Test Cases Document

Table of Contents

Document Version Control and Revision History	2
I. Test Plan for Creating an Account (White Box Testing)	4
a. Purpose of the method	4
b. Code Analysis for the method	4
c. Flow Chart	8
d. Type of Coverage	9
e. List of Test Cases	9
f. Detailed Test Case	11
g. Test Case Demonstration	12
II. Test Plan for Creating a Course (Black Box Testing)	13
a. Purpose of the method	13
b. Description of the Equivalent classes	13
c. Choosing representative input values for the test method	14
d. Constructing a set of Test Cases using Representative Values	15
e. Detailed Test Case	18

I. Test Plan for Creating an Account (White Box Testing)

a) Purpose of the method

The purpose of the method is to take in user's inputs from the GUI text field boxes in the account creation page and convert these inputs into appropriate format for database insertion. In order to create an account successfully, there are a few pre-conditions which must be satisfied by the user. When an account is being created, the user is required to fill in all the necessary information which includes the first name, last name, employee ID, user name, a temporary password, and at least one user role for the account owner. The method will automatically reject the creation request if any of the required information is missing, or if any incorrect type of information is entered at each of the text field boxes. Since account management is solely performed by System Administrator, this method is only available and used once a user logged in as a System Administrator. And lastly, this method will only work if the Streamlined Grading System software is connected to the system database.

b) Code Analysis for the method

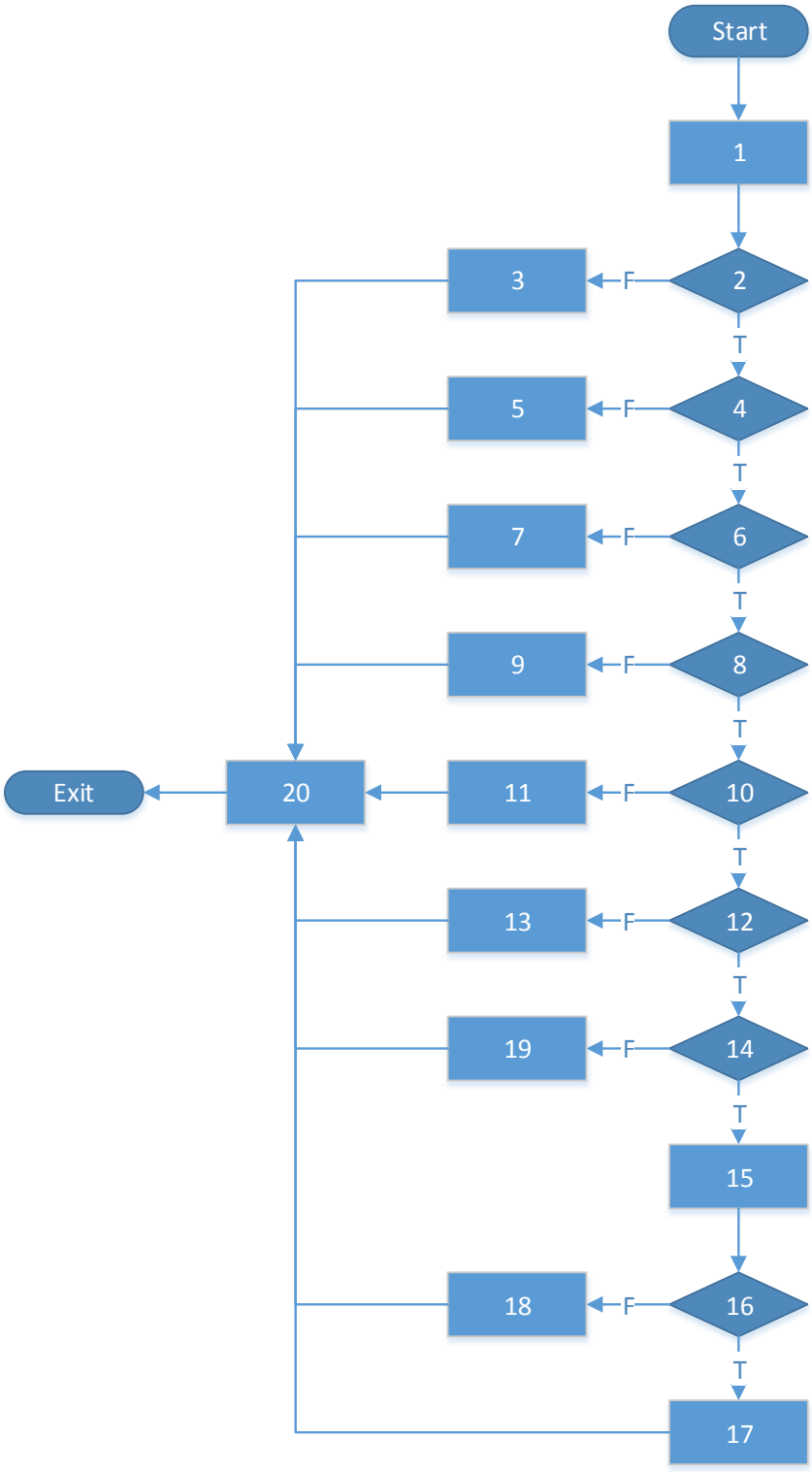
```
bool StreamlinedGradingSystem::createNewAccount()
{

/* BLOCK 1
 * Initializes values which will be used in the method.
 * */
  bool rv = true;
  SysAdmin systemAdmin;
  bool result;

/* BLOCK 2
 * Checks if all the required information for creating account is entered
 * */
  if (  (ui->FirstNameCreate->text().size() ==0) ||
        (ui->LastNameCreate->text().size() == 0) ||
        (ui->EmployeeIDCreate->text().size() == 0) ||
        (ui->UserIDCreate->text().size() == 0) ||
        (ui->UserIDCreate->text().size() == 0) ||
        (ui->PasswordCreate->text().size() == 0)
     )
  {

/* BLOCK 3
 * Sends out error feedback to user if required information is missing
 * */
    printLogMessage(ERROR_MESSAGE_TEXT, "Required information is empty");
    return false;
  }
```

```
/* BLOCK 4
 * Checks if user's input type is matching the required input type. (e.g. employeeID must be an INT)
 * */
 QRegExp re("\\d*");
 if ( !re.exactMatch(ui->EmployeeIDCreate->text()))
 {

/* BLOCK 5
 * Sends out error feedback to user for incorrect input type (Employee ID)
 * */
   printLogMessage(ERROR_MESSAGE_TEXT, "Employee ID must be number");
   return false;
 }

/* BLOCK 6
 * Checks if the length of User ID is within 8 characters
 * */
 if (ui->UserIDCreate->text().size() > 8)
 {

/* BLOCK 7
 * Sends out error feedback to user for incorrect input length (User ID)
 * */
   printLogMessage(ERROR_MESSAGE_TEXT, "User ID must be within 8 letters");
   return false;
 }

/* BLOCK 8
 * Checks if the length of Employee ID is 12 digits
 * */
 if (ui->EmployeeIDCreate->text().size() != 12)
 {

/* BLOCK 9
 * Sends out error feedback to user for incorrect input length (Employee ID)
 * */
   printLogMessage(ERROR_MESSAGE_TEXT, "Employee ID must be 12 digits");
   return false;
 }

/* BLOCK 10
 * Checks if at least one role is selected for the account in creation
 * */
 if (!((ui->Instructor_checkBox->isChecked() == true) ||
       (ui->TA_checkBox->isChecked() == true) ||
       (ui->SA_checkBox->isChecked() == true) ||
       (ui->AdministrativeAdmin_checkBox->isChecked() == true) ||
       (ui->Admin_checkBox->isChecked() == true)
     ))
 {
```

```
/* BLOCK 11
 * Sends out error feedback to user for missing input (Role Selection)
 * */
   printLogMessage(ERROR_MESSAGE_TEXT, "User must have at least one role");
   return false;
  }

/* BLOCK 12
 * Checks if user inputted Employee ID is already existed in the system
 * */
  if (true == systemAdmin.employeeIDExists(ui->EmployeeIDCreate->text()))
  {

/* BLOCK 13
 * Sends out error feedback to user for entering existed Employee ID
 * */
   printLogMessage(ERROR_MESSAGE_TEXT, "EmployeeID already exists");
   return false;
  }

/* BLOCK 14
 * Checks if user inputted User Name has already existed in the system
 * */
  if ( false == systemAdmin.userNameExists(ui->UserIDCreate->text()))
  {

/* BLOCK 15
 * Insert user inputted information into the system database
 * */
   result = systemAdmin.createNewAccount(
            ui->FirstNameCreate->text(),
            ui->MiddleNameCreate->text(),
            ui->LastNameCreate->text(),
            ui->UserIDCreate->text(),
            ui->EmployeeIDCreate->text(),
            ui->PasswordCreate->text(),
            ui->TA_checkBox->isChecked(),
            ui->Instructor_checkBox->isChecked(),
            ui->Admin_checkBox->isChecked(),
            ui->AdministrativeAdmin_checkBox->isChecked(),
            ui->SA_checkBox->isChecked());

/* BLOCK 16
 * Checks if information has successfully entered into the database
 * */
   if (true == result)
   {

/* BLOCK 17
 * Set "Account is created" to true
 * */
     rv = true;
   }
   else
   {
```

```
/* BLOCK 18
 * Set "Account is created" to false
 * */
      printLogMessage(ERROR_MESSAGE_TEXT, "Failed to create account. Database error");
      rv = false;
    }
  }

/* BLOCK 19
 * Sends out error feedback to user for entering existed User ID
 * */
  else
  {
    printLogMessage(ERROR_MESSAGE_TEXT, "Username already exists");
    rv = false;
  }

/* BLOCK 20
 * Return "Account is created" value
 * */
  return rv;
}
```

c) Flow Chart


Start
1
2
F
3
T
4
F
5
T
6
F
7
T
8
F
9
T
10
F
11
T
12
F
13
T
14
F
19
T
15
16
F
18
T
17
20
Exit

d) Type of Coverage

Since the method being tested consists of a series of conditional statements to check for the user's valid inputs, in order to get a full coverage of all the cases, the **Branch Coverage** testing method will be used to test the method as it will cause each source statement in the method to be executed at least once.

e) List of Test Cases

Test Case ID:	Coverage:	Description:
1B	Start-1-2-3-20-Exit	Test for the case when any of the required text fields for account creation is left empty.
2B	Start-1-2-4-5-20-Exit	Test for the case when the user inputted Employee ID is not of an integer type.
3B	Start-1-2-4-6-7-20-Exit	Test for the case when the user inputted User ID exceeded 8 characters.
4B	Start-1-2-4-6-8-9-20-Exit	Test for the case when the user inputted Employee ID is not 12 digits.
5B	Start-1-2-4-6-8-10-11-20-Exit	Test for the case when none of the user roles is selected by the user.
6B	Start-1-2-4-6-8-10-12-13-20-Exit	Test for the case when the user inputted Employee ID is already existed in the system.

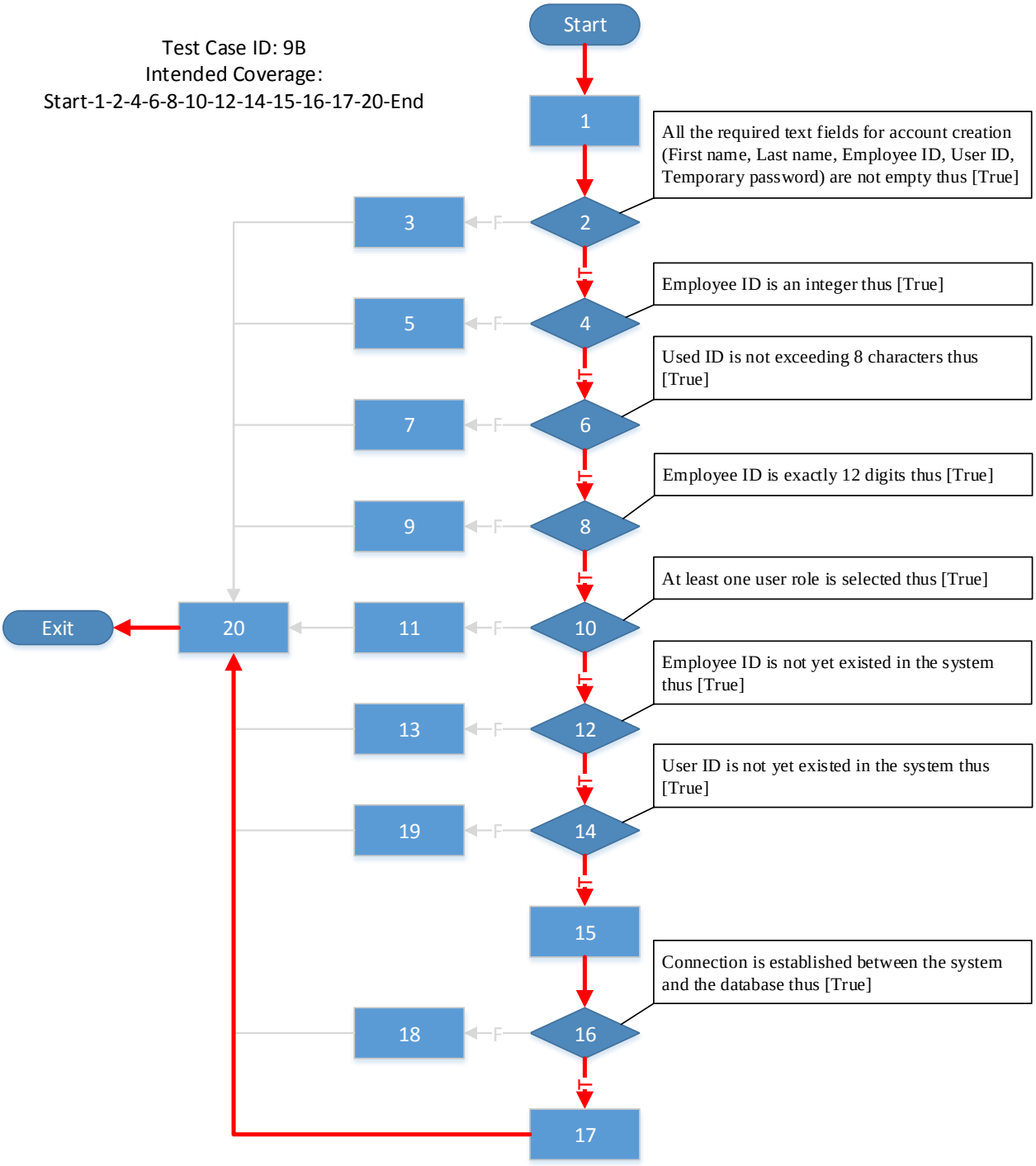
7B	Start-1-2-4-6-8-10-12-14-19-20-Exit	Test for the case when the user inputted User ID is already existed in the system
8B	Start-1-2-4-6-8-10-12-14-15-16-18-20-Exit	Test for the case when the SQL server fails to insert the user provided information into the system database.
9B	Start-1-2-4-6-8-10-12-14-15-16-17-20-Exit	Test for the case when all user inputs are valid and the inputs are successfully inserted into the system database. (Passes all conditions)

f) Detailed Test Case

Test Case ID:	9B
Test Purpose:	Test for the case when all user inputs are valid and the inputs are successfully inserted into the system database. (Passes all conditions)
Requirement Number:	2.1.1
Inputs:	• All the required text fields for account creation (First name, Last name, Employee ID, User ID, Temporary password) are not empty • Employee ID is an integer • User ID is within 8 characters • Employee ID is exactly 12 digits • At least one user role is selected • Employee ID is not already existed in the system • User ID is not already existed in the system • Connection between the system and the database is established
Testing Procedure:	1. Log-in as a System Administrator and select 'Create Account' on the main menu. 2. Insert all the required valid inputs to their corresponding fields in the Account Creation GUI 3. Click on 'Create Account' button in the Account Creation GUI [Call createNewAccount()]
Evaluation:	No step required
Expected Behaviors and results	Inputs will pass all conditions in the method and will be inserted to the database table. A feedback message "Account is created" will be shown on the Account Creation GUI to indicate the success of the operation.
Actual Behaviors and results	"Account is created" message is shown on the GUI and all test inputs are inserted into the database Account table.

g) Test Case Demonstration

Test Case ID: 9B
Intended Coverage:
Start-1-2-4-6-8-10-12-14-15-16-17-20-End



II. Test Plan for Creating a Course (Black Box Testing)

a) Purpose of the method

bool Course::isError(QString courseName, QString courseNumber, QDate startDate, QDate endDate);

The purpose of the method is to compare each of the 4 parameters to a set of requirements to check for correctness or errors in a process of course creation. It returns false to Course object indicating that there has been an error in the process, and it returns true if none of the parameters have violated the requirement. Pre-condition of the method is that the Course object has been instantiated by the SGS GUI course page. Post-condition of the method is that GUI has received a message from isError method indicating whether there is a problem with the user's input values.

b) Description of the Equivalent classes

Equivalence classes for courseName, courseNumber, startDate, and endDate:

Note: Values for courseName and courseNumber represents the number of characters. (e.g. courseName = 4 means courseName has 4 characters)

- Course Name: courseName [1 to 50 Length of CHAR]

Invalid	Valid	Invalid
< 1	1 - 50	> 50

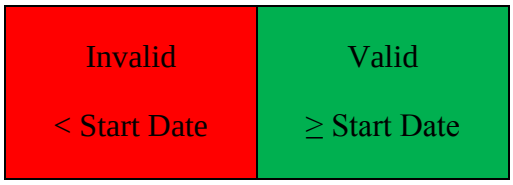
- Course Number: courseNumber [1 to 50 Length of CHAR]

Invalid	Valid	Invalid
< 1	1 - 50	> 50

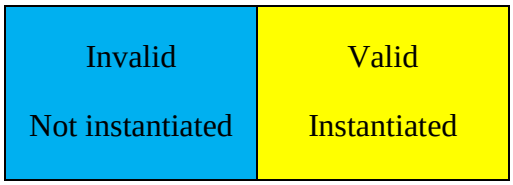
- Start Date: startDate

Invalid	Valid
< Current date of system	≥ Current date of system

- End Date: endDate



- Course Object courseObject



c) Choosing representative input values for the test method

- We choose one value from invalid range(s) and one from valid range(s). Also, we choose the boundary values. The candidate values are as follow:

courseName [Length of CHAR]	0, 1, 5, 50, 55
courseNumber [Length of CHAR]	0, 1, 10, 50, 51

- Since endDate test values depend on startDate test values; Thus, we have to choose the values for endDate based on startDate chosen values.
- Also, we are using currentDate(), addMonths(), and addDays() functions in QDate class to access the current date of the system and manipulate it during the test.
- Before we start with the values for testing, we have:

Qdate date = Qdate::currentDate();

startDate	[Date = addMonths(-2)], date, [date = addDays(5)]

- Finally, we store the chosen values for startDate before choosing the values for endDate (stored as “sDate”).

endDate	[sDate = addDays(-3)], sDate, [sDate = addMonths(3)]

d) Constructing a set of Test Cases using Representative Values

Test Case ID	Brief Description	Input Values
1	All variables in valid range	courseName = 5 courseNumber = 10 startDate = [date = addDays(5)] endDate = [sDate = addMonths(3)] courseObject instantiated
2	All variables in valid range	courseName = 5 courseNumber = 10 startDate = [date = addDays(5)] endDate = [sDate = addMonths(3)] courseObject **NOT** instantiated
3	cousreNumber, startDate, endDate valid, courseName invalid	courseName = 0 courseNumber = 10 startDate = [date = addDays(5)] endDate = [sDate = addMonths(3)] courseObject instantiated
4	courseNumber, startDate, endDate valid, courseName invalid (different class)	courseName = 55 courseNumber = 10 startDate = [date = addDays(5)] endDate = [sDate = addMonths(3)] courseObject instantiated
5	courseName, startDate, endDate valid, courseNumber invalid	courseName = 5 courseNumber = 0 startDate = [date = addDays(5)] endDate = [sDate = addMonths(3)]

		courseObject instantiated

6	courseName, startDate, endDate valid, courseNumber invalid (different class)	courseName = 5 courseNumber = 51 startDate = [date = addDays(5)] endDate = [sDate = addMonths(3)] courseObject instantiated
7	courseName, courseNumber, endDate valid, startDate invalid	courseName = 5 courseNumber = 10 startDate = [date = addMonths(-2)] endDate = [sDate = addMonths(3)] courseObject instantiated
8	courseName, courseNumber, startDate valid, endDate invalid	courseName = 5 courseNumber = 10 startDate = [date = addDays(5)] endDate = [sDate = addDays(-3)] courseObject instantiated
9	courseNum, startDate, endDate valid, courseName with boundary value	courseName = 1 courseNumber = 10 startDate = [date = addDays(5)] endDate = [sDate = addMonths(3)] courseObject instantiated
10	courseNum, startDate, endDate valid, courseName with boundary value (another equivalence class)	courseName = 50 courseNumber = 10 startDate = [date = addDays(5)] endDate = [sDate = addMonths(3)] courseObject instantiated
11	courseName, startDate, endDate valid, courseNumber valid with boundary value	courseName = 5 courseNumber = 1 startDate = [date = addDays(5)] endDate = [sDate = addMonths(3)] courseObject instantiated

12	courseName, startDate, endDate valid, courseNumber valid with boundary value (another equivalence class)	courseName = 5 courseNumber = 50 startDate = [date = addDays(5)] endDate = [sDate = addMonths(3)] courseObject instantiated
13	courseName, courseNumber, endDate valid, startDate with boundary value	courseName = 5 courseNumber = 10 startDate = date endDate = [sDate = addMonths(3)] courseObject instantiated
14	courseName, courseNumber, startDate valid, endDate valid with boundary value	courseName = 5 courseNumber = 10 startDate = [date = addDays(5)] endDate = sDate, courseObject instantiated

By continuing this approach with different combinations of equivalence classes we can have a more comprehensive subset to test our method.

e) Detailed Test case

Test Case ID:	1
Test Purpose:	Test for the case when all inputs are valid. (Passes all conditions)
Requirement Number:	3.1.1.1
Inputs:	• courseName = “MONEY” (5 CHAR) • courseNumber = “1234567895” (10 CHAR) • startDate = [date = addDays(5)] • endDate = [date = addMonths(3)] • Course object has been instantiated
Testing Procedure:	1. Create an object of the Course class type. 2. Call and Pass the four input variables to isError() method.
Evaluation:	No step required
Expected Behaviors and results	Return true indicating that the all inputs are valid.
Actual Behaviors and results	As expected.