# PE12 – Defining Class Members

1. Write code that defines a public class MyClass with the public virtual method GetString(). This method should return the string stored in the private field myString.
2. Make myString a write-only field by adding the property MyString. Note that a property can have a **get** command, or a **set** command, or both, where **get** is the "read" command and **set** is the "write" command. How would you make the property to be write-only? Write the code for the class including the MyString property.
3. Create a console application and derive the public class MyDerivedClass from MyClass. Override the GetString() method to return the string from the base class by using the base implementation of the method, but append the text " (output from the derived class)" to the returned string. Have the Main() instantiate a MyDerivedClass object and output the string returned by the GetString() method.

Remember how we could use **base** to call the base constructor.   
For example in the Shape example:

public Rectangle(double h, double w) : base(h,w)

The Rectangle constructor calls the constructor in the parent class by using **base**.

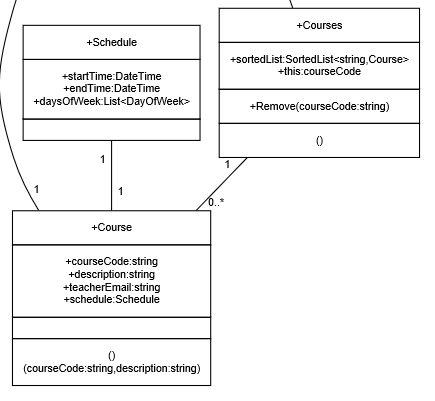
In the same way, for any derived class, you can call an accessible method in a parent class by using base.MethodName(), even if the method name is the same and being overridden.

GitHub URL:

1. Using Visual Studio Code, create the yuml file and generate the schUML SVG file for MyClass and MyDerivedClass.
2. Create a .NET Framework Class Library DLL called "CourseLib" based on the following schUML. Refer to the People class in PeopleLib to create the indexer property ("this : courseCode") and the Remove() method.

Ensure that daysOfWeek is defined as:

**List<DayOfWeek> daysOfWeek = new List<DayOfWeek>();**



Add this default constructor to the Courses class which will create a list of 100 courses:

public Courses()

{

Course thisCourse;

Schedule thisSchedule;

Random rand = new Random();

// generate courses IGME-200 through IGME-299

for (int i = 200; i < 300; ++i)

{

// use constructor to create new course object with code and description

thisCourse = new Course(($"IGME-{i}"), ($"Description for IGME-{i}"));

// create a new Schedule object

thisSchedule = new Schedule();

for (int dow = 0; dow < 7; ++dow)

{

// 50% chance of the class being on this day of week

if (rand.Next(0, 2) == 1)

{

// add to the daysOfWeek list

thisSchedule.daysOfWeek.Add((DayOfWeek)dow);

// select random hour of day

int nHour = rand.Next(0, 24);

// set start and end times of minute duration

// select fixed date to allow time calculations

thisSchedule.startTime = new DateTime(1, 1, 1, nHour, 0, 0);

thisSchedule.endTime = new DateTime(1, 1, 1, nHour, 50, 0);

}

}

// set the schedule for this course

thisCourse.schedule = thisSchedule;

// add this course to the SortedList

this[thisCourse.courseCode] = thisCourse;

}

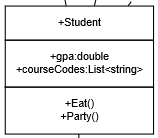
}

GitHub URL:

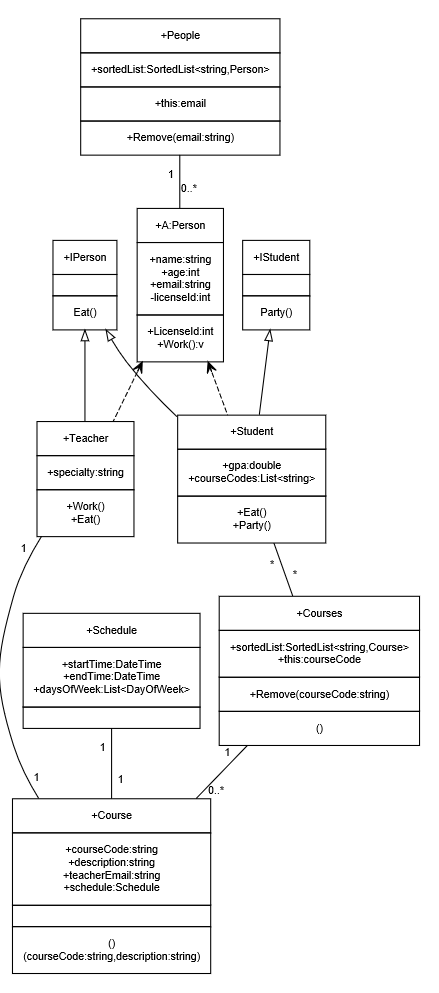
1. Modify the Student class to implement the following schUML. You will need to create a new PeopleLib.dll to be used by the changed PeopleApp down below.

Ensure courseCodes is defined as:

**List<string> courseCodes = new List<string>();**



1. After those changes, the whole class library looks like this:



1. Modify PeopleApp by adding both PeopleLib.dll and CourseLib.dll as references and add "Using CourseLib;" to the top of the source file. Note that when you add the references, you need to check the boxes for both dll's.
2. At the beginning of Main(), add: Courses courses = new Courses(); This will create our sorted list of 100 courses IGME-200 through IGME-299 with random schedules.
3. Start debugging and step into the application and F10 over the new line of code you added.
4. Right-click "courses" and choose Quickwatch from the menu. Maximize the window. Notice how each entry in the list is of type KeyValuePair.
5. Expand sortedList[1] (IGME-201). Expand the Value field (which is an instance of Course), write the class schedule below, and perhaps feel relieved that we don't meet at those times!

IGME-201 schedule:

1. Enhance EditPerson() to prompt for course codes and add them to the courseCodes list until an empty code is entered (ie. inputString.Length == 0). (Reminder: to add to the List use something like student.courseCodes.Add(string)).
2. For the "list" function, add inner foreach() loops that output the associated courses for each student, showing courseCode, courseDescription, the days of week and start and end time. Note that schedule.daysOfWeek is a List<DayOfWeek> that you will have to iterate through to list the days of week. Note that you can output the time formatted as "12:00PM" by using the format string "hh:mmtt" as follows:

Console.Write($"{thisCourse.schedule.startTime:hh:mmtt}");

or

Console.Write("{0:hh:mmtt}", thisCourse.schedule.startTime);

Example output (note that the course schedules are generated randomly every time, so yours will not match this example):

Add, Edit, Delete, List, Live, Quit => add

Person type (student/teacher) => student

Email () => me@gmail.com

Name () => David Schuh

Age (0)=> 21

Drivers License ID (0) => 123456

GPA (0)=> 3.99

Course code=> IGME-201

Course code=> IGME-210

Course code=> IGME-234

Course code=> IGME-256

Course code=> IGME-299

Course code=>

Add, Edit, Delete, List, Live, Quit => list

1: me@gmail.com | David Schuh | 21 | 123456 | 3.99

IGME-201 - Description for IGME-201 Sunday Monday Wednesday Friday 07:00AM - 07:50AM

IGME-210 - Description for IGME-210 Tuesday Wednesday Saturday 06:00AM - 06:50AM

IGME-234 - Description for IGME-234 Wednesday Thursday Friday Saturday 10:00AM - 10:50AM

IGME-256 - Description for IGME-256 Sunday Monday Tuesday 07:00AM - 07:50AM

IGME-299 - Description for IGME-299 Monday Wednesday Thursday 04:00PM - 04:50PM

Add, Edit, Delete, List, Live, Quit =>

GitHub URL:

## Submission

Upload this completed document, the schUML SVG for #4 and the GitHub URL's for #3, #5 and #7 to the corresponding MyCourses dropbox.