1、规范目的

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。为了执行规范，每个软件开发人员必须一致遵守编码规范；
4. 使用统一编码规范的主要原因，是使应用程序的结构和编码风格标准化，以便于阅读和理解这段代码；
5. 好的编码约定可使源代码严谨、可读性强且意义清楚，与其它语言约定相一致，并且尽可能的直观。

2、适用范围

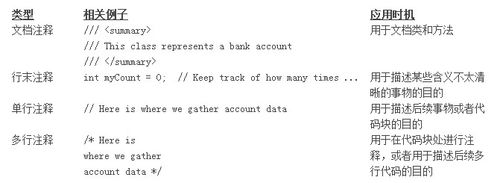
1. 本规范主要以C#为开发语言的规范；
2. 由于本规范是为撰写程序而设计，所以适用于一切有关程序撰写的工作事项。对于具体的每个项目，可能需要对之进行裁剪和补存。
3. 适用人员：软件工程专业的学生；
4. 适用产品：以C#编写的程序。

3 、代码行注释规范

　　1>  如果处理某一个功能需要很多行代码实现，并且有很多逻辑结构块，类似此种代码应该在代码开始前添加注释，说明此块代码的处理思路及注意事项等

　　2>  注释从新行增加，与代码开始处左对齐

3>  双斜线与注释之间以空格分开，

4>**建议**先写注释，后写代码，注释和代码一起完成

4、命名的基本约定

　　1>  要使用可以准确说明变量/字段/类的完整的英文描述符，如firstName。对一些作用显而易见的变量可以采用简单的命名，如在循环里的递增（减）变量就可以被命名为 “i”。

　　2>  要尽量采用项目所涉及领域的术语。

　　3>  要采用大小写混合，提高名字的可读性。为区分一个标识符中的多个单词，把标识符中的每个单词的首字母大写。不采用下划线作分隔字符的写法。

　　　　有两种适合的书写方法，适应于不同类型的标识符：

　　　　　　PasalCasing：标识符的第一个单词的字母大写；

　　　　　　camelCasing：标识符的第一个单词的字母小写。

　5>  避免使用缩写，如果一定要使用，就谨慎使用。同时，应该保留一个标准缩写的列表，并且在使用时保持一致。

　　6>  对常见缩略词，两个字母的缩写要采用统一大小写的方式（示例：ioStream， getIOStream）；多字母缩写采用首字母大写，其他字母小写的方式（示例： getHtmlTag）；

　　7>  避免使用长名字（最好不超过 15 个字母）。

8>  避免使用相似或者仅在大小写上有区别的名字。

5、变量声明

　　1>为了保持更好的阅读习惯，请不要把多个变量声明写在一行中，即一行只声明一个变量。

　　例如：

　　String strTest1, strTest2;

　　应写成：

　　String strTest1;

　　String strTest2;

2>避免方法中有超过5个参数的情况，一般以2,3个为宜。如果超过了，则应使用struct来传递多个参数。

　　3>为了更容易阅读，代码行请不要太长，最好的宽度是屏幕宽度（根据不同的显示分辨率其可见宽度也不同）。请不要超过您正在使用的屏幕宽度。（每行代码不要超过80个字符。）

　　4>程序中不应使用goto语句。

　　5>在switch语句中总是要default子句来显示信息。

　　6>方法参数多于8个时采用结构体或类方式传递

　　7>操作符/运算符左右空一个半角空格

8>所有块的{}号分别放置一行，并嵌套对齐，不要放在同一行上

9>每行只有一个声明，如果是声明i,j,k之类的简单变量可以放在一行;

9>除了for循环外，声明要放在块的最开始部分。for循环中的变量声明可以放在for语句中。如：for(int i = 0; I < 10; i++) 。

´ 避免块内部的变量与它外部的变量名相同。

6、错误处理

　　1>  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。

　　2>  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。

　　3>  只捕捉特定的异常，而不是一般的异常。

7、 其它

　　1>  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。

　　2>  使用C#的特有类型，而不是System命名空间中定义的别名类型。

　　3>  别在程序中使用固定数值，用常量代替。

　　4>  避免使用很多成员变量。声明局部变量，并传递给方法。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。

　　5>  别把成员变量声明为 public 或 protected。都声明为 private 而使用 public/protected 的属性

　　6>  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。

　　7>  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。必要时检查数据库连接。出现任何问题给用户一个友好的提示。

　　8>  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。

　　9>  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。

　　10>  DataColumn取其列时要用字段名，不要用索引号。  
　　　　例： 正确DataColumn[“Name”]  
    　　　　  错误 DataColumn[0]

　　11>  在一个类中，字段定义全部统一放在class的头部、所有方法或属性的前面。

12>  在一个类中，所有的属性全部定义在一个属性块中：

8、空白：

　1> 空行将逻辑相关的代码段分隔开，以提高可读性。

　2> 下列情况应该总是使用两个空行：

　　　　a)  一个源文件的两个片段(section)之间。

　　　　b)  类声明和接口声明之间。

　3>下列情况应该总是使用一个空行：

　　　　a)  两个方法之间。

　　　　b)  方法内的局部变量和方法的第一条语句之间。

　　　　c)  块注释（参见"5.1.1"）或单行注释（参见"5.1.2"）之前。

　　　　d)  一个方法内的两个逻辑段之间，用以提高可读性。

　4> 下列情况应该总是使用空格：

　　　　a)  空白应该位于参数列表中逗号的后面，如：

　　　　　　void UpdateData(int a, int b)

　　　　b)  所有的二元运算符，除了"."，应该使用空格将之与操作数分开。一元操作符和操作数之间不因该加空格，比如：负号("-")、自增("++")和自减("--")。例如：

　　　　　　　　a += c + d;

　　　　　　　　d++;

　　　　c)  for 语句中的表达式应该被空格分开，例如：

　　　　　　　　for (expr1; expr2; expr3)

　　　　d)  强制转型后应该跟一个空格，例如：

　　　　　　　　char c;

　　　　　　　　int a = 1;

　　　　　　　　c = (char) a;

private void button2\_Click(object sender, EventArgs e)

{

double d, r;

d = Convert.ToDouble(textBox1.Text);

r = d \* Math.PI / 180;

textBox2.Text = Convert.ToString(r);

}

private void button3\_Click(object sender, EventArgs e)

{

double d, r;

r = Convert.ToDouble(textBox2.Text);

d = r \* 180 / Math.PI;

textBox1.Text = Convert.ToString(d);

}

private void button4\_Click(object sender, EventArgs e)

{

Application.Exit()；

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

}