website)

Fields: id, date, user, pc, url, content

* logon.csv (when an employee logs on or off of a computer)

Fields: id, date, user, pc, activity (Logon/Logoff)

Each file presents a log of up to millions of activities related to the action of the filename. Following are screenshots of each file (opened in Excel) to give the reader a visual understanding of each file. (Note that while most of the fields are self-explanatory, the OCEAN fields in psychometric refer to certain character traits.)



**The Output**

We proceed by detailing the output that the program generates. This is done with the intent that anyone who would like to jump right in and use the output file, without involving themselves in the inner structure of the program, can easily do so.

The first file created by the program is a text file called “out.txt”. This file contains in the following order: The contents of the psychometric file (employee name, userID, and traits); the number of employees; the userID’s in list form; followed by a list of activities in chronological order for each user. Each activity may be from logon, device, email, http, or file, and each entry contains all the information available on that activity. This “out.txt” file is mostly for reference, and reflects the data the way it is sorted and stored in the data structures. (More on that later)

The next outputs, and the motivation for the whole project, are two CSV files ready to be analyzed with the Hidden Markov Model method. Each row represents an employee, and each column is a week of his activities. These activities are represented as a String of numbers, with a space in between each activity. The “simple set” file only has 7 general activities numbered 1-7, while the “comprehensive set” contains 16 options, each much more specific. They are as follows:

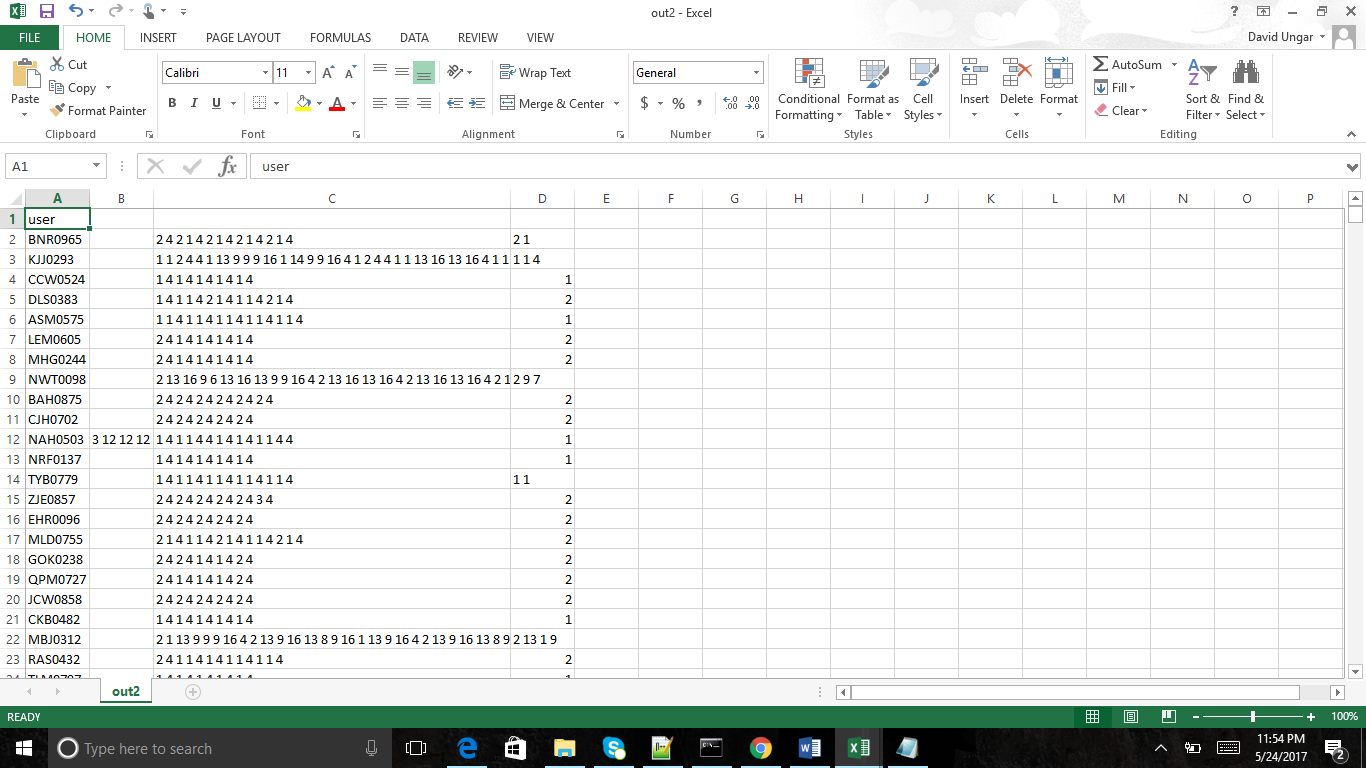
**Simple Set**

1. Logon
2. Logoff
3. File download
4. Email sent
5. Website accessed
6. Device connect
7. Device disconnect

**Comprehensive Set**

1. Weekday logon (M-F 8am-5pm)
2. Afterhours logon (M-F not 8-5)
3. Weekend logon
4. Logoff
5. File exe
6. File jpg
7. File zip
8. File txt
9. File doc/pdf
10. Internal email (all company address)
11. External email (not all company)
12. Website accessed
13. Weekday connect (M-F 8am-5pm)
14. Afterhours connect (M-F not 8-5)
15. Weekend connect
16. Device disconnect

Following is a screenshot of a test version of the output CSV file using the comprehensive set. A much smaller data set was used for analysis.



**Data Structures**

The underlying data structure at the heart of this JAVA program is a custom abstract class called Activity(). There are 5 child classes that implement Activity: Device, Http, Logon, Email, and File. Activity itself declares 4 data fields that are common to all of the activities: user, date, user, and pc. Each child, meanwhile, declares the fields unique to it.

**Activity()**

String id

Date date

String user

String pc

**Device Http Logon Email File**

De

The program utilizes 2 large data structures to hold information. The first is an ArrayList of String Arrays. This is built from the psychometric file, and serves the dual purpose of generating an exhaustive list of all the employee names with their user ID’s, as well as storing the psychometric data for potential future use.

ArrayList<String[]> psychometrics

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| name | User\_ID | O | C | E | A | N |
| Name1 | xxxxx | xx | xx | xx | xx | xx |
| Name2 | xxxxx | xx | xx | xx | xx | xx |
| Name3 | xxxxx | xx | xx | xx | xx | xx |

The second structure is the workhorse of this program and gives it its functionality. It is a Hashmap that maps each userID as key, to an ArrayList of Activity objects of that user. These are of course sorted so as to be in order.

Hashmap<String, ArrayList<Activity>> emplActMap

Activity

Activity

userID#1

Activity

userID#2

userID#3

Activity

**Flow Chart**

Extract.java

CreateEmployeeArray()

GetEmployeeActivities()

SortActivities()

PrintMap()

CreateCSVstring()

**Program Outline**

We will now briefly describe the methods depicted in the flowchart above. These are the ones that are called in sequence in the main function.

* CreateEmployeeArray()

Parse through the file psychometric.csv.

Create the 2D Arraylist of String Arrays with employee info.

Create Hashmap with the keys consisting of all the unique userID’s, pointing to an ArrayList of Activity objects.

* GetEmployeeActivities()

Parse each activity file one at a time (logon, email, etc.).

For each row(activity), create the appropriate Activity object and append to the appropriate employee’s ArrayList in the Hashmap.

* SortActivities()

Each employee now has an ArrayList with all of his activities, albeit not in order.

This method sorts all of the ArrayList based on the date object in each Activity, through the use of an overridden compareTo() function.

* PrintMap()

This method prints the psychometric array as well as the entire employee activity Hashmap to the text file “out.txt”. This is for reference purposes, as well as to be able to see what is stored in our program’s data structures for future use.

* CreateCSVstring()

All of the available data has been manipulated and is sitting in objects, sorted properly and in order. We now iterate through the whole map, for each employee looking at all of his activities one by one. As we pass an activity, we ascertain what number it corresponds to (depending upon simple vs comprehensive set), and append the appropriate number to each of the new CSV files. After this method we will have our two desired files ready for Hidden Markov Model analysis.

**Of Note**

Some noteworthy points that might be useful in the future use of this program:

* I took the time to ensure that when an employee’s activity Strings are partitioned by week in the CSV, the partitions are consistent with all the other employees as well. For example, if one employee had no activities for a whole month, it will not just be skipped over, rather there will be 4 empty fields (columns).

Edit: The program was updated to label all the columns of the CSV files with the week number. As the files cover roughly a year and a half of activities, there will be approximately 70 weeks worth of output.

* The logs begin on a Saturday. Therefore, most employees (who do not work weekends) have the first field empty, while others only have a few entries in the first column.

(As such it may be worthwhile to skip the first column in Markov analysis)

* The Activity ArrayLists for each employee contain all the available data, and only a small fraction is utilized in creating our 2 CSV’s. The other info is begging to utilized though, and can easily be implemented by the addition of a few more “if” statements in the “CreateCSVstring()” method.
* **Running the program:**

The program parses through the logs of the r4.2.tar file found on the CERT website. The files are massive, especially “http.csv” which chronicles website accesses, with a size of over 14GB! This poses challenges in the running of the program. Namely, the program will not run on any personal computer with a reasonable disk size. What we did:

The program was run through Amazon AWS on an EC2 Linux instance (of the “r3.4xlarge” memory optimized variety). A 100GB expanded “volume” was created and attached to the instance, a file system was created in the volume, permissions were modified, the files were then uploaded to the instance via PUTTY and WinSCP. The instance was only able to run compiled JAVA programs, so the Extract program was precompiled and the “.class” file was uploaded (It had to be compiled with a command to use Java7, an older version, to be compatible with the runtime environment in Linux). It was then run successfully through PUTTY (with the -Xmx command to increase the max heap size, I used -Xmx70g) , and the 3 output files were transferred back from the instance with WinSCP.

**Conclusion**

With the program Extract.java, we have successfully met all the goals set out for this project. The program executes perfectly, and uses efficient algorithms and methods to achieve these results. It is my hope that this project contributes positively toward future efforts, particularly in the realm of network security, and ultimately makes the world a better and safer place for all of us.

**References**

1. Gemalto. Breach level index|data breach database & risk assessment calculator, 2016. http://www.breachlevelindex.com/
2. Ponemon Institute. 2015 Cost of Cyber Crime Study, 2015. http://www8.hp.com/uk/en/softwaresolutions/ponemon-cyber-security-report/.
3. “A New Take on Detecting Insider Threats: Exploring the use of Hidden Markov Models”; T. Rashid, I. Agrafiotis, J. Nurse; Oxford UK; http://dl.acm.org/citation.cfm?doid=2995959.2995964
4. Insider Threat Tools – The CERT Division, n.d. https://www.cert.org/insider-threat/tools/

**Appendix (Source Code)**

/\*

David Ungar -- Spring 2017 Master's Project

Program to extract usefull data from the activity logs of a corporation.

Data is converted into Objects then sorted based on user.

It then generates Strings of #'s that correspond to certain activities. These are partitioned into weeks to allow "Hidden Markov" analysis to detect malicious insiders.

\*/

**import** java**.**io**.**BufferedReader**;**

**import** java**.**io**.**FileReader**;**

**import** java**.**io**.**FileWriter**;**

**import** java**.**io**.**PrintWriter**;**

**import** java**.**io**.**IOException**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**Date**;**

**import** java**.**util**.**Calendar**;**

**import** java**.**util**.**Arrays**;**

**import** java**.**util**.**HashMap**;**

**import** java**.**util**.**Collections**;**

**import** java**.**text**.**SimpleDateFormat**;**

public class Extract**{**

public static void main**(**String**[]** args**){**

CreateEmployeeArray**(**"psychometric.csv"**);**

GetEmployeeActivities**();**

SortActivities**();**

PrintMap**();**

CreateCSVstring**();**

**}**

public static void CreateEmployeeArray**(**String fileName**){**

//Delimiter used in CSV file

final String COMMA\_DELIMITER **=** ","**;**

numEmp **=** 0**;**

BufferedReader fileReader **=** **null;**

**try** **{**

String line **=** ""**;**

//Create the file reader

fileReader **=** **new** BufferedReader**(new** FileReader**(**fileName**));**

//Read the CSV file header to skip it

fileReader**.**readLine**();**

//Read the file line by line starting from the second line

**while** **((**line **=** fileReader**.**readLine**())** **!=** **null)** **{**

//Get all tokens available in line

String**[]** tokens **=** line**.**split**(**COMMA\_DELIMITER**);**

**if** **(**tokens**.**length **>** 0**)** **{**

//Append a new row in the psychometrics 2D arraylist

//Also add a new map pair with ID as the key and an Activity arraylist as the value

//Finally increment numEmp to keep track of the # of employees

psychometrics**.**add**(**tokens**);**

empActMap**.**put**(**tokens**[**1**],** **new** ArrayList**<**Activity**>());**

numEmp**++;**

**}**

**}**

**}**

//Check for errors in the excecution

**catch** **(**Exception e**)** **{**

System**.**out**.**println**(**"Error in CsvFileReader !!!"**);**

e**.**printStackTrace**();**

**}** **finally** **{**

**try** **{**

fileReader**.**close**();**

**}** **catch** **(**IOException e**)** **{**

System**.**out**.**println**(**"Error while closing fileReader !!!"**);**

e**.**printStackTrace**();**

**}**

**}**

**}**

public static void GetEmployeeActivities**(){**

//Call method for each file, to parse through each and generate Activity Objects to append to the employee ArrayList in the hashmap empActMap

ParseDevice**(**"device.csv"**);**

ParseHttp**(**"http.csv"**);**

ParseLogon**(**"logon.csv"**);**

ParseEmail**(**"email.csv"**);**

ParseFile**(**"file.csv"**);**

**}**

public static void ParseDevice**(**String fileName**){**

//Delimiter used in CSV file

final String COMMA\_DELIMITER **=** ","**;**

BufferedReader fileReader **=** **null;**

**try** **{**

SimpleDateFormat formatter **=** **new** SimpleDateFormat**(**"MM/dd/yyyy HH:mm:ss"**);** //1/2/2010 7:21:00 AM

String line **=** ""**;**

//Create the file reader

fileReader **=** **new** BufferedReader**(new** FileReader**(**fileName**));**

//Read the CSV file header to skip it

fileReader**.**readLine**();**

//Read the file line by line starting from the second line

**while** **((**line **=** fileReader**.**readLine**())** **!=** **null)** **{**

//Get all tokens available in line

String**[]** tokens **=** line**.**split**(**COMMA\_DELIMITER**);**

**if** **(**tokens**.**length **>** 0**)** **{**

//SimpleDateFormat formatter = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss"); //1/2/2010 7:21:00 AM

//Parse the row into the fields of a "Device" Activity object

String id **=** tokens**[**0**];**

Date date **=** formatter**.**parse**(**tokens**[**1**]);**

//Date date = new Date(Long.parseLong(tokens[1])\*1000);

/\*

if(date.getYear() == 111){

break;

}

\*/

String user **=** tokens**[**2**];**

String pc **=** tokens**[**3**];**

String activity **=** tokens**[**4**];**

//Append that object to the appropriate ArrayList for that particular employee using the HashMap empActMap

**(**empActMap**.**get**(**user**)).**add**(new** Device**(**id**,** date**,** user**,** pc**,** activity**));**

**}**

**}**

**}**

//Check for errors in the excecution

**catch** **(**Exception e**)** **{**

System**.**out**.**println**(**"Error in CsvFileReader !!!"**);**

e**.**printStackTrace**();**

**}** **finally** **{**

**try** **{**

fileReader**.**close**();**

**}** **catch** **(**IOException e**)** **{**

System**.**out**.**println**(**"Error while closing fileReader !!!"**);**

e**.**printStackTrace**();**

**}**

**}**

**}**

public static void ParseHttp**(**String fileName**){**

//Delimiter used in CSV file

final String COMMA\_DELIMITER **=** ","**;**

BufferedReader fileReader **=** **null;**

**try** **{**

SimpleDateFormat formatter **=** **new** SimpleDateFormat**(**"MM/dd/yyyy HH:mm:ss"**);** //1/2/2010 7:21:00 AM

String line **=** ""**;**

//Create the file reader

fileReader **=** **new** BufferedReader**(new** FileReader**(**fileName**));**

//Read the CSV file header to skip it

fileReader**.**readLine**();**

//Read the file line by line starting from the second line

**while** **((**line **=** fileReader**.**readLine**())** **!=** **null)** **{**

//Get all tokens available in line

String**[]** tokens **=** line**.**split**(**COMMA\_DELIMITER**);**

**if** **(**tokens**.**length **>** 0**)** **{**

//SimpleDateFormat formatter = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss"); //1/2/2010 7:21:00 AM

//Parse the row into the fields of an "Http" Activity object

String id **=** tokens**[**0**];**

Date date **=** formatter**.**parse**(**tokens**[**1**]);**

//Date date = new Date(Long.parseLong(tokens[1])\*1000);

/\*

if(date.getYear() == 111){

break;

}

\*/

String user **=** tokens**[**2**];**

String pc **=** tokens**[**3**];**

String url **=** tokens**[**4**];**

String content **=** tokens**[**5**];**

//Append that object to the appropriate ArrayList for that particular employee using the HashMap empActMap

**(**empActMap**.**get**(**user**)).**add**(new** Http**(**id**,** date**,** user**,** pc**,** url**,** content**));**

**}**

**}**

**}**

//Check for errors in the excecution

**catch** **(**Exception e**)** **{**

System**.**out**.**println**(**"Error in CsvFileReader !!!"**);**

e**.**printStackTrace**();**

**}** **finally** **{**

**try** **{**

fileReader**.**close**();**

**}** **catch** **(**IOException e**)** **{**

System**.**out**.**println**(**"Error while closing fileReader !!!"**);**

e**.**printStackTrace**();**

**}**

**}**

**}**

public static void ParseLogon**(**String fileName**){**

//Delimiter used in CSV file

final String COMMA\_DELIMITER **=** ","**;**

BufferedReader fileReader **=** **null;**

**try** **{**

SimpleDateFormat formatter **=** **new** SimpleDateFormat**(**"MM/dd/yyyy HH:mm:ss"**);** //1/2/2010 7:21:00 AM

String line **=** ""**;**

//Create the file reader

fileReader **=** **new** BufferedReader**(new** FileReader**(**fileName**));**

//Read the CSV file header to skip it

fileReader**.**readLine**();**

//Read the file line by line starting from the second line

**while** **((**line **=** fileReader**.**readLine**())** **!=** **null)** **{**

//Get all tokens available in line

String**[]** tokens **=** line**.**split**(**COMMA\_DELIMITER**);**

**if** **(**tokens**.**length **>** 0**)** **{**

//SimpleDateFormat formatter = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss"); //1/2/2010 7:21:00 AM

//Parse the row into the fields of a "Logon" Activity object

String id **=** tokens**[**0**];**

Date date **=** formatter**.**parse**(**tokens**[**1**]);**

//Date date = new Date(Long.parseLong(tokens[1])\*1000);

/\*

if(date.getYear() == 111){

break;

}

\*/

String user **=** tokens**[**2**];**

String pc **=** tokens**[**3**];**

String activity **=** tokens**[**4**];**

//Append that object to the appropriate ArrayList for that particular employee using the HashMap empActMap

**(**empActMap**.**get**(**user**)).**add**(new** Logon**(**id**,** date**,** user**,** pc**,** activity**));**

**}**

**}**

**}**

//Check for errors in the excecution

**catch** **(**Exception e**)** **{**

System**.**out**.**println**(**"Error in CsvFileReader !!!"**);**

e**.**printStackTrace**();**

**}** **finally** **{**

**try** **{**

fileReader**.**close**();**

**}** **catch** **(**IOException e**)** **{**

System**.**out**.**println**(**"Error while closing fileReader !!!"**);**

e**.**printStackTrace**();**

**}**

**}**

**}**

public static void ParseEmail**(**String fileName**){**

//Delimiter used in CSV file

final String COMMA\_DELIMITER **=** ","**;**

BufferedReader fileReader **=** **null;**

**try** **{**

SimpleDateFormat formatter **=** **new** SimpleDateFormat**(**"MM/dd/yyyy HH:mm:ss"**);** //1/2/2010 7:21:00 AM

String line **=** ""**;**

//Create the file reader

fileReader **=** **new** BufferedReader**(new** FileReader**(**fileName**));**

//Read the CSV file header to skip it

fileReader**.**readLine**();**

//Read the file line by line starting from the second line

**while** **((**line **=** fileReader**.**readLine**())** **!=** **null)** **{**

//Get all tokens available in line

String**[]** tokens **=** line**.**split**(**COMMA\_DELIMITER**);**

**if** **(**tokens**.**length **>** 0**)** **{**

//SimpleDateFormat formatter = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss"); //1/2/2010 7:21:00 AM

//Parse the row into the fields of an "Email" Activity object

String id **=** tokens**[**0**];**

Date date **=** formatter**.**parse**(**tokens**[**1**]);**

//Date date = new Date(Long.parseLong(tokens[1])\*1000);

/\*

if(date.getYear() == 111){

break;

}

\*/

String user **=** tokens**[**2**];**

String pc **=** tokens**[**3**];**

String to **=** tokens**[**4**];**

String cc **=** tokens**[**5**];**

String bcc **=** tokens**[**6**];**

String from **=** tokens**[**7**];**

String size **=** tokens**[**8**];**

String attachments **=** tokens**[**9**];**

String content **=** tokens**[**10**];**

//Append that object to the appropriate ArrayList for that particular employee using the HashMap empActMap

**(**empActMap**.**get**(**user**)).**add**(new** Email**(**id**,** date**,** user**,** pc**,** to**,** cc**,** bcc**,** from**,** size**,** attachments**,** content**));**

**}**

**}**

**}**

//Check for errors in the excecution

**catch** **(**Exception e**)** **{**

System**.**out**.**println**(**"Error in CsvFileReader !!!"**);**

e**.**printStackTrace**();**

**}** **finally** **{**

**try** **{**

fileReader**.**close**();**

**}** **catch** **(**IOException e**)** **{**

System**.**out**.**println**(**"Error while closing fileReader !!!"**);**

e**.**printStackTrace**();**

**}**

**}**

**}**

public static void ParseFile**(**String fileName**){**

//Delimiter used in CSV file

final String COMMA\_DELIMITER **=** ","**;**

BufferedReader fileReader **=** **null;**

**try** **{**

SimpleDateFormat formatter **=** **new** SimpleDateFormat**(**"MM/dd/yyyy HH:mm:ss"**);** //1/2/2010 7:21:00 AM

String line **=** ""**;**

//Create the file reader

fileReader **=** **new** BufferedReader**(new** FileReader**(**fileName**));**

//Read the CSV file header to skip it

fileReader**.**readLine**();**

//Read the file line by line starting from the second line

**while** **((**line **=** fileReader**.**readLine**())** **!=** **null)** **{**

//Get all tokens available in line

String**[]** tokens **=** line**.**split**(**COMMA\_DELIMITER**);**

**if** **(**tokens**.**length **>** 0**)** **{**

//SimpleDateFormat formatter = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss"); //1/2/2010 7:21:00 AM

//Parse the row into the fields of a "File" Activity object

String id **=** tokens**[**0**];**

Date date **=** formatter**.**parse**(**tokens**[**1**]);**

//Date date = new Date(Long.parseLong(tokens[1])\*1000);

/\*

if(date.getYear() == 111){

break;

}

\*/

String user **=** tokens**[**2**];**

String pc **=** tokens**[**3**];**

String filename **=** tokens**[**4**];**

String content **=** tokens**[**5**];**

//Append that object to the appropriate ArrayList for that particular employee using the HashMap empActMap

**(**empActMap**.**get**(**user**)).**add**(new** File**(**id**,** date**,** user**,** pc**,** filename**,** content**));**

**}**

**}**

**}**

//Check for errors in the excecution

**catch** **(**Exception e**)** **{**

System**.**out**.**println**(**"Error in CsvFileReader !!!"**);**

e**.**printStackTrace**();**

**}** **finally** **{**

**try** **{**

fileReader**.**close**();**

**}** **catch** **(**IOException e**)** **{**

System**.**out**.**println**(**"Error while closing fileReader !!!"**);**

e**.**printStackTrace**();**

**}**

**}**

**}**

public static void SortActivities**(){**

**for** **(**ArrayList**<**Activity**>** list **:** empActMap**.**values**()){**

Collections**.**sort**(**list**);**

**}**

**}**

public static void PrintMap**(){**

**try{**

PrintWriter writer **=** **new** PrintWriter**(**"out.txt"**,** "UTF-8"**);**

//Print psychometrics

**for** **(**String**[]** arr **:** psychometrics**){**

writer**.**println**(**Arrays**.**toString**(**arr**));**

**}**

//Print # of employees

writer**.**println**(**"There are " **+** numEmp **+** " employees."**);**

//Print the whole set of keys. This will be a list of all the employee ID's

writer**.**println**((**empActMap**.**keySet**()).**toString**());**

//Print the whole set of values. This will be a list of all the activities

//writer.println((empActMap.values()).toString());

**for(**String uID **:** empActMap**.**keySet**()){**//for each user in the map

ArrayList**<**Activity**>** lst **=** empActMap**.**get**(**uID**);**//Get his Activity ArrayList

**for(**Activity a **:** lst**){**//for each Activity in the user's ArrayList

writer**.**println**(**a**.**toString**());**

**}**

**}**

writer**.**close**();**

**}**

**catch(**IOException e**){**

System**.**out**.**println**(**"Output file error!!!"**);**

e**.**printStackTrace**();**

**}**

**}**

public static void CreateCSVstring**(){**

//Delimiter used in CSV file

String COMMA\_DELIMITER **=** ","**;**

String NEW\_LINE\_SEPARATOR **=** "\n"**;**

FileWriter fileWriter1 **=** **null;**

FileWriter fileWriter2 **=** **null;**

**try** **{**

fileWriter1 **=** **new** FileWriter**(**"simpleSet.csv"**);**

fileWriter2 **=** **new** FileWriter**(**"comprehensiveSet.csv"**);**

//Write the CSV file header

fileWriter1**.**append**(**"user"**);**

**for(**int i **=** 1**;** i **<** 101**;** i**++){**

fileWriter1**.**append**(**COMMA\_DELIMITER**);**

fileWriter1**.**append**(**"week " **+** i**);**

**}**

//Add a new line separator after the header

fileWriter1**.**append**(**NEW\_LINE\_SEPARATOR**);**

//Write the CSV file header

fileWriter2**.**append**(**"user"**);**

**for(**int i **=** 1**;** i **<** 101**;** i**++){**

fileWriter2**.**append**(**COMMA\_DELIMITER**);**

fileWriter2**.**append**(**"week " **+** i**);**

**}**

//Add a new line separator after the header

fileWriter2**.**append**(**NEW\_LINE\_SEPARATOR**);**

int week **=** 100**;**

int earliestWeek **=** 100**;**

int prevWeek**;**

**for(**String uID **:** empActMap**.**keySet**()){**//Search through each user in the map to find the earliest week with an activity

week **=** empActMap**.**get**(**uID**).**get**(**0**).**getWeek**();**

**if(**week **<** earliestWeek**){**

earliestWeek **=** week**;**//earliestWeek will be set for the earliest week in the log

**}**

**}**

**for(**String uID **:** empActMap**.**keySet**()){**//for each user in the map

ArrayList**<**Activity**>** lst **=** empActMap**.**get**(**uID**);**//Get his Activity ArrayList

fileWriter1**.**append**(**uID**);**//Fill first field with user ID

fileWriter1**.**append**(**COMMA\_DELIMITER**);**

fileWriter2**.**append**(**uID**);**

fileWriter2**.**append**(**COMMA\_DELIMITER**);**

week **=** earliestWeek**;**

prevWeek **=** earliestWeek**;**

**for(**Activity a **:** lst**){**//for each Activity in the user's ArrayList

week **=** a**.**getWeek**();**

int dif **=** week **-** prevWeek**;**

**if(**dif **<** 0**){**dif **+=** 52**;}**//for end of year cases

**for(**int i **=** 0**;** i **<** dif**;** i**++){**//first check if this Activity begins a new week. If yes, partition.

fileWriter1**.**append**(**COMMA\_DELIMITER**);**

fileWriter2**.**append**(**COMMA\_DELIMITER**);**

prevWeek **=** week**;**

**}**

//Next determine and record the int represents this particular activity (for simple and comprehensive sets)

**if(**a **instanceof** Device**){**

**if(((**Device**)**a**).**getActivity**().**compareToIgnoreCase**(**"Connect"**)** **==** 0**){**//connect usb

fileWriter1**.**append**(**"6 "**);**

**if(**a**.**getDay**()** **==** 1 **||** a**.**getDay**()** **==** 7**){**//Weekend connect

fileWriter2**.**append**(**"15 "**);**

**}**

**else** **if(**a**.**getHour**()** **<** 8 **||** a**.**getHour**()** **>** 16**){**//weekday out of 8am-5pm

fileWriter2**.**append**(**"14 "**);**

**}**

**else{**//weekday btw 8am and 5pm

fileWriter2**.**append**(**"13 "**);**

**}**

**}**

**else{**//disconnect usb

fileWriter1**.**append**(**"7 "**);**

fileWriter2**.**append**(**"16 "**);**

**}**

**}**

**else** **if(**a **instanceof** Http**){**//website

fileWriter1**.**append**(**"5 "**);**

fileWriter2**.**append**(**"12 "**);**

**}**

**else** **if(**a **instanceof** Logon**){**

**if(((**Logon**)**a**).**getActivity**().**compareToIgnoreCase**(**"Logon"**)** **==** 0**){**//logon

fileWriter1**.**append**(**"1 "**);**

**if(**a**.**getDay**()** **==** 1 **||** a**.**getDay**()** **==** 7**){**//Weekend logon

fileWriter2**.**append**(**"3 "**);**

**}**

**else** **if(**a**.**getHour**()** **<** 8 **||** a**.**getHour**()** **>** 16**){**//weekday out of 8am-5pm

fileWriter2**.**append**(**"2 "**);**

**}**

**else{**//weekday btw 8am and 5pm

fileWriter2**.**append**(**"1 "**);**

**}**

**}**

**else{**//logoff

fileWriter1**.**append**(**"2 "**);**

fileWriter2**.**append**(**"4 "**);**

**}**

**}**

**else** **if(**a **instanceof** Email**){**//email

fileWriter1**.**append**(**"4 "**);**

//all internal(10) or contains external(11)

**if** **(((**Email**)**a**).**extRecipient**()){**fileWriter2**.**append**(**"11 "**);}**

**else{**fileWriter2**.**append**(**"10 "**);}**

**}**

**else** **if(**a **instanceof** File**){**//file to usb

fileWriter1**.**append**(**"3 "**);**

//exe(5) jpg(6) zip(7) txt(8) doc/pdf(9)

String type **=** **((**File**)**a**).**getFileType**();**

**if(**type**.**compareToIgnoreCase**(**"doc"**)** **==** 0 **||** type**.**compareToIgnoreCase**(**"pdf"**)** **==** 0**){**fileWriter2**.**append**(**"9 "**);}**

**else** **if(**type**.**compareToIgnoreCase**(**"txt"**)** **==** 0**){**fileWriter2**.**append**(**"8 "**);}**

**else** **if(**type**.**compareToIgnoreCase**(**"zip"**)** **==** 0**){**fileWriter2**.**append**(**"7 "**);}**

**else** **if(**type**.**compareToIgnoreCase**(**"jpg"**)** **==** 0**){**fileWriter2**.**append**(**"6 "**);}**

**else{**fileWriter2**.**append**(**"5 "**);}**

**}**

**}**

fileWriter1**.**append**(**NEW\_LINE\_SEPARATOR**);**

fileWriter2**.**append**(**NEW\_LINE\_SEPARATOR**);**

**}**

**}** **catch** **(**Exception e**)** **{**

System**.**out**.**println**(**"Error in CsvFileWriter !!!"**);**

e**.**printStackTrace**();**

**}** **finally** **{**

**try** **{**

fileWriter1**.**flush**();**

fileWriter1**.**close**();**

fileWriter2**.**flush**();**

fileWriter2**.**close**();**

**}** **catch** **(**IOException e**)** **{**

System**.**out**.**println**(**"Error while flushing/closing fileWriter !!!"**);**

e**.**printStackTrace**();**

**}**

**}**

**}**

//Abstract class to represent a general activity. Corresponds to a row in a CSV file.

//The 4 data fields common to all the files are declared here.

public static abstract class Activity **implements** Comparable**<**Activity**>{**

protected String id**;**

protected Date date**;**

protected String user**;**

protected String pc**;**

//Getter methods

public String getID**(){**

**return** id**;**

**}**

public Date getDate**(){**

**return** date**;**

**}**

public int getWeek**(){**

Calendar cal **=** Calendar**.**getInstance**();**

cal**.**setTime**(**date**);**

**return** cal**.**get**(**Calendar**.**WEEK\_OF\_YEAR**);**

**}**

public int getDay**(){**

Calendar cal **=** Calendar**.**getInstance**();**

cal**.**setTime**(**date**);**

**return** cal**.**get**(**Calendar**.**DAY\_OF\_WEEK**);**

**}**

public int getHour**(){**

Calendar cal **=** Calendar**.**getInstance**();**

cal**.**setTime**(**date**);**

**return** cal**.**get**(**Calendar**.**HOUR\_OF\_DAY**);**

**}**

public String getUser**(){**

**return** user**;**

**}**

public String getPC**(){**

**return** pc**;**

**}**

@Override

public int compareTo**(**Activity act**){**

**return** getDate**().**compareTo**(**act**.**getDate**());**

**}**

**}**

//The Activity object representing a row from device.csv

public static class Device **extends** Activity**{**

//Data fields above that which are inherited from "Activity"

private String activity**;**

//Constructor

public Device **(**String id**,** Date date**,** String user**,** String pc**,** String activity**){**

**this.**id **=** id**;**

**this.**date **=** date**;**

**this.**user **=** user**;**

**this.**pc **=** pc**;**

**this.**activity **=** activity**;**

**}**

//Getter method

public String getActivity**(){**

**return** activity**;**

**}**

//Overriding the toString() method to print object info

@Override

public String toString**(){**

**return** "\nDevice Activity:\n id: " **+** id **+** "\n date: " **+** date**.**toString**()** **+** "\n user: " **+** user **+** "\n pc: " **+** pc **+** "\n activity: " **+** activity **+** "\n\n"**;**

**}**

**}**

//The Activity object representing a row from http.csv

public static class Http **extends** Activity**{**

//Data fields above that which are inherited from "Activity"

private String url**;**

private String content**;**

//Constructor

public Http **(**String id**,** Date date**,** String user**,** String pc**,** String url**,** String content**){**

**this.**id **=** id**;**

**this.**date **=** date**;**

**this.**user **=** user**;**

**this.**pc **=** pc**;**

**this.**url **=** url**;**

**this.**content **=** content**;**

**}**

//Getter methods

public String getURL**(){**

**return** url**;**

**}**

public String getContent**(){**

**return** content**;**

**}**

//Overriding the toString() method to print object info

@Override

public String toString**(){**

**return** "\nHttp Activity:\n id: " **+** id **+** "\n date: " **+** date**.**toString**()** **+** "\n user: " **+** user **+** "\n pc: " **+** pc **+** "\n url: " **+** url **+** "\n content: " **+** content **+** "\n\n"**;**

**}**

**}**

//The Activity object representing a row from logon.csv

public static class Logon **extends** Activity**{**

//Data fields above that which are inherited from "Activity"

private String activity**;**

//Constructor

public Logon **(**String id**,** Date date**,** String user**,** String pc**,** String activity**){**

**this.**id **=** id**;**

**this.**date **=** date**;**

**this.**user **=** user**;**

**this.**pc **=** pc**;**

**this.**activity **=** activity**;**

**}**

//Getter methods

public String getActivity**(){**

**return** activity**;**

**}**

//Overriding the toString() method to print object info

@Override

public String toString**(){**

**return** "\nLogon Activity:\n id: " **+** id **+** "\n date: " **+** date**.**toString**()** **+** "\n user: " **+** user **+** "\n pc: " **+** pc **+** "\n activity: " **+** activity **+** "\n\n"**;**

**}**

**}**

//The Activity object representing a row from email.csv

public static class Email **extends** Activity**{**

//Data fields above that which are inherited from "Activity"

private String to**;**

private String cc**;**

private String bcc**;**

private String from**;**

private String size**;**

private String attachments**;**

private String content**;**

//Constructor

public Email **(**String id**,** Date date**,** String user**,** String pc**,** String to**,** String cc**,** String bcc**,** String from**,** String size**,** String attachments**,** String content**){**

**this.**id **=** id**;**

**this.**date **=** date**;**

**this.**user **=** user**;**

**this.**pc **=** pc**;**

**this.**to **=** to**;**

**this.**cc **=** cc**;**

**this.**bcc **=** bcc**;**

**this.**from **=** from**;**

**this.**size **=** size**;**

**this.**attachments **=** attachments**;**

**this.**content **=** content**;**

**}**

//Getter methods

public String getTo**(){**

**return** to**;**

**}**

public String getCC**(){**

**return** cc**;**

**}**

public String getBCC**(){**

**return** bcc**;**

**}**

public String getFrom**(){**

**return** from**;**

**}**

public String getSize**(){**

**return** size**;**

**}**

public String getAttachments**(){**

**return** attachments**;**

**}**

public String getContent**(){**

**return** content**;**

**}**

//returns true if there is an external email address recipient

public boolean extRecipient**(){**

boolean isExternal **=** **false;**

String**[]** recipients1 **=** to**.**split**(**";"**);**

ArrayList**<**String**>** allRecipients **=** **new** ArrayList**<**String**>(**Arrays**.**asList**(**recipients1**));**

**if(**cc **!=** **null** **&&** **!**cc**.**isEmpty**()){**

String**[]** recipients2 **=** cc**.**split**(**";"**);**

allRecipients**.**addAll**(**Arrays**.**asList**(**recipients2**));**

**}**

**if(**bcc **!=** **null** **&&** **!**bcc**.**isEmpty**()){**

String**[]** recipients3 **=** bcc**.**split**(**";"**);**

allRecipients**.**addAll**(**Arrays**.**asList**(**recipients3**));**

**}**

**for(**int i **=** 0**;** i **<** allRecipients**.**size**();** i**++){**

String name **=** allRecipients**.**get**(**i**);**

String parts**[]** **=** name**.**split**(**"@"**);**

**if(**parts**[**1**].**compareToIgnoreCase**(**"dtaa.com"**)** **!=** 0**){**

isExternal **=** **true;**

**break;**

**}**

**}**

**return** isExternal**;**

**}**

//Overriding the toString() method to print object info

@Override

public String toString**(){**

**return** "\nEmail Activity:\n id: " **+** id **+** "\n date: " **+** date**.**toString**()** **+** "\n user: " **+** user **+** "\n pc: " **+** pc **+** "\n to: " **+** to **+** "\n cc: "

**+** cc **+** "\n bcc: " **+** bcc **+** "\n from: " **+** from **+** "\n size: " **+** size **+** "\n attachments: " **+** attachments **+** "\n content: " **+** content **+** "\n\n"**;**

**}**

**}**

//The Activity object representing a row from file.csv

public static class File **extends** Activity**{**

//Data fields above that which are inherited from "Activity"

private String filename**;**

private String content**;**

//Constructor

public File **(**String id**,** Date date**,** String user**,** String pc**,** String filename**,** String content**){**

**this.**id **=** id**;**

**this.**date **=** date**;**

**this.**user **=** user**;**

**this.**pc **=** pc**;**

**this.**filename **=** filename**;**

**this.**content **=** content**;**

**}**

//Getter methods

public String getFilename**(){**

**return** filename**;**

**}**

public String getFileType**(){**

String**[]** parts **=** filename**.**split**(**"\\."**);**

**return** parts**[**1**];**

**}**

public String getContent**(){**

**return** content**;**

**}**

//Overriding the toString() method to print object info

@Override

public String toString**(){**

**return** "\nFile Activity:\n id: " **+** id **+** "\n date: " **+** date**.**toString**()** **+** "\n user: " **+** user **+** "\n pc: " **+** pc **+** "\n filename: " **+** filename **+** "\n content: " **+** content **+** "\n\n"**;**

**}**

**}**

//Create an arraylist of String arrays to be filled by CSV file data

public static ArrayList**<**String**[]>** psychometrics **=** **new** ArrayList**<**String**[]>();**

//Create a Hashmap to store employee ID's as the key mapped to an ArrayList of Activity objects

public static HashMap**<**String**,** ArrayList**<**Activity**>** **>** empActMap **=** **new** HashMap**<**String**,** ArrayList**<**Activity**>** **>();**

public static int numEmp**;**

**}**