****

**软件编码规范**

**姓 名：王星雨**

**班 级：土1703-1班**

**学 号：20170630**

**指导教师：王辉**

**一、C#软件开发编码规范**

**1.1简介**

**1.2编写目的**

为了保证从业人员编写出的程序都符合相同的规范，保证一致性、统一性而建立了程序编码规范。

编码规范对于程序员而言尤为重要，有以下几个原因：

1)一个软件的生命周期中，80%的花费在于维护。

2)几乎没有任何一个软件，在其整个生命周期中，均由最初的开发人员来维护。

3)编码规范可以改善软件的可读性，可以让程序员尽快而彻底地理解新的代码。

因此，每个软件开发人员都必须遵守统一的编码规范。这是从业人员必备的素质。

**1.3适用范围**

本规范适用于用C#语言编写程序的情况。

**1.4基本要求**

代码是复杂难懂的，应该尽量使代码简单直白，提高代码可读性。

**2. 命名规范**

**2.1字母大小写约定**

**2.1.1说明**

表达清晰的命名规范是程序规划的核心，如果规范的命名能清晰的表达出相应的功能，就可以让人“望文知意”，提高开发效率和系统的可维护性。而且，C#编程语言是区分大小写的，A与a表达的是两个不同的字符和含义。反之，如果命名不能表达其含义，例如“aaa”、“bbb ()”，那么将适得其反。

**2.1.2 Pascal风格**

包含一到多个单词，每一个单词第一个字母大写，其余字母均小写。例如：HelloWorld、SetName等。

**2.1.3 Camel风格**

包含一到多个单词，第一个单词首字母小写，其余单词首字母大写。例如：name、productId等。

**2.2 标识符的大小写规则**

**1)**除了参数与变量外，所有命名空间名称、类型（类、结构、委托、接口）、字段、属性、方法、事件的命名，优先考虑使用英文（尽量使用英文），如果实在没有合适的英文进行描述，可以使用拼音，使用中文是不符合要求的。而且使用 Pascal 风格。

例如： 类、结构和接口命名

1. .按照 Pascal 大小写格式，使用名词或名词短语为类、接口和值类型名。
2. **.**接口命名以字母 I 为前缀。

例如：IComponent

1. .派生类的末尾使用基类名称。

例如，从 Stream 继承的 Framework 类型以 Stream 结尾，从 Exception 继承的类型以 Exception 结尾。

**2)**参数与变量的命名，使用Camel风格。

**2.3  通用命名约定**

约定的是如何选择最适当的名称，这些准则适用于所有标识符命名。

**2.3.1 选择名称**

**1)**请选择易读的英文名称

例如，英文 Order的意思为规则、次序、订购等，如果用在排序列中就不是很合适，用来表示订单则更具可读性。

可读性比详细描述更重要，比如表示坐标名称ScreenX就比Screen Horizontally 更具可读性。

**2)**除下划线外，不要使用连字符或任何其他非字母数字字符。

在数据库表字段名称设计时，与其他表字段有关联时，适当的使用表名+下横线+字段名，可以更清晰的表现出该字段与关联表对应字段的关系。

比如产品分类表Product Class有字段Id与Name，那么产品表绑定这两个字段的名称可命名为ProductClass\_Id与ProductClass\_Name，这样在查看产品表时就可以清晰的知道这两个字段与分类表的关系。

**3)**避免使用与常用编程语言的关键字冲突的标识符。

**2.3.2 字母缩写词**

通常，不应使用缩写。所有类型、字段、属性、方法、事件尽量不使用缩写，包括大家熟知的缩写，例如msg。除非这种缩写已广泛接受，又或者团队当中大家都认可一种缩写。

例如，使用 OnButtonClick，如果团队中普遍认可OnBtnClick这种写法也是可以的。

**2.4 命名空间命名**

命名空间命名采用Pascal风格，取名的一般规则如下。

CompanyName. Project Name （公司名称.项目名称）

例如：Microsoft.Office

**1)**需要用复数时，请使用复数。

例如，使用System. Collections而不是System. Collection。

**2)**需要缩写时，不需要加复数。

例如：使用System.IO而不是System. IOS。

**2.5 逻辑层类命名**

按照 Pascal 大小写格式，使用名词或名词短语命名，并加上后缀Logic。

**2.6 文件夹命名**

文件夹以功能模块名称，按照 Pascal 大小写格式命名。

例如：后端管理功能以及权限相关功能，全部放到Systems文件夹里。

**3. 注释规范**

**3.1 模块（类）注释规范**

模块开始必须以以下形式书写模块注释：

///<summary>

   ///模块编号：<模块编号，可以引用系统设计中的模块编号>

   ///作用：<对此类的描述，可以引用系统设计中的描述>

   ///作者：作者中文名

   ///编写日期：<模块创建日期，格式：YYYY-MM-DD>

   ///</summary>

如果模块有修改，则每次修改必须添加以下注释：

     ///<summary>

     ///Log编号：<Log编号,从1开始一次增加>

     ///修改描述：<对此修改的描述>

     ///作者：修改者中文名

     ///修改日期：<模块修改日期，格式：YYYY-MM-DD>

     ///</summary>

**3.2 类属性注释规范**

在类的属性必须以以下格式编写属性注释：

///<summary>

///属性说明

///</summary>

**3.3 方法注释规范**

在类的方法声明前必须以以下格式编写注释

///<summary>

///说明：<对该方法的说明>

///</summary>

///<param name="<参数名称>"><参数说明></param>

///<returns>

///<对方法返回值的说明，该说明必须明确说明返回的值代表什么含义>

///</returns>

**3.4 代码间注释规范**

**3.4.1 代码间单行注释**

单行注释：

//<单行注释>

**3.4.2 代码间多行注释**

**1)**多行注释：

   /\*多行注释1

   多行注释2

   多行注释3\*/

**2)**代码行数太多而不容易区分时注释：

         /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

      \*   代码块功能名称

          \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**3)**或者可以如此书写：

/\*多行注释1

     多行注释2\*/

**4)**或者也可以使用下面方法：

/\*\*\*\*\*\*\*\*\* 代码块功能名称开始\*\*\*\*\*\*\*\*\*\*\*\*/

//<单行注释>

//<单行注释>

/\*\*\*\*\*\*\*\*\* 代码块功能名称结束\*\*\*\*\*\*\*\*\*\*\*\*/

**3.4.2 代码间注释说明**

**1)**代码中遇到语句块时必须添加注释（if,for,foreach,……）,添加的注释必须能够说明此语句块的作用和实现手段（所用算法等等）。对一个数值变量采用不是0,-1等的数值初始化，给出选择该值的理由。

**2)**尽量多点注释，就算能一目了然的命名最好也顺便写一写注释，方便以后接收的人能更容易理解程序（方便不太懂英文的程序员）。

**3)**如果因为某种原因使用了复杂艰涩的原理，为程序配备良好的文档和更多的注释。

**4.编码规范**

**1）**缩进和间隔：缩进用TAB，不用 SPACES。

**2）**注释需和代码对齐。多使用#regedit和#endregion代码块。

**3）**在代码中应注意左括号和右括号是否配对。

例如：

if (x == 0)

{

     Response. Write("用户编号必须输入！");

}

不允许以下情况：

if (x == 0) {

    {Response. Write("用户编号必须输入！");

}

或者：

if (x == 0) {Response. Write("用户编号必须输入！"); }}

**4）**适当的增加空行，来增加代码的可读性。

在下列情况下应该有两行空行：

     同一文件的不同部分之间；

     在类，接口以及彼此之间；

在下列情况之间应该有一行空行：

     方法之间；

     局部变量和它后边的语句之间；

     方法内的功能逻辑部分之间；

**5）**避免使用大文件。如果一个文件里的代码超过300～400行，必须考虑将代码分开到不同类中。当然模板生成类与逻辑层类除外。

**6）**避免写太长的方法。一个典型的方法代码在1～50行之间。如果一个方法发代码超过50行，应该考虑将其分解为不同的方法。

**7）**为了防止在阅读代码时不得不滚动源代码编辑器，为方便阅读，可要求每行代码或注释在1024\*768的显示频率下不得超过一显示屏

**8）**在大多数运算符之前和之后使用空格，这样做是不会改变代码的意图却可以使代码容易阅读。

例：

int j = i + k;

而不应写为

int j=i+k;

**9）**使用括号时该注意的事项

**①**括号和它里面的字符之间不应该出现空格。括号应该和它前边的关键词留有空格。

例：while (true)

{

};

**②**但是方法名和左括号之间不应该有空格，而且，参数之间的逗号后应该加一空格。

例：method1(int i1, int i2)

**③**for语句里的表达式之间加一空格。

例：for (expr1; expr2; expr3)

**④**强制类型转换时，在类型和变量之间加一空格。

例：(int) i;

**10）**所有可供用户输入的字段值，必须需忽略前后空白后(不包含密码)；在对字段值进行有效性验证。对提交进数据库的内容必须进行SQL注入过滤与XSS过滤。

**11）**一个方法只完成一个任务。不要把多个任务组合到一个方法中，即使那些任务非常小。

**12）**避免使用很多成员变量，声明局部变量，并传递给方法。

**13）**不要在方法间共享成员变量，如果在几个方法间共享一个成员变量，那就很难知道是哪个方法在什么时候修改了它的值。

**14）**不在代码中使用具体的路径和驱动器名，使用相对路径，并使路径可编程。永远别设想你的代码是在“C:”盘运行。你不会知道，一些用户在网络或“Z:”盘运行程序。

**15）**应用程序启动时作些“自检”并确保所需文件和附件在指定的位置。

如果需要的配置文件找不到，应用程序需能自己创建使用默认值的一份。如果在配置文件中发现错误值，应用程序要抛出错误，给出提示消息告诉用户正确值。

**16）**出现任何问题给用户一个友好的提示，错误消息需能帮助用户解决问题。

永远别用像“应用程序出错”，“发现一个错误”等错误消息。而应给出像“更新数据库失败，请确保登陆id和密码正确” 的具体消息。显示错误消息时，除了说哪里错了，还应提示用户如何解决问题。不要用像“更新数据库失败”这样的，要提示用户怎么做：“更新数据库失败，请确保登陆id和密码正确”

**17）**错误处理和异常事件

不要“捕捉了异常却什么也不做”。如果隐藏了一个异常，你将永远不知道异常到底发生了没有。

发生异常时，给出友好的消息给用户，但要精确记录错误的所有可能细节，包括发生的时间，和相关方法，类名等。

别写太大的 try-catch 模块。如果需要，为每个执行的任务编写单独的 try-catch 模块。 这将帮你找出哪一段代码产生异常，并给用户发出特定的错误消息

如果应用程序需要，可以编写自己的异常类。自定义异常不应从基类System Exception派生，而要继承于. IApplicationException。

**18）**类型名称和源文件名称一致

当类型命名为Product时，其源文件命名只能是Product.cs。

**19）**不在代码中使用具体的路径和驱动器名。 使用相对路径，并使路径可复

**20）**如果if语句块的内容只有一行，可以不加花括号，并且最好和if语句位于同一行，或者，给内容添加花括号{}。

**21）**类型成员的排列顺序

类型成员的排列顺序自上而下依次为：

字段：私有字段、受保护字段

属性：私有属性、受保护属性、公有属性

事件：私有事件、受保护事件、公有事件

构造函数：参数数量最多的构造函数，参数数量中等的构造函数，参数数量最少的构造函数

方法：重载方法的排列顺序与构造函数相同，从参数数量最多往下至参数最少

1. **C#语言编码**

从txt或者Excel文件中输入测量值（水平角、竖直角、起始坐标方位角、已知点的坐标以及斜距等等），求未知点的坐标：

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Data.OleDb;//添加类库

using System.IO;//添加类库 输入和输出

namespace \_20170630\_王星雨\_4

{

publicpartialclassForm1 : Form

{

public Form1()

{

InitializeComponent();

}

publicdouble dmstorad(string s)//封装角度转弧度的计算方法

{

string[] ss = s.Split(newchar[3] { '°', '′', '″' }, StringSplitOptions.RemoveEmptyEntries);

//用°、′、″将输入的角度值字符ss分为三个字符，并且删掉空格

double[] d = newdouble[ss.Length];//定义一个双精度数组d，存放ss的长度

for (int i = 0; i < d.Length; i++)

d[i] = Convert.ToDouble(ss[i]);//将分割后的每个字符利用循环结构转为双精度double类型

double sign = d[0] >= 0.0 ? 1.0 : -1.0;//判断首个数值是否大于零

double rad = 0;

if (d.Length == 1)

rad = Math.Abs(d[0]) \* Math.PI / 180;//若只有°前有数字，则转化后的弧度值等于该数字取绝对值后，乘以π，除以180

elseif (d.Length == 2)//当只有°、′前有数字的情况

rad = (Math.Abs(d[0]) + d[1] / 60) \* Math.PI / 180;

else//当°、′、″处都有数字时

rad = (Math.Abs(d[0]) + d[1] / 60 + d[2] / 60 / 60) \* Math.PI / 180;

rad = sign \* rad;//将°前的正负号“还”回来

return rad;//返回所得的弧度值

}

publicstring radtodms(double rad)//封装弧度转角度的计算方法

{

double sign = rad >= 0.0 ? 1.0 : -1.0;//判断弧度的正负

rad = Math.Abs(rad) \* 180 / Math.PI;//将弧度转化为角度值

double[] d = newdouble[3];//定义一个双精度数组d

d[0] = (int)rad;//取整得度

d[1] = (int)((rad - d[0]) \* 60);//将弧度值减去°后，求′，并保留整数

d[2] = (rad - d[0] - d[1] / 60) \* 60 \* 60;//将弧度值减去°、′后，求″，不取整，保留精度

d[2] = Math.Round(d[2], 2);//取″保留精度后的两位小数

if (d[2] == 60) //特殊情况判断，若″前数字为60

{

d[1] += 1;//向′进一位

d[2] -= 60;//″的值再减去60

if (d[1] == 60)//若′前数字为60

{

d[0] += 1;//向°进一位

d[1] -= 60;//′的值再减去60

}

}

d[0] = sign \* d[0];//将弧度值的正负号再“还给”角度值

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

//定义一个字符串s，存放将所得的°、′、″前的数字加上符号后整合后，并双精度类型转化为字符串类型后的值

return s;//返回s的值

}

publicdouble fangweijiaotuisuan(double[] sdr, double [] cr)//封装推算坐标方位角的计算方法

{

double sum = 0;//定义一个双精度变量sum，初值为0

for (int i=1; i<sdr.Length;i++)//计算坐标方位角的循环结构

{

cr[i] = cr[i - 1] + sdr[i] - Math.PI;

//坐标方位角计算公式，坐标方位角=上一个角度的坐标方位角+此时的观测角度（左角）-π

if (cr[i] >= Math.PI \* 2) cr[i] -= Math.PI \* 2;//特殊情况判断，当坐标方位角>360°时，应减去360°

elseif (cr[i] < 0.0) cr[i] += Math.PI \* 2;//特殊情况判断，当坐标方位角<0°时，应加上360°

sum += sdr[i];//求总和

}

return sum;//返回sum的最终结果

}

privatevoid button1\_Click(object sender, EventArgs e)//计算主程序

{

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

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

double[] cr = newdouble[sdr.Length];//新建一个数组，存放计算的坐标方位角

double sum = 0;//定义一个双精度变量，，存放角度总值，初值为0

cr[0] = dmstorad(Convert.ToString(dataGridView1.Rows[0].Cells[4].Value));

//获取第一个坐标方位角，并将其转化为弧度，放入cr[]数组第一个元素中

double acd = dmstorad(Convert.ToString(dataGridView1.Rows[dataGridView1.RowCount - 6].Cells[4].Value));

//获取中边坐标方位角，并将其转化为弧度，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，并转化成弧度值存放在sdr数组中

{

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

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

}

sum = fangweijiaotuisuan(sdr, cr);//计算改正前坐标方位角和观测角度的总和，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;//定义两个双精度类型的变量fd,fdx

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

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

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（观测左角）

double sumvd = 0;

for (int i = 1; i<sdr.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))//秒保留2位对应弧度是8位

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 = fangweijiaotuisuan(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);

//将改正后的观测角度总和存入表格

for (int i =1; i< cr.Length-1; i++)//将改正后的坐标方位角存入表格

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

}

}

}

privatevoid button2\_Click(object sender, EventArgs e)//“关闭”按钮的程序

{

Application.Exit();//关闭窗口

}

privatevoid excel文件ToolStripMenuItem\_Click\_1(object sender, EventArgs e)//Excel文件导入程序

{

dataGridView1.DataSource = null;//清除数据源，清空表格里的所有数据

dataGridView1.Rows.Clear();//清空行

dataGridView1.Columns.Clear();//清空列

OpenFileDialog file = newOpenFileDialog();//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，限制打开的文件后缀只能为xls或xlsx

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

{

string fname = file.FileName;//定义一个字符串fname，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 = newOleDbConnection(strSource);//Excel文件源放入conn中

string sqlstring = "SELECT \* FROM [Sheet1$]";//准备选择表中的sheet1

OleDbDataAdapter adapter = newOleDbDataAdapter(sqlstring, conn);//声明数据配置器adapter

DataSet da = newDataSet();//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;

}

privatevoid txt文件ToolStripMenuItem\_Click\_1(object sender, EventArgs e)//txt文件导入程序

{

//txt输入代码

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

dataGridView1.Rows.Clear();//清空数据表格的行

dataGridView1.Columns.Clear();//清空数据表格的列

OpenFileDialog file = newOpenFileDialog();//声明 打开文件类file

file.Filter = "文本文件|\*.txt";//文件过滤器，只显示txt文件

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

{

StreamReader sr = newStreamReader(file.FileName, System.Text.Encoding.Default);

//声明文本读取流，并以文本编码格式读取

textBox1.Text = sr.ReadToEnd();//将sr中的内容全部放到textBox中

sr.Close();

}

else

return;

//textBox1.Text 存入数组，然后存入dataGridView1

string[] str = textBox1.Text.Split(newstring[] { "\r\n" }, StringSplitOptions.RemoveEmptyEntries);

//将textBox1.Text中按行分割，并放在一维字符串数组中

string[][] k = newstring[str.Length][];//定义字符串交错数组，行数与str的长度相同

for (int i = 0; i < str.Length; i++)

k[i] = str[i].Split(',');//将str中对应的字符串以逗号（英文状态）分割，并放在k中

dataGridView1.RowCount = k.Length;//定义表格控件的行数，与str长度相同

dataGridView1.ColumnCount = k[0].Length;//定义表格列数，与k[0]长度相同

for (int i = 0; i < k[0].Length; i++)

dataGridView1.Columns[i].HeaderText = k[0][i];//将k中第0行元素放入表格的表头

for (int i = 1; i < k.Length; i++)//将k中数据元素放入对应的表格中

{

for (int j = 0; j < k[i].Length; j++)

dataGridView1.Rows[i - 1].Cells[j].Value = k[i][j];

}

}

privatevoid dataGridView1\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

}

privatevoid Form1\_Load(object sender, EventArgs e)

{

}

privatevoid textBox1\_TextChanged(object sender, EventArgs e)

{

}

}

}