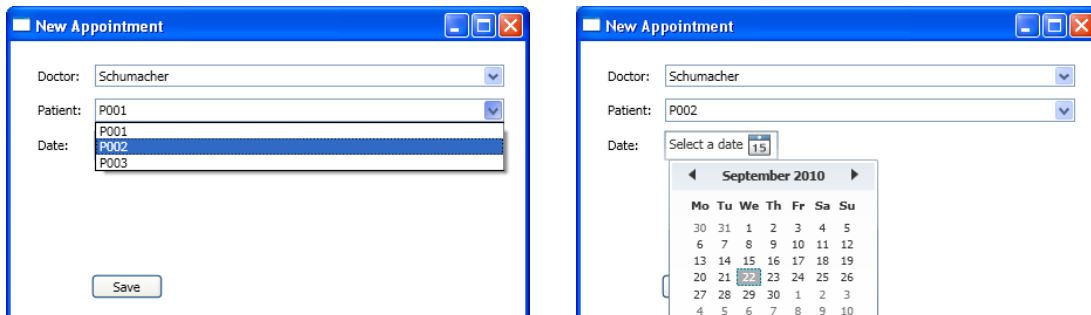


LAB 3: Recording Appointments (continued)

Getting started

In this lab you will continue to develop the patient appointment management program you started in lab 2. Recall that this program presents a form which allows the user to select a doctor (by name) and patient (by ID) from drop-down boxes and to select an appointment date. The user can then click a Save button to record the appointment.

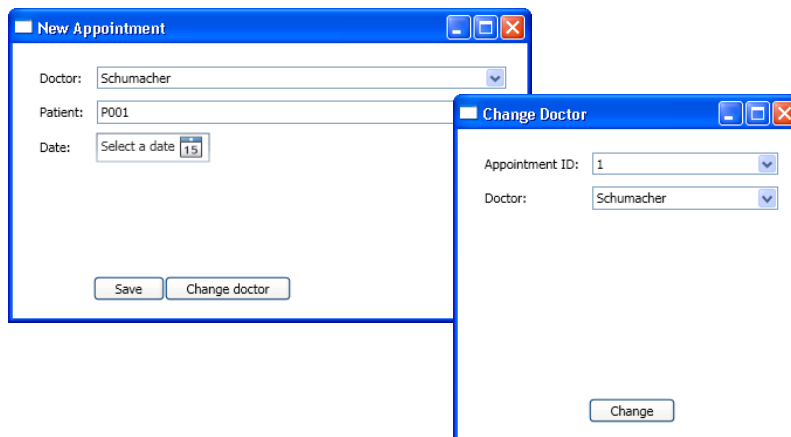


The program uses **C# classes** to **model the information** (doctors, patients, appointments) and **logic** (creating appointments). It also has a graphical user interface (GUI) to allow the user to interact with the program.

In this lab you will look **only at the code for the C# model classes**, and use the GUI simply to test the program. The GUI will be given to you – you will need to create the model classes. We will look at how the GUI is constructed later in the module.

Task 1: Changing the doctor for an appointment

The starting point for this task is a version of the Appointments project which includes *Doctor*, *Patient* and *Appointment* classes. This version has an additional window which opens when the Change Doctor button is clicked. This window lets the user select an appointment and select a new doctor for this appointment.



Open the project:

1. Start a VM which has Visual Studio 2010 installed, and start Visual Studio.
2. Download the file *lab3.zip* from Blackboard, and extract the contents.
3. In Visual Studio, select **File>Open Project..** and browse to find the file *Appointments.sln* inside the *task1/Appointments* folder, then click **Open**.
4. Build the project. You should get no errors as the project has all the classes it needs.

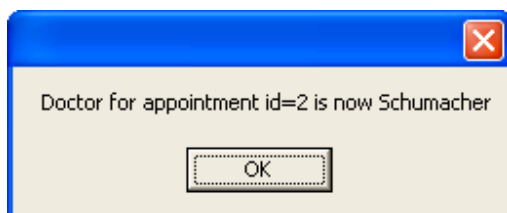
Test:

1. Run the program. You should see the **New Appointment** window.
2. Create the same two appointments you did in task 2 of lab 2:

Doctor: **Webber**
Patient: **P001**
Date: **current date**

Doctor: **Webber**
Patient: **P002**
Date: **current date**

3. Click **Change doctor**. You should see the **Change Doctor** window. Check that the top drop-down box lists two appointments, and that the lower drop-down box shows “Webber” for whichever one you select.
4. Select appointment 2, and change the selection of doctor to “Schumacher”. Click **Save**. You should see the following message:



5. Click OK and you should be returned to the New Appointment window. Click Change Doctor again, and check that the expected doctor name appears for each appointment ID you select in the Change Doctor window.

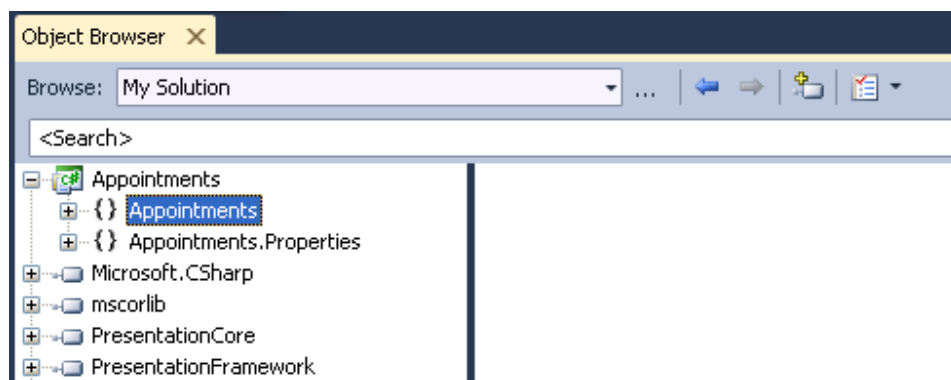
Questions:

- **Draw an object diagram** which shows the final state and associations of the objects involved in steps 2 to 5 above.
- What can you say about the association between *Appointment* and *Doctor*? Can you say the same about *Appointment* and *Patient*? Why?

Task 2: Commenting your code

In this task you will continue to work with the project you used in task 1.

1. Add XML comments for all public members of the *Doctor*, *Patient* and *Appointment* classes.
2. Open the Visual Studio object browser – select **View > Object Browser**.
3. Make sure the Object Browser is browsing **My Solution**, and find the *Doctor*, *Patient* and *Appointments* classes in the Appointments namespace. Check that the content of your XML comments are shown in the object browser.



Task 3: Using an enum for patient status

The starting point for this task is a version of the Appointments project which has a modification to the way *Patient* objects are created.

The *Status* property of *Patient* is a string value. In the test objects, the status is either “NHS” or “PRIVATE”. However, because it is a string, there is currently no way of making sure that when a *Patient* object is created the status value is valid. A convenient way to constrain a value to one of a specific set of possible values is to use an **enum**,

Open the project and make modifications:

1. In Visual Studio, select **File>Open Project..** and browse to find the file *Appointments.sln* inside the *task3/Appointments* folder, then click **Open**.
2. Open *Patients.cs*.
3. Above the *Patient* class, but within the *Appointments* namespace, add a public *enum* called *PatientStatus*, with three values, *NHS*, *PRIVATE* and *UNKNOWN*.

4. Change the type of the appropriate instance variable, property and constructor parameter from string to *PatientStatus*.
5. Modify the *RecordAppointment* method of *Appointment* to check patient status using the appropriate type. For example, where the value of Status was "PRIVATE" previously, it should be *PatientStatus.PRIVATE* in the modified version of *Appointment*.

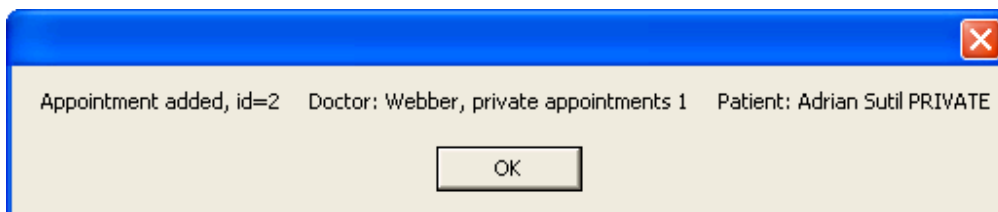
Test:

1. Run the program. You should see the **New Appointment** window.
2. Create the same two appointments you did in task 1:

Doctor: **Webber**
Patient: **P001**
Date: **current date**

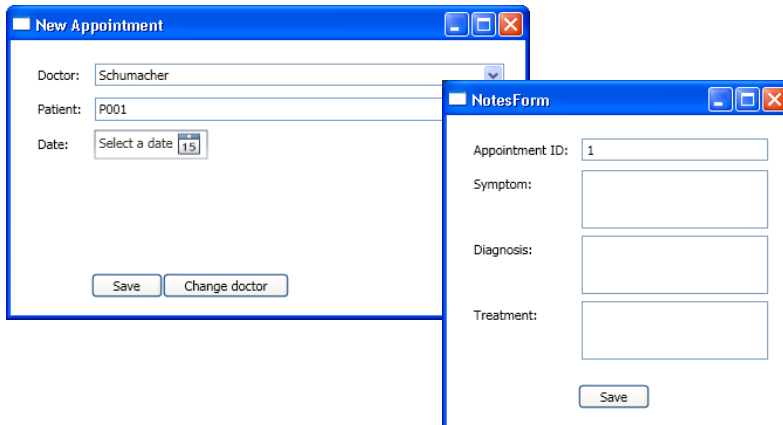
Doctor: **Webber**
Patient: **P002**
Date: **current date**

3. Check that the patient status is shown correctly, and that the correct number of private appointments has been recorded:



Task 4: Recording appointment notes

The starting point for this task is a version of the Appointments project which allows the user to store notes when an appointment is created. This version has an additional notes window which opens when the Save button is clicked.



In this task you will add a new **struct** to store the information entered in this form.

Open the project and make modifications:

1. In Visual Studio, select **File>Open Project..** and browse to find the file *Appointments.sln* inside the *task4/Appointments* folder, then click **Open**.
2. Right click on the project root and select **Add > Class..** Choose the **Class** template in the dialog box and name it *Note.cs*:
3. Open the file *Note.cs*. Change the class definition to a **struct** definition:

```
public struct Note
{
```

4. Add code to the *Note* struct so that it meets the following specification:

Note
-symptom : string
-diagnosis : string
-treatment : string

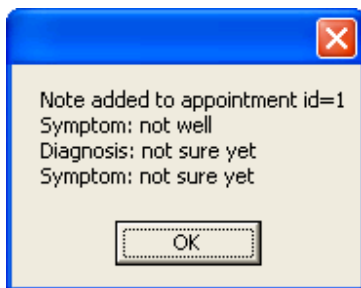
- All instance variables should be encapsulated in read/write properties
- There should be a constructor which takes three string parameters which are used to set the values of the instance variables.

Test:

1. Run the program. You should see the **New Appointment** window.
2. Create the following appointment:

Doctor: **Webber**
Patient: **P001**
Date: **current date**

The notes form should open. Enter some information and click Save. Check that the information you entered is shown in a message box similar to this:



Question: You could write *Note* as a **class** rather than a **struct**, and you would notice no significant difference when you run the program. There would be a difference, though - what would be different about what happens in the computer's memory when a method is called with a parameter of type *Note*? When might the difference between using structs and classes have a real effect on a program?

Programming challenge

The program allows the user to change the doctor for an appointment. However, remember that when an appointment is made for a patient with *status* = "PRIVATE", the doctors *numberOfPaidAppointments* value is incremented.

However, if you change the doctor for an appointment, the paid appointment should be credited to the new doctor instead of the original. The *numberOfPaidAppointments* value for the original doctor should be decremented, and the value for the new doctor incremented instead.

1. Review the code for the *Appointment* and *Doctor* classes to remind yourself how paid appointments are recorded. You should look at the *RecordAppointment* method of *Appointment*, the constructor of *Appointment* and the *RecordPaidAppointment* method of *Doctor*.
2. Modify the *Appointment* and *Doctor* class so that when the doctor for an appointment is changed, the record of the appointment (in *numberOfPaidAppointments*) is removed from the original doctor and added to the new doctor. You will need to do the following:
 - add a new *RemoveAppointment* method in *Appointment*
 - modify the setter for the *Doctor* property in *Appointment* so that the appropriate methods are called
 - add a new *RemovePaidAppointment* method in *Doctor*

The new methods for removing will be similar to the existing methods for recording.

3. Test your modified program.

Appendix: Test Objects

The following objects are created when the Appointments program starts up:

```
new Doctor("Schumacher", "ENT");
new Doctor("Barrichello", "Paediatrics");
new Doctor("Webber", "Surgery");
new Patient("Sebastien", "Vettel", "P001", "9/8/1990", "NHS");
new Patient("Adrian", "Sutil", "P002", "10/11/1988", "PRIVATE");
new Patient("Nico", "Rosberg", "P003", "12/3/1990", "PRIVATE");
```