**Click Click Boom**

Phase III

4/12/12

Leader: Andrew Zurn

Preston Hardy

Wiley Radomski

**Comments**

**[This page is intentionally left blank]**

**Table of Contents**

Title Page………………………………………………………………………………………….1

Comments……………………………………………………………………………………........2

Table of Contents………………………………………………………………………………….3

Class Descriptions……………………………………………………………………………..4-27

Source Package………………………………………………………………………...4-25

AdminController..................................................................................................4-7

Admin.....................................................................................................................7

DatabaseController.............................................................................................7-13

FiveRecSchoolsController................................................................................13-14

Person...............................................................................................................14-15

SearchController...............................................................................................15-20

University.........................................................................................................20-24

UserController..................................................................................................24-26

User…...................................................................................................................26

Functional Testing Class………………………………………...…………………...27-28

Meeting Minutes…………………………………………………………...…………………29-30

Task Decomposition…………………………………………………………...……………...…31

**Class Descriptions – Source Package**

**AdminController** **–** Class AdminController will be used to serve the basic functions of the administrator for the "Choose My College" software package being constructedin CS230 of the Spring of 2012. The intended purpose of this class is toallow for the modifying of information important to the User and Universityclass also included in the csci230radomskizurnhardy package.

**public static void addUniversity(String school, String state, String location, String control, int numberOfStudents, double percentFemales, double SATVerbal, double SATMath,double expenses, double percentFinancialAid, int numberOfApplicants, double percentAdmitted, double percentEnrolled, int academicsScale,**

**int socialScale, int qualityOfLifeScale)**

Objective: Will be used to add a new university to the CMC system.

**Basic Test** of addUniversity (Use Case #15). This a rather simple test, as the function

case diagram merely needs to add the University to the database. To determine if the function

works properly, we will simple find the university within the list of universities in the database.

Input: new University

Output: Boolean - true, if school is found after the addUniversity method is called.

**public static void addUniversityEmphasis(String school, String emphasis)**

Objective: Will add an emphasis to a school.

**Basic Test** of addUniversityEmphasis method (Use Case 15/16), of class AdminController. Another rather simple test, this will add an emphasis to a given university, and then determine if the emphasis was successfully added to that university.

Input: Add Computer Science to RadomskiHardyZurnUniversityTest

Output: Boolean - true, if computer science can be found in this school's emphases.

**public static void addUser(String firstName, String lastName, String username,**

**String password, char type)**

Objective: Add a user to the database.

**Basic Test** of the addUser method of Admin Controller. This will add a user to the database, and then run through all the users in the database, and upon finding the added user, set a boolean to true. If it does not find it, the boolean will remain false.

Input: new User, username AZurn91

Output: Boolean - true, if the user AZurn91 is found within the database.

**public static void deactivateUser(int Id)**

Objective: Will deactivate the user within the CMC system.

**Basic Test** of Use Case 14: Deactivate User. The user will be deactivated in the database, and then will be found in the user list stored in the database, and its status will be returned. On a successful changing of its status, a boolean will be set to true. If not, the boolean will be false.

Input: deactivateUser of ID number 2

Output: Boolean - true, if status of user with ID number 2 is 'N'

**public static void deactivateUser(String username)**

Objective: Will deactivate the user within the CMC system.

**Basic Test** of Use Case 14: Deactivate User. The user will be deactivated in the database, and then will be found in the user list stored in the database, and its status will be returned. On a successful changing of its status, a boolean will be set to true. If not, the boolean will be false.

Input: deactivateUser of username "AZurn91"

Output: Boolean - true, if status of user with username "AZurn91" is 'N'

**public static void editUniversity(String school, String state, String location, String control, int numberOfStudents, double percentFemales, double SATVerbal, double SATMath,double expenses, double percentFinancialAid, int numberOfApplicants, double percentAdmitted, double percentEnrolled, int academicsScale,**

**int socialScale, int qualityOfLifeScale)**

Objective: Will edit a university within the CMC system.

**Basic Test** of editUniversity method (Use Case #16), of class AdminController. This is a rather simple test, that will edit the university, see if the university still exists within the database, and then determine if the fields that were edit, were indeed edited to the wanted correction. In this case, the STATE, NUMBER OF STUDENTS, EXPENSES, and ACADEMIC SCALE will be altered.

Input: School: RadomskiHardyZurnUniversityTest Edit: State: Iowa, Number of Students: 10000, Expenses: 15500.0, Academics Scale: 3

Output: Boolean - true, if the above changes are found to be true.

**public static void editUniversityEmphases(String school, ArrayList<String> emphases)**

Objective: Will edit the emphases of a university.

**Basic Test**, that will use a list of strings that contain emphases and edit the list stored within a school with them. The school will then be found in the database, and tested to see if its emphases have been altered. If the alteration was successful, a boolean will be set to true; if not, the boolean will be set to false.

Input: Will edit RadomskiHardyZurnUniversityTest's emphases to include Science, Math, and Computer Science.

Output: Boolean - true, if the above changes are found to be true.

**public static void editUser(String username, String firstName, String lastName, String password)**

Objective: Will edit a user within the CMC system.

**Basic Test**, to see if the first name of a specific user can be changed in the system. If it was changed correctly, a boolean will be set to true, if not, it will be set to false.

Input: Edit username: AZurn91 Change firstName: Andy

Output: Boolean - true, if the firstName is changed to Andy

**public static void editUserByAdmin(String username, String firstName, String lastName, String password, char type, char status)**

Objective: Will edit a user with administrator level access.

**Basic Test**, to see if the first name and status of a specific user can be changed in the system. If it was changed correctly, a boolean will be set to true, if not, it will be set to false.

Input: Edit username: AZurn91 Changes firstName: Andrew, status: 'N'

Output: Boolean - true, if firstName is changed to Andrew, and status is changed to 'N'

**public static ArrayList<String> getAllPossibleEmphases()**

Objective: Will return a list of all possible emphases.

**Basic Test** of getAllPossibleEmphases method, of class AdminController.

Output: An ArrayList<String> that is not empty. Its first value should be ACCOUNTING.

**public static TreeMap<Integer, ArrayList<String>> getAllSavedSchools()**

Objective: Will return a map of all the users and their associated schools within the CMC system.

**Basic Test** of getAllUniversities method, of class AdminController.

Output: ArrayList<University> with the entire list of universities within the database.

**public static ArrayList<University> getAllUniversities()**

Objective: Will return a list of all the universities within the CMC system.

**Basic Test** of getAllSavedSchools method, of class AdminController.

Output: A TreeMap<Integer, ArrayList<String>> that is not empty.

**public static TreeMap<String, ArrayList<String>> getAllUniversityNamesWithEmphases()**

Objective: Will return all the universities with a list of all the emphases.

**Basic Test** to determine if the database is returning something other than null for the getAllUniversityNamesWithEmphases method.

Output: A TreeMap<String, ArrayList<String>> that contains each school and their emphases.

**public static ArrayList<User> getAllUsers()**

Objective: Will return an ArrayList of all users within the CMC system.

**Basic Test** to determine if the database is returned something other than a empty array list.

Output: An ArrayList<User> of each user found in the database.

**public static User getSpecificUser(Integer Id)**

Objective: Will return a specific user within the CMC system.

**Basic Test** to see if the database will return a specific user from the user list.

Input: ID number 2

Output: "juser" has ID number 2

**public static User getSpecificUser(String username)**

Objective: Will return a specific user within the CMC system.

**Basic Test** to see if the database will return a specific user from the user list.

Input: "juser"

Output: "juser"

**public static boolean login(String username, String givenPassword)**

Objective: Logs on Admin if username and password are correct

**Black Box Test** of the login method.

Case 1: a good password will return a boolean that is true.

Case 2: a bad password will return a boolean that is false.

Input: 1: AZurn91, password 2: AZurn91, PASSWORD

Output: 1: success 2: failed

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Admin –** A class that will be used to represent an administrator within the CMC system.

**Please reference the Person class to find methods associated with the Admin class.**

**Test for these methods can be found in the AdminTest and UserTest.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**DatabaseController –** This is the only class that deals directly with the database. Anything requesting or wanting information from the database must go through this class, the DatabaseController. It converts the output from the database into more usable formats such as ArrayLists and TreeMaps.

**public static void addUniversity(String school, String state, String location, String control, int numberOfStudents, double percentFemales, double SATVerbal, double SATMath,double expenses, double percentFinancialAid, int numberOfApplicants, double percentAdmitted, double percentEnrolled, int academicsScale,**

**int socialScale, int qualityOfLifeScale)**

Objective:When called, this method tells the database to create a new university record using the information provided in the parameters. Callers of this method must ensure that the school name is unique among all universities, otherwise, an SQLException is thrown by the corresponding method.

**Basic Test** of addUniversity. This a rather simple test, as the function case diagram merely needs to add the University to the database. To determine if the function works properly, we will simple find the university within the list of universities in the database.

Input: new University

Output: Boolean - true, if school is found after the addUniversity method is called.

**public static void addUniversityEmphasis(String school, String emphasis)**

Objective:When called, this method tells the database to add a new emphasis areaf or the specified school. Nothing happens if an invalid school name is specified or if the specified emphasis already exists for the specified school.

**Basic Test** of addUniversityEmphasis method of class DatabaseController. Another rather simple test, this will add an emphasis to a given university, and then determine if the emphasis was successfully added to that university.

Input: Add MATH to "ClickClickBoomUniversity1"

Output: Boolean - true, if math can be found in this school's emphases.

**public static void addUser(String firstName, String lastName, String username, String password, char type)**

Objective: When called, this method tells the database to create a new user record using the information provided in the parameters. The user will be active by default and will be assigned a unique Id equal to one plus the current largest Id among all users in the database. Callers of this method must ensure that the specified username is unique among all users, otherwise, an SQLException is thrown by the database method.

**Basic Test** of the addUser method of DatabseController. This will add a user to the database, and then run through all the users in the database, and upon finding the added user, set a boolean to true. If it does not find it, the boolean will remain false.

Input: new User, username CCBUsername3

Output: Boolean - true, if the user CCBUsername3 is found within the database.

**public static boolean addUserSchool(int id, String school)**

Objective: When called, this method has the database save the specified school to the user's list of saved schools. Nothing happens if an invalid user Id is specified or if the specified school is already in user's list of saved schools.

**Basic Test** of the addUserSchool method of DatabseController. This will add a school to a user's saved school list and then check to see if that user has that saved school. If it does not find it, the boolean will remain false.

Input: User with ID:2 and school name: "ClickClickBoomUniversity1"

Output: Boolean - true, if the user ID:2 has "ClickClickBoomUniversity1"

**public static void deactivateUser(int Id)**

Objective: When called, this method has the database change the status of the user, whose Id is specified as a parameter, to inactive. In other words, it sets user status to 'N' instead of 'Y'. Nothing happens if an invalid Id is specified.

**Basic Test** deactivateUser(int ID) method of DatabseController. The user will be deactivated in the database, and then will be found in the user list stored in the database, and its status will be returned. On a successful changing of its status, a boolean will be set to true. If not, the boolean will be false.

Input: deactivateUser of ID number 1

Output: Boolean - true, if status of user with ID number 1 is 'N'

**public static void deactivateUser(String username)**

Objective:When called, this method has the database change the status of the user, whose username is specified as a parameter, to inactive. In other words, it sets user status to 'N' instead of 'Y'. Nothing happens if an invalid username is specified.

**Basic Test** deactivateUser(String username) method of DatabseController. The user will be deactivated in the database, and then will be found in the user list stored in the database, and its status will be returned. On a successful changing of its status, a boolean will be set to true. If not, the boolean will be false.

Input: deactivateUser of username "CCBUsername1"

Output: Boolean - true, if status "CCBUsername1" is 'N'

**public static void editUniversity****(String school, String state, String location, String control, int numberOfStudents, double percentFemales, double SATVerbal, double SATMath,double expenses, double percentFinancialAid, int numberOfApplicants, double percentAdmitted, double percentEnrolled, int academicsScale,**

**int socialScale, int qualityOfLifeScale)**

Objective: When called, this method has the database update the university record for the university whose name is specified as a parameter, using the information provided in the parameters. This method can update all university fields except for the emphases which are processed separately. Nothing happens if an invalid university name is specified.

**Basic Test** of editUniversity method of class DatabaseController. This is a rather simple test, that will edit the university, see if the university still exists within the database, and then determine if the fields that were edited, were indeed edited to the desired correction. In this case the STATE will be altered.

Input: School: "ClickClickBoomUniversity1" Edit: State: Iowa

Output: Boolean - true, if the above changes are found to be true.

**public static void editUniversityEmphases(String school, ArrayList<String> emphases)**

Objective: When called, this method has the database delete all the old emphasis areas for the specified school from the database and replaces them with the ones specified in the ArrayList parameter. Nothing happens if an invalid school name is specified.

**Basic Test** of editUniversityEmphases method of class DatabaseController. This is a rather simple test, that will edit the university's emphases, see if the emphases were added into the database, and then determine if the fields that were edited, were indeed edited to the desired correction. In this case, SCIENCE is added as an emphases

Input: School: "ClickClickBoomUniversity1" with emphases SCIENCE

Output: Boolean - true, if the above changes are found to be true.

**public static void editUser(String username, String firstName, String lastName, String password)**

Objective: When called, this method has the database update the record of the user, whose username is specified as a parameter, using the information provided in the parameters. The user fields that can be modified are: first name, last name and password. This method will not modify the rest of the fields in the user's record. Nothing happens if an invalid username is specified.

**Basic Test** of editUser method of class DatabaseController. This is a rather simple test, that will add and edit a user, get the use from within the database, and then determine if the fields that were edited, were indeed edited to the desired correction.

Input: Username: "CCBUsername1" Edit: firstName = "ClickClickBoomUser1Edit"; lastName = "CCBUser1Edit"; password = "CCBPasswordEdit";

Output: 3 Booleans - true, if the above changes are found to be true.

**public static void editUserByAdmin(int ID, String firstName, String lastName, String password, char type, char status)**

Objective: When called by an administrator, this method has the database update the record of the user, whose username is specified as a parameter, using the information provided in the parameters. The user fields that can be modified are: first name, last name, password, type, and status. Nothing happens if an invalid username is specified.

**Basic Test** of editUserByAdmin\_ID method of class DatabaseController. This is a rather simple test, that will add and edit a user, get the use from within the database, and then determine if the fields that were edited, were indeed edited to the desired correction.

Input: ID:2 Edit: firstName = "ClickClickBoomUser2Edit"; lastName = "CCBUser2Edit"; password = "CCBPassword2Edit"; type = 'u';

Output: 4 Booleans - true, if the above changes are found to be true.

**public static void editUserByAdmin(String username, String firstName, String lastName, String password, char type, char status)**

Objective: When called by an administrator, this method has the database update the record of the user, whose username is specified as a parameter, using the information provided in the parameters. The user fields that can be modified are: first name, last name, password, type, and status. Nothing happens if an invalid username is specified.

**Basic Test** of editUserByAdmin\_USERNAME method of class DatabaseController. This is a rather simple test, that will add and edit a user, get the use from within the database, and then determine if the fields that were edited, were indeed edited to the desired correction.

Input: Username: "CCBUsername2"; Edit: firstName = "ClickClickBoomUser2Edit2"; lastName = "CCBUser2Edit2"; password = "CCBPassword2Edit2"; type = 'u'; status = 'N'

Output: 4 Booleans - true, if the above changes are found to be true.

**Basic Test** of editUserByAdmin\_USERNAME method of class DatabaseController. This is a rather simple test, that will add and edit a user, get the use from within the database, and then determine if the fields that were edited, were indeed edited to the desired correction.

Input: Username: "CCBUsername2"; Edit: firstName = "ClickClickBoomUser2Edit2"; lastName = "CCBUser2Edit2";

password = "CCBPassword2Edit2"; type = 'u'; status = 'N'

Output: 4 Booleans - true, if the above changes are found to be true.

**public static ArrayList<String> getAllPossibleEmphases()**

Objective: When called, this method returns an ArrayLsit of Strings containing all possible university emphases that occur in the database. Every array row contains a single field storing an occurring emphasis. Duplicate emphases are eliminated.

**Basic Test** of getAllPossibleEmphases method of class DatabaseController. This is a rather simple test that will check to see if a few examples are contained within the list of all emphases.

Input: MATH and LIBERAL-ARTS

Output: 2 Booleans - true, if the above emphases are present

**public static TreeMap<Integer, ArrayList<String>> getAllSavedSchools()**

Objective: When called, this method returns a TreeMap of Integers and an ArrayList of Strings containing all user Ids along with their saved schools in the database. Users with multiple saved schools will have their schools in an ArrayList corresponding to the Ids. Users with no saved schools will not be included in the TreeMap.

**Basic Test** of getAllSavedSchools method of class DatabaseController. This is a rather

simple test that will add a school to a user's saved universities. Then it will make sure

that school was added.

Input: ID:2 and "ClickClickBoomUniversity1"

Output: Boolean - true, if the above saved school is present in user ID:2's list

**public static ArrayList<University> getAllUniversities()**

Objective: When called, this method returns an ArrayList of Universities containing all universities in the database. Every University is made up of the following fields: School String (must be unique among universities), State String, Location String, Control String, NumberOfStudents int, PercentFemales double (between 0 and 100), SATVerbal double (up to 800), SATMath double (up to 800), Expenses double, PercentFinancialAid double (between 0 and 100), NumberOfApplicants int, PercentAdmitted double (between 0 and 100), PercentEnrolled double (between 0 and 100), AcademicsScale int (between 1 and 5 where is best), SocialScale int (between 1 and 5 where is best), QualityOfLifeScale int (between 1 and 5 where is best). \* Please note that some universities have missing field information which have been indicated by the String "-1" for String fields and the number -1 for number fields.

**Basic Test** of getAllUniversities method of class DatabaseController. This is a rather simple test that will check to see if the array that gets the universities is not null.

Input: ArrayList<University> result = DatabaseController.getAllUniversities();

Output: Boolean - true, if the above array is not null

**public static TreeMap<String, ArrayList<String>> getAllUniversityNamesWithEmphases()**

Objective: When called, this method returns a TreeMap of Strings and an ArrayList of Strings containing all university names along with their emphases in the database. Universities with multiple emphases will have their names once in the TreeMap and their corresponding ArrayList will contain the different emphasis. Universities with no emphases will not be included in the output.

**Basic Test** of getAllUniversityNamesWithEmphases method of class DatabaseController. This is a rather simple test, that will add to a university's emphases, see if that emphasis was added into the database, and then determine if the fields that were edited, were indeed edited to the desired correction. In this case, MATH is added as an emphases to "ClickClickBoomUniversity1"

Input: School: "ClickClickBoomUniversity1" with emphases MATH

Output: Boolean - true, if the above changes are found to be true.

**public static ArrayList<User> getAllUsers()**

Objective: When called, this method returns an ArrayList of Users containing all users in the database. Every User is made up of the following fields: Id int (a new unique number is assigned for every user), FirstName String, LastName String, Username String (must be unique among users), Password String, Type char (can be either 'a' for administrators or 'u' for regular users), Status char (can be either 'Y' for active accounts or 'N' inactive ones).

**Basic Test** of getAllUsers method of class DatabaseController. This is a rather simple test that will check to see if the array that gets the users is not null.

Input: ArrayList<User> result = DatabaseController.getAllUsers();

Output: Boolean - true, if the above array is not null

**public static int getID(String username)**

Objective: When called, this method returns an integer referring to the ID number of the specified User. If the parameter username matches one in the database, it will be returned. Otherwise, a -1 is returned.

**Basic Test** of the getID method of DatabseController. This will check to see that the ID number of a know user is fetched correctly,

Input: User with username: "nadmin"

Output: Boolean - true, if this username returns an ID of 1.

**public static User getSpecificUser(int Id)**

Objective: When called, this method returns a User containing the int ID specified as a parameter.

**Basic Test** of getSpecificUser\_int() method of class DatabaseController. This is a rather simple test, that will edit a user(in case the entries have changed since the creation of this test), get the user from within the database by the ID number, and then determine if the fields that were edited, refer to the edited user.

Input: ID:2 Edit: firstName = "ClickClickBoomUser2Edit2"; lastName = "CCBUser2Edit2"; password = "CCBPassword2Edit2"; type = 'u'; status = 'N'

Output: 4 Booleans - true, if the above changes are found to be true.

**public static User getSpecificUser(String username)**

Objective: When called, this method returns a User containing the User whose Username is specified as a parameter.

**Basic Test** of getSpecificUser\_String() method of class DatabaseController. This is a rather simple test, that will edit a user(in case the entries have changed since the creation of this test), get the user from within the database by the username, and then determine if the fields that were edited, refer to the edited user.

Input: "CCBUsername2";

Output: 4 Booleans - true, if the above changes are found to be true.

**public static void removeUserSchool(int Id, String school)**

Objective: When called, this method asks the Database to remove the specified school from the user's list of saved schools. Nothing happens if an invalid user Id is specified or if the specified school is not already in user's list of saved schools

**Basic Test** of the removeUserSchool method of DatabseController. This will remove a school from a user's saved school list and then check to see if that user does not have that saved school.

Input: User with ID:2 and school name: "ClickClickBoomUniversity1"

Output: Boolean - true, if the user ID:2 does not have "ClickClickBoomUniversity1"

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**FiveRecSchoolsController–** Will test the methods associated with the Five Rec Schools Controller class.

**private void setMinAndMax()**

Objective: Used to set comparative values in the case that the numbers received are null.

**Testing** for this case is done with getFiveMatches(), as that method is reliant on this private method to succeed.

**private void findDifference()**

Objective: Computes the distance each school is from the wanted school, and assigns a value to each of those schools.

**Testing** for this case is done with getFiveMatches(), as that method is reliant on this private method to succeed.

**private void findFiveClosest()**

Objective: Finds the five closest values to the searched school, and adds their corresponding university to the array list that is to be returned to the user.

**Testing** for this case is done with getFiveMatches(), as that method is reliant on this private method to succeed.

**public ArrayList<University> getFiveMatches()**

Objective: Returns that five closest schools to the searched school in the form of an array list.

**Basic Test** of getFiveMatches method, of class FiveRecSchoolsController. Prior to running this test, 6 very similar schools are added to the database. One of the schools is then fed through the search algorithm and we must assert that the other 5 similar schools are the 5 that are returned.

Input: School with name "ClickClickBoomSchool1"

Output: the other 5 schools ranging in ClickClickBoomSchool[2-6]

The test fails if any one of them is not found.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Person–** A class that will be extended by Admin and User to hold data values that are similar between the two classes.

**public String getFirstName()**

Objective: Will return the first name of the person.

**Testing** is done within setFirstName(), as it is reliant on returning firstName to succeed.

**public void setFirstName(String firstName)**

Objective: Will set the first name of the person.

**Basic Test** of setFirstName.

Input: Andy

Output: Andy

**public String getLastName()**

Objective: Will return the last name of the person.

**Testing** is done within setLastName(), as it is reliant on returning lastName to succeed.

**public void setLastName(String lastName)**

Objective: Will set the last name of the person.

**Basic Test** of setLastName

Input: NotZurn

Input: NotZurn

**public String getUserName()**

Objective: Will return the first name of the person.

**Testing** is done within setUserName(), as it is reliant on returning userName to succeed.

**public void setUserName(String userName)**

Objective:Will set the username of the person.

**Basic Test** of setUserName

Input: Awzurn1991

Output: Awzurn1991

**public String getPassword()**

Objective: Will return the password of the person.

**Testing** is done within setPassword(), as it is reliant on returning password to succeed.

**public void setPassword(String password)**

Objective: Will set the password of the person.

**Basic Test** of setPassword

Input: PASSWORD

Output: PASSWORD

**public char getType()**

Objective: Will return the type of user that the person is.

**Testing** is done within setType(), as it is reliant on returning type to succeed.

**public void setType(char type)**

Objective: Will set the type of user that the person is.

**Basic Test** of setType

Input: u

Output: u

**public char getStatus()**

Objective: Will return the status of user to be associated with the person.

**Testing** is done within setStatus(), as it is reliant on returning status to succeed.

**public void setStatus(char status)**

Objective:Will set the status of user to be associated with the person.

**Basic Test** of setStatus

Input: N

Output: N

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**SearchController –** This class will be used to perform the searching that a user will need to find schools that they are interested in, and also schools that should be recommended for them to consider.

**public static ArrayList<University> search(String school, String state, String location, String control, int numberOfStudents, double percentFemales, double SATVerbal, double SATMath,double expenses, double percentFinancialAid, int numberOfApplicants, double percentAdmitted, double percentEnrolled, int academicsScale,**

**int socialScale, int qualityOfLifeScale)**

Objective: This method will call upon other private methods within this class to compile a map of all relevant schools to the user's query and also recommended schools that the user should consider.

This method is tested by testing its individual parameters, of which hold their own private search methods. Please see below.

Five combination tests are also done; their descriptions are listed below.

**public void test\_SearchCombo1()**

**Black Box Test** for a combination of search criteria.

Equivalence case: Multiple search entries - State, Number of Students, Percent Females

Input: wantedState - CCBState1, lowWantedNumber - 3999 highWantedNumber - 4001, lowPercentFemales - 44.9 highPercentFemales - 45.1

Output: School in the state of CCBState1, has 4000 students, and 45 percent females.

In this case, school ClickClickBoomSchoolD matches the criteria.

**public void test\_SearchCombo2()**

**Black Box Test** for a combination of search criteria.

Equivalence case: Multiple search entries - SAT Math, Number of Applicants, Percent Admitted

Input: lowSATMath - 399.0 highSATMath - 401.0, lowNumApplicants - 7999 highNumApplicants – 8001, lowAdmitted - 44.9 highAdmitted - 45.1

Output: School in the with SATMath score of 400, 8000 number of applicants, and 60 percent admitted.

In this case, school ClickClickBoomSchoolD matches the criteria.

**public void test\_SearchCombo3()**

**Black Box Test** for a combination of search criteria.

Equivalence case: Multiple search entries - State, Location, Control

Input: wantedState - CCBState1, wantedLocation - Suburban, wantedControl - Public

Output: There is no school that is located in a Suburban location.

Null is returned.

**public void test\_SearchCombo4()**

**Black Box Test** for a combination of search criteria.

Equivalence case: Multiple search entries - State and Location

Input: wantedState - CCBState1, wantedLocation - Urban

Output: School in the state of CCBState1, with an Urban location.

In this case, schools ClickClickBoomSchoolD and ClickClickBoomSchoolE match the criteria.

**public void test\_SearchCombo5()**

**Black Box Test** for a combination of search criteria.

Equivalence case: Multiple search entries - Percent Enrolled, Academic, Social and Quality Scale

Input: lowEnrolled - 74.9 highEnrolled 75.1, lowAcademicScale - 5 highAcademicScale – 5 lowSocialScale - 3 highSocialScale - 3, lowQualityScale - 5 highQualityScale - 5

Output: School in the with SATMath score of 400, 8000 number of applicants, and 60 percent admitted.

In this case, school ClickClickBoomSchoolA and ClickClickBoomSchoolJ match the criteria.

**private static void searchWantedSchools(String wantedSchool)**

Objective: User to keep track of schools that are still eligible in the search for the group of universities that will be returned to the user.

**Basic Test** of the search school method. ClickClickBoomSchool A is search, and properly returned.

Input: ClickClickBoomSchoolA

Output: ClickClickBoomSchoolA

**private static void searchWantedState(String wantedState)**

Objective: Will search the name of the given school against the list of schools in the database.

**Basic Test** of the search state method. The first item in the returned ArrayList will hold the value CCBState1 for its state.

Input: CCBState1

Output: School holding CCBState1 as its state.

**private static void searchWantedLocation(String wantedLocation)**

Objective:Will search the name of the given state against the list of schools in the database.

**Basic Test** of the search location method. The first item in the returned ArrayList will hold the value Urban for its location.

Input: Urban

Output: School holding Urban as its location.

**private static void searchWantedControl(String wantedControl)**

Objective: Will search the name of the given control against the list of schools in the database.

**Basic Test** of the search control method. The first item in the returned ArrayList will hold the value Private for its control.

Input: Private

Output: School holding Private as its control.

**private static void searchWantedNumberOfStudents(Integer lowNumberWanted, Integer highNumberWanted)**

Objective: Will search the name of the given control against the list of schools in the database.

**Basic Test** of the search number of students method. The first item in the returned ArrayList will hold the value 1000 number of students

Input: Lower - 999 Upper - 1001

Output: School containing 1000 students

**private static void searchWantedPercentFemales(Integer lowPercentFemales, Integer highPercentFemales)**

Objective: Will search the name of the given percent of females against the list of schools in the database.

**Basic Test** of the search percentage of females method. The first item in the returned ArrayList will hold the value 75.0 percent females.

Input: Lower - 74.0 Upper - 76.0

Output: School with 75 percent females.

**private static void searchWantedSATVerbal(Double lowSATVerbal, Double highSATVerbal)**

Objective: Will search the name of the given percent of females against the list of schools in the database.

**Basic Test** of the search SATVerbal method. The first item in the returned ArrayList will hold the value 450 for its SATVerbal score.

Input: Lower - 449.0 Upper - 451.0

Output: School with 450 SAT verbal score.

**private static void searchWantedSATMath(Double lowSATMath, Double highSATMath)**

Objective: Will search the name of the given SAT verbal score against the list of schools in the database.

**Basic Test** of the search SATMath method. The first item in the returned ArrayList will hold the value 400 for its SATMath score.

Input: Lower - 399.0 Upper - 401.0

Output: School with 400 SAT math score.

**private static void searchWantedExpenses(Double lowExpenses, Double highExpenses)**

Objective: Will search the name of the given expense against the list of schools in the database.

**Basic Test** of the search Expenses method. The first item in the returned ArrayList will hold the value 40000 for its expenses.

Input: Lower - 39999.0 Upper - 40001.0

Output: School with 40000 expense.

**private static void searchWantedPercentFinancialAid(Double lowFinAid, Double highFinAid)**

Objective: Will search the name of the given percent of financial aid against the list of schools in the database.

**Basic Test** of the search percent financial aid method. The first item in the returned ArrayList will hold the value 55.0 for its percent financial aid.

Input: Lower - 54.9 Upper - 55.4

Output: School with 55.0 percent financial aid.

**private static void searchWantedNumberOfApplicants(Integer lowNumApplicants, Integer highNumApplicants)**

Objective: Will search the name of the given number of applicants against the list of schools in the database.

**Basic Test** of the search number of applicants method. The first item in the returned ArrayList will hold the value 16000 for its number of applicants.

Input: Lower - 15999 Upper - 16001

Output: School with 16000 number of applicants.

**private static void searchWantedPercentAdmitted(Double lowAdmitted, Double highAdmitted)**

Objective: Will search the name of the given percent admitted against the list of schools in the database.

**Basic Test** of the search percent admitted method. The first item in the returned ArrayList will hold the value 60 for percent admitted.

Input: Lower - 59.9 Upper - 60.1

Output: School with 60 percent admitted.

**private static void searchWantedPercentEnrolled(Double lowEnrolled, Double highEnrolled)**

Objective: Will search the name of the given percent enrolled against the list of schools in the database.

**Basic Test** of the search percent enrolled method. The first item in the returned ArrayList will hold the value 75.0 for its percent enrolled.

Input: Lower - 74.9 Upper - 75.1

Output: School with 75 percent enrolled.

**private static void searchWantedAcademicsScale(Integer lowAcademicScale, Integer highAcademicScale)**

Objective: Will search the name of the given academics score against the list of schools in the database.

**Basic Test** of the search academics scale method. The first item in the returned ArrayList will hold the value 5 for its academic score.

Input: Lower - 5 Upper - 5

Output: School with 5 for academic score.

**private static void searchWantedSocialScale(Integer lowSocialScale, Integer highSocialScale)**

Objective: Will search the name of the given social score against the list of schools in the database.

**Basic Test** of the search social scale method. The first item in the returned ArrayList will hold the value 5 for its social score.

Input: Lower - 5 Upper - 5

Output: School with 5 for social score.

**private static void searchWantedQualityOfLifeScale(Integer lowQualityScale, Integer highQualityScale)**

Objective: Will search the name of the given quality of life score against the list of schools in the database.

**Basic Test** of the search quality of live scale method. The first item in the returned ArrayList will hold the value 5 for its quality of life score.

Input: Lower - 5 Upper - 5

Output: School with 5 for quality of life score.

**private static void searchWantedEmpheses(ArrayList<String> emphases)**

Objective: Will search through the given emphases, and compare them to all the schools to find the schools with like emphases.

**Basic Test** of the search emphases method. The first item in the returned ArrayList will have Computer Science and Math as emphases.

Input: ArrayList<String> containing 0: Computer Science, 1: Math

Output: School containing emphases of Computer Science and Math

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**University –** This class creates an object of type University complete with all attributes and the ability to set and get the University's Emphases.

**public String getSchool()**

Objective: Get name of the school.

**Testing** for this method is done in setSchool(), as that method is reliant on getSchool() to succeed.

**public void setSchool(String school)**

Objective: Set name of the school.

**Basic Test** of setting and getting a school name.

Input: "testSchool"

Output: "testSchool" set to name of University

**public String getState()**

Objective: Get state the school is in.

**Testing** for this method is done in setState(), as that method is reliant on getState() to succeed.

**public void setState(String state)**

Objective: Set state the school is in.

**Basic Test** of setting and getting the state of a University.

Input: MN"

Output: "MN" set state for a University

**public String getLocation()**

Objective: Get location of the school.

**Testing** for this method is done in setLocation(), as that method is reliant on getLocation() to succeed.

**public void setLocation(String location)**

Objective: Set location of the school.

**Basic Test** of setting and getting a location of a University.

Input: "rural"

Output: "rural" set location for a University

**public String getControl()**

Objective: Get control of the school.

**Testing** for this method is done in setControl(), as that method is reliant on getControl() to succeed.

**public void setControl(String control)**

Objective: Set control of the school.

**Basic Test** of setting and getting the control of a University.

Input: "private"

Output: "private" set control for a University

**public int getNumberOfStudents()**

Objective: Get number of students at the school.

**Testing** for this method is done in setNumberOfStudents(), as that method is reliant on getNumberOfStudents() to succeed.

**public void setNumberOfStudents(int numberOfStudents)**

Objective: Set number of students at the school.

**Basic Test** of setting and getting the number of students at a University.

Input: 100

Output: A university with a set number of students 100

**public double getPercentFemales()**

Objective: Get percent of female students at the school.

**Testing** for this method is done in setPercentFemales(), as that method is reliant on getPercentFemales() to succeed.

**public void setPercentFemales(double percentFemales)**

Objective: Set percent of female students at the school.

**Basic Test** of setting and getting the percent females at a University.

Input: 50.0

Output: A university with a set percent female 50.0

**public double getSATVerbal()**

Objective: Get the SAT Verbal score of the school.

**Testing** for this method is done in setSATVerbal(), as that method is reliant on getSATVerbal() to succeed.

**public void setSATVerbal(double SATVerbal)**

Objective: Set the SAT Verbal score of the school.

**Basic Test** of Setting and getting the SATVerbal of a University.

Input: 1.0

Output: A university with a set SATVerbal 1.0

**public double getSATMath()**

Objective: Get the SAT Math score of the school.

**Testing** for this method is done in setSATMath(), as that method is reliant on getMath() to succeed.

**public void setSATMath(double SATMath)**

Objective: Set the SAT Math score of the school.

**Basic Test** of setting and getting the SATMath of a University.

Input: 1.0

Output: A university with a set SATMath 1.0

**public double getExpenses()**

Objective: Get the expense for the school.

**Testing** for this method is done in setExpenses(), as that method is reliant on getExpenses() to succeed.

**public void setExpenses(double expenses)**

Objective: Set the expense for the school.

**Basic Test** of Setting and Getting the expenses of a University.

Input: 1.0

Output: A university with a set expenses 1.0

**public double getPercentFinancialAid()**

Objective: Get the percent of finanicial aid from the school.

**Testing** for this method is done in setPercentFinancialAid(), as that method is reliant on getPercentFinancialAid() to succeed.

**public void setPercentFinancialAid(double percentFinancialAid)**

**Basic Test** of setting and getting percent financial aid offered by a University.

Input: 1.0

Output: A university with a set Percent Financial aid 1.0

**public int getNumberOfApplicants()**

Objective: Get number of applicants to the school.

**Testing** for this method is done in setNumberOfApplicants(), as that method is reliant on getNumberOfApplicants() to succeed.

**public void setNumberOfApplicants(int numberOfApplicants)**

Objective: Set number of applicants to the school.

**Basic Test** of setting and getting the number of applicants a University has.

Input: 1

Output: A university with a set number of applicants 1

**public double getPercentAdmitted()**

Objective: Get percent of applicants admitted to the school.

**Testing** for this method is done in setPercentAdmitted(), as that method is reliant on getPercentAdmitted() to succeed.

**public void setPercentAdmitted(double percentAdmitted)**

Objective: Set percent of applicants admitted to the school.

**Basic Test** of setting and getting percent admitted to a University

Input: 1.0

Output: A university with a set percent admitted 1.0

**public double getPercentEnrolled()**

Objective: Get the percent of students admitted that enroll at the school.

**Testing** for this method is done in setPercentEnrolled(), as that method is reliant on getPercentEnrolled() to succeed.

**public void setPercentEnrolled(double percentEnrolled)**

Objective: Set the percent of students admitted that enroll at the school.

**Basic Test** of setting and getting percent enrolled at a university

Input: 1.0

Output: A university with a set percent enrolled 1.0

**public int getAcademicsScale()**

Objective: Get the academic score of the school.

**Testing** for this method is done int setAcademicsScale(), as that method is reliant on getAcademicsScale() to succeed.

**public void setAcademicsScale(int academicsScale)**

Objective: Set the academic score of the school.

**Basic Test** of setting and getting academic scale at a University.

Input: 1

Output: A University with a set Academic Scale 1

**public int getSocialScale()**

Objective: Get the social score of the school.

**Testing** for this method is done int setSocialScale(), as that method is reliant on getSocialScale() to succeed.

**public void setSocialScale(int socialScale)**

Objective: Get the quality of life score of the school.

**Basic Test** of setting and getting the social scale of a university.

Input: 1

Output: A university with a set Social Scale 1

**public int getQualityOfLifeScale()**

Objective: Get the social score of the school.

**Testing** for this method is done int setQualityOfLifeScale(), as that method is reliant on getQualityOfLifeScale() to succeed.

**public void setQualityOfLifeScale(int qualityOfLifeScale)**

Objective: Get the quality of life score score of the school.

**Basic Test** of setting and getting quality of life scale at a university.

Input: 1

Output: A University with a set Quality of Life scale 1

**public ArrayList<String> getEmphases()**

Objective: Get the emphases associated with this school.

**Testing** for this method is done int setEmphases(), as that method is reliant on getEmphases() to succeed.

**public void setEmphases(ArrayList<String> emphases)**

Objective: Set the emphases associated with this school.

**Basic Test** of setting and getting emphases of a University

Input: A ArrayList of Emphases

Output: A university with set emphases

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**UserController –** Class UserController serves the purpose of performing the necessary functionalities for Users on CMC

**public static void addUserSchool(int id, String school)**

Objective: Adds a school to an individual User's Array of saved schools given the User's ID and the name of the University.

**Basic Test** which adds a school to a User's list of saved Schools then confirms the school is added.

Input: Auburn university, ID - 2

Output: Boolean - true, if user is found to have saved school.

**public static void removeUserSchool(int id, String school)**

Objective: Removes a school from an individual User's Array of saved schools whose ID is given as a parameter.

**Basic Test** that removes a school from a User's saved Schools then confirms the school is removed.

Input: Auburn university, ID - 2

Output: Boolean - true, if university was removed from user's list.

**public static ArrayList<String> getAllSavedSchools(int id)**

Objective: Takes the id of a user and returns a Array List of their saved schools.

**Basic Test** returns a list of Saved Schools of a given user and compares to the expected saved school list.

Input: A user's id

Output: an ArrayList of the User's Saved Schools

**public static void editUser(String username, String firstName, String lastName, String password)**

Objective: Allows the User to edit his/her information.

**Basic Test** which edits a user's firstname lastname and password then confirms the changes made are correct.

Input: firstname - Click, lastname - Click, password - BOOM

Output: firstname - Click, lastname - Click, password - BOOM Tests - Passed

**public static boolean login(String username, String password)**

Objective: Logs on Users in order to search schools, manage their information, and view their saved schools

**White Box Test** of login.

Case 1: Input: correct username and correct password

Output: login returns true

Case 2: Input: correct username and incorrect password

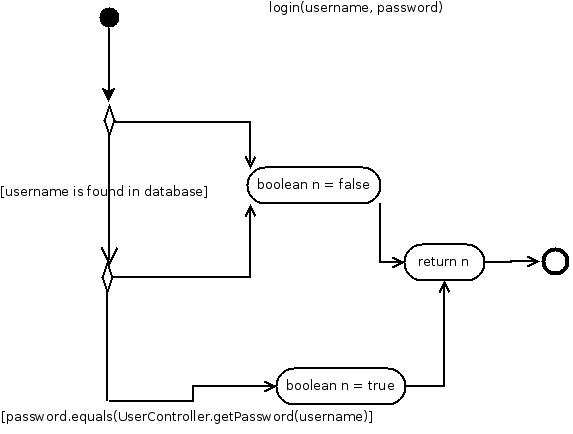
Output: login returns false

Case 3: Input: invalid username and correct password

Output: login returns false

Case 4: invalid username and invalid password

Output: login returns false



**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**User –** Will test the methods associated with the User class.

**Please reference the Person class to find methods associated with the Admin class.**

**Test for these methods can be found in the AdminTest and UserTest.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Class Descriptions – Functional Testing Class**

TestUseCase #1:Login

This functionality is shown by the method:

testLogin in UserControllerTest

TestUseCase #2: Manage Profile

This functionality is shown by the method:

testEditUser in UserControllerTest

TestUseCase #3: Search

This functionality is shown by the class:

SearchControllerTest which makes sure

each test parameter is running correctly

TestUseCase #4: Manage Saved Schools

This functionality is shown by the methods:

testAddUserSchool in UserControllerTest

testRemoveUserSchool in UserControllerTest

testGetAllSavedSchools in UserControllerTest

TestUseCase #5: Manage Universities

This functionality is shown by the methods:

testAddUniversity in AdminControllerTest

testAddUniversityEmphasis in AdminControllerTest

testEditUniversity in AdminControllerTest

testEditUniversityEmphases in AdminControllerTest

testGetAllUniversities in AdminControllerTest

TestUseCase #6: Manage Users

This functionality is shown by the methods:

testAddUser in AdminControllerTest

testDeactivateUser\_int in AdminControllerTest

testDeactivateUser\_String in AdminControllerTest

testEditUser in AdminControllerTest

testEditUserByAdmin in AdminControllerTest

testGetSpecificUser\_Integer in AdminControllerTest

testGetSpecificUser\_String in AdminControllerTest

TestUseCase #7: View Results

This functionality is shown by the class:

SearchControllerTest which makes

the necessary Schools available to view

TestUseCase #8: View Additional School Info and Recommended Schools

This functionality is shown by the class:

FiveRecSearchControllerTest which

takes a school and finds the five most

similar ones

TestUseCase #9: Save School

This functionality is shown by the method:

testAddUserSchool in UserControllerTest

TestUseCase #10: View Additional School Info

This functionality is shown by the method:

testGetAllSavedSchools in UserControllerTest

TestUseCase #11: Remove School

This functionality is shown by the method:

testRemoveUserSchool in UserControllerTest

TestUseCase #12: Add User

This functionality is shown by the method:

testAddUser in AdminControllerTest

TestUseCase #13: Edit User

This functionality is shown by the methods:

testEditUser in AdminControllerTest

testEditUserByAdmin in AdminControllerTest

TestUseCase #14: Deactivate User

This functionality is shown by the methods:

testDeactivateUser\_int in AdminControllerTest

testDeactivateUser\_String in AdminControllerTest

TestUseCase #15: Add School

This functionality is shown by the methods:

testAddUniversity in AdminControllerTest

testAddUniversityEmphasis in AdminControllerTest

TestUseCase #16: Edit School

This functionality is shown by the methods:

testEditUniversity in AdminControllerTest

testEditUniversityEmphases in AdminControllerTest

TestUseCase #17: Edit User by User

This functionality is shown by the methods:

testEditUser in UserControllerTest

**Meeting Minutes**

March 28th

·     Discussed requirements for Phase III.

o     Focused primarily on Testing

·     Decided to writing test classes for the respective classes we had previously worked on individually in Phase II.

·     Discussed what type of testing methods we would need to use.

·     Agreed to focus solely on the testing classes until the next time we would meet.

·     Reached a consensus to meet on April 1st (Was later changed to April 3rd).

April 3rd

     Shared progress on our individual test classes.

     Went over Phase III requirements.

     Clarified what we needed to have done.

     Worked together on some of the more difficult method tests.

     Divided up the required JSP pages to complete by following meeting (April 11th).

     Went over Phase II comments.

     Agreed on next day to meet as the Monday April 9th.

April 9th

     Went over progress we had made individually over Easter Break.

     Discussed Phase III requirements.

     Agreed to continue working on and improving/commenting created test classes.

     Decided to meet to put together final project Wednesday April 11th.

April 11th

     Discussed whether or not to make adjustments on JSP pages.

     Figure6SchoolSearchMenu.jsp layout debated.

o     Decided to keep it the way it was, since only a static website was needed.

     Discovered Wiley had been checking-out/diffing/updating/committing to the incorrect project.

     Added necessary comments to test classes.

     Divided final work (meeting minutes, work breakdown, final compiling).

**Task Decomposition**

Andrew Zurn:

* Final Report Compilation
* Method Descriptions
* CMC Static Web Pages:
  + figure3ViewEditUser.jsp
  + figure8ViewMatchingSchoolWithRecommendations.jsp
  + figure11AddUniversity.jsp
  + figure12ViewEditUniversity.jsp
  + figure14AddNewUser.jsp
* JUnit Test Classes:
  + AdminControllerTest
  + AdminTest
  + SearchControllerTest

Preston Hardy:

* Task Decomposition
* CMC Static Web Pages:
  + figure2UserMenuPage.jsp
  + figure4ManageSavedSchoolsPage.jsp
  + figure9AdminMenuPage.jsp
  + figure10ManageUniversitiesOption.jsp
  + figure13ManageUsersOption.jsp
* JUnit Test Classes:
  + DatabaseControllerTest
  + FiveRecSchoolsControllerTest
  + PersonTest
  + ProjectTestSuite
  + UseCasesTest

Wiley Radomski:

* CMC Static Web Pages:
  + figure1Login.jsp
  + figure5ViewSchool.jsp
  + figure6SchoolSeachMenu.jsp
  + figure7SearchResults.jsp
  + figure15ViewEditUser.jsp
* JUnit Test Classes:
  + UniversityTest
  + UserControllerTest
  + UserTest
* Login Flowchart
* Meeting Minutes