University of Hull

Advanced Methods

C# Programming

Variables as parameters

```
int i = 3;
int j = 2;
int result = add (i, j);
```

- You can use variables as parameters to a method
- Their **value** is passed into the method

Variable values as parameters

```
static void changer (int i) {
    i = i + 1;
}
static void Main () {
    int q = 0;
    changer (q);
    Console.WriteLine (q);
}
```

- Because the **value** of q is passed into the method the program would print 0

Using references as parameters

```
static void changer (ref int i) {
    i = i + 1;
}
static void Main () {
    int q = 0;
    changer (ref q);
    Console.WriteLine (q);
}
```

- Because a **reference** to q is passed into the method the program would print 1

References and Security

- If I give a method a reference to a variable it has complete access to that variable
  - The method can both read and write the value in the parameter variable
- Sometimes it might not be meaningful (or sensible) for code in the method to read the parameter value
  - The method might be supposed to just set the parameter to a result

Using the out parameter type

```
static void readName ( out string name)
{
  Console.WriteLine("Enter name: ");
  name = Console.ReadLine() ;
}
```

- The `readName` method has the job of reading a name, it should not use the value supplied in name – therefore the parameter is defined as out
- The compiler will now also check that we actually set a value for name in the method

Methods and design

- Methods are an important tool for a programmer
- All programming languages support methods
- We use methods for two reasons
  - To allow us to reuse code
  - To allow us to break a problem down into smaller chunks

Code reuse

- Programmers are very careful to do the minimum amount of work
- This means that they will try to write as little code as possible
- Methods let them write the code once and then use it all over the system
- It also makes the code easier to test and update

Breaking down a problem

- Methods are also very useful to break a problem down into a series of smaller ones
- You can use the method headers to show what a method works on and what it needs to return
- You can also test each method individually

Method Summary

- A method is a block of code with a header that give the name, type and parameters the method has
- Parameters can be passed by value, reference or out
- The call of a method must agree exactly with its declaration

Classes and Methods

- Every program that we have written so far has been written in a class
- Now we are going to consider what a class actually is
- Later we will use classes as the building blocks for much larger programs

What is a class?

- A class is a collection of *members*
- A member of a class can either be data or a method
- The idea is that you put everything into a class to do with performing a particular task
- They are a fundamental part of C#
- Everything must exist inside a class

Variables in a block

```
{
    int i = 0;
}
Console.WriteLine ("i is :" + i);
```

- The variable `i` is declared inside the block
- It cannot be used anywhere outside the block
  - The code above would not compile because i is used outside the block where it is declared

The Scope of a Variable

```
{
    int i = 0;
}
Console.WriteLine ("i is :" + i);
```

- In programming terms we say that the *scope* of the variable is the block in which it is declared
- This is sometimes the body of a method

Data scope in a method

```
class MethodExample {
  public static void Main () {
    int i = 0;
    Console.WriteLine ("i is :" + i);
  }
}
```

- The variable `i` is declared inside `Main`
- It cannot be used outside the `Main` method body

Data in a class

```
class ClassExample {
  static int i = 0;
  public static void Main () {
    Console.WriteLine ("i is :" + i);
  }
}
```

- The variable `i` is a data member of the class
- It can be used by any method in the class

A Working Class

```
using System ;

class ClassExample {
  public static void Main () {
    Console.WriteLine ("Hello World");
  }
}
```

- The **`Main`** method is special because it provides an entry point for the program

Static class members

- Note that before every member of the class we put the keyword `static`
- This tells the compiler that the member is to be held as part of the class
- In this context `static` means always present
- It does **not** mean cannot be changed

Non-Static Class Members

- It is possible to make class members which are non-static
- These are held within an instance of the class
- We will do this when we start creating our own data types
- For now we will make all our data part of the class

Class data members

- We have seen how methods let us break a program into chunks
- But until now the only way to feed data into a method was by parameters
- A data member in a class can be shared by all the methods in a class
- Very useful for global data which is used throughout the program

Global Dangers

- The only problem with sharing data is that someone else might do something bad to it
- So you need to be careful what you make global inside your classes
- Deciding on how visible data is should be a part of the design process

Class Summary

- A class is the basic building block of C# programs
- It can contain data (variables) and methods
- At the moment we are making all the data and methods part of the class (i.e. static)
- This means that data in a class can be shared amongst the methods in it

Summary

- Methods allow us to break a program into a smaller chunks and reuse code
- They are created outside the Main method
- A method can receive values to work on (parameter) and return a result
- The compiler will make sure that a method is called correctly when it is used