# Assignment 5

**Group 9:**
Claire Huri
Jiajun Ni
Ang Zheng
Feng Liu

# Task 1

## Test Case 1: test_tokenGenerator

**Description:** Ensures the functionality of login. When user sends the log in request ,ShibbolethAuth.tokenGenerator() is invoked, and generates the token with expected type (STUDENT, ADMIN, BOTH, UNDEFINED).

**Test Inputs:**

1. x500='Abc1001',password='1000'(Assume this x500 exists in the X500ACCOUNT colume of SHIBBOLETHAUTH Table, with the entry's Type='STUDENT' and this password equals the entry's X500ACCOUNT value)
2. x500='Abc1005',password='5000'(Assume this x500 exists in the X500ACCOUNT colume of SHIBBOLETHAUTH Table, with the entry's Type='ADMIN' and this password equals the entry's X500ACCOUNT value)
3. x500='Abc1002',password='2000'(Assume this x500 exists in the X500ACCOUNT colume of SHIBBOLETHAUTH Table, with the entry's Type='BOTH' and this password equals the entry's X500ACCOUNT value)
4.x500='Abc1111',password='12345'(Assume this x500 doesn't exist in the X500ACCOUNT colume of SHIBBOLETHAUTH Table)
5.x500='Abc1001',password='2000'(Assume this x500 exists in the X500ACCOUNT colume of SHIBBOLETHAUTH Table but this password doesn't equal the entry's X500ACCOUNT value)

**Expected Results:**

1.Token object with type=STUDENT
2.Token object with type=ADMIN
3.Token object with type=BOTH
4.Token object with type=UNDEFINED
5.Token object with type=UNDEFINED

**Dependencies:** None

**Initialization:** All x500 and passwords are stored in the SUBBOLETHAUTH Table.

**Test Step:**

1.The uses send the login request by invoke ShibbolethAuth.tokenGenerator() method with x500 and password
2.The correct Token object is returned

## Test Case 2: test_queryclass

**Description:** Ensures the functionality of queryclass. When user sends the queryclass request ,ISCRS.queryclass() is invoked, it takes *courseID, courseName,location,term, department, classType, instructorName* as input and return a list of Arraylist of expected result. Every Arraylist stores the following information *ID,NAME,CREDITS,CAPACITY,TERM,FIRSTDAY,LASTDAY,CLASSBEGINTIME,CLASSEND TIME,ROUTINES,LOCATION,TYPE,PREREQUISITE,DESCRIPTION,DEPARTMENNT* in order. If query is not valid or no matching results, the method returns an empty list.

**Test Inputs:**

1. -1,'Advanced Algorithm','East Campus','Fall 2015','','',''(all required criteria included with none optional criteria)
2.-1,'Advanced Algorithm','East Campus','Fall 2015','CS','',''(all required criteria included with one optional criteria department)
3.-1,'Software Engineering','East Campus','Fall 2015','','','Kevin Wendt'(all required criteria included with one optional criteria instructorName)
4.2,'Software Engineering','East Campus','Fall 2015','','','Kevin Wendt'(all required criteria included with two optional criteria courseID and instructorName)
5.-1,'','','','','',''(None criteria included)
6.-1,'','','Fall 2015','','',''(not all required criteria included)

**Expected Results:**

1.For 1-4, a list of Arraylist as specified in description or empty list if no matching results
2.For 5-6, empty list

**Dependencies:** None

**Initialization:** Course Table, Instructor TABLE and InstructorAndCourse Table stored in the database

**Test Step:**

1.The uses send the queryclass request by invokeISCRS.queryclass() method with inputs as in test inputs
2.Expected results returned

## Test Case 3: test_queryInstructor

**Description:** Ensures the functionality of queryInstructor. When user sends the queryInstructor request ,ISCRS.queryInstructor() is invoked. It takes *token, instructorID* as input and return a list of Arraylist of expected result. The list should only contain one arraylist as instructorID is unique, and the arraylist stores the following information *ID,FIRSTNAME,LASTNAME,DATEOFBIRTH,,GENDER,TITLE,SALARY,DEPARTMENT* in order. If query is not valid or no matching results, the method returns an empty list.

**Test Inputs:**
1. *token(id=1,type=ADMIN,timestamp=”2015.08.01.17.30.05”),1* (Assume the instructor with ID=1 exists in database)
2. *token(id=1,type=BOTH,timestamp=”2015.08.01.17.30.05”),1* (Assume the instructor with ID=1 exists in database)
3. *token(id=1,type=ADMIN,timestamp=”2015.08.01.17.30.05”),10* (Assume the instructor with ID=10 doesn’t exist in database)
4. *token(type=UNDEFINED,timestamp=”2015.08.01.17.30.05”),1* (the token type is not ADMIN or BOTH)
5. *token(type=STUDENT,timestamp=”2015.08.01.17.30.05”),1* (the token type is not ADMIN or BOTH)

**Expected Results:**
1.For 1-2, return a list of Arraylist as specified in description
2.For 3-5, return a empty list
**Dependencies:** 1
**Initialization:** Instructor Table is stored in the database
**Test Step:**
1.The user sends log in request and a token object generated
2.The user sends the queryInstructor request by invokeISCRS.queryInstructor() method with inputs as specified in test inputs
3.Expected list returned

## Test Case 4: test_studentAddClass

**Description:** Ensures the functionality of student Add Class. When user sends the studentAddClass request ,ISCRS.studentAddClass() is invoked. It takes *token, courseID, grading, courseTerm* as input and return a boolean value(true is student successfully adds the class, false otherwise).

**Test Inputs:**

1. *token(id=1,type=STUDENT,timestamp=”2015.08.01.17.30.05”),3,”A-F”,”Fall 2015”* (Assume the course with ID=3 is not full and the student with USERID=1 will not exceed 30 credits after the adding the course)
2. *token(id=2,type=BOTH,timestamp=”2015.08.01.17.30.05”),3,”A-F”,”Fall 2015”* (Assume the course with ID=3 is not full and the student with USERID=2 will not exceed 30 credits after the adding the course)
3. *token(id=0,type=ADMIN,timestamp=”2015.08.01.17.30.05”),3,”A-F”,”Fall 2015”* (the token type is not STUDENT or BOTH)
4. *token(type=UNDEFINED,timestamp=”2015.08.01.17.30.05”),3,”A-F”,”Fall 2015”* (the token type is not STUDENT or BOTH)
5. *token(id=1,type=STUDENT,timestamp=”2015.08.01.17.30.05”),4,”A-F”,”Fall 2015”* (Assume the course with ID=4 is not full and the student with USERID=1 will exceed 30 credits after the adding the course)
6. *token(id=1,type=STUDENT,timestamp=”2015.08.01.17.30.05”),5,”A-F”,”Fall 2015”*(Assume the course with ID=5 is full)
7. *token(id=1,type=STUDENT,timestamp=”2015.09.05.17.30.05”),3,’A-F’,’Fall 2015’* (Timeframe has passed)

**Expected Results:**

1.For 1-2, return true

2.For 3-7, return false

**Dependencies:** 1

**Initialization:** Course Table, Student TABLE and StudentAndCourse Table are stored in the database

**Test Step:**

1.The user sends log in request and a token object generated

2.The user sends the student add class request by invokeISCRS.studentAddClass() method with inputs as specified in test inputs

3.Expected boolean value returned

## Test Case 5: test_studentEditClass

**Description:** Ensures the functionality of student Edit Class. When user sends the studentEditClass request ,ISCRS.studentEditClass() is invoked. It takes *token, courseID, grading, courseTerm* as input and return a boolean value(true is student successfully edits the class, false otherwise).

**Test Inputs:**

1. *token(id=1,type=STUDENT,timestamp=”2015.08.01.17.30.05”),3,”A-F”,”Fall 2015”* (all required criteria included)
2. *token(id=0,type=ADMIN,timestamp=”2015.08.01.17.30.05”),3,”A-F”,”Fall 2015”* (the token type is not STUDENT or BOTH)
3. *token(type=UNDEFINED,timestamp=”2015.08.01.17.30.05”),3,”A-F”,”Fall 2015”* (the token type is not STUDENT or BOTH)
4. *token(id=1,type=STUDENT,timestamp=”2015.09.05.17.30.05”),3,’A-F’,’Fall 2015’* (Timeframe has passed)

**Expected Results:**

1.For 1, return true

2.For 2-4, return false

**Dependencies:** 1

**Initialization:** Course Table, Student Table and StudentAndCourse Table are stored in the database

**Test Step:**

1.The user sends log in request and a token object generated

2.The user sends the student edit class request by invokeISCRS.studentEditClass() method with inputs as specified in test inputs

3.Expected boolean value returned

## Test Case 6: test_studentDropClass

**Description:** Ensures the functionality of student Drop Class. When user sends the studentDropClass request ,ISCRS.studentDropClass() is invoked. It takes *token, courseID* as input and return a boolean value(true is student successfully drops the class, false otherwise).

**Test Inputs:**

1. *token(id=0,type=STUDENT,timestamp=”2015.08.01.17.30.05”),3* (all required criteria included)
2. token*(id=0,type=STUDENT,timestamp=”2015.08.01.17.30.05”),4* (Assume course has is not in the sutdent’s registered course list)
3. *token(id=0,type=ADMIN,timestamp=”2015.08.01.17.30.05”),3* (the token type is not STUDENT or BOTH)
4. *token(type=UNDEFINED,timestamp=”2015.08.01.17.30.05”),3* (the token type is not STUDENT or BOTH)
5. *token(id=1,type=STUDENT,timestamp=”2015.09.05.17.30.05”),3* (Timeframe has passed)

**Expected Results:**

1.For 1, return true

2.For 2-5, return false

**Dependencies:** 1

**Initialization:** Course Table Student Table and StudentAndCourse Table are stored in the database

**Test Step:**

1.The user sends log in request and a token object generated

2.The user sends the student drop class request by invokeISCRS.studentDropClass() method with inputs as specified in test inputs

3.Expected boolean value returned

## Test Case 7: test_queryStudentPersonalData

**Description:** Ensures the functionality of queryStudentPersonalData. When user sends the queryStudentPersonalData request ,ISCRS.queryStudentPersonalData() is invoked, it takes *token, studentID* as input and return a list of Arraylist of expected result. Every Arraylist stores the following information *ID,FIRSTNAME, LASTNAME, DATEOFBIRTH, TYPE, GENDER, ADVISOR, CREDITS, DEPARTMENT* in order. If query is not valid or no matching results, the method returns an empty list.

**Test Inputs:**

1. *token(id=0,type=STUDENT, timestamp=”2015.08.01.17.30.05”), 0* (student query student )
2. *token(id=0,type=ADMIN,timestamp=”2015.08.01.17.30.05”), 0* (admin query student )
3. *token(id=0,type=STUDENT, timestamp=”2015.08.01.17.30.05”), 1* (a student query another student’s personal data)
4. *token(id=0,type=UNDEFINED,timestamp=”2015.08.01.17.30.05”)*, 0 (the token type is not STUDENT, ADMIN or BOTH)
5. *token(id=0,type=ADMIN,timestamp=”2015.08.01.17.30.05”)*, 10 (no such student)

**Expected Results:**

1.For 1-2, a list of Arraylist as specified in description or empty list if no matching results
2.For 3-5, empty list

**Dependencies:** None
**Initialization:** Student Table and Administrator Table stored in the database
**Test Step:**

1.The user sends log in request and a token object generated
2.The user sends the queryStudentPersonalData request by invokeISCRS.queryStudentPersonalData() method with inputs as specified in test inputs
3.Expected results returned

## Test Case 8: test_queryStudentRegistrationHistory

**Description:** Ensures the functionality of queryStudentRegistrationHistory. When user sends the queryStudentRegistrationHistory request ,ISCRS.queryStudentRegistrationHistory() is invoked, it takes *token, studentID* as input and return a list of Arraylist of expected result. Every Arraylist stores the following information *CLASSID, CLASSNAME, REGISTRATION TIME, CREDITS* in order. If query is not valid or no matching results, the method returns an empty list.

**Test Inputs:**

1. *token(id=0,type=STUDENT, timestamp="2015.08.01.17.30.05"), 0* (student query student )
2. *token(id=0,type=ADMIN,timestamp="2015.08.01.17.30.05"), 0* (admin query student )
3. *token(id=0,type=STUDENT, timestamp="2015.08.01.17.30.05"), 1* (a student query another student)
4. *token(id=0,type=UNDEFINED,timestamp="2015.08.01.17.30.05")*, 0 (the token type is not STUDENT, ADMIN or BOTH)
5. *token(id=0,type=ADMIN,timestamp="2015.08.01.17.30.05")*, 10 (no such student)

**Expected Results:**

1.For 1-2, a list of Arraylist as specified in description or empty list if no matching results
2.For 3-5, empty list

**Dependencies:** None

**Initialization:** Student Table, StudentAndCourse Table, Course Table and Administrator Table stored in the database

**Test Step:**

1.The user sends log in request and a token object generated

2.The user sends the queryStudentRegistrationHistoryrequest by invokeISCRS.queryStudentRegistrationHistory() method with inputs as specified in test inputs

3.Expected results returned

## Test Case 9: test_adminAddClass

**Description:** Ensures the functionality of admin Add Class. When user sends the adminAddClass request ,ISCRS.adminAddClass() is invoked. It takes *token, courseID, courseName, courseCredits, courseCapacity, term, instructorID, firstDay, lastDay, classBeginTime, classEndTime, weekDays, location, type, prerequisite, description, department* as input and return a boolean value(true is admin successfully adds the class, false otherwise).

**Test Inputs:**

1. *token(id=1,type=ADMIN,timestamp="2015.08.01.17.30.05"),3, "Operating Systems", 4, 2, "Fall2015", 1, "20150913", "20151208", "09:30", "10:45", "Tu, Th", "ME 321", "Lecture", "Intro to OS" and "CS"* (Assume the course with ID=3 is empty)
2. *token(id=1,type=ADMIN,timestamp="2015.08.01.17.30.05"),3, "Operating Systems", 4, 2, "Fall2015", 1, "20150913", "20151208", "09:30", "10:45", "Tu, Th", "ME 321", "Lecture", "" and "CS"* (Assume the course with ID=3 is empty and description is empty)
3. *token(id=1,type=ADMIN,timestamp="2015.08.01.17.30.05"),1, "Operating Systems", 4, 2, "Fall2015", 1, "20150913", "20151208", "09:30", "10:45", "Tu, Th", "ME 321", "Lecture", "Intro to OS" and "CS"* (Assume the course with ID=1 is not empty)
4. *token(id=1,type=ADMIN,timestamp="2015.08.01.17.30.05"),3, "Operating Systems", 4, 2, "Fall2015", 1, "20150913", "20151208", "09:30", "10:45", "Tu, Th", "ME 321", "Lecture", "Intro to OS" and "CS"* (the token type is not ADMIN or BOTH)
5. *token(id=1,type=ADMIN,timestamp="2015.08.01.17.30.05"),3, "Operating Systems", 4, 2, "Fall2015", 1, "20150913", "20151208", "09:30", "10:45", "Tu, Th", "ME 321", "Lecture", "Intro to OS" and "CS"* (the token type is not ADMIN or BOTH)
6. *token(id=1,type=ADMIN,timestamp="2015.08.01.17.30.05"),3, "Operating Systems", 5, 2, "Fall2015", 1, "20150913", "20151208", "09:30", "10:45", "Tu, Th", "ME 321", "Lecture", "Intro to OS" and "CS"* (credits larger than 4)
7. *token(id=1,type=ADMIN,timestamp="2015.08.01.17.30.05"),3, "Operating Systems", 4, 31, "Fall2015", 1, "20150913", "20151208", "09:30", "10:45", "Tu, Th", "ME 321", "Lecture", "Intro to OS" and "CS"* (capacity larger than 30)
8. *token(id=1,type=ADMIN,timestamp="2015.08.01.17.30.05"),3, "Operating Systems", 4, 31, "Fall2015", 1, "20150913", "20151208", "09:30", "10:45", "Tu, Th", "ME 321", "a", "Intro to OS" and "CS"* (type is not lecture or seminar)
9. *token(id=1,type=ADMIN,timestamp="2015.08.01.17.30.05"),3, "", 4, 29, "", 1, "", "", "", "", "", "", "", "", ""* (required field is empty)

**Expected Results:**

1.For 1-2, return true

2.For 3-9, return false

**Dependencies:** 1

**Initialization:** Administrator Table, Course Table and StudentAndCourse Table  are stored in the database

**Test Step:**

1.The user sends log in request and a token object generated

2.The user sends the admin add class request by invokeISCRS.admintAddClass() method with inputs as specified in test inputs

3.Expected boolean value returned

## Test Case 10: test_adminDropStudentRegisteredClass

**Description:** Ensures the functionality of admin Drop Student Registered Class. When user sends the adminDropStudentRegisteredClass request ,ISCRS.adminDropStudentRegisteredClass() is invoked. It takes *token, studentID, courseID* as input and return a boolean value(true is admin successfully drops the student from the class, false otherwise).

**Test Inputs:**

1. *token(id=1,type=ADMIN,timestamp=”2015.08.01.17.30.05”),1,1(Assume the student with ID=1 has course with ID=1 in his registered course list)*
2. *token(id=1,type=BOTH,timestamp=”2015.08.01.17.30.05”),1,1((Assume the student with ID=1 has course with ID=1 in his registered course list)*
3. *token(id=1,type=ADMIN,timestamp=”2015.08.01.17.30.05”),1,10((Assume the student with ID=1 has no course with ID=10 in his registered course list)*
4. *token(id=0,type=STUDENT,timestamp=”2015.08.01.17.30.05”),3,1*
(the token type is not ADMIN or BOTH)
5. *token(type=UNDEFINED,timestamp=”2015.08.01.17.30.05”),3,1*
(the token type is not ADMIN or BOTH)

**Expected Results:**

1.For 1-2, return true
2.For 3-5, return false

**Dependencies:** 1

**Initialization:** Course Table, Administrator Table, StudentAndCourse Table are stored in the database

**Test Step:**

1.The user sends login request and a token object generated
2.The user sends the admin drop student registered class request by invokeISCRS.adminDropStudentRegisteredClass() method with inputs as specified in test inputs
3.Expected boolean value returned

## Test Case 11: test_adminEditStudentRegisteredClass

**Description:** Ensures the functionality of admin Edit Student Registered Class. When user sends the adminEditStudentRegisteredClass request ,ISCRS.adminEditStudentRegisteredClass() is invoked. It takes *token, studentID, courseID, grading, courseTerm* as input and return a boolean value(true is admin successfully edits the information of student registered class, false otherwise).

**Test Inputs:**

1. *token(id=1,type=ADMIN,timestamp="2015.08.01.17.30.05"),1,1,"S/N", "Fall2015"*
(Assume the student with ID=1 has course with ID=1 in his registered course list)
2. *token(id=2,type=BOTH,timestamp="2015.08.01.17.30.05"),1,1,"S/N", "Fall2015"*
(Assume the student with ID=1 has course with ID=1 in his registered course list)
3. *token(id=2,type=ADMIN,timestamp="2015.08.01.17.30.05"),1,10,"S/N", "Fall2015"*
(Assume the student with ID=1 has no course with ID=10 in his registered course list)
4. *token(id=0,type=STUDENT,timestamp="2015.08.01.17.30.05"),3,1,"A-F","Fall 2015"*
(the token type is not ADMIN or BOTH)
5. *token(type=UNDEFINED,timestamp="2015.08.01.17.30.05"),3,1,"A-F","Fall 2015"*
(the token type is not ADMIN or BOTH)

**Expected Results:**

1.For 1-2, return true
2.For 3-5, return false

**Dependencies:** 1

**Initialization:** Course Table, Administrator Table, StudentAndCourse Table are stored in the database

**Test Step:**

1.The user sends login request and a token object generated
2.The user sends the admin edit student registered class request by invokeISCRS.adminEditStudentRegisteredClass() method with inputs as specified in test inputs
3.Expected boolean value returned

## Test Case 12: test_adminAddStudentToClass

**Description:** Ensures the functionality of admin Add Student to Class. When user sends the adminAddStudentToClass request ,ISCRS.adminAddStudentToClass() is invoked. It takes *token, studentID, courseID, grading, courseTerm* as input and return a boolean value(true is admin successfully adds the student to the class, false otherwise).

**Test Inputs:**

1. *token(id=1,type=ADMIN,timestamp="2015.08.01.17.30.05"),1,1,"A-F", "Fall2015"*
(Assume the course with ID=1 is not full and the student with USERID=1 will not exceed 30 credits after the adding the course)
2. *token(id=2,type=BOTH,timestamp="2015.08.01.17.30.05"),1,1,"A-F", "Fall2015"*
(Assume the course with ID=1 is not full and the student with USERID=1 will not exceed 30 credits after the adding the course)
3. *token(id=0,type=STUDENT,timestamp="2015.08.01.17.30.05"),3,1,"A-F","Fall 2015"*
(the token type is not ADMIN or BOTH)
4. *token(type=UNDEFINED,timestamp="2015.08.01.17.30.05"),3,1,"A-F","Fall 2015"*
(the token type is not ADMIN or BOTH)
5. *token(id=1,type=ADMIN,timestamp="2015.08.01.17.30.05"),4,1,"A-F","Fall 2015"*
(Assume the course with ID=4 is not full and the student with USERID=1 will exceed 30 credits after the adding the course)
6. *token(id=1,type=ADMIN,timestamp="2015.08.01.17.30.05"),5,1,"A-F","Fall 2015"*
(Assume the course with ID=5 is full)

**Expected Results:**

1.For 1-2, return true
2.For 3-6, return false

**Dependencies:** 1

**Initialization:** Course Table, Administrator Table, StudentAndCourse Table are stored in the database

**Test Step:**

1.The user sends log in request and a token object generated
2.The user sends the admin add student to class request by invokeISCRS.adminAddStudentToClass() method with inputs as specified in test inputs
3.Expected boolean value returned

## Test Case 13: test_adminEditClass

**Description:** Ensures the functionality of administrator Edit Class. When user sends the admimEditClass request ,ISCRS.adminEditClass() is invoked. It takes *token, courseID, courseName, courseCredits, instructorID, firstDay, lastDay, classBeginTime, classEndTime, weekDays, location, type, prerequisite, description and department* as input and return a boolean value(true is admin successfully edits the class, false otherwise).

**Test Inputs:**

1. *token(id=1,type=ADMIN,timestamp=”2015.08.01.17.30.05”),3, “Operating Systems”, 4, 2, “20150913”, “20151208”, “09:30”, “10:45”, “Tu, Th”, “ME 321”, “Lecture”, “Intro to OS” and “CS”*
(Assume the course with ID=3 is empty)
2. *token(id=1,type=ADMIN,timestamp=”2015.08.01.17.30.05”),-1, “”, -1, -1, “”, “”, “”, “”, “”, “”, “”, “”, “good class” and “”*
(Assume the course with ID=3 is registered by some students)
3. *token(id=2,type=BOTH,timestamp=”2015.08.01.17.30.05”),3, “Operating Systems”, 4, 2, “20150913”, “20151208”, “09:30”, “10:45”, “Tu, Th”, “ME 321”, “Lecture”, “Intro to OS” and “CS”*
(Assume the course with ID=3 is empty)
4. *token(id=1,type=BOTH,timestamp=”2015.08.01.17.30.05”), -1, “”, -1, -1, “”, “”, “”, “”, “”, “”, “”, “”, “good class” and “”*
(Assume the course with ID=3 is registered by some students)
5. *token(id=0,type=STUDENT,timestamp=”2015.08.01.17.30.05”),3, “Operating Systems”, 4, 2, “20150913”, “20151208”, “09:30”, “10:45”, “Tu, Th”, “ME 321”, “Lecture”, “Intro to OS” and “CS”*
(the token type is not ADMIN or BOTH)
5. *token(type=UNDEFINED,timestamp=”2015.08.01.17.30.05”),3, “Operating Systems”, 4, 2, “20150913”, “20151208”, “09:30”, “10:45”, “Tu, Th”, “ME 321”, “Lecture”, “Intro to OS” and “CS”*
(the token type is not ADMIN or BOTH)
6. *token(id=1,type=ADMIN,timestamp=”2015.08.01.17.30.05”),4, “Operating Systems”, 4, 2, “20150913”, “20151208”, “09:30”, “10:45”, “Tu, Th”, “ME 321”, “Lecture”, “Intro to OS” and “CS”*
(Assume the course with ID=4 is registered by some students)
7. *token(id=1,type=ADMIN,timestamp=”2015.08.01.17.30.05”),5, -1, “”, -1, -1, “”, “”, “”, “”, “”, “”, “”, “”, “” and “”*
(Assume the course with ID=5 is registered by some students)

**Expected Results:**

1.For 1-4, return true
2.For 5-7, return false
**Dependencies:** 1
**Initialization:** Administrator Table, Course Table, Student Table, Instructor Table, InstructorAndCourse Table and StudentAndCourse Table are stored in the database
**Test Step:**

1.The user sends login request and a token object generated

2.The user sends the admin edit class request by invoke ISCRS.adminEditClass() method with inputs as specified in test inputs

3.Expected boolean value returned

## Test Case 14: test_adminDeleteClass

**Description:** Ensures the functionality of administrator Delete Class. When user sends the admimDeleteClass request ,ISCRS.adminDeleteClass() is invoked. It takes *token* and *courseID* as input and return a boolean value(true is admin successfully deletes the class, false otherwise). The course can be successfully deleted if no student has registered the class.

**Test Inputs:**

1. *token(id=1,type=ADMIN,timestamp="2015.08.01.17.30.05"),3*
(Assume the course with ID=3 is empty)
2. *token(id=2,type=BOTH,timestamp="2015.08.01.17.30.05"),3*
(Assume the course with ID=3 is empty)
3. *token(id=0,type=STUDENT,timestamp="2015.08.01.17.30.05"),3*
(the token type is not ADMIN or BOTH)
4. *token(type=UNDEFINED,timestamp="2015.08.01.17.30.05"),3*
(the token type is not ADMIN or BOTH)
5. *token(id=1,type=ADMIN,timestamp="2015.08.01.17.30.05"),4*
(Assume the course with ID=4 is registered by some students)

**Expected Results:**

1.For 1-2, return true
2.For 3-5, return false

**Dependencies:** 1

**Initialization:** Administrator Table, Course Table, Student Table and StudentAndCourse Table are stored in the database

**Test Step:**

1.The user sends login request and a token object generated
2.The user sends the admin delete class request by invoke ISCRS.adminDeleteClass() method with inputs as specified in test inputs
3.Expected boolean value returned

## Test Case 15: test_queryAdminPersonalData

**Description:** Ensures the functionality of queryAdminPersonalData. When user sends the queryAdminPersonalData request ,ISCRS.queryAdminPersonalData() is invoked, it takes *token* as input and return a list of Arraylist of expected result. The list should contain only one arraylist as the admin id is unique, and the arraylist stores the following information *ID,FIRSTNAME, LASTNAME, DATEOFBIRTH, GENDER, DEPARTMENT* in order. If query is not valid or no matching results, the method returns an empty list.

**Test Inputs:**
1. *token(id=0,type=ADMIN,timestamp=”2015.08.01.17.30.05”)*
2. *token(id=0,type=BOTH,timestamp=”2015.08.01.17.30.05”)*
3. *token(id=0,type=STUDENT,timestamp=”2015.08.01.17.30.05”)*, 0 (the token type is not ADMIN or BOTH)
4. *token(id=0,type=UNDEFINED,timestamp=”2015.08.01.17.30.05”)*, 0 (the token type is not ADMIN or BOTH)

**Expected Results:**
1.For 1-2, a list of Arraylist as specified in description
2.For 3-4, empty list

**Dependencies:** None
**Initialization:** Administrator Table stored in the database
**Test Step:**
1.The user sends log in request and a token object generated
2.The user sends the queryAdminPersonalData request by invokeISCRS.queryAdminPersonalData() method with inputs as specified in test inputs
3.Expected results returned

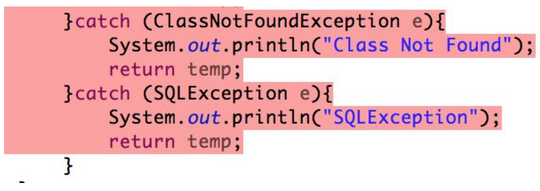
## Task 2

Report is in the file “Task2_CoverageReport_CreatedByEclEmma.zip”.
It is created by EclEmma.

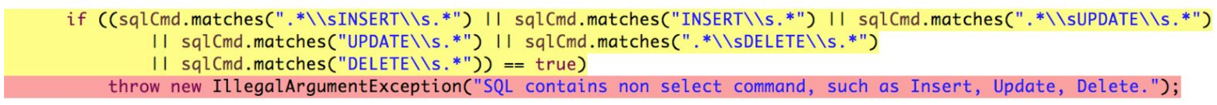
## Task 3

After we carefully set the test cases, the code coverage is about 80% for the whole project. And for the level ISCRS class, statement coverage(line coverage) is to 100%(missed 0/262). The other main functional classes, such as Course, Student, StudentAndCourse etc., the misses mainly happen because the exceptions will never be raised.

1. As said above, given the definition of given interface and our implementation, the exceptions IllegalArgumentException, ClassNotFoundException, SQLException, ParseException should never be raised since we can ensure that our arguments passed to the functions in DBcoordinator will always construct valid SQL commands; all class types passed in should exist, and all SQL statements can be successfully parsed due to our upper level function definitions. If the exception is raised, it indicates there are bugs in our code such that the incorrect sql cmd is sent to the DBcoordinator.



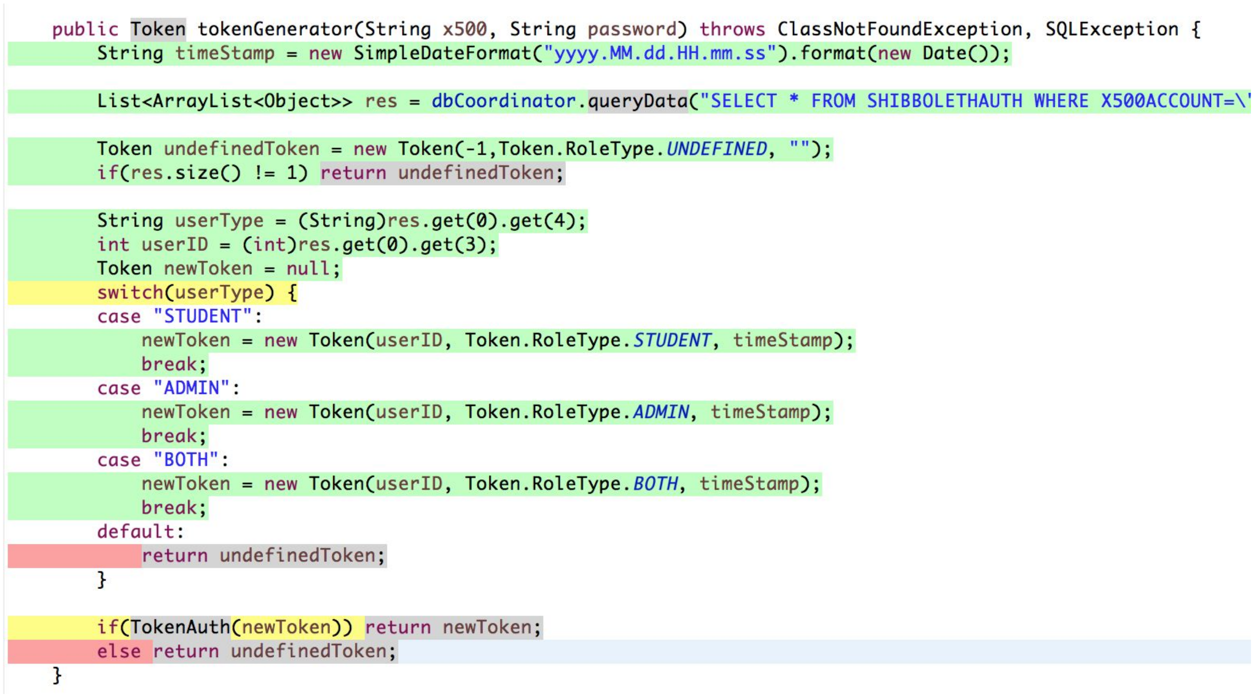
```
    }catch (ClassNotFoundException e){
        System.out.println("Class Not Found");
        return temp;
    }catch (SQLException e){
        System.out.println("SQLException");
        return temp;
    }
```



```
    if ((sqlCmd.matches(".*\\sINSERT\\s.*") || sqlCmd.matches("INSERT\\s.*") || sqlCmd.matches(".*\\sUPDATE\\s.*")
            || sqlCmd.matches("UPDATE\\s.*") || sqlCmd.matches(".*\\sDELETE\\s.*")
            || sqlCmd.matches("DELETE\\s.*")) == true)
        throw new IllegalArgumentException("SQL contains non select command, such as Insert, Update, Delete.");
```

2. Sometimes the it is the code logic that makes it impossible to cover all statements. There would be some statements that would never been executed given any input, if the code

is not written well. For example, In tokenGenerator() method of ShibbolethAuth Class, if all information in the database are correct(e.g. In the ShibbolethAuth Table, there would not be some user with USERTYPE='HAHA' or anything not 'STUDENT' or 'ADMIN' or 'BOTH' ),  our program will never get into default case in the switch() statement as below.



```
public Token tokenGenerator(String x500, String password) throws ClassNotFoundException, SQLException {
    String timeStamp = new SimpleDateFormat("yyyy.MM.dd.HH.mm.ss").format(new Date());

    List<ArrayList<Object>> res = dbCoordinator.queryData("SELECT * FROM SHIBBOLETHAUTH WHERE X500ACCOUNT=\'

    Token undefinedToken = new Token(-1,Token.RoleType.UNDEFINED, "");
    if(res.size() != 1) return undefinedToken;

    String userType = (String)res.get(0).get(4);
    int userID = (int)res.get(0).get(3);
    Token newToken = null;
    switch(userType) {
    case "STUDENT":
        newToken = new Token(userID, Token.RoleType.STUDENT, timeStamp);
        break;
    case "ADMIN":
        newToken = new Token(userID, Token.RoleType.ADMIN, timeStamp);
        break;
    case "BOTH":
        newToken = new Token(userID, Token.RoleType.BOTH, timeStamp);
        break;
    default:
        return undefinedToken;
    }

    if(TokenAuth(newToken)) return newToken;
    else return undefinedToken;
}
```

3.The lacking of data in the database also makes it difficult for us to cover all statements. Such as in the adminAddStudentToClass() and StudentAddClass() function, we have to insert some data in the database, such that there is a student who will exceed the credit limit after he/she register another course, in order to reach full line coverages.

## Task 4

The development of SCRS system is a great opportunity for us to practice and utilize the software engineering knowledge to solve real problem. During developing process, not only was programming skills strengthened but also we acquire a deep understanding about how the software process works. Software engineering is definitely not a just programming process. The whole design process is built up in a progressive way by five individual assignments, which requires us to have a rigorous thinking to avoid the conflicts in the following steps. Accomplishing each assignment can be treated as a milestone in our design process. The communication and coordination with different groups of people cannot be avoided. For instance, the user interface and database were designed by our teaching assistants and some design process have to be built on top of those components. We got through the whole process even though we encountered lots of confusion and problems in each stage. In the following paragraphs, we will address the problems in each stage and what we could have done better.

In milestone one, the purpose of constructing the requirements document is to be clear about what the software is used for and what functionality it features. First of all, we extracted all the user cases from the general requirement documents to help us to develop requirements. Requirements are very important since it reflects the need of customers. Moreover, the requirements are written in a natural language, which are easy to be understood by customers, or colleagues who do not have sufficient knowledge in this area. Furthermore, all requirements are written in details without any ambiguities. It is noted that the requirements are divided into two main categories, which are functional and non-functional. It is very straight forward for us to come up with the idea about the functional requirements but from the non-functional requirements' perspective, it is relatively hard for us due to the flexibility and some settings are based on the experience that are what we are lack of.

In milestone two, we were trying to revise requirement document we had created in milestone one and continue to develop a collection of requirement-based tests. The hardest part is to think about all situations including the exceptions during the requirement tests. For the positive requirements tests, it is very straightforward which is to achieve the major functionalities. However, exceptions have to be taken into considerations since the software are not only designed to handle valid input but also invalid or unexpected ones. It is noted that several requirements might have conflict situations, which require us to carefully state all potential conflicting problems and constraint the software to avoid potential risk. Looking back to this stage, we could have done better on the test parts. Even though we have tried very hard to come up with possibilities as many as we can, some interrelationship between different requirements are still out of the scope, which reflects in the stage five. For positive side, all requirements are well organized and met with customers' purpose.

In milestone three, we went into the design process including design structure, UML diagram and dynamic model sequential diagram. Before we dive into it, all of our teammates contribute their own ideas about how the design structure would look like. After discussion and combining each one's design benefits, architecture is finalized which exhibits four levels. Good design architecture is supposed to have properties of high cohesion and low coupling. On the top level, the classes are used to communicate with predefined user interface and all methods are called in the second level, which are categorized by administrator function, student function and instructor function. On the third level, classes are created based on methods provided in DBcoordinator. The last level is the classes that have direct interaction with database. The initial purpose of having second level is to group all functions that belong to each role that will use the software. However, it turns out to be confusing and useless since those classes will call all methods stored in the third levels. The major problem in this stage is to understand and coordinate with the files provided from our teaching assistant. Even though we have some freedom to our design, it is confined by other factors such as pre-set user interface and DBcoordinaor. It showed us how important the communication between groups. Moreover, in order to make the architecture more efficient, some redundant classes should be deleted such as the second level in our initial design. Accurate understanding about the other module designed by third party is important since they are parts of the final design and are coupled to our design. It gives us better visualization about the dynamic model by creating sequential diagram, where the corresponding functions are invoked in sequential steps.

In milestone four, the most fun part- coding part was undertaken. Even though the design in milestone three is carefully revised, it is inevitable to modify it here and there to adapt for the real coding process. Several redundant methods were got rid of and the system was further refined. Since our implementation is based on the user interface and DBcoordinator, we have to make changes according to the modification from those two modules, which makes the design process not as fluent as we expected.

In final stage, milestone five, the test implementation was undertaken. It is important since the test process guarantees all functionalities work and the software has the ability to handle all exceptions. The coverage of the test code can be calculated by either software tool and hand calculation. Even though this program is not very big, it is still very time consuming to manually count all coverage percentage. As a result, the test code was made to test the whole system. The problem is that the database does not have sufficient data for us to test all cases. We decided to firstly create all data in the database and then implemented all tests by codes.

Through this semester, we have not only gained sufficient knowledge about the software engineering and implemented the term project by going through from ground to final product. At the beginning of the semester, several software developing models were introduced such as waterfall model and spiral, etc. The waterfall is strict and efficient but far from realistic. It is just impossible to develop every component at one time without any changes later on. It is more common to use incremental model especially for the project where cooperation involves.

Moreover, the coordination between different groups, such as TAs, is also very important. Since the design process exhibits the property of modularity, how to couple the imported modules to ours is crucial. Some problems were introduced at the first time such as requirement-based tests at early stages, especially for non-functional requirements. Some other problems were not foreseen until the design process, such as design structure and required functions in different classes. It took a lot time to clarify the user cases and requirements, which seems to be waste of time initially but it benefits a lot in actual design process. The project could not go this well without all teammates' collaboration and contribution.