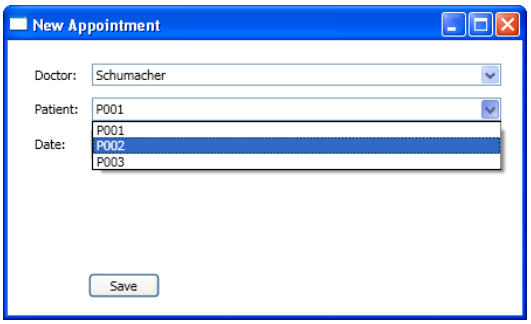
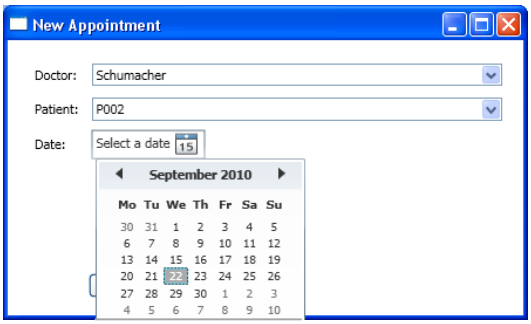
LAB 2: Recording Appointments

Getting started

In this lab you will complete a program which represents part of a system used by a doctors' surgery to manage appointments. This particular 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.

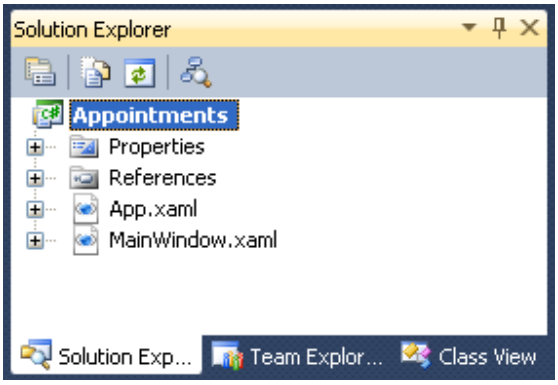
Task 1: Creating Doctor and Patient classes

Open the project:

1. Start a VM which has Visual Studio 2010 installed, and start Visual Studio.

2. Download the file *lab2.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**.



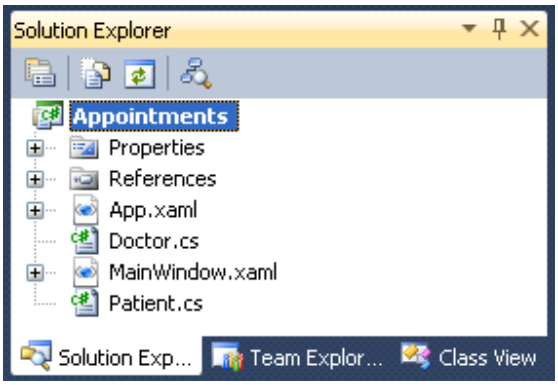
The files *App.xaml* and *MainWindow.xaml* define the user interface.

4. Build the project. You will get a large list of errors reported, because this program requires the classes which you haven't yet created, such as *Doctor*. Error messages will be similar to this:

```
The type or namespace name 'Doctor' could not be found (are you missing a using
directive or an assembly reference?)
```

Add model classes:

5. Right click on the project root and select **Add > Class..** Choose the **Class** template in the dialog box and name it *Doctor.cs*. Repeat this to add another class, *Patient.cs*. You should now be able to see these new items in Solution Explorer:



6. Open the file *Doctor.cs*. Add the key word **public** before the class name and save the file. Repeat for *Patient.cs*

```
public class Doctor
{
}
```

7. Add code to the *Doctor* class so that it meets the following specification:

Doctor
-name : string -speciality : string -numberOfPaidAppointments : int

- *name* and *speciality* instance variables should be encapsulated in read/write properties
- *numberOfPaidAppointments* instance variable should be encapsulated in a read-only property
- there should be a constructor which takes two string parameters which are used to set name and speciality. The value of *numberOfPaidAppointments* should be set to zero in the constructor.

8. Add code to the *Patient* class so that it meets the following specification:

Patient
-firstName : string -lastName : string -patientID : string -dateOfBirth : DateTime -status : string

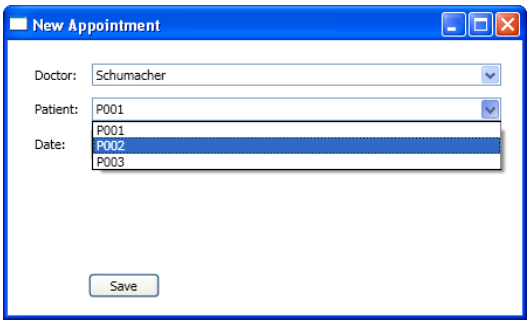
- all instance variables should be encapsulated in read/write properties
- there should be a constructor which takes five string parameters which are used to set all the instance variable values. The string representing date of birth will need to be converted within the constructor to type *DateTime*, using an appropriate method of the .NET framework ***Convert*** class. You should search for *Convert* in the MSDN Library to find documentation for this class.

Build and test:

9. Build the project. If you have created the classes correctly there should be no errors.

   If there are errors (and there probably will be to start with), use the error messages to try to find and correct the errors. Ask for help if you need it. Don't get frustrated – take a methodical approach to finding and correcting errors. Finding and correcting errors is an important skill in programming, and you become better at it the more experience you get.

10. Run the program. You should see the **New Appointment** window. There should be three doctors' names in the top drop-down box and three patient IDs in the lower drop-down box.



Note – test data

In this program a set of test objects is created when it starts up. In a real application, the data would be stored in and retrieved from a database. Details of the test objects are given at the end of this lab sheet.

Task 2: Creating the Appointment class

1. The starting point for this task is another version of the Appointments project which includes Doctor and Patient classes. This version requires an Appointment class in order to run.

2. In Visual Studio, select **File>Open Project**.. and browse to find the file *Appointments.sln* inside the *task2/Appointments* folder, then click **Open**.

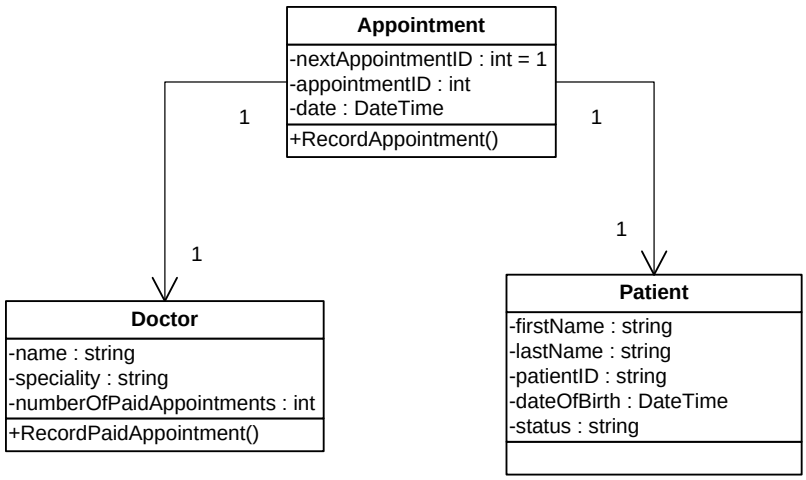
3. Build the project. You will get a list of errors reported. Error messages will be similar to this:

```
The type or namespace name 'Appointment' could not be found (are you missing a
using directive or an assembly reference?)
```

Add the model class:

4. Add a class called *Appointment* to your project. Add the key word **public** before the class name and save the file.

5. Add code to the *Appointment* class so that it meets the following specification:



- *nextAppointmentID* is **static**, with initial value = 1
- all instance variables should be encapsulated in read/write properties – what instance variables does this diagram imply? Remember that an association with another class can be implemented with an instance variable.
- there should be a constructor with parameters which are used to set the date and the associated *Doctor* and *Patient* objects. The value of *appointmentID* should be set to the current value of *nextAppointmentID,* which should then be incremented.

- The *RecordAppointment* method returns no value, and should take two parameters, of type *Doctor* and *Patient*. If the status of the patient is "PRIVATE" then the *RecordPaidAppointment* method of Doctor should be called.
- The *RecordAppointment* method should be called in the constructor of *Appointment*.

6. You will need to add the *RecordPaidAppointment* method to the *Doctor* class. This method returns no value and takes no parameters. It simply needs to increment the value of *numberOfPaidAppointments*.

Build and test:

7. Build the project. If you have created the classes correctly there should be no errors. Find and correct any errors which are reported.

8. Run the program. You should see the **New Appointment** window which should look the same as in task 1.

9. In the New Appointment window, select the following information and click Save:

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

   You should see the following message. Click OK to dismiss this message box.

10. In the New Appointment window, select the following information and click Save:

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

    You should see the following message. Check that you get the correct values for "id" and "private appointments".

Questions on task 2:

- **Draw an object diagram** which shows the objects involved in steps 9 and 10 above.(see Test Objects section at end of lab sheet for full attribute data)

- When the user clicks the Save button, the following steps happen, but not in this order. **Write these in the correct order**. You can review the code to help with this.

A	**A Patient object is retrieved from a data source**
B	**The Doctor object increments the value of its number of paid appointments**
C	**A new Appointment object is created and associated with the Doctor and Patient**
D	**A Doctor object is retrieved from a data source**
E	**The Appointment object sends a message to the Doctor object if the Patient is private**
F	**The Appointment object sends a message to the Patient object to get its status value**

- **Draw a sequence diagram**, similar to the example in chapter 2 of your lecture notes, which shows the messages which are passed between Appointment, Doctor and Patient objects in this process.

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");
```