**1命名规范**

所有的接口名、类名和方法名都应该使用 **PascalCasing** 风格来定义，对于接口参数、构造函数参数和方法参数，其参数名则使用 **camelCase** 风格。对于方法体内的局部变量，则使用 **camelCase** 风格。

1、类变量命名:类变量命名一律用下划线开头，多个单词组成，第一个字母小写的，其他单词第一个字母大写。

public class Hello

{

private string \_firstName;

private DateTime \_date;

}

2、类命名规范:

2.1、名字应该能够标识事物的特性；

2.2、首字母大写，缩略语可以全部字母大写；

2.3、名字可以有两个或三个单词组成，但通常不应多于三个；

2.4、在名字中，每个单词的第一个字母大写。例如：IsSuperUser；包含ID的，ID全部大写，如CustomerID；

2.5、使用名词或名词短语命名类；

2.6、少用缩写，除非是被广泛使用的；

2.7、不要使用下划线字符 (\_)。

3、局部变量命名：由小写字母开头，其他单词开头字母大写的各单词组成。

int  localVar;

4、函数命名：一律由大写字母开头的各个单词组成。

public void GetValue(int type)

5、枚举类型命名:一律由大写字母”E”开头,第一个字母大写的各个单词组。

public enum Esample

并确保,枚举类型中有个约定的EMPTY元素。

3.6、接口命名规范: 和类命名规范相同，唯一区别是接口在名字前加上"I"前缀。

///

///接口样例

///

public interface ISample

{

///

///获取对象

///

object GetValue();

}

7、[抽象类](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7766757&ss_c=ssc.citiao.link" \t "_blank)的命名:一律由大写字母”A”开头,第一个字母大写的各单词组成。

8、方法（Method）、属性（Atribute）命名规范

8.1方法命名应采用首字母大写的方式；

8.2返回值是Bool值的方法和属性以Is开头；

8.3使用返回值是Bool值的方法（属性）的时候，不要将bool值方法（属性）与if语句放在同一行，如应该这样写：

bool valid = IsValid();

if (valid)

{

}

9类的实例命名规范：类的实例第一个字母小写，其他单词的第一个字母大写。

FileStream fileStream=new FileStream();

10委托和事件命名规范：

1委托，以EventHandler作为后缀名。

2.事件，以其对应的委托类型，去掉EventHandler后缀，添加On前缀。

**2字段**

所有的 **internal** 和 **private** 字段使用 **\_camelCase** 风格，即在字段添加 **\_** 前缀，并尽可能使用 **readonly** 。对于静态字段添加 **s\_** 前缀，对于线程静态字段添加 **t\_** 前缀。如果使用了静态的字段而且可以设置 **readonly** ，需要把 **readonly** 放在 **static** 后面（即 **static readonly** 而不是 **readonly static** ）。公开(public)字段应谨慎使用，如果需要公开字段，则应使用 **PascalCasing** 风格，并且不带前缀。

**3访问修饰**

即使访问可见性是默认的，我们也要始终指定，我们也总是指定可见性（例如， **private string \_foo** 而不是 **string \_foo** ）。访问可见性应该是第一个修饰符（例如， **public abstract** 而不是 **abstract public** ）

**4 布局规范**

1. 代码缩进

代码缩进4个空格(在方法中无参数，可在一行，有参数则将花括号独自占一行)

2. 代码换行

单行代码限制在120个字符，超长处理：

第二行相对第一行缩进4空格，从第三行开始保持和第二行一致。运算符及方法调用“.”需要跟随换行，但逗号不需要。

3. 代码空行

1).空两行

 接口和类定义之间。

 枚举和类定义之间。

类与类定义之间。

2).空一行

与方法、属性与属性、属性与方法、属性与字段、方法与字段之间。

方法中变量声明和语句之间。

方法与方法之间。

方法中不同逻辑块之间。

方法中返回语句和其他语句之间。

注释与它注释的语句间不空行，但与其它的语句间空一行。

4. 代码空格

1).关键字和左括符“(”

2).方法名和左括符“(”不要使用。

3).多个参数用逗号分隔，逗号后需要添加一个空格。

4).除了“.”之外，所有二元操作符需要空格分开。

5).一元操作符、++、--不能用空格分隔。

6).语句表达式之间用空格分隔，常见于for, if 语句中。

5. 代码括号(花括号)

1).在没有必要的时候，不要再返回语句中使用括号().

2).左括符“(”和关键字空格。

3).左括符“(”和方法名, 不添加空格。

4).左花括号放在关键字或者方法名下一行，左右花括号对齐。

5).if、while、do语句后一定要使用{}，即使{}号中为空或只有一条语句。

6).右花括号 “}” 后建议加一个注释以便于方便的找到与之相应的 {。

每个花括号在新的一行开始，而且花括号不进行缩进，花括号里面的代码缩进。

while (x == y){ something(); somethingelse();}

对于单行语句块，可以不带花括号，但需要把它放在同一行

if (xx == null) xx = new Foo();

一个例外是，允许 using 语句嵌套在另一个 using 语句中，从同一缩进级别的下一行开始，即使嵌套的 using 包含受控块也是如此。

 尽管编译器可以忽略空行与空格，但是空行与空格可以将不同逻辑的代码单元分离，从而提高代码的可读性。

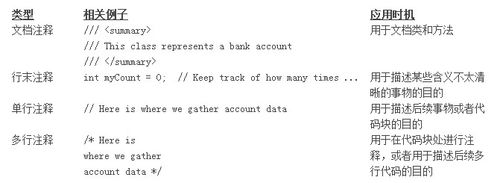
随时避免出现多个空白行。例如，类型的成员之间不要有两个空白行。

避免多余的空格。例如，避免 **if（someVar == 0）...** ，其中点标记为多余空格。如果使用 Visual Studio 辅助检测，通过启用 Visual Studio中“查看空格（Ctrl + R，Ctrl + W）”或“编辑->高级->查看空格”。

**4、注释**

  注释是为了增加代码的清晰度，保持注释的简洁，不是任何代码都需要注释的，过多的注释反而会影响代码的可读性。注释不要包括其他的特殊字符，建议先写注释，后写代码，注释和代码一起完成。如果语句块（比如循环和条件分枝的代码块）代码太长，嵌套太多，则在其结束“｝”要加上注释，标志对应的开始语句。如果分支条件逻辑比较复杂，也要加上注释。

下表为每一种类型列出了一个例子，说说明了每一种类型的应用时机。



**5、隐式类型**

仅在明显可以知道对象类型时，可以使用 **var** ，如 **var stream = new FileStream(...)** 。在无法明显知道对象类型时，不可以使用 **var** ，如 **var stream = OpenStandardInput()** 。

**6、关键字**

使用语言的关键字代替 BCL 类型，如使用 **int, string, float** 代替 **Int32，String，Single** 。

7、常量命名

我们使用 **PascalCasing** 风格来命名所有常量局部变量和字段。唯一的例外是互操作代码，其中常量值应与您通过互操作调用的代码的名称和值完全匹配。

**8、变量名字符串**

如果使用变量名的常量，必须使用 **nameof** 关键字。

**9、其它字符**

在源代码中包含非 ASCII 字符时，需要使用 Unicode 转义序列（\uXXXX）代替文字字符。文字非 ASCII 字符有时会被工具或编辑器弄乱。

**10、条件约定**

如果使用条件语句，则遵循以下约定：

if(source == null) throw new ArgumentNullException("source")

if/else if/.../else

if/else if/.../else

**11、代码使用规范**

1. 使用语言特性

 1). this, 使用this区分类型中的属性，变量、静态成员。除非绝对必要，应避免使用 **this.** 。在扩展方法中，需要使用 **this** 。

 2). var, 适当的使用可以提高开发效率，用在不知道返回的具体类型和不需要知道返回类型

2. 异常

1). 当程序出现与预期不符时应该抛出异常让程序上有处理。

2). 尽可能使用内置的异常类型。

3). 捕获异常必须处理。

4). 获取指定异常而非统一使用Exception.

5). 做测试时，在对应的方法或者函数内部直接抛出尽可能出现的异常，以供解决问题 。

3. 安全准则

1). 在代码中不要使用具体的路径，驱动名称，使用相对并保证复用。

2). 在代码中注意资源的释放。

3). 安全协议的使用(HTTP, HTTPClient )。

4. 字符串内插

字符串内插提高可读性。

**15.示例**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Data.OleDb;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApplication1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

}

public double dmstorad(string s)

{

string[] ss = s.Split(new char[3] { '°', '′', '″' },

StringSplitOptions.RemoveEmptyEntries); //用°、 ′、 ″分割字符串s

double[] d = new double[ss.Length]; //新建一个双精度数值数组

for (int i = 0; i < d.Length; i++) //将度分秒存入这个双精度数值数组中

d[i] = Convert.ToDouble(ss[i]);

double sign = d[0] >= 0.0 ? 1.0 : -1.0;//判断角度值是否为负值

double rad = 0;

if (d.Length == 1) //根据数组的长度进行判断计算

rad = Math.Abs(d[0]) \* Math.PI / 180; //将度取绝对值，并转换为弧度

else if (d.Length == 2)

rad = (Math.Abs(d[0]) + d[1] / 60) \* Math.PI / 180; //将度取绝对值，分化为度，并最终转换为弧度

else

rad = (Math.Abs(d[0]) + d[1] / 60 + d[2] / 60 / 60) \* Math.PI / 180; //将度取绝对值，分化为度，秒化为度，并最终转换为弧度

rad = sign \* rad; //弧度前边添加正负号

return rad;//返回弧度值

}

public string radtodms(double rad)

{

double sign = rad >= 0.0 ? 1.0 : -1.0;

rad = Math.Abs(rad) \* 180 / Math.PI;

double[] d = new double[3];

d[0] = (int)rad;//取整得度

d[1] = (int)((rad - d[0]) \* 60);

d[2] = (rad - d[0] - d[1] / 60) \* 60 \* 60;

d[2] = Math.Round(d[2], 2);

if (d[2] == 60) //特殊情况判断

{

d[1] += 1;

d[2] -= 60;

if (d[1] == 60)

{

d[0] += 1;

d[1] -= 60;

}

}

d[0] = sign \* d[0];

string s = Convert.ToString(d[0]) + '°' + Convert.ToString(d[1]) + '′' + Convert.ToString(d[2]) + '"';

return s;

}

public double fangweijiao(double[] sdr, double[] cr)

{

double sum = 0;

for (int i = 1; i < sdr.Length; i++)//从第二行开始循环计算

{

sum += sdr[i];

cr[i] = cr[i - 1] + sdr[i] - Math.PI;//计算 坐标方位角

if (cr[i] >= Math.PI \* 2)//判断坐标方位角是否在0到360之间

cr[i] -= Math.PI \* 2;

else if (cr[i] < 0.0)

cr[i] += Math.PI \* 2;

}

return sum;

}

private void excel文件ToolStripMenuItem\_Click(object sender, EventArgs e)

{

dataGridView1.DataSource = null; //清除数据源

dataGridView1.Rows.Clear(); //清空数据表格的行列

dataGridView1.Columns.Clear();

OpenFileDialog file = new OpenFileDialog(); //声明 打开文件对话框 file

file.Filter = "Excel文件|\*.xls|Excel文件|\*.xlsx"; //文件过滤器，只显示Excel文件

if (file.ShowDialog() == DialogResult.OK) //如果文件正常打开

{

string fname = file.FileName; //获取打开的文件名称

string strSource = "provider=Microsoft.ACE.OLEDB.12.0;" + "Data Source=" + fname + ";Extended Properties='Excel 8.0; HDR=Yes;IMEX=1'"; //准备文件来源信息

OleDbConnection conn = new OleDbConnection(strSource); //Excel文件源放到conn中

string sqlstring = "SELECT \* FROM [Sheet1$]"; //准备选择表中的Sheet1

OleDbDataAdapter adapter = new OleDbDataAdapter(sqlstring, conn); //声明数据适配器adapter

DataSet da = new DataSet(); //声明数据集da

adapter.Fill(da); //使用adapter填充方法

dataGridView1.DataSource = da.Tables[0]; //将da.Tables[0]作为dataGridView1的数据源

}

else

return;

}

private void button1\_Click(object sender, EventArgs e)

{

string[] sd = new string[dataGridView1.RowCount - 5]; //新建一个数组存放观测角度的原始值

double[] sdr = new double[sd.Length]; //新建一个数组存放观测角度的弧度值

double[] cr = new double[sd.Length]; //新建一个数组存放计算的坐标方位角

double sum = 0;

cr[0] = dmstorad(Convert.ToString(dataGridView1.Rows[0].Cells[4].Value)); //获取第一个坐标方位角，并将其转换成弧度，放入cr[]数组第一个元素中

double acd = dmstorad(Convert.ToString(dataGridView1.Rows[sd.Length - 1].Cells[4].Value)); //获取终边坐标方位角，并将其转换成弧度，放入放入acd中用于计算和检核

for (int i = 1; i < sd.Length; i++) //从第二行开始循环，将观测角度的原始值放入sd[]数组中,并转换成弧度值存放在sdr数组中

{

sd[i] = Convert.ToString(dataGridView1.Rows[i].Cells[1].Value);

sdr[i] = dmstorad(sd[i]);

}

sum = fangweijiao(sdr, cr); //计算改正前坐标方位角和观测角度总和，分别存储在 cr数组和sum中

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[1].Value = radtodms(sum); //将观测角度总和放入表格中

double fd, fdx;

fd = cr[cr.Length - 1] - acd;//计算角度闭合差

fdx = 60 \* Math.Sqrt(sd.Length - 1);//计算角度闭合差限差

dataGridView1.Rows[dataGridView1.RowCount - 3].Cells[1].Value = Convert.ToString(Math.Round(fd \* 180 / Math.PI \* 3600, 2)) + "″"; //将角度闭合差存入表格中

dataGridView1.Rows[dataGridView1.RowCount - 2].Cells[1].Value = Convert.ToString(Math.Round(fdx, 2)) + "″";//将角度闭合差限差存入表格中

if (Math.Abs(fd \* 180 / Math.PI \* 3600) > fdx)//检查角度闭合差是否满足要求

MessageBox.Show("角度闭合差超限！");

else

{

double vd = -fd / (sd.Length - 1);//分配角度闭合差

double sumvd = 0;

for (int i = 1; i < sd.Length; i++)

{

sdr[i] += vd;//计算改正后的观测角度，并存入sdr数组中

sumvd += vd;

dataGridView1.Rows[i].Cells[2].Value =

Convert.ToString(Math.Round(vd \* 180 / Math.PI \* 3600, 2)) + "″"; //将角度改正数存入表格中

dataGridView1.Rows[i].Cells[3].Value = radtodms(sdr[i]);

}

if (Math.Round(sumvd, 8) != Math.Round(-fd, 8))

MessageBox.Show("角度改正数分配有误！");

else

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[2].Value =

Convert.ToString(Math.Round(sumvd \* 180 / Math.PI \* 3600, 2)) + "″"; //将角度改正数总和存入表格中

sum = fangweijiao(sdr, cr);//推算改正后的坐标方位角

if (Math.Round(cr[cr.Length - 1], 8) != Math.Round(acd, 8))

MessageBox.Show("坐标方位角推算有误！");

else

{

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[3].Value = radtodms(sum); //将改正后观测角度总和放入表格中

for (int i = 0; i < sd.Length; i++)//将改正后坐标方位角存入表格

dataGridView1.Rows[i].Cells[4].Value = radtodms(cr[i]);

}

}