Лабораторная работа №9

«Многопоточность, TPL, асинхронное программирование»

**Цели работы:**

1. Научиться работать со средствами многопоточности языка C#.
2. Научиться работать с паттерном асинхронного программирования платформы .Net языка C#.

**Задание№1**

Создайте многопоточное приложение для получения средних цен акций за год.

Используйте сайт <https://finance.yahoo.com/> для получения дневных котировок списка акций из файла ticker.txt. Формат ссылки следующий:

[https://query1.finance.yahoo.com/v7/finance/download/{Код\_бумаги}?period1={Начальная\_дата}&period2={Конечная\_дата}&interval=1d&events=history&includeAdjustedClose=true](https://query1.finance.yahoo.com/v7/finance/download/%7b%D0%9A%D0%BE%D0%B4_%D0%B1%D1%83%D0%BC%D0%B0%D0%B3%D0%B8%7d?period1=%7b%D0%9D%D0%B0%D1%87%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%B4%D0%B0%D1%82%D0%B0%7d&period2=%7b%D0%9A%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D0%B0%D1%8F_%D0%B4%D0%B0%D1%82%D0%B0%7d&interval=1d&events=history&includeAdjustedClose=true)

,где:

Код\_бумаги – тикер из списка акций

Начальная\_дата – метка времени начала запрашиваемого периода в UNIX формате (год назад).

Конечная\_дата – метка времени конца запрашиваемого периода в UNIX формате (текущая дата).

Например, формат ссылки для AAPL:

<https://query1.finance.yahoo.com/v7/finance/download/AAPL?period1=1629072000&period2=1660608000&interval=1d&events=history&includeAdjustedClose=true>

# По мере получения данных выполните запуск задачи(Task), которая будет считать среднюю цену акции за год (используйте среднее значение для каждого дня как (High+Low)/2. Сложите все полученные значения и поделите на число дней).

# Результатом работы задачи будет являться среднее значение цены за год, которое необходимо вывести в файл в формате «Тикер:Цена». При этом обеспечьте потокобезопасный доступ к файлу между всеми задачами.

**Задание№2**

# На основе Лабораторной работы №6 создайте графическое приложение, получающее текущую погоду в разных городах мира.

# Используйте WinForms или WPF.

# Загрузите список городов с координатами из файла city.txt.

# Добавьте в интерфейс приложения 2 элемента – один для отображения списка городов, второй элемент – кнопку, по нажатию на которую происходит загрузка текущей погоды в выбранном по API из Лабораторной работы №6.

# Используйте ключевые слова async/await для загрузки данных и обновления интерфейса приложения таким образом, чтобы графический интерфейс не блокировался на время загрузки.

# Если разработка ведется на \*NIX или MACOS, допускается использовать консольное приложение, тем не менее следует использовать ключевые слова async/await при реализации методов.

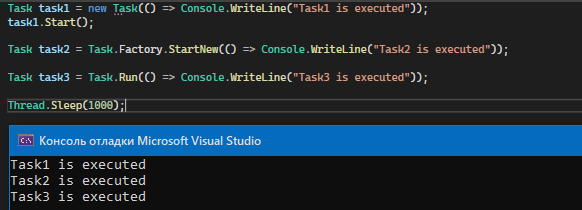
**Теоретические сведения**

**TPL- Task Parallel Library**

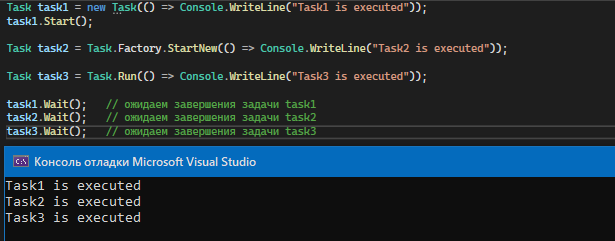
# TPL (Task Parallel Library) в пространстве имен System.Threading.Tasks упрощает работу с многопроцессорными, многоядерными системами. Кроме того, она упрощает работу по созданию новых потоков. Поэтому обычно рекомендуется использовать именно TPL и ее классы для создания многопоточных приложений, хотя стандартные средства и класс Thread по-прежнему находят широкое применение.

# В основе библиотеки TPL лежит концепция задач, каждая из которых описывает отдельную продолжительную операцию. В библиотеке классов .NET задача представлена специальным классом - классом Task, который находится в пространстве имен System.Threading.Tasks. Данный класс описывает отдельную задачу, которая запускается асинхронно в одном из потоков из пула потоков. Хотя ее также можно запускать синхронно в текущем потоке. Однако в любом случае следует отметить, что задача - это не поток.

Способы создания и запуска задач:



Чтобы приложение ожидало завершения задачи, можно использовать метод Wait() объекта Task:



Консольный вывод не детерминирован, поскольку задачи не выполняются последовательно. Первая запущенная задача может завершить свое выполнение после последней задачи.

### Свойства класса Task

Класс Task имеет ряд свойств, с помощью которых мы можем получить информацию об объекте. Некоторые из них:

* AsyncState: возвращает объект состояния задачи
* CurrentId: возвращает идентификатор текущей задачи (статическое свойство)
* Id: возвращает идентификатор текущей задачи
* Exception: возвращает объект исключения, возникшего при выполнении задачи
* Status: возвращает статус задачи. Представляет перечисление System.Threading.Tasks.TaskStatus, которое имеет следующие значения:
  + Canceled: задача отменена
  + Created: задача создана, но еще не запущена
  + Faulted: в процессе работы задачи произошло исключение
  + RanToCompletion: задача успешно завершена
  + Running: задача запущена, но еще не завершена
  + WaitingForActivation: задача ожидает активации и постановки в график выполнения
  + WaitingForChildrenToComplete: задача завершена и теперь ожидает завершения прикрепленных к ней дочерних задач
  + WaitingToRun: задача поставлена в график выполнения, но еще не начала свое выполнение
* IsCompleted: возвращает true, если задача завершена
* IsCanceled: возвращает true, если задача была отменена
* IsFaulted: возвращает true, если задача завершилась при возникновении исключения
* IsCompletedSuccessfully: возвращает true, если задача завершилась успешно

## Асинхронные методы, async и await

Асинхронность позволяет вынести отдельные задачи из основного потока в специальные асинхронные методы и при этом более экономно использовать потоки. Асинхронные методы выполняются в отдельных потоках. Однако при выполнении продолжительной операции поток асинхронного метода возвратится в пул потоков и будет использоваться для других задач. А когда продолжительная операция завершит свое выполнение, для асинхронного метода опять выделяется поток из пула потоков, и асинхронный метод продолжает свою работу.

Ключевыми для работы с асинхронными вызовами в C# являются два оператора: async и await, цель которых - упростить написание асинхронного кода. Они используются вместе для создания асинхронного метода.

Асинхронный метод обладает следующими признаками:

* В заголовке метода используется модификатор async
* Метод содержит одно или несколько выражений await
* В качестве возвращаемого типа используется один из следующих:
  + void
  + Task
  + Task<T>
  + ValueTask<T>

Асинхронный метод, как и обычный, может использовать любое количество параметров или не использовать их вообще. Однако асинхронный метод не может определять параметры с модификаторами out, ref и in.

Также стоит отметить, что слово async, которое указывается в определении метода, НЕ делает автоматически метод асинхронным. Оно лишь указывает, что данный метод может содержать одно или несколько выражений await.

