Lab Answer Key: Module 1: Review of Visual C# Syntax

Lab: Developing the Class Enrollment Application

Exercise 1: Implementing Edit Functionality for the Students List

Task 1: Detect whether the user has pressed the Enter key

1. Start the MSL-TMG1 virtual machine if it is not already running.
2. Start the 20483B-SEA-DEV11 virtual machine.
3. Log on to Windows 8 as **Student** with the password **Pa$$w0rd**. If necessary, click **Switch User** to display the list of users.
4. Switch to the Windows 8 Start window and then type Explorer.
5. In the **Apps** list, click **File Explorer**.
6. Navigate to the **E:\Mod01\Labfiles\Databases** folder, and then double-click **SetupSchoolDB.cmd**.
7. Close File Explorer.
8. Switch to the Windows 8 **Start** window.
9. Click **Visual Studio 2012**.
10. In Visual Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.
11. In the **Open Project** dialog box, browse to **E:\Mod01\Labfiles\Starter\Exercise 1**, click **School.sln**, and then click **Open**.
12. In Solution Explorer, expand **School**, and then expand **MainWindow.xaml**.

13. Double-click **MainWindow.xaml.cs**.

14. In Visual Studio, on the **View** menu, click **Task List**.

15. In the **Task List** window, in the **Categories** list, click **Comments**.

16. Double-click the **TODO: Exercise 1: Task 1a: If the user pressed Enter, edit the details for the currently selected student** task.

17. In the code editor, click at the beginning of the comment line, press Enter, and in the blank space above the comment, type the following code:

```
switch (e.Key)
{
```

18. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

```
    case Key.Enter: Student student =
this.studentsList.SelectedItem as Student;
```

19. After all the comments in this method, type the following code:

```
        break;
}
```

Task 2: Initialize the StudentForm window and populate it with the details of the currently selected student

1. In the **Task List** window, double-click the **TODO: Exercise 1: Task 2a: Use the StudentsForm to display and edit the details of the student** task.

2. In the code editor, click at the end of the comment line, press Enter, and then

type the following code:

```
StudentForm sf = new StudentForm();
```

3. In the **Task List** window, double-click the **TODO: Exercise 1: Task 2b: Set the title of the form and populate the fields on the form with the details of the student** task.
4. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

```
sf.Title = "Edit Student Details";
sf.firstName.Text = student.FirstName;
sf.lastName.Text = student.LastName;
sf.dateOfBirth.Text = student.DateOfBirth.ToString("d");
```


Task 3: Display the StudentForm window and copy the updated student details entered back to the Student object

1. In the **Task List** window, double-click the **TODO: Exercise 1: Task 3a: Display the form** task.
2. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

```
if (sf.ShowDialog().Value)
{
```

3. After all the comments in this method, add the following code:

```
}
```

4. In the **Task List** window, double-click the **TODO: Exercise 1: Task 3b: When the user closes the form, copy the details back to the student** task.

5. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

```
student.FirstName = sf.firstName.Text;
student.LastName = sf.lastName.Text;
student.DateOfBirth =
DateTime.Parse(sf.dateOfBirth.Text);
```

6. In the **Task List** window, double-click the **TODO: Exercise 1: Task 3c: Enable saving (changes are not made permanent until they are written back to the database)** task.

7. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

```
saveChanges.IsEnabled = true;
```

Task 4: Run the application and verify that the edit functionality works as expected

1. On the **Build** menu, click **Build Solution**.

2. On the **Debug** menu, click **Start Without Debugging**.

3. Verify that the application starts and displays the initial list of students.

The initial students list should look like this:

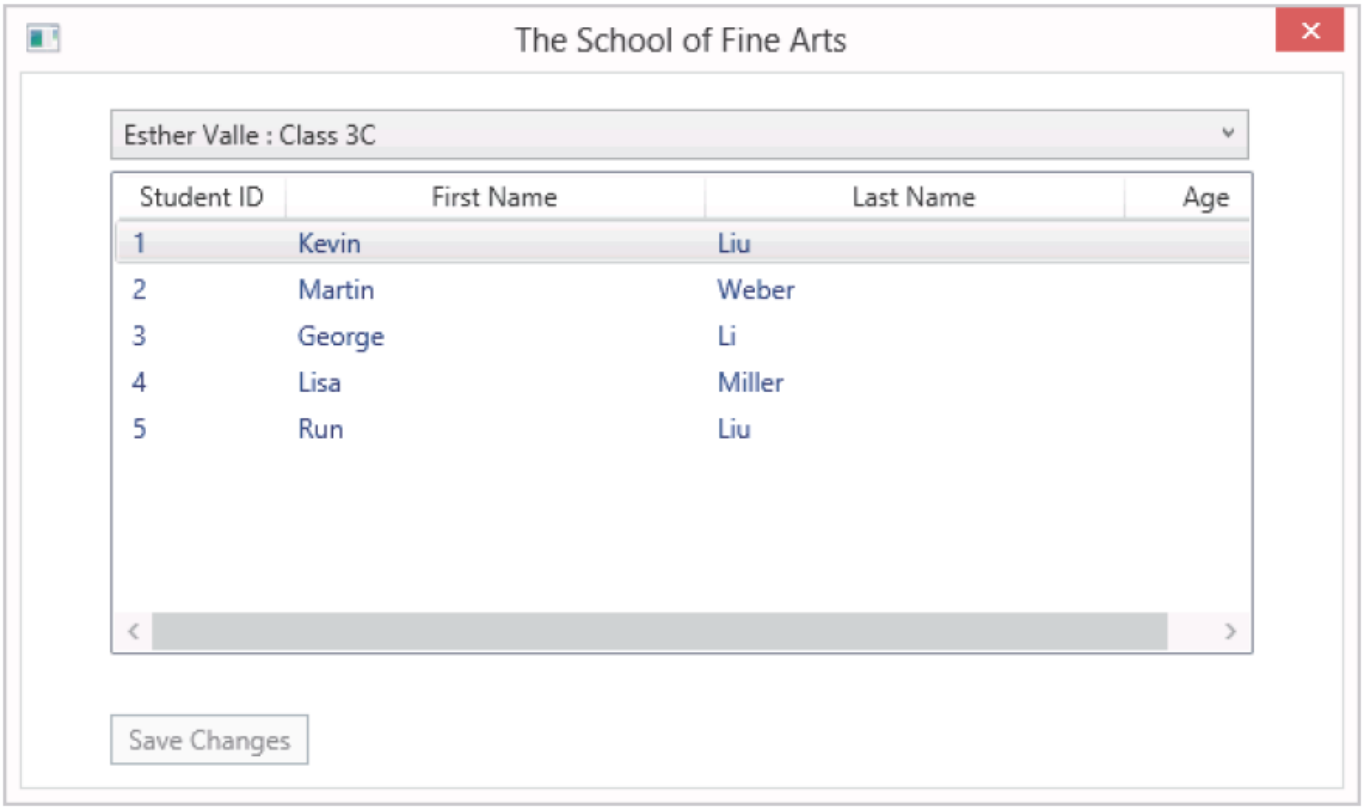

FIGURE 01.1:THE INITIAL STUDENTS LIST

4. Click the row containing the name **Kevin Liu**.
5. Press Enter and verify that the **Edit Student Details** window appears and displays the correct details:

The **Edit Student Details** window should look similar to the following:

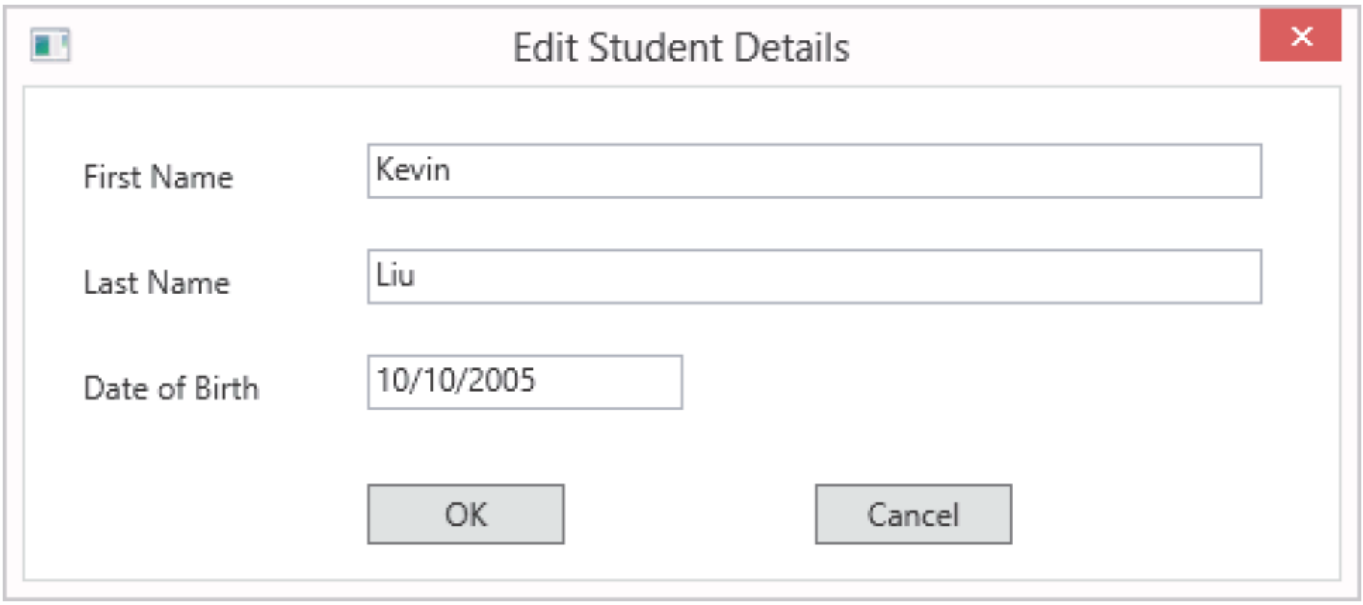

FIGURE 01.2:EDIT STUDENT DETAILS FORM

6. In the **Last Name** text box, delete the existing contents, type **Cook**, and then click **OK**.

7. Verify that Liu has changed to Cook in the students list, and that the **Save Changes** button is now enabled.

8. Close the application.

Task 5: Use the Visual Studio Debugger to step through the code.

1. In Visual Studio, in the **Task List** window, double-click the **TODO: Exercise 1: Task 2b: Set the title of the form and populate the fields on the form with the details of the student** task.

2. In the following line of code, right-click the word **Title** in **sf.Title = "Edit Student Details";**, point to **Breakpoint**, and then click **Insert Breakpoint**.

3. On the **Debug** menu, click **Start Debugging**.

4. Click the row containing the name **George Li**, and then press Enter.

5. When Visual Studio enters break mode, in the bottom left window, click the **Watch 1** tab.

6. In the **Watch 1** window, click below **Name** to create a blank row.

7. In the **Name** column, type **sf.Title**, and then press Enter.

8. In the **Watch 1** window, click below **sf.Title** to create a blank row.

9. Type **sf.firstName.Text**,and then press Enter.

10. In the **Watch 1** window, click below **sf.firstName.Text** to create a blank row.

11. Type **sf.lastName.Text**, and then press Enter.

12. In the **Watch 1** window, click below **sf.lastName.Text** to create a blank row.

13. Type **sf.dateOfBirth.Text**, and then press Enter.

14. On the **Debug** menu, click **Step Over**.

15. Repeat step 14 three times.
16. In the bottom middle window, click the **Immediate Window** tab.
17. In the **Immediate Window**, type **sf.firstName.Text**, and then press Enter.
18. Verify that **"George"** is displayed.
19. In the **Watch 1** window, in the **sf.firstName.Text** row, right-click the **Value** field, and then click **Edit Value**.
20. Type **"Dominik"** and press Enter.
21. In the **Immediate Window**, type **sf.lastName.Text**, and then press Enter.
22. Verify that **"Li"** is displayed
23. Type **sf.lastName.Text = "Dubicki";**, and then press Enter.
24. In the **Watch 1** window, in the **sf.lastName.Text** row, verify that the **Value** column has changed to **"Dubicki"**.
25. On the **Debug** menu, click **Continue**.
26. Verify that the Edit Student Details form contains the information in the following table:

Field	Value
First Name	Dominik
Last Name	Dubicki
Date of Birth	8/10/2005

27. Close the application.
28. In Visual Studio, on the **Debug** menu, click **Delete All Breakpoints**.
29. In the **Microsoft Visual Studio** dialog box, click **Yes**.
30. On the **File** menu, click **Close Solution**.
31. In the **Microsoft Visual Studio** dialog box, click **Yes**.

Results: After completing this exercise, users will be able to edit the details of a student.

Exercise 2: Implementing Insert Functionality for the Students List

Task 1: Add logic to the key down method to detect if the Insert key has been pressed.

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:\Mod01\Labfiles\Starter\Exercise 2**, click **School.sln**, and then click **Open**.
3. In the **Task List** window, double-click the **TODO: Exercise 2: Task 1a: If the user pressed Insert, add a new student** task.
4. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

```
case Key.Insert:
```

Task 2: Initialize the student form

1. In the **Task List** window, double-click the **TODO: Exercise 2: Task 2a: Use the StudentsForm to get the details of the student from the user** task.
2. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

```
sf = new StudentForm();
```

3. In the **Task List** window, double-click the **TODO: Exercise 2: Task 2b: Set the title of the form to indicate which class the student will be added to (the class for the currently selected teacher)** task.

4. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

```
sf.Title = "New Student for Class " + teacher.Class;
```

Task 3: Display the StudentForm window and enable the user to provide the details of the new student

1. In the **Task List** window, double-click the **TODO: Exercise 2: Task 3a: Display the form and get the details of the new student** task.

2. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

```
if (sf.ShowDialog().Value)
{
```

3. After all the comments in this method, add the following code:

```
}
break;
```

4. In the **Task List** window, double-click the **TODO: Exercise 2: Task 3b: When the user closes the form, retrieve the details of the student from the form and use them to create a new Student object** task.

5. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

```
Student newStudent = new Student();
newStudent.FirstName = sf.firstName.Text;
newStudent.LastName = sf.lastName.Text;
newStudent.DateOfBirth = DateTime.Parse(sf.dateOfBirth.Text);
```


Task 4: Assign the new student to a class and enable the user to save the details of the new student

1. In the **Task List** window, double-click the **TODO: Exercise 2: Task 4a: Assign the new student to the current teacher** task.
2. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

```
this.teacher.Students.Add(newStudent);
```

3. In the **Task List** window, double-click the **TODO: Exercise 2: Task 4b: Add the student to the list displayed on the form** task.
4. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

```
this.studentsInfo.Add(newStudent);
```

5. In the **Task List** window, double-click the **TODO: Exercise 2: Task 4c: Enable saving (changes are not made permanent until they are written back to the database)** task.
6. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

```
saveChanges.IsEnabled = true;
```

Task 5: Run the application and verify that the insert functionality works as expected

1. On the **Build** menu, click **Build Solution**.
2. On the **Debug** menu, click **Start Without Debugging**.
3. Verify that the application starts and displays the initial list of students.
4. Click the row containing the name **Kevin Liu**.
5. Press Insert and verify that the new student window appears:
6. In the **First Name** text box, type **Darren**.
7. In the **Last Name** text box, type **Parker**.
8. In the **Date of Birth** text box, type **02/03/2006**, and then click **OK**.
9. Verify that Darren Parker has been added to the students list, and that the **Save Changes** button is now enabled. The ID of a new student will be 0 until they are saved to the database in the next lab.
10. Close the application.
11. On the **File** menu, click **Close Solution**.

Results: After completing this exercise, users will be able to add new students to a class.

Exercise 3: Implementing Delete Functionality for the Students List

Task 1: Add logic to the key down method to detect if the Delete key has been pressed.

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:\Mod01\Labfiles\Starter\Exercise 3**, click **School.sln**, and then click **Open**.
3. In the **Task List** window, double-click the **TODO Exercise: 3: Task 1a: If the user pressed Delete, remove the currently selected student** task.
4. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

```
case Key.Delete: student = this.studentsList.SelectedItem as
Student;
```

Task 2: Prompt the user to confirm that they want to remove the selected student from the class

1. In the **Task List** window, double-click the **TODO: Exercise 3: Task 2a: Prompt the user to confirm that the student should be removed** task.
2. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

```
MessageBoxResult response = MessageBox.Show(
string.Format("Remove {0}", student.FirstName + " " +
student.LastName),
"Confirm", MessageBoxButton.YesNo, MessageBoxImage.Question,
MessageBoxResult.No);
```

Task 3: Remove the student and enable the user to save the changes

1. In the **Task List** window, double-click the **TODO: Exercise 3: Task 3a: If the user clicked Yes, remove the student from the database** task.
2. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

```
if (response == MessageBoxResult.Yes)
{
    this.schoolContext.Students.DeleteObject(student);
```

3. After the final comment in this method, type the following code:

```
}
break;
```

4. In the **Task List** window, double-click the **TODO: Exercise 3: Task 3b: Enable saving (changes are not made permanent until they are written back to the database)** task.
5. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

```
saveChanges.IsEnabled = true;
```

Task 4: Run the application and verify that the delete functionality works as expected

1. On the **Build** menu, click **Build Solution**.

2. On the **Debug** menu, click **Start Without Debugging**.
3. Verify that the application starts and displays the initial list of students.
4. Click on the drop-down menu containing the text **Esther Valle: Class 3C**.
5. Click the list item containing the text **David Waite : Class 4B**.
6. Click the row containing the name **Jon Orton**.
7. Press Delete and verify that the confirmation prompt appears.
8. In the **Confirm** dialog box, click **Yes**, verify that Jon Orton is removed from the students list, and then verify that the **Save Changes** button is enabled.
9. Close the application.
10. On the **File** menu, click **Close Solution**.

Results: After completing this exercise, users will be able to remove students from classes.

Exercise 4: Displaying a Student's Age

Task 1: Examine the MainWindow XAML

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:\Mod01\Labfiles\Starter\Exercise 4**, click **School.sln**, and then click **Open**.
3. On the **Build** menu, click **Build Solution**.
4. In Solution Explorer, expand the **School**, and then double-click the **MainWindow.xaml** and view the XAML markup.
5. Take note of the following lines of markup:

```
<app:AgeConverter x:key="ageConverter"/>
. . .
<GridViewColumn Width="75" Header="Age"
DisplayMemberBinding="{Binding Path=DateOfBirth, Converter=
{StaticResource
ageConverter}}" />
```


Task 2: Add logic to the AgeConverter class to calculate a student's age from their date of birth

1. In the **Task List** window, double-click the **TODO: Exercise 4: Task 2a: Check that the value provided is not null. If it is, return an empty string** task.
2. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

```
if (value != null)
{
```

3. In the code editor, after all the comments in this method, delete the following line of code:

```
return "";
```

4. In the **Task List** window, double-click the **TODO: Exercise 4: Task 2b: Convert the value provided into a DateTime value** task.
5. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

```
DateTime studentDateOfBirth = (DateTime)value;
```

6. In the **Task List** window, double-click the **TODO: Exercise 4: Task 2c: Work out the difference between the current date and the value provided** task.

7. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

```
TimeSpan difference =
DateTime.Now.Subtract(studentDateOfBirth);
```

8. In the **Task List** window, double-click the **TODO: Exercise 4: Task 2d: Convert this result into a number of years** task.

9. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

```
int ageInYears = (int)(difference.Days / 365.25);
```

10. In the **Task List** window, double-click the **TODO: Exercise 4: Task 2e: Convert the number of years into a string and return it** task.

11. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

```
        return ageInYears.ToString();
}
else
{
        return "";
}
```

Task 3: Run the application and verify that the student's age now appears correctly

1. On the **Build** menu, click **Build Solution**.
2. On the **Debug** menu, click **Start Without Debugging**.
3. Verify that the application starts and displays the initial list of students, with their ages.
4. Click the row containing the name **Kevin Liu**.
5. Press Insert.
6. In the new student window, enter your first name in the **First Name** box, your last name in the **Last Name** box and your date of birth in the **Date of Birth** box.
7. Click **OK** and verify that your name and age display correctly in the student list.
8. Close the application.
9. On the **File** menu, click **Close Solution**.

Results: After completing this exercise, the application will display a student's age in years.