

**C# PROGRAMMING FOUNDATIONS:
First Steps in Object Oriented
Programming (OOP)**



Copyright Notice

C# Foundations by Eric Matthews is licensed under a Creative Commons Attribution 3.0 Unported License. This license allows you to

- Copy, distribute and transmit the work
- Adapt the work
- Make commercial use of the work

Under the following conditions:

You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work

With the understanding that:

Waiver — Any of the above conditions can be waived if you get permission from the copyright holder.

Public Domain — Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.

Other Rights — In no way are any of the following rights affected by the license:

- Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;
- The author's moral rights;
- Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.

Notice — For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to one or more of the following web pages.

www.DeveloperGeekResources.com

www.EducationAnytime.com

Contents

Copyright Notice	2
<i>First Steps Into Object Oriented Programming</i>	4
Before we even discuss OOP	4
Before we even discuss OOP	4
REUSE and COMMUNICATION	5
OOP's Three Main Ideas Defined	6
Encapsulation	6
Inheritance	9
Polymorphism	10
C# Namespaces	11
C# and OOP have a lot of Class	11
What is a class?	11
What is an object?	11
Building our first class	11
Using our first class	12
Accessing the data in our class	12
Instantiating a class	13
Referring to data in a class	13
Getting Input from the Terminal when you run your program.....	15
static methods in a class	16
Creating and Using a Static Method	16
Adding a constructor	18
Constructor overloading	19
Method overloading.....	21
Variables and variable scope	24
Put into practice – Overloaded Methods	27
Quick inventory	28

FIRST STEPS INTO OBJECT ORIENTED PROGRAMMING

BEFORE WE EVEN DISCUSS OOP

This subject, unlike the previous one, will have a great deal of narrative and a great deal of the author's perspective as to this subject. It is important that you read all of the text and even more important that you understand the ideas that I am presenting. If you are like me your instinct will be to blow past the text and to get to the examples. I ask that you avoid this instinct and read and understand the text. To do Object Oriented Programming (from here on out referred to as OOP) you must learn to think in OOP. This type of programming is very different than the structured programming you are accustomed to.

A final note, these ideas and concepts are not easy to understand and they are even harder to master and apply. Do not get discouraged if this subject does not come to you on the first go around. Be persistent, and stay focused on learning and internalizing the three tenants of OOP. I learned them and so can you.

BEFORE WE EVEN DISCUSS OOP

This subject for me is the most critical in these modules. Now that you have a good understanding of basic C# syntax it is my intent to walk you through the doors of OOP. This is my slim window of opportunity to draw you in and get you to see the value of this way of programming, and to begin thinking of programming in terms of objects.

For you, the stakes may be even higher. Look at the main languages in vogue today; Java, C++, C#, Php, Java Script. These languages are all object oriented or object based. Keeping up with technology today takes more than learning the mechanics of a language. You must understand more than technical details. In order to be a good object oriented programmer you must understand the ideas and concepts.

I am not here to bash structured programming. In fact structured programming is alive and quite well in oop. It is and has been the case for a long time that every problem is not solved solely by creating objects. There are many oop zealots out there that did not get this memo. I recently stumbled across an interview where Java inventor James Gosling responded to the question "What is wrong with Java"

Though we are learning C# here the response is still 100% applicable!

This was his response...

"Probably the most pervasive mistake is not doing a tasteful job of object-oriented programming. There are these people who don't understand it at all, who just do some procedural kind of stuff and write their program as one huge class with a lot of methods in it; they try to do it in a C style. Then there are the people who go, "Ooo, objects! They're cool! Let's make gazillions of them!" Actually, that feels like almost a fault of the educational system, because if you look what lots of kids fresh out of college have been taught, it's, "Objects are good, define lots of them!" So you get programs with zillions of little adapters, with methods that are one or two lines long, and you start profiling them and they're spending all of their time doing method dispatching. Then the only kinds of optimizing compilers that do any good

are the ones that spend almost all of their time trying to eliminate method dispatching and doing lots of full inlining.”

Structured programming is very much alive and viable within C#. I am currently working on a UI that allows C# scripting against screen display elements. The programming here is almost solely structured programming.

A main goal of computer science and technology is and has always been to evolve and build a better widget. SQL was created to better deal with entity relationships, something Cobol did not do well. C++, as I said before is an iteration of C, an attempt to build a better C. Java was created to better handle programming tasks that C++ was giving a developer of the language fits with. C# brings its own oop spin into the mix. Whether these languages actually achieve their objective is certainly a subject to debate. Whether they “did or did not” is not my point. It is the fact that they set out to build a better language that is my point.

I do not want to make this a history lesson about computer science. I do want to stress this one point. Those of us passionate about software engineering have one thing we can generally agree on. Good software engineering at its foundation is about achieving (or knowing when to and when not to achieve) two central principles...

REUSE and COMMUNICATION

One of the best books on this subject (and many others as well) is a book titled “The Mythical Man-Month” by Fredrick Brooks. I feel this should be required reading by anyone in our profession. I remember the first time I read this in the mid-nineties thinking he had used my company to do his research. I was floored to find that the book had been written in 1975. It is sad that even today, even with the advent of OOP, and the advancements in computer science, that the issues and problems this man spoke of in 1975 are still present today (note I originally wrote this paragraph in 2001 and sadly as I reuse it in 2011 I report that little has change). Why is this?

I myself am not wise enough to offer a full answer here. Nor would addressing an answer in the full context of this piece be appropriate for the subject at hand. I would encourage you to pick-up “The Mythical Man-Month” and read it though. It is quite enlightening!

I do want to offer my perspective on reuse and communication. These are concepts I believe to be very important prior to discussing the basics of OOP. One of the failures in software engineering is that we are so focused on writing code that we sometimes forget the plans and organization required to write and maintain good code. You can have a pile of lumber and materials on your building site but without a set of blueprints it is unlikely that you will build anything that comes close to approximating a house. How many times have you brought something home from the hardware store, like a barbecue, looked at the picture on the box and figured, “instructions - I don’t need no stinking instructions”. Only to find that you did need them and even then you ended up with a few leftover hardware items. Well software is not any different in this respect. Because it is less visual under the hood, it is even more important that we define a reasonable blueprint or means of communicating what we have done and why. And if this is so obvious to all of us, why aren’t many organizations taking the time to do it?

As for reuse, I will venture out on a limb and say that every programmer has at some time in his or her career employed some aspect of reuse. You have no doubt written a piece of code that was very complex and took you a great deal of time and agony to accomplish. Down the road you were faced

with solving a similar problem and instead of starting from scratch you used your prior solution as a starting point to solve the new problem. And, how many times have you cut-n-pasted a routine from one program into another. Well guess what, you were practicing a form of reuse. Now this is a pretty crude form that is not likely to win you any accolades in the world of computer science. But hey, it was reuse! And why did you do this?

You employ reuse because it saves you time and you do not have to reinvent the wheel.

OOP at its most foundational level exists to support code reuse. Reuse is something you are already familiar with so all we really need to delve into is how Java deals with reuse.

There are three important ideas to OOP. They are

- **Encapsulation**
- **Inheritance**
- **Polymorphism**

OOP'S THREE MAIN IDEAS DEFINED

At this point we are only going to discuss these ideas without respect to code. I want you to get a very high level understanding of these three principles. We will take these ideas throughout this subject and in other parts of the book and expand upon them. It is important that you “buy-into” and embody these three ideas as they are the backbone of OOP. Most stuff I have read on OOP only talks about the benefits of OOP. This bothers me and I plan on addressing the tough questions as well.

Encapsulation

Encapsulation is a principle that allows a software engineer to do a couple of things. First, it allows data and code to live together. This by itself will not be a new concept for most of you. However, if you are a Cobol programmer this is indeed a very new concept. In Cobol, data is defined separate (Data Division) from the program code (Procedure Division). This makes reuse very difficult. By placing data and program code together we have the ability to more easily reuse code. Of course this same principle can be achieved in most other languages. So what is so special about encapsulation? (Keep this question in the back of your hat for the moment).

Second, by grouping data and code together we can potentially hide our code from other programmers, but allow them to reuse it. This sounds a bit unusual so let me further explain how this is done. Suppose I am building an application and want to globally define an address and all the functions one might want to do with the address. I might define the address data as follows:

```
Addressee
Street1
Street2
City
State
ZIP Code
```

I might then define the address functions as follows:

- Write an address to the address table

- Retrieve an address from the address table given partial address information
- Given the City and State determine and fill-out the zip code
- Given the zip code determine and fill-in the state and give the user a list of cities to choose from
- Provide a list of all addressees given a city or zip code
- Check for and prevent duplicate addresses in the address table
- Etc...

Now further suppose that I want to make this the standard anytime the application needs to deal with address. I also want to keep the code from being changed by just anyone in the shop. These two requirements seem to oppose one another. How can I write code that other developers can reuse but cannot access?

With Encapsulation I can write code whose access is restricted, but I can provide an interface that exposes the functionality of what I wrote so that it can be used by my fellow developers. This may still be a foreign idea so let's look at an example of this.

Source: getTime.cs

```
using System;

namespace GetCurrentTime
{
    class GetTime
    {
        public static void Main(string[] args)
        {
            Console.WriteLine(DateTime.Now.ToShortTimeString());

            Console.Write("Press any key to continue . . . ");
            Console.ReadKey(true);
        }
    }
}
```

The following code returns today's date.



By calling `DateTime.Now.ToShortTimeString()` we can get the current time. In C# (and other OOP languages) `ToShortTimeString()` is referred to as a METHOD. A method is the oop name for a function or subroutine. The example I have used is calling a method that is part of the C# language. There are many such methods for dealing with and manipulating date and time. We are not going to get caught up at this time going into all the details of the syntax.

Ponder for a moment the concept of encapsulation. It can be quite powerful. I can call some predefined programming service and get a predictable result without having a clue as to how the code is

implemented. I have read a number of books and dissertations on OOP and few of them address what I think are some of the most important things with respect to this subject.

1. How do I know what methods there are to reuse?

This is the number one question in my mind with respect to OOP or any language that practices reuse. If I am a busy developer, how do I know what is available for me to reuse so that I do not reinvent what has already been done? C# comes with thousands of pages of documentation that covers thousands of different methods that can be reused. How do I know what there is to reuse?

As a subject in this book, we will be examining some of these built-in methods. My goal will not be to teach you all of them, but to show you how to access them. Once you understand how to access one method you can pretty much access any method.

2. Can I rely on getting the same result if the method changes?

You better be able to!

Since you may one day be writing encapsulated code that you want others to reuse, but not be able to access and change please carefully listen to the following. If you are writing code that others are going to reuse it is of vast importance that you maintain your code in such a manner that if you change it, it DOES NOT have any impact on the other developers' code!

This is not just a concept related to OOP. Imagine if you upgrade to Windows 7 and none of the programs you run under Windows XP worked. To say you would be "pissed off" would be an understatement. It is your responsibility to ensure that the methods you write for others remain backwards compatible (at all costs and to the best of your ability). And if for whatever reason you cannot, you had better plan on some communication that allows your "customers" time to deal with your changes.

Ever wonder why each new release of Windows seems to double in size. It is because they are typically very good at practicing this principle.

3. Should all code be written so that it can be reused?

Of course not! Sometimes when studying the subject of OOP you get the idea that all code should be written to be reused. **This is an absolutely insane notion**, and if for no other reason here is why. Assume I have a staff of 50 programmers. Given that each programmer not only needs to understand their own code and write for reuse, but they also need to be aware of what the other 49 programmers are doing and what they have to reuse. Let's get off the idealism for a moment and examine this. This would require a level of communication that is not only impractical to achieve, but would require so much of each developers time that any savings in the code they wrote for reuse would be lost in all of the time they were spending trying to figure out what the others were doing. Pleeeeeeease, I love idealism, but it must be tempered with some sense of realism.

Then what should be reused? Common IO routines should be written with reuse in mind. How many times does one need to write a program that retrieves an order by invoice number? Other good candidates are general-purpose routines that upshift data, calculate dates or perform string manipulation. While there is no magic formula that can be applied with respect to this subject, code that is needed by more than one program should at least be given some degree of consideration for reuse. I would personally give greater consideration to reusing a routine that was going to be used in six

programs more than one that was only needed in two. Knowing where you will need to use the same code is more of an art than a science, and past experience and prior development will generally give you a glimpse at the answer. Ideally it is better to be proactive than retrospective with regard to this subject. This is easier said than done.

4. How does this idea affect how Development groups operate?

Regardless of the programming paradigm, if one is going to employ a strategy that heavily includes reuse and wants to create code “object” or business services or whatever the heck you want call them, then some thought must be given to how you organize the development shop. Who is going to be focused on the reuse effort? Does your shop have well defined roles for Software Architects, Software Designers, Software Engineers, and for Programmers? Or, are these just titles that are thrown around like paper in a windstorm?

Generally the Senior Developers and Architects will be the ones that need to drive the analysis and efforts that result in reuse. One needs to have a very profound and in depth understanding of the project to be able to determine what can and cannot be reused. One also needs strong communication and leadership to document and get this information into the working hands of everyone else. Organizations that do not spend more time planning to code reuse than actually coding are usually doomed to fail. Personally, I would never let anyone lead a reuse project that does not have some prior experience, be it success or failure (assuming they learned valuable lessons from their failure).

5. How do other developers know what there is to reuse?

This is a really good question that you better have a really good answer for or no one will be reusing what has been built for reuse. Considering that it takes longer to write reusable code, you will not have gained a whole lot except spending more time for little gain.

You may have noticed that I have been using the term reuse more than I have been using the term encapsulation. For me these terms are interchangeable in the context of this discussion. It is my contention that encapsulation is just a more narrowly defined implementation of reuse that falls under the reuse umbrella.

Inheritance

You can relax. I gave encapsulation more coverage than I plan on giving inheritance and polymorphism.

Inheritance is an idea that somewhat extends the idea of encapsulation by answering the following question. **What can I do if the code I am using does not fully meet my needs?** Inheritance allows the programmer to reuse and to extend or modify a piece of code that has already been written. I am not going to go into a lot of detail now on this subject but let me offer you this analogy. I might create a generic widget called a saw. All saws have sharp teeth. All saws cut wood. There are many kinds of saws out there (circular, jig, band, hand-crosscut, hand-rip, coping, table, etc...). The concept of Inheritance would work like this. I carefully analyze all of the attributes/properties (how many teeth per inch) and methods (cuts wood) that are common to all saws and create a generic saw object. Then I would inherit this object and tweak it further to satisfy the requirements of the particular saw I am dealing with. This in a nutshell is the concept of inheritance.

While inheritance is a very neat concept it is, in reality, one that requires even more attention than does encapsulation. At least with encapsulation you can call a method as a “black box” and expect a specific outcome. With inheritance you have to have a better understanding of all of the specific behaviors of the methods if you are to tweak them. It is this author’s opinion that inheritance requires a greater degree of documentation and education than does encapsulation. It most certainly creates a higher degree of code abstraction.

Polymorphism

This is probably the most abstract and ill explained idea of all the OOP concepts. About ten years ago I interviewed a very experienced C++ programmer for a job and while he eloquently defined the previous two terms and provided practical examples of implementing each, he passed on defining this term. Fast forward to today. The one thing you find missing or in short supply on the subject of oop is polymorphism. This is not to imply that it is not a powerful facet of oop if you understand and can implement it in your design.

While in a sense abstract, I think that this idea is made to be harder to understand than it really is. First let’s start by dissecting the word. Poly means *more than one*; and morphic *pertains to form*. Now as the definition of morphic may still be nebulous for some, let’s look to an example.

Consider a thermostat. A thermostat could be a polymorphic device in the sense that it displays temperature and is used to regulate temperature (hmmmm, these sound like methods to me). It is however not necessarily polymorphic because in most cases a thermostat is dependent on the type of heating or cooling system attached to it (pertains to specific function, not form). So what types of heating systems are out there: Natural gas, Wood burning, Electric, Oil, to name just a few (and I forgot the cooling systems).

A polymorphic thermostat is one that will work regardless of the heating or cooling system that it is attached to. In other words, **a polymorphic thermostat has the innate ability to recognize what it is interfaced to!** Polymorphism is a pretty hard thing to achieve in terms of applied engineering. A trip to your local hardware store will verify this. You can find thermostats that can handle electric and gas, but most thermostats are specific to the system it is going to be attached to.

In software development there are, many times, where if understood, situations where polymorphism could provide an elegant solution to our problems. Consider something as simple as an address. The address is not such a simple data structure in software engineering. Does a U.S. address look like a Canadian or a European Address? Is the address for a person or a business?

Not so simple after all. This is a problem where a polymorphic solution would be quite elegant. Build an interface that can understand what type of address it is.

Now having offered the explanation and example, you will many times hear this term defined as “**one interface, multiple implementations**”.

Sounds like magic but you will soon see that it is not. You might be thinking why not implement the solution using inheritance. You certainly could, and if this thought hit you, great, you are starting to think in OOP. If you are at this point unclear as to the difference between inheritance and polymorphism then you should go back and reread the discussions on both topics.

You will be exposed to this concept in code later in this section.

C# NAMESPACES

In C# a namespace is a logical container that holds one or more classes. namespaces are not required.

C# AND OOP HAVE A LOT OF CLASS

If you have looked closely at the code you have been writing and the examples that I have been presenting you, you will have noticed that all of our code starts with this...

```
class <name of class>
{
}
}
```

...syntax. This is because in C# pretty much everything is a class or part of a class.

What is a class?

A class is a blueprint used to create objects. Objects are created through instantiation of a class. Instantiation means create an instance of. A class is **not** the source code! The C# source is just the source code. It is not a class. It only becomes a class once we compile it!

What is an object?

An object is an instantiation of a class. In C#, one does not run a class. Remember a class is just a blueprint or template for an object. One runs and executes objects (or instances of a class).

Building our first class

Before jumping straight into writing the code for our first class, it would be a good idea to determine, at a high level, a set of objectives we need to meet in order to build a class.

- Define and create the data elements that the class will need
- Define the method that the class will use

We can (and will) expand on this later. But for now these two bullets set enough direction for us to build our own class.

Let's build a class called phone. The phone class will have three data elements (area code, exchange, extension) and will have two methods. Display phone in U.S format and display phone in European format.

Source phone.cs

```
class phone {
    public String area_cde = " ";
    public String exchange = " ";
    public String extension = " ";

    public void phone_nbr_us() {
        String phn = "";
        phn = (area_cde + "-" + exchange + "-" + extension);
        Console.WriteLine(phn);
    }

    public void phone_nbr_europe() {
        String phn = "";
        phn = (area_cde + "." + exchange + "." + extension);
        Console.WriteLine(phn);
    }
} //end phone class
```

First we define a class named “phone”. I then create the three pieces of data that represent a phone number. While there is no requirement to initialize the data, I do anyway. I prefer to initialize all my data. Here I set each field to a single space.

Next I create the methods. The first method I name “phone_nbr_us”. Since this method is not going to return anything I must make it type “void”. In the method I create a string named “phn”. I then assign phn the result of my three phone fields concatenated by the “-“. Finally I print “phn”.

My second method is identical to the first except my method name has changed and I am using the “.” to concatenate the number. All said the code is pretty straightforward.

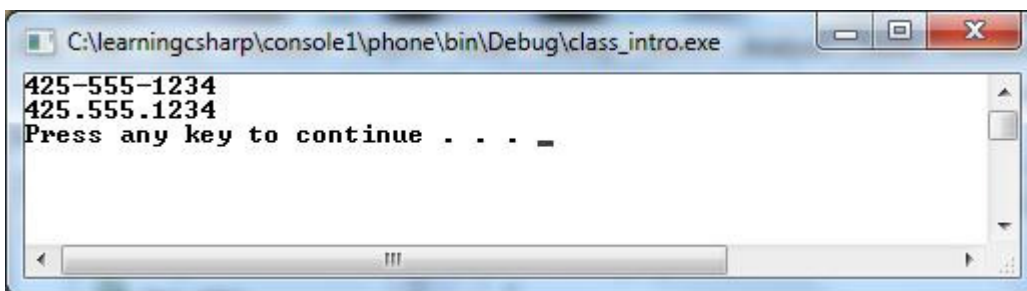
Next I need to compile the class.

Using our first class

The class I have created is not a standalone program like the ones we have created so far. We now need to create a C# program (class) or two that will use the “phone” class.

Accessing the data in our class

Source use_phone.cs



```
public class use_phone
```

```
{
public static void Main(string[] args)
{
    class_intoduction.phone employee_homephone = new class_intoduction.phone();
    employee_homephone.area_cde = "425";
    employee_homephone.exchange = "555";
    employee_homephone.extension = "1234";
    employee_homephone.phone_nbr_us();
    Console.Write("Press any key to continue . . . ");
    Console.ReadKey(true);
}
}
```

Not a great deal of code here to review, but there are a few key nuggets of gold to cover. First, let's examine the line...

Instantiating a class

```
class_intoduction.phone employee_homephone = new class_intoduction.phone();
```

On the left side of the expression we are creating a handle (or alias) named "employee_homephone" for the class "phone" that we want to instantiate. On the right side of the expression we are instantiating the "phone" class by using the key word **new**. If the phone class accepted parameters we would have to pass these parameters to the class through the parentheses "phone()". But our Phone class does not accept any parameters. However, we still require the parentheses. At this point, the general syntax for instantiating a class looks like. Also, note that I had to reference the namespace (*class_intoduction*) as the class I am creating here is not part of that namespace.

Syntax

```
<namespace>.<class_name> <alias_name> = <class_name> (<optional_arguments>) ;
```

Note: This type of method is referred to in oop an "Instance Method".

Referring to data in a class

Since we have instantiated, or made a copy of the phone class, we now have the ability to access both the data and methods that are exposed to us in the class. In the code below it is our intent to set the values for our phone number,

```
employee_homephone.area_cde = "422";
employee_homephone.exchange = "255";
employee_homephone.extension = "1903";
```

Notice that we refer to the data by using the alias (or handle) we made for the class.

Syntax

```
<alias_name>.<data_name_from_class>
```

Finally, after setting the data we print the values we set by calling the appropriate method.

```
employee_homephone.phone_nbr_us();
```

So what if we want to create two phone numbers? The answer is very easy. All we need to do is instantiate and use a second class.

Source use_phone2.cs

```
public class use_phone2
{
    public static void Main(string[] args)
    {
        //home phone
        class_introduction.phone employee_homephone = new class_introduction.phone();
        employee_homephone.area_cde = "425";
        employee_homephone.exchange = "555";
        employee_homephone.extension = "1234";
        Console.WriteLine("Home phone is: ");
        employee_homephone.phone_nbr_us();
        //cell phone
        class_introduction.phone employee_cellphone = new class_introduction.phone();
        employee_cellphone.area_cde = "360";
        employee_cellphone.exchange = "555";
        employee_cellphone.extension = "4321";
        Console.WriteLine("Cell phone is: ");
        employee_cellphone.phone_nbr_us();

        Console.Write("Press any key to continue . . . ");
        Console.ReadKey(true);
    }
}
```

This is pretty slick if you think about it. We have created a “phone” class and have placed no restrictions as to how many phone numbers one can have in their program. We have also not only defined the schema (data), but we have also associated code with the data.

There is a lot more information we need to cover on classes but this is a good time to stop and put into practice what has been covered so far.

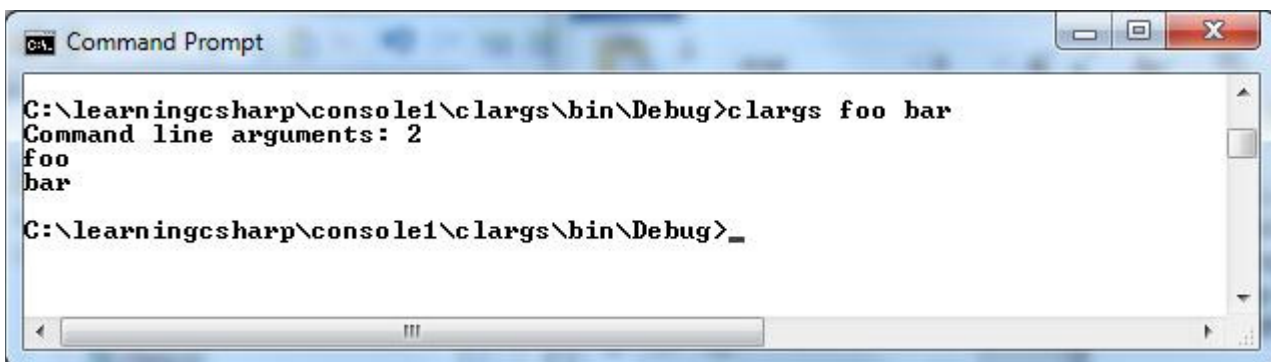
GETTING INPUT FROM THE TERMINAL WHEN YOU RUN YOUR PROGRAM

This may seem like an odd spot to cover this subject. The reason I have placed this topic here is that as I expand the discussion of classes I am going to increase the complexity of the examples and exercises such that we are going to want to test our classes without having to hardcode the values in our test program (the program/class that instantiates the class we create). You saw this topic covered in the module “Writing Hello World Programs”. Here we go again. This time I am showing you how to work with more than one argument.

Source clarg.cs

```
using System;
/*
 * NOTE to run this program you will need to open the command line prompt and go
 * directly to the executable.
 * \learningcsharp\console1\clargs\bin\Debug>clargs foo bar
 */
namespace clargs
{
    class clargs
    {
        public static void Main(string[] args)
        {
            int cnt = args.Length;
            Console.WriteLine("Command line arguments: " + cnt);
            for (int i=0; i < args.Length; i++)
            {
                Console.WriteLine(args[i]);
            }
        }
    }
}
```

To help explain the code, look at the following example.



The screenshot shows a Windows Command Prompt window titled "Command Prompt". The current directory is "C:\learningcsharp\console1\clargs\bin\Debug". The user has entered the command "clargs foo bar". The output of the program is displayed as follows:

```
C:\learningcsharp\console1\clargs\bin\Debug>clargs foo bar
Command line arguments: 2
foo
bar
C:\learningcsharp\console1\clargs\bin\Debug>_
```

Arguments can be passed in when running your program. These arguments automatically get assigned to an array named “args”. The array, as you can see from the code, is zero based

The other piece of code to mention is

```
args.Length
```

Since args is an array we can get the number of elements in the array by accessing the “Length” property.

STATIC METHODS IN A CLASS

You just finished a section where I created a phone object. I was able to take this object and make multiple instances of it to suit my needs (home phone, work phone, cell phone, etc...). You have also been exposed to the Main method which is a specialized static method. A static method means there is only one created. Main is specialized in the sense that it is only available to the class it reside in. But we can also create other types of static methods within a class.

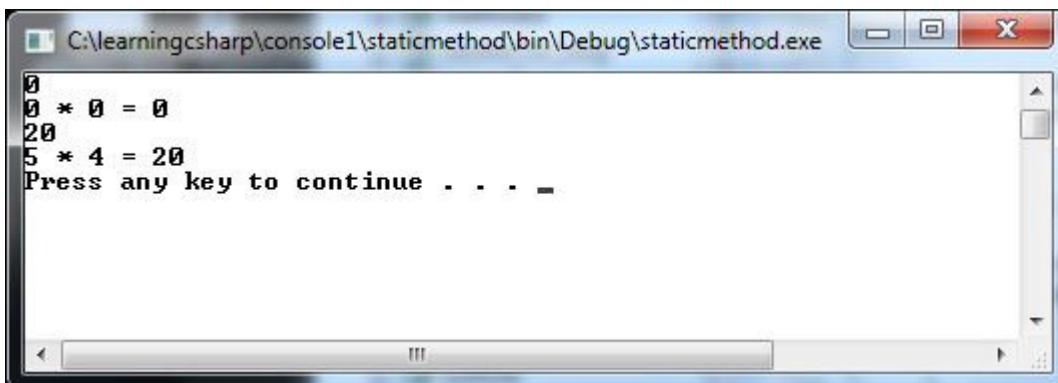
Whereas it makes good sense to create a generic phone class that we can create instances of for the type of phone we want to implement, if our goal is to make a method for a commonly used type of calculation does it make sense to need multiple instances of it?

Well, the answer to this question really depends on a number of issues like scalability, how much utilization the method will get. There are also factors involving system resources. Albeit to say, this is not the time or place to delve into these issues. But, these matters need to be ultimately considered by someone.

For now I want you to only understand that the difference between a static method and a method is the number of instances that occur. With a method, ever invocation of that method creates a unique instance of the method. With a static method only one instance gets created, and every invocation uses that single instance.

Creating and Using a Static Method

Source staticmethod1.cs



```
C:\learningcsharp\console1\staticmethod\bin\Debug\staticmethod.exe
0
0 * 0 = 0
20
5 * 4 = 20
Press any key to continue . . . -
```

```
using System;

class staticmethod
{
    static int x=0;
```

```

public static void Main(string[] args)
{
    int a=0;
    int b=0;
    calc_num(a,b);
    Console.WriteLine(a + " * " + b + " = " + x);
    a=5;
    b=4;
    calc_num(a,b);
    Console.WriteLine(a + " * " + b + " = " + x);

    Console.Write("Press any key to continue . . . ");
    Console.ReadKey(true);
}

static void calc_num(int a2,int b2)
{
    x=(a2*b2);
    Console.WriteLine(x);
}
}

```

Though only a few lines of code, a number of things need to be discussed here. First is the declaration of variables within a class. We can declare a set of GLOBAL variables in a class that are outside of a method, but are available to any method within the class. C# calls these static variables. See how we did this.

```
static int x=0;
```

Had we forgotten the static keyword when declaring these variables, we would have received the following compiler error based on our code.

```
int x=0;
```



The reason we needed to make “x” a static (global) variable is so the results of the method “calc_num” is exposed to the “main” method. Otherwise we have no way of returning the result to main, or short of coding this as a class.

The second thing to show about this code is how we can pass values into a method and return back a value (or values). Consider the code fragment below.

```

a=5;
b=4;
calc_num(a,b);

```

We have declared and initialized two variables in our main method. We call the “calc_num” and pass the two variables with the call. Now examine the “calc_num” method header.

```
static void calc_num(int a2,int b2)
```

I purposely created different names for the variables here. I did this because I wanted you to be sure that these were, in fact, a different set of variables than the ones declared in “main”. The method void requires that we pass it two arguments of type integer, which we have done. Notice that the method is declared as static. By doing this we have exposed this method so that it can be called by other methods relative to the class. Had we forgotten the static keyword, we would have the same error message previously reported.

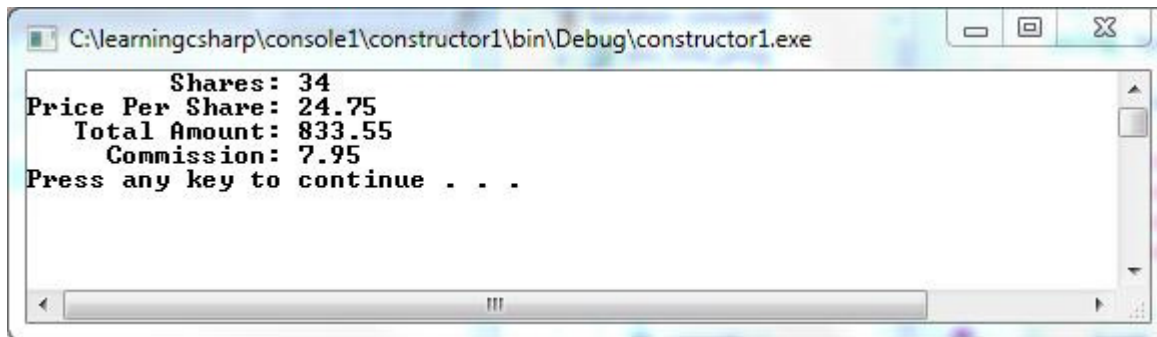
*Remember that global variables within a class get defined by using the **static** keyword.*

Also remember that variables declared within a method are local in scope and in the case of static methods cannot be referenced outside the method.

ADDING A CONSTRUCTOR

A constructor is a special method that allows a class to be automatically initialized when it is instantiated. The constructor must have the same name as the class. Remember back to the first class we wrote. We had to set the values for each data item in our class by explicitly referencing it. Well, we can also use the constructor to implicitly set the data values. Let’s look at the class below.

Source Fund_stock.cs



```
using System;

public class Fund_stock
{
    double shares;
    double price_per_share;
    double commission;
    double total_amount;

    // this is the constructor
    Fund_stock(double shrs, double pps, double comm)
    {
```

```

    shares = shrs;
    price_per_share = pps;
    commission = comm;
    total_amount = ((shrs * pps) - comm);
}

public static void Main(string[] args)
{
    Fund_stock fs = new Fund_stock(34, 24.75, 7.95);
    Console.WriteLine("        Shares: " + fs.shares);
    Console.WriteLine("Price Per Share: " + fs.price_per_share);
    Console.WriteLine("    Total Amount: " + fs.total_amount);
    Console.WriteLine("        Commission: " + fs.commission);

    Console.Write("Press any key to continue . . . ");
    Console.ReadKey(true);
}
}

```

This class accepts three values: number of shares, price_per_share, and commission. Given these three values, it calculates total amount. Now let's look at the code that calls this class.

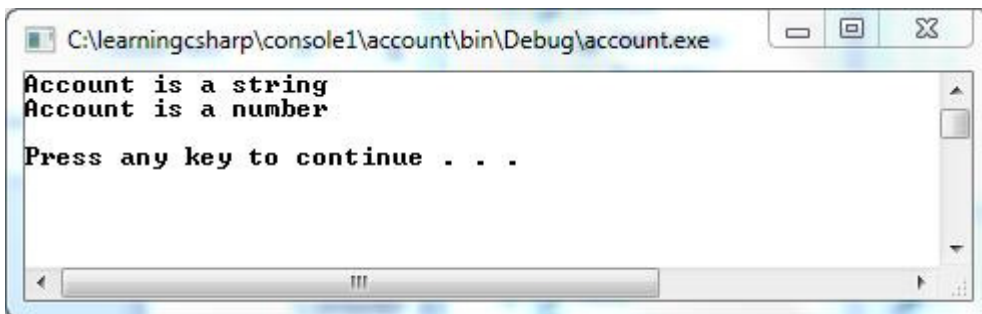
As you can see, we create an instance of the class and we must also pass it the three values it requires. While you do not have to explicitly reference the data members, you still need to know about them and what they represent. Remember the documentation ☺.

CONSTRUCTOR OVERLOADING

Overloading is a totally novel concept related to OOP. In my opinion it is one of the cornerstone concepts that sets OOP apart from non-OOP languages. Overloading a constructor allows you to have more than one constructor for a class. Why would you need more than one constructor for a class?

Consider an account number. An account number can be defined as a number (for example 003425) or it can also be defined as a number with alphanumeric data (for example AR34820). As interfacing between disparate systems is the rage these days, wouldn't it be nice if there was an elegant solution that knew what data type was what and acted upon each accordingly?

Source account.cs



```
using System;
```

```
class account
{
    int Iaccount;
    String Saccount;

    account(int acct) {
        Iaccount = acct;
        Console.WriteLine("Account is a number");
        Console.WriteLine("");
    }

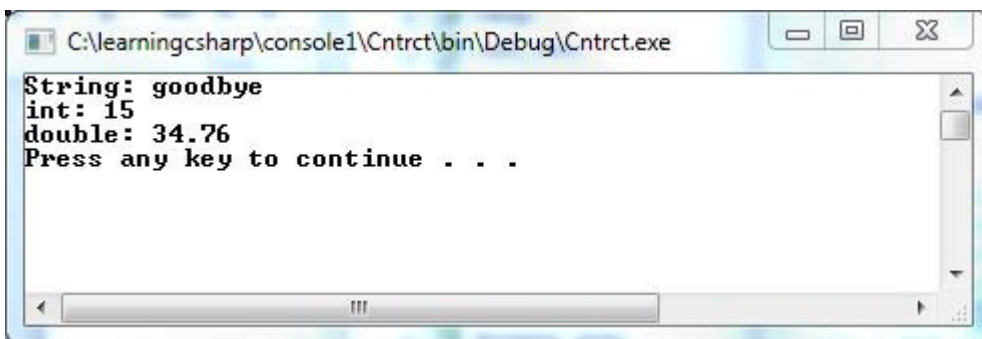
    account(String acct) {
        Saccount = acct;
        Console.WriteLine("Account is a string");
    }

    public static void Main(string[] args)
    {
        account a1 = new account("EB346");
        account a2 = new account(24533);

        Console.Write("Press any key to continue . . . ");
        Console.ReadKey(true);
    }
}
```

You really need to spend some time thinking about this example. As simple as it is, it demonstrates a totally cool, mind blowing concept that should just get you jazzed! The concept of overloading is just totally cool. Here is another more generic example for you to examine.

Source Cntrct.cs



```
using System;
```

```
class Cntrct
{
    /*
```

```
Demonstrates constructor overloading. cntrct_run is uses this class. Notice we have 3 data types represented for the same class. This is a very simple example of polymorphism, a very powerful technique in object oriented programming.
```

```
*/
```

```
int x;
double y;
String z;

Ctrrct(int a) {
    x = a;
    Console.WriteLine("int: " + a);
}

Ctrrct(double b) {
    y = b;
    Console.WriteLine("double: " + b);
}

Ctrrct(String c) {
    z = c;
    Console.WriteLine("String: " + z);
}

public static void Main(string[] args)
{
    /*
    This is an example of calling on the same object using different
    data type arguments. While this is a pretty trivial example, it
    is pretty cosmic if you think about it. This example demonstrates
    the polymorphic aspect of object oriented programming. Imagine a
    more practical example, like passing a system a medical record number
    and not having to be concerned with data type issues.
    */

    // the class Ctrrct will accept String, int, or double data types
        Ctrrct q = new Ctrrct("goodbye");
        Ctrrct r = new Ctrrct(15);
        Ctrrct s = new Ctrrct(34.76);

        Console.Write("Press any key to continue . . . ");
        Console.ReadKey(true);
    }
}
```

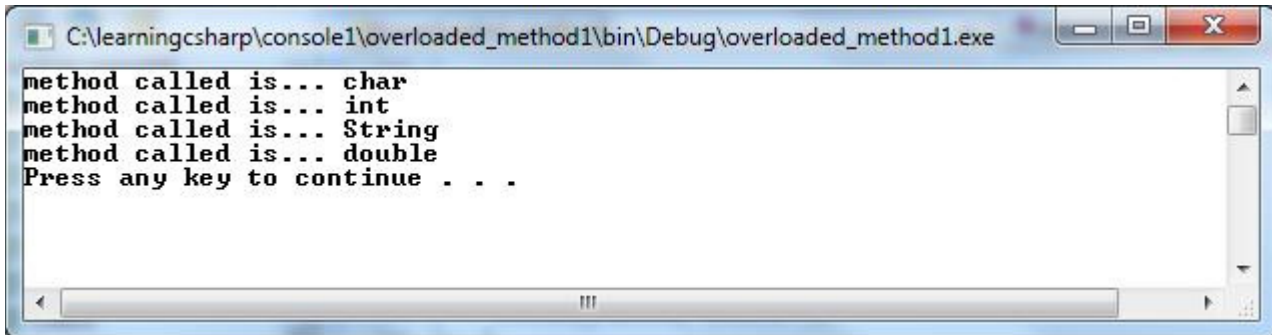
Overloading is one of the coding techniques you use to achieve polymorphism (one interface, multiple implementations).

METHOD OVERLOADING

An overloaded method works the same as an overloaded constructor. An overloaded method is achieved when two or more methods share the same name. Overloaded methods are most useful as they help you to code variations of a similar concept using the same naming convention. This makes documentation and reuse easier and can be used to abstract the interface between consumers of your code.

The “Console.WriteLine()” method we have used in most our program so far is an example of an overloaded method.

Source overloaded_method1.cs.

A screenshot of a Windows console application window. The title bar shows the file path: C:\learningcsharp\console1\overloaded_method1\bin\Debug\overloaded_method1.exe. The console output displays four lines of text: "method called is... char", "method called is... int", "method called is... String", and "method called is... double". Below these lines, it says "Press any key to continue . . .". The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

```
using System;

class overloaded_method1
{
    static void Method_called (String str)
    {
        Console.WriteLine("method called is... " + str);
    }

    void datatype(char c)
    {
        Method_called("char");
    }

    void datatype(int b)
    {
        Method_called("int");
    }

    void datatype(String s)
    {
        Method_called("String");
    }

    void datatype(double d)
    {
        Method_called("double");
    }

    public static void Main(string[] args)
    {
        overloaded_method1 p1 = new overloaded_method1();
        p1.datatype('2');
        p1.datatype(2);
        p1.datatype("2");
        p1.datatype(2.01);

        Console.Write("Press any key to continue . . . ");
    }
}
```

```
    Console.ReadKey(true);  
}  
  
}
```

You can see from the program above that the method called is determinant on the value that is passed in the argument of the method call. Interesting enough, sometimes a data type gets “promoted to another data type”. Consider the code below (in particular the parts that are bolded):

Source overloaded_method2.cs.

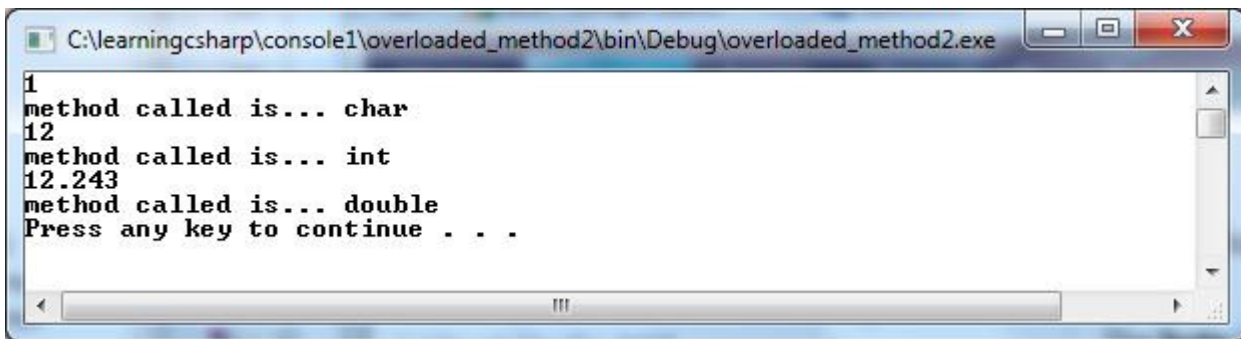
```
using System;  
  
class overloaded_method2  
{  
  
    static void overloaded_method (String str)  
    {  
        Console.WriteLine("method called is... " + str);  
    }  
  
    void datatype(char c)  
    {  
        char v='0';  
        v=c;  
        Console.WriteLine(v);  
        overloaded_method("char");  
    }  
  
    void datatype(float f)  
    {  
        float v=0;  
        v=f;  
        Console.WriteLine(f);  
        overloaded_method("float");  
    }  
  
    void datatype(int i)  
    {  
        int v=0;  
        v=i;  
        Console.WriteLine(v);  
        overloaded_method("int");  
    }  
  
    void datatype(double d)  
    {  
        double v=0;  
        v=d;  
        Console.WriteLine(v);  
        overloaded_method("double");  
    }  
  
    void datatype(byte b)  
    {
```

```
byte v=0;
v=b;
Console.WriteLine(v);;
overloaded_method("byte");
}

public static void Main(string[] args)
{
    overloaded_method2 p1 = new overloaded_method2();
    p1.datatype('1');
    p1.datatype(12);
    double x = 12.243;
    p1.datatype(x);

    Console.Write("Press any key to continue . . . ");
    Console.ReadKey(true);
}
}
```

We might expect that when we pass the value 12 to the method we would get the byte method as this number best fits this data type. However, the value is promoted to type “int”.



VARIABLES AND VARIABLE SCOPE

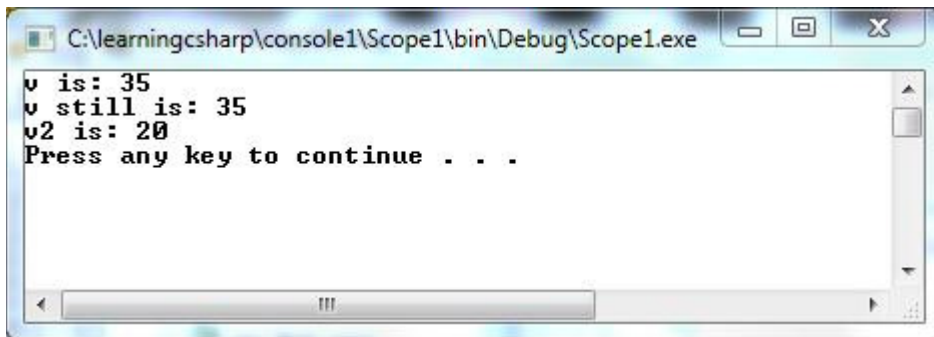
We actually have covered this subject, though we have done so indirectly. I did not want to get into a specific discussion on this while we were elbow deep into classes for the first time. Now is the time that we are going to quickly review what you have already seen and coded yourself.

In C# there are three classifications of variables. They are...

- Instance variables
- Static variables
- Local variables

You have seen and used all three types thus far. First and foremost, both static and instance variables are global in their scope. The difference between a static variable and an instance variable is best explained with some toy code - as a picture is worth 1000 clichés.

Source Scope1.cs



```
C:\learningcsharp\console1\Scope1\bin\Debug\Scope1.exe
v is: 35
v still is: 35
v2 is: 20
Press any key to continue . . .
```

```
using System;

class Scope1
{
    public static void Main(string[] args)
    {
        Var v = new Var();
        v.myint = 35;
        Console.WriteLine("v is: " + v.myint);

        Var v2 = new Var();
        v2.myint = 20;
        Console.WriteLine("v still is: " + v.myint);
        Console.WriteLine("v2 is: " + v2.myint);

        Console.Write("Press any key to continue . . . ");
        Console.ReadKey(true);
    }

    class Var
    {
        public int myint;
    }
}
```

In this example each time we instantiate the class, a copy of the variable “v” is created on the heap. These types of variables are referred to as **instance** variables.

Next...

Source Scope2.cs

```
class Scope2
using System;

class Scope2
{
    public static void Main(string[] args)
    {
        Var.myint = 35;
    }
}
```

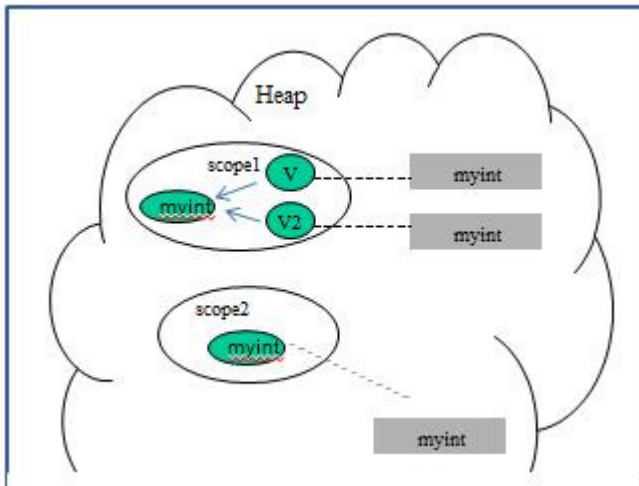
```
    Console.WriteLine("v is: " + Var.myint);

    // results in compile error.
    //Var v = new Var();

    Console.Write("Press any key to continue . . . ");
    Console.ReadKey(true);
}

class Var
{
    public static int myint;
}
}
```

In this example, because `myint` was declared as `static`, it is only defined once on the heap once! As such, C# does not allow you to create an instance. Instead you must work with the variable through direct reference. Following is a picture that graphically depicts the concept I just showed you in code. Oh, by the way, the heap is an addressable portion of memory. The heap is where C# objects live. This should be enough detail on this subject.



Finally, there are local variables. These are variables that are declared within a method. The scope of these variables is local to the method. Actually scope of a variable is even more granular than this (in or outside of a method). Scope is defined within a block. Consider the code below:

Source Scope3.cs

```
using System;

class Scope3
{
    public static void Main(string[] args)
    {
        Var v = new Var();
    }
}
```

```

    v.mymethd();

    Console.WriteLine("\nPress any key to continue . . . ");
    Console.ReadKey(true);
}

}

class Var
{
    static int myint=0;    //myint in scope

    public void mymethd()
    {
        int foo=30;      //foo in scope

        {
            int a=5;      //a in scope
            {
                int b=3;    //b in scope
                {
                    int c=10;    //c in scope
                    myint += c;
                    Console.WriteLine("Sum: " + myint);
                } //c no longer in scope
                myint += b;
                Console.WriteLine("\nSum: " + myint);
            } //b no longer in scope
            myint += a;
            Console.WriteLine("\nSum: " + myint);
        } //a no longer in scope
        myint += foo;
        Console.WriteLine("\nSum: " + myint);
        myint += myint;
        Console.WriteLine("\nSum: " + myint);
    } //foo no longer in scope
} //myint no longer in scope

```

The comment codes in the program explain what is happening so no further narrative is offered. You should be able to grasp scope of a variable at the class, method and block level.

PUT INTO PRACTICE – OVERLOADED METHODS



Create a class that accepts the following types of phone numbers using overloaded methods.

<area_code><exchange><number>

Or

<exchange><number>

In other words, one method accepts three input arguments, the other accepts two. Verify your method is working.

```
Overloaded Method with 3 args used
425-555-1212
Overloaded Method with 2 args used
555-1234
Press any key to continue . . . _
```

Note: The – 's were inserted in the method itself. They are not part of the input argument!

Additionally, add code to your methods to validate that
<area_code> is only 3 characters in length
<exchange> is only 3 characters in length
<number> is only 4 characters in length

QUICK INVENTORY

We have covered a great deal of ground. I have broken the subject of OOP into multiple chapters so that you have time to digest and absorb each item covered.

- Thus far you need to have a conceptual understanding of the three main principles of OOP. Since OOP is about reuse you should be getting onto the reuse bandwagon. You should have an idea of the benefits, and just as important, the drawback's of reuse as it is applied to OOP.
- Understand the relationship between a class and a method.
- Understand the difference between a "Static Method" and an "Instance Method"
- Begin understanding the use of variables within a class and within methods.
- Understand and use constructors and overloaded constructors in classes.
- Understand and use overloaded methods in classes.

If you can check off this inventory in its entirety, you are ready for module #3.