Министерство цифрового развития, связи и массовых коммуникаций
Российской Федерации Сибирский Государственный Университет
Телекоммуникаций и Информатики СибГУТИ

Кафедра прикладной математики и кибернетики

Лабораторная работа №7
по дисциплине “Современные технологии программирования”
Абстрактный тип данных р-ичное число

Выполнил:
Студент группы ИП-916
Меньщиков Д.А.

Работу проверил:
Агалаков А. А.

Новосибирск, 2022

Задание

1. Реализовать абстрактный тип данных «р-ичное число», используя класс, в соответствии с приведенной ниже спецификацией.
2. Протестировать каждую операцию, определенную на типе данных по критерию С2, используя средства модульного тестирования Visual Studio.
3. Если необходимо, предусмотрите возбуждение исключительных ситуаций.

Выполнение лабораторной работы

Код программы:

```
namespace lab7
{
    public class PNumber
    {
        double num = 0.0;
        double numBase = 0.0;
        double accuracy = 0.0;
        public PNumber(double a_ = 0, double b_ = 0, double c_ = 0)
        {
            if (b_ < 2 || b_ > 16)
            {
                throw new Exception("Base must be in range [2..16]");
            }

            num = a_;
            numBase = b_;
            accuracy = c_;
        }

        public PNumber(string str_)
        {
            var delimeterPos = str_.Split(",");

            double num_ = double.Parse(delimeterPos[0]);
            int base_ = int.Parse(delimeterPos[1]);
            int accuracy_ = int.Parse(delimeterPos[2]);

            if (base_ < 2 || base_ > 16)
            {
                throw new Exception("Base must be in range [2..16]");
            }

            num = num_;
            numBase = base_;
            accuracy = accuracy_;
        }
```

```
43
44          public static PNumber operator +(PNumber lhs, PNumber rhs)
45          {
46              if (lhs.numBase != rhs.numBase && lhs.accuracy != rhs.accuracy)
47              {
48                  throw new Exception("Base and accuracy must be equals");
49              }
50
51              return new PNumber(lhs.num + rhs.num, lhs.numBase, lhs.accuracy);
52          }
53
54          public static PNumber operator -(PNumber lhs, PNumber rhs)
55          {
56              if (lhs.numBase != rhs.numBase && lhs.accuracy != rhs.accuracy)
57              {
58                  throw new Exception("Base and accuracy must be equals");
59              }
60
61              return new PNumber(lhs.num - rhs.num, lhs.numBase, lhs.accuracy);
62          }
63
64          public static PNumber operator *(PNumber lhs, PNumber rhs)
65          {
66              if (lhs.numBase != rhs.numBase && lhs.accuracy != rhs.accuracy)
67              {
68                  throw new Exception("Base and accuracy must be equals");
69              }
70
71              return new PNumber(lhs.num * rhs.num, lhs.numBase, lhs.accuracy);
72          }
73
74          public static PNumber operator /(PNumber lhs, PNumber rhs)
75          {
76              if (lhs.numBase != rhs.numBase && lhs.accuracy != rhs.accuracy)
77              {
78                  throw new Exception("Base and accuracy must be equals");
79              }
80
81              return new PNumber(lhs.num / rhs.num, lhs.numBase, lhs.accuracy);
82          }
```

```
83
84          public static bool operator ==(PNumber lhs, PNumber rhs)
85          {
86              return lhs.num == rhs.num && lhs.numBase == rhs.numBase && lhs.accuracy == rhs.accuracy;
87          }
88
89          public static bool operator !=(PNumber lhs, PNumber rhs)
90          {
91              return lhs.num != rhs.num || lhs.numBase != rhs.numBase || lhs.accuracy != rhs.accuracy;
92          }
93
94          public static PNumber Revers(PNumber lhs)
95          {
96              return new PNumber(1 / lhs.num, lhs.numBase, lhs.accuracy);
97          }
98
99          public static PNumber Pow(PNumber lhs, int degree = 2)
100         {
101             return new PNumber(Math.Pow(lhs.num, degree), lhs.numBase, lhs.accuracy);
102         }
103
104         public static double GetNum(PNumber lhs)
105         {
106             return lhs.num;
107         }
108
109         public static string GetString(PNumber lhs)
110         {
111             return $"{lhs.num}, {lhs.numBase}, {lhs.accuracy}";
112         }
113
114         public static double GetBase(PNumber lhs)
115         {
116             return lhs.numBase;
117         }
118
119         public static string GetBaseString(PNumber lhs)
120         {
121             return $"{lhs.numBase}";
122         }
```

```
123
124         public static double GetAccuracy(PNumber lhs)
125         {
126             return lhs.accuracy;
127         }
128
129         public static string GetAccuracyString(PNumber lhs)
130         {
131             return $"{lhs.accuracy}";
132         }
133
134         public void SetBase(double newBase)
135         {
136             if (newBase < 2 || newBase > 16)
137             {
138                 throw new Exception("Base must be in range [2..16]");
139             }
140
141             if (newBase < numBase)
142             {
143                 throw new Exception("Base must be bigger");
144             }
145
146             numBase = newBase;
147         }
148
149         public void SetBase(string newBase_)
150         {
151             int newBase = int.Parse(newBase_);
152
153             if (newBase < 2 || newBase > 16)
154             {
155                 throw new Exception("Base must be in range [2..16]");
156             }
157
158             if (newBase < numBase)
159             {
160                 throw new Exception("Base must be bigger");
161             }
162
163             numBase = newBase;
164         }
```

```
165
166         public void SetAccuracy(double newAccuracy)
167         {
168             if (newAccuracy < 0)
169             {
170                 throw new Exception("Base must be higher than zero");
171             }
172
173             accuracy = newAccuracy;
174         }
175
176         public void setAccuracy(string accuracy_)
177         {
178             int newAccuracy = int.Parse(accuracy_);
179
180             if (newAccuracy < 0)
181             {
182                 throw new Exception("Base must be higher than zero");
183             }
184
185             accuracy = newAccuracy;
186         }
187
188         public void Show()
189         {
190             Console.WriteLine($"Number: {num}");
191             Console.WriteLine($"Number: {numBase}");
192             Console.WriteLine($"Number: {accuracy}");
193         }
194     }
195 }
```

Модульные тесты:

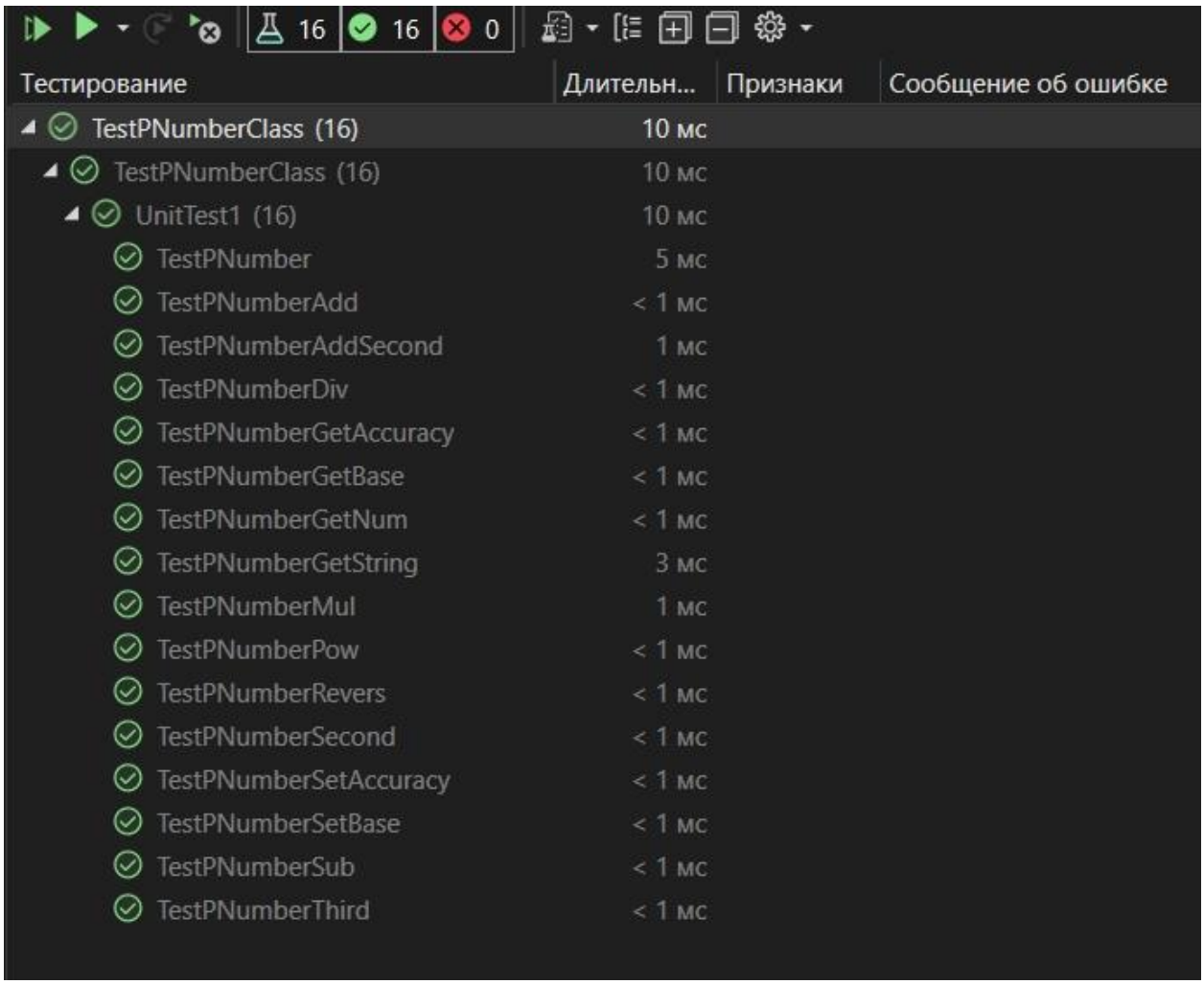
```
5   namespace TestPNumberClass
6   {
7       [TestClass]
8       public class UnitTest1
9       {
10          [TestMethod]
11          public void TestPNumber()
12          {
13              var _ = new PNumber(1, 2, 3);
14          }
15
16          [TestMethod]
17          [ExpectedException(typeof(Exception))]
18          public void TestPNumberSecond()
19          {
20              var _ = new PNumber(1, 0, 3);
21          }
22
23          [TestMethod]
24          [ExpectedException(typeof(Exception))]
25          public void TestPNumberThird()
26          {
27              var _ = new PNumber("1, 29, 1");
28          }
29
30          [TestMethod]
31          public void TestPNumberAdd()
32          {
33              PNumber a = new PNumber(1, 2, 3);
34              PNumber b = new PNumber(5, 2, 3);
35
36              PNumber actual = a + b;
37              PNumber expected = new PNumber(6, 2, 3);
38
39              Assert.IsTrue(expected == actual);
40          }
```

```
41
42          [TestMethod]
43          [ExpectedException(typeof(Exception))]
44          public void TestPNumberAddSecond()
45          {
46              PNumber a = new PNumber(1, 1, 3);
47              PNumber b = new PNumber(5, 2, 3);
48
49              PNumber actual = a + b;
50              PNumber expected = new PNumber(6, 2, 3);
51
52              Assert.IsTrue(actual == expected);
53          }
54
55          [TestMethod]
56          public void TestPNumberSub()
57          {
58              PNumber a = new PNumber(0, 2, 3);
59              PNumber b = new PNumber(1, 2, 3);
60
61              PNumber actual = a - b;
62              PNumber expected = new PNumber(-1, 2, 3);
63
64              Assert.IsTrue(actual == expected);
65          }
66
67          [TestMethod]
68          public void TestPNumberMul()
69          {
70              PNumber a = new PNumber(2, 2, 3);
71              PNumber b = new PNumber(1, 2, 3);
72
73              PNumber actual = a * b;
74              PNumber expected = new PNumber(2, 2, 3);
75
76              Assert.IsTrue(actual == expected);
77          }
```

```
79          [TestMethod]
80          public void TestPNumberDiv()
81          {
82              PNumber a = new PNumber(2, 2, 3);
83              PNumber b = new PNumber(1, 2, 3);
84
85              PNumber actual = a / b;
86              PNumber expected = new PNumber(2, 2, 3);
87
88              Assert.IsTrue(actual == expected);
89          }
90
91          [TestMethod]
92          public void TestPNumberPow()
93          {
94              PNumber a = new PNumber(2, 2, 3);
95
96              PNumber actual = PNumber.Pow(a, 2);
97              PNumber expected = new PNumber(4, 2, 3);
98
99              Assert.IsTrue(actual == expected);
100         }
101
102         [TestMethod]
103         public void TestPNumberRevers()
104         {
105             PNumber a = new PNumber(2, 2, 3);
106
107             PNumber actual = PNumber.Revers(a);
108             PNumber expected = new PNumber(1.0 / 2, 2, 3);
109
110             Assert.IsTrue(actual == expected);
111         }
```

```
113         [TestMethod]
114         public void TestPNumberGetNum()
115         {
116             PNumber a = new PNumber(2, 2, 3);
117 
118             var actual = PNumber.GetNum(a);
119             var expected = 2;
120 
121             Assert.IsTrue(actual == expected);
122         }
123 
124         [TestMethod]
125         public void TestPNumberGetString()
126         {
127             PNumber a = new PNumber(2, 2, 3);
128 
129             var actual = PNumber.GetString(a);
130             var expected = "2, 2, 3";
131 
132             Assert.IsTrue(actual == expected);
133         }
134 
135         [TestMethod]
136         public void TestPNumberGetBase()
137         {
138             PNumber a = new PNumber(2, 2, 3);
139 
140             var actual = PNumber.GetBase(a);
141             var expected = 2;
142 
143             Assert.IsTrue(actual == expected);
144         }
145 
146         [TestMethod]
147         public void TestPNumberGetAccuracy()
148         {
149             PNumber a = new PNumber(2, 2, 3);
150 
151             var actual = PNumber.GetAccuracy(a);
152             var expected = 3;
153 
154             Assert.IsTrue(actual == expected);
155         }
```

```
156
157         [TestMethod]
158         public void TestPNumberSetBase()
159         {
160             PNumber actual = new PNumber(2, 2, 3);
161
162             actual.SetBase(5);
163             var expected = new PNumber(2, 5, 3);
164
165             Assert.IsTrue(actual == expected);
166         }
167
168         [TestMethod]
169         public void TestPNumberSetAccuracy()
170         {
171             PNumber actual = new PNumber(2, 2, 3);
172
173             actual.SetAccuracy(5);
174             var expected = new PNumber(2, 2, 5);
175
176             Assert.IsTrue(actual == expected);
177         }
178     }
179 }
```


16
16
0
Тестирование
Длительн...
Признаки
Сообщение об ошибке
TestPNumberClass (16)
10 мс
TestPNumberClass (16)
10 мс
UnitTest1 (16)
10 мс
TestPNumber
5 мс
TestPNumberAdd
< 1 мс
TestPNumberAddSecond
1 мс
TestPNumberDiv
< 1 мс
TestPNumberGetAccuracy
< 1 мс
TestPNumberGetBase
< 1 мс
TestPNumberGetNum
< 1 мс
TestPNumberGetString
3 мс
TestPNumberMul
1 мс
TestPNumberPow
< 1 мс
TestPNumberRevers
< 1 мс
TestPNumberSecond
< 1 мс
TestPNumberSetAccuracy
< 1 мс
TestPNumberSetBase
< 1 мс
TestPNumberSub
< 1 мс
TestPNumberThird
< 1 мс