Лабораторна робота 7. Асинхронне програмування

Вправа 1. Робота з компонентом BackgroundWorker (5 балів)
Вправа 2. Використання делегатів (5 балів)
Вправа 3. Асинхронний запуск довільного методу (5 балів)

Мета роботи: Вивчення можливостей, які реалізовують асинхронне програмування та отримання навичок щодо роботи в програмі з потоками.

Вправа 1. Робота з компонентом BackgroundWorker (5 балів)

Паралелізм це техніка для досягнення асинхронності, але асинхронність це не завжди паралелізм. ***Асинхронна ситуація*** це коли наявна така затримка між запитом і результатом запиту, що ви можете продовжити роботу під час очікування. ***Паралелізм*** це підхід для досягнення асинхронності за допомогою найманих працівників – потоків – кожен з яких виконує свої задачі синхронно, але всі вони паралельні.

Тут може допомогти аналогія. Уявімо, що ви на кухні ресторану. Надійшли два замовлення, одне на тост і одне на яєшню. ***Синхронна послідовність була б такою:*** вставити хліб у тостер, зачекати доки тостер не поверне готовий тост, віддати тост, розбити яйця на пательню, зачекати до готовності, віддати яєшню.

Асинхронним варіантом було б вставити хліб у тостер, поки тостер працює розбити яйця на пательню. Далі потрібно почергово перевіряти готовність тоста, яєшні і слідкувати за надходженням нових замовлень, виконання яких можна одразу ж розпочати. Перше готове замовлення віддається першим.

Асинхронним паралельним варіантом було б таке, просто сидіти і чекати на замовлення. Щоразу як надходить замовлення іти до холодильника де ви тримаєте поварів, розморожувати одного з них і надавати йому це замовлення. Отже ви маєте повара для яєшні, повара для тоста і поки вони готують ви очікуєте на наступні замовлення. Щойно повар завершить свою роботу, ви віддаєте замовлення і кладете повара назад у холодильник.

Можна зауважити, що справжні ресторани обирають другий механізм т. т. асинхронним варіантом, бо він поєднує менші витрати на робочу силу – повари дорогі – з реагованістю і пропускною здатністю. Перший підхід має погану реагованість і пропускну здатність. А третій метод вимагає оплати багатьох поварів, які просто сидять і чекають.

Клас **BackgroundWorker** дозволяє виконати операцію в окремому, виділеному потоці. Операції, які вимагають багато часу, такі як завантаження і транзакції бази даних, можуть створювати враження, що інтерфейс користувача перестав відповідати на дії користувача. Якщо потрібно забезпечити швидке реагування інтерфейсу користувача, а подібні операції приводять до тривалих затримок, ефективним рішенням може стати клас **BackgroundWorker**.

Щоб запустити операцію, яка займає багато часу у фоновому режимі, треба створити екземпляр **BackgroundWorker** і відстежувати події, які повідомляють про хід виконання операції і сигналізують про її завершення. Можна створити об'єкт **BackgroundWorker** програмними засобами або перетягнути його у форму з вкладки **Компоненты панели элементов**. Клас **BackgroundWorker**, створений в конструкторі Windows Forms, з'являється в області компонентів, а його властивості відображаються у вікні **Свойства**.

Виконавши цю вправу, ви навчитеся використовувати компонент **BackgroundWorker**. Вам потрібно додати до застосування компонент **BackgroundWorker** і написати метод, який виконуватиметься в окремому потоці і займатиме тривалий час. Потім ви повідомите про просування потоку і реалізуєте функціональність відміни фонового процесу.

1. Створіть новий проект Windows Forms. Назвіть його **WinBackgroundWorker**.
2. Додайте на форму елементи управління: два елементи **Label, TextBox, ProgressBar** і дві кнопки **Button**.
3. Для елементів вкажіть властивості відповідно до таблиці:

Елемент	**Властивість**	**Значення**
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	217; 100
	Text	Start
	Size	75; 25
button2	Location	217; 140
	Text	Cancel
	Size	75; 25
Form1	Size	320; 220

4. Для елемента **textBox1** допустимими значеннями будуть тільки цифри, тому в обробнику події **KeyPress** для текстового поля **textBox1** вкажіть код:

```
if (!char.IsDigit(e.KeyChar))
{
        e.Handled = true;
        MessageBox.Show("Поле повинне містити цифри");
}
```

5. Із **Toolbox** перетягніть елемент **BackgroundWorker** у форму.
6. У вікні **Properties** встановіть властивості **WorkerSupportsCancellation** і **WorkerReportsProgress** в **True** (для підтримки асинхронної відміни і можливості повідомлення основному потоку інформації про просування фонового процесу відповідно).
7. Двічі клацніть **BackgroundWorker**, щоб відкрити обробник події **backgroundWorker1_DoWork** за замовчуванням. Додайте до цього обробника події наступний 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 елемента **backgroundWorker1** у вікні **Properties** клацніть кнопку **Events**, потім двічі клацніть **ProgressChanged**, щоб відкрити вікно коду обробника події **backgroundWorker1_ProgressChanged**. Додайте наступний код:

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

9. Аналогічним чином додайте обробник події **RunWorkerCompleted**, в тілі обробника додайте наступний код:

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

10. Для кнопки **Start** відкрийте обробник події **Click**. Додайте наступний код:

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

11. Для кнопки **Cancel** відкрийте обробник події **Click**. Додайте наступний код:

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

12. Побудуєте і виконайте застосування, і перевірте його функціональність.

Вправа 2. Використання делегатів (5 балів)

Виконавши цю вправу, ви створите застосування, подібне до того, що було створене у **Вправі 1**. Воно виконуватиме тривалу операцію в окремому потоці з можливістю відміни, показу просування операції і повідомляти користувача про її завершення. Для реалізації цієї функціональності ви використаєте ***делегати*** та ***асинхронний виклик***.

1. Створіть нове Windows-застосування і назвіть його **WinAsynchDelegate**.
2. Додайте на форму елементи управління: два елементи **Label, TextBox, ProgressBar** і дві кнопки **Button**.
3. Для елементів вкажіть властивості відповідно до таблиці:

Елемент	**Властивість**	**Значення**
label1	Location	10; 15
	Text	Second to sleep
label2	Location	48; 40
	Text	Progress
textBox1	Location	105; 13

	Size	162; 23
progressBar	Location	110; 40
	Size	162; 23
button1	Location	197; 87
	Text	Start
	Size	75; 25
button2	Location	197; 125
	Text	Cancel
	Size	75; 25
Form1	Size	320; 220

4. Для елемента **textBox1** допустимими значеннями будуть тільки цифри, тому в обробнику події **KeyPress** для текстового поля **textBox1** вкажіть код:

```
if (!char.IsDigit(e.KeyChar))
{
        e.Handled = true;
        MessageBox.Show("Поле повинне містити цифри");
}
```

5. Додайте в застосування наступний метод:

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

6. Використайте в застосуванні делегат, що відповідає методу **TimeConsumingMethod**:

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

7. Додайте метод, що повідомляє про просування операції, який встановлює значення елемента управління **ProgressBar** потокобезпечним способом, і делегата цього методу:

```
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;
        }
}
```

8. Для повідомлення про просування операції додайте наступний рядок коду в цикл **For** методу **TimeConsumingMethod**:

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

9. Додайте в застосування логічну змінну з ім'ям **Cancel:**

```
bool Cancel;
```

10. Додайте наступні рядки коду в цикл **For** методу **TimeConsumingMethod:**

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

11. Додайте наступний код після циклу **For** методу **TimeConsumingMethod:**

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

12. У конструкторі двічі клацніть кнопку **Start**, щоб відкрити для неї обробник події **Click** за замовчуванням, і додайте наступний код:

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

13. У конструкторі двічі клацніть кнопку **Cancel**, щоб відкрити для неї обробник події **Click** за замовчуванням, і додайте наступний код:

```
Cancel = true;
```

14. Скомпілюйте і перевірте ваше застосування.

Вправа 3. Асинхронний запуск довільного методу (5 балів)

При розробці програмного забезпечення найчастіше вимагається запускати асинхронно власні методи. У .NET Framework можна асинхронно викликати будь-який метод. Для цього потрібно визначити ***делегат з тією ж сигнатурою***, що і у методу, який викликається.

Середовище CLR автоматично визначає для цього делегата методи **BeginInvoke** та **EndInvoke** з відповідними сигнатурами.

Для асинхронного запуску треба виконати наступні кроки:

1. Створити і запустити делегат з потрібною сигнатурою. Після цього можна працювати зі своїм методом так само, як і з методами з вбудованою підтримкою асинхронної моделі програмування.
2. Вибрати механізм сповіщення про завершення і підготувати для нього все потрібне.
3. Запустити метод асинхронно.
4. Отримати результати в основному потоці і відновити інтерфейс користувача.

Хоча компонент **BackgroundWorker** забезпечує зручний спосіб виконання простих завдань у фоновому потоці, іноді може знадобитися здійснити тонший контроль за фоновими процесами. У цій вправі ви навчитеся асинхронно виконувати методи використанням делегатів.

1. Створіть нове Windows-застосування і назвіть його **WinAsynchMethod**.
2. Встановіть властивостям форми **Size** значення **425; 200** і **Text - "Асинхронний запуск"**.
3. Додайте на форму **два** написи, **два** текстові поля і **дві** кнопки, і встановіть їм наступні властивості:

Властивість	**Значення**
button1	
Name	btnRun
Location	11; 64
Text	Сума
button2	
Name	btnWork
Location	219; 64
Text	Робота
label1	
Name	lblA
Location	8; 24
Text	Значення A
label2	
Name	lblB
Location	216; 24
Text	Значення B
textBox1	
Name	txbA
Location	88; 24
Text	
textBox2	
Name	txbB
Location	296; 24
Text	

4. Створіть делегат:

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

5. Створіть метод **Summ**, в якому додаватимуть числа, які вводяться в два текстові поля, і вкажіть затримку операції на **9** секунд:

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

6. Реалізуйте обробник кнопки **btnRun**, який включатиме також дії з організації асинхронного виклику:
   Створіть екземпляр делегата і проініціалізуйте його методом **Summ**:

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

   Для використання механізму **Callback** створіть екземпляр делегата **AsyncCallBack**:

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

   Після того, як делегат ініціалізований методом, можна запускати прикріплений до делегата метод асинхронно за допомогою методу **BeginInvoke**. Цей метод приймає дві змінні типу **int a** і **b**, екземпляр **cb** делегата **AsyncCallback** та екземпляр **summdelegate** делегата **SummDelegate**:

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

7. У результаті обробник кнопки **btnRun** виглядатиме наступним чином:

```
private void btnRun_Click(object sender, System.EventArgs e)
{
        int a, b;
        try
        {
                // Перетворення типів даних.
                a = Int32.Parse(txbA.Text);
                b = Int32.Parse(txbB.Text);
        }
        catch(Exception)
        {
                MessageBox.Show("При виконанні перетворення типів виникла помилка");
                txbA.Text = txbB.Text = "";
                return;
        }
        AsyncSumm summdelegate = new AsyncSumm(Summ);
        AsyncCallback cb = new AsyncCallback(CallBackMethod);
        summdelegate.BeginInvoke(a, b, cb, summdelegate);
}
```

8. Створіть метод **CallBackMethod**, який прив'язаний до делегата **summdelegate**:

```
private void CallBackMethod(IAsyncResult ar)
{
        string str;
        AsyncSumm summdelegate = (AsyncSumm) ar.AsyncState;
        str = String.Format("Сума введених чисел дорівнює {0}", summdelegate.EndInvoke(ar));
        MessageBox.Show(str, "Результат операції");
}
```

9. Для демонстрації асинхронності виконання методу реалізуйте обробник натиснення кнопки **Робота**, наприклад, таким чином:

```
MessageBox.Show("Робота кипить!!!");
```

10. Побудуйте і запустіть застосування. Після натиснення кнопки **Сума**, поки виконуватиметься операція, натисніть кнопку **Робота**. Перевірте, що метод знаходження суми дійсно реалізований асинхронно.