Homework 7 (Total Points: 100): A Simplified NIST RBAC System
Due Date*: 11:59pm 11/11/2021, Cutoff Deadline*: 11:59pm 11/13/2021
***Late penalty will not be applied, ****Submission will NOT be accepted or graded after the cutoff deadline**

Relevant Topics: Discussed in the lectures/classes ***before or on*** **the Due Date.** ***Lab 1*** is related to Task II.

Submission: (1) Upload and submit your .java file(s) and .txt input files in **Canvas**. Only ONE attempt is allowed for each student. NO Paper/Hardcopy or Email submission please! (2) UPLOAD all .java files and all input .txt files to **cs3750a.msudenver.edu**, COMPILE the .java files into .class files, TEST your program on **cs3750a (**see Task II for more details**),** and LEAVE all files on **cs3750a, which will be used for testing and grading by the instructor on cs3750a.**

An option of peer programming: You may choose to work on this assignment individually or in a team of two students. If you choose to work in a ***team of two students***, you **must** 1) **add** both team members' first and last names as the **comments** on Canvas when submitting the .java files (***both team members are required to submit*** the same .java files on Canvas), and 2) put a ***team.txt*** file including both team members' names under your HW07/ on cs3750a (***both team members are required to complete Task II under both home directories on cs3750a***). For grading, I will ***randomly*** pick the submission in ***one of the two home directories of the team members*** on cs3750a, and then give both team members the ***same grade***. If the team information is *missing* on Canvas or cs3750a or not all students involved work in the same team, two or more submissions with similar source codes with just variable name/comment changes will be considered to be a violation of the Integrity described in the course policies and both or all will be graded as 0. Here are a few reminders on "What IS NOT Allowed" to hopefully avoid any incident of violating Academic Integrity.
1. Other than the example programs provided by the instructor to all students fairly, you may NOT look at code created by another person (even to debug) or provided by any online/offline resource (including but not limited to tutors and online/offline tutoring service) for any programming assignment/project until after you have completed and submitted the assignment yourself (or with your teammate if allowed and claimed in team.txt).
2. Students absolutely may NOT turn in someone else's code or solution with simple cosmetic changes (say, changed variable names or changed order of statements) to any homework, project, or exam.
3. It is strictly forbidden for any student to refer to code or solutions from previous offerings of this course or from any online/offline resource including but not limited to tutors and tutoring services unless this information is provided by the instructor to all students fairly.

Grading: Your programs will be graded via testing and points are associated with how much task it can complete.
1. You are highly recommended **NOT to use any IDE**, *online* or *on your computer*, to compile, debug, and test your programs. Instead, you should compile, debug, and test your programs in the command-line Linux environment on **cs3750a** and/or **cs3750b**, which is part of the learning objectives in this upper-division CS course. Being able to make your programs only work in an IDE (online or on your computer) but NOT on **cs3750a**/**cs3750b** ***cannot*** be used as an excuse for re-grading or more points either.
2. ALL the ***output messages*** that are REQUIRED to be ***displayed*** by your program in Task I are the only way to demonstrate how much task your program can complete. So, whether your program can display those messages correctly with correct information is critical for your program to earn points for the work it may complete.
3. It is also extremely important for your program to be able to correctly READ and TAKE the information from the ***input file(s)*** and/or the ***user inputs*** that follow the format REQUIRED in this programming assignment, instead of some different format created by yourself. Otherwise, your program may crash or get incorrect information from the input file(s) or user inputs that follow the required format.
4. Grading is NOT based on reading/reviewing your source code although the source code will be used for plagiarism check. A program that cannot be compiled or crashes while running will receive up to 5% of the total points. A submission of source code files that are similar to any online source code with only variable name changes will receive 0% of the total points.

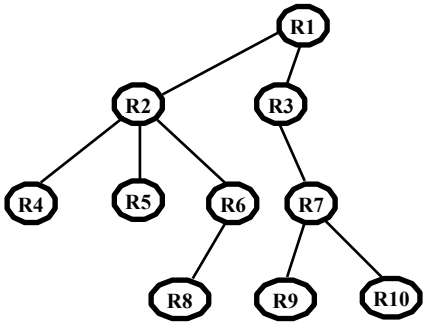
Programming in Java is highly recommended, since all requirements in this assignment are guaranteed to be supported in Java. If you choose to use *any other language* such as C/C++ or Python, it is YOUR responsibility, before the cutoff deadline, to (2) set up the compiling and running environment on cs3750a, (2) make sure that I can run/test your programs in your home directory on c3750a, and (3) provide a README file under HW07/ on cs3750a to include the commands for me to use.

Task I (90%). Write a Java program to implement a simplified NIST RBAC model. (The topology and examples here are provided for you to test your program. The instructor will use similar but different test cases to test your program.)

1. Data structures: it is up to you to choose **appropriate data structures** to maintain *role hierarchy*, *user-role matrix*, *role-object matrix*, etc.

2. Role Hierarchy: this program must support role hierarchy defined in NIST RBAC.

2.1 This program reads a role hierarchy from file "*roleHierarchy.txt*". Each line in this file is formatted as "<ascendant> <descendant>", in which a *tab* ('\t') is the separator in between. For example, for the role hierarchy on the right, the first several lines in roleHiearchy.txt is




```
R8          R6
R9          R7
R10         R7
……
```

2.2 The contents of this text file must be **validated** to enforce the "limited role hierarchies" defined in NIST RBAC (See the slide titled "**Hierarchical RBAC**"). For an **invalid** file, (1) **close** the file, (2) **display** "invalid line is found in roleHierarchy.txt: line <line # of the first invalid line>, enter any key to read it again" to allow user to fix the problem, (3) once reading any user input, go back to **2.2**. **Continue** if it's **valid**.

2.3 Display the role hierarchy **top down**. If you may display it as a tree, it is great. Otherwise, the following format is fine.

```
R1--->R2, R3
R2--->R4, R5, R6
R3--->R7
……
```

3. Objects: This program reads all ***resource objects*** from file "*resourceObjects.txt*", which contains ONE line. For example,

```
F1      F2      F3      F4      P1      P2      P3      D1      D2
```

3.1 Validate objects for duplication. For an **invalid** file, (1) **close** the file, (2) **display** "duplicate object is found: < object>, enter any key to read it again" to allow user to fix the problem, (3) once reading any user input, go back to **3.1**. **Continue** if it's **valid**.

3.2 Display the current *role-object matrix* with null entries. If necessary, you may break this matrix into multiple sub-matrices for displaying by limiting up to 5 columns per sub-matrix. For example,

```
        R1      R2      R3      R4      R5
R1
……
        R6      R7      R8      R9      R10
R1
……
… … … … … … …
```

4. Granting Permissions to Roles with Inheritance:

4.1 This program must support role inheritance described as "the upper role includes all of the access rights of the lower role, as well as other access rights not available to the lower role" in the slide titled "**Role Hierarchy**". In **4.3**, **4.4**, and **4.5**, whenever a permission is granted to a role, it must be **inherited** by **all** the **descendants** of this role.

4.2 Redundancy must be avoided. For example, when adding "read" by R2 to F1, if "read" is already there, don't add it again.

4.3 Update the *role-object matrix* to grant "control" by EACH role to itself. (See **4.1**). E.g., R8, R6, R2, and R1 "control" R8.

4.4 Update the *role-object matrix* to grant "own" by the descendent of EACH role to it. (See **4.1**). E.g., R6, R2, and R1 "own" R8.

4.5 This program also reads permissions to roles from file "*permissionsToRoles.txt*". Each line in this file is formatted as "<role> <access right> <object>", where a permission is an access right to an object. For example, after reading the following two lines, R6, R2, and R1 have "write*" to F3, and R2 and R1 have "seek" to D1.

```
R6      write*          F3
R2      seek            D1
```

4.6 Display the current *role-object matrix* in the same format as what is used in **step 3.2**.

5. SSD: This program reads ***mutually exclusive*** (when $n = 2$) and ***cardinality constraints*** (when n>= 3) from file "*roleSetsSSD.txt*".

5.1 Each line includes a constraint given by the value of *n* and a set of roles. For example,

```
3       R2      R4      R5      R6      R8
2       R1      R2      R4
……
```

Note: ***n*** MUST be >= 2 to be valid. For an **invalid** case, (1) **close** the file, (2) **display** "invalid line is found in roleSetsSSD.txt: line <line # of the first invalid line>, enter any key to read it again" to allow user to fix the problem, (3) once reading any user input, go back to Step **5**. **Continue** if it's **valid.**

5.2 Display all the constraints, one in a line. For example,

```
Constraint 1, n = 3, set of roles = {R2, R4, R5, R6, R8}
…
```

6. This program constructs the *user-role matrix* by reading file *usersRoles.txt*. Each line is formatted as "<user> <list of roles>".

6.1 While reading each line and updating the *user-role matrix*, the **SSD constraints** configured in Step **5 must be enforced**. It is **also invalid** if two or more lines contain the same user. For an **invalid** file, (1) **close** the file, (2) **display** "invalid line is found in usersRoles.txt: line <line # of the first invalid line> due to <constraint #? or duplicated user>, enter any key to read it again" to allow user to fix the problem, (3) once reading any user input, go back to Step **6**. **Continue** if it's **valid.**

```
U1      R2      R5
.....
```

6.2 Display the *user-role matrix*. For example,

```
        R1      R2      R3      R4      R5      R6      R7      R8      R9      R10
U1              +                       +
.....
```

7. Query for a given User, a given pair of User and Object, or a given combination of (User, Access Right, Object)

7.1 Display the following messages to get three user inputs: **`user`**, **`object`**, and **`accessRight`**.

```
Please enter the user in your query:
Please enter the object in your query (hit enter if it's for any):
Please enter the access right in your query (hit enter if it's for any):
```

7.2 if **`user`** isn't in the *user-role matrix*, display "invalid user, try again." and go back to Step 7.1. Otherwise, **continue**.

7.3 if **`object`** and **`accessRight`** are empty, **display** all the objects that this user has access rights to as "<object> <list of access rights to this object>", and then **go to** Step 7.7. Otherwise, **continue**. To keep it simple, if a user has the same or different lists of access rights to the same object due to different roles assumed by this user, you don't have to merge them. For example,

```
R4      control, own
F1      read, write, execute
R4      own
...
```

7.4 if **`object`** isn't in the *user-role matrix*, display "invalid object, try again." and go back to Step 7.1. Otherwise, continue.

7.5 if **`accessRight`** is empty, **display** all the access rights this user has to this object and **go to** Step 7.7. Otherwise, continue.

7.6 check whether this **`user`** has the given **`accessRight`** to this **`object`**. If yes, display "authorized", otherwise, display "rejected". Continue.

7.7 Display "Would you like to continue for the next query?" If the user inputs "yes", go back to Step 7.1. Otherwise, exit.

Task II (10%): **Upload and test your programs on the virtual server cs3750a.msudenver.edu.**

Warning: to complete this part, especially when you work at home, you must first (1) **connect to GlobalProtect** using your NetID account (please read "**how to connect to GlobalProtect …**" at the links given in Lab 1); then (2) **connect to the virtual servers cs3750a** using ***sftp*** and ***ssh*** command on MAC/Linux or ***PUTTY*** and ***PSFTP*** on Windows. For details, please refer to **Lab 1 manual/video.** ITS only supports GlobalProtect on MAC and Windows machines. If your home computer has a different OS, it is your responsibility to figure out how to connect to cs3750a for programming assignments and submit your work by the cutoff deadline. Such issues cannot be used as an excuse to request any extension.

1. MAKE a directory "**HW07**" under your home directory on **cs3750a.msdenver.edu**.
2. UPLOAD all .java file(s) and all input .txt files to "**HW07**" on **cs3750a.msdenver.edu**, COMPILE the .java file(s) into .class file(s) on **cs3750a**, and TEST your program there. Please make sure that all .java file(s), .class file(s), and input .txt files that you used for testing are left under "**HW07**" on **cs3750a.msdenver.edu**.

```
javac prog_name.java    //compile .java file into byte code into .class file
java prog_name {args}   //run prog_name with optional arguments for testing
```

3. SAVE one ***file named "tstResults.txt"*** under "**HW07**" on **cs3750a.msudenver.edu**, which captures the outputs of your programs as a proof that you have tested both programs on cs3750a. You can use the following commands to redirect the standard output (stdout) to a **file** on UNIX, Linux, or Mac, and view the contents of the file

```
java prog_name {args} | tee tstResults.txt  // run program with optional arguments
                                            // while copying stdout to the given .txt file
cat tstResults.txt      //display the result file's contents to check.
```

4. If you work in a team of two students, you must put a ***team.txt*** file including both team members' names under your HW07/ on cs3750a (***both team members are required to complete Task II under their own home directories on cs3750a***). For grading, I will ***randomly pick*** the submission in one of the two *home directories* of the team members on cs3750a, and then give both team members the ***same*** grade.