

# Лабораторная работа 7. Асинхронное программирование

## Цель работы

Изучение возможностей, реализующих асинхронное программирование и получение навыков по работе в программе с потоками.

### **Упражнение 1. Работа с компонентом *BackgroundWorker***

Класс **BackgroundWorker** позволяет выполнить операцию в отдельном, выделенном потоке. Операции, требующие много времени, такие как загрузка и транзакции базы данных, могут создавать впечатление, что пользовательский интерфейс перестал отвечать на действия пользователя. Если необходимо обеспечить быстрое реагирование пользовательского интерфейса, а подобные операции приводят к длительным задержкам, эффективным решением может стать класс **BackgroundWorker**.

Чтобы запустить занимающую много времени операцию в фоновом режиме, следует создать экземпляр **BackgroundWorker** и отслеживать события, сообщающие о ходе выполнения операции и сигнализирующие о ее завершении. Можно создать объект **BackgroundWorker** программными средствами или перетащить его в форму из вкладки Компоненты Панели элементов. Класс **BackgroundWorker**, созданный в конструкторе Windows Forms, появляется в области компонентов, а его свойства отображаются в окне "Свойства".

Выполнив это упражнение, вы научитесь применять компонент **BackgroundWorker**. Вам предстоит добавить к приложению компонент **BackgroundWorker** и написать отнимающий продолжительное время

метод, который будет выполняться в отдельном потоке. Затем вы сообщите о продвижении потока и реализуете функциональность отмены фонового процесса.

1. Создайте новый проект Windows Forms. Назовите его WinBackgroundWorker.

2. Добавьте на форму элементы управления: два элемента **Label**, **TextBox**, **ProgressBar** и две кнопки **Button**.

3. Для элементов укажите свойства в соответствии с таблицей:

Элемент	Свойство	Значение
label1	Location	10;15
	Text	Second to sleep
label2	Location	10; 40
	Text	Progress
textBox1	Location	105; 13
	Size	80; 20
progressBar	Location	110; 40
	Size	240; 20
button1	Location	195; 12
	Text	Start
	Size	75; 25
button2	Location	270; 12
	Text	Cancel
	Size	75; 25
Form1	Size	370;110

4. Для элемента **textBox1** допустимыми значениями будут только цифры, поэтому в обработчике события **KeyPress** для текстового поля **textBox1** укажите код:

```
if (!char.IsDigit(e.KeyChar))
{
    e.Handled = true;
    MessageBox.Show("Поле должно содержать цифры");
}
```

5. Из Toolbox перетащите элемент **BackgroundWorker** в форму.

6. В окне Properties установите свойства **WorkerSupportsCancellation** и **WorkerReportsProgress** в *True* (для поддержки асинхронной отмены и возможности сообщения основному потоку информации о продвижении фонового процесса соответственно).

7. Дважды щелкните **BackgroundWorker**, чтобы открыть обработчик события **backgroundWorker1\_DoWork** по умолчанию. Добавьте к этому обработчику события следующий код:

```
int i;
i = int.Parse(e.Argument.ToString());
for (int j=1; j <= i; j++)
{
    if (backgroundWorker1.CancellationPending)
```

```

    {
        e.Cancel = true;
        return;
    }
    System.Threading.Thread.Sleep(1000);
    backgroundWorker1.ReportProgress((int)(j *100/ i));
}

```

8. Для элемента **backgroundWorker1** в окне Properties щелкните кнопку Events, затем дважды щелкните *ProgressChanged*, чтобы открыть окно кода обработчика события *backgroundWorker1\_ProgressChanged*. Добавьте следующий код:

```
progressBar1.Value = e.ProgressPercentage;
```

9. Аналогичным способом добавьте обработчик события *RunWorkerCompleted*, в теле обработчика добавьте следующий код:

```

if (!(e.Cancelled))
    System.Windows.Forms.MessageBox.Show("Run Completed!");
else
    System.Windows.Forms.MessageBox.Show("Run Cancelled");

```

10. Для кнопки **Start** откройте обработчик события *Click*. Добавьте следующий код:

```

if (!(textBox1.Text == ""))
{
    int i = int.Parse(textBox1.Text);
    backgroundWorker1.RunWorkerAsync(i);
}

```

11. Для кнопки **Cancel** откройте обработчик события *Click*. Добавьте следующий код

```
backgroundWorker1.CancelAsync();
```

12. Постройте и выполните приложение, и проверьте его функциональность.

## **Упражнение 2. Использование делегатов**

Выполнив это упражнение, вы создадите приложение, подобное тому, что было создано в упражнении 1. Оно будет выполнять продолжительную операцию в отдельном потоке с возможностью отмены, показывать продвижение операции и уведомлять пользователя о ее завершении. Для реализации этой функциональности вы используете делегаты и асинхронный вызов.

1. Создайте новое Windows-приложение и назовите его *WinAsynchDelegate*.

2. Повторите шаги 2 – 4 предыдущего упражнения для создания требуемой формы.

3. Добавьте в приложение следующий метод:

```

private void TimeConsumingMethod(int seconds)
{
    for (int j = 1; j <= seconds; j++)
        System.Threading.Thread.Sleep(1000);
}

```

4. Используйте в приложении делегат, соответствующий методу *TimeConsumingMethod*:

```
    private delegate void TimeConsumingMethodDelegate(int  
seconds);
```

5. Добавьте сообщающий о продвижении операции метод, который устанавливает значение элемента управления *ProgressBar* потокобезопасным способом, и делегата этого метода:

```
    public delegate void SetProgressDelegate(int val);  
    public void SetProgress(int val)  
    {  
        if (progressBar1.InvokeRequired)  
        {  
            SetProgressDelegate del = new  
SetProgressDelegate(SetProgress);  
            this.Invoke(del, new object[] { val });  
        }  
        else  
        {  
            progressBar1.Value = val;  
        }  
    }
```

6. Для сообщения о продвижении операции добавьте следующую строку кода в цикл *For* метода *TimeConsumingMethod*:

```
    SetProgress((int)(j * 100) / seconds);
```

7. Добавьте в приложение логическую переменную с именем *Cancel*:

```
    bool Cancel;
```

8. Добавьте следующие строки кода в цикл *For* метода *TimeConsumingMethod*:

```
    if (Cancel)  
        break;
```

9. Добавьте следующий код после цикла *For* метода *TimeConsumingMethod*:

```
    if (Cancel)  
    {  
        System.Windows.Forms.MessageBox.Show("Cancelled");  
        Cancel = false;  
    }  
    else  
    {  
        System.Windows.Forms.MessageBox.Show("Complete");  
    }
```

10. В конструкторе дважды щелкните кнопку *GO!*, чтобы открыть для нее обработчик события *Click* по умолчанию, и добавьте следующий код:

```
    TimeConsumingMethodDelegate del = new  
TimeConsumingMethodDelegate(TimeConsumingMethod);  
    del.BeginInvoke(int.Parse(textBox1.Text), null, null);
```

11. В конструкторе дважды щелкните кнопку *Cancel*, чтобы открыть для нее обработчик события *Click* по умолчанию, и добавьте следующий код:

```
    Cancel = true;
```

12. Скомпилируйте и проверьте ваше приложение.

### **Упражнение 3. Асинхронный запуск произвольного метода**

При разработке программного обеспечения наиболее часто требуется запускать асинхронно собственные методы. В .NET Framework можно асинхронно вызывать любой метод. Для этого необходимо определить делегат с той же сигнатурой, что и у вызываемого метода.

Среда CLR автоматически определяет для этого делегата методы **BeginInvoke** и **EndInvoke** с соответствующими сигнатурами.

Для асинхронного запуска нужно проделать следующие шаги:

1. Создать и запустить делегат с необходимой сигнатурой. После этого можно работать со своим методом так же, как и с методами со встроенной поддержкой асинхронной модели программирования.

2. Выбрать механизм оповещения о завершении и подготовить для него все необходимое.

3. Запустить метод асинхронно.

4. Получить результаты в основном потоке и обновить пользовательский интерфейс.

Хотя компонент **BackgroundWorker** обеспечивает удобный способ выполнения простых задач в фоновом потоке, иногда может потребоваться осуществить более тонкий контроль за фоновыми процессами. В этом упражнении вы научитесь асинхронно выполнять методы с использованием делегатов.

1. Создайте новое Windows-приложение и назовите его **WinAsynchMethod**.

2. Установите свойствам формы **Size** значение **425;200** и **Text** – "Асинхронный запуск".

3. Добавьте на форму три надписи, два текстовых поля и две кнопки, и установите им следующие свойства:

Свойство	Значение
<b>button1</b>	
Name	btnRun
Location	16; 64
Text	Сумма
<b>button2</b>	
Name	btnWork
Location	120; 128
Text	Работа
<b>label1</b>	
Name	lblA
Location	8; 24
Text	Значение А
<b>label2</b>	
Name	lblB
Location	216; 24
Text	Значение В

Свойство	Значение
<b>textBox1</b>	
Name	txbA
Location	88; 24
Text	
<b>textBox2</b>	
Name	txbB
Location	296; 24
Text	

4. Создайте делегат:

```
private delegate int AsyncSumm(int a, int b);
```

5. Создайте метод Summ, в котором будут складываться числа, вводимые в два текстовых поля, и укажите задержку операции на 9 секунд:

```
private int Summ(int a, int b)
{
    System.Threading.Thread.Sleep(9000);
    return a+b;
}
```

6. Реализуйте обработчик кнопки **btnRun**, который будет включать также действия по организации асинхронного вызова:

Создайте экземпляр делегата и проинициализируйте его методом Summ:

```
AsyncSumm summdelegate = new AsyncSumm(Summ);
```

Для использования механизма **Callback** создайте экземпляр делегата AsyncCallBack:

```
AsyncCallback cb = new AsyncCallback(CallBackMethod);
```

После того как делегат инициализирован методом, можно запускать прикрепленный к делегату метод асинхронно с помощью метода **BeginInvoke**. Этот метод принимает две переменные типа int a и b, экземпляр cb делегата AsyncCallback и экземпляр summdelegate делегата SummDelegate:

```
summdelegate.BeginInvoke(a, b, cb, summdelegate);
```

7. В итоге обработчик кнопки **btnRun** будет выглядеть следующим образом:

```
private void btnRun_Click(object sender, System.EventArgs e)
{
    int a, b;
    try
    {
        // Преобразование типов данных.
        a = Int32.Parse(txbA.Text);
        b = Int32.Parse(txbB.Text);
    }
    catch(Exception)
    {
        MessageBox.Show("При выполнении преобразования типов
возникла ошибка");
        txbA.Text = txbB.Text = "";
        return;
    }
}
```

```
        AsyncSumm summdelegate = new AsyncSumm(Summ);
        AsyncCallback cb = new
AsyncCallback(CallBackMethod);
        summdelegate.BeginInvoke(a, b, cb, summdelegate);
    }
```

8. Создайте метод CallBackMethod, который привязан к делегату summdelegate:

```
private void CallBackMethod(IAsyncResult ar)
{
    string str;
    AsyncSumm summdelegate = (AsyncSumm)ar.AsyncState;
    str = String.Format("Сумма введенных чисел равна
{0}", summdelegate.EndInvoke(ar));
    MessageBox.Show(str, "Результат операции");
}
```

9. Для демонстрации асинхронности выполнения метода реализуйте обработчик нажатия кнопки **Работа**, например, следующим образом:

```
    MessageBox.Show("Работа кипит!!!!");
```

10. Постройте и запустите приложение. После нажатия кнопки **Сумма**, пока будет выполняться операция, нажмите кнопку **Работа**. Проверьте, что метод суммирования действительно реализован асинхронно.