ПРАКТИЧНЕ ЗАНЯТТЯ №1

ТЕМА: «Технології виміру часу та їх практичне використання»

Звіт про практичне заняття-файл у форматі .doc або .docx. У файлі повинна бути вказана прізвище, група, номер пз, кожному завданню з пз повинні відповідати постановка завдання, скріншоти, що віддзеркалюють **всі етапи та результати** виконання завдання, після кожного завдання наводяться висновки.

Завдання 1: За методичними вказівками для виконання практичних занять (заняття 1 стр 6) виконати завдання 3,6.

Завдання 2:

У Visual Studio 2017 створіть новий проект Visual C # і виберіть пункт Console App (.NET Core) (Консольний додаток (.NET Core)), надайте йому ім'я WorkingWithTasks, вкажіть розташування, введіть ім'я рішення Chapter1, а потім натисніть кнопку OK.

У Visual Studio Code створіть папку Chapter1 з підкаталогом WorkingWithTasks, а потім відкрийте цей підкаталог. На панелі Integrated Terminal (Інтегрований термінал) виконайте команду dotnet new console. У Visual Studio 2017 і Visual Studio Code переконайтеся, що в коді імпортуються такі простори імен і статичні типи: using System; using System.Threading; using System.Threading.Tasks; using System.Diagnostics; using static System.Console; Як уже згадувалося, в програмі будуть використані три методи, які необхідно виконати: перший займає три секунди, другий - дві, а третій - одну. Для імітації такого процесу можна застосувати клас Thread, що дозволяє поточному потоку засинати на задане число мілісекунд.

Додайте в клас Program код, наведений нижче:

```
static void MethodA() {

WriteLine("Starting Method A...");
Thread.Sleep(3000); // імітація роботи протягом трьох секунд
 WriteLine("Finished Method A.");
}
static void MethodB() {
```

```
WriteLine("Starting Method B...");
Thread.Sleep(2000); // імітація роботи протягом двох  секунд
WriteLine("Finished Method B.");
}
static void MethodC() {

WriteLine("Starting Method C...");
Thread.Sleep(1000); // імітація роботи протягом однієї секунди
WriteLine("Finished Method C.");
}
```

Додайте в метод Main наступні інструкції:

```
static void Main(string[] args)
{
var timer = Stopwatch.StartNew();
WriteLine("Running methods synchronously on one thread.");
MethodA();
MethodB();
MethodC();
WriteLine($"{timer.ElapsedMilliseconds:#,##0}ms elapsed.");
WriteLine("Press ENTER to end.");
ReadLine();
}
```

Запустіть консольний додаток, проаналізуйте результат виведення і зверніть увагу ось на те, що так як використовується тільки один потік, виконання програми займе трохи більше шести секунд:

У версії C # 4 був доданий клас Task, що представляє собою оболонку для потоку, яка дозволила спростити створення і управління потоками. Обгортання декількох потоків в завдання дозволить коду виконуватися асинхронно. Клас Task має властивості Status і CreationOptions, а також метод ContinueWith, який можна налаштувати, вдавшись до перерахування TaskContinuationOptions, і управляти за допомогою класу TaskFactory. Розглянемо три способи запуску методів з використанням примірників класу Task. Кожен з них дещо відрізняється синтаксисом, але всі вони визначають цей клас і запускають його. Закоментуйте виклики трьох методів і пов'язане з ними консольне повідомлення, а потім додайте нові інструкції, як показано в наступному лістингу:

```
static void Main(string[] args) {
```

```
var timer = Stopwatch.StartNew();
//WriteLine("Running methods synchronously on one thread.");
//MethodA();
//MethodB();
//MethodC();
WriteLine("Running methods asynchronously on multiple threads.");
Task taskA = new Task(MethodA);
taskA.Start();
Task taskB = Task.Factory.StartNew(MethodB);
Task taskC = Task.Run(new Action(MethodC));
WriteLine($"{timer.ElapsedMilliseconds:#,##0}ms elapsed.");
WriteLine("Press ENTER to end.");
ReadLine();
}
```

Запустіть консольний додаток і проаналізуйте результат виведення. Зверніть увагу: кількість мілісекунд, фактично витрачена на роботу програми, залежить від продуктивності процесора в вашому комп'ютері.

Іноді потрібно дочекатися завершення завдання, перш ніж продовжувати роботу. Для цього використовується метод Wait екземпляра класу Task або статичні методи WaitAll і WaitAny для масиву завдань.

Метод	Опис
t.Wait()	Чекає завершення виконання завдання t
Task.WaitAny(Task[])	Чекає завершення виконання будь-якого завдання в масиві
Task.WaitAll(Task[])	Чекає завершення виконання всіх завдань у масиві

Додайте наступні інструкції в метод Main відразу після коду створення трьох завдань і перед виведенням витраченого часу. Так ви об'єднаєте в масив посилання на три завдання і передасте його методу WaitAll. Тепер вихідний потік буде зупинятися на цій інструкції, чекаючи завершення всіх трьох завдань, перш ніж виведе значення витраченого часу.

```
Task[] tasks = { taskA, taskB, taskC };
Task.WaitAll(tasks);
```

Перезапустіть консольний додаток і проаналізуйте результат виведення: