CSI 5342 Assignment 13.1

By: Matthew Tuan and Robbie Dutton

After setting up our initial Class diagram (as seen below), we began adding in constraints to ensure our model behaves as expected.

A screenshot of a computer

AI-generated content may be incorrect.

The first constraints we added were the basic SSD and DSD role constraints:  
A screenshot of a computer screen

AI-generated content may be incorrect.

We confirmed these invariants with two example script files InvalidSSDTest.x and InvalidDSDTest.x:

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Next we add in additional invariants for to check the SSD Permissions constraint:

A close-up of a sign

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

For the operation Session::CheckSession(), we needed to find a way to make sure that a user who is in a session, has access to perform operation op, on object obj. To do this we had to make a few updates.   
  
The first, we needed to update the Descendants function on Role class, to return a list of all of its role descendants. Then, update the Session::CheckSession() method. To test this, we created a ValidSessionCheckSession.x file, ran that, and then opened the OCL window and ran s1.CheckSession(o1, op1). This returned true, which meant this session has access to that object with that operation.

A computer screen shot of a computer

AI-generated content may be incorrect.

Next, we developed models that would make this false. To do that we created an InvalidSessionCheck.x file, that did not associate the role to the object. We got false!

A screenshot of a computer

AI-generated content may be incorrect.

For checking on the Role::CheckAccess(object, operation) that was created. We can now check for that by calling the same on that role.

A screenshot of a computer

AI-generated content may be incorrect.

To test an invalid case. We can then make a new role, and not assign it to anything. I just dragged a new Class onto the object diagram, not attaching it to any operations. Then I sent the same o1 and op1 and got false!  
A screenshot of a computer

AI-generated content may be incorrect.  
  
  
Lastly, for the PermissionCheck, we create a method that ensures the object and permission that enter in, are included in the permission.

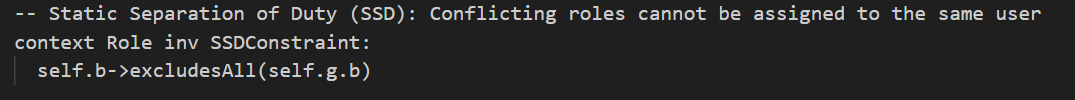
A computer screen shot of a computer

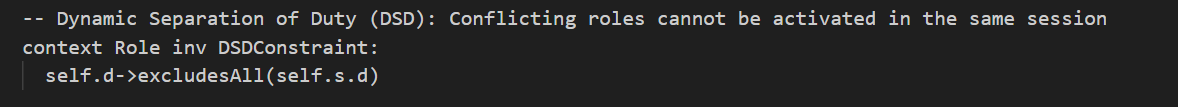
AI-generated content may be incorrect.  
  
Now, for the false test, I added a new permission object, but only connected it to op1, and not o1. So it should return false, like pictured below.

A screenshot of a computer

AI-generated content may be incorrect.

**Role Hierarchies and SSD/DSD Constraints in the RBAC model.**

By looking at the various papers, specifically [1], and reading through the figures and documentation, we were able to find a great understanding for the role hierarchies and how they play an important role. When we link the roles via the association, we give the ability for all the senior roles to view the junior roles. This way, when we use the CheckAccess methods, specifically in the Session class, we were able to union all roles together, and easily check if any role was indeed proper.   
  
For SSD, we needed to ensure that no conflicting roles were assigned to the same user. To do this we wrote the SSDConstraint invariant.  


This invariant ensured that no 2 roles that are conflicting, could be assigned to the same user.  
  
Next, for the DSD constrain, we wrote the following invariant.  


This invariant ensured that within the role, we make sure that the sessions of that role did not contain a conflicting role. This is different than SSD, because we are only looking at the single session. For example, a user could activate a new role in one session, but then deactivate it, then go ahead and activate another role without any issue.  
  
The above tests on our OCL USE file were able to demonstrate a proper working RBAC system.