# Tutorial 6: Pair programming with C#

**Purpose:** To experience and practice the Agile development approach of *pair programming* while learning a little C# at the same time.

## A prototype-driven approach to completing Assignment 2

Your second, and largest, assignment involves the implementation in C# of a database-backed Windows application. It will thus require knowledge of the C# language, how to communicate with a database using C# (and LINQ) and how to design and construct GUIs using the Windows Presentation Foundation (WPF). These topics will be spread out over the coming weeks, so:

- While your early implementations *may* have a GUI, it will be simple and merely allow you to exercise the key functionality you have implemented. It should be possible to replace or improve the GUI easily. Early implementations may even be console applications.
- Until you have connected your application to the database, whatever class(es) you have for doing so will instead need to return artificial data.
- (A less obvious implication) While your work in the coming tutorials will contribute to your assignment's success, most of the work on the assignment will be done outside of class time.

## 1 Introduction to pair programming

Pair programming is an Agile software development technique (originally part of 'extreme programming') in which two programmers work together on a single computer. One takes on the role of **driver**, and is responsible for writing the code while the other, variously known as the **observer** or **navigator**, reviews each line of code as it is entered. The driver focuses on low-level (tactical) decisions about how to implement each piece of functionality, while the navigator considers the bigger picture (i.e., the overall strategy). The navigator can help pick up mistakes in the low-level code but, more importantly, helps to identify potential incompatibilities between the particular piece of code (such as a method) that the driver is writing and other parts of the system.

This wikiHow article is a good summary of the approach and includes some practical tips for how to do it well (on MyLO you will need to copy this link or open in a new tab, since it is insecure HTTP): http://www.wikihow.com/Pair-Program. **Skim read it now, then read it in full later.**

**For the rest of today's exercise form pairs, and swap roles of driver and navigator frequently.**

## 2 Programming task: Simple Employee Management System

Today's task is the development of a very simple console application for managing a collection of employees (although there won't be much functionality to actually manage them). The tasks below progress through:

- Creating a Console Application (we will see how to add a GUI later)
- Adding a new class to the project to represent an 'employee'
- Defining fields (attributes) in that class
- Adding a ToString method to Employee
- Generating some test data (sample Employee objects) and displaying them on the console
- Filtering the list of Employees

Given the wide range of prior programming experience in the unit, some of you will find the task simple, while others may not finish it during the tutorial time. A sample solution will be available through MyLO.

**Tip:** Compile and run your program often (as soon as it will actually *do* something). When building console applications, use Ctrl + F5 to run them as this will keep the console window open when the program has finished executing.

## 2.1 Getting started with Visual Studio

Create a new **Console Application** project via the menu item File | New Project…, then in the tree of options on the left navigate to Templates → Visual C#. Name the project whatever you like, since it is for practice, *not* directly for your assignment. The project name will become the namespace (i.e., package name in OO terminology) for the code you write. This will create a new source file called Program.cs with a main program in it, indicated by the presence of the Main() method. We'll come back to this file later.

## 2.2 Developing the application

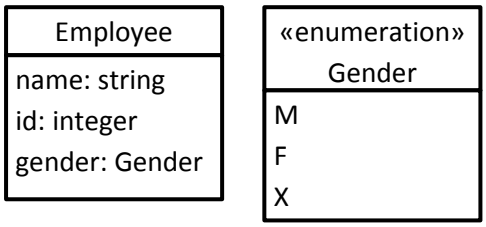
### 2.2.1 Adding the Employee class source file

From the **Project** menu select **Add Class…** Change the name to **Employee** (it will add the .cs file extension) and click **Add**.

### 2.2.2 Add fields to the Employee class

Each Employee in this system is described by the following attributes:

- A string-valued **name** (do not worry about separating family and given name now)
- An integer-valued **ID**
- The employee's **gender**, which is one of M, F or X (this may be a string or, if you like, an enum); these three alternatives align line with the options available on Australian passports




For today make these fields *public* (next tutorial we'll look at the options for protecting data, which are more varied in C# than languages such as Java). For instance, you might define:

```
public string name;
```

**Testing:** Back in Program.cs, define and instantiate a new Employee object using C#'s object initializer syntax (use any values you like for the fields):

```
Employee e = new Employee{ name="Jane", id=1, gender=Gender.F };
```

Then add a Console.WriteLine() statement to print the variable e. What's the result?

### 2.2.3 Adding a ToString method to Employee

As in languages like Java, C# classes all have a ToString() method to produce a string representation of the object for use in console and GUI applications. Override the inherited ToString() method by defining a method with the following header:

```
public override string ToString()
```

Make this method return the Employee's name, id and gender, separated by spaces or tab characters.

**Testing:** Rerun the program. What is the output now?

### 2.2.4 Generating some test data

The test data for today's practice will merely be hand-crafted and hard-coded into your program. First, in Main(), **declare a new List to hold Employee objects**, like this:

```
List<Employee> employees = new List<Employee>();
```

The List type is more flexible than an array and will grow as needed to hold the data added to it. It is also one of the *Generic* types, meaning that the class of objects it will hold, in this case Employee, is part of its declaration.

Next, **instantiate some additional Employee objects and add them to the list of employees**. Ensure that they have a mix of genders. **Optional:** define a new (static) method in Program.cs to generate this list of test data.

**Testing:** In Main(), add a `foreach` loop that prints each element of `employees` to the console.

**Tip:** Make a method for printing a list of Employee objects.

### 2.2.5 Filtering the list by gender

Write an additional method in Program with one of the following headers:

```
//if you used a string for gender
static List<Employee> FilterByGender(List<Employee> staff, string gender)
//or, if you defined an enum (called Gender here) for gender
static List<Employee> FilterByGender(List<Employee> staff, Gender gender)
```

This method will create a new list of Employee objects and add to it any Employees in staff whose gender matches the parameter `gender`. Later we'll see how to do this more easily using the language feature called LINQ, but for the moment just use a loop and if statement.

**Testing:** Try printing various subsets of Employee objects by using your new filtering function.

Note that in the assignment you may filter "Staff" objects in a similar way to here or by reloading the list of staff from the database.

## 3 If you have time left over…

Pick one of these extensions to today's work:

- Work through Module 04b Introduction to SVN. The module is small and predominantly activity-driven, so if you do it now you can ask your tutor questions as you go.
- Take a look at the next tutorial, in which you'll replace the simple fields of Employee with *properties* and add more classes to manage a collection of Employee objects.

## 4 Progress towards the assignment

You can practice some of today's skills by defining a new (console) project to which you can add basic implementations of the key entity classes representing staff, units, and other associated entities. You will have ample time to modify field visibility and other features later. You will also be able to add these source files to the WPF Application that will be your final assignment submission.