Object Oriented Software Development

3. Creating C# classes

C# classes

- Create an OO program by writing classes
- Need to understand structure and syntax of C# classes
- Programming language syntax is the set of rules which specify what is valid within the language
- C# syntax is similar to Java in many ways, but there are some important differences
- We will look in detail at example C# classes

C# example class code

- **ClassesDemo project**
- **Employee.cs**
- **Program.cs**
- **Location.cs**
- **Department.cs**
- **TimeSheet.cs**

Code blocks

- Related code enclosed in brackets { }
  - namespace
  - class
  - method
  - if/for/while
  - try/catch
- Each opening bracket

  {

  must have a matching closing bracket

  }

Code blocks

- Blocks are often nested
- Indent code inside blocks for readable code
- Makes structure of code much more understandable
- VS usually automatically indents, if there are no syntax errors in code
- Can force VS to format code with Edit > Format Document menu option

Namespaces

- We are creating a class called Employee
- Someone else might also create a class called Employee
- No problem...
- ...unless two classes with the same name become part of the same application
- Could happen if you include classes from a class library in your application

Code re-use

- It is very common to use classes in more than one program
- Encapsulation makes this straightforward in object oriented programming
- Each class is a self-contained component with a public interface
- Class libraries are groups of classes designed to be used in other programs
- Most programs will use .NET Framework library classes, and often other libraries

Namespaces

- Solution - namespaces
- Creating class within a namespace gives the class a (more) unique name

```
namespace ClassesDemo
{
    /// <summary>
    /// represents an employee
    /// </summary>
    public class Employee
    {
```

- Name of class is **ClassesDemo.Employee**
- This is called the **fully qualified name**

Namespaces

- Can define each class in a separate file or define multiple classes within a file
- Can define multiple classes within a namespace block in a file
- Can specify the same namespace in separate files
- Usually all the classes in a project belong to the same namespace

Using other namespaces

- Can use classes which are not part of your project
- May need to add Reference within your project
- Put **using** statement(s) at the top of your code file allows you to use the class name
- Otherwise would need to use fully qualified name

Using other namespaces

- Often need to include namespaces for .NET framework class library classes

```
using System;

namespace ClassesDemo
{
```

- Allows you to use the **System.Console** class
  - needed to print output in console applications

```
Console.WriteLine("Email address for Susan is {0}", emp2.Email);
```

- Can have as many usings as you need

Instance variables (fields)

- Define the attributes which each instance of the class (i.e. objects) can have
- Each object can have its own values for the instance variables
- **Declaring** an instance variable:
  - Specify access (public/private) for each field
  - Specify type
- By convention, name of variable is not capitalised

Instance variable declarations

Instance variables in Employee class

Note that type can be the name of another class in your application to set up "has-a" relationship

```
private string name;
private string username;
private Location currentLocation;
private string phoneNumber;
```

access modifier type name

Constants

- Constants
  - Value can't be changed once set
  - Use **const** key word

```
private const string EMAIL_SUFFIX = "@example.com";
```


Static variables

- Static variables
  - Same value for all instances of a class
  - Use **static** key word
  - Also know as **class variables**

```
public static int maxEntries;
```

in TimeSheet class

  - Can be accessed using name of class, without creating an instance

```
TimeSheet.maxEntries = 100;
```

  - Not constant, can be changed, change applies to all instances of class

Constructors

- Constructor is called when an object is created
- Used to initialise new object
- Constructor has same name as class
- Can specify parameters for constructor
- Can have multiple constructors with different parameter lists (overloading)
  - Allows objects to be initialised in different ways

Constructors

- Default constructor
  - No parameters
  - Implicit if no constructors defined
- Creating objects
- Use **new** keyword
  - Constructor selected according to parameters supplied
  - Compiler error if no matching constructor found

Constructors

```
public Employee(string name, string username,
    Location location, string phoneNumber)
{
    this.name = name;
    this.username = username;
    this.currentLocation = location;
    this.phoneNumber = phoneNumber;
}
```

```
Employee emp1 = new Employee("Michael", "michael", loc, "1234");
Employee emp2 = new Employee("Susan", "susan", loc, "4321");
Employee emp3 = new Employee();
```

```
public Employee()
{
    this.name = "default";
    this.username = "default";
    this.currentLocation = null;
    this.phoneNumber = "0000";
}
```

Methods

- A method defines a single action which an object can perform
- Method can return a value
- Method may need information (parameters)
- *Signature* is **method name** + **return type** + **parameter types**
- Can have methods in a class with same name but different signatures - overloading
- Code to perform action defined in code block

Cohesion of methods

- Good object oriented design aims for high cohesion
  - Each method should perform a single task
  - Name of method should describe what the task is
  - A method should perform a task related to the class it is in
- As a result, methods often contain relatively short segments of code
- Can be as short as a single statement, or can contain a more complex algorithm

Algorithms

- To write a method you need to devise an algorithm to solve the problem
- Set of instructions for carrying out the method's task
- Construct from:
  - Sequence – individual statements, in order
  - Selection
  - Iteration

Selection and iteration

- Useful programming constructs which may be needed within class methods
- Selection
  - Choosing from two or more actions to take based on the value of a variable
- Iteration
  - Repeating actions
  - Loops

Selection: if-else

```
Console.Write("Enter a character: ");
char c = (char)Console.Read();

if (Char.IsUpper(c))
{
    Console.WriteLine("The character is uppercase.");
}
else if (Char.IsLower(c))
{
    Console.WriteLine("The character is lowercase.");
}
else if (Char.IsDigit(c))
{
    Console.WriteLine("The character is a number.");
}
else
{
    Console.WriteLine("The character is not alphanumeric.");
}
```


Selection: switch

```
Console.WriteLine("Coffee sizes: 1=Small 2=Medium 3=Large");
Console.Write("Please enter your selection: ");
string s = Console.ReadLine();
int n = int.Parse(s);
int cost = 0;
switch (n)
{
    case 1:
        cost += 25;
        break;
    case 2:
        cost += 40;
        break;
    case 3:
        cost += 50;
        break;
    default:
        Console.WriteLine("Invalid selection.");
        break;
}
```

Iteration

- **for**

```
for (int i = 1; i <= 5; i++)
{
    Console.WriteLine(i);
}
```

- **while**

```
int n = 1;
while (n < 6)
{
    Console.WriteLine("Current value of n is {0}", n);
    n++;
}
```

- also have **do-while**, **foreach-in**

Method example

- RecordOvertime method of Employee class returns no value – return type is **void**
- Code for method includes an if-else construct

```
public void RecordOvertime(TimeSheet timeSheet, int hours,
    bool isWeekend)
{
    // send message to time sheet object to ask it to
    // record information
    if (isWeekend)
    {
        timeSheet.AddEntry(name, hours * 2);
    }
    else
    {
        timeSheet.AddEntry(name, hours);
    }
}
```


Calling methods

- Call method by specifying method name and parameters

```
emp1.RecordOvertime(ts, 5, true);
```

- This sends a message to Employee object *emp1*
- Note that code in RecordOvertime method of Employee sends message to TimeSheet object by calling its AddEntry method

Calling methods

- Set value of variable to return value if method return type is not void
- Example – calling Employee's TotalOvertime method

```
public int TotalOvertime(TimeSheet timeSheet)
{
```

```
int overTime = emp1.TotalOvertime(ts);
```


Static methods

- Class methods – don't need to create an instance to use method
- Example – IncreaseMaxEntriesBy method in TimeSheet class

```
public static void IncreaseMaxEntriesBy(int increment)
{
    TimeSheet.maxEntries += increment;
}
```

```
TimeSheet.IncreaseMaxEntriesBy(50);
```


Static methods

- Often used in utility classes which provide methods which can be called without an instance
- Example - **System.Math** framework library class
  - constants, e.g. **PI**
  - methods, e.g. **Sin**

```
double angle = Math.PI;
double result = Math.Sin(angle);
```

Main method

- The **Main** method is the **entry point** of an .exe program; it is where the program control starts and ends
- Main is declared inside a class or struct
- Main must be static and it should not be public
- Main can either have a void or int return type.
- The Main method can be declared with or without a string[] parameter that contains command-line arguments

Properties

- Classes can have attributes, or instance variables which are usually declared as private
- Sometimes need to provide a way for other classes to read or change the values of attributes
- Can write getter and setter methods
- C# provides a neater solution – **properties**
- Public properties **encapsulate** private instance variables

Properties

- Property (usually) encapsulates an instance variable
- Property is public
- By convention property names are capitalised
  - e.g. **name** variable – **Name** property
- Control access by providing **get**, **set** blocks
- Read-only access by providing get block only
- Get/set blocks usually simply read/set variable value, but can include other code

Employee class properties

Attribute	Property
name	**Name**: **get** only
username	**Username**: **get** only
location	*no property, changed by Move method*
phoneNumber	**PhoneNumber**: **get** and **set**
none	**Email**: **get**, depends on value of username attribute

- Note – this version of class defines Email as a property rather than a method

Using properties

- Properties are accessed using simple syntax
- Properties are not methods – no brackets or parameters

```
// use properties
string uname = emp2.Username;
Console.WriteLine("Email address for Susan is {0}", emp2.Email);
emp2.PhoneNumber = "5678";
```


Static properties

- Can encapsulate class variables in static properties
- Example – MaxEntries property in TimeSheet

```
private static int maxEntries;

public static int MaxEntries
{
    get { return TimeSheet.maxEntries; }
    set { TimeSheet.maxEntries = value; }
}
```

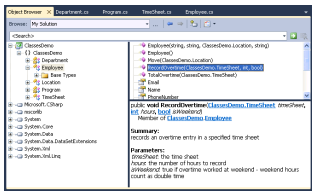
```
TimeSheet.MaxEntries = 100;
```

Comments

- Code comments
  - Comment line starts with //
  - To help programmer reading code
- XML comments
  - Comment line starts with ///
  - XML describes purpose, parameters, return types, etc
  - To help programmer reusing code
  - Used in documentation/VS object browser

XML comments

```
/// <summary>
/// records an overtime entry in a specified time sheet
/// </summary>
/// <param name="timeSheet">the time sheet</param>
/// <param name="hours">the number of hours to record</param>
/// <param name="isWeekend">true if overtime worked at weekend
/// weekend hours count as double time</param>
public void RecordOvertime(TimeSheet timeSheet, int hours,
    bool isWeekend)
```



Members

- The following are collectively known as the **members** of a class
  - Properties
  - Methods
  - Events (we'll look at these later)

Further reading

- C# classes can have some features which are not found in other OO languages
  - Events, delegates, indexers
  - We will look at some of these later on as we need them
  - MSDN has information on these
  - http://msdn.microsoft.com/en-us/library/67ef8sbd.aspx
- The following article is closely related to this chapter
  - http://www.aspfree.com/c/a/C-Sharp/C-Sharp-Classes-Explained/

Key OO concepts

- Code-reuse
- Encapsulation
- Information hiding

What's next?

- We will look in more detail at C# and .NET types and the way in which variables in a .NET program are stored