## Практическая работа № 6

### Создание многопоточного Windows-приложения

На этом практическом занятии мы реализуем многопоточное Windows приложение, которое асинхронно выполняет три различных метода.

- Первый метод отрисовывает в бесконечном цикле прямоугольники.
- Второй метод отрисовывает в бесконечном цикле эллипсы.
- Третий метод генерировал бы N число символов и записывал бы их в TextBox.

1. Создадим проект Windows Form Application и назовем его «MultithreadingWinFormApplication»:

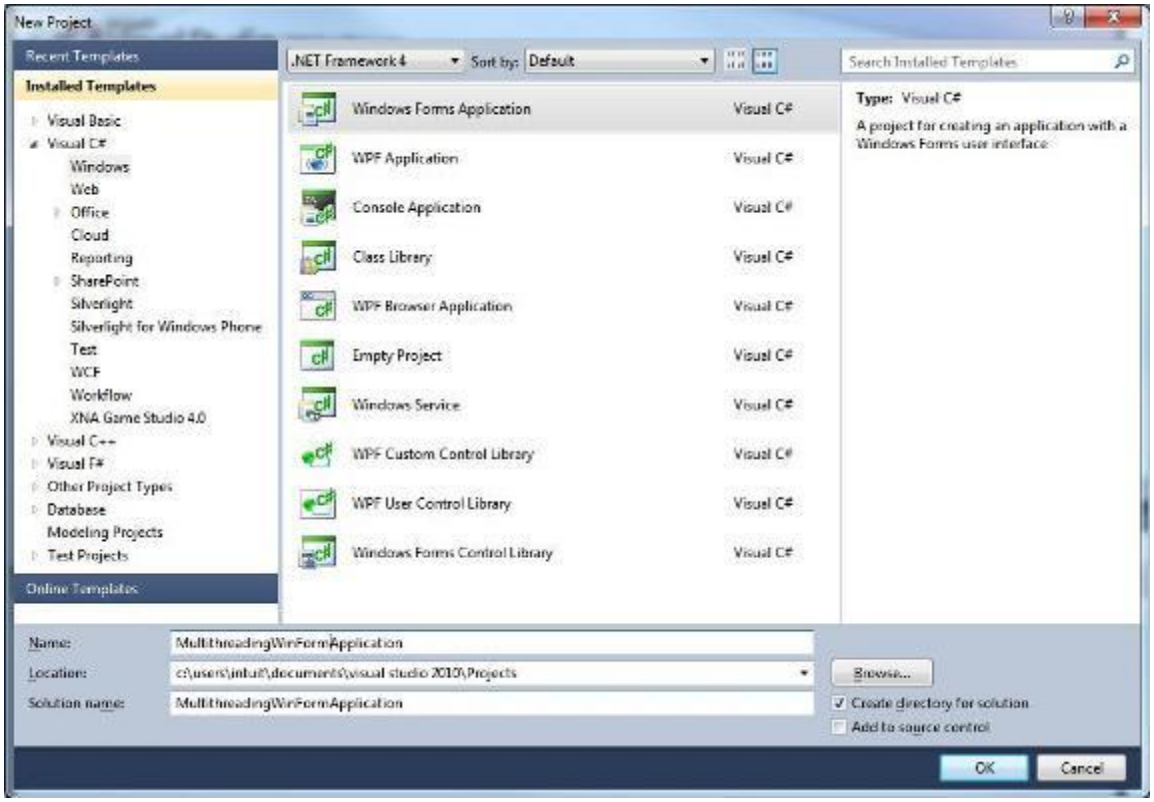


**Рис .1.**

2. В созданном новом проекте разместите следующие элементы управления следующим образом:


**Рис. 2.**

На главной форме располагаются следующие ЭУ:

- 4 Button - ЭУ кнопка, для управления потоками.
- 2 Panel - ЭУ панель, для отрисовки графических объектов (Rectangle,Ellipse).
- 1 TextBox (Multiline) -ЭУ текстовое поле, для вывода 500 случайно сгенерированных чисел.

3. Реализуем следующий код в методах:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;
namespace MultithreadingWinFormApplication
{
   public partial class Form1 : Form
  {
     public Form1()
     {
         InitializeComponent();

      }

     private void DrawRectangle()
      {
         try
         {
            Random rnd = new Random();
            Graphics g = panel1.CreateGraphics();
            while (true)
```

```
            {
                Thread.Sleep(4);
                g.DrawRectangle(Pens.Pink, 0, 0, rnd.Next(this.Width), rnd.Next(this.Height));
            }
        }
        catch (Exception ex) { }
    }
    private void DrawEllipse()
    {
        try
        {
            Random rnd = new Random();
            Graphics g = panel2.CreateGraphics();
            while (true)
            {
                Thread.Sleep(4);
                g.DrawEllipse(Pens.Yellow, 0, 0, rnd.Next(this.Width), rnd.Next(this.Height));
            }
        }
        catch (Exception ex) { }
    }
    private void RandomNumber()
    {
     try
        {
        Random rnd = new Random();
        for (int i = 0; i < 500; i++)
        {
            RandomNumberTextBox.Text += rnd.Next().ToString();
        }

        }
        catch (Exception ex) { MessageBox.Show(ex.Message); }
    }
    private void StartThreadingButton_Click(object sender, EventArgs e)
    {
            DrawRectangle();
            DrawEllipse();
            RandomNumber();
    }
private void FirstMethodButton_Click(object sender, EventArgs e)
    {
        DrawRectangle();
    }
    private void SecondMethodButton_Click(object sender, EventArgs e)
    {
        DrawEllipse();
    }
    private void ThirdMethodButton_Click(object sender, EventArgs e)
    {
        RandomNumber();
    }
```

```
    }
}
```

Метод DrawRectangle() - данный метод выполняет отрисовку прямоугольников, в элементе управления Panel1, в бесконечном цикле while;

Метод DrawEllipse() - данный метод выполняет отрисовку эллипсов, в элементе управления Panel2, в бесконечном циклewhile;

Метод RandomNumber() - данный метод генерирует 500 случайных чисел и записывает их в элемент управления TextBox;

Метод StartThreadingButton_Click() - обработчик события, который пытается запустить все методы (именно методы, не потоки!) одновременно (методы отрисовки прямоугольников, эллипсов и генератор случайных чисел) при нажатии на кнопку StartThreadingButton;

Метод FirstMethodButton_Click() - обработчик события, который запускает метод DrawRectangle() при нажатии на кнопкуFirstMethodButton;

Метод SecondMethodButton_Click() - обработчик события, который запускает метод DrawEllipse() при нажатии на кнопкуSecondMethodButton;

Метод ThirdMethodButton_Click() - обработчик события, который запускает метод RandomNumber() при нажатии на кнопкуThirdMethodButton.

4. Запустим программу нажатием кнопки «Отрисовка прямоугольников». Запустится процесс отрисовки прямоугольников в первой панели (panel1), при этом все остальные ЭУ и сама форма будут не доступны для каких либо действий, т.к программа выполняет действие в один поток, который полностью "уходит" в бесконечный цикл, что приводит к зависанию формы и соответственно программы:

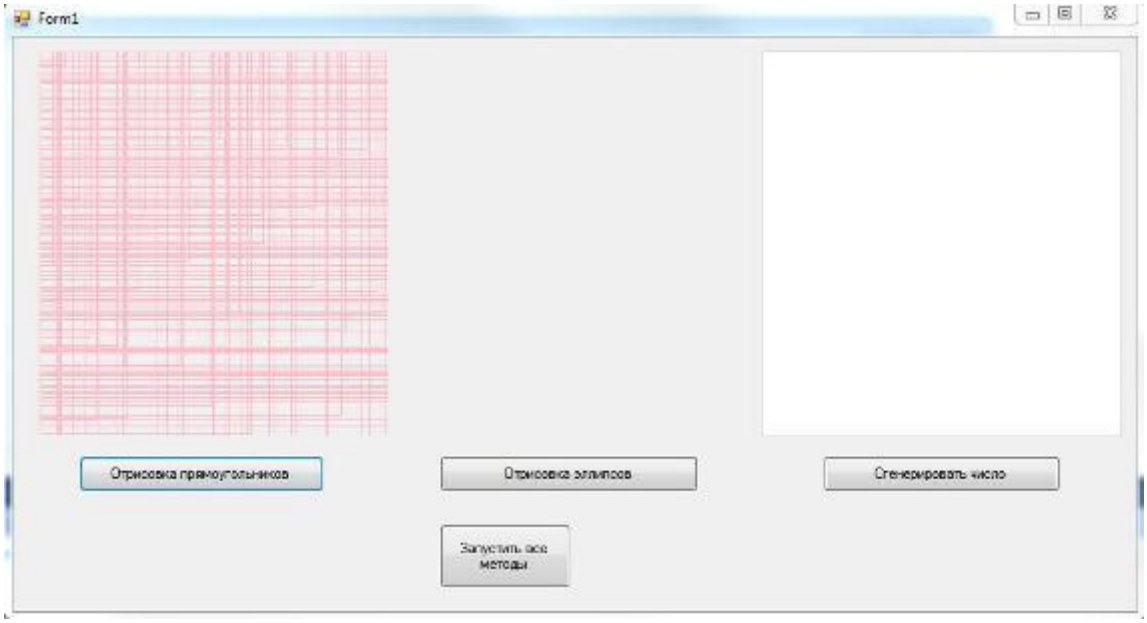


**Рис. 3.**

5. Для того чтобы программа не зависала, необходимо распараллелить выполнение методов:

```
namespace MultithreadingWinFormApplication
{
```

```
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            thread1 = new Thread(new ThreadStart(DrawRectangle));
            thread2 = new Thread(new ThreadStart(DrawEllipse));
            thread3 = new Thread(new ThreadStart(RandomNumber));
        }
        Thread thread1;
        Thread thread2;
        Thread thread3;
}
       private void StartThreadingButton_Click(object sender, EventArgs e)
        {
            try
            {
                thread1.Start();
                thread2.Start();
                thread3.Start();
            }
            catch (Exception ex) { }
        }
        private void FirstMethodButton_Click(object sender, EventArgs e)
        {

            thread1.Start();

        }
        private void SecondMethodButton_Click(object sender, EventArgs e)
        {

            thread2.Start();

        }
        private void ThirdMethodButton_Click(object sender, EventArgs e)
        {

            thread3.Start();
         }
```

Как видно из когда, теперь методы будут выполняться в три различных потока:

- thread1 - вызывает метод DrawRectangle();
- thread2 -вызывает метод DrawEllipse();
- thread3 - вызывает метод RandomNumber().

6. Запустим программу. Первые два метода, по отрисовке рисунка (с использованием бесконечных циклов) отработают нормально, и при этом независимо друг от друга:

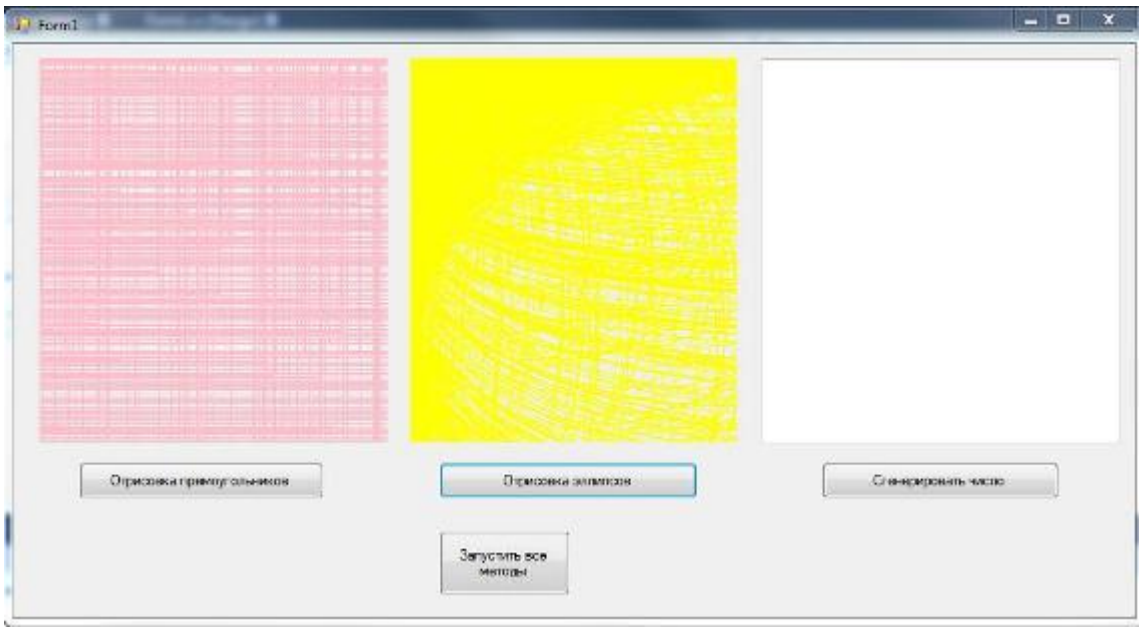


**Рис. 4.**

Третий метод, при выполнение, выдаст ошибку при обращении к элементу TextBox из дочернего потока:

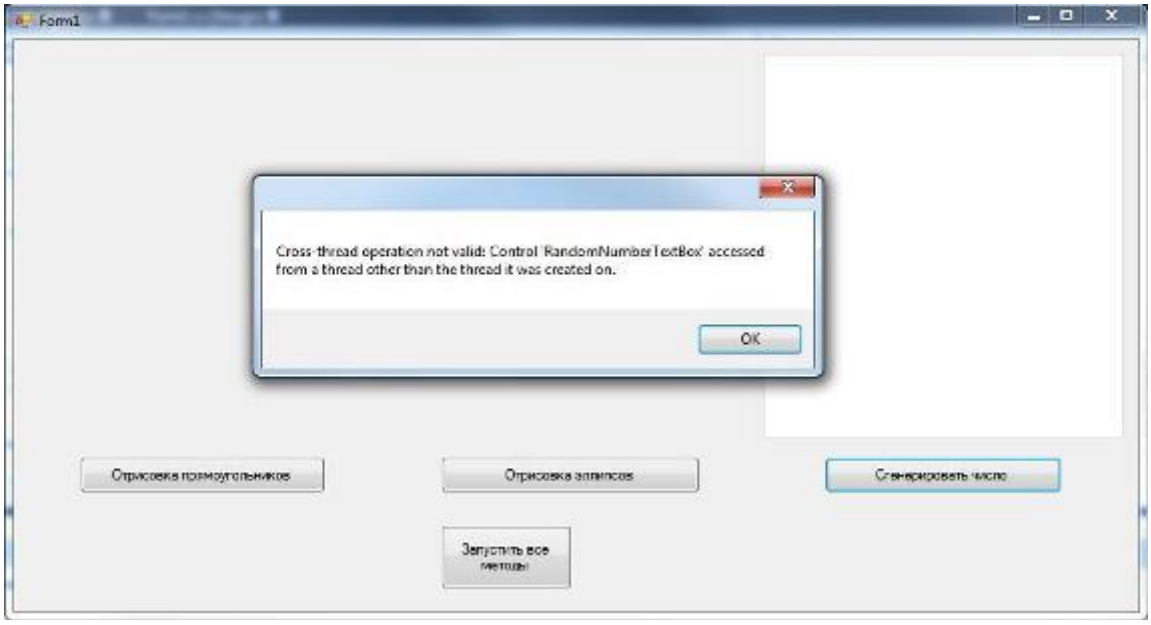


**Рис. 5.**

**Примечание.** Визуальные компоненты WinForms устроены таким образом, что доступ к ним разрешается только из главного потока. У каждого наследника Control (в нашем случае TextBox), есть метод Invoke, который "заставляет" главный поток выполнить метод указанный в Invoke.

7. Для доступа к элементу TextBox необходимо использовать специальный метод Invoke (см примечание):

```
private void RandomNumber()
      {
         try
         {
         Random rnd = new Random();
         RandomNumberTextBox.Invoke((MethodInvoker)delegate()
        {
           for (int i = 0; i < 500; i++)
```

```
            {
                RandomNumberTextBox.Text += rnd.Next().ToString();
            }
        });

        }
        catch (Exception ex) { MessageBox.Show(ex.Message); }
    }
```

8. Повторно запустим программу и нажмем на кнопку «Запустить все методы». В итоге в программе отобразится следующее:

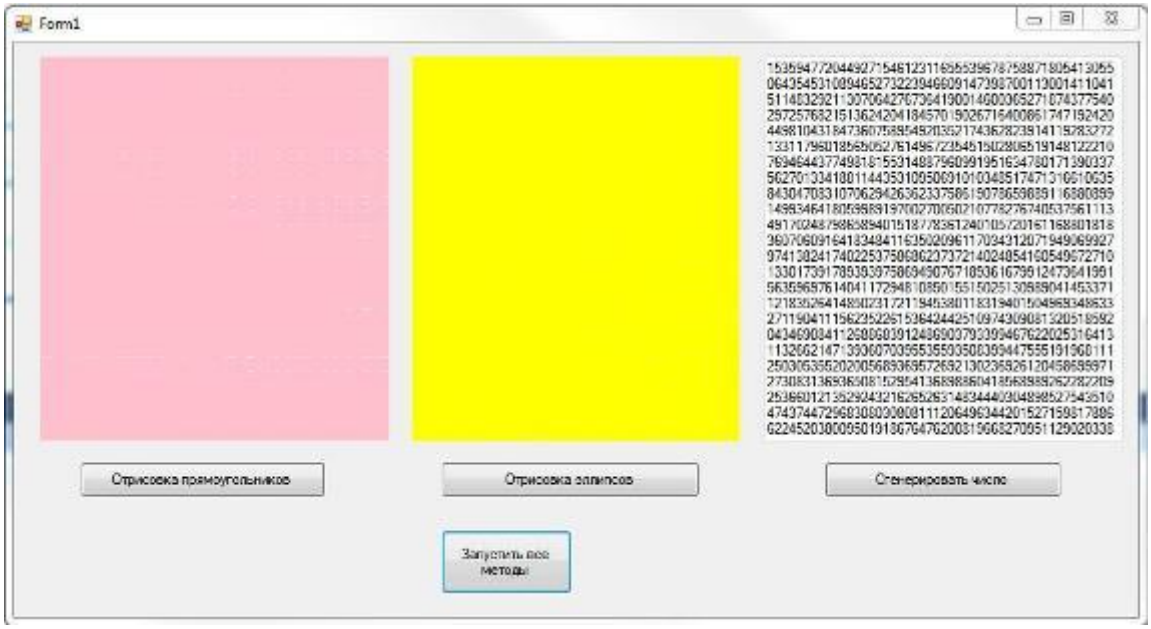


**Рис. 6.**

9. Выполнение метода RandomNumber(), который генерирует 500 случайных чисел и записывает их в TextBox, выполняется медленно, и будет целесообразно распараллелить выполнение генерирования случайных чисел в цикле for(). Для этого используем цикл Parallel.For():

```
private void RandomNumber()
    {
        try
        {
        Random rnd = new Random();

        Parallel.For(0, 500, i =>
              {   RandomNumberTextBox.Invoke((MethodInvoker)delegate()
               {
                RandomNumberTextBox.Text += rnd.Next().ToString();
               });
        });

        }
       catch (Exception ex) { MessageBox.Show(ex.Message); }
     }
```

10. Для того, что бы была возможность штатно останавливать выполнение методов в различных потоках на форме, разместим на форме – еще одну кнопку:


**Рис. 7.**

11. Добавим кнопке StopThreadsButton – событие StopThreadsButton_Click() в котором вызовем метод Suspend() у каждого потока:

```
private void StopThreadsButton_Click(object sender, EventArgs e)
    {
        thread1.Suspend();
       thread2.Suspend();
       thread3.Suspend();
     }
```

**Примечание.** Метод Suspend() – приостанавливает работу потока; если работа потока уже приостановлена, не оказывает влияния.

12. Также добавим самой форме обработчик событий, который бы вызывался после закрытия формы – FormClosed:

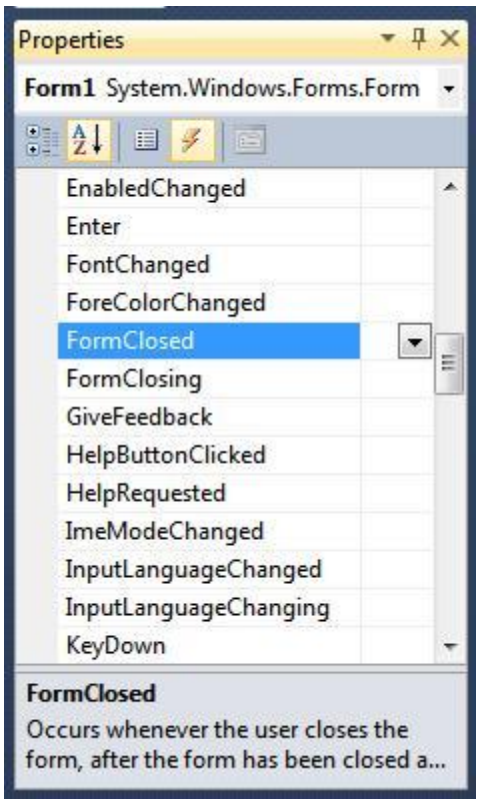


**Рис. 8.**

13. Добавим в событие Form1_FormClosed код, который бы завершал работу всех потоков при закрытии формы с помощью методаAbort():

```
private void Form1_FormClosed(object sender, FormClosedEventArgs e)
    {
       thread1.Abort();
       thread2.Abort();
       thread3.Abort();
    }
```

Примечание: Метод Abort() инициирует процесс завершения потока.

Листинг кода программы:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using System.Threading.Tasks;
namespace MultithreadingWinFormApplication
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
```

```
    thread1 = new Thread(new ThreadStart(DrawRectangle));
    thread2 = new Thread(new ThreadStart(DrawEllipse));
    thread3 = new Thread(new ThreadStart(RandomNumber));
}
Thread thread1;
Thread thread2;
Thread thread3;
private void DrawRectangle()
{
    try
    {
        Random rnd = new Random();
        Graphics g = panel1.CreateGraphics();
        while (true)
        {
            Thread.Sleep(4);
            g.DrawRectangle(Pens.Pink, 0, 0, rnd.Next(this.Width), rnd.Next(this.Height));
        }
    }
    catch (Exception ex) { }
}
private void DrawEllipse()
{
    try
    {
        Random rnd = new Random();
        Graphics g = panel2.CreateGraphics();
        while (true)
        {
            Thread.Sleep(4);
            g.DrawEllipse(Pens.Yellow, 0, 0, rnd.Next(this.Width), rnd.Next(this.Height));
        }
    }
    catch (Exception ex) { }
}
private void RandomNumber()
{
    try
    {
    Random rnd = new Random();

    Parallel.For(0, 500, i =>
          {   RandomNumberTextBox.Invoke((MethodInvoker)delegate()
            {
              RandomNumberTextBox.Text += rnd.Next().ToString();
            });
   });

    }
    catch (Exception ex) { MessageBox.Show(ex.Message); }
}
private void StartThreadingButton_Click(object sender, EventArgs e)
```

```
        {
            try
            {
                thread1.Start();
                thread2.Start();
                thread3.Start();
            }
            catch (Exception ex) { }
        }
        private void FirstMethodButton_Click(object sender, EventArgs e)
        {

                thread1.Start();

        }
        private void SecondMethodButton_Click(object sender, EventArgs e)
        {

            thread2.Start();

        }
        private void ThirdMethodButton_Click(object sender, EventArgs e)
        {

            thread3.Start();

        }
        private void StopThreadButton_Click(object sender, EventArgs e)
        {
            thread1.Suspend();
            thread2.Suspend();
            thread3.Suspend();

        }
        private void Form1_FormClosed(object sender, FormClosedEventArgs e)
        {
            thread1.Abort();
            thread2.Abort();
            thread3.Abort();
        }


    }
}
```