**Object-Oriented Analysis and Design (Spring, 2012)**

Student Information System (SIS)

**Homework #7**

**Danh sách thành viên**

**Vo Phuong Binh 100599008**

# Menu

# Hệ thống

|  |  |  |
| --- | --- | --- |
| **Revision** | **Description** | **Date** |
| **Iteration I** | | |
| Homework #1 | Problem statement  Software Environments | 19/3/2024 |
| Homework #2 | Tóm tắt các tính năng của hệ thống  Sơ đồ Use case  Ràng buộc phi chức năng | Mar 14, 2012 |
| Homework #3 | *Summary of System Features refinement*  *Non-functional Requirements and Constraints refinement*  Domain Class Diagram Add Associations  Add Attributes | Mar 28, 2012 |
| Homework #4 | *Cover pager refinement*  *Use Cases (UC3) refinement (by adding the* ***Qualifying*** *attribute of Extend Information)* Logical Architecture  System Sequence Diagram Operation Contract Operation Sequence Diagram  Design Class Model | April 16, 2012 |
| Homework #5 | *Logical Architecture refinement*  *System Sequence Diagram refinement* | April 26, 2012 |

|  |  |  |
| --- | --- | --- |
|  | *Operation Sequence Diagram refinement Design Class Model refinement* Implementation Class Model Programming  Unit Testing |  |
| **Iteration II** | | |
| Homework #6 | *Use Cases refinement. Classes Identified refinement. Bad Classes refinement.*  *Good Classes refinement.*  *Domain Class Diagram refinement* | May 22, 2012 |
| Homework#7 | *System Sequence Diagram refinement Operation Sequence Diagram refinement Design Class Model refinement Implementation Class Model refinement Programming refinement*  *Unit Testing refinement* | June 12, 2012 |

## Vấn đề liên quan đến hệ thống

Hệ thống nhà trọ Đà Lạt là một trang web cho phép

Student Information System (SIS) is a GUI-based application used in the Lab of the university. SIS is used for both advisor and students. The students can manage their personal information, courses, papers, and qualifying. The advisor can add students to the SIS and see their personal information. Therefore, the advisor can control the study progress of his/her students.

The advisor can:

* Đăng nhập và đăng xuất khỏi hệ thống
* Cập nhật

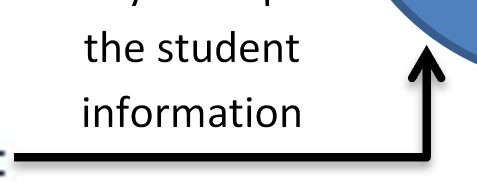
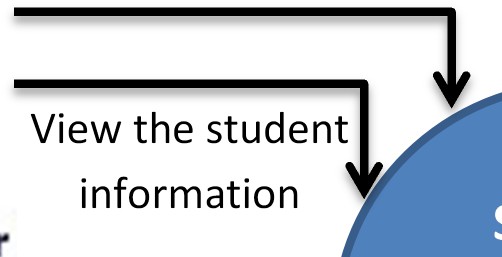
Đối tượng sinh viên có thể:

* Đăng nhập, đăng ký trong hệ thống

## System Context Diagram

View the student

information



Add, edit, and delete users to/in the SIS

Student

Information System (SIS)

Save data

**Database**

Modify and update

Load data

***Figure 1.1 – System Context Diagram***

the student

information

## Tóm tắt tính năng hệ thống

|  |  |
| --- | --- |
| **Feature ID** | **Feature Description** |
| **FEA-01** | **Đăng nhập và đăng ký** |
| **FEA-02** | **Đối tượng đến thuê: sinh viên có thể tìm kiếm thông tin, liên hệ với chủ trọ** |
| **FEA-03** | **User can be checked his/her role (as advisor or student) on**  **log in the system.** |
| **FEA-04** | **Advisor can find and choose a student to view the student**  **information and the study progress.** |
| **FEA-05** | **Student can modify, update one more the student**  **information such as courses, papers, qualifying, and so on.** |

## Sơ đồ Use Case

**A diagram of a network

Description automatically generated**

***Figure 1.2 - Use Case Diagram***

## Danh sách các Use Cases

### SinhVien

|  |  |
| --- | --- |
| **Use Case UC1: SinhVien** | |
| **Scope** | Student Information System (SIS). |
| **Level** | User-goal |
| **Primary Actor** | Advisor |

|  |  |
| --- | --- |
| **Use Case UC1: Manage Student** | |
| **Stakeholders and Interests** | * Advisor: wants to add and update the primary information of the student information easily. * Students: wants their information updated exactly by the advisor. |
| **Preconditions** | Advisor is identified and authenticated. |
| **Success Guarantee** | The student information is saved into database exactly. |
| **Main Success Scenario** | 1. Advisor starts a new transaction (log in). 2. Advisor adds, updates and deletes the primary information of the student information. 3. The student information is saved into database. 4. System presents the saved student information. (*Advisor repeats steps 2-4 until indicates done*) |
| **Extensions** | 1a Advisor enter user name and password   * System finds in database. * If found, system gets the role of this user to check whether this user is advisor. * Otherwise, inform to advisor a message.   1b. Advisor forgot password   * Advisor selects a forgotten password function of the system. * Then, advisor enters a secret information to get the old password and go back **1a**.   2a. Add a new student   * Advisor enters information such as student identifier, name, study program (Master or Ph.D), and startDate. * If advisor enters a student identifier which is existed   + System informs to advisor an error.   + System allows advisor re-enter a correct student identifier.   2b. Update a existed student   * System allows advisor find student to update by entering student identifier or name. * If system doesn’t find student then inform to |

|  |  |
| --- | --- |
| **Use Case UC1: Manage Student** | |
|  | advisor a message.   * Otherwise, system presents fields to advisor choose and edit.   2c. Delete a existed student   * System allows advisor find student to delete by entering student identifier or name. * If system doesn’t find student then inform to advisor a message. * Otherwise, system presents information to advisor confirm and delete.. |
| **Special Requirements** | - System can present a list of study program to advisor choose. |
| **Technology and Data Variations List** | None. |
| **Frequency of Occurrence** | Every semester. |
| **Miscellaneous** | - Can advisor use student card to enter student identifier? |

### ChuTro

|  |  |
| --- | --- |
| **Use Case UC2: ChuTro** | |
| **Scope** |  |
| **Level** | User-goal |
| **Primary Actor** | Advisor |

|  |  |
| --- | --- |
| **Use Case UC2: View Information** | |
| **Stakeholders and Interests** | * Advisor: wants to see student information clearly and generally. * Students: wants their information presented exactly to the advisor. |
| **Preconditions** | Advisor is identified and authenticated. |
| **Success Guarantee** | The student information must be not changed. |
| **Main Success Scenario** | 1. Advisor starts a new transaction (log in). 2. Advisor finds students by student identifier or name. 3. Advisor chooses a student to see.   (*Advisor repeats steps 2-6 until indicates done*) |
| **Extensions** | 1a Advisor enter user name and password   * System finds in database. * If found, system gets the role of this user to check whether this user is advisor. * Otherwise, inform to advisor a message.   1b. Advisor forgot password   * Advisor selects a forgotten password function of the system. * Then, advisor enters a secret information to get the old password and go back **1a**.   2a. Advisor enters a student identifier to find   * If system doesn’t find student then inform to advisor a message. * Otherwise, system presents a list of students which have student identifier nearly like (equivalent) this student identifier. |

|  |  |
| --- | --- |
| **Use Case UC2: View Information** | |
|  | 2b. Advisor enters a student name to find   * If system doesn’t find student then inform to advisor a message. * Otherwise, system presents a list of students which have student name nearly like (equivalent) this name. |
| **Special Requirements** | - System can present a list of student identifier or name to advisor choose. |
| **Technology and Data Variations List** | None. |
| **Frequency of Occurrence** | Weekly. |
| **Miscellaneous** | - Can advisor use student card to find student information? |

### QuanTriVien

|  |  |
| --- | --- |
| **Use Case UC3: QuanTriVien** | |
| **Scope** | Student Information System (SIS). |
| **Level** | User-goal |
| **Primary Actor** | Student |
| **Stakeholders and Interests** | * Advisor: wants students modify the extend information of the student information exactly. * Students: wants to modify the extend information of the student information easily. |
| **Preconditions** | Student is identified and authenticated. |
| **Success Guarantee** | Student’s modified information is saved into database exactly. |
| **Main Success Scenario** | 1. Student starts a new transaction (log in). 2. Student selects and enters which information need to modify such as personal information, courses, papers, and qualifying. |

|  |  |
| --- | --- |
| **Use Case UC3: Manage Information** | |
|  | 3. System presents the relative information of the student information which is modified.  (*Student repeats steps 2-3 until indicates done*) |
| **Extensions** | 1a Student enter user name and password   * System finds in database. * If found, system gets the role of this user to check whether this user is student. * Otherwise, inform to student a message. |
| **Special Requirements** | None |
| **Technology and Data Variations List** | None. |
| **Frequency of Occurrence** | Weekly. |
| **Miscellaneous** | - Can student upload papers (e.g. PDF file) to the system? |

## Non-functional Requirements and Constraints

|  |  |  |
| --- | --- | --- |
| **NFR ID** | **Category** | **Description** |
| **NFA-01** | **Performance** | The student wants to complete the information  modifying easily and quickly. |
| **NFA-02** | **Usability** | The advisor and student will be worked with a clear display of the SIS. Therefore:   * Text should be easily visible (as large as possible). * Avoid colors associated with common forms   of color blindness. |
| **NFA-03** | **Reliability** | Advisor can see exactly information modified by  students. |
| **NFA-04** | **Adaptability** | SIS has a unique rule for all students: only modified  or update oneself; do not allow delete users. |

## Thuật ngữ

|  |  |
| --- | --- |
| **Term** | **Definition or Description** |
| **SinhVien** | Đối tượng sinh viên |
| **ChuTro** | Những người cho thuê trọ |
| **User information** | Includes user name, password to log in the system. |
| **View Information** | List and see the student information. |
| **Quản lý thông tin** | Modify, update the student information. |
| **Save data** | Save advisor or student information into database. |
| **Load data** | Get advisor or student information from database. |
| **Assign role** | Assign the role to users as advisor or student. |

|  |  |
| --- | --- |
| **Study progress** | Includes courses and papers which student finished. |
| **Field** | An attribute of the student information (e.g. name,  papers, and so on). |
| **Criteria** | The requirements of a class (Master or Ph.D) which a  student must finish to get the degree relatively. |

## Công nghệ và công cụ sử dụng

* Dự án được viết bằng ngôn ngữ C#
* Môi trường phát triển: Microsoft Visual Studio 2022

# Domain Class Model

* 1. **Domain Class Diagram Showing Only Concepts List conceptual classes from UC1, UC2, and UC3:**

Advisor

Student Information

Qualifying

Class

Student

Primary Information

Course

Progress (new)

Login

Extend Information

Paper

Point (new)

Password

Study Progress

Criteria (new)

**List bad conceptual classes (\*):**

* **Login:** only is an operation of advisor or student.
* **Password:** only is an attribute of advisor or student.
* **Study Progress:** only is a few attributes of student information.
* **Class:** only is an attribute of Primary Information.
* **(new) Progress:** only is a few attributes of Extend Information

Advisor

Student Information

Qualifying

Class (\*)

Student

Primary Information

Course

**Progress(\*) (new)**

Login (\*)

Extend Information

Paper

**Point(\*) (new)**

Password (\*)

Study Progress (\*)

**Criteria (new)**

**List good conceptual classes:**

Advisor

Student Information

Qualifying

**Criteria (new)**

Student

Primary Information

Course

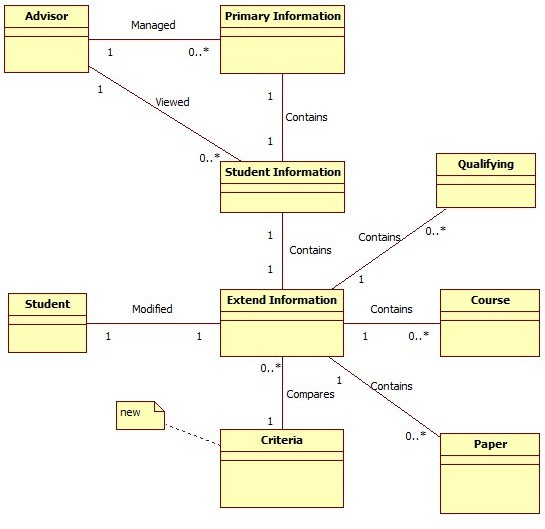
Extend Information

Paper

## Liên kết giữa các thuộc tính

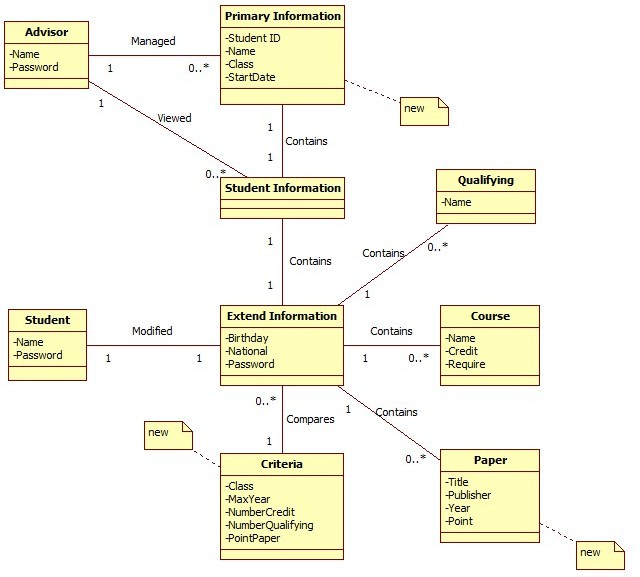
|  |
| --- |
| **Mô tả các quan hệ** |
| Một **sinh viên** có thể đến thuê tại một phòng trong một khoảng thời gian nhất định. |
| Một **chủ trọ** có thể cho nhiều sinh viên thuê tại một khu trọ  Có thể có nhiều khu trọ |
|  |
| One Student Information **Contains** only one Primary Information. |
| One Student Information **Contains** only one Extend Information. |

|  |
| --- |
| One Extend Information **Contains** zero or more than one Qualifying |
| One Extend Information **Contains** zero or more than one Course. |
| One Extend Information **Contains** zero or more than one Paper. |
| One Extend Information **Compares** to one Criteria**.** One Criteria **is Compared**  by zero or more than one Extend Information. |



***Figure 2.1 Domain Class Model with Associations***

## Add Attributes



***Figure 2.2 Domain Class Model with Associations and Attributes***

## Kiến trúc logic

**Descriptions**

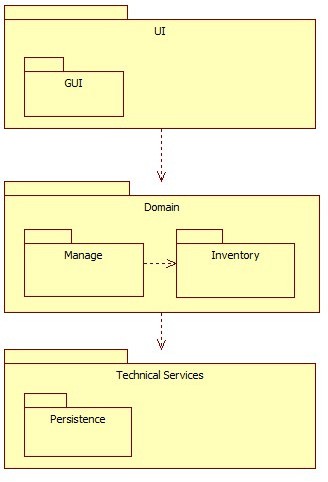
* **UI**
  + **GUI:** use Graphic User Interface (GUI) of the C# language to perform the User Interface (UI) of software.

Domain

* + **Manage:** the main package of software represents domain concepts for managing student information.
  + **Inventory:** the package for accessing to a particular set of persistent data.

Technical Services

* **Persistence:** the package that provide access to databases.

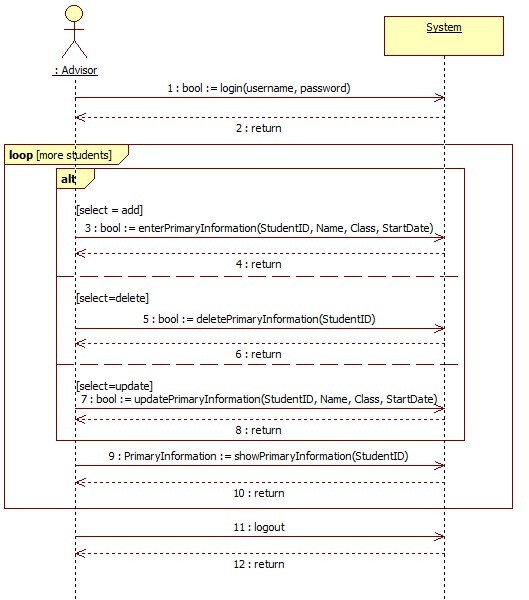


***Figure 3.1 Logical Architecture in UML package diagram***

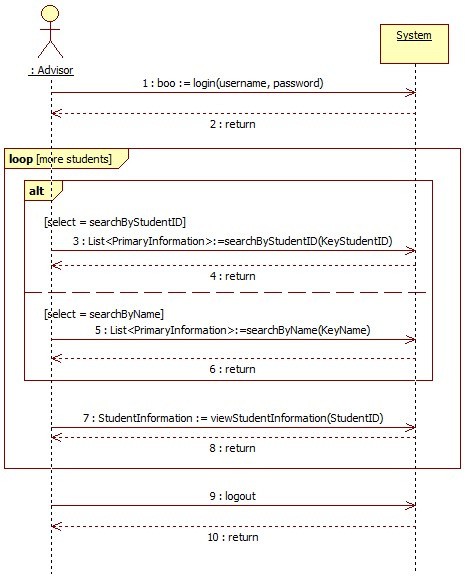
# Class Design

## Use-Case Realizations with GRASP Patterns

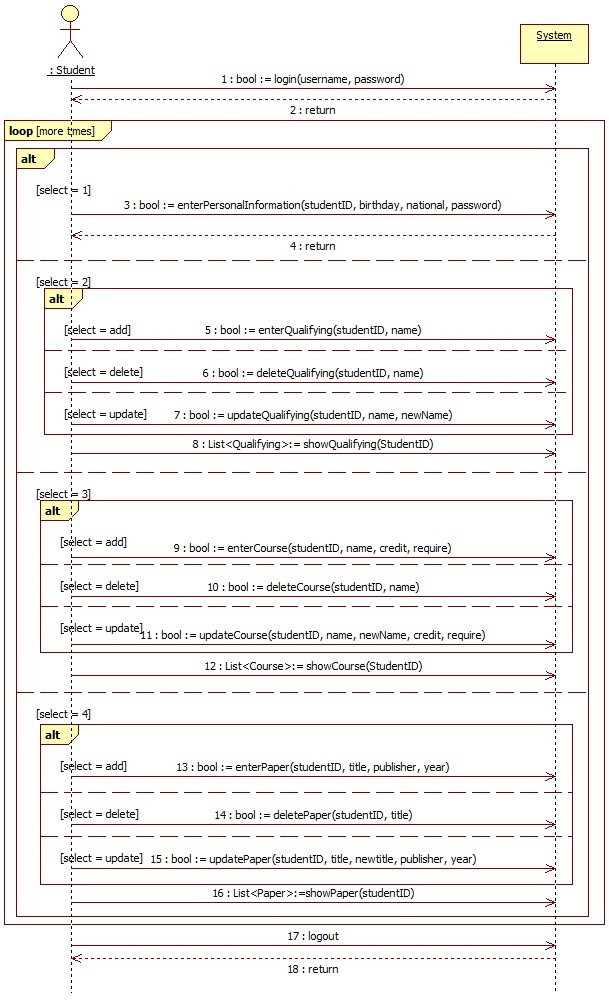
### System Sequence Diagram (new)



***Figure 4.1 Sequence Diagram of use case Manage Student***



***Figure 4.2 Sequence Diagram of use case View Information***



***Figure 4.3 Sequence Diagram of use case Manage Information***

### Operation Contract

|  |  |
| --- | --- |
| **Contract ID** | **Operation Name** |
| CO-01 | Đăng nhập |
| CO-02 | Đăng ký |
| CO-03 | deletePrimaryInformation (new) |
| CO-04 | updatePrimaryInformation (new) |
| CO-05 | showPrimaryInformation |
| CO-06 | Đăng xuất |
| CO-07 | TimTheoKhoangCach() |
| CO-08 | TimTheoGia() |
| CO-09 | TimTheoViTri() |
| CO-10 | enterPersonalInformation |
| CO-11 | enterCourse |
| CO-12 | deleteCourse (new) |
| CO-13 | updateCourse (new) |
| CO-14 | showCourse (new) |
| CO-15 | enterQualifying |
| CO-16 | deleteQualifying (new) |
| CO-17 | updateQualifying (new) |
| CO-18 | showQualifying (new) |
| CO-19 | enterPaper |
| CO-20 | ThemThongTin(new) |
| CO-21 | XoaThongTin(new) |
| CO-22 |  |

#### Đăng nhập

|  |  |
| --- | --- |
| **Operation** | login(userName: string, password: string) |
| **Cross References** | Use Case: Manage Student, View Information,  Manage Information |

|  |  |
| --- | --- |
| **Preconditions** | None |
| **Post-conditions** | * Một user (Sinh viên hoặc chủ cho thuê, quản trị). * Được liên kết với thông tin cho sinh viên was associated with Primary Information, Student Information (for Advisor) and   Extend Information (for Student). |

#### Đăng ký

|  |  |
| --- | --- |
| **Operation** | enterPrimaryInformation (studentID: string, name:  string, class: string, startDate: string) |
| **Cross References** | Use Case: Manage Student |
| **Preconditions** | The advisor instance was created (logined). |
| **Post-conditions** | * A Primary Information *pi* was created. * *pi.StudentID* became studentID. * *pi.Name* became name. * *pi.Class* became class. * A Student Information *si* was created and *si*   contains *pi*. |

#### deletePrimaryInformation

|  |  |
| --- | --- |
| **Operation** | deletePrimaryInformation (studentID: string) |
| **Cross References** | Use Case: Manage Student |
| **Preconditions** | The advisor instance was created (logined). |
| **Post-conditions** | A Primary Information *pi*, which has *pi.StudentID*  equal to studentID, is destroyed.  A Student Information *si,* which contains *pi,* is destroyed. |

|  |  |
| --- | --- |
| **Operation** | updatePrimaryInformation (studentID: string,  name: string, class: string, startDate: string) |
| **Cross References** | Use Case: Manage Student |
| **Preconditions** | The advisor instance was created (logined). |
| **Post-conditions** | A Primary Information *pi*, which has *pi.StudentID*  equal to studentID, is updated:   * *pi.StudentID* became studentID. * *pi.Name* became name. * *pi.Class* became class. * *pi.StartDate* became startDate.   A Student Information *si* was updated with *si*  contains *pi*. |

#### showPrimaryInformation

|  |  |
| --- | --- |
| **Operation** | showPrimaryInformation (studentID: string) |
| **Cross References** | Use Case: Manage Student |
| **Preconditions** | The Primary Information instances were created  and had the same studentID. |
| **Post-conditions** | None |

#### Đăng xuất

|  |  |
| --- | --- |
| **Operation** | LogOut(userName: string, password: string) |
| **Cross References** | Use case: |

|  |  |
| --- | --- |
| **Điều kiện tiên quyết** | User đang đăng nhập hệ thống |

# searchByStudentID (new)

|  |  |
| --- | --- |
| **Operation** | searchByStudentID (keyStudentID: string) |
| **Cross References** | Use Case: Manage Student |
| **Preconditions** | The Primary Information instances were created  and had the StudentID like keyStudentID. |
| **Post-conditions** | None |

# searchByName (new)

|  |  |
| --- | --- |
| **Operation** | searchByName (keyName: string) |
| **Cross References** | Use Case: Manage Student |
| **Preconditions** | The Primary Information instances were created  and had the Name like keyName. |
| **Post-conditions** | None |

# viewStudentInformation

|  |  |
| --- | --- |
| **Operation** | viewStudentInformation (studentID: string) |
| **Cross References** | Use Case: View Information |
| **Preconditions** | * The advisor instance was created (logined). * The Student Information instance was created and had the same studentID. |
| **Post-conditions** | None |

# enterPersonalInformation

|  |  |
| --- | --- |
| **Operation** | enterPersonalInformation (studentID: string,  birthday: string, national: string, password: string) |
| **Cross References** | Use Case: Manage Information |
| **Preconditions** | * The student instance was created (logined). * A Extend Information instance was created. |
| **Post-conditions** | * A Extend Information instance *ei* was created or updated. * *ei.Birthday* became birthday. * *ei.National* became national. * *ei.Password* became password. |

# enterQualifying

|  |  |
| --- | --- |
| **Operation** | enterQualifying (studentID: string, name: string) |
| **Cross References** | Use Case: Manage Information |
| **Preconditions** | * The student instance was created (logined). * A Extend Information instance was created. |
| **Post-conditions** | * A Qualifying instance *q* was created or updated with studentID. * *q.Name* became name. |

#### ThemThongTin

|  |  |
| --- | --- |
| **Operation** | deleteQualifying (studentID: string, name: string) |
| **Cross References** | Use Case: Manage Student |
| **Preconditions** | The student instance was created (logined). |
| **Post-conditions** | A Qualifying *qi*, which has *qi.StudentID* equal to studentID and *qi.Name* equal to name, is updated:  - *qi.StudentID* became studentID. |

|  |  |
| --- | --- |
|  | - *qi.Name* became newName. |

# XoaThongTin

|  |  |
| --- | --- |
| **Operation** | updateQualifying (studentID: string, name: string,  newName: string) |
| **Cross References** | Use Case: Manage Student |
| **Preconditions** | The student instance was created (logined). |
| **Post-conditions** | A Qualifying *qi*, which has *qi.StudentID* equal to studentID and *qi.Name* equal to name, is updated:   * *qi.StudentID* became studentID. * *qi.Name* became newName. |

# showQualifying (new)

|  |  |
| --- | --- |
| **Operation** | showQualifying (studentID: string) |
| **Cross References** | Use Case: Manage Student |
| **Preconditions** | The Qualifying instances were created and had the  same studentID. |
| **Post-conditions** | None |

# enterCourse

|  |  |
| --- | --- |
| **Operation** | enterCourse (studentID: string, name: string, credit:  number, require: boolean) |
| **Cross References** | Use Case: Manage Information |
| **Preconditions** | * The student instance was created (logined). * A Extend Information instance was created. |
| **Post-conditions** | - A Course instance *c* was created or updated |

|  |  |
| --- | --- |
|  | with studentID.   * *c.Name* became name. * *c.Credit* became credit. * *c.Require* became require. |

# deleteCourse (new)

|  |  |
| --- | --- |
| **Operation** | deleteCourse (studentID: string, name: string) |
| **Cross References** | Use Case: Manage Student |
| **Preconditions** | The student instance was created (logined). |
| **Post-conditions** | A Course *ci*, which has *ci.StudentID* equal to  studentID and *ci.Name* equal to name, is destroyed. |

# updateCourse (new)

|  |  |
| --- | --- |
| **Operation** | updateCourse (studentID: string, name: string,  newName:string, credit: number, require:bool) |
| **Cross References** | Use Case: Manage Student |
| **Preconditions** | The student instance was created (logined). |
| **Post-conditions** | A Course *ci*, which has *ci.StudentID* equal to studentID and *ci.Name* equal to name, is updated:   * *ci.StudentID* became studentID. * *ci.Name* became newName. * *ci.Credit* became credit. * *ci.Require* became require. |

# showCourse (new)

|  |  |
| --- | --- |
| **Operation** | showCourse (studentID: string) |

|  |  |
| --- | --- |
| **Cross References** | Use Case: Manage Student |
| **Preconditions** | The Course instances were created and had the  same studentID. |
| **Post-conditions** | None |

# enterPaper

|  |  |
| --- | --- |
| **Operation** | enterPaper (studentID: string, title: string,  publisher: string, year: string) |
| **Cross References** | Use Case: Manage Information |
| **Preconditions** | * The student instance was created (logined). * A Extend Information instance was created. |
| **Post-conditions** | * A Paper instance *p* was created or updated with studentID. * *c.Title* became title. * *c.Publisher* became publisher. * *c.Year* became year. |

# deletePaper (new)

|  |  |
| --- | --- |
| **Operation** | deletePaper (studentID: string, title: string) |
| **Cross References** | Use Case: Manage Student |
| **Preconditions** | The student instance was created (logined). |
| **Post-conditions** | A Paper *pi*, which has *pi.StudentID* equal to  studentID and *ci.Title* equal to title, is destroyed. |

# updatePaper (new)

|  |  |
| --- | --- |
| **Operation** | updatePaper (studentID: string, title: string, |

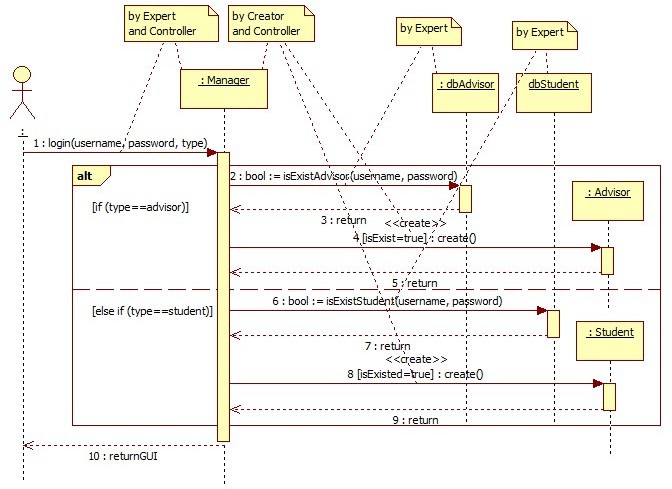
|  |  |
| --- | --- |
|  | newTitle:string, publisher: string, year: string,  point:number) |
| **Cross References** | Use Case: Manage Student |
| **Preconditions** | The student instance was created (logined). |
| **Post-conditions** | A Paper *pi*, which has *pi.StudentID* equal to studentID and *ci.Title* equal to name, is updated:   * *ci.StudentID* became studentID. * *ci.Title* became newTitle. * *Ci.Publisher* became publisher. * *ci.Year* became year. * *ci.Point* became point. |

# showPaper (new)

|  |  |
| --- | --- |
| **Operation** | showPaper (studentID: string) |
| **Cross References** | Use Case: Manage Student |
| **Preconditions** | The Paper instances were created and had the same  studentID. |
| **Post-conditions** | None |

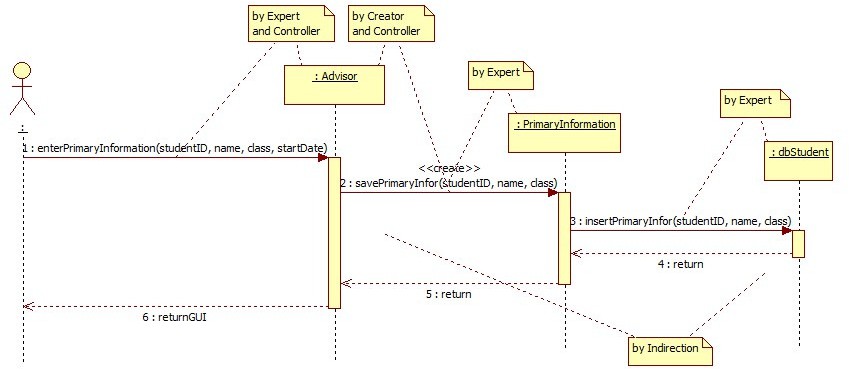
### Trình tự hoạt động

# login



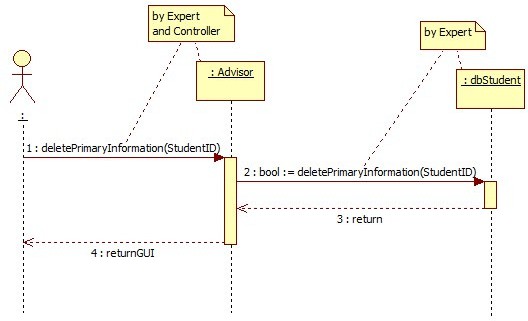
***Figure 4.4 Sequence Diagram of operator: login***

# enterPrimaryInformation



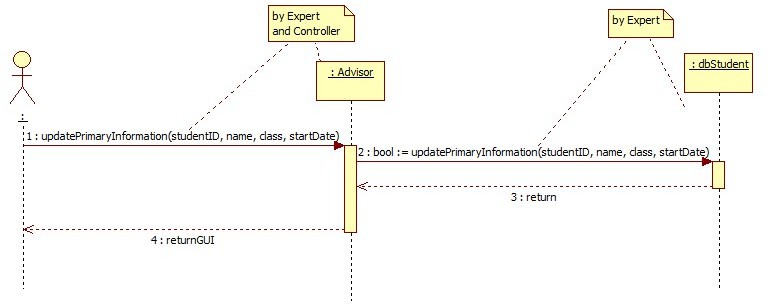
***Figure 4.5 Sequence Diagram of operator: enterPrimaryInformation***

# deletePrimaryInformation (new)



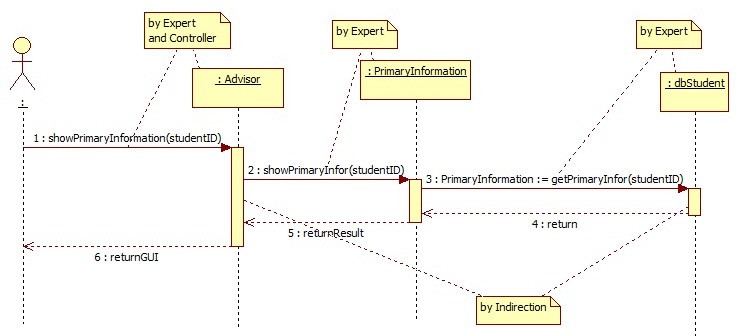
***Figure 4.6 Sequence Diagram of operator: deletePrimaryInformation***

# updatePrimaryInformation (new)



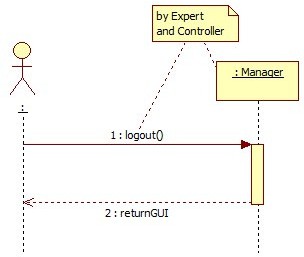
***Figure 4.7 Sequence Diagram of operator: updatePrimaryInformation***

# showPrimaryInformation



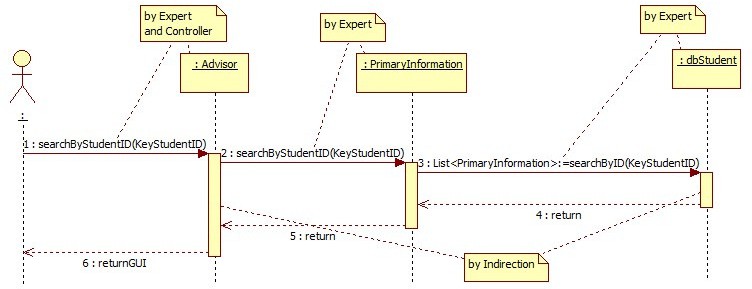
***Figure 4.8 Sequence Diagram of operator: showPrimaryInformation***

# logout



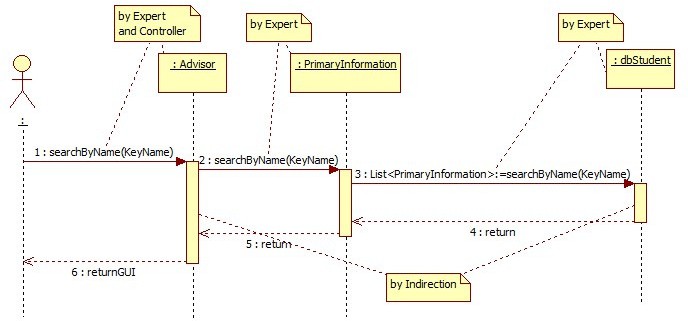
***Figure 4.9 Sequence Diagram of operator: logout***

# searchByStudentID (new)



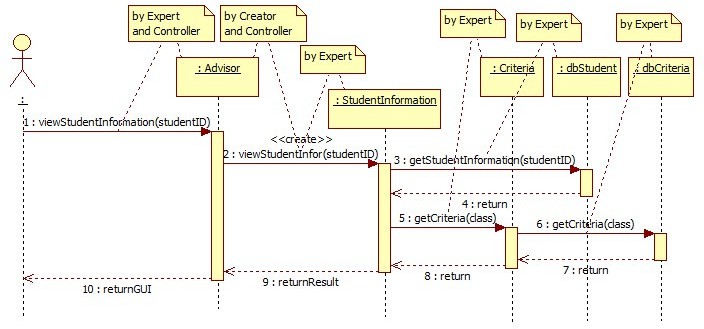
***Figure 4.10 Sequence Diagram of operator: searchByStudentID***

# searchByName (new)



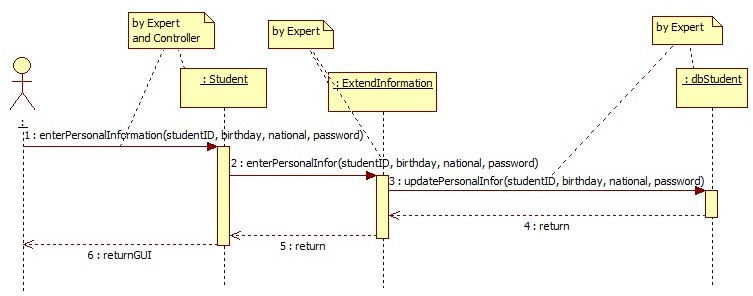
***Figure 4.11 Sequence Diagram of operator: searchByName***

# viewStudentInformation



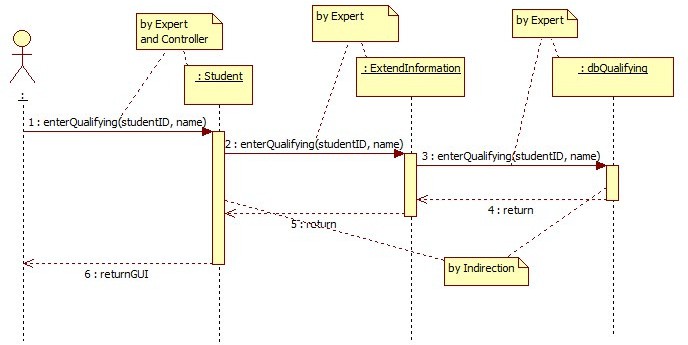
***Figure 4.12 Sequence Diagram of operator: viewStudentInformation***

# enterPersonalInformation



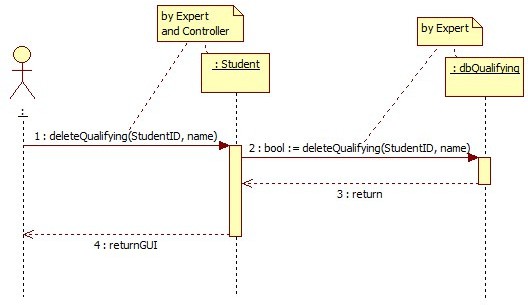
***Figure 4.13 Sequence Diagram of operator: enterPersonalInformation***

# enterQualifying



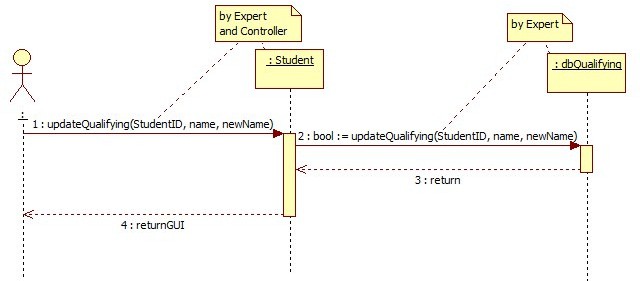
***Figure 4.14 Sequence Diagram of operator: enterQualifying***

# deleteQualifying (new)



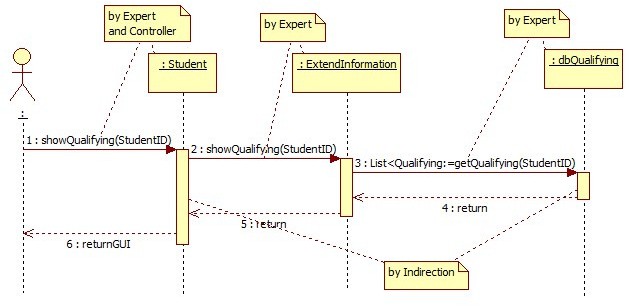
***Figure 4.15 Sequence Diagram of operator: deleteQualifying***

# updateQualifying (new)



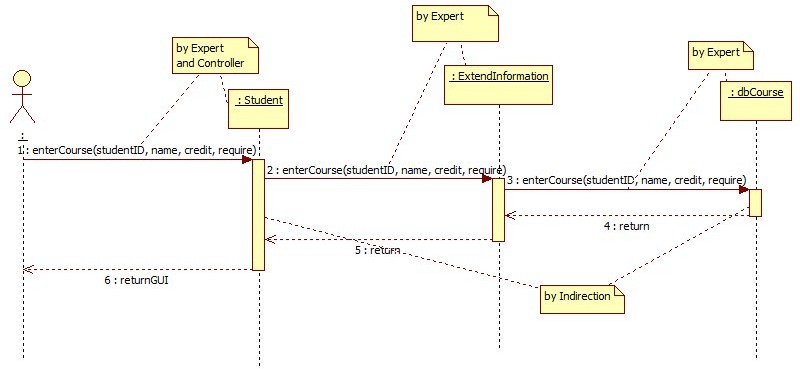
***Figure 4.16 Sequence Diagram of operator: updateQualifying***

# showQualifying (new)



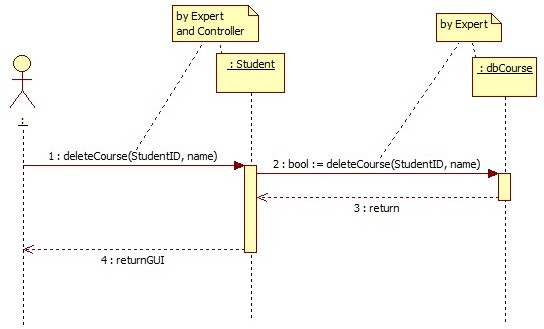
***Figure 4.17 Sequence Diagram of operator: showQualifying***

# enterCourse



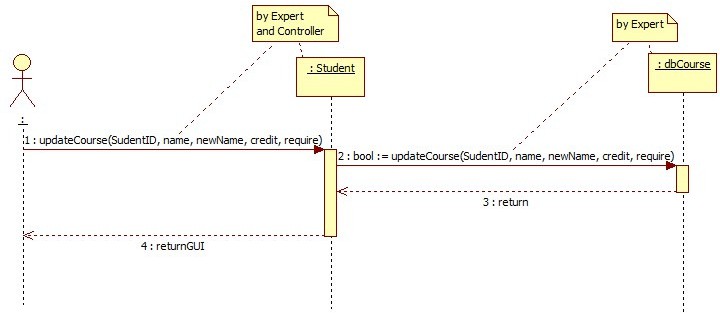
***Figure 4.18 Sequence Diagram of operator: enterCourse***

# deleteCourse (new)



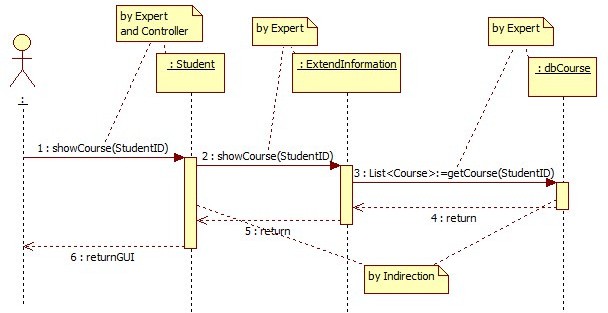
***Figure 4.19 Sequence Diagram of operator: deleteCourse***

# updateCourse (new)



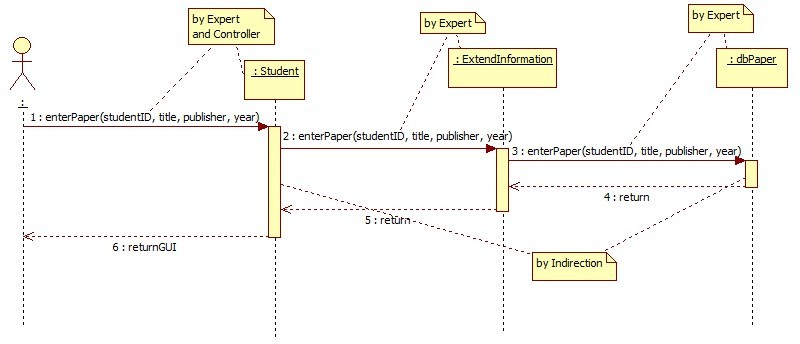
***Figure 4.20 Sequence Diagram of operator: updateCourse***

# showCourse (new)



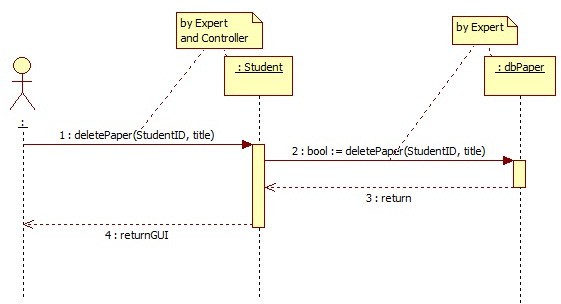
***Figure 4.21 Sequence Diagram of operator: showCourse***

# enterPaper



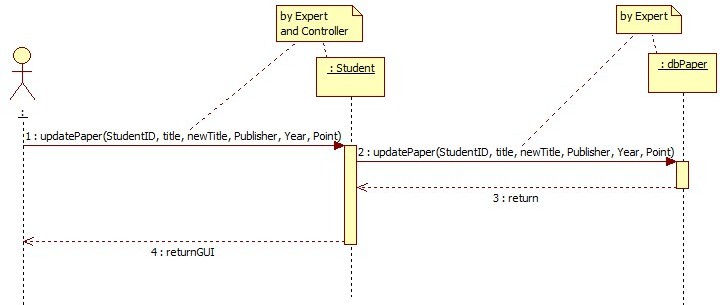
***Figure 4.22 Sequence Diagram of operator: enterPaper***

# deletePaper (new)



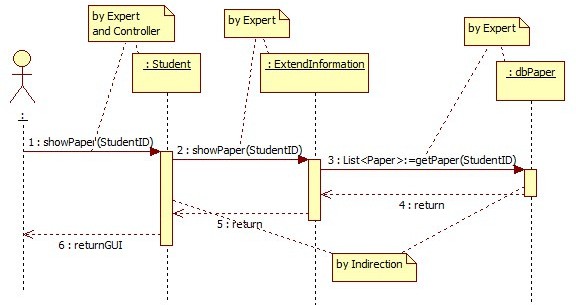
***Figure 4.23 Sequence Diagram of operator: deletePaper***

# updatePaper (new)



***Figure 4.24 Sequence Diagram of operator: updatePaper***

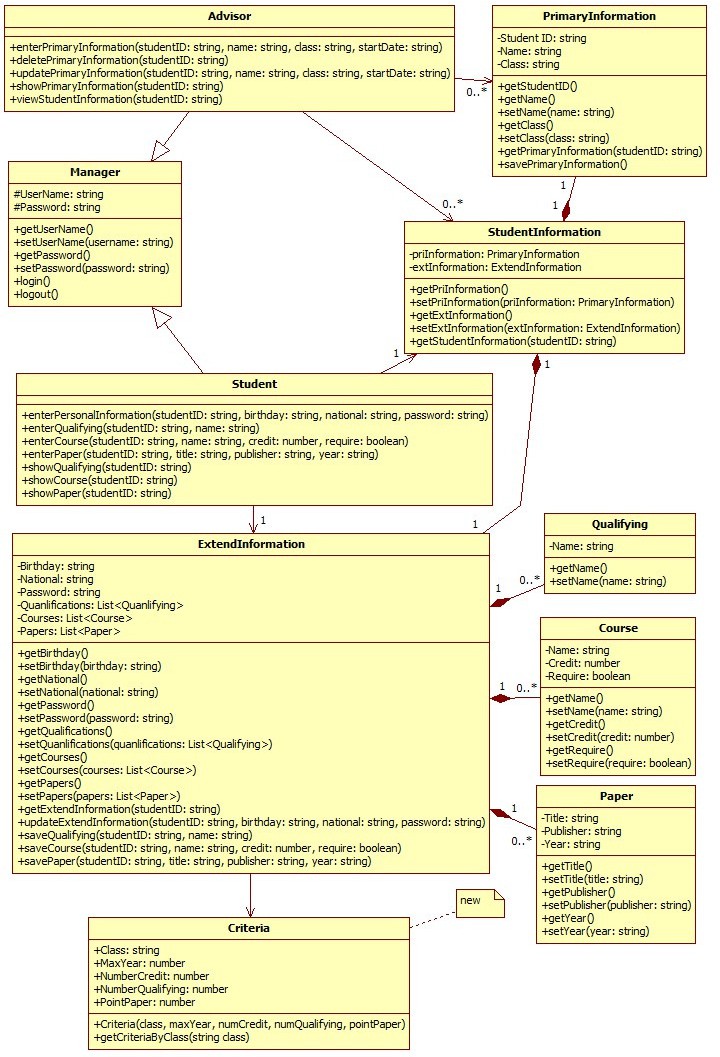
# showPaper (new)



***Figure 4.25 Sequence Diagram of operator: showPaper***

# Design Class Model

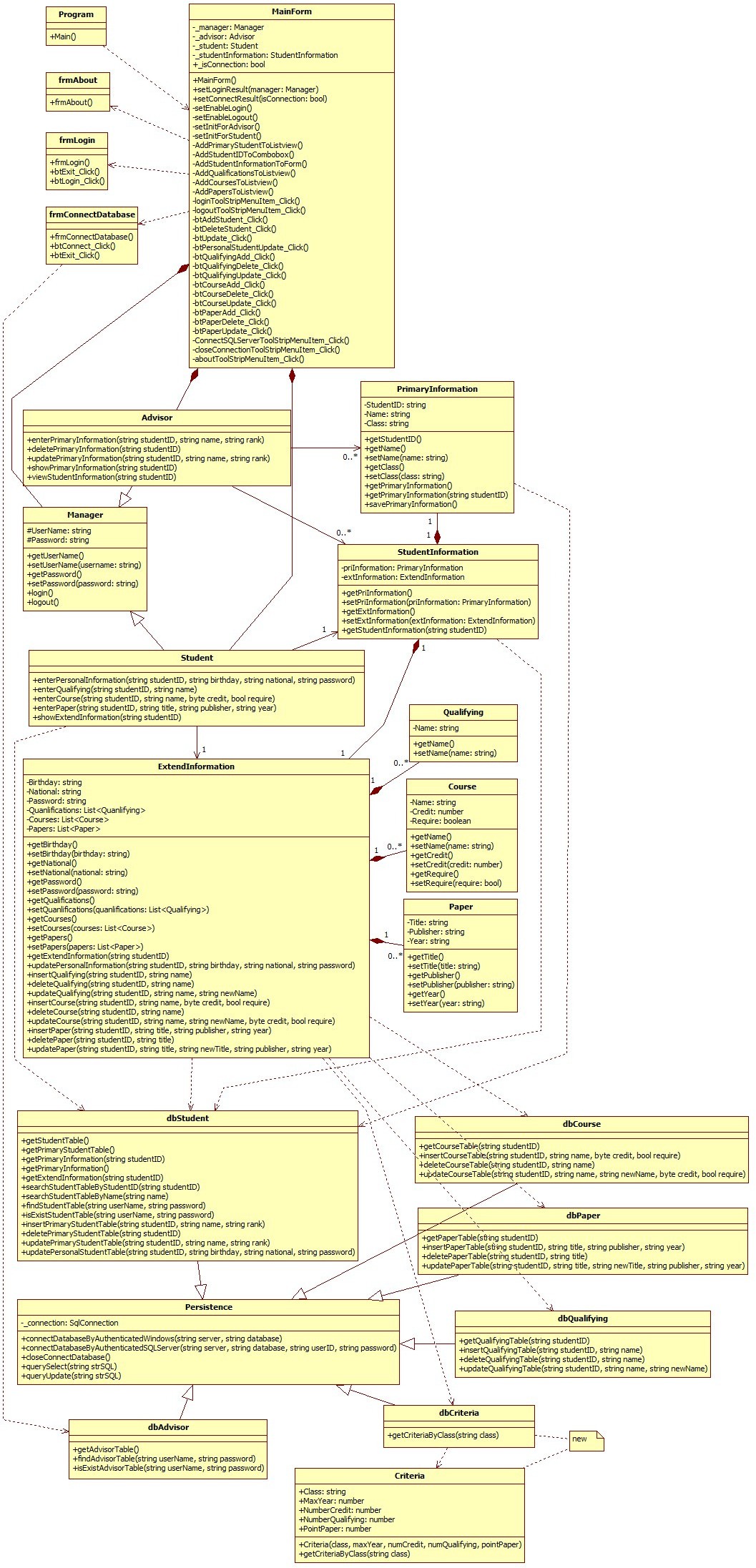
Student / Advisor is a kind of Manager (extend).



***Figure 4.11 Design Class Model (new)***

# Implementation Class Model

# Draw an implementation class diagram



***Figure 5.1 Implementation Class Diagram (new)***

* 1. **Show the difference between implementation class model and design class model (new)**

**Table 5.2.1: Comparison with design and implementation class**

|  |  |  |  |
| --- | --- | --- | --- |
| Class | Method | Design | Imp. |
| **Advisor** | enterPrimaryInformation() | **Yes** | **Yes** |
| deletePrimaryInformation() | **Yes** | **Yes** |
| updatePrimaryInformation() | **Yes** | **Yes** |
| showPrimaryInformation() | **Yes** | **Yes** |
| viewStudentInformation() | **Yes** | **Yes** |
| **Course** | Course() | **No** | **Yes** |
| deleteCourse() | **No** | **Yes** |
| insertCourse() | **No** | **Yes** |
| updateCourse() | **No** | **Yes** |
| **Criteria** | **Criteria()** | **Yes** | **Yes** |
| **getCriteriaByClass()** | **Yes** | **Yes** |
| **ExtendInformation** | deleteCourse() | **No** | **Yes** |
| deletePaper() | **No** | **Yes** |
| deleteQualifying() | **No** | **Yes** |
| ExtendInformation() | **No** | **Yes** |
| getExtendInformation() | **Yes** | **Yes** |
| insertCourse() | **Yes** | **Yes** |
| insertPaper() | **Yes** | **Yes** |

|  |  |  |  |
| --- | --- | --- | --- |
|  | insertQualifying() | **Yes** | **Yes** |
| updateCourse() | **No** | **Yes** |
| updatePaper() | **No** | **Yes** |
| updatePersonalInformation() | **Yes** | **Yes** |
| updateQualifying() | **No** | **Yes** |
| **Manager** | Login() | **Yes** | **Yes** |
| Logout() | **Yes** | **Yes** |
| Manager() | **No** | **Yes** |
| **Paper** | deletePaper() | **No** | **Yes** |
| insertPaper() | **No** | **Yes** |
| Paper() | **No** | **Yes** |
| updatePaper() | **No** | **Yes** |
| **PrimaryInformation** | getPrimaryInformation() | **Yes** | **Yes** |
| PrimaryInformation() | **No** | **Yes** |
| savePrimaryInformation() | **Yes** | **Yes** |
| **Qualifying** | deleteQualifying() | **No** | **Yes** |
| insertQualifying() | **No** | **Yes** |
| Qualifying() | **No** | **Yes** |
| updateQualifying() | **No** | **Yes** |
| **Student** | deleteCourse() | **No** | **Yes** |
| deletePaper() | **No** | **Yes** |
| deleteQualifying() | **No** | **Yes** |

|  |  |  |  |
| --- | --- | --- | --- |
|  | enterCourse() | **Yes** | **Yes** |
| enterPaper() | **Yes** | **Yes** |
| enterPersonalInformation() (new) | Yes | Yes |
| enterQualifying() | **Yes** | **Yes** |
| showExtendInformation() | **Yes** | **Yes** |
| Student() | **No** | **Yes** |
| updateCourse() | **No** | **Yes** |
| updatePaper() | **No** | **Yes** |
| updateQualifying() | **No** | **Yes** |
| **StudentInformation** | getStudentInformation() | **Yes** | **Yes** |
| StudentInformation() | **No** | **Yes** |
| **Program** | Main() | **No** | **Yes** |
| **frmAbout** | frmAbout() | **No** | **Yes** |
| **frmLogin** | frmLogin() | **No** | **Yes** |
| btLogin\_Click() | **No** | **Yes** |
| btExit\_Click() | **No** | **Yes** |
| **frmConnectDatabase** | btConnect\_Click() | **No** | **Yes** |
| btExit\_Click() | **No** | **Yes** |
| cbAuthenticatedSQL\_CheckedChanged() | **No** | **Yes** |
| frmConnectDatabase() | **No** | **Yes** |
| **Persistence** | connectDatabaseByAuthenticatedWindows() | **No** | **Yes** |
| connectDatabaseByAuthenticatedSQLServer() | **No** | **Yes** |

|  |  |  |  |
| --- | --- | --- | --- |
|  | closeConnectDatabase() | **No** | **Yes** |
| querySelect() | **No** | **Yes** |
| queryUpdate() | **No** | **Yes** |
| **dbAdvisor** | getAdvisorTable() | **No** | **Yes** |
| findAdvisorTable() | **No** | **Yes** |
| isExistAdvisorTable() | **No** | **Yes** |
| **dbCourse** | getCourseTable() | **No** | **Yes** |
| insertCourseTable() | **No** | **Yes** |
| deleteCourseTable() | **No** | **Yes** |
| updateCourseTable() | **No** | **Yes** |
| **dbCriteria** | **getCriteriaByClass()** | **No** | **Yes** |
| **dbPaper** | getPaperTable() | **No** | **Yes** |
| insertPaperTable() | **No** | **Yes** |
| deletePaperTable() | **No** | **Yes** |
| updatePaperTable() | **No** | **Yes** |
| **dbQualifying** | getQualifyingTable() | **No** | **Yes** |
| insertQualifyingTable() | **No** | **Yes** |
| deleteQualifyingTable() | **No** | **Yes** |
| updateQualifyingTable() | **No** | **Yes** |
| **dbStudent** | getStudentTable() | **No** | **Yes** |
| getPrimaryStudentTable() | **No** | **Yes** |
| getPrimaryInformation() | **No** | **Yes** |

|  |  |  |  |
| --- | --- | --- | --- |
|  | getPrimaryInformation() | **No** | **Yes** |
| getExtendInformation() | **No** | **Yes** |
| findStudentTableByStudentID() | **No** | **Yes** |
| findStudentTableByName() | **No** | **Yes** |
| findStudentTable() | **No** | **Yes** |
| isExistStudentTable() | **No** | **Yes** |
| insertPrimaryStudentTable() | **No** | **Yes** |
| deletePrimaryStudentTable() | **No** | **Yes** |
| updatePrimaryStudentTable() | **No** | **Yes** |
| updatePersonalStudentTable() | **No** | **Yes** |
| **MainForm** | aboutToolStripMenuItem\_Click() | **No** | **Yes** |
| AddCoursesToListview() | **No** | **Yes** |
| AddPapersToListview() | **No** | **Yes** |
| AddPrimaryStudentToListview() | **No** | **Yes** |
| AddQualificationsToListview() | **No** | **Yes** |
| AddStudentIDToCombobox() | **No** | **Yes** |
| AddStudentInformationToForm() | **No** | **Yes** |
| btAddStudent\_Click() | **No** | **Yes** |
| btCancelStudent\_Click() | **No** | **Yes** |
| btCourseAdd\_Click() | **No** | **Yes** |
| btCourseCancel\_Click() | **No** | **Yes** |
| btCourseDelete\_Click() | **No** | **Yes** |

|  |  |  |  |
| --- | --- | --- | --- |
|  | btCourseUpdate\_Click() | **No** | **Yes** |
| btDeleteStudent\_Click() | **No** | **Yes** |
| btPaperAdd\_Click() | **No** | **Yes** |
| btPaperCancel\_Click() | **No** | **Yes** |
| btPaperDelete\_Click() | **No** | **Yes** |
| btPaperUpdate\_Click() | **No** | **Yes** |
| btPersonalStudentCancel\_Click() | **No** | **Yes** |
| btPersonalStudentUpdate\_Click() | **No** | **Yes** |
| btQualifyingAdd\_Click() | **No** | **Yes** |
| btQualifyingCancel\_Click() | **No** | **Yes** |
| btQualifyingDelete\_Click() | **No** | **Yes** |
| btQualifyingUpdate\_Click() | **No** | **Yes** |
| btUpdate\_Click() | **No** | **Yes** |
| cbViewStudentID\_SelectedIndexChanged() | **No** | **Yes** |
| cbViewStudentID\_TextChanged() | **No** | **Yes** |
| ClearCourseForm() | **No** | **Yes** |
| ClearPaperForm() | **No** | **Yes** |
| ClearPersonalStudentForm() | **No** | **Yes** |
| ClearPrimaryStudentForm() | **No** | **Yes** |
| ClearQualifyingForm() | **No** | **Yes** |
| ClearViewStudentForm() | **No** | **Yes** |
| closeConnectionToolStripMenuItem\_Click() | **No** | **Yes** |

|  |  |  |  |
| --- | --- | --- | --- |
|  | ConnectSQLServerToolStripMenuItem\_Click() | **No** | **Yes** |
| exitToolStripMenuItem\_Click() | **No** | **Yes** |
| loginToolStripMenuItem\_Click() | **No** | **Yes** |
| logoutToolStripMenuItem\_Click() | **No** | **Yes** |
| lvCourse\_SelectedIndexChanged() | **No** | **Yes** |
| lvPaper\_SelectedIndexChanged() | **No** | **Yes** |
| lvPrimaryStudent\_SelectedIndexChanged() | **No** | **Yes** |
| lvQualifying\_SelectedIndexChanged() | **No** | **Yes** |
| MainForm() | **No** | **Yes** |
| qualifyingToolStripMenuItem\_Click() | **No** | **Yes** |
| setConnectResult() | **No** | **Yes** |
| setEnableLogin() | **No** | **Yes** |
| setEnableLogout() | **No** | **Yes** |
| setInitForAdvisor() | **No** | **Yes** |
| setInitForStudent() | **No** | **Yes** |
| setLoginResult() | **No** | **Yes** |

**Table 5.2.2: Summary of implementation class/method changed**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Number of added** | **Number of removed** | **Number of modified** |
| **Class** | 12 | 0 | 3 |
| **Method** | 78 | 0 | 2 |

* 1. **Calculate Line of Code (new)**

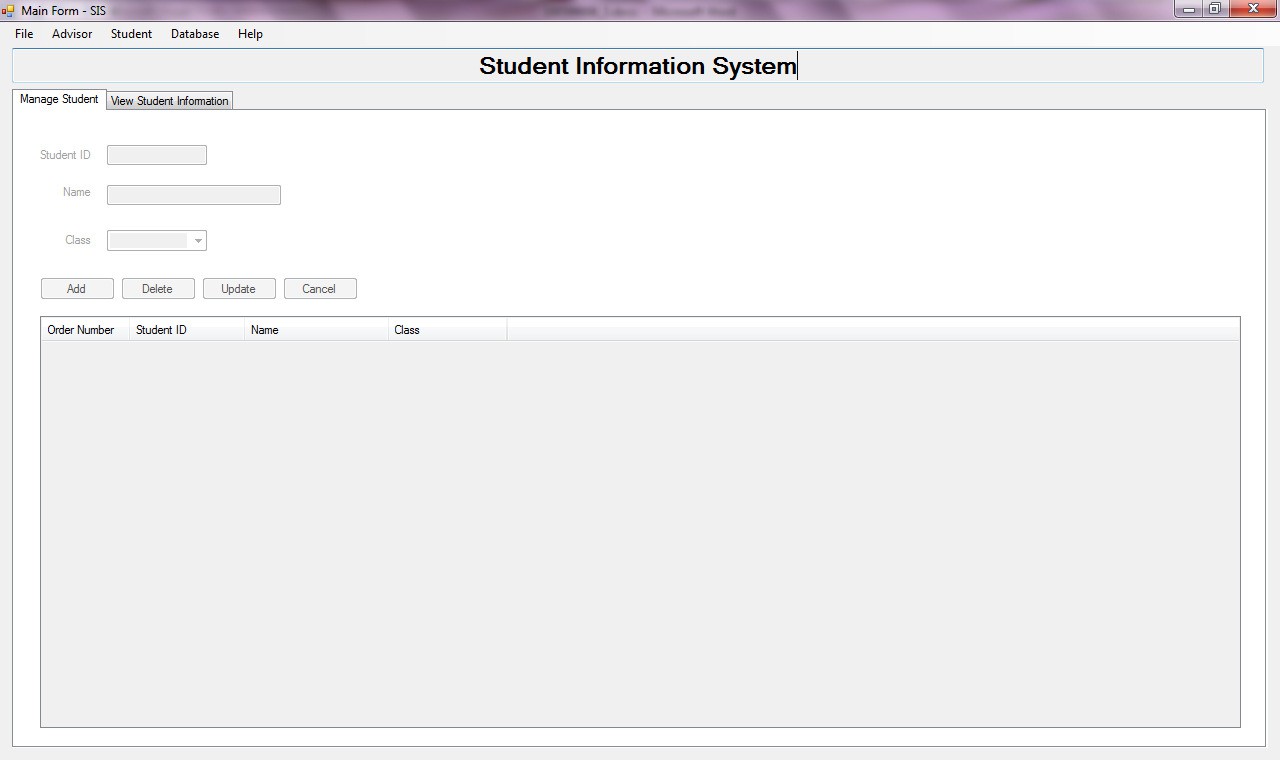
**Table 5.3.1: Line of Code of classes**

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Class Name** | **Number of methods** | **Line of Code in Class**  **(without comment)** |
| **1** | Program | 1 | 19 |
| **2** | MainForm | 31 | 603 |
| **3** | frmAbout | 1 | 19 |
| **4** | frmLogin | 3 | 46 |
| **5** | frmConnectDatabase | 3 | 62 |
| **6** | Advisor | 5 | 57 |
| **7** | Manager | 6 | 92 |
| **8** | Student | 5 | 79 |
| **9** | PrimaryInformation | 8 | 86 |
| **10** | StudentInformation | 5 | 77 |
| **11** | ExtendInformation | 23 | 183 |
| **12** | Qualifying | 5 | 51 |
| **13** | Course | 6 | 84 |
| **14** | **Criteria** | **8** | **105** |
| **15** | Paper | 6 | 83 |
| **16** | dbStudent | 13 | 151 |
| **17** | Persistence | 5 | 88 |
| **18** | dbAdvisor | 3 | 35 |
| **19** | **dbCriteria** | **1** | **32** |
| **20** | dbCourse | 4 | 49 |

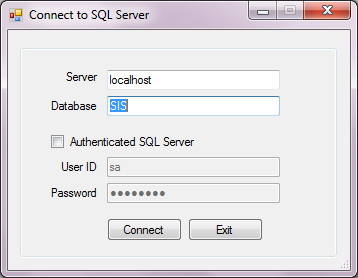
|  |  |  |  |
| --- | --- | --- | --- |
| **21** | dbPaper | 4 | 49 |
| **22** | dbQualifying | 4 | 46 |
| **SUM** | | **150** | **2096** |

# Programming

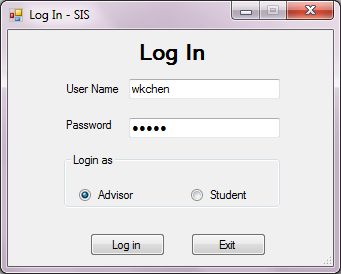
# Snapshots of system execution



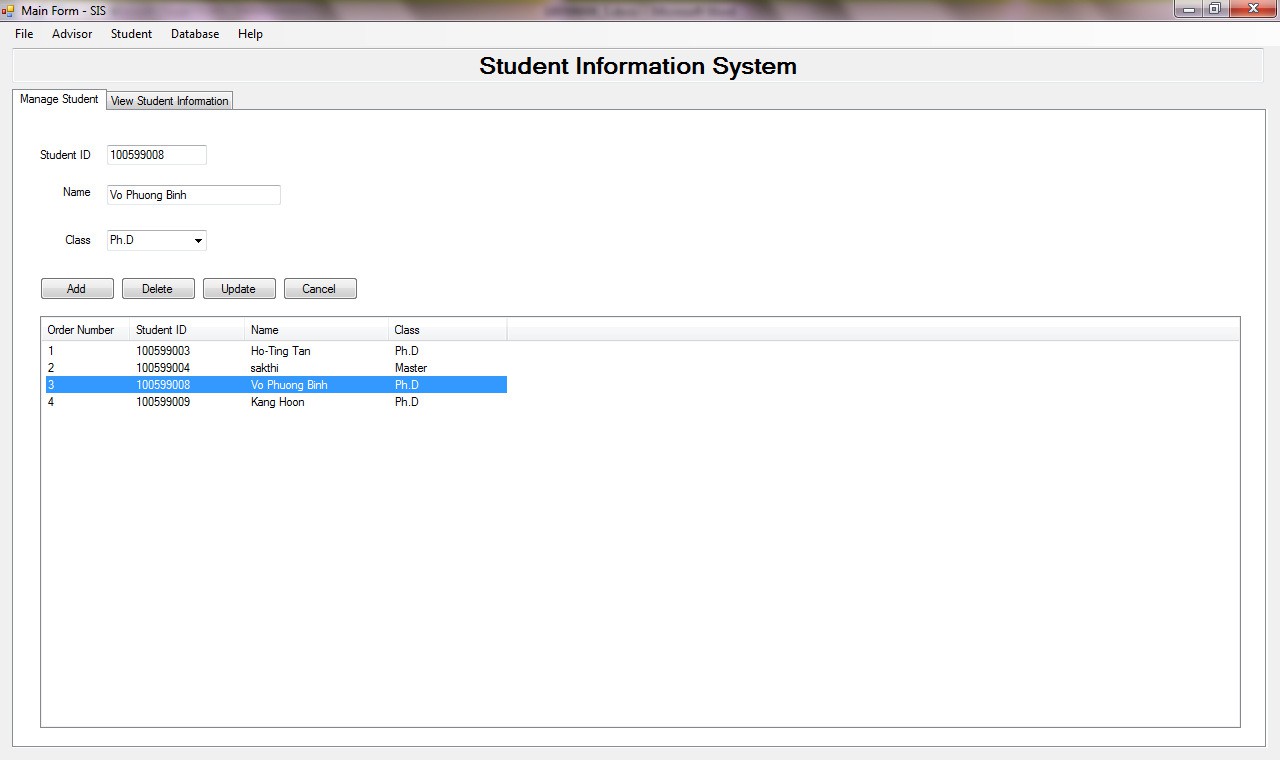
***Figure 6.1 Main form of SIS***



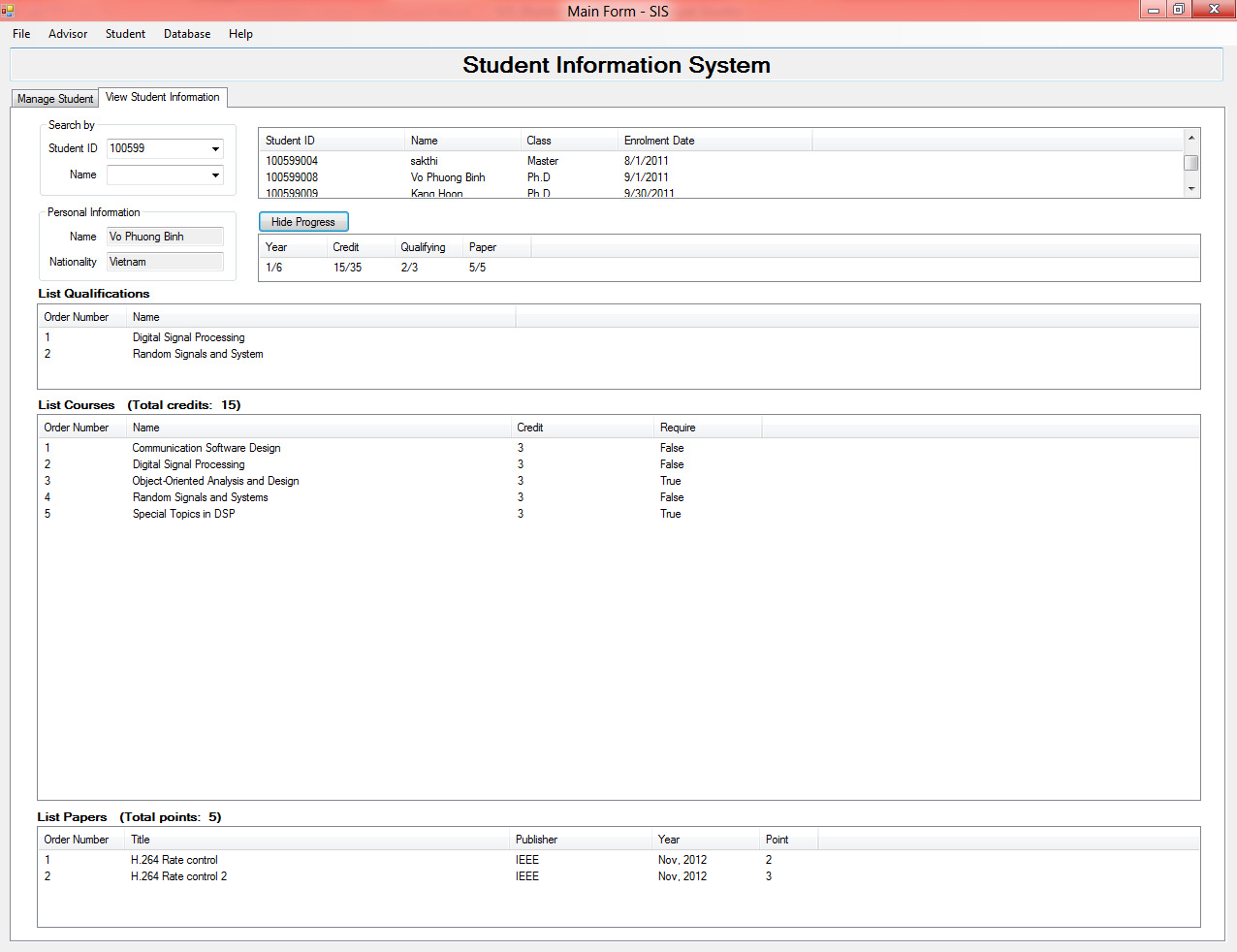
***Figure 6.2 Connect to SQL Server (menu: Database/Connect SQL Server)***



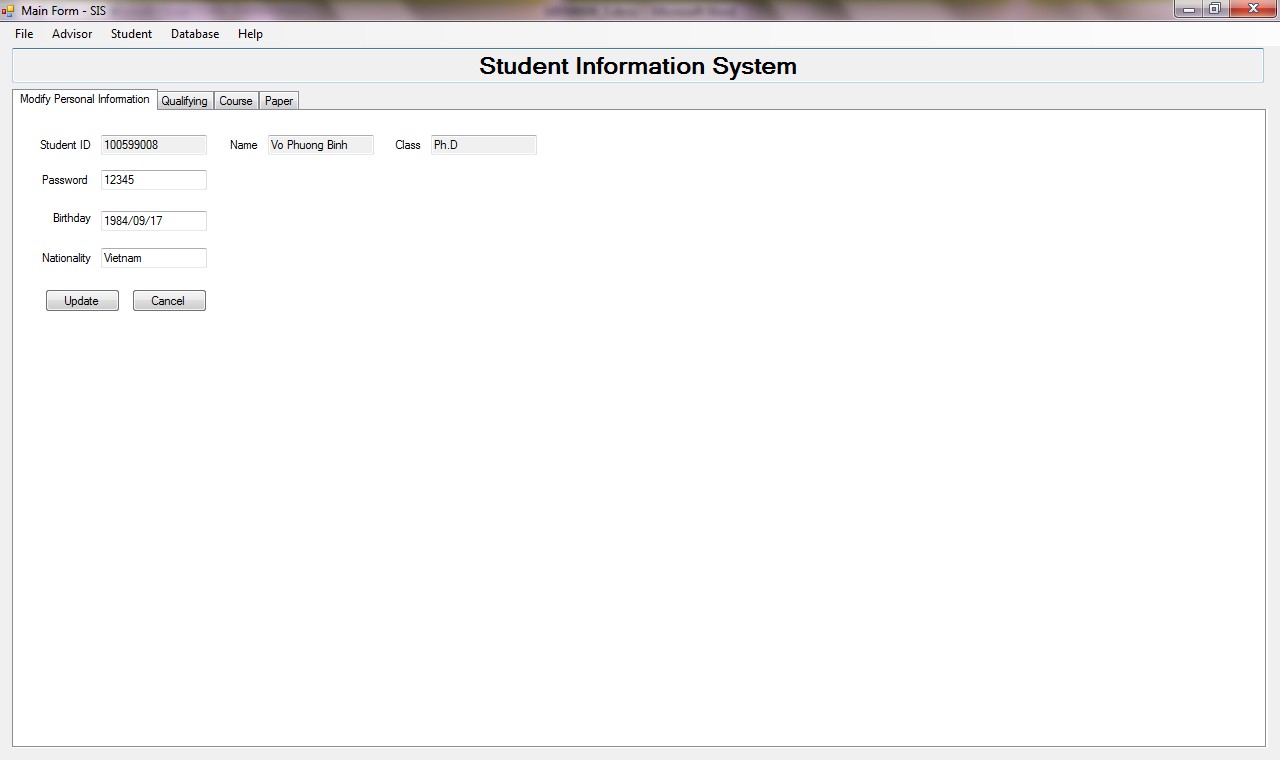
***Figure 6.3 Login to SIS as an Advisor or a Student (menu: File/Login)***



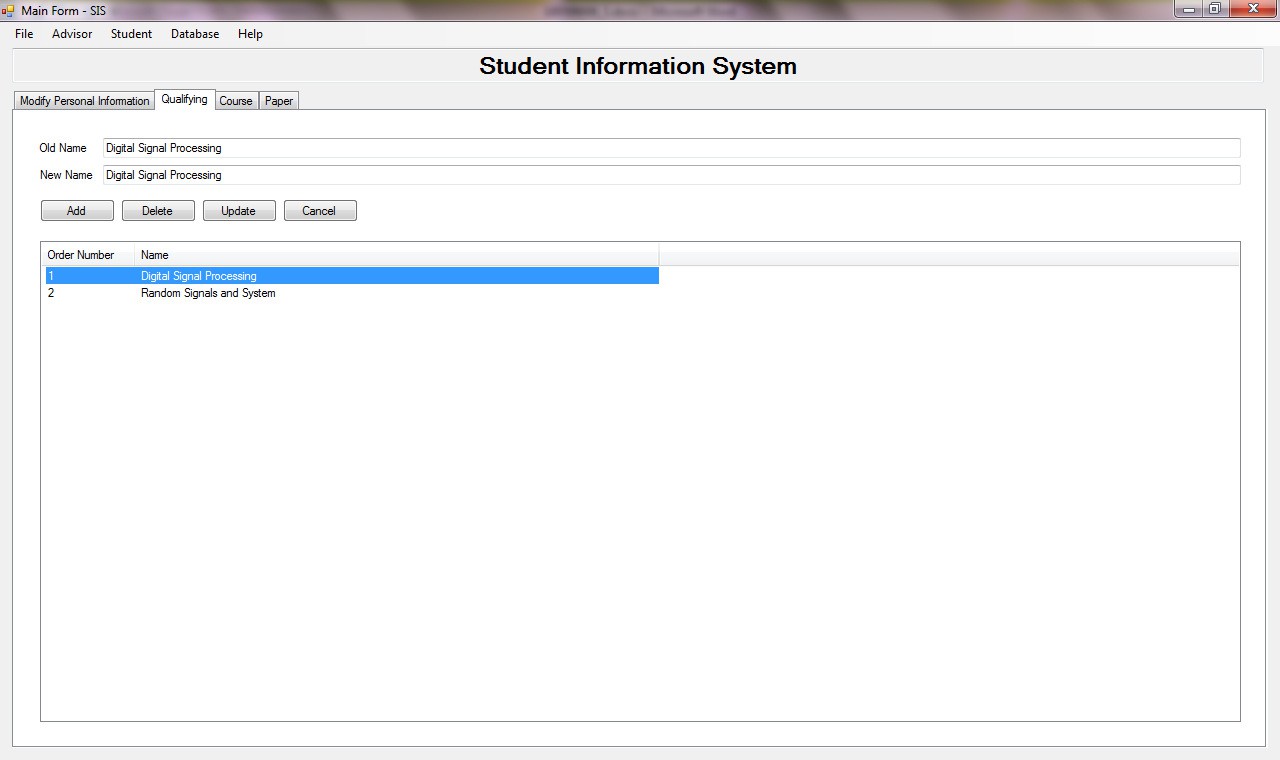
***Figure 6.4 GUI of manage Student for the Advisor logged in***



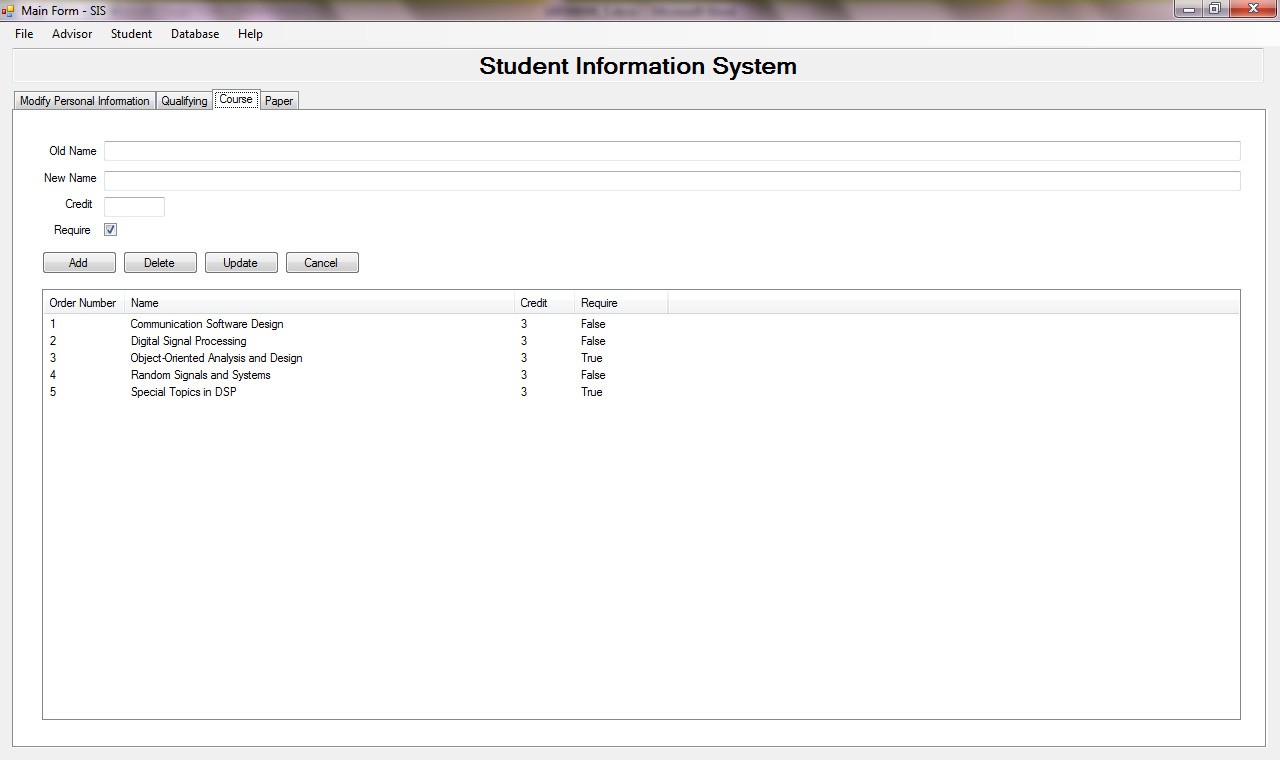
***Figure 6.5 GUI of View Student Information for the Advisor logged in***



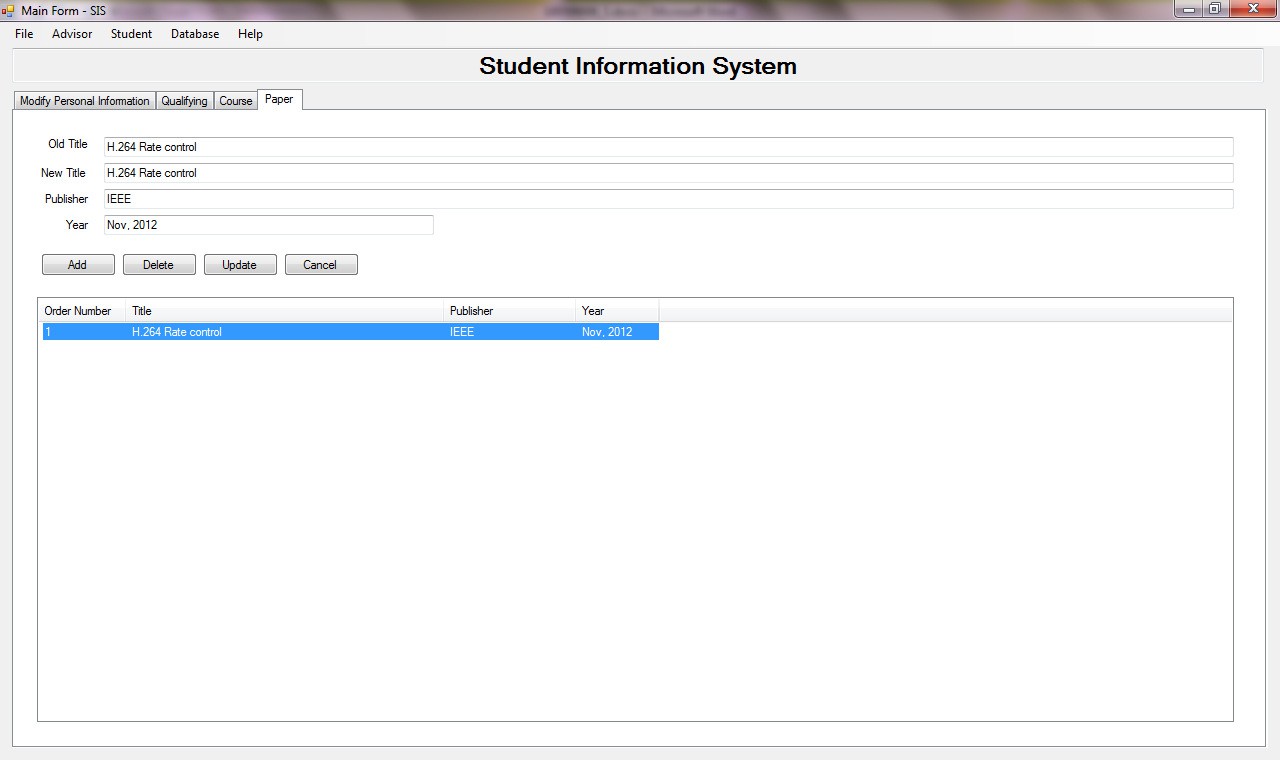
***Figure 6.6 GUI of Modify Personal Information for the Student logged in***



***Figure 6.6 GUI of Modify Qualifying for the Student logged in***



***Figure 6.6 GUI of Modify Course for the Student logged in***



***Figure 6.6 GUI of Modify Course for the Student logged in***

# Source Code Listing

# Program.cs

using System;

using System.Collections.Generic; using System.Linq;

using System.Windows.Forms; using SIS.UI.GUI;

namespace SIS

{

static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary> [STAThread]

static void Main()

{

Application.EnableVisualStyles(); Application.SetCompatibleTextRenderingDefault(false); Application.Run(new MainForm());

}

}

}

# MainForm.cs

using System;

using System.Collections.Generic; using System.ComponentModel; using System.Drawing;

using System.Linq; using System.Text;

using System.Windows.Forms;

using SIS.TechnicalServices.Persistence; using SIS.Domain.Manage;

using SIS.Domain.Inventory;

namespace SIS.UI.GUI

{

public partial class MainForm : Form

{

private static Manager \_manager = new Manager(); private Advisor \_advisor = new Advisor(); private Student \_student = new Student();

private StudentInformation \_studentInformation = new StudentInformation();

private static bool \_isConnection = false;

public static Manager \_Manager

{

get

{

}

set

{

return \_manager;

\_manager = value;

}

}

public static bool \_IsConnection

{

get

{

}

set

{

}

}

return \_isConnection;

\_isConnection = value;

public MainForm()

{

InitializeComponent();

int x = tabAdvisor.Location.X; int y = tabAdvisor.Location.Y; int width = tabAdvisor.Width; int height = tabAdvisor.Height;

tabStudent.SetBounds(x, y, width, height);

}

private void exitToolStripMenuItem\_Click(object sender, EventArgs e)

{

Close();

}

private void MainForm\_Load(object sender, EventArgs e)

{

}

private void AddPrimaryStudentToListview()

{

List<PrimaryInformation> \_priInformations = PrimaryInformation.getPrimaryInformation();

lvPrimaryStudent.Items.Clear();

for (int i = 0; i < \_priInformations.Count; i++)

{

ListViewItem lvi = new ListViewItem(new string[] { (i + 1).ToString(), \_priInformations[i].StudentID, \_priInformations[i].Name,

\_priInformations[i].Rank });

lvPrimaryStudent.Items.Add(lvi);

}

}

private void loginToolStripMenuItem\_Click(object sender, EventArgs e)

{

new frmLogin().ShowDialog();

if (\_manager != null)

{

setEnableLogin();

}

}

public static void setLoginResult(Manager manager)

{

\_manager = manager;

}

public static void setConnectResult(bool isConnection)

{

\_isConnection = isConnection;

}

private void logoutToolStripMenuItem\_Click(object sender, EventArgs

e)

{

setEnableLogout();

}

private void setEnableLogin()

{

// Menus loginToolStripMenuItem.Enabled = false; logoutToolStripMenuItem.Enabled = true;

if (\_manager.IsAdvisor)

{

// Create Advisor

\_advisor = new Advisor(\_manager.UserName, \_manager.Password);

// Set init for Advisor setInitForAdvisor();

}

else

{

// Read PrimaryStudent to Listview AddPrimaryStudentToListview();

// Create Student

\_student = new Student(\_manager.UserName, \_manager.Password);

// Set init for Student setInitForStudent();

}

}

private void setInitForAdvisor()

{

//Menus manageStudentToolStripMenuItem.Enabled = true;

viewStudentInformationToolStripMenuItem.Enabled = true;

PersonalInformationToolStripMenuItem.Enabled = false; qualifyingToolStripMenuItem.Enabled = false; courseToolStripMenuItem.Enabled = false; paperToolStripMenuItem.Enabled = false;

// Set enable for components tabAdvisor.Visible = true; tabAdvisor.Enabled = true;

tabStudent.Visible = false;

// Set initial for class combobox cbStudentClass.SelectedIndex = 0;

// Read StudentID and add to combobox AddStudentIDToCombobox();

}

private void setInitForStudent()

{

// Menus manageStudentToolStripMenuItem.Enabled = false;

viewStudentInformationToolStripMenuItem.Enabled = false;

PersonalInformationToolStripMenuItem.Enabled = true; qualifyingToolStripMenuItem.Enabled = true; courseToolStripMenuItem.Enabled = true; paperToolStripMenuItem.Enabled = true;

// Set enable for components tabAdvisor.Visible = false;

tabStudent.Visible = true; tabStudent.Enabled = true;

// Get Student Information from database

\_studentInformation = StudentInformation.getStudentInformation(\_student.UserName);

// Set Personal Information txtPersonalStudentID.Text =

\_studentInformation.PriInformation.StudentID;

txtPersonalName.Text = \_studentInformation.PriInformation.Name; txtPersonalClass.Text = \_studentInformation.PriInformation.Rank; txtPersonalBirthday.Text =

\_studentInformation.ExtInformation.Birthday; txtPersonalNationality.Text =

\_studentInformation.ExtInformation.National; txtPersonalPassword.Text =

\_studentInformation.ExtInformation.Password;

// Set Qualifying

List<Qualifying> \_qualifications =

\_studentInformation.ExtInformation.Qualifications;

for (int i = 0; i < \_qualifications.Count; i++)

{

ListViewItem lvi = new ListViewItem(new string[] { (i + 1).ToString(), \_qualifications[i].Name });

lvQualifying.Items.Add(lvi);

}

// Set Course List<Course> \_course =

\_studentInformation.ExtInformation.Courses;

for (int i = 0; i < \_course.Count; i++)

{

ListViewItem lvi = new ListViewItem(new string[] { (i + 1).ToString(), \_course[i].Name, \_course[i].Credit.ToString(),

\_course[i].Require.ToString() });

lvCourse.Items.Add(lvi);

}

// Set Paper

List<Paper> \_papers = \_studentInformation.ExtInformation.Papers; for (int i = 0; i < \_papers.Count; i++)

{

ListViewItem lvi = new ListViewItem(new string[] { (i + 1).ToString(), \_papers[i].Title, \_papers[i].Publisher, \_papers[i].Year });

lvPaper.Items.Add(lvi);

}

}

private void AddStudentIDToCombobox()

{

List<string> \_listStudentID = dbStudent.getStudentIDTable(); cbViewStudentID.Items.Clear();

for (int i = 0; i < \_listStudentID.Count; i++)

{

// Add StudentID to combobox cbViewStudentID.Items.Add(\_listStudentID[i]);

}

if (cbViewStudentID.Items.Count > 0)

{

cbViewStudentID.SelectedIndex = 0;

}

}

private void setEnableLogout()

{

// Menus loginToolStripMenuItem.Enabled = true;

logoutToolStripMenuItem.Enabled = false; manageStudentToolStripMenuItem.Enabled = false; viewStudentInformationToolStripMenuItem.Enabled = false; PersonalInformationToolStripMenuItem.Enabled = false; qualifyingToolStripMenuItem.Enabled = false; courseToolStripMenuItem.Enabled = false; paperToolStripMenuItem.Enabled = false;

// Components tabAdvisor.Visible = true; tabAdvisor.Enabled = false;

tabStudent.Visible = false;

}

private void btCancelStudent\_Click(object sender, EventArgs e)

{

ClearPrimaryStudentForm();

}

// Clear all fields of PrimaryStudent Form private void ClearPrimaryStudentForm()

{

// Clear all fields of PrimaryStudent Form txtStudentID.Text = ""; txtStudentName.Text = ""; cbStudentClass.SelectedIndex = 0;

}

private void btAddStudent\_Click(object sender, EventArgs e)

{

// Insert PrimaryStudent to Database

\_advisor.enterPrimaryInformation(txtStudentID.Text, txtStudentName.Text, cbStudentClass.Text);

// Refresh the listview data of PrimaryStudent AddPrimaryStudentToListview();

// Read StudentID and add to combobox AddStudentIDToCombobox();

}

private void cbViewStudentID\_SelectedIndexChanged(object sender, EventArgs e)

{

if (\_advisor == null) return;

StudentInformation \_studentInformation =

\_advisor.viewStudentInformation(cbViewStudentID.Text);

if (\_studentInformation == null)

{

//Clear all fields of components ClearViewStudentForm();

return;

}

//Add all fields to components AddStudentInformationToForm(\_studentInformation);

}

private void cbViewStudentID\_TextChanged(object sender, EventArgs e)

{

if (\_advisor == null) return;

StudentInformation \_studentInformation =

\_advisor.viewStudentInformation(cbViewStudentID.Text);

if (\_studentInformation == null)

{

//Clear all fields of components ClearViewStudentForm();

return;

}

//Add all fields to components AddStudentInformationToForm(\_studentInformation);

}

private void AddStudentInformationToForm(StudentInformation

\_studentInformation)

{

// Set student fields to components

txtViewStudentName.Text =

\_studentInformation.PriInformation.Name; txtViewStudentClass.Text =

\_studentInformation.PriInformation.Rank; txtViewStudentBirthday.Text =

\_studentInformation.ExtInformation.Birthday; txtViewStudentNational.Text =

\_studentInformation.ExtInformation.National;

// Qualifications

List<Qualifying> \_qualifications =

\_studentInformation.ExtInformation.Qualifications; lvViewStudentQualifying.Items.Clear();

for (int i = 0; i < \_qualifications.Count; i++)

{

ListViewItem lvi = new ListViewItem(new string[] { (i+1).ToString(), \_qualifications[i].Name });

lvViewStudentQualifying.Items.Add(lvi);

}

// Courses List<Course> \_courses =

\_studentInformation.ExtInformation.Courses; lvViewStudentCourse.Items.Clear(); int totalCredit = 0;

for (int i = 0; i < \_courses.Count; i++)

{

ListViewItem lvi = new ListViewItem(new string[] { (i + 1).ToString(), \_courses[i].Name, \_courses[i].Credit.ToString(),

\_courses[i].Require.ToString() });

lvViewStudentCourse.Items.Add(lvi);

totalCredit += \_courses[i].Credit;

}

lbTotalCredit.Text = "(Total credits: " + totalCredit.ToString()

+ ")";

// Papers

List<Paper> \_papers = \_studentInformation.ExtInformation.Papers; lvViewStudentPaper.Items.Clear();

for (int i = 0; i < \_papers.Count; i++)

{

ListViewItem lvi = new ListViewItem(new string[] { (i + 1).ToString(), \_papers[i].Title, \_papers[i].Publisher, \_papers[i].Year });

lvViewStudentPaper.Items.Add(lvi);

}

}

private void ClearViewStudentForm()

{

txtViewStudentName.Text = ""; txtViewStudentClass.Text = ""; txtViewStudentBirthday.Text = ""; txtViewStudentNational.Text = ""; lvViewStudentQualifying.Items.Clear(); lvViewStudentCourse.Items.Clear();

lvViewStudentPaper.Items.Clear();

}

private void lvPrimaryStudent\_SelectedIndexChanged(object sender, EventArgs e)

{

try

{

}

ListViewItem lvi = lvPrimaryStudent.SelectedItems[0]; txtStudentID.Text = lvi.SubItems[1].Text; txtStudentName.Text = lvi.SubItems[2].Text; cbStudentClass.Text = lvi.SubItems[3].Text;

catch { }

}

private void btDeleteStudent\_Click(object sender, EventArgs e)

{

// Delete PrimaryStudent from Database

\_advisor.deletePrimaryInformation(txtStudentID.Text);

// Refresh the listview data of PrimaryStudent AddPrimaryStudentToListview();

// Read StudentID and add to combobox AddStudentIDToCombobox();

}

private void btUpdate\_Click(object sender, EventArgs e)

{

// Update PrimaryStudent to Database

\_advisor.updatePrimaryInformation(txtStudentID.Text, txtStudentName.Text, cbStudentClass.Text);

// Refresh the listview data of PrimaryStudent AddPrimaryStudentToListview();

// Read StudentID and add to combobox AddStudentIDToCombobox();

}

private void lvQualifying\_SelectedIndexChanged(object sender, EventArgs e)

{

try

{

}

ListViewItem lvi = lvQualifying.SelectedItems[0]; txtQualifyingNameOld.Text = lvi.SubItems[1].Text; txtQualifyingName.Text = lvi.SubItems[1].Text;

catch { }

}

private void btPersonalStudentCancel\_Click(object sender, EventArgs

e)

{

ClearPersonalStudentForm();

}

private void ClearPersonalStudentForm()

{

txtPersonalBirthday.Text = ""; txtPersonalNationality.Text = "";

}

private void btPersonalStudentUpdate\_Click(object sender, EventArgs

e)

{

bool isOK =

\_student.enterPersonalInformation(\_studentInformation.StudentID, txtPersonalBirthday.Text, txtPersonalNationality.Text, txtPersonalPassword.Text);

if (isOK)

{

}

else

{

}

}

MessageBox.Show("Updated successfully!");

MessageBox.Show("Updated error!");

private void lvCourse\_SelectedIndexChanged(object sender, EventArgs

e)

{

try

{

ListViewItem lvi = lvCourse.SelectedItems[0]; txtCourseNameOld.Text = lvi.SubItems[1].Text; txtCourseName.Text = lvi.SubItems[1].Text; txtCourseCredit.Text = lvi.SubItems[2].Text; rdCourseRequire.Checked =

(lvi.SubItems[3].Text.CompareTo("True") == 0);

}

catch { }

}

private void lvPaper\_SelectedIndexChanged(object sender, EventArgs e)

{

try

{

}

ListViewItem lvi = lvPaper.SelectedItems[0]; txtPaperTitleOld.Text = lvi.SubItems[1].Text; txtPaperTitle.Text = lvi.SubItems[1].Text; txtPaperPublisher.Text = lvi.SubItems[2].Text; txtPaperYear.Text = lvi.SubItems[3].Text;

catch { }

}

private void btQualifyingCancel\_Click(object sender, EventArgs e)

{

ClearQualifyingForm();

}

private void ClearQualifyingForm()

{

txtQualifyingName.Text = "";

txtQualifyingNameOld.Text = "";

}

private void btQualifyingAdd\_Click(object sender, EventArgs e)

{

// Insert into database bool isOK =

\_student.enterQualifying(\_studentInformation.StudentID, txtQualifyingName.Text);

if (!isOK)

{

MessageBox.Show("This qualifying existed."); return;

}

// Refresh the listview data of Qualifications AddQualificationsToListview();

}

private void AddQualificationsToListview()

{

List<Qualifying> \_quanlifications = dbQualifying.getQualifyingTable(\_studentInformation.StudentID);

lvQualifying.Items.Clear();

for (int i = 0; i < \_quanlifications.Count; i++)

{

ListViewItem lvi = new ListViewItem(new string[] { (i + 1).ToString(), \_quanlifications[i].Name });

lvQualifying.Items.Add(lvi);

}

}

private void btQualifyingDelete\_Click(object sender, EventArgs e)

{

// Delete from database

\_student.deleteQualifying(\_studentInformation.StudentID, txtQualifyingName.Text);

// Refresh the listview data of Qualifications AddQualificationsToListview();

}

private void btQualifyingUpdate\_Click(object sender, EventArgs e)

{

// Update to database

\_student.updateQualifying(\_studentInformation.StudentID, txtQualifyingNameOld.Text, txtQualifyingName.Text);

// Refresh the listview data of Qualifications AddQualificationsToListview();

}

private void btCourseAdd\_Click(object sender, EventArgs e)

{

// Check valid credit byte credit;

bool isOK = byte.TryParse(txtCourseCredit.Text, out credit); if (!isOK)

{

MessageBox.Show("Invalid Credit!"); return;

}

// Insert into database

\_student.enterCourse(\_studentInformation.StudentID, txtCourseName.Text, credit, rdCourseRequire.Checked);

// Refresh the listview data of Courses AddCoursesToListview();

}

private void AddCoursesToListview()

{

List<Course> \_courses = dbCourse.getCourseTable(\_studentInformation.StudentID);

lvCourse.Items.Clear();

for (int i = 0; i < \_courses.Count; i++)

{

ListViewItem lvi = new ListViewItem(new string[] { (i + 1).ToString(), \_courses[i].Name, \_courses[i].Credit.ToString(),

\_courses[i].Require.ToString() });

lvCourse.Items.Add(lvi);

}

}

private void btCourseDelete\_Click(object sender, EventArgs e)

{

// Delete from database

\_student.deleteCourse(\_studentInformation.StudentID, txtCourseName.Text);

// Refresh the listview data of Courses AddCoursesToListview();

}

private void btCourseUpdate\_Click(object sender, EventArgs e)

{

// Check valid credit byte credit;

bool isOK = byte.TryParse(txtCourseCredit.Text, out credit); if (!isOK)

{

MessageBox.Show("Invalid Credit!"); return;

}

// Update into database

\_student.updateCourse(\_studentInformation.StudentID, txtCourseNameOld.Text, txtCourseName.Text, credit, rdCourseRequire.Checked);

// Refresh the listview data of Courses AddCoursesToListview();

}

private void btCourseCancel\_Click(object sender, EventArgs e)

{

ClearCourseForm();

}

private void ClearCourseForm()

{

txtCourseNameOld.Text = ""; txtCourseName.Text = ""; txtCourseCredit.Text = ""; rdCourseRequire.Checked = false;

}

private void btPaperCancel\_Click(object sender, EventArgs e)

{

ClearPaperForm();

}

private void ClearPaperForm()

{

txtPaperTitleOld.Text = ""; txtPaperTitle.Text = ""; txtPaperPublisher.Text = ""; txtPaperYear.Text = "";

}

private void btPaperAdd\_Click(object sender, EventArgs e)

{

// Insert into database

\_student.enterPaper(\_studentInformation.StudentID, txtPaperTitle.Text, txtPaperPublisher.Text, txtPaperYear.Text);

// Refresh the listview data of Courses AddPapersToListview();

}

private void AddPapersToListview()

{

List<Paper> \_papers = dbPaper.getPaperTable(\_studentInformation.StudentID);

lvPaper.Items.Clear();

for (int i = 0; i < \_papers.Count; i++)

{

ListViewItem lvi = new ListViewItem(new string[] { (i + 1).ToString(), \_papers[i].Title, \_papers[i].Publisher, \_papers[i].Year });

lvPaper.Items.Add(lvi);

}

}

private void btPaperDelete\_Click(object sender, EventArgs e)

{

// Insert into database

\_student.deletePaper(\_studentInformation.StudentID, txtPaperTitle.Text);

// Refresh the listview data of Courses AddPapersToListview();

}

private void btPaperUpdate\_Click(object sender, EventArgs e)

{

// Insert into database

\_student.updatePaper(\_studentInformation.StudentID, txtPaperTitleOld.Text, txtPaperTitle.Text, txtPaperPublisher.Text, txtPaperYear.Text);

// Refresh the listview data of Courses AddPapersToListview();

}

private void ConnectSQLServerToolStripMenuItem\_Click(object sender, EventArgs e)

{

// Open connection form to SQL Server new frmConnectDatabase().ShowDialog();

loginToolStripMenuItem.Enabled = \_isConnection; ConnectSQLServerToolStripMenuItem.Enabled = !\_isConnection; closeConnectionToolStripMenuItem.Enabled = \_isConnection;

}

private void closeConnectionToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (PersistenceSQLServer.closeConnectDatabase())

{

}

else

{

}

}

MessageBox.Show("Connection closed successfully."); closeConnectionToolStripMenuItem.Enabled = false; ConnectSQLServerToolStripMenuItem.Enabled = true; loginToolStripMenuItem.Enabled = false;

MessageBox.Show("Connection not closed.");

private void aboutToolStripMenuItem\_Click(object sender, EventArgs e)

{

new frmAbout().ShowDialog();

}

private void qualifyingToolStripMenuItem\_Click(object sender, EventArgs e)

{

tabStudent.TabPages[1].Focus();

}

}

}

# frmLogin.cs

using System;

using System.Collections.Generic; using System.ComponentModel; using System.Data;

using System.Drawing;

using System.Linq; using System.Text;

using System.Windows.Forms;

using SIS.TechnicalServices.Persistence; using SIS.Domain.Manage;

namespace SIS.UI.GUI

{

public partial class frmLogin : Form

{

public frmLogin()

{

InitializeComponent();

}

private void btExit\_Click(object sender, EventArgs e)

{

MainForm.setLoginResult(null); Close();

}

private void btLogin\_Click(object sender, EventArgs e)

{

string userName = txtUserName.Text; string password = txtPassword.Text; bool isAdvisor = rdAdvisor.Checked;

Manager manager = new Manager(userName, password, isAdvisor); bool isExist = manager.Login();

if (!isExist)

{

}

else

{

}

}

}

}

MessageBox.Show("UserName and Password are not existed!");

MainForm.setLoginResult(manager); Close();

# frmConnectDatabase.cs

using System;

using System.Collections.Generic; using System.ComponentModel; using System.Data;

using System.Drawing; using System.Linq; using System.Text;

using System.Windows.Forms;

using SIS.TechnicalServices.Persistence;

namespace SIS.UI.GUI

{

public partial class frmConnectDatabase : Form

{

public frmConnectDatabase()

{

InitializeComponent();

}

private void cbAuthenticatedSQL\_CheckedChanged(object sender, EventArgs e)

{

txtUserID.Enabled = cbAuthenticatedSQL.Checked; txtPassword.Enabled = cbAuthenticatedSQL.Checked;

}

private void btConnect\_Click(object sender, EventArgs e)

{

if (cbAuthenticatedSQL.Checked == false)

{

bool isOK = PersistenceSQLServer.connectDatabaseByAuthenticatedWindows(txtServer.Text, txtDatabase.Text);

if (!isOK)

{

MessageBox.Show("Connection error!");

}

else

{

MainForm.setConnectResult(true); Close();

}

}

else

{

bool isOK = PersistenceSQLServer.connectDatabaseByAuthenticatedSQLServer(txtServer.Text, txtDatabase.Text, txtUserID.Text, txtPassword.Text);

if (!isOK)

{

MessageBox.Show("Connection error!");

}

else

{

MainForm.setConnectResult(true); Close();

}

}

}

private void btExit\_Click(object sender, EventArgs e)

{

MainForm.setConnectResult(false); Close();

}

}

}

# frmAbout.cs

using System;

using System.Collections.Generic; using System.ComponentModel; using System.Data;

using System.Drawing; using System.Linq; using System.Text;

using System.Windows.Forms;

namespace SIS.UI.GUI

{

public partial class frmAbout : Form

{

public frmAbout()

{

InitializeComponent();

}

}

}

# Persistence.cs

using System;

using System.Collections.Generic; using System.Linq;

using System.Text; using System.Data;

using System.Data.SqlClient;

namespace SIS.TechnicalServices.Persistence

{

public class PersistenceSQLServer

{

//Private Variables:

static SqlConnection \_connection = new SqlConnection();

public static SqlConnection Connection

{

get

{

}

set

{

}

}

return \_connection;

\_connection = value;

// Open Connection with Authenticated Windows

public static bool connectDatabaseByAuthenticatedWindows(string server, string database)

{

// Close current connection if opened closeConnectDatabase();

// Connect to database

string strSQL = "Data Source='" + server + "';Initial Catalog='"

+ database + "';Integrated Security=True"; Connection = new SqlConnection(strSQL); Connection.Open();

return (Connection.State == ConnectionState.Open);

}

// Open Connection with Authenticated SQL Server

public static bool connectDatabaseByAuthenticatedSQLServer(string server, string database, string userID, string password)

{

// Close current connection if opened closeConnectDatabase();

// Connect to database

string strSQL = "Data Source='" + server + "';Initial Catalog='"

+ database + "';uid='" + userID + "';pwd='" + password + "'"; Connection = new SqlConnection(strSQL); Connection.Open();

return (Connection.State == ConnectionState.Open);

}

// Close Connection

public static bool closeConnectDatabase()

{

if(Connection.State == ConnectionState.Open) Connection.Close();

return (Connection.State == ConnectionState.Closed);

}

// Query a select SQL, return a Table

public static DataTable querySelect(string strSQL)

{

SqlCommand cmd = new SqlCommand(strSQL, \_connection); SqlDataAdapter adt = new SqlDataAdapter(cmd); DataTable dtResult = new DataTable(); adt.Fill(dtResult);

return dtResult;

}

// Query a insert or update SQL

public static bool queryUpdate(string strSQL)

{

try

{

}

SqlCommand cmd = new SqlCommand(strSQL, \_connection); return (cmd.ExecuteNonQuery() > 0);

catch { return false; }

}

}

}

# Advisor.cs

using System;

using System.Collections.Generic; using System.Linq;

using System.Text; using System.Data;

using SIS.Domain.Inventory;

namespace SIS.Domain.Manage

{

public class Advisor: Manager

{

public Advisor()

: base()

{

}

public Advisor(string userName, string password)

: base(userName, password)

{

}

public bool enterPrimaryInformation(string studentID, string name, string rank)

{

// Create Primary Information PrimaryInformation \_priInformation = new

PrimaryInformation(studentID, name, rank);

// Insert into database

return \_priInformation.savePrimaryInformation();

}

public bool deletePrimaryInformation(string studentID)

{

// Delete from database

return dbStudent.deletePrimaryStudentTable(studentID);

}

public bool updatePrimaryInformation(string studentID, string name, string rank)

{

rank);

}

// Update to database

return dbStudent.updatePrimaryStudentTable(studentID, name,

public PrimaryInformation showPrimaryInformation(string studentID)

{

// Get from database

return PrimaryInformation.getPrimaryInformation(studentID);

}

public StudentInformation viewStudentInformation(string studentID)

{

// Get from database

return StudentInformation.getStudentInformation(studentID);

}

}

}

# Course.cs

using System;

using System.Collections.Generic; using System.Linq;

using System.Text;

using SIS.Domain.Inventory;

namespace SIS.Domain.Manage

{

public class Course

{

private string \_name; private byte \_credit; private bool \_require;

public string Name

{

get

{

}

set

{

}

}

return \_name;

\_name = value;

public byte Credit

{

get

{

}

set

{

}

}

return \_credit;

\_credit = value;

public bool Require

{

get

{

return \_require;

}

set

{

}

}

\_require = value;

public Course()

{

Name = ""; Credit = 0; Require = false;

}

public Course(string name, byte credit, bool require)

{

Name = name; Credit = credit; Require = require;

}

public static bool insertCourse(string studentID, string name, byte credit, bool require)

{

return dbCourse.insertCourseTable(studentID, name, credit,

require);

}

public static bool deleteCourse(string studentID, string name)

{

return dbCourse.deleteCourseTable(studentID, name);

}

public static bool updateCourse(string studentID, string name, string newName, byte credit, bool require)

{

return dbCourse.updateCourseTable(studentID, name, newName, credit, require);

}

}

}

# Criteria.cs (new)

using System;

using System.Collections.Generic; using System.Linq;

using System.Text;

using SIS.Domain.Inventory;

namespace SIS.Domain.Manage

{

public class Criteria

{

private string \_rank; private byte \_maxYear; private byte \_numberCredit;

private byte \_numberQualifying; private byte \_pointPaper;

public string Rank

{

get

{

}

set

{

}

}

return \_rank;

\_rank = value;

public byte MaxYear

{

get

{

}

set

{

}

}

return \_maxYear;

\_maxYear = value;

public byte NumberCredit

{

get

{

}

set

{

}

}

return \_numberCredit;

\_numberCredit = value;

public byte NumberQualifying

{

get

{

}

set

{

}

}

return \_numberQualifying;

\_numberQualifying = value;

public byte PointPaper

{

get

{

}

set

{

}

}

return \_pointPaper;

\_pointPaper = value;

public Criteria()

{

Rank = "Ph.D"; MaxYear = 6;

NumberCredit = 35;

NumberQualifying = 3;

PointPaper = 5;

}

public Criteria(string rank, byte maxYear, byte numberCredit, byte numberQualifying, byte pointPaper)

{

Rank = rank; MaxYear = maxYear;

NumberCredit = numberCredit; NumberQualifying = numberQualifying; PointPaper = pointPaper;

}

public static Criteria getCriteriaByClass(string rank)

{

return dbCriteria.getCriteriaByClass(rank);

}

}

}

# ExtendInformation.cs

using System;

using System.Collections.Generic; using System.Linq;

using System.Text;

using SIS.Domain.Inventory;

namespace SIS.Domain.Manage

{

public class ExtendInformation

{

private string \_birthday; private string \_national; private string \_password;

private List<Qualifying> \_qualifications; private List<Course> \_courses;

private List<Paper> \_papers;

public string Birthday

{

get

{

}

set

{

}

}

return \_birthday;

\_birthday = value;

public string National

{

get

{

}

set

{

}

}

return \_national;

\_national = value;

public string Password

{

get

{

}

set

{

}

}

return \_password;

\_password = value;

public List<Qualifying> Qualifications

{

get

{

}

set

{

}

}

return \_qualifications;

\_qualifications = value;

public List<Course> Courses

{

get

{

}

return \_courses;

set

{

}

}

\_courses = value;

public List<Paper> Papers

{

get

{

}

set

{

}

}

return \_papers;

\_papers = value;

public ExtendInformation()

{

Birthday = ""; National = ""; Password = "";

Qualifications = new List<Qualifying>(); Courses = new List<Course>();

Papers = new List<Paper>();

}

public ExtendInformation(string birthday, string national, string password,

List<Qualifying> qualifications, List<Course> courses, List<Paper> papers)

{

Birthday = birthday; National = national; Password = password;

Qualifications = qualifications; Courses = courses;

Papers = papers;

}

public static ExtendInformation getExtendInformation(string studentID)

{

// Get from database

return dbStudent.getExtendInformation(studentID);

}

public static bool updatePersonalInformation(string studentID, string birthday, string national, string password)

{

// Update to database

return dbStudent.updatePersonalStudentTable(studentID, birthday, national, password);

}

public static bool insertQualifying(string studentID, string name)

{

// Insert to database

return dbQualifying.insertQualifyingTable(studentID, name);

}

public static bool deleteQualifying(string studentID, string name)

{

// Delete to database

return dbQualifying.deleteQualifyingTable(studentID, name);

}

public static bool updateQualifying(string studentID, string name, string newName)

{

// Update to database

return dbQualifying.updateQualifyingTable(studentID, name,

newName);

}

public static bool insertCourse(string studentID, string name, byte credit, bool require)

{

require);

}

// Insert to database

return dbCourse.insertCourseTable(studentID, name, credit,

public static bool deleteCourse(string studentID, string name)

{

// Delete from database

return dbCourse.deleteCourseTable(studentID, name);

}

public static bool updateCourse(string studentID, string name, string newName, byte credit, bool require)

{

// Update to database

return dbCourse.updateCourseTable(studentID, name, newName, credit, require);

}

public static bool insertPaper(string studentID, string title, string publisher, string year)

{

year);

}

// Insert to database

return dbPaper.insertPaperTable(studentID, title, publisher,

public static bool deletePaper(string studentID, string title)

{

// Delete from database

return dbPaper.deletePaperTable(studentID, title);

}

public static bool updatePaper(string studentID, string title, string newTitle, string publisher, string year)

{

// Update to database

return dbPaper.updatePaperTable(studentID, title, newTitle, publisher, year);

}

}

}

# Manager.cs

using System;

using System.Collections.Generic; using System.Linq;

using System.Text;

using SIS.Domain.Inventory;

namespace SIS.Domain.Manage

{

public class Manager

{

protected string \_userName; protected string \_password; private bool \_isAdvisor;

public string UserName

{

get

{

}

set

{

}

}

return \_userName;

\_userName = value;

public string Password

{

get

{

}

set

{

}

}

return \_password;

\_password = value;

public bool IsAdvisor

{

get

{

}

set

return \_isAdvisor;

{

\_isAdvisor = value;

}

}

public Manager()

{

UserName = ""; Password = ""; IsAdvisor = false;

}

public Manager(string userName, string password)

{

UserName = userName; Password = password;

}

public Manager(string userName, string password, bool isAdvisor)

{

UserName = userName; Password = password; IsAdvisor = isAdvisor;

}

public bool Login()

{

if (IsAdvisor)

{

this.Password);

}

// Check in Advisor Table

return dbAdvisor.isExistAdvisorTable(this.UserName,

// Check in Student Table

return dbStudent.isExistStudentTable(this.UserName, this.Password);

}

public bool Logout()

{

return true;

}

}

}

# Paper.cs

using System;

using System.Collections.Generic; using System.Linq;

using System.Text;

using SIS.Domain.Inventory; namespace SIS.Domain.Manage

{

public class Paper

{

private string \_title; private string \_publisher; private string \_year;

public string Title

{

get

{

}

set

{

}

}

return \_title;

\_title = value;

public string Publisher

{

get

{

}

set

{

}

}

return \_publisher;

\_publisher = value;

public string Year

{

get

{

}

set

{

}

}

return \_year;

\_year = value;

public Paper()

{

Title = ""; Publisher = ""; Year = "";

}

public Paper(string title, string publisher, string year)

{

Title = title; Publisher = publisher; Year = year;

}

public static bool insertPaper(string studentID, string title, string publisher, string year)

{

return dbPaper.insertPaperTable(studentID, title, publisher,

year);

}

public static bool deletePaper(string studentID, string title)

{

return dbPaper.deletePaperTable(studentID, title);

}

public static bool updatePaper(string studentID, string title, string newTitle, string publisher, string year)

{

return dbPaper.updatePaperTable(studentID, title, newTitle, publisher, year);

}

}

}

# PrimaryInformation.cs

using System;

using System.Collections.Generic; using System.Linq;

using System.Text;

using SIS.Domain.Inventory;

namespace SIS.Domain.Manage

{

public class PrimaryInformation

{

private string \_studentID; private string \_name; private string \_rank;

public string StudentID

{

get

{

}

set

{

}

}

return \_studentID;

\_studentID = value;

public string Name

{

get

{

return \_name;

}

set

{

}

}

\_name = value;

public string Rank

{

get

{

}

set

{

}

}

return \_rank;

\_rank = value;

public PrimaryInformation()

{

StudentID = ""; Name = "";

Rank = "";

}

public PrimaryInformation(string studentID, string name, string rank)

{

StudentID = studentID; Name = name;

Rank = rank;

}

public static List<PrimaryInformation> getPrimaryInformation()

{

// Get from database

return dbStudent.getPrimaryInformation();

}

public static PrimaryInformation getPrimaryInformation(string studentID)

{

// Get from database

return dbStudent.getPrimaryInformation(studentID);

}

public bool savePrimaryInformation()

{

Rank);

}

}

}

// Insert into database

return dbStudent.insertPrimaryStudentTable(StudentID, Name,

# Qualifying.cs

using System;

using System.Collections.Generic; using System.Linq;

using System.Text;

using SIS.Domain.Inventory;

namespace SIS.Domain.Manage

{

public class Qualifying

{

private string \_name;

public string Name

{

get

{

}

set

{

}

}

return \_name;

\_name = value;

public Qualifying()

{

Name = "";

}

public Qualifying(string name)

{

Name = name;

}

public static bool insertQualifying(string studentID, string name)

{

return dbQualifying.insertQualifyingTable(studentID, name);

}

public static bool deleteQualifying(string studentID, string name)

{

return dbQualifying.deleteQualifyingTable(studentID, name);

}

public static bool updateQualifying(string studentID, string name, string newName)

{

return dbQualifying.updateQualifyingTable(studentID, name,

newName);

}

}

}

# Student.cs (new)

using System;

using System.Collections.Generic; using System.Linq;

using System.Text;

using SIS.Domain.Inventory;

namespace SIS.Domain.Manage

{

public class Student : Manager

{

public Student()

: base()

{

}

public Student(string userName, string password)

: base(userName, password)

{

}

public bool enterPersonalInformation(string studentID, string birthday, string national, string password)

{

return ExtendInformation.updatePersonalInformation(studentID, birthday, national, password);

}

public bool enterQualifying(string studentID, string name)

{

return ExtendInformation.insertQualifying(studentID, name);

}

public bool deleteQualifying(string studentID, string name)

{

return ExtendInformation.deleteQualifying(studentID, name);

}

newName)

public bool updateQualifying(string studentID, string name, string

{

return ExtendInformation.updateQualifying(studentID, name,

newName);

}

public List<Qualifying> getQualifying(string studentID)

{

return dbQualifying.getQualifyingTable(studentID);

}

public bool enterCourse(string studentID, string name, byte credit, bool require)

{

return ExtendInformation.insertCourse(studentID, name, credit,

require);

}

public bool deleteCourse(string studentID, string name)

{

return ExtendInformation.deleteCourse(studentID, name);

}

public bool updateCourse(string studentID, string name, string newName, byte credit, bool require)

{

return ExtendInformation.updateCourse(studentID, name, newName, credit, require);

}

public List<Course> getCourse(string studentID)

{

return dbCourse.getCourseTable(studentID);

}

// new method with parmater point

public bool enterPaper(string studentID, string title, string publisher, string year, byte point)

{

return ExtendInformation.insertPaper(studentID, title, publisher, year, point);

}

public bool deletePaper(string studentID, string title)

{

return ExtendInformation.deletePaper(studentID, title);

}

// new method with parmater point

public bool updatePaper(string studentID, string title, string newTitle, string publisher, string year, byte point)

{

return ExtendInformation.updatePaper(studentID, title, newTitle, publisher, year, point);

}

public List<Paper> getPaper(string studentID)

{

return dbPaper.getPaperTable(studentID);

}

public ExtendInformation showExtendInformation(string studentID)

{

// Get extend information from database

return ExtendInformation.getExtendInformation(studentID);

}

}

}

# StudentInformation.cs

using System;

using System.Collections.Generic; using System.Linq;

using System.Text;

using SIS.Domain.Inventory;

namespace SIS.Domain.Manage

{

public class StudentInformation

{

private PrimaryInformation \_priInformation; private ExtendInformation \_extInformation;

public string StudentID

{

get

{

}

set

{

}

}

return \_priInformation.StudentID;

\_priInformation.StudentID = value;

public PrimaryInformation PriInformation

{

get

{

}

set

{

}

}

return \_priInformation;

\_priInformation = value;

public ExtendInformation ExtInformation

{

get

{

}

set

{

}

}

return \_extInformation;

\_extInformation = value;

public StudentInformation()

{

PriInformation = new PrimaryInformation(); ExtInformation = new ExtendInformation();

}

public StudentInformation(PrimaryInformation priInformation, ExtendInformation extInformation)

{

PriInformation = priInformation; ExtInformation = extInformation;

}

public static StudentInformation getStudentInformation(string studentID)

{

PrimaryInformation \_priInformation = PrimaryInformation.getPrimaryInformation(studentID);

ExtendInformation \_extInformation = ExtendInformation.getExtendInformation(studentID);

if ((\_priInformation == null) || (\_extInformation == null)) return null;

return new StudentInformation(\_priInformation, \_extInformation);

}

}

}

# dbAdvisor.cs

using System;

using System.Collections.Generic; using System.Linq;

using System.Text; using System.Data;

using SIS.TechnicalServices.Persistence;

namespace SIS.Domain.Inventory

{

public class dbAdvisor : PersistenceSQLServer

{

public static DataTable getAdvisorTable()

{

string strSQL = "Select \* From Advisor"; DataTable dtAdvisor = querySelect(strSQL);

return dtAdvisor;

}

public static DataTable findAdvisorTable(string userName, string password)

{

string strSQL = "Select \* From Advisor Where UserName='" + userName + "' and Password='" + password + "';";

DataTable dtAdvisor = querySelect(strSQL);

return dtAdvisor;

}

public static bool isExistAdvisorTable(string userName, string password)

{

DataTable dtAdvisor = findAdvisorTable(userName, password);

return (dtAdvisor.Rows.Count > 0);

}

}

}

# dbCourse.cs

using System;

using System.Collections.Generic; using System.Linq;

using System.Text; using System.Data;

using SIS.TechnicalServices.Persistence; using SIS.Domain.Manage;

namespace SIS.Domain.Inventory

{

public class dbCourse : PersistenceSQLServer

{

public static List<Course> getCourseTable(string studentID)

{

string strSQL = "Select \* From Course Where StudentID = '" + studentID + "';";

DataTable dtCourse = querySelect(strSQL); List<Course> \_courses = new List<Course>(); for (int i = 0; i < dtCourse.Rows.Count; i++)

{

string \_name = dtCourse.Rows[i]["Name"].ToString(); byte \_credit =

byte.Parse(dtCourse.Rows[i]["Credit"].ToString());

bool \_require = bool.Parse(dtCourse.Rows[i]["Require"].ToString());

\_courses.Add(new Course(\_name, \_credit, \_require));

}

return \_courses;

}

public static bool insertCourseTable(string studentID, string name, byte credit, bool require)

{

string strSQL = "Insert Into Course(StudentID, Name, Credit, Require) Values('" + studentID + "', '" + name + "', " + credit + "," + (require == true ? 1 : 0) + ");";

return queryUpdate(strSQL);

}

public static bool deleteCourseTable(string studentID, string name)

{

string strSQL = "Delete From Course Where StudentID = '" + studentID + "' and Name = '" + name + "';";

return queryUpdate(strSQL);

}

public static bool updateCourseTable(string studentID, string name, string newName, byte credit, bool require)

{

string strSQL = "Update Course Set Name = '" + newName + "', Credit = " + credit + ", Require = " + (require == true ? 1 : 0) + " Where StudentID = '" + studentID + "' and Name = '" + name + "';";

return queryUpdate(strSQL);

}

}

}

# dbCriteria.cs (new)

using System;

using System.Collections.Generic; using System.Linq;

using System.Text; using System.Data;

using SIS.TechnicalServices.Persistence; using SIS.Domain.Manage;

namespace SIS.Domain.Inventory

{

public class dbCriteria : PersistenceSQLServer

{

public static Criteria getCriteriaByClass(string rank)

{

"';";

string strSQL = "Select \* From Criteria Where Class = '" + rank +

DataTable dtCriteria = querySelect(strSQL); Criteria \_criteria = new Criteria(); if(dtCriteria.Rows.Count>0)

{

string \_rank = dtCriteria.Rows[0]["Class"].ToString(); byte \_maxYear =

byte.Parse(dtCriteria.Rows[0]["MaxYear"].ToString()); byte \_numberCredit =

byte.Parse(dtCriteria.Rows[0]["NumberCredit"].ToString()); byte \_numberQualifying =

byte.Parse(dtCriteria.Rows[0]["NumberQualifying"].ToString()); byte \_pointPaper =

byte.Parse(dtCriteria.Rows[0]["PointPaper"].ToString());

return new Criteria(\_rank, \_maxYear, \_numberCredit,

\_numberQualifying, \_pointPaper);

}

return null;

}

}

}

# dbPaper.cs

using System;

using System.Collections.Generic; using System.Linq;

using System.Text; using System.Data;

using SIS.TechnicalServices.Persistence; using SIS.Domain.Manage;

namespace SIS.Domain.Inventory

{

public class dbPaper : PersistenceSQLServer

{

public static List<Paper> getPaperTable(string studentID)

{

string strSQL = "Select \* From Paper Where StudentID = '" + studentID + "';";

DataTable dtPaper = querySelect(strSQL); List<Paper> \_papers = new List<Paper>();

for (int i = 0; i < dtPaper.Rows.Count; i++)

{

string \_title = dtPaper.Rows[i]["Title"].ToString();

string \_publisher = dtPaper.Rows[i]["Publisher"].ToString(); string \_year = dtPaper.Rows[i]["Year"].ToString();

\_papers.Add(new Paper(\_title, \_publisher, \_year));

}

return \_papers;

}

public static bool insertPaperTable(string studentID, string title, string publisher, string year)

{

string strSQL = "Insert Into Paper(StudentID, Title, Publisher, Year) Values('" + studentID + "', '" + title + "', '" + publisher + "', '" + year + "');";

return queryUpdate(strSQL);

}

public static bool deletePaperTable(string studentID, string title)

{

string strSQL = "Delete Paper Where StudentID = '" + studentID + "' and Title = '" + title + "';";

return queryUpdate(strSQL);

}

public static bool updatePaperTable(string studentID, string title, string newTitle, string publisher, string year)

{

string strSQL = "Update Paper Set Title = '" + newTitle + "', Publisher = '" + publisher + "', Year = '" + year + "' Where StudentID = '" + studentID + "' and Title = '" + title + "';";

return queryUpdate(strSQL);

}

}

}

# dbQualifying.cs

using System;

using System.Collections.Generic; using System.Linq;

using System.Text; using System.Data;

using SIS.TechnicalServices.Persistence; using SIS.Domain.Manage;

namespace SIS.Domain.Inventory

{

public class dbQualifying : PersistenceSQLServer

{

public static List<Qualifying> getQualifyingTable(string studentID)

{

string strSQL = "Select \* From Qualifying Where StudentID = '" + studentID + "';";

DataTable dtQualifying = querySelect(strSQL); List<Qualifying> \_quanlifications = new List<Qualifying>(); for (int i = 0; i < dtQualifying.Rows.Count; i++)

{

\_quanlifications.Add(new Qualifying(dtQualifying.Rows[i]["Name"].ToString()));

}

return \_quanlifications;

}

name)

public static bool insertQualifyingTable(string studentID, string

{

string strSQL = "Insert Into Qualifying(StudentID, Name)

Values('" + studentID + "', '" + name + "');";

return queryUpdate(strSQL);

}

name)

public static bool deleteQualifyingTable(string studentID, string

{

string strSQL = "Delete From Qualifying Where StudentID = '" +

studentID + "' and Name ='" + name + "';";

return queryUpdate(strSQL);

}

public static bool updateQualifyingTable(string studentID, string name, string newName)

{

string strSQL = "Update Qualifying Set Name = '" + newName + "' Where StudentID = '" + studentID + "' and Name = '" + name + "';";

return queryUpdate(strSQL);

}

}

}

# dbStudent.cs

using System;

using System.Collections.Generic; using System.Linq;

using System.Text; using System.Data;

using SIS.TechnicalServices.Persistence; using SIS.Domain.Manage;

namespace SIS.Domain.Inventory

{

public class dbStudent : PersistenceSQLServer

{

public static DataTable getStudentTable()

{

string strSQL = "Select \* From Student;"; DataTable dtStudent = querySelect(strSQL);

return dtStudent;

}

public static DataTable getPrimaryStudentTable()

{

string strSQL = "Select StudentID, Name, Class From Student;"; DataTable dtStudent = querySelect(strSQL);

return dtStudent;

}

public static PrimaryInformation getPrimaryInformation(string studentID)

{

DataTable dtStudent = findStudentTableByStudentID(studentID); if (dtStudent.Rows.Count > 0)

{

string \_studentID = dtStudent.Rows[0]["StudentID"].ToString();

string \_name = dtStudent.Rows[0]["Name"].ToString(); string \_class = dtStudent.Rows[0]["Class"].ToString();

return new PrimaryInformation(\_studentID, \_name, \_class);

}

return null;

}

public static List<PrimaryInformation> getPrimaryInformation()

{

DataTable dtStudent = getPrimaryStudentTable(); List<PrimaryInformation> \_priInformations = new

List<PrimaryInformation>();

for (int i=0;i< dtStudent.Rows.Count; i++)

{

string \_studentID = dtStudent.Rows[i]["StudentID"].ToString();

string \_name = dtStudent.Rows[i]["Name"].ToString(); string \_class = dtStudent.Rows[i]["Class"].ToString();

\_priInformations.Add(new PrimaryInformation(\_studentID,

\_name, \_class));

}

return \_priInformations;

}

public static ExtendInformation getExtendInformation(string studentID)

{

DataTable dtStudent = findStudentTableByStudentID(studentID); if (dtStudent.Rows.Count > 0)

{

string \_birthday = dtStudent.Rows[0]["Birthday"].ToString(); string \_national = dtStudent.Rows[0]["National"].ToString(); string \_password = dtStudent.Rows[0]["Password"].ToString();

List<Qualifying> \_qualifications = dbQualifying.getQualifyingTable(studentID);

List<Course> \_courses = dbCourse.getCourseTable(studentID); List<Paper> \_papers = dbPaper.getPaperTable(studentID);

return new ExtendInformation(\_birthday, \_national, \_password,

\_qualifications, \_courses, \_papers);

}

return null;

}

public static List<string> getStudentIDTable()

{

string strSQL = "Select StudentID From Student;"; DataTable dtStudent = querySelect(strSQL); List<string> \_listStudentID = new List<string>(); for (int i = 0; i < dtStudent.Rows.Count; i++)

{

\_listStudentID.Add(dtStudent.Rows[i]["StudentID"].ToString());

}

return \_listStudentID;

}

public static DataTable findStudentTableByStudentID(string studentID)

{

string strSQL = "Select \* From Student Where StudentID='" + studentID + "';";

DataTable dtStudent = querySelect(strSQL);

return dtStudent;

}

public static DataTable findStudentTableByName(string name)

{

"';";

string strSQL = "Select \* From Student Where Name='" + name + DataTable dtStudent = querySelect(strSQL);

return dtStudent;

}

public static DataTable findStudentTable(string userName, string password)

{

string strSQL = "Select \* From Student Where StudentID='" + userName + "' and Password='" + password + "';";

DataTable dtStudent = querySelect(strSQL);

return dtStudent;

}

public static bool isExistStudentTable(string userName, string password)

{

DataTable dtStudent = findStudentTable(userName, password);

return (dtStudent.Rows.Count > 0);

}

public static bool insertPrimaryStudentTable(string studentID, string name, string rank)

{

string strSQL = "Insert Into Student(StudentID, Name, Class) Values('" + studentID + "', '" + name + "', '" + rank + "');";

return queryUpdate(strSQL);

}

public static bool deletePrimaryStudentTable(string studentID)

{

string strSQL = "Delete From Student Where StudentID = '" + studentID + "';";

return queryUpdate(strSQL);

}

public static bool updatePrimaryStudentTable(string studentID, string name, string rank)

{

string strSQL = "Update Student Set Name = '" + name + "', Class

= '" + rank + "' Where StudentID = '" + studentID + "';";

return queryUpdate(strSQL);

}

public static bool updatePersonalStudentTable(string studentID, string birthday, string national, string password)

{

string strSQL = "Update Student Set Birthday = '" + birthday + "', [National] = '" + national + "', Password = '" + password + "' Where StudentID = '" + studentID + "';";

return queryUpdate(strSQL);

}

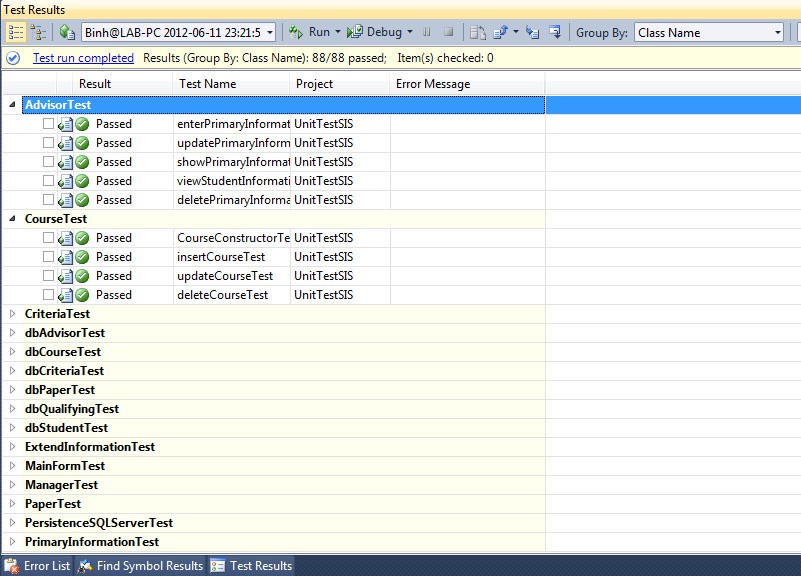
}

}

# Unit Testing

Use the Unit Testing Tool of Visual Studio 2010.

# Snapshots of testing result



# Unit Testing Code Listing

# AdvisorTest.cs

using SIS.Domain.Manage;

using Microsoft.VisualStudio.TestTools.UnitTesting; using System;

namespace UnitTestSIS

{

/// <summary>

///This is a test class for AdvisorTest and is intended

///to contain all AdvisorTest Unit Tests

///</summary> [TestClass()]

public class AdvisorTest

{

/// <summary>

///A test for enterPrimaryInformation

///</summary> [TestMethod()]

public void enterPrimaryInformationTest()

{

Advisor target = new Advisor(); string studentID = "100599000"; string name = "TestName"; string rank = "Master";

bool expected = true; bool actual;

actual = target.enterPrimaryInformation(studentID, name, rank); Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for updatePrimaryInformation

///</summary> [TestMethod()]

public void updatePrimaryInformationTest()

{

Advisor target = new Advisor(); string studentID = "100599000"; string name = "newTestName"; string rank = "Master";

bool expected = true; bool actual;

actual = target.updatePrimaryInformation(studentID, name, rank); Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for showPrimaryInformation

///</summary> [TestMethod()]

public void showPrimaryInformationTest()

{

Advisor target = new Advisor(); string studentID = "100599000";

PrimaryInformation expected = new PrimaryInformation("100599000", "newTestName", "Master");

PrimaryInformation actual;

actual = target.showPrimaryInformation(studentID); Assert.AreEqual(expected.StudentID, actual.StudentID.Trim()); Assert.AreEqual(expected.Name, actual.Name.Trim()); Assert.AreEqual(expected.Rank, actual.Rank.Trim());

}

/// <summary>

///A test for viewStudentInformation

///</summary> [TestMethod()]

public void viewStudentInformationTest()

{

Advisor target = new Advisor(); string studentID = "100599000";

StudentInformation actual;

actual = target.viewStudentInformation(studentID); Assert.AreEqual(studentID, actual.StudentID.Trim());

}

/// <summary>

///A test for deletePrimaryInformation

///</summary> [TestMethod()]

public void deletePrimaryInformationTest()

{

Advisor target = new Advisor(); string studentID = "100599000";

bool expected = true; bool actual;

actual = target.deletePrimaryInformation(studentID); Assert.AreEqual(expected, actual);

}

}

}

# CourseTest.cs

using SIS.Domain.Manage;

using Microsoft.VisualStudio.TestTools.UnitTesting; using System;

namespace UnitTestSIS

{

/// <summary>

///This is a test class for CourseTest and is intended

///to contain all CourseTest Unit Tests

///</summary> [TestClass()]

public class CourseTest

{

/// <summary>

///A test for Course Constructor

///</summary> [TestMethod()]

public void CourseConstructorTest()

{

string name = "TestName"; byte credit = 0;

bool require = false;

Course target = new Course(name, credit, require); Assert.AreEqual(name, target.Name); Assert.AreEqual(credit, target.Credit); Assert.AreEqual(require, target.Require);

}

/// <summary>

///A test for insertCourse

///</summary>

[TestMethod()]

public void insertCourseTest()

{

string studentID = "100599008"; string name = "TestName";

byte credit = 3;

bool require = false; bool expected = true;

bool actual;

actual = Course.insertCourse(studentID, name, credit, require); Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for updateCourse

///</summary> [TestMethod()]

public void updateCourseTest()

{

string studentID = "100599008"; string name = "TestName"; string newName = "newTestName"; byte credit = 3;

bool require = true;

require);

}

bool expected = true; bool actual;

actual = Course.updateCourse(studentID, name, newName, credit, Assert.AreEqual(expected, actual);

/// <summary>

///A test for deleteCourse

///</summary> [TestMethod()]

public void deleteCourseTest()

{

string studentID = "100599008"; string name = "newTestName";

bool expected = true; bool actual;

actual = Course.deleteCourse(studentID, name); Assert.AreEqual(expected, actual);

}

}

}

# CriteriaTest.cs

using SIS.Domain.Manage;

using Microsoft.VisualStudio.TestTools.UnitTesting; using System;

namespace UnitTestSIS

{

/// <summary>

///This is a test class for CriteriaTest and is intended

///to contain all CriteriaTest Unit Tests

///</summary> [TestClass()]

public class CriteriaTest

{

/// <summary>

///A test for Criteria Constructor

///</summary> [TestMethod()]

public void CriteriaConstructorTest()

{

string rank = "Ph.D"; byte maxYear = 6;

byte numberCredit = 35; byte numberQualifying = 3; byte pointPaper = 5;

Criteria target = new Criteria(rank, maxYear, numberCredit, numberQualifying, pointPaper);

Assert.AreEqual(rank, target.Rank); Assert.AreEqual(maxYear, target.MaxYear); Assert.AreEqual(numberCredit, target.NumberCredit); Assert.AreEqual(numberQualifying, target.NumberQualifying); Assert.AreEqual(pointPaper, target.PointPaper);

}

/// <summary>

///A test for getCriteriaByClass

///</summary> [TestMethod()]

public void getCriteriaByClassTest()

{

string rank = "Ph.D";

Criteria expected = new Criteria("Ph.D", 6, 35, 3, 5); Criteria actual;

actual = Criteria.getCriteriaByClass(rank); Assert.AreEqual(expected.Rank, actual.Rank); Assert.AreEqual(expected.MaxYear, actual.MaxYear); Assert.AreEqual(expected.NumberCredit, actual.NumberCredit); Assert.AreEqual(expected.NumberQualifying,

actual.NumberQualifying);

Assert.AreEqual(expected.PointPaper, actual.PointPaper);

}

}

}

# dbAdvisorTest.cs

using SIS.Domain.Inventory;

using Microsoft.VisualStudio.TestTools.UnitTesting; using System;

using System.Data;

namespace UnitTestSIS

{

/// <summary>

///This is a test class for dbAdvisorTest and is intended

///to contain all dbAdvisorTest Unit Tests

///</summary> [TestClass()]

public class dbAdvisorTest

{

/// <summary>

///A test for getAdvisorTable

///</summary> [TestMethod()]

public void getAdvisorTableTest()

{

//DataTable expected = null; // TODO: Initialize to an appropriate value

DataTable actual;

actual = dbAdvisor.getAdvisorTable(); Assert.AreEqual(1, actual.Rows.Count);

}

/// <summary>

///A test for findAdvisorTable

///</summary> [TestMethod()]

public void findAdvisorTableTest()

{

string userName = "wkchen"; string password = "12345";

//DataTable expected = null; // TODO: Initialize to an appropriate value

DataTable actual;

actual = dbAdvisor.findAdvisorTable(userName, password); Assert.AreEqual(1, actual.Rows.Count);

}

/// <summary>

///A test for isExistAdvisorTable

///</summary> [TestMethod()]

public void isExistAdvisorTableTest()

{

string userName = "wkchen"; string password = "12345"; bool expected = true;

bool actual;

actual = dbAdvisor.isExistAdvisorTable(userName, password); Assert.AreEqual(expected, actual);

}

}

}

# dbCourseTest.cs

using SIS.Domain.Inventory;

using Microsoft.VisualStudio.TestTools.UnitTesting; using System;

using SIS.Domain.Manage;

using System.Collections.Generic;

namespace UnitTestSIS

{

/// <summary>

///This is a test class for dbCourseTest and is intended

///to contain all dbCourseTest Unit Tests

///</summary> [TestClass()]

public class dbCourseTest

{

/// <summary>

///A test for insertCourseTable

///</summary> [TestMethod()]

public void insertCourseTableTest()

{

string studentID = "100599008"; string name = "TestName";

byte credit = 3;

bool require = false; bool expected = true;

require);

}

bool actual;

actual = dbCourse.insertCourseTable(studentID, name, credit, Assert.AreEqual(expected, actual);

/// <summary>

///A test for updateCourseTable

///</summary> [TestMethod()]

public void updateCourseTableTest()

{

string studentID = "100599008"; string name = "TestName"; string newName = "newTestName"; byte credit = 3;

bool require = true;

bool expected = true; bool actual;

actual = dbCourse.updateCourseTable(studentID, name, newName, credit, require);

Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for deleteCourseTable

///</summary> [TestMethod()]

public void deleteCourseTableTest()

{

string studentID = "100599008"; string name = "newTestName";

bool expected = true; bool actual;

actual = dbCourse.deleteCourseTable(studentID, name); Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for getCourseTable

///</summary> [TestMethod()]

public void getCourseTableTest()

{

string studentID = "100599008"; List<Course> actual;

actual = dbCourse.getCourseTable(studentID); Assert.AreEqual(5, actual.Count);

}

}

}

# dbCriteriaTest.cs

using SIS.Domain.Inventory;

using Microsoft.VisualStudio.TestTools.UnitTesting; using System;

using SIS.Domain.Manage;

namespace UnitTestSIS

{

/// <summary>

///This is a test class for dbCriteriaTest and is intended

///to contain all dbCriteriaTest Unit Tests

///</summary> [TestClass()]

public class dbCriteriaTest

{

/// <summary>

///A test for getCriteriaByClass

///</summary> [TestMethod()]

public void getCriteriaByClassTest()

{

string rank = "Ph.D";

Criteria expected = new Criteria("Ph.D", 6, 35, 3, 5); Criteria actual;

actual = dbCriteria.getCriteriaByClass(rank); Assert.AreEqual(expected.Rank, actual.Rank); Assert.AreEqual(expected.MaxYear, actual.MaxYear); Assert.AreEqual(expected.NumberCredit, actual.NumberCredit); Assert.AreEqual(expected.NumberQualifying,

actual.NumberQualifying);

Assert.AreEqual(expected.PointPaper, actual.PointPaper);

}

}

}

# dbPaperTest.cs

using SIS.Domain.Inventory;

using Microsoft.VisualStudio.TestTools.UnitTesting; using System;

using SIS.Domain.Manage;

using System.Collections.Generic;

namespace UnitTestSIS

{

/// <summary>

///This is a test class for dbPaperTest and is intended

///to contain all dbPaperTest Unit Tests

///</summary> [TestClass()]

public class dbPaperTest

{

/// <summary>

///A test for insertPaperTable

///</summary> [TestMethod()]

public void insertPaperTableTest()

{

string studentID = "100599008"; string title = "TestName";

string publisher = "TestPublisher"; string year = "Nov, 2012";

year);

}

bool expected = true; bool actual;

actual = dbPaper.insertPaperTable(studentID, title, publisher, Assert.AreEqual(expected, actual);

/// <summary>

///A test for updatePaperTable

///</summary> [TestMethod()]

public void updatePaperTableTest()

{

string studentID = "100599008"; string title = "TestName"; string newTitle = "newTestName";

string publisher = "TestPublisher"; string year = "Nov, 2012";

bool expected = true; bool actual;

actual = dbPaper.updatePaperTable(studentID, title, newTitle, publisher, year);

Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for deletePaperTable

///</summary> [TestMethod()]

public void deletePaperTableTest()

{

string studentID = "100599008"; string title = "newTestName";

bool expected = true; bool actual;

actual = dbPaper.deletePaperTable(studentID, title); Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for getPaperTable

///</summary> [TestMethod()]

public void getPaperTableTest()

{

string studentID = "100599008"; List<Paper> actual;

actual = dbPaper.getPaperTable(studentID); Assert.AreEqual(1, actual.Count);

}

}

}

# dbQualifyingTest.cs

using SIS.Domain.Inventory;

using Microsoft.VisualStudio.TestTools.UnitTesting; using System;

using SIS.Domain.Manage;

using System.Collections.Generic;

namespace UnitTestSIS

{

/// <summary>

///This is a test class for dbQualifyingTest and is intended

///to contain all dbQualifyingTest Unit Tests

///</summary> [TestClass()]

public class dbQualifyingTest

{

/// <summary>

///A test for insertQualifyingTable

///</summary> [TestMethod()]

public void insertQualifyingTableTest()

{

string studentID = "100599008"; string name = "TestName";

bool expected = true; bool actual;

actual = dbQualifying.insertQualifyingTable(studentID, name); Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for updateQualifyingTable

///</summary> [TestMethod()]

public void updateQualifyingTableTest()

{

newName);

}

string studentID = "100599008"; string name = "TestName"; string newName = "newTestName"; bool expected = true;

bool actual;

actual = dbQualifying.updateQualifyingTable(studentID, name, Assert.AreEqual(expected, actual);

/// <summary>

///A test for deleteQualifyingTable

///</summary> [TestMethod()]

public void deleteQualifyingTableTest()

{

string studentID = "100599008"; string name = "newTestName"; bool expected = true;

bool actual;

actual = dbQualifying.deleteQualifyingTable(studentID, name); Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for getQualifyingTable

///</summary> [TestMethod()]

public void getQualifyingTableTest()

{

string studentID = "100599008";

List<Qualifying> expected = new List<Qualifying>(); expected.Add(new Qualifying("Digital Signal Processing")); expected.Add(new Qualifying("Random Signals and System"));

List<Qualifying> actual;

actual = dbQualifying.getQualifyingTable(studentID); Assert.AreEqual(2, actual.Count); Assert.AreEqual(expected[0].Name, actual[0].Name.Trim()); Assert.AreEqual(expected[1].Name, actual[1].Name.Trim());

}

}

}

# dbStudentTest.cs

using SIS.Domain.Inventory;

using Microsoft.VisualStudio.TestTools.UnitTesting; using System;

using System.Data;

using SIS.Domain.Manage;

using System.Collections.Generic;

namespace UnitTestSIS

{

/// <summary>

///This is a test class for dbStudentTest and is intended

///to contain all dbStudentTest Unit Tests

///</summary> [TestClass()]

public class dbStudentTest

{

/// <summary>

///A test for insertPrimaryStudentTable

///</summary> [TestMethod()]

public void insertPrimaryStudentTableTest()

{

rank);

}

string studentID = "100599000"; string name = "TestName"; string rank = "TestRank";

bool expected = true; bool actual;

actual = dbStudent.insertPrimaryStudentTable(studentID, name, Assert.AreEqual(expected, actual);

/// <summary>

///A test for updatePrimaryStudentTable

///</summary> [TestMethod()]

public void updatePrimaryStudentTableTest()

{

rank);

}

string studentID = "100599000"; string name = "TestName updated"; string rank = "TestRank updated"; bool expected = true;

bool actual;

actual = dbStudent.updatePrimaryStudentTable(studentID, name, Assert.AreEqual(expected, actual);

/// <summary>

///A test for updatePersonalStudentTable

///</summary> [TestMethod()]

public void updatePersonalStudentTableTest()

{

string studentID = "100599000"; string birthday = "1982/09/17"; string national = "Taiwan"; string password = "test";

bool expected = true; bool actual;

actual = dbStudent.updatePersonalStudentTable(studentID,

birthday, national, password);

Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for deletePrimaryStudentTable

///</summary> [TestMethod()]

public void deletePrimaryStudentTableTest()

{

string studentID = "100599000"; bool expected = true;

bool actual;

actual = dbStudent.deletePrimaryStudentTable(studentID); Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for findStudentTable

///</summary> [TestMethod()]

public void findStudentTableTest()

{

string userName = "100599008"; string password = "12345";

//DataTable expected = null; // TODO: Initialize to an appropriate value

DataTable actual;

actual = dbStudent.findStudentTable(userName, password); Assert.AreEqual(1, actual.Rows.Count);

}

/// <summary>

///A test for findStudentTableByName

///</summary> [TestMethod()]

public void findStudentTableByNameTest()

{

string name = "Vo Phuong Binh";

//DataTable expected = null; // TODO: Initialize to an appropriate value

DataTable actual;

actual = dbStudent.findStudentTableByName(name); Assert.AreEqual(1, actual.Rows.Count);

}

/// <summary>

///A test for findStudentTableByStudentID

///</summary> [TestMethod()]

public void findStudentTableByStudentIDTest()

{

string studentID = "100599008"; DataTable actual;

actual = dbStudent.findStudentTableByStudentID(studentID); Assert.AreEqual(1, actual.Rows.Count);

}

/// <summary>

///A test for getExtendInformation

///</summary> [TestMethod()]

public void getExtendInformationTest()

{

string studentID = "100599009"; string \_birthday = "1981/09/17"; string \_national = "Korea"; string \_password = "test";

List<Qualifying> \_qualifications = new List<Qualifying>(); List<Course> \_courses = new List<Course>();

List<Paper> \_papers = new List<Paper>();

ExtendInformation expected = new ExtendInformation(\_birthday,

\_national, \_password, \_qualifications, \_courses, \_papers);

ExtendInformation actual;

actual = dbStudent.getExtendInformation(studentID); Assert.AreEqual(expected.Birthday, actual.Birthday.Trim()); Assert.AreEqual(expected.National, actual.National.Trim()); Assert.AreEqual(expected.Password, actual.Password.Trim()); Assert.AreEqual(expected.Qualifications.Count,

actual.Qualifications.Count);

Assert.AreEqual(expected.Courses.Count, actual.Courses.Count); Assert.AreEqual(expected.Papers.Count, actual.Papers.Count);

}

/// <summary>

///A test for getPrimaryInformation

///</summary> [TestMethod()]

public void getPrimaryInformationTest()

{

string studentID = "100599008";

PrimaryInformation expected = new PrimaryInformation("100599008", "Vo Phuong Binh", "Ph.D");

PrimaryInformation actual;

actual = dbStudent.getPrimaryInformation(studentID); Assert.AreEqual(expected.StudentID, actual.StudentID.Trim()); Assert.AreEqual(expected.Name, actual.Name.Trim()); Assert.AreEqual(expected.Rank, actual.Rank.Trim());

}

/// <summary>

///A test for getPrimaryInformation

///</summary> [TestMethod()]

public void getPrimaryInformationTest1()

{

List<PrimaryInformation> actual;

actual = dbStudent.getPrimaryInformation(); Assert.AreEqual(4, actual.Count);

}

/// <summary>

///A test for getPrimaryStudentTable

///</summary> [TestMethod()]

public void getPrimaryStudentTableTest()

{

DataTable actual;

actual = dbStudent.getPrimaryStudentTable(); Assert.AreEqual(4, actual.Rows.Count);

}

/// <summary>

///A test for getStudentIDTable

///</summary> [TestMethod()]

public void getStudentIDTableTest()

{

List<string> expected = new List<string>(); expected.Add("100599003"); expected.Add("100599004"); expected.Add("100599008"); expected.Add("100599009");

List<string> actual;

actual = dbStudent.getStudentIDTable(); Assert.AreEqual(4, actual.Count); Assert.AreEqual(expected[0], actual[0].Trim()); Assert.AreEqual(expected[1], actual[1].Trim()); Assert.AreEqual(expected[2], actual[2].Trim()); Assert.AreEqual(expected[3], actual[3].Trim());

}

/// <summary>

///A test for getStudentTable

///</summary> [TestMethod()]

public void getStudentTableTest()

{

DataTable actual;

actual = dbStudent.getStudentTable(); Assert.AreEqual(4, actual.Rows.Count);

}

/// <summary>

///A test for isExistStudentTable

///</summary> [TestMethod()]

public void isExistStudentTableTest()

{

string userName = "100599008"; string password = "12345"; bool expected = true;

bool actual;

actual = dbStudent.isExistStudentTable(userName, password); Assert.AreEqual(expected, actual);

}

}

}

# ExtendInformationTest.cs

using SIS.Domain.Manage;

using Microsoft.VisualStudio.TestTools.UnitTesting; using System;

using System.Collections.Generic;

namespace UnitTestSIS

{

/// <summary>

///This is a test class for ExtendInformationTest and is intended

///to contain all ExtendInformationTest Unit Tests

///</summary> [TestClass()]

public class ExtendInformationTest

{

/// <summary>

///A test for updatePersonalInformation

///</summary> [TestMethod()]

public void updatePersonalInformationTest()

{

string studentID = "100599009"; string birthday = "1981/09/17"; string national = "Korea"; string password = "test";

bool expected = true; bool actual;

actual = ExtendInformation.updatePersonalInformation(studentID, birthday, national, password);

Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for getExtendInformation

///</summary> [TestMethod()]

public void getExtendInformationTest()

{

string studentID = "100599009"; string \_birthday = "1981/09/17"; string \_national = "Korea"; string \_password = "test";

List<Qualifying> \_qualifications = new List<Qualifying>(); List<Course> \_courses = new List<Course>();

List<Paper> \_papers = new List<Paper>();

ExtendInformation expected = new ExtendInformation(\_birthday,

\_national, \_password, \_qualifications, \_courses, \_papers);

ExtendInformation actual;

actual = ExtendInformation.getExtendInformation(studentID); Assert.AreEqual(expected.Birthday, actual.Birthday.Trim());

Assert.AreEqual(expected.National, actual.National.Trim()); Assert.AreEqual(expected.Password, actual.Password.Trim()); Assert.AreEqual(expected.Qualifications.Count,

actual.Qualifications.Count);

Assert.AreEqual(expected.Courses.Count, actual.Courses.Count); Assert.AreEqual(expected.Papers.Count, actual.Papers.Count);

}

/// <summary>

///A test for insertCourse

///</summary> [TestMethod()]

public void insertCourseTest()

{

string studentID = "100599008"; string name = "TestName";

byte credit = 3;

bool require = false; bool expected = true;

require);

}

bool actual;

actual = ExtendInformation.insertCourse(studentID, name, credit, Assert.AreEqual(expected, actual);

/// <summary>

///A test for updateCourse

///</summary> [TestMethod()]

public void updateCourseTest()

{

string studentID = "100599008"; string name = "TestName"; string newName = "newTestName"; byte credit = 3;

bool require = true;

bool expected = true; bool actual;

actual = ExtendInformation.updateCourse(studentID, name, newName, credit, require);

Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for deleteCourse

///</summary> [TestMethod()]

public void deleteCourseTest()

{

string studentID = "100599008"; string name = "newTestName";

bool expected = true; bool actual;

actual = ExtendInformation.deleteCourse(studentID, name); Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for insertPaper

///</summary> [TestMethod()]

public void insertPaperTest()

{

string studentID = "100599008"; string title = "TestName";

string publisher = "TestPublisher"; string year = "Nov, 2012";

bool expected = true; bool actual;

actual = ExtendInformation.insertPaper(studentID, title, publisher, year);

Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for updatePaper

///</summary> [TestMethod()]

public void updatePaperTest()

{

string studentID = "100599008"; string title = "TestName"; string newTitle = "newTestName";

string publisher = "TestPublisher"; string year = "Nov, 2012";

bool expected = true; bool actual;

actual = ExtendInformation.updatePaper(studentID, title, newTitle, publisher, year);

Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for deletePaper

///</summary> [TestMethod()]

public void deletePaperTest()

{

string studentID = "100599008"; string title = "newTestName";

bool expected = true; bool actual;

actual = ExtendInformation.deletePaper(studentID, title); Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for insertQualifying

///</summary> [TestMethod()]

public void insertQualifyingTest()

{

string studentID = "100599008"; string name = "TestName";

bool expected = true; bool actual;

actual = ExtendInformation.insertQualifying(studentID, name); Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for updateQualifying

///</summary> [TestMethod()]

public void updateQualifyingTest()

{

newName);

}

string studentID = "100599008"; string name = "TestName"; string newName = "newTestName"; bool expected = true;

bool actual;

actual = ExtendInformation.updateQualifying(studentID, name, Assert.AreEqual(expected, actual);

/// <summary>

///A test for deleteQualifying

///</summary> [TestMethod()]

public void deleteQualifyingTest()

{

string studentID = "100599008"; string name = "newTestName"; bool expected = true;

bool actual;

actual = ExtendInformation.deleteQualifying(studentID, name); Assert.AreEqual(expected, actual);

}

}

}

# MainFormTest.cs

using SIS.UI.GUI;

using Microsoft.VisualStudio.TestTools.UnitTesting; using System;

using SIS.Domain.Manage;

namespace UnitTestSIS

{

/// <summary>

///This is a test class for MainFormTest and is intended

///to contain all MainFormTest Unit Tests

///</summary> [TestClass()]

public class MainFormTest

{

/// <summary>

///A test for setLoginResult

///</summary> [TestMethod()]

public void setLoginResultTest()

{

Manager manager = new Manager("wkchen", "12345"); MainForm.setLoginResult(manager); Assert.AreEqual(manager.UserName, MainForm.\_Manager.UserName); Assert.AreEqual(manager.Password, MainForm.\_Manager.Password);

}

/// <summary>

///A test for setConnectResult

///</summary> [TestMethod()]

public void setConnectResultTest()

{

bool isConnection = true; MainForm.setConnectResult(isConnection); Assert.AreEqual(isConnection, MainForm.\_IsConnection);

}

}

}

# ManagerTest.cs

using SIS.Domain.Manage;

using Microsoft.VisualStudio.TestTools.UnitTesting; using System;

namespace UnitTestSIS

{

/// <summary>

///This is a test class for ManagerTest and is intended

///to contain all ManagerTest Unit Tests

///</summary> [TestClass()]

public class ManagerTest

{

/// <summary>

///A test for Manager Constructor

///</summary> [TestMethod()]

public void ManagerConstructorTest()

{

string userName = "wkchen"; string password = "12345"; bool isAdvisor = true;

Manager target = new Manager(userName, password, isAdvisor); Assert.AreEqual(userName, target.UserName); Assert.AreEqual(password, target.Password); Assert.AreEqual(isAdvisor, target.IsAdvisor);

}

/// <summary>

///A test for Manager Constructor

///</summary> [TestMethod()]

public void ManagerConstructorTest1()

{

string userName = "wkchen"; string password = "12345";

Manager target = new Manager(userName, password); Assert.AreEqual(userName, target.UserName); Assert.AreEqual(password, target.Password);

}

/// <summary>

///A test for Login

///</summary> [TestMethod()]

public void LoginTest()

{

Manager target = new Manager("wkchen", "12345", true);

bool expected = true; bool actual;

actual = target.Login(); Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for Logout

///</summary> [TestMethod()]

public void LogoutTest()

{

Manager target = new Manager("wkchen", "12345", true);

bool expected = true; bool actual;

actual = target.Logout(); Assert.AreEqual(expected, actual);

}

}

}

# PaperTest.cs

using SIS.Domain.Manage;

using Microsoft.VisualStudio.TestTools.UnitTesting; using System;

namespace UnitTestSIS

{

/// <summary>

///This is a test class for PaperTest and is intended

///to contain all PaperTest Unit Tests

///</summary> [TestClass()]

public class PaperTest

{

/// <summary>

///A test for Paper Constructor

///</summary> [TestMethod()]

public void PaperConstructorTest()

{

string title = "TestName"; string publisher = "Test1"; string year = "Test2";

Paper target = new Paper(title, publisher, year); Assert.AreEqual(title, target.Title); Assert.AreEqual(publisher, target.Publisher); Assert.AreEqual(year, target.Year);

}

/// <summary>

///A test for insertPaper

///</summary> [TestMethod()]

public void insertPaperTest()

{

string studentID = "100599008"; string title = "TestName";

string publisher = "TestPublisher"; string year = "Nov, 2012";

bool expected = true; bool actual;

actual = Paper.insertPaper(studentID, title, publisher, year); Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for updatePaper

///</summary> [TestMethod()]

public void updatePaperTest()

{

string studentID = "100599008"; string title = "TestName"; string newTitle = "newTestName";

string publisher = "TestPublisher"; string year = "Nov, 2012";

year);

}

bool expected = true; bool actual;

actual = Paper.updatePaper(studentID, title, newTitle, publisher, Assert.AreEqual(expected, actual);

/// <summary>

///A test for deletePaper

///</summary>

[TestMethod()]

public void deletePaperTest()

{

string studentID = "100599008"; string title = "newTestName";

bool expected = true; bool actual;

actual = Paper.deletePaper(studentID, title); Assert.AreEqual(expected, actual);

}

}

}

# PersistenceSQLServerTest.cs

using SIS.TechnicalServices.Persistence;

using Microsoft.VisualStudio.TestTools.UnitTesting; using System;

using System.Data;

namespace UnitTestSIS

{

/// <summary>

///This is a test class for PersistenceSQLServerTest and is intended

///to contain all PersistenceSQLServerTest Unit Tests

///</summary> [TestClass()]

public class PersistenceSQLServerTest

{

/// <summary>

///A test for connectDatabase with parameters

///</summary> [TestMethod()]

public void connectDatabaseByAuthenticatedSQLServerTest()

{

string server = "(local)"; string database = "SIS"; string userID = "sa";

string password = "ooad2012"; bool expected = true;

bool actual; actual =

PersistenceSQLServer.connectDatabaseByAuthenticatedSQLServer(server, database, userID, password);

Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for connectDatabaseByAuthenticatedWindows

///</summary> [TestMethod()]

public void connectDatabaseByAuthenticatedWindowsTest()

{

string server = "(local)"; string database = "SIS";

bool expected = true; bool actual;

actual = PersistenceSQLServer.connectDatabaseByAuthenticatedWindows(server, database);

Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for closeConnectDatabase

///</summary> [TestMethod()]

public void closeConnectDatabaseTest()

{

bool expected = true; bool actual;

actual = PersistenceSQLServer.closeConnectDatabase(); Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for querySelect

///</summary> [TestMethod()]

public void querySelectTest()

{

string strSQL = "Select StudentID, Name, Birthday From Student Where StudentID='100599008';";

DataTable actual;

actual = PersistenceSQLServer.querySelect(strSQL); Assert.AreEqual(1, actual.Rows.Count); Assert.AreEqual("100599008",

actual.Rows[0]["StudentID"].ToString().Trim()); Assert.AreEqual("Vo Phuong Binh",

actual.Rows[0]["Name"].ToString().Trim()); Assert.AreEqual("1984/09/17",

actual.Rows[0]["Birthday"].ToString().Trim());

}

/// <summary>

///A test for queryUpdate

///</summary> [TestMethod()]

public void queryUpdateTest()

{

string strSQL = "Insert Into Student(StudentID, Name, Class) Values('100599000', 'TestName', 'Master');";

bool expected = true; bool actual;

actual = PersistenceSQLServer.queryUpdate(strSQL); Assert.AreEqual(expected, actual);

strSQL = "Delete From Student Where StudentID = '100599000';"; expected = true;

actual = PersistenceSQLServer.queryUpdate(strSQL); Assert.AreEqual(expected, actual);

}

}

}

# PrimaryInformationTest.cs

using SIS.Domain.Manage;

using Microsoft.VisualStudio.TestTools.UnitTesting; using System;

using System.Collections.Generic;

namespace UnitTestSIS

{

/// <summary>

///This is a test class for PrimaryInformationTest and is intended

///to contain all PrimaryInformationTest Unit Tests

///</summary> [TestClass()]

public class PrimaryInformationTest

{

/// <summary>

///A test for PrimaryInformation Constructor

///</summary> [TestMethod()]

public void PrimaryInformationConstructorTest()

{

name, rank);

}

string studentID = "test1"; string name = "test2"; string rank = "test3";

PrimaryInformation target = new PrimaryInformation(studentID,

Assert.AreEqual(studentID, target.StudentID); Assert.AreEqual(name, target.Name); Assert.AreEqual(rank, target.Rank);

/// <summary>

///A test for savePrimaryInformation

///</summary> [TestMethod()]

public void savePrimaryInformationTest()

{

name, rank);

string studentID = "100599008"; string name = "TestName"; string rank = "TestRank";

PrimaryInformation target = new PrimaryInformation(studentID,

bool expected = false; bool actual;

actual = target.savePrimaryInformation(); Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for getPrimaryInformation

///</summary> [TestMethod()]

public void getPrimaryInformationTest()

{

string studentID = "100599008";

PrimaryInformation expected = new PrimaryInformation("100599008", "Vo Phuong Binh", "Ph.D");

PrimaryInformation actual;

actual = PrimaryInformation.getPrimaryInformation(studentID); Assert.AreEqual(expected.StudentID, actual.StudentID.Trim()); Assert.AreEqual(expected.Name, actual.Name.Trim()); Assert.AreEqual(expected.Rank, actual.Rank.Trim());

}

/// <summary>

///A test for getPrimaryInformation

///</summary> [TestMethod()]

public void getPrimaryInformationTest1()

{

List<PrimaryInformation> actual;

actual = PrimaryInformation.getPrimaryInformation(); Assert.AreEqual(4, actual.Count);

}

}

}

# QualifyingTest.cs

using SIS.Domain.Manage;

using Microsoft.VisualStudio.TestTools.UnitTesting; using System;

namespace UnitTestSIS

{

/// <summary>

///This is a test class for QualifyingTest and is intended

///to contain all QualifyingTest Unit Tests

///</summary> [TestClass()]

public class QualifyingTest

{

/// <summary>

///A test for Qualifying Constructor

///</summary> [TestMethod()]

public void QualifyingConstructorTest1()

{

string name = "test";

Qualifying target = new Qualifying(name); Assert.AreEqual(name, target.Name);

}

/// <summary>

///A test for Qualifying Constructor

///</summary> [TestMethod()]

public void QualifyingConstructorTest()

{

string name = "TestName";

Qualifying target = new Qualifying(name); Assert.AreEqual(name, target.Name);

}

/// <summary>

///A test for insertQualifying

///</summary> [TestMethod()]

public void insertQualifyingTest()

{

string studentID = "100599008"; string name = "TestName";

bool expected = true; bool actual;

actual = Qualifying.insertQualifying(studentID, name); Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for updateQualifying

///</summary> [TestMethod()]

public void updateQualifyingTest()

{

string studentID = "100599008"; string name = "TestName"; string newName = "newTestName";

bool expected = true; bool actual;

actual = Qualifying.updateQualifying(studentID, name, newName); Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for deleteQualifying

///</summary> [TestMethod()]

public void deleteQualifyingTest()

{

string studentID = "100599008"; string name = "newTestName";

bool expected = true; bool actual;

actual = Qualifying.deleteQualifying(studentID, name); Assert.AreEqual(expected, actual);

}

}

}

# StudentInformationTest.cs

using SIS.Domain.Manage;

using Microsoft.VisualStudio.TestTools.UnitTesting; using System;

namespace UnitTestSIS

{

/// <summary>

///This is a test class for StudentInformationTest and is intended

///to contain all StudentInformationTest Unit Tests

///</summary> [TestClass()]

public class StudentInformationTest

{

/// <summary>

///A test for getStudentInformation

///</summary> [TestMethod()]

public void getStudentInformationTest()

{

string studentID = "100599008";

StudentInformation actual;

actual = StudentInformation.getStudentInformation(studentID); Assert.AreEqual(studentID,

actual.PriInformation.StudentID.Trim());

Assert.AreEqual(2, actual.ExtInformation.Qualifications.Count); Assert.AreEqual(5, actual.ExtInformation.Courses.Count); Assert.AreEqual(1, actual.ExtInformation.Papers.Count);

}

}

}

# StudentTest.cs

using SIS.Domain.Manage;

using Microsoft.VisualStudio.TestTools.UnitTesting; using System;

using System.Collections.Generic;

namespace UnitTestSIS

{

/// <summary>

///This is a test class for StudentTest and is intended

///to contain all StudentTest Unit Tests

///</summary> [TestClass()]

public class StudentTest

{

/// <summary>

///A test for enterPersonalInformation

///</summary> [TestMethod()]

public void enterPersonalInformationTest()

{

Student target = new Student(); string studentID = "100599009";

string birthday = "1981/09/17"; string national = "Korea"; string password = "test";

bool expected = true; bool actual;

actual = target.enterPersonalInformation(studentID, birthday, national, password);

Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for showExtendInformation

///</summary> [TestMethod()]

public void showExtendInformationTest()

{

Student target = new Student(); string studentID = "100599009"; string \_birthday = "1981/09/17"; string \_national = "Korea"; string \_password = "test";

List<Qualifying> \_qualifications = new List<Qualifying>(); List<Course> \_courses = new List<Course>();

List<Paper> \_papers = new List<Paper>();

ExtendInformation expected = new ExtendInformation(\_birthday,

\_national, \_password, \_qualifications, \_courses, \_papers);

ExtendInformation actual;

actual = target.showExtendInformation(studentID); Assert.AreEqual(expected.Birthday, actual.Birthday.Trim()); Assert.AreEqual(expected.National, actual.National.Trim()); Assert.AreEqual(expected.Password, actual.Password.Trim()); Assert.AreEqual(expected.Qualifications.Count,

actual.Qualifications.Count);

Assert.AreEqual(expected.Courses.Count, actual.Courses.Count); Assert.AreEqual(expected.Papers.Count, actual.Papers.Count);

}

/// <summary>

///A test for enterCourse

///</summary> [TestMethod()]

public void enterCourseTest()

{

Student target = new Student(); string studentID = "100599008"; string name = "TestName";

byte credit = 3;

bool require = false; bool expected = true;

bool actual;

actual = target.enterCourse(studentID, name, credit, require); Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for updateCourse

///</summary> [TestMethod()]

public void updateCourseTest()

{

Student target = new Student(); string studentID = "100599008"; string name = "TestName"; string newName = "newTestName"; byte credit = 3;

bool require = true;

require);

}

bool expected = true; bool actual;

actual = target.updateCourse(studentID, name, newName, credit, Assert.AreEqual(expected, actual);

/// <summary>

///A test for deleteCourse

///</summary> [TestMethod()]

public void deleteCourseTest()

{

Student target = new Student(); string studentID = "100599008"; string name = "newTestName";

bool expected = true; bool actual;

actual = target.deleteCourse(studentID, name); Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for enterPaper

///</summary> [TestMethod()]

public void enterPaperTest()

{

Student target = new Student(); string studentID = "100599008"; string title = "TestName";

string publisher = "TestPublisher"; string year = "Nov, 2012";

bool expected = true; bool actual;

actual = target.enterPaper(studentID, title, publisher, year); Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for updatePaper

///</summary> [TestMethod()]

public void updatePaperTest()

{

Student target = new Student(); string studentID = "100599008"; string title = "TestName"; string newTitle = "newTestName";

string publisher = "TestPublisher"; string year = "Nov, 2012";

bool expected = true; bool actual;

actual = target.updatePaper(studentID, title, newTitle, publisher, year);

Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for deletePaper

///</summary> [TestMethod()]

public void deletePaperTest()

{

Student target = new Student(); string studentID = "100599008"; string title = "newTestName";

bool expected = true; bool actual;

actual = target.deletePaper(studentID, title); Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for enterQualifying

///</summary> [TestMethod()]

public void enterQualifyingTest()

{

Student target = new Student(); string studentID = "100599008"; string name = "TestName";

bool expected = true; bool actual;

actual = target.enterQualifying(studentID, name); Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for updateQualifying

///</summary> [TestMethod()]

public void updateQualifyingTest()

{

Student target = new Student(); string studentID = "100599008"; string name = "TestName"; string newName = "newTestName"; bool expected = true;

bool actual;

actual = target.updateQualifying(studentID, name, newName); Assert.AreEqual(expected, actual);

}

/// <summary>

///A test for deleteQualifying

///</summary> [TestMethod()]

public void deleteQualifyingTest()

{

Student target = new Student(); string studentID = "100599008"; string name = "newTestName"; bool expected = true;

bool actual;

actual = target.deleteQualifying(studentID, name); Assert.AreEqual(expected, actual);

}

}

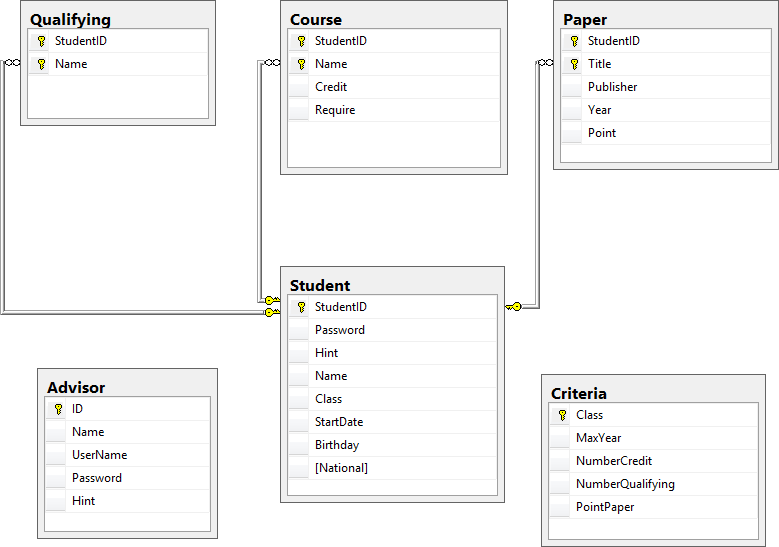
}

# Measurement

|  |  |
| --- | --- |
| 2012/06/01 19:00-21:00 | 2.0 H |
| 2012/06/02 19:00-21:00 | 2.0 H |
| 2012/06/03 09:00-12:00 | 3.0 H |
| 2012/06/07 20:00-24:00 | 4.0 H |
| 2012/06/08 09:00-12:00 | 3.0 H |
| 2012/06/08 20:00-24:00 | 4.0 H |
| 2012/06/09 09:00-12:00 | 3.0 H |
| 2012/06/10 19:00-23:00 | 4.0 H |
| 2012/06/11 09:00-12:00 | 3.0 H |
| **Total** | **28.0 H** |

**Appendix**

**SQL Server Database Tables**



**Test Data**

