CSC 207 Assignment0 (DONE INDIVIDUALLY)
Due Date: 30th September 2015

Part 1)

You are first required to *checkout* Assignment0 starter code from your SVN repository.

a) Your SVN repository location is of the following form:

https://142.1.44.22/svn/csc207h/UTORID

where UTORID is your UTORID.

b) You are free to use any IDE that you wish.

c) If you plan on using Eclipse, then there is a short tutorial I have posted on Blackboard under Assignments folder that may be useful in explaining how to checkout the project using *Eclipse* and *Subclipse* (SVN client that integrates with Eclipse as a Eclipse Plugin). The link to the tutorial is reproduced here as well: http://goo.gl/tL4uYC
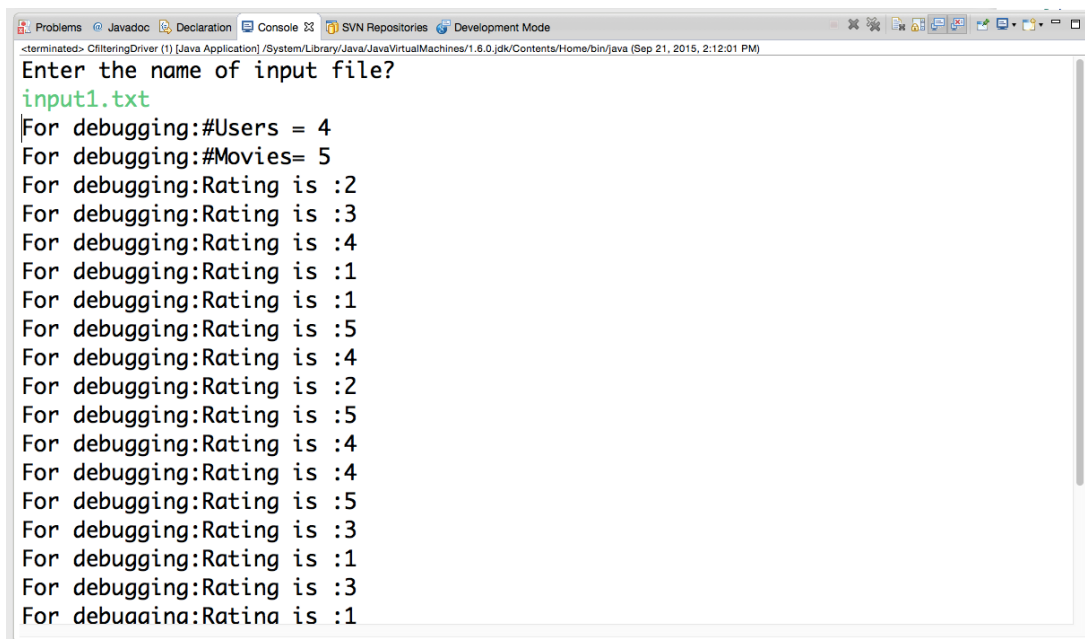
d) The last revision that is present in your SVN repository at time of due date; will be used for grading.

e) Once you have checked out your project, you will notice that there are two Java files.
> 1) `Cfiltering.java`
> 2) `CfilteringDriver.java`

You will use both these files to write your code.


```
Problems  @ Javadoc  Declaration  Console  SVN Repositories  Development Mode
<terminated> CfilteringDriver (1) [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Sep 21, 2015, 2:12:01 PM)
Enter the name of input file?
input1.txt
For debugging:#Users = 4
For debugging:#Movies= 5
For debugging:Rating is :2
For debugging:Rating is :3
For debugging:Rating is :4
For debugging:Rating is :1
For debugging:Rating is :1
For debugging:Rating is :5
For debugging:Rating is :4
For debugging:Rating is :2
For debugging:Rating is :5
For debugging:Rating is :4
For debugging:Rating is :4
For debugging:Rating is :5
For debugging:Rating is :3
For debugging:Rating is :1
For debugging:Rating is :3
For debugging:Rating is :1
```

Note: Without making any change to the code that you have just checked out. You should be able to **compile** and **run** your code as it is and be able to generate the following output: (input1.txt is also on your SVN server and is checked out as well if you follow the steps correctly on the slides posted under Assignment0 folder on Blackboard.)

Part 2) ONLY PROCEED WITH PART2, IF YOU HAVE THE OUTPUT AS SEEN IN PART1.
Write a Java program on Collaborative Filtering (`Cfiltering.java`)

Sites such as Amazon and Ebay, track-purchasing history of its users and shoppers. When the user logs in, these sites use the information to suggest products that may be of interest to you. Amazon for instance can recommend movies you might like, even if you have purchased books or music from it before.

The data by these sites can be mined in very different ways. Data can be (I like / I dislike), (yes/ no) vote, or as ratings (1 to 5) etc. On the other hand, dating websites try to match up people based on how similar they are. If Jack and Jill have similar interests, Jack may get a recommendation on his profile page *'Hi Jack, Jill has similar interests as you, do you like to send her a message OR add her as friend?'*

For this assignment i.e. assignment0 the input file will be of the following format: (Please do not deviate from this format). Note there are *line break*, *new line*, and *spaces*.
4
5

2 3 4 1 1
5 4 2 5 4
4 5 3 1 3
1 1 1 5 5

Where *'4' on the first line represents the number of users* and *'5' on the second line represents the number of movies*. The matrix that is followed is a **#user x #movie** (4x5) matrix. Where rows= #of users=4 and columns= #of movies=5.

The first row of (2 3 4 1 1) means that user1 has rated :
(movie1,movie2,movie3,movie4,movie5) as *(2,3,4,1,1).*

Likewise second row of 5,4,2,5,4 means that user2 has rated:
(movie1,movie2,movie3,movie4,movie5) as *(5,4,2,5,4).*

Your job is to find a similarity score between pair of users that tells how similar they are i.e. Are *user1* and *user2* more similar based on their rating of movies OR are *user3* and *user1* more similar? etc.

You can safely assume:

1) Number of users will be between: 3 ≤ # of users ≤ 9

2) Number of movies will be between: 3 ≤ # of movies ≤ 9.

3) Rating of movies will be in the range: 1 ≤ rating ≤ 5

Euclidean Distance Score (How to calculate similarity score?):

The basic idea behind Euclidean distance score is to calculate distance between any two points. For example, the two points are as follows:

1) Point1 (x,y)= (1,1)
2) Point2 (x,y)=(3,1)

<u>What is the distance between the above two points?</u>

What is the distance between the above two points?
$$distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$
$$distance = \sqrt{(3 - 1)^2 + (1 - 1)^2}$$
$$distance = \sqrt{4}$$
$$distance = 2$$

Before you read any further, make sure you understand above how the distance formula works in 2 dimensions x and y

Using the above idea, the movies that have been rated in common between users serve as axis or dimensions. The actual users are now data point on the movie graph. One example is provided here:

5 Blue dots represent users plotted against Happy Feet and Toy Story
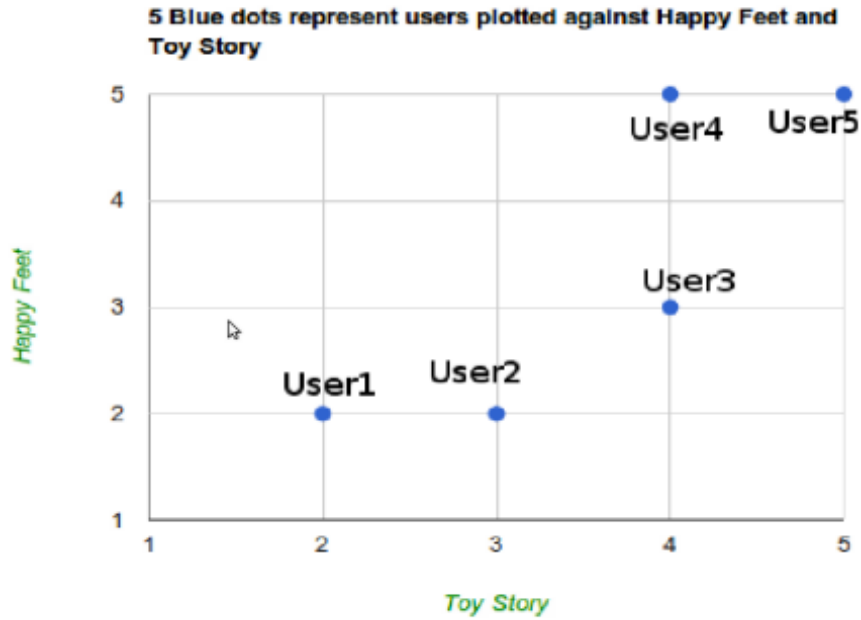
In the figure, there are 5 blue dots (each blue dot represents a single user) plotted against two movies i.e. *Happy Feet* and *Toy Story*. The *5 blue dots* or *5 users* are as follows:

1) (x,y) i.e. (ToyStory=2, HappyFeet=2) is *user1*
2) (x,y) i.e. (ToyStory=3, HappyFeet=2) is *user2*
3) (x,y) i.e. (ToyStory=4, HappyFeet=3) is *user3*
4) (x,y) i.e. (ToyStory=4, HappyFeet=5) is *user4*
5) (x,y) i.e. (ToyStory=5, HappyFeet=5) is *user5*

To calculate the Euclidean Distance between pairs of user in the above figure, we use the Euclidean Distance formula i.e.

$$\text{distance(user5,user1)}= \sqrt{(ToyStory_{rating\ user5} - ToyStory_{rating\ user1})^2 + (HappyFeet_{rating\ user5} - HappyFeet_{rating\ user1})^2}$$
$$\sqrt{(5-2)^2 + (5-2)^2} = 4.24$$

$$\text{distance(user5,user4)}= \sqrt{(ToyStory_{rating\ user5} - ToyStory_{rating\ user4})^2 + (HappyFeet_{rating\ user5} - HappyFeet_{rating\ user4})^2}$$
$$\sqrt{(5-4)^2 + (5-5)^2} = 1$$

This formula actually calculates the distance and we know that for users that are very similar (or very close on the chart) the distance will be **less** compared to users that are very dissimilar (or very far on the chart).

In order to get higher values (similarity score) for users who are similar we take the inverse of the distance and add 1 to the denominator to avoid division by 0. Hence the final similarity score between these pairs of users *(user5 and user1), (user5 and user4)*:

$$\text{similarity\_score(user5,user1)} = \frac{1}{1+distance(user5,user1)} = \frac{1}{(1+4.24)} = .190$$

$$\text{similarity\_score(user5,user4)} = \frac{1}{1+distance(user5,user4)} = \frac{1}{(1+1)} = .5$$

The similarity score is a value between 0 and 1. Value of 0 indicates that the pair of users are very different (dissimilar) and a value of 1 indicates that the pair of users have exactly identical taste. Based on the above-calculated similarity score we can say that (user5 and user4) are more similar compared to (user5 and user1).

In the above figure there are two dimensions *(Toy Story and HappyFeet)*. For multi-dimensions (more than two movies), the Euclidean Distance formula is:

distance (user1, user2) =

$$= \sqrt{(M1_{rating\ user1} - M1_{rating\ user2})^2 + (M2_{rating\ user1} - M2_{rating\ user2})^2 + ... + (MN_{rating\ user1} - MN_{rating\ user2})^2}$$

```
Where:
M1=Movie1
M2=Movie2
MN=MovieN
```

a) The **userUserMatrix** that your program calculates (where *columns* and *rows* are User1, User2, User3 and User4) is the following:

```
1    .1380  .2171  .1284
.1380  1    .1827  .1613
.2171  .1827  1    .125
.1284  .1613  .125  1
```

b) **Most similar pair of users** *(both users in the pair **MUST** be different. If there are more than one pair of different users with the same score, you **must** list all of them but do not repeat them for example the pair (User1 and User3) is identical to (User3 and User1)):*
From the userUserMatrix, the most similar pair of users are:
User1 and User3.

c) **Most dissimilar pair of users** *(both users **MUST** be different. If there are more than one pair of different users with the same score, you **must** list all of them but do not repeat them for example the pair (User3 and User4) is identical to (User4 and User3)):*
From the userUserMatrix, the most dissimilar pair of users are:
User3 and User4.

Note:

--Precision in the *userUserMatrix* must be rounded off to *four decimal places.*

--The **output** from your program MUST contain three entities (SEE THE SAMPLE OUTPUT PROVIDED ON THE LAST PAGE).
a) userUserMatrix
b) most similar pair(s) of users
c) most dissimilar pair(s) of users.

--Print/output your *userUserMatrix* in a readable form, so it becomes easy for the TA to mark it.

-- The template code has lots of comments for you to get started, but your program must have atleast the following functions **(you are free to define any signature of the function i.e. take in any number of input of any type and return value of any type. BUT DO NOT CHANGE THE NAME OF THE FUNCTION):**
        a) `calculateSimilarityScoreForAllPairsOfUser` —> this function will calculate the similarity score for all pairs of User
        b) `printUserUserMatrix` —>this function will print the *userUserMatrix* to console
        c) `findAndprintMostSimilarPairOfUsers` —>this function will find and then print the most similar pair of users
        d) `findAndprintMostDissimilarPairOfUsers` —>this function will find and then print the most dissimilar pair of users

--You MUST create a `test` folder in your SVN repository, which will reside inside here. (i.e. test folder must be created inside the 207Assignment0 folder). The name of the folder is case sensitive.

https://142.1.44.22/svn/csc207h/UTORID/207Assignment0/

The `test` folder will contain FOUR test input files that you used to test your code against and that will be different from the two provided.

--At time of marking your assignment, we will not change the input file in any form. All your test input files MUST obey the format of the provided *input1.txt* and *input2.txt*

--Have sufficient comments in your code. You do not need to generate `Javadoc` for this assignment.

--Have sufficient (atleast 6 revisions towards assignment0) revisions with comments in your repository. Do not have all 6 revisions on the day the assignment is due.

--Remove any debugging print messages from your starter code.

--**You must not exceed the 80-character limit on each line**. You can set this easily in Eclipse. (See the slides on this inside the Assignment folder on Blackboard, on how to do this in Eclipse). Also, for this course we will follow the Google Style Java Code. The .xml is available on Blackboard for you to import directly into Eclipse.

Here is a an output, that your program should display when executed with the provided input1.txt:

```
Enter the name of input file?
input1.txt


userUserMatrix is:
[1.0000, 0.1380, 0.2171, 0.1285]
[0.1380, 1.0000, 0.1827, 0.1614]
[0.2171, 0.1827, 1.0000, 0.1250]
[0.1285, 0.1614, 0.1250, 1.0000]


The most similar pairs of users from above userUserMatrix are:
User1 and User3
with similarity score of 0.2171


The most dissimilar pairs of users from above userUserMatrix are:
User3 and User4
with similarity score of 0.1250
```

Here is a an output, that your program should display when executed with the provided input2.txt:
Enter the name of input file?
input2.txt

userUserMatrix is:
[1.0000, 1.0000, 1.0000, 1.0000]
[1.0000, 1.0000, 1.0000, 1.0000]
[1.0000, 1.0000, 1.0000, 1.0000]
[1.0000, 1.0000, 1.0000, 1.0000]

The most similar pairs of users from above userUserMatrix are:
User1 and User2,
User1 and User3,
User1 and User4,
User2 and User3,
User2 and User4,
User3 and User4
with similarity score of 1.0000

The most dissimilar pairs of users from above userUserMatrix are:
User1 and User2,
User1 and User3,
User1 and User4,
User2 and User3,
User2 and User4,
User3 and User4
with similarity score of 1.0000

Note: There are two line break separate each section in the above output.