Declaring variable names and datatypes

Define Variable names using following:

1. **Use PascalCasing for class names and method names:**

```
public class ClientActivity
{
        public void ClearStatistics()
        {
        //...
        }
        public void CalculateStatistics()
        {
        //...
        }
}
```

2. **Use camelCasing for method arguments and local variables:**

```
public class UserLog
{
        public void Add(LogEvent logEvent)
        {
           int itemCount = logEvent.Items.Count;
           // ...
        }
}
```

3. **Do not use Hungarian notation or any other type identification in identifiers:**

```
// Correct int counter; string name;

// Avoid

int iCounter;
string strName;
```

4. **Do not use Screaming Caps for constants or readonly variables**

```
// Correct
public static const string ShippingType = "DropShip";

// Avoid
public static const string SHIPPINGTYPE = "DropShip";
```

5. **Use meaningful names for variables. The following example uses seattleCustomers for customers who are located in Seattle:**

```
var seattleCustomers = from cust in customers
                                  where cust.City == "Seattle"
                                  select cust.Name;
```

6. **Avoid using Abbreviations. Exceptions: abbreviations commonly used as names, such as Id, Xml, Ftp, Uri.**

```
// Correct
UserGroup userGroup;
Assignment employeeAssignment;

// Avoid
UserGroup usrGrp;
Assignment empAssignment;

// Exceptions
CustomerId customerId;
XmlDocument xmlDocument;
FtpHelper ftpHelper;
UriPart uriPart;
```

7. **Use PascalCasing for abbreviations 3 characters or more (2 chars are both uppercase)**

```
HtmlHelper htmlHelper;
FtpTransfer ftpTransfer;
UIControl uiControl;
```

8. **Do not use Underscores in identifiers. Exception: you can prefix private static variables with an underscore:**

```
// Correct
public DateTime clientAppointment;
public TimeSpan timeLeft;

// Avoid
public DateTime client_Appointment;
public TimeSpan time_Left;

// Exception (Class field)
private DateTime _registrationDate;
```

9. **Use appropriate datatypes for storing values by determining the maximum possible value to be stored.**

   For example, if you need to store value which will not be greater than 10000 then it is better to use Int16 instead of Int.

10. **Use implicit type var for local variable declarations. Exception: primitive types (int, string, double, etc) use predefined names.**

```
var stream = File.Create(path); var customers = new Dictionary();
// Exceptions

int index = 100; string timeSheet; bool isCompleted;
```

11. **Use noun or noun phrases to name a Class.**

```
public class Employee
{
}
public class BusinessLocation
{
}
public class DocumentCollection
{
}
```

12. **Use prefix interfaces with the letter I. Interface names are noun (phrases) or adjectives.**

```
public interface IShape
{
}
public interface IShapeCollection
{
}
public interface IGroupable
{
}
```

13. **Use name source files according to their main classes. Exception: file names with partial classes reflect their source or purpose, e.g. designer, generated, etc.**

```
// Located in Task.cs
public partial class Task
{
//...
}
// Located in Task.generated.cs

public partial class Task
{
//...
}
```

14. **Use organized namespaces with a clearly defined structure:**

```
// Examples
namespace Company.Product.Module.SubModule
namespace Product.Module.Component
namespace Product.Layer.Module.Group
```

15. **Use vertically align curly brackets:**

```
// Correct
class Program
{
    static void Main(string[] args)
    {
    }
}
```

16. **Declare all member variables at the top of a class, with static variables at the very top.**

```
// Correct
public class Account
{
    public static string BankName;
    public static decimal Reserves;
    public string Number {get; set;}
    public DateTime DateOpened {get; set;}
    public DateTime DateClosed {get; set;}
    public decimal Balance {get; set;}

    // Constructor public Account()
    {
            // ...
    }
}
```

17. **Use singular names for enums. Exception: bit field enums.**

```
// Correct
public enum Color
{
    Red,
    Green,
    Blue,
    Yellow,
    Magenta,
    Cyan
}

// Exception [Flags]
public enum Dockings
{
    None = 0, Top = 1, Right = 2, Bottom = 4, Left = 8
}
```

18. **Do not explicitly specify a type of an enum or values of enums (except bit fields)**

```
// Don't
public enum Direction : long
{
    North = 1,
    East = 2,
    South = 3,
    West = 4
}

// Correct
public enum Direction
{
    North,
    East,
    South,
    West
}
```

19. **Do not use an “Enum” suffix in enum type names.**

```
// Don't
public enum CoinEnum
{
    Penny,
    Nickel,
    Dime,
    Quarter,
    Dollar
}

// Correct
public enum Coin
{
    Penny,
    Nickel,
    Dime,
    Quarter,
    Dollar
}
```

20. Do not use “Flag” or “Flags" suffixes in enum type names.

```
//Don't
[Flags]
public enum DockingsFlags
{
    None = 0,
    Top = 1,
    Right = 2,
    Bottom = 4,
    Left = 8
}
//Correct
[Flags]
public enum Dockings
{
    None = 0,
    Top = 1,
    Right = 2,
    Bottom = 4,
    Left = 8
}
```

21. **Use suffix EventArgs at creation of the new classes comprising the information on event:**

```
// Correct
public void BarcodeReadEventArgs : System.EventArgs
```

22. **Use name event handlers (delegates used as types of events) with the "EventHandler" suffix, as shown in the following example:**

```
public delegate void ReadBarcodeEventHandler(object se
nder, ReadBarcodeEventArgs e);
```

23. **Do not create names of parametres in methods (or constructors) which differ only the register:**

```
// Avoid
private void MyFunction(string name, string Name)
```

24. **Use two parameters named sender and e in event handlers. The sender parameter represents the object that raised the event. The sender parameter is typically of type object, even if it is possible to employ a more specific type.**

25. **Use suffix Exception at creation of the new classes comprising the information on exception:**

```
// Correct
public void BarcodeReadException : System.Exception
```

26. **Do not declare all the variables Global. Use local variables when a variable is not required to be accessed outside the given scope.**

Declaring variable names and datatypes

```
// Avoid
public class Account
{
      public static string BankName;
      public static decimal Reserves;
      public string Number {get; set;}
      public DateTime DateOpened {get; set;}
      public DateTime DateClosed {get; set;}
      public decimal Balance {get; set;}

      // Methods
       ………
}


// Correct

public class Account
{
      public void AccountDetails()
      {
         public static string BankName;
         public static decimal Reserves;
                public string Number {get; set;}
                public DateTime DateOpened {get; set;}
                public DateTime DateClosed {get; set;}
                public decimal Balance {get; set;}
      }

      // other Methods
       ………
}
```

27. Use of datatypes must be according to the minimum and maximum value.

Type	Description	Range
byte	8-bit unsigned integer	0 to 255
sbyte	8-bit signed integer	-128 to 127
short	16-bit signed integer	-32,768 to 32,767
ushort	16-bit unsigned integer	0 to 65,535
int	32-bit signed integer	-2147483648 to 2147483647
uint	32-bit unsigned integer	0 to 4,294,967,295
long	64-bit signed integer	-9223372036854770000 to 9223372036854770000
ulong	64-bit unsigned integer	0 to 18,446,744,073,709,551,615
float	32-bit Single-precision floating point type	-3.402823e38 to 3.402823e38
double	64-bit double-precision floating point type	-1.79769313486232e308 to 1.79769313486232e308
decimal	128-bit decimal type for financial and monetary calculations	(+ or -)1.0 x 10e-28 to 7.9 x 10e28
char	16-bit single Unicode character	Any valid character, e.g. a,*, \x0058 (hex), or\u0058 (Unicode)
bool	8-bit logical true/false value	True or False
DateTime	Represents date and time	0:00:00am 1/1/01 to 11:59:59pm 12/31/9999