UNIVERSITY OF Hull

Programs and Libraries

Rob Miles

Department of Computer Science

08120 Programming 2

Simple C# Programs

```
class MyClass
{
    static data members
    static method members
    static void Main ()
    {
    }
}
```

- The first programs that we wrote were simple
- They contained a single class and static data
- They did not make any instances of any classes

Multi-Class C# Programs

```
class Account
{
}

class MyClass
{
    static void Main ()
    {
        Account a = new Account();
    }
}
```

- More advanced programs create instances of different classes
- We have seen this with Bank Accounts and WPF

Class instances in your programs

- A class instance (also called an *object*) will contain data and methods that let it do something for you
  - Bank Account
  - Alien Sprite
  - WPF Window
- The class itself will be described in a C# source file
- The class will be compiled to produce an output file that can be run on the target computer

C# Programs

```
csc MyProg.cs
```

- When we wrote our first program we put the entire program in a single source file
- The program was also made up of a single class which we could compile and run
- The output of the compiler in this situation is an *executable* file
- It has the language extension .exe

Compiled Programs

- In Microsoft .NET the output of a compilation is called an *assembly* file
- This contains a collection of classes and the resources that they use
  - An assembly file can contain images and sounds as well
- There are two kinds of assembly file
  - Executable assemblies contain a Main method that is used to start the program
  - Library assemblies just contain object descriptions and the compiled code for them

Library Assemblies

- Some parts of a program do not have a `Main` method
- They are simply objects that are created and used by the program when it runs
- It is often useful to be able to share such objects amongst a number of different programs
- We can do this by creating a library assembly
- This cannot be run as a program, but it can be used as an ingredient in other programs

An Account class

```
class Account
{
    private decimal accountBalance;

    public void PayInFunds (decimal amount)
    {
        accountBalance = accountBalance + amount;
    }
}
```

- This source file contains part of a class that could be used to manage a bank account
  - There will be lots of other members in the finished on
- There will not be a Main method though, since this is not a program

Compiling a Library Class

```
csc AccountClass.cs
error CS5001: Program 'c:\AccountClass.exe'
does not contain a static 'Main' method
suitable for an entry point
```

- If we try to compile this class we get an error because there is no `Main` method to run when the program starts
- The compiler is trying to make an executable assembly and will complain because it has no entry point

Compiling a Library Class

```
csc /target:library accountclass.cs
```

- We can tell the compiler to create a library assembly rather than an executable one
- The `/target:library` part of the command does this
  - It does not have to be typed in red
- The output from this compilation is a different file type
  - The file that is created is a *dynamic link library*
  - It has the language extension `.dll`
- This must **not** contain a `Main` method, since it is not a program

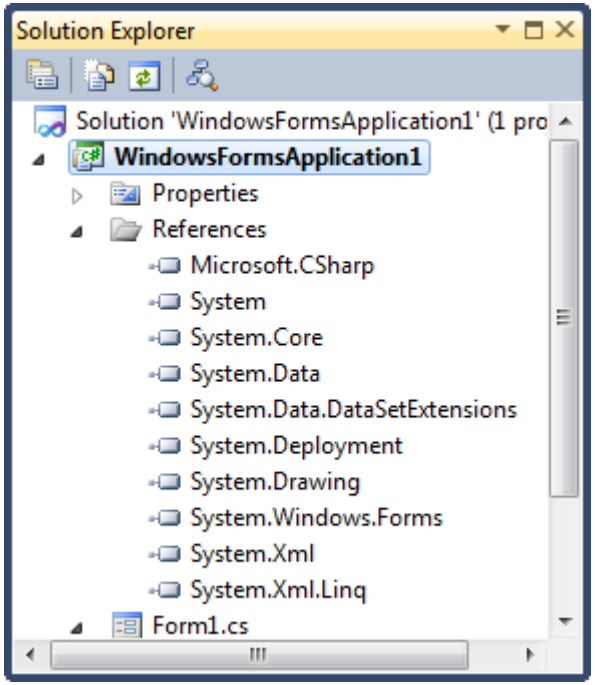
Using a Library Class

```
csc /reference:AccountClass.dll Bank.cs
```

- The C# compiler can be given a list of references to use when it creates a program
- This would make it possible for code in the Bank.cs file to make use of the Account object declared in the library file
- The classes in the Account library would be loaded and used when the program runs
- The run-time of the program is now spread over two different files
- But only one of them contains the Main method

Libraries in Visual Studio

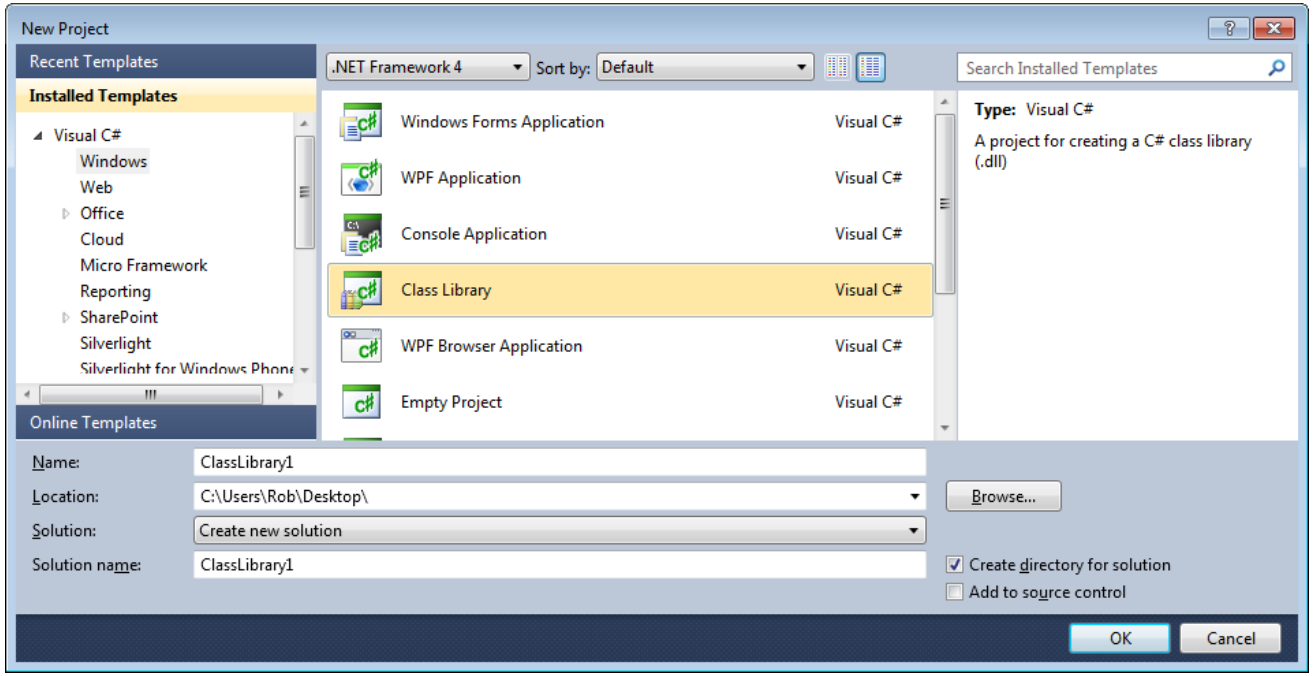
- Visual Studio will manage library references for us
- The Solution Explorer has a References tab which holds all the libraries that a program is using
- WPF Applications contain lots of references



System Libraries

- Some libraries are stored as part of the System
- This means that a single, central, version of that library can be used by all the programs on a particular computer
- All Forms applications share the same set of library files
- Visual Studio will link to these files and they will also be picked up when the program runs

User Libraries



- You can create “user” libraries in Visual Studio
- These produce a dll output that can be added to other projects

Namespaces

Identifiers

- Whenever we make something in our program we have to think of a name for it
- We should try to ensure that the name we select is meaningful:
  - `windowHeightInMetres`
  - `PayInFunds`
- These names exist in our program and they have to be unique

Identifier Clashes

- We have a problem when we are working with other people, or when we try to use code from a library
- We might have picked the same names as they have, leading to confusion
- There needs to be a way in which we can resolve this problem and make sure names don't clash

C# Namespaces

- C# provides a solution to this problem, it is called a *namespace*
- This is exactly what the name implies, a space for names, or more specifically, a place where particular names have meaning
- We can create our own namespaces for the programs that we write

Creating a namespace

```
namespace BankClasses
{
    class Account
    {
    }
    class Address
    {
    }
}
```

Using items in namespaces

- Items in the same namespace can just refer to each other directly
- Outside the namespace you have to use the "fully qualified name":

```
BankClasses.Account myAccount =
    new BankClasses.Account ();
```

Using `using`

- If you want to get things from a particular namespace you can put a `using` statement at the top of your program:

```
using AccountClasses;
...
Account a = new Account();
```

- Note that we have been adding "`using System`" to all our programs so we can use items in the System namespace

Sensible using

- If you add too many using directives it is hard to tell where something comes from when you use it in your program
- I therefore tend to use the fully qualified name a lot of the time
- It makes the code easier to follow

Name Clashes

- You can add as many using statements as you like at the top of your code
- If two namespaces contain an item with the same name you must use the fully qualified name to access that item so the compiler can tell which one to use
- Otherwise you will get an error

Nesting Namespaces

- A namespace can contain other namespaces
- This way you can build up a hierarchy of names
- The C# system does this, System.IO is the namespace which contains all the input/output classes

Namespaces and Libraries

- If you are using code from other libraries you have to make sure that the library is incorporated into the program when it is actually built
- The System library is always present, but if you make your own libraries you will have to make sure these are included when the program is built

Namespaces and source files

- A namespace can be spread over several source files
- A source file can contain some items in different namespaces
- To manage this you need to build a project to keep track of the different items
- Visual Studio is very good for this

Libraries and Namespaces Summary

- Libraries help you break a program into a number of smaller chunks
- It also allows you to reuse elements in multiple projects
- Namespaces allow you to manage the names of the objects in your programs so that they can be partitioned