## Lab: Extending the Class Enrollment Application Functionality

### Lab Setup

Estimated Time: **90 minutes**

### Preparation Steps

1. Prepare your database:
   * Open File Explorer and navigate to **E:/Allfiles/Mod02/Labfiles/Databases**.
   * Double-click on **SetupSchoolDB.cmd**.
   * If a Windows protected your PC dialog appears, click **More info** and then click **Run Anyway**.
   * Close File Explorer.

### Exercise 1: Refactoring the Enrollment Code

#### Task 1: Copy the code for editing a student into the studentsList\_MouseDoubleClick event handler

1. Open **Visual Studio 2017**.
2. In **Visual Studio**, on the **File** menu, point to **Open**, and then click **Project/Solution**.
3. In the **Open Project** dialog box, browse to **E:/Allfiles/Mod02/Labfiles/Exercise 1**, click **School.sln**, and then click **Open**.

**Note :** If any Security warning dialog box appears, clear **Ask me for every project in this solution** check box and then click **OK**.

1. On the **View** menu, click **Task List**.
2. Double-click the **// TODO: Exercise 1: Task 1a: Copy code for editing the details for that student** task, below the comment locate the code, and then copy the following code to the clipboard:

Student student = this.studentsList.SelectedItem as Student;

* // TODO: Exercise 1: Task 3a: Refactor as the editStudent method
* // Use the StudentForm to display and edit the details of the student StudentForm sf = new StudentForm();
* // Set the title of the form and populate the fields on the form with the
* details of the student
* sf.Title = “Edit Student Details”;
* sf.firstName.Text = student.FirstName; sf.lastName.Text = student.LastName;
* sf.dateOfBirth.Text = student.DateOfBirth.ToString(“d”);
* // Format the date to omit the time element
* // Display the form if (sf.ShowDialog().Value) {
* // When the user closes the form, copy the details back to the student
* student.FirstName = sf.firstName.Text;
* student.LastName = sf.lastName.Text; remov
* student.DateOfBirth = DateTime.Parse(sf.dateOfBirth.Text);
* // Enable saving (changes are not made permanent until they are written
* // back to the database)  
   saveChanges.IsEnabled = true;
* }

1. Double-click the **TODO: Exercise 1: Task 1b: If the user double-clicks a student, edit the details for the student** task.
2. Paste the code from the clipboard into **studentsList\_MouseDoubleClick** method.

#### Task 2: Run the application and verify that the user can now double-click a student to edit their details

1. On the **Build** menu, click **Build Solution**.
2. On the **Debug** menu, click **Start Without Debugging**.
3. Click the row containing the name **Kevin Liu** and then press Enter.
4. Verify that the **Edit Student Details** window appears, displaying the correct details.
5. In the **Last Name** text box, delete the existing contents, type **Cook**, and then click **OK**.
6. Verify that **Liu** has changed to **Cook** in the student list, and that the **Save Changes** button is now enabled.
7. Double-click the row containing the name **George Li**.
8. Verify that the **Edit Student Details** window appears, displaying the correct details.
9. In the **First Name** text box, delete the existing contents, and then type **Darren**.
10. In the **Last Name** text box, delete the existing contents, type **Parker**, and then click **OK**.
11. Verify that **George Li** has changed to **Darren Parker**.
12. Close the application.

#### Task 3: Refactor the logic that adds and deletes a student into the AddNewStudent and DeleteStudent and EditStudent methods

1. In the **Task List** window, double-click the **TODO: Exercise 1: Task 3a: Refactor as the AddNewStudent method** task.
2. In the code editor, locate the **case Key.Insert:** block, and then highlight the following code:

// TODO: Exercise 1: Task 3a: Refactor as the addNewStudent method

* // Use the StudentsForm to get the details of the student from the user
* sf = new StudentForm();
* // Set the title of the form to indicate which class the student will be added
* to (the class for the currently selected teacher)
* sf.Title = “New Student for Class” + teacher.Class;
* // Display the form and get the details of the new student
* if (sf.ShowDialog().Value) {
* // When the user closes the form, retrieve the details of the student from
* // the form
* // and use them to create a new Student object  
   Student newStudent = new Student();  
   newStudent.FirstName = sf.firstName.Text;  
   newStudent.LastName = sf.lastName.Text;  
   newStudent.DateOfBirth = DateTime.Parse(sf.dateOfBirth.Text);  
    
   // Assign the new student to the current teacher  
   this.teacher.Students.Add(newStudent);  
    
   // Add the student to the list displayed on the form  
   this.studentsInfo.Add(newStudent);  
    
   // Enable saving (changes are not made permanent until they are written
* // back to the database)  
   saveChanges.IsEnabled = true;
* }

1. On the **Edit** menu, point to **Refactor**, and then click **Extract Method**.
2. In the code editor rename the “NewMethod” to “AddNewStudent” and then click **Apply**.
3. Locate the **AddNewStudent** method and in the method, modify the **sf = new StudentForm();** code to look like the following code (if necessary):

StudentForm sf = new StudentForm();

1. In the **Task List** window, double-click the **TODO: Exercise 1: Task 3b: Refactor as the removeStudent method** task.
2. In the code editor, locate the **case Key.Delete** block, and cut the following code to the clipboard:

// TODO: Exercise 1: Task 3b: Refactor as the removeStudent method

* // Prompt the user to confirm that the student should be removed MessageBoxResult
* response = MessageBox.Show(String.Format(“Remove {0}”, student.FirstName + " "
* + student.LastName), “Confirm”, MessageBoxButton.YesNo,
* MessageBoxImage.Question, MessageBoxResult.No);
* // If the user clicked Yes, remove the student from the database
* if (response == MessageBoxResult.Yes) {
* this.schoolContext.Students.DeleteObject(student);
* // Enable saving (changes are not made permanent until they are written
* // back to the database)  
   saveChanges.IsEnabled = true;
* }

1. In the code editor, in the **case Key.Delete:** block, click at the end of the **student = this.studentsList.SelectedItem as Student;** code line, press Enter, then type the following code:

RemoveStudent(student);

1. Right-click the **RemoveStudent(student);** method call, point to **Quick Action and Refactorings..**, and then click **Generate method ‘MainWindow.RemoveStudent’**.
2. Locate the **RemoveStudent** method below the **studentsList\_KeyDown** method, delete all the generated code in this method, and then paste the code from the clipboard.
3. Double-click the **// TODO: Exercise 1: Task 3c: create Edit student method** task below the comment, add the following code:

* private void EditStudent(Student student)  
  {  
    
  }

1. Double-click the **// TODO: Exercise 1: Task 3d: Refactor as the EditStudent method** task, below the comment locate the code, and cut the following code to the clipboard:

>**Note :** Make sure the cursor is in studentsList\_MouseDoubleClick method.

* // Set the title of the form and populate the fields on the form
* // with the details of the student  
  sf.Title = "Edit Student Details";  
  sf.firstName.Text = student.FirstName;  
  sf.lastName.Text = student.LastName;
* // Format the date to omit the time element  
  sf.dateOfBirth.Text = student.DateOfBirth.ToString("d");
* // Display the form  
  if (sf.ShowDialog().Value)  
  {  
   // When the user closes the form, copy the details back to the student  
   student.FirstName = sf.firstName.Text;  
   student.LastName = sf.lastName.Text;  
   student.DateOfBirth = DateTime.Parse(sf.dateOfBirth.Text);  
    
   // Enable saving (changes are not made permanent until they are written
* // back to the database)  
   saveChanges.IsEnabled = true;  
  }

1. Replace the code with the following code:

* EditStudent(student);

1. Paste the code that you cut from the previous task to the **EditStudent** method.
2. Locate the **studentsList\_MouseDoubleClick** method and replace the code inside the method with the following code:

* Student student = this.studentsList.SelectedItem as Student;  
  EditStudent(student);

#### Task 4: Verify that students can still be added and removed from the application

1. On the **Build** menu, click **Build Solution**.
2. On the **Debug** menu, click **Start Without Debugging**.
3. Click the row containing the name **Kevin Liu**, and then press Insert.
4. Verify that the **New Student for Class 3C** window appears.
5. In the **First Name** text box, type **Dominik**.
6. In the **Last Name** text box, type **Dubicki**.
7. In the **Date of Birth** text box, type **02/03/2006** and then click **OK**.
8. Verify that **Dominik Dubicki** has been added to the student list.
9. Click the row containing the name **Run Liu**, and then press Delete.
10. Verify that the confirmation prompt appears.
11. Click **Yes**, and then verify that **Run Liu** is removed from the student list.
12. Close the application.

#### Task 5: Debug the application and step into the new method calls

1. In the code editor, locate the **studentsList\_MouseDoubleClick** method, right-click the **Student student = this.studentsList.SelectedItem as Student** statement, point to **Breakpoint**, and then click **Insert Breakpoint**.
2. On the **Debug** menu, click **Start Debugging**.
3. Double Click the row that contains the name **Kevin Liu** and press Enter.
4. In **Visual Studio**, click the **Call Stack** tab.
5. Note that the current method name is displayed in the **Call Stack** window.
6. In **Visual Studio**, click the **Locals** tab.
7. Note that the local variables, **this**, **sender, e**, and **student**, are displayed in the **Locals** window.
8. On the **Debug** menu, click **Step Over**.
9. Look at the **Locals** window, and note that after stepping over the **Student student = this.studentsList.SelectedItem as Student;** code, the value for the **student** variable has changed from **null** to **School.Data.Student**.
10. In the **Locals** window, expand **student** and note the values for **\_FirstName** and **\_LastName**.
11. On the **Debug** menu, click **Step Into**.
12. Note that execution steps into the **EditStudent** method, and the name of this method has been added to the call stack.
13. In the **Locals** window, note that the local variables have been changed.
14. On the **Debug** menu,click **Step Into**.
15. On the **Debug** menu, click **Step Over**.
16. Repeat step 15, five times.
17. In the **Edit Student Details** window, click **Cancel**.
18. On the **Debug** menu,click **Step Into**.
19. Note that execution returns to the **studentsList\_MouseDoubleClick** method.
20. On the **Debug** menu, click **Step Over**.
21. On the **Debug** menu, click **Continue**.
22. On the **New Student** panel, click **Cancel**.
23. Close the application
24. In **Visual Studio**, on the **Debug** menu, click **Delete All Breakpoints**.
25. In the **Microsoft Visual Studio** message box, click **Yes**.
26. On the **File** menu, click **Close Solution**.

**Results:** After completing this exercise, you should have updated the application to refactor the duplicate code into reusable methods.

### Exercise 2: Validating Student Information

#### Task 1: Run the application and observe that student details that are not valid can be entered

1. In **Visual Studio**, on the **File** menu, point to **Open**, and then click **Project/Solution**.
2. In the **Open Project** dialog box, browse to **E:/Allfiles/Mod02/Labfiles/Exercise 2**, click **School.sln**, and then click **Open**.

>**Note :** If any Security warning dialog box appears, clear **Ask me for every project in this solution** check box and then click **OK**.

1. On the **Build** menu, click **Build Solution**.
2. On the **Debug** menu, click **Start Without Debugging**.
3. Click the row containing the name **Kevin Liu**, and then press Insert.
4. Leave the **First Name** and **Last Name** text boxes empty.
5. In the **Date of Birth** text box, type **10/06/3012**, and then click **OK**.
6. Verify that a new row has been added to the student list, containing a blank first name, blank last name, and a negative age.
7. Close the application.

#### Task 2: Add code to validate the first name and last name fields

1. In the **Task List** window, double-click the **TODO: Exercise 2: Task 2a: Check that the user has provided a first name** task.
2. In the code editor, click at the end of the comment line, press **Enter**, and then type the following code:

if (String.IsNullOrEmpty(this.firstName.Text)) {

MessageBox.Show("The student must have a first name", "Error",

MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

1. In the **Task List** window, double-click the **TODO: Exercise 2: Task 2b: Check that the user has provided a last name** task.
2. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

if (String.IsNullOrEmpty(this.lastName.Text)){

MessageBox.Show("The student must have a last name",

"Error", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

#### Task 3: Add code to validate the date of birth

1. In the **Task List** window, double-click the **TODO: Exercise 2: Task 3a: Check that the user has entered a valid date for the date of birth** task.
2. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

DateTime result;

* if (!DateTime.TryParse(this.dateOfBirth.Text, out result)) {
* MessageBox.Show(“The date of birth must be a valid date”, “Error”,
* MessageBoxButton.OK, MessageBoxImage.Error);
* return;
* }

1. In the **Task List** window, double-click the **TODO: Exercise 2: Task 3b: Verify that the student is at least 5 years old** task.
2. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

TimeSpan age = DateTime.Now.Subtract(result);

* if (age.Days / 365.25 < 5) {
* MessageBox.Show(“The student must be at least 5 years old”, “Error”,
* MessageBoxButton.OK, MessageBoxImage.Error);
* return;
* }

#### Task 4: Run the application and verify that student information is now validated correctly

1. On the **Build** menu, click **Build Solution**.
2. On the **Debug** menu, click **Start Without Debugging**.
3. Click the row containing the name **Kevin Liu**, and then press Insert.
4. Leave the **First Name**, **Last Name**, and **Date of Birth** text boxes empty and click **OK**.
5. Verify that an error message appears containing the text **The student must have a first name**.
6. In the **Error** message box, click **OK**.
7. In the **new student** window, in the **First Name** text box, type **Darren**, and then click **OK**.
8. Verify that an error message appears containing the text **The student must have a last name**.
9. In the **Error** message box, click **OK**.
10. In the **new student** window, in the **Last Name** text box, type **Parker**, and then click **OK**.
11. Verify that an error message that says **The date of birth must be a valid date** appears.
12. In the **Error** message box, click **OK**.
13. In the **new student** window, in the **Date of Birth** text box, type **10/06/3012**, and then click **OK**.
14. Verify that an error message that says **The student must be at least 5 years old** appears.
15. On the **Error** message box, click **OK**.
16. In the **new student** window, in the **Date of Birth** text box, delete the existing date, type **10/06/2006**, and then click **OK**.
17. Verify that **Darren Parker** is added to the student list with an age appropriate to the current date.
18. Close the application.
19. On the **File** menu, click **Close Solution**.

**Results:** After completing this exercise, the student data will be validated before it is saved.

### Exercise 3: Saving Changes to the Class List

#### Task 1: Verify that data changes are not persisted to the database

1. In **Visual Studio**, on the **File** menu, point to **Open**, and then click **Project/Solution**.
2. In the **Open Project** dialog box, browse to **E:/Allfiles/Mod02/Labfiles/Exercise 3**, click **School.sln**, and then click **Open**. >**Note :** If any Security warning dialog box appears, clear **Ask me for every project in this solution** check box and then click **OK**.
3. On the **Build** menu, click **Build Solution**.
4. On the **Debug** menu, click **Start Without Debugging**.
5. Click the row containing the name **Kevin Liu**.
6. Press Enter and verify that the **Edit Student Details** window appears displaying the correct details.
7. In the **Last Name** text box, delete the existing contents, type **Cook**, and then click **OK**.
8. Verify that **Liu** has changed to **Cook** in the student list, and that the **Save Changes** button is now enabled.
9. Click the row containing the student **George Li**, and then press Delete.
10. Verify that the confirmation prompt appears, and then click **Yes**.
11. Verify that **George Li** is removed from the student list.
12. Close the application.
13. On the **Debug** menu, click **Start Without Debugging**.
14. Verify that the application displays the original list of students.
15. Verify that **Kevin Liu** appears in the list instead of **Kevin Cook** and **George Li** is back in the student list.
16. Close the application.

#### Task 2: Add code to save changes back to the database

1. In **Visual Studio**, in the **Task List** window, double-click the **TODO: Exercise 3: Task 2a: Bring the System.Data and System.Data.Objects namespace into scope** task.
2. In the code editor, click in the blank line above the comment, and then type the following code:
3. using System.Data;
4. using System.Data.Objects;
5. In **Visual Studio**, in the **Task List** window, double-click the **TODO: Exercise 3: Task 2b: Save the changes by calling the SaveChanges method of the schoolContext object** task.
6. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

try {

// Save the changes

this.schoolContext.SaveChanges();

* // Disable the Save button (it will be enabled if the user
* // makes more changes)
* saveChanges.IsEnabled = false;
* }

#### Task 3: Add exception handling to the code to catch concurrency, update, and general exceptions

1. In the code editor, enclose the code that you wrote in the previous task in a **try** block. Your code should look like the following:

try {

// Save the changes

this.schoolContext.SaveChanges();

* // Disable the Save button (it will be enabled if the
* //user makes more changes)  
   saveChanges.IsEnabled = false;
* }

1. In the **Task List** window, double-click the **TODO: Exercise 3: Task 3a: If an OptimisticConcurrencyException occurs then another user has changed the same students earlier then overwrite their changes with the new data (see the lab instructions for details)** task.
2. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

catch (OptimisticConcurrencyException){

// If the user has changed the same students earlier,

// then overwrite their changes with the new data

this.schoolContext.Refresh(RefreshMode.StoreWins, schoolContext.Students);

this.schoolContext.SaveChanges();

}

1. In the **Task List** window, double-click the **TODO: Exercise 3: Task 3b: If an UpdateException occurs then report the error to the user and rollback (see the lab instructions for details)** task.
2. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

catch (UpdateException uEx) {

// If some sort of database exception has occurred, then display the reason

// for the exception and rollback

* MessageBox.Show(uEx.InnerException.Message, "Error saving changes");  
   this.schoolContext.Refresh(RefreshMode.StoreWins, schoolContext.Students);
* }

1. In the **Task List** window, double-click the **TODO: Exercise 3: Task 3c: If some other sort of error has occurs, report the error to the user and retain the data so the user can try again - the error may be transitory (see the lab instructions for details)** task.
2. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

catch (Exception ex){

// If some other exception occurs, report it to the user

MessageBox.Show(ex.Message, "Error saving changes");

this.schoolContext.Refresh(RefreshMode.ClientWins, schoolContext.Students);

}

#### Task 4: Run the application and verify that data changes are persisted to the database

1. On the **Build** menu, click **Build Solution**.
2. On the **Debug** menu, click **Start Without Debugging**.
3. Click the row containing the student **Kevin Liu**.
4. Press Enter, in the **Last Name** text box delete the existing contents, type **Cook**, and then click **OK**.
5. Verify that **Liu** has changed to **Cook** in the student list, and that the **Save Changes** button is now enabled.
6. Click **Save Changes** and verify that the **Save Changes** button is now disabled.
7. Click the row containing the student **George Li** and press Delete.
8. Verify that the confirmation prompt appears, and then click **Yes**.
9. Verify that the **Save Changes** button is now enabled.
10. Click **Save Changes** and verify that the button is now disabled.
11. Close the application.
12. On the **Debug** menu, click **Start Without Debugging**.
13. Verify that the changes you made to the student data have been saved to the database and are reflected in the student list.
14. Close the application.
15. On the **File** menu, click **Close Solution**.

**Results:** After completing this exercise, modified student data will be saved to the database.