Chapter 1

Getting Started

Copyright © Thomas P. Skinner
A Simple Class

```csharp
public class Demo
{
    public int count;
    public string name = "Tom";
    public void Increment()
    {
        ++count;
    }
}
```
A Complete Program

```java
public class Demo
{
    public int count;
    public string name = "Tom";
    public void Increment()
    {
        ++count;
    }
    public static void Main()
    {
        Demo demo = new Demo();
        demo.Increment();
    }
}
```
Use of Static

```csharp
public class Demo
{
    public static int count;
    public static string name = "Tom";
    public static void Increment()
    {
        ++count;
    }
    public static void Main()
    {
        Increment();
    }
}
```
Creating a New Project

![New Project dialog box with options to create a new Console Application in Visual C#]
The Console Class

```csharp
using System;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("My first program in C#.");
        }
    }
}
```
static void Main(string[] args)
{
    int n=2;
    for (int i = 1; i <= 8; ++i)
    {
        Console.WriteLine("{0} {1}" , i, n);
        n *= 2;
    }
}
Program Output

Press any key to continue . . .
## C# Data Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbyte</td>
<td>Signed 8-bit</td>
</tr>
<tr>
<td>byte</td>
<td>Unsigned 8-bit</td>
</tr>
<tr>
<td>char</td>
<td>Unicode 16-bit character</td>
</tr>
<tr>
<td>short</td>
<td>Signed 16-bit</td>
</tr>
<tr>
<td>ushort</td>
<td>Unsigned 16-bit</td>
</tr>
<tr>
<td>int</td>
<td>Signed 32-bit</td>
</tr>
<tr>
<td>uint</td>
<td>Unsigned 32-bit</td>
</tr>
<tr>
<td>long</td>
<td>Signed 64-bit</td>
</tr>
<tr>
<td>ulong</td>
<td>Unsigned 64-bit</td>
</tr>
<tr>
<td>decimal</td>
<td>128-bit decimal number (28-29 digits)</td>
</tr>
<tr>
<td>float</td>
<td>32-bit floating-point (7 digits)</td>
</tr>
<tr>
<td>double</td>
<td>64-bit floating-point (15-16 digits)</td>
</tr>
</tbody>
</table>
Other Data Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A Unicode character string of arbitrary length</td>
</tr>
<tr>
<td>bool</td>
<td>A boolean \textit{true} or \textit{false} value</td>
</tr>
<tr>
<td>object</td>
<td>The base class for all C# classes</td>
</tr>
</tbody>
</table>
Value vs. Reference Types

Stack

int count 1234

string s (address)

Heap

The character string.
Boxing and Unboxing

//Boxing
int i = 0;
object o = i;

//Unboxing
int j = (int) o;
Unboxing Error

//The runtime will not accept this
int i = 0;
object o = i;
long j = (long) o;

//This works
long k = (int) o;
Runtime Exception

InvalidCastException was unhandled

An unhandled exception of type 'System.InvalidCastException' occurred in ConsoleApplication1.exe.

Additional information: Specified cast is not valid.

Troubleshooting tips:
- Make sure the source type is convertible to the destination type.
- When casting from a number, the value must be a number less than infinity.
- Get general help for this exception.
- Search for more Help Online...

Exception settings:
- Break when this exception type is thrown

Actions:
- View Detail...
- Copy exception detail to the clipboard
- Open exception settings
The Ref Keyword

static void Main(string[] args)
{
    int i=0;
    foo(ref i);
    Console.WriteLine(i);
}
static void foo(ref int i)
{
    ++i;
}

//The output is 1 and not 0
//Without using ref the output would be 0
The Out Keyword

• Variables must be initialized prior to being passed to a method unless “out” is used

//This fails even though out is used
static void foo(out int i)
{
    ++i;  //what is the value of I prior to increment?
}

//On the other hand, this would work just fine:
static void foo(out int i)
{
    i = 100;
}

• Why is there no "in" keyword?
## Access Modifiers

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>Access is not restricted.</td>
</tr>
<tr>
<td>protected</td>
<td>Access is limited to the containing class or types derived from the containing class.</td>
</tr>
<tr>
<td>internal</td>
<td>Access is limited to the current assembly.</td>
</tr>
<tr>
<td>protected internal</td>
<td>Access is limited to the current assembly or types derived from the containing class.</td>
</tr>
<tr>
<td>private</td>
<td>Access is limited to the containing type.</td>
</tr>
</tbody>
</table>