## Lab: Refactoring Common Functionality into the User Class

### Lab Setup

Estimated Time: **60 minutes**

### Exercise 1: Creating and Inheriting from the User Base Class

#### Task 1: Create the User abstract base class

1. Click **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/Mod05/Labfiles/Starter/Exercise 1**, click **GradesPrototype.sln**, and then click **Open**.
4. In **Visual Studio**, on the **View** menu, click **Task List**.
5. In the **Task List** window, double-click the **TODO: Exercise 1: Task 1a: Create the User abstract class with the common functionality for Teachers and Students** task.
6. In the code editor, click at the end of the comment, press Enter, and then type the following code:

public abstract class User {

1. Click at the end of the last comment in the block (before the **Grade** class declaration), press Enter, and then type the following code:

}

1. In the **Task List** window, double-click the **TODO: Exercise 1: Task 1b: Add the UserName property to the User class** task.
2. In the code editor, click at the end of the comment, press Enter, and then type the following code:

public string UserName { get; set; }

1. In the **Task List** window, double click the **TODO: Exercise 1: Task 1c: Add the Password property to the User class** task.
2. In the code editor, click at the end of the comment, press Enter, and then type the following code:

* // Generate a random password by default  
  private string \_password = Guid.NewGuid().ToString();  
  public string Password  
  {  
   set  
   {
* \_password = value;  
   }  
  }

1. In the **Task List** window, double click the **TODO: Exercise 1: Task 1d: Add the VerifyPassword method to the User class** task.
2. In the code editor, click at the end of the comment, press Enter, and then type the following code:

* public bool VerifyPassword(string pass)  
  {  
   return (String.Compare(pass, \_password) == 0);  
  }

#### Task 2: Modify the Student and Teacher classes to inherit from the User class

1. In the **Task List** window, double-click the **TODO: Exercise 1: Task 2a: Inherit from the User class** task.
2. In the code editor, modify the statement below this comment as shown below in bold:

**public class Student: User, IComparable<Student>**

1. In the **Task List** window, double-click the **TODO: Exercise 1: Task 2b: Remove the UserName property (now inherited from User)** task.
2. In the code editor, delete the following statement from below the comment:

public string UserName { get; set; }

1. In the **Task List** window, double-click the **TODO: Exercise 1: Task 2c: Remove the Password property (now inherited from User)** task.
2. In the code editor, delete the following block of code from below the comment:

private string \_password = Guid.NewGuid().ToString(); // Generate a random password

by default

public string Password {

set { \_password = value; }

}

1. In the **Task List** window, double-click the **TODO: Exercise 1: Task 2d Remove the VerifyPassword method (now inherited from User)** task.
2. In the code editor, delete the following method from below the comment:

public bool VerifyPassword(string pass)

{

return (String.Compare(pass, \_password) == 0);

}

1. In the **Task List** window, double-click the **TODO: Exercise 1: Task 2e: Inherit from the User class** task.
2. In the code editor, modify the statement below this comment as shown below in bold:

* public class **Teacher: User**

1. In the **Task List** window, double-click the **TODO: Exercise 1: Task 2f: Remove the UserName property (now inherited from User)** task.
2. In the code editor, delete the following statement from below the comment:

* public string UserName { get; set; }

1. In the **Task List** window, double-click the **TODO: Exercise 1: Task 2g: Remove the Password property (now inherited from User)** task.
2. In the code editor, delete the following block of code from below the comment:

* private string \_password = Guid.NewGuid().ToString(); // Generate a random password
* by default  
  public string Password  
  {  
   set  
   {  
   \_password = value;  
   }  
  }

1. In the **Task List** window, double-click the **TODO: Exercise 1: Task 2h Remove the VerifyPassword method (now inherited from User)** task.
2. In the code editor, delete the following method from below the comment:

* public bool VerifyPassword(string pass)  
  {  
   return (String.Compare(pass, \_password) == 0);  
  }

#### Task 3: Run the application and test the log on functionality

1. On the **Build** menu, click **Build Solution**.
2. On the **Debug** menu, click **Start Without Debugging**.
3. When the application starts, in the **Username** text box, type **vallee**, in the **Password** text box, type **password**, and then click **Log on**.
4. Verify that the list of students for teacher **Esther Valle** is displayed.
5. Click **Kevin Liu**, and verify that the report card with the grades for **Kevin Liu** is displayed.
6. Click **Log off**.
7. In the **Username** text box, type **liuk**, in the **Password** text box, type **password**, and then click **Log on**.
8. Verify that the report card showing the grades for **Kevin Liu** is displayed again.
9. Click **Log off**.
10. Close the application.
11. In **Visual Studio**, on the **File** menu, click **Close Solution**.

**Result:** After completing this exercise, you should have removed the duplicated code from the **Student** and **Teacher** classes, and moved the code to an abstract base class called **User**.

### Exercise 2: Implementing Password Complexity by Using an Abstract Method

#### Task 1: Define the SetPassword abstract method

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/Mod05/Labfiles/Starter/Exercise 2**, click **GradesPrototype.sln**, and then click **Open**.
3. In **Visual Studio**, on the **View** menu, click **Task List**.
4. In the **Task List** window, double-click the **TODO: Exercise 2: Task 1a: Define an abstract method for setting the password** task.
5. In the code editor, review the comment below this line, click at the end of the comment, press Enter, and then type the following code:

public abstract bool SetPassword(string pwd);

1. In the **Task List** window, double-click the **TODO: Exercise 2: Task 1b: Use the SetPassword method to set the password** task.
2. In the code editor, delete the following statement:

\_password = value;

1. Add the following block of code in the place of the statement that you just deleted:

if (!SetPassword(value))

{

throw new ArgumentException("Password not complex enough", "Password");

}

#### Task 2: Implement the SetPassword method in the Student and Teacher classes

1. In the **Task List** window, double-click the **TODO: Exercise 2: Task 2a: Make \_password a protected field rather than private** task.
2. In the code editor, modify the statement below the comment as shown below in bold:

**protected string \_password = Guid.NewGuid().ToString(); // Generate a random password**

**by default**

1. In the **Task List** window, double-click the **TODO: Exercise 2: Task 2b: Implement SetPassword to set the password for the student** task.
2. In the code editor, review the comment below this line, click at the end of the comment, press Enter, and then type the following code:

public override bool SetPassword(string pwd)

{

// If the password provided as the parameter is at least 6 characters

// long then save it and return true

if (pwd.Length >= 6)

{

\_password = pwd;

return true;

}

// If the password is not long enough, then do not save it and

// return false

return false;

}

1. In the **Task List** window, double-click the **TODO: Exercise 2: Task 2c: Implement SetPassword to set the password for the teacher** task.
2. In the code editor, review the comment below this line, click at the end of the comment, press Enter, and then type the following code:

public override bool SetPassword(string pwd)

{

// Use a regular expression to check that the password contains at

// least two numeric characters

Match numericMatch = Regex.Match(pwd, @".\**[0-9]+.\**[0-9]+.\*");

* // If the password provided as the parameter is at least 8 characters long and
* // contains at least two numeric characters then save it and return true
* if (pwd.Length >= 8 && numericMatch.Success)
* {  
   \_password = pwd;
* return true;
* }  
   // If the password is not complex enough, then do not save it and
* //return false  
   return false;
* }

#### Task 3: Set the password for a new student

1. In the **Task List** window, double-click the **TODO: Exercise 2: Task 3a: Use the SetPassword method to set the password.** task.
2. In the code editor, delete the statement below this comment and replace it with the following block of code:

if (!newStudent.SetPassword(sd.password.Text))

{

throw new Exception("Password must be at least 6 characters long. Student” +

“ not created");

}

#### Task 4: Change the password for an existing user

1. On the **Build** menu, click **Build Solution**.
2. In **Solution Explorer**, expand the **GradesPrototype** project, and then double-click **MainWindow.xaml**.
3. In the XAML editor, scroll down to line 27 and review the following block of XAML code

<Button Grid.Column="2" Margin="5" HorizontalAlignment="Right"

Click="ChangePassword\_Click">

<TextBlock Text="Change Password" FontSize="24"/>

</Button>

1. In **Solution Explorer**, expand **MainWindow.xaml** and then double-click **MainWindow.xaml.cs**.
2. In the code editor, expand the **Event Handlers** region, and locate the **ChangePassword\_Click** method.
3. Review the code in this method:

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

{

// Use the ChangePasswordDialog to change the user’s password

ChangePasswordDialog cpd = new ChangePasswordDialog();

* // Display the dialog
* if (cpd.ShowDialog().Value)
* {  
   // When the user closes the dialog by using the OK button,
* // the password should have been changed  
   // Display a message to confirm  
   MessageBox.Show("Password changed", "Password",
* MessageBoxButton.OK, MessageBoxImage.Information);  
   }
* }

1. In **Solution Explorer**, expand **Controls**, and then double-click **ChangePasswordDialog.xaml**.
2. In **Solution Explorer**, expand **ChangePasswordDialog.xaml** and then double-click **ChangePasswordDialog.xaml.cs**.
3. Review the code in the **ok\_Click** method:

// If the user clicks OK to change the password, validate the information

// that the user has provided

private void ok\_Click(object sender, RoutedEventArgs e)

{

// TODO: Exercise 2: Task 4a: Get the details of the current user

* // TODO: Exercise 2: Task 4b: Check that the old password is correct
* //for the current user
* // TODO: Exercise 2: Task 4c: Check that the new password and
* //confirm password fields are the same
* // TODO: Exercise 2: Task 4d: Attempt to change the password
* // If the password is not sufficiently complex, display an error message
* // Indicate that the data is valid  
   this.DialogResult = true;
* }

1. In the **Task List** window, double-click the **TODO: Exercise 2: Task 4a: Get the details of the current user** task.
2. In the code editor, in the blank line below this comment, type the following code:

* User currentUser;  
    
  if (SessionContext.UserRole == Role.Teacher)  
  {  
   currentUser = SessionContext.CurrentTeacher;  
  }  
  else  
  {  
   currentUser = SessionContext.CurrentStudent;  
  }

1. In the **Task List** window, double-click the **TODO: Exercise 2: Task 4b: Check that the old password is correct for the current user** task.
2. In the code editor, in the blank line below this comment, type the following code:

* string oldPwd = oldPassword.Password;  
    
  if (!currentUser.VerifyPassword(oldPwd))  
  {  
   MessageBox.Show("Old password is incorrect", "Error",
* MessageBoxButton.OK, MessageBoxImage.Error);  
   return;  
  }

1. In the **Task List** window, double-click the **TODO: Exercise 2: Task 4c: Check that the new password and confirm password fields are the same** task.
2. In the code editor, in the blank line below this comment, type the following code:

* string newPwd = newPassword.Password;  
  string confirmPwd = confirm.Password;  
    
  if (String.Compare(newPwd, confirmPwd) != 0)  
  {  
   MessageBox.Show("The new password and confirm password fields are different",
* "Error", MessageBoxButton.OK, MessageBoxImage.Error);  
   return;  
  }

1. In the **Task List** window, double-click the **TODO: Exercise 2: Task 4d: Attempt to change the password** task.
2. In the code editor, review the comment below this line, click at the end of the comment, press Enter, and then type the following code:

* if (!currentUser.SetPassword(newPwd))  
  {  
   MessageBox.Show("The new password is not sufficiently complex",
* "Error", MessageBoxButton.OK, MessageBoxImage.Error);  
   return**;**  
  }

#### Task 5: Run the application and test the change password functionality

1. On the **Build** menu, click **Build Solution**.
2. On the **Debug** menu, click **Start Without Debugging**.
3. When the application starts, in the **Username** text box, type **vallee**, in the **Password** text box, type **password99**, and then click **Log on**.
4. In the **The School of Fine Arts** window, click **Change Password**.
5. In the **Change Password Dialog** window, in the **Old Password** text box, type **password99**, in the **New Password** text box, type **pwd101**, in the **Confirm** text box, type **pwd101**, and then click **OK**.
6. Verify that the message **The new password is not sufficiently complex** is displayed, and then click **OK**.
7. In the **New Password** text box, type **password101**, in the **Confirm** text box, type **password101**, and then click **OK**.
8. Verify that the message **Password changed** is displayed, and then click **OK**.
9. Click **Log off**.
10. In the **Username** text box, type **vallee**, in the **Password** text box, type **password101**, and then click **Log on**.
11. Click **New Student**.
12. In the **New Student Details** window, in the **First Name** text box, type **Luka**, in the **Last Name** text box, type **Abrus**, in the **Password** text box, type **1234**, and then click **OK**.
13. Verify that the message **Password must be at least 6 characters long. Student not created** appears, and then click **OK**.
14. Click **New Student**.
15. In the **New Student Details** window, in the **First Name** text box, type **Luka**, in the **Last Name** text box, type **Abrus**, in the **Password** text box, type **abcdef**, and then click **OK**.
16. Click **Enroll Student**.
17. In the **Assign Student** window, verify that the student **Luka Abrus** appears.
18. Click **Close**.
19. Click **Log off**.
20. Close the application.
21. In **Visual Studio**, on the **File** menu, click **Close Solution**.

**Result:** After completing this exercise, you should have implemented a polymorphic method named **SetPassword** that changes behavior based on whether the user is a student or s teacher. You will also have modified the application to enable users to change their passwords.

### Exercise 3: Creating the ClassFullException Custom Exception

#### Task 1: Implement the ClassFullException class

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/Mod05/Labfiles/Starter/Exercise 3**, click **GradesPrototype.sln**, and then click **Open**.
3. In **Visual Studio**, on the **View** menu, click **Task List**.
4. In the **Task List** window, double-click the **TODO: Exercise 3: Task 1a: Add custom data: the name of the class that is full** task.
5. In the code editor, review the comment below this task, click at the end of the comment, press Enter, and then type the following code:

private string \_className;

public virtual string ClassName

{

get

{

return \_className;

}

}

1. In the **Task List** window, double-click the **TODO: Exercise 3: Task 1b: Delegate functionality for the common constructors directly to the Exception class** task.
2. In the code editor, click at the end of the comment, press Enter, and then type the following code:

public ClassFullException() { }

* public ClassFullException(string message) : base(message) { }
* public ClassFullException(string message, Exception inner) : base(message, inner) { }

1. In the **Task List** window, double-click the **TODO: Exercise 3: Task 1c: Add custom constructors that populate the \_className field** task.
2. In the code editor, review the comment below this task, click at the end of the comment, press Enter, and then type the following code:

public ClassFullException(string message,

string cls) : base(message)

{

\_className = cls;

}

* public ClassFullException(string message, string cls,
* Exception inner) : base(message, inner)
* {
* \_className = cls;
* }

#### Task 2: Throw and catch the ClassFullException

1. In the **Task List** window, double-click the **TODO: Exercise 3: Task 2a: Set the maximum class size for any teacher** task.
2. In the code editor, click at the end of the comment, press Enter, and then type the following code:

private const int MAX\_CLASS\_SIZE = 8;

1. In the **Task List** window, double-click the **TODO: Exercise 3: Task 2b: If the class is already full, then another student cannot be enrolled** task.
2. In the code editor, review the comment below this task, click at the end of the comment, press Enter, and then type the following code:

if (numStudents == MAX\_CLASS\_SIZE)

{

// Throw a ClassFullException and specify the class that is full

throw new ClassFullException("Class full: Unable to enroll student", Class);

}

1. In the **Task List** window, double-click the **TODO: Exercise 3: Task 2c: Catch and handle the ClassFullException** task.
2. In the code editor, click at the end of the comment, press Enter, and then type the following code:

catch (ClassFullException cfe)

{

MessageBox.Show(String.Format("{0}. Class: {1}", cfe.Message, cfe.ClassName),

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

}

#### Task 3: Build and test the solution

1. On the **Build** menu, click **Build Solution**.
2. On the **Debug** menu, click **Start Without Debugging**.
3. When the application starts, in the **Username** text box, type **vallee**, in the **Password** text box, type **password99**, and then click **Log on**.
4. In **The School of Fine Arts** window, click **New Student**.
5. In the **New Student Details** window, enter the following details, and then click **OK**.

|  |  |
| --- | --- |
| * **Field** | * **Value** |
| * First Name | * **Walter** |
| * Last Name | * **Harp** |
| * Password | * **abcdef** |

* **Note:** New students will not be listed in the main application window because this displays students in the **Users** class, and the new students are yet to be assigned to a class.

1. In the **The School of Fine Arts** window, click **New Student**.
2. In the **New Student Details** window, enter the following details, and then click **OK**.

|  |  |
| --- | --- |
| * **Field** | * **Value** |
| * First Name | * **Andrew** |
| * Last Name | * **Harris** |
| * Password | * **abcdef** |

1. In the **The School of Fine Arts** window, click **New Student**.
2. In the **New Student Details** window, enter the following details, and then click **OK**.

|  |  |
| --- | --- |
| * **Field** | * **Value** |
| * First Name | * **Toni** |
| * Last Name | * **Poe** |
| * Password | * **abcdef** |

1. In the **The School of Fine Arts** window, click **New Student**.
2. In the **New Student Details** window, enter the following details, and then click **OK**.

|  |  |
| --- | --- |
| * **Field** | * **Value** |
| * First Name | * **Ben** |
| * Last Name | * **Andrews** |
| * Password | * **abcdef** |

1. In the **The School of Fine Arts** window, click **Enroll Student**.
2. In the **Assign Student** window, click **Walter Harp**.
3. In the **Confirm** message box, click **Yes**.
4. In the **Assign Student** window, click **Andrew Harris**.
5. In the **Confirm** message box, click **Yes**.
6. In the **Assign Student** window, click **Toni Poe**.
7. In the **Confirm** message box, click **Yes**.
8. In the **Assign Student** window, click **Ben Andrews**.
9. In the **Confirm** message box, click **Yes**.
10. Verify that the message **Class full: Unable to enroll student: Class: 3C** is displayed, and then click **OK**.
11. In the **Assign Student** window, click **Close**.
12. Click **Log off**.
13. Close the application.
14. In **Visual Studio**, on the **File** menu, click **Close Solution**.

**Result:** After completing this exercise, you should have created a new custom exception class and used it to report when too many students are enrolled in a class.

©2018 Microsoft Corporation. All rights reserved.