**ЛЕКЦИЯ 1**

**Что такое веб приложение?**

Веб-приложение - клиент-серверное приложение, в котором клиент взаимодействует с веб-сервером при помощи браузера. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети

Web-приложения:

• Традиционные сайты: новости, блоги, wiki, визитки

• Глобальные приложения: почтовые сервисы, поиск, социальные сети

• E-commerce: магазины,бронирования, цифровая дистрибуция

• Замена десктоп-приложений: корпоративный софт, CRM, банковские клиенты

**Как работают веб приложения?**

Через браузер

Пользователь отправляет запрос(request). Сервер обрабатывает и отправляет ответ(response)

**Кто отправляет данные пользователю?**

Сервер.

**Как клиент/сервер знают куда посылать запрос/ответ?**

С помощью протоколов(http, в нашем случае).

Основным объектом манипуляции в HTTP является ресурс, на который указывает URL в запросе клиента.

**• URL (Uniform Resource Locator)** – унифицированный указатель на ресурс.

**• URI (Uniform Resource Identifier)** – унифицированный (единообразный) идентификатор ресурса.

• **URN (Uniform Resource Name)** – единообразное название (имя) ресурса.

**Что они отправляют друг другу?**

Протоколы.

**Как сервер понимает какие данные генерировать для клиента?**

Благодаря методам.

Методы, которые определяют необходимое действие (например, получить файл, сохранить или обновить какие-либо данные). Различные методы/команды(пример ниже)

**GET** – получить определённый ресурс. со стороны сервера на клиента

Самый используемый запрос

– Ввод в адресную строку

– Нажатие на ссылку

• Ничего не меняет на сервере (идемпотентный)

• Может начать какой-либо процесс. В этом случае в ответе надо дать информацию о ходе процесса

GET /path/resource?param1=value1&param2=value2 HTTP/1.1

**POST** – создать новый ресурс. клиент отправляет post запросы

Метод для изменения чего-то на сервере, обычно добавления

– Требующего предварительной обработки

• Часто используется для отправки данных с веб-форм или загружаются файлы на сервер

• Не является идемпотентным

• Хранит параметры в теле запроса

– Поэтому пароли и другую конфиденциальную информацию посылаем внутри POST-запросов

**Как сервер по запросу клиента определяет что нужно генерировать?**

Благодаря Ajax.

Кроме Ajax учитываются любые действия пользователя, GET,POST запросы в основном.

**Как обновлять содержимое страницы?**

Динамическая генерация кода.

Содержимое ответа генерируется сервером при обработке запроса.

**ЛЕКЦИЯ 2**

**Как обрабатываются запросы и отправляются ответы?**

После установки соединения браузер отправляет специальный запрос, в котором просит сервер отправить данные для отображения страницы. В этом запросе содержится информация о самом браузере, временные файлы, требования к соединению и так далее. В общении браузера и сервера выделяют два типа запросов. GET-запрос используется для получения данных с сервера — например, отобразить картинку, текст или видео. POST-запрос — используется для отправки данных из браузера на сервер, например, когда пользователь отправляет сообщение, картинку и т.д.

Сервер обрабатывает запрос с помощью веб-серверов(Apache,например) - веб-сервер отправляет на дальнейшую обработку к программе обработчику(ASP.NET, к примеру). Прога собирает ответ и отправляет его браузеру. Как правило, содержит контент для отображения веб-страниц. Сервер обрабатывает и рисует страницу.

**В чѐм разница между приложением и процессом?**

Процесс является чем-то исполняем. В приложении же может быть запущено несколько процессов.

**Что вернѐт http-запрос, если сервер не**

**работает?**

Ошибки с 500 до 599.

**Порт?**

На сервере могут быть несколько приложений, которые занимаются обработкой запросов клиента. Но как понять какое именно приложение должно получить запрос?

***Порт*** - целое неотрицательное число, записываемое в заголовках протоколов транспортного уровня сетевой модели OSI (TCP, UDP, SCTP, DCCP) или точка подключения к хосту.

К каждому ***порту*** на сервере можно привязывать своё приложение

Разделением ***портов*** занимается ОС

Диапазон принимаемых портов: 0-65536(216)

Один хост – много портов

Записываются в заголовках протокола транспортного уровня OSI (TCP, UDP, ...)

http://10.12.13.14:1234/

• Определяют процесс-получатель пакета

Открытый порт – доступный извне порт, прослушиваемый процессом

Существуют порты по умолчанию – HTTP: 80

– HTTPS: 443 – SSH: 22

– FTP: 21

Но! Внутри сервера запрос может переадресовываться другому приложению на другом порту

• Порты одного протокола не пересекаются с портами другого (TCP и UDP, например)

• Во многих ОС прослушивание портов 0- 1023 требует особых привилегий

**Что такое маршрутизация URL?**

Это сопоставление пути соответствующим обработчиком

**Как маршрутизировать запросы?**

• Парсить запрос, исполнять подходящий код

– Как сделать без хардкода?

• Установить соответствие классам обработчикам по шаблонам

– Можно рефлексией

**Что такое CGI?**

Общий интерфейс шлюза(CGI) – стандарт интерфейса, используемого для связи внешней программы с веб-сервером

**Отличие веб-сервера от сервера приложений?**

Основное отличие веб-сервера от сервера приложений заключается в том, что веб-сервер предназначен для обслуживания статических страниц, например HTML и CSS, тогда как сервер приложений отвечает за генерацию динамического содержимого путём выполнения кода на стороне сервера

**Что такое HTTPListener?**

Класс из библиотеки System.Net для прослушивания подключений по протоколу HTTP.

**Как работать с HTTPListener?**

Есть свойство Prefixes, через которое нужно установить адреса URI с помощью метода Add(), которые он будет прослушивать.

var listener = new HttpListener();

listener.Prefixes.Add(“http://localhost:8080/”);

listener.Start()

listener.Stop()

**ЛЕКЦИЯ 3**

У одного приложения – несколько обработчиков (handler), возможно исполняемых компонент.

Каждая компонента может обрабатывать несколько запросов.

у одного приложения - несколько клиентов.

данные можно сохранять в ассоциативную коллекцию(словарь) ключ-значение.

Сессии

ряд последовательных запросов, в течение некоторого времени.

С помощью сессии можно проверить, был ли авторизован пользователь

• Можно в словаре сессии проверить наличие значения по ключу (например, currentUser)

– В зависимости от результата направить к обработке или к форме авторизации

Идентификация – представление пользователя, сообщение о том, кто он такой

• Аутентификация – проверка доказательств (факторов) подтверждения личности – Идентификация – часть аутентификации

• Авторизация – проверка разрешений действий аутентифицированной персоне.

• Redirect – автоматический переход на другую страницу, инициируемый сервером.

Как сервер вычисляет пользователя?  
-Куки файлы! (ассоциативная коллекция, сервер проверяет наличие пользовательских куки по ключу и авторизует пользователя, при обнаружении)

хранятся на клиенте.

у куки есть время жизни, имя-значение, область действия(в какой части доменного имени и path применяется)

Почему Cookie хранятся на компьютере пользователя ?  
Любая другая реализация была бы бессмысленна, потому что хранение куков по IP, к примеру, привело бы к следующим проблемам:  
1) Изменилась сеть - изменилось IP, и реализация Cookie теряет смысл  
2) Ваш IP адрес может уйти другому пользователю и тогда он сможет неумышленно попасть в ваш профиль.

Обработчик(Handler)

генератор содержимого на запрос HTTP-request можно наследовать интерфейс httpHandler, но проще использовать шаблонизаторы.

Шаблонизаторы

- Использование html-шаблонов с вкраплениями кода для генерации конечных html-страниц

разделение представления от исходного кода.

Шаблон содержит

статические элементы ( html, css, js)

-динамические элементы, генерирующие содержимое

пример

Razor pages

модель страницы

.cs файт в котором прописана логика взаимодействия страницы с серверной частью.

мастер страницы /shared

**ЛЕКЦИЯ 4**

GUI-набор окон, содержащий изменяющиеся элементы, в зависимости от взаимодействия с объектом меняются данные приложения.

каждый элемент- это объект, его можно описать как класс. пользователь может инициировать события.

Что из этого пишем сами?

-обработчик события - обработчики подписываем на события.

что используем?

-библиотеки GUI

**Преимущества web-приложений**

• Не нужна установка – только браузер и интернет

• Легко поддерживать – Обновил на сервере и готово

• Кросс-платформенность – Нужна кроссбраузерность, но еѐ обеспечить проще

код исполняемый в браузере это Javascript ≈ ECMAScript + DOM + BOM

<script type="application/javascript" src="http://Путь\_к\_файлу\_со\_скриптом"> </script> -получение скрипта из файла.

**DOM и BOM**

• DOM –   
Document Object Model – Позволяет получать доступ к структуре HTML (в виде дерева) и менять его содержимое (DHTML) – 1998 DOM 1, 2000 DOM 2, 2004 DOM 3 – HTML-элементы – объекты со свойствами. DOM даѐт JS возможность изменять эти свойства

• BOM – Browser Object Model – Надмножество DOM – Плохо стандартизировано

**JS синтаксис:**

* Родственный C, Java
* Есть:

–Динамическая, утиная типизация

–Объектно-ориентированный подход (прототипирование)

–Функции-объекты первого рода

–Анонимные функции

–Замыкания

–eval

–…

**Утиная типизация**

* тест на утку (если что-то ходит, как утка, и крякает, как утка, то это утка)
* Объект реализует интерфейс, если реализует все его методы, независимо от связей в иерархии

**Объектно-ориентирвоанный**

* объектно, а не классо-ориентированный
* в JS нет классов
* один объект может быть прототипом другого

**Конструктор**

* создавать прототипы можно через конструктор

**Функции-это объекты**

var f = function () {

alert("I am F!");

}

var g = f; //вызова функции еще не было

g();//вот вызов

* JS позволяет изменять содержимое HTML

**Отличия JS от C#**

1. **C# компилируемый язык, а Javascript - интерпретируемый.**
2. **C# это язык для ОС, Javascript - для веба и веб-приложений.**
3. **C# это типизированный язык, Javascript условно-типизированный.**
4. **C# имеет поддержку полноценного варианта ООП для наследования, Javascript - прототипная модель наследования.**

**Ajax (Asynchronous Javascript and XM)**

- Фоновый

**Преимущества AJAX**

• Экономия трафика

• Интерфейс остается отзывчивым во время выполнения длительной операции

• Возможность реализации динамического графического интерфейса

**Недостатки AJAX**

• История посещений

• Поисковики(до определенного момента)

• Статистика

• Усложнение проекта

• Включенный JS

• Кодировки

• Плохое поведение на нестабильных соединениях

**Лекция 5**

**Пассивный сервер -** события генерируются только клиентом, сервер лишь обрабатывает запросы.

ВОПРОС: Что делать, если нужно, чтобы и сервер мог генерировать события для обновления данных на клиенте?

* **Polling** - опрос сервера клиентом на наличие изменений через определенные промежутки времени. Более быстро, но информация менее актуальна.
* **Long Polling** - клиент посылает запрос, сервер создает соединения и держит его открытым, во время того, как соединение открыто, сервер может передавать на клиент изменения. По истечении времени жизни соединения (TimeOut), соединение закрывается и создается новое.

Актуальная информация, но постоянно открывать и поддерживать новые соединения сложно и долго.

Открывать соединения сложно и долго + приходится постоянно поддерживать соединения.

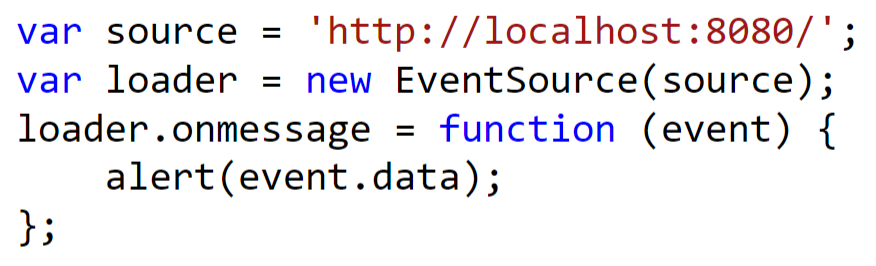
**Активный сервер -** сервер имеет возможность генерации событий. Пользователь должен “слушать” и обрабатывать изменения сервера.

**Server-sent events(SSE) -** позволяет на основе HTML5 автоматически получать обновления со стороны сервера. (push-сообщения). Используются протоколы HTTP/HTTPS.

Не поддерживается Internet Explorer-ом.

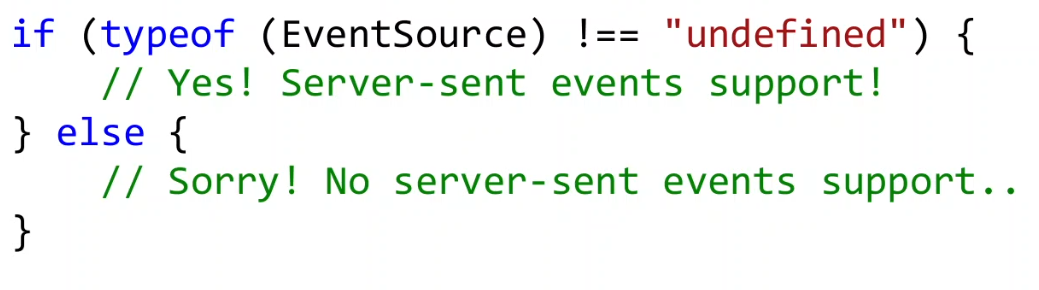
SSE реализуется с помощью EventSource и события onmessage(js) - срабатывает, когда от сервера приходит какое-нибудь сообщение.

**Подписка на событие**

****

1. source - это url, по которому будет установлено соединение
2. loader - объект класса EventSource с параметром source
3. loader.onmessage - определяет событие, которое будет отрабатывать каждый раз, когда на сервере происходят изменения. В приведенном примере просто будут появляться предупреждающее сообщение в браузере.

**Проверка поддержки SSE браузером**

****

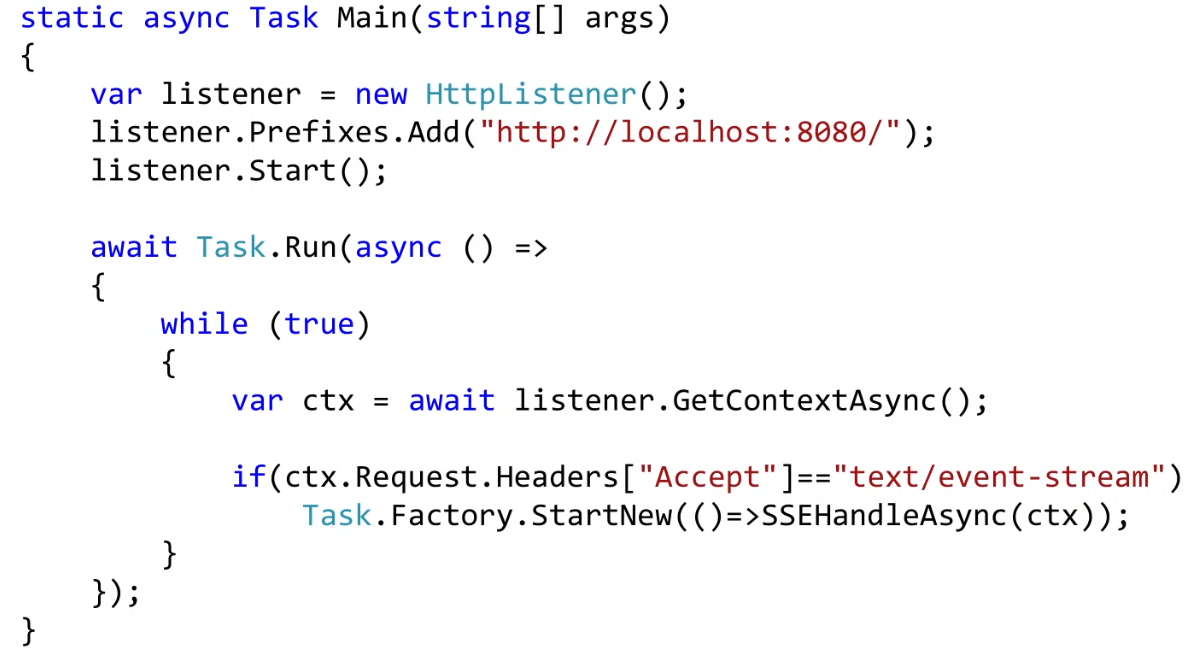
Если сервер поддерживает SSE, можно устанавливать соединение. Если нет, то нужно использовать **Polling**, либо **Long Polling.**

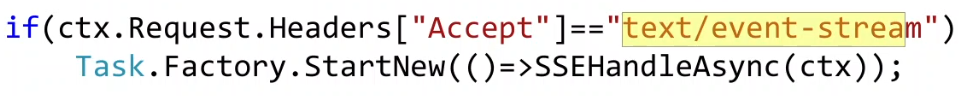
ВОПРОС: Что будет происходить на сервере при подписке клиента на его события?

1. Сервер будет получать request за заголовком **Accept:text/event-stream.**
2. Далее сервер формирует новое соединение и держит его постоянно открытым.
3. Сервер генерирует ответы в текстовом формате.

Ответ содержит ContentType, http код 200(успешный запрос), при этом ответ не содержит размер передаваемых данных. Также сервер может генерировать в качестве ответа многострочный текст(используя /n).

Для завершения ответа и передачи получившегося сообщения пишется \n\n. После этого сообщения будет передано клиенту.

**Пример**: 

1. listener - прослушивает определенный url (видимо на наличие изменений на сервере).
2. 

если в заголовке запроса приходит заголовок, который обозначает SSE соединение, то запускаем SSEHandleAsync (в него мы закладываем свой смысл).

Ниже сам Handle.



В данном примере на странице в браузере будут тикать часики.

1. msg - передаем туда строку с текущем временем. (\n\n - обозначаем, что строка закончена, ее можно отправлять клиенту)
2. bytes - заворачиваем строку в байты.
3. response.OutputStream… - отправляет сообщение клиенту.
4. response.OutputStream.Flush - обновляет соединение и фиксирует, что сообщение отправлено клиенту.
5. Task.Delay(1000) - между итерация цикла ждем 1 секунду.

В результате на сайте просто будет идти время.

**SSE - это односторонняя связь.**

Клиент слушает сервер - сервер генерирует события. Если клиент хочет отправить сообщение серверу, то ему нужно будет отправлять request.

ВОПРОС: Что делать, если нужна двусторонняя связь?

Нужно использовать **WebSocket** - это:

* Протокол (не HTTP) двунаправленной связи поверх TCP для обмена сообщениями между сервером и клиентом в реальном времени
* Для установления соединения, участники по HTTP устанавливают соединение
* Поддерживается всеми современными популярными веб-браузерами

**В чем смысл?** - Устанавливается соединение между клиентом и сервером, после того, как оно установлено, сообщения могут передаваться в обоих направлениях (сервер - клиент, клиент - сервер). Прежде чем установить сообщение по этому протоколу, клиент и сервер устанавливают соединение по обычному HTTP ( просто для проверки, что они вообще могут соединится хотя бы по HTTP).

**Установление соединения**



С помощью WebSocket можно отправить картинку.

Лекция 6

### **База данных (БД) –**

совокупность взаимосвязанных, хранящихся вместе данных при наличии такой минимальной избыточности, которая допускает их использование оптимальным образом для одного или нескольких приложений.

Создание базы данных, ее поддержка и обеспечение доступа пользователей к ней осуществляется централизованно с помощью специального программного инструментария – системы управления базами данных.

**Система управления базами данных (СУБД)** – это комплекс программных и языковых средств, необходимых для создания баз данных, поддержания их в актуальном состоянии и организации поиска в них необходимой информации.

### **Модели данных**

Данные в БД хранятся и обрабатываются согласно модели данных:

* Описание типов и структур данных в БД
* Манипулирование данными
* Поддержка целостности БД

### **Классификация БД по модели**

* **Навигационные** (ссылки)
  + Иерархические
  + Сетевые
  + Графовые
* **Реляционные**
* **Объектные** (объектно-ориентированные)
* **Документо-ориентированные**

### **История**

* 1960-е Навигационные СУБД
* 1970-е Реляционные СУБД
* 1980-е SQL, СУБД для настольных ПК
* 1990-е Объектно-ориентированные СУБД
* 2000-e NoSQL и NewSQL

**Реляционные БД**

* 1969, Эдгар Франк Кодд
* Реляционная модель данных
* БД –связанные таблицы
* Наиболее популярная модель БД

### **Синонимы**

* Таблица, отношение, реляция
* Кортеж, строка, объект, сущность
* Столбец, поле, атрибут

### **Связи между таблицами**

Виды связей:

* Один-к-одному
  + Студент-зачетка
* Один-ко-многим
  + Группа-студенты
* Многие-ко-многим
  + Предметы-преподаватели

### **Реализация связей**

**Первичный ключ (Primary Key, PK)**

* У каждой таблицы должен быть ***уникальный набор полей*** ***для однозначной идентификации записей***
  + Части PK могут быть не уникальны
* Очень часто –целочисленное поле ID
  + Часто генерируемый автоматически

**Внешний ключ (Foreign Key, FK)**

* Набор полей, по которым однозначно идентифицируется ДРУГАЯ строка в ДРУГОЙ таблице
* **Ссылка** на Primary Key в другой таблицы

### **Проблемы БД**

**Избыточность**

* Дублируются связи
  + Студент-группа
  + Группа-факультет
* Очевидно, предметы связаны с группами, не нужно их писать для каждого студента
* Ссылочная целостность
* Корректность, достоверность, адекватность, логичность БД
* Если расформировывают группу, куда девать связанных с ней студентов?
  + Оставить?–ссылка на несуществующую группу
  + Удалить?(каскадно) –но они же учатся

### **Аномалии**

* **Модификации** – изменение данных может повлечь просмотр всей таблицы и соответствующее изменение некоторых записей
* **Удаления** – при удалении записи из таблицы может пропасть информация, напрямую не связанная с удаляемой записью
* **Добавления** – невозможность помещения неполной информации, необходимость дополнительного просмотра таблицы

### **Нормальные формы**

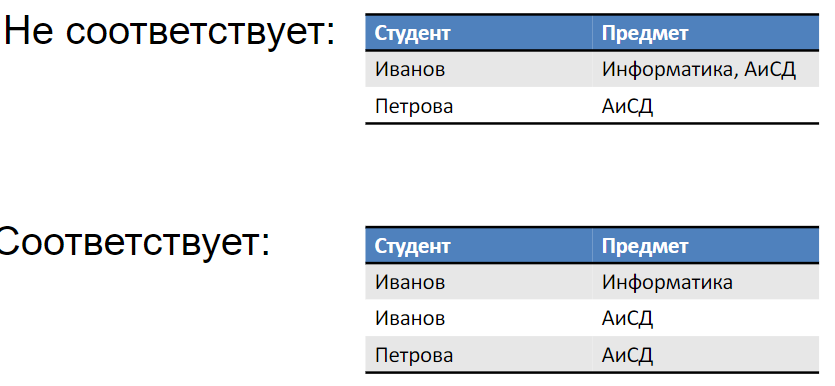
**Normal Forms**

* Для устранения наиболее частых ошибок проектирования БД (аномалий)
* Один из самых популярных вопросов на собеседованиях

**Метод нормальных форм**

* **Анализ** полей отношения, **декомпозиция** отношения на несколько взаимосвязанных отношений на основе процедур нормализации

**1НФ**

* В ячейке одно значение
  + Простое, цельное, неделимое
* Не должно быть повторений строк
  + Пример нарушения:
    - Список значений в ячейке
    - Решение –декомпозиция на несколько записей

**Функциональная зависимость**

* B зависит от A (A определяет B)
* A -> B
* Если два объекта совпадают по A, то они совпадают и по B
  + Если у студентов одна группа, то у них один факультет. Значит, есть функциональная зависимость «Группа->Факультет»

**2НФ**

* 1НФ
* Каждый неключевой атрибут ***неприводимо*** зависит от первичного ключа ***Неприводимость*** – в составе первичного ключа отсутствует подмножество полей, от которых можно вывести данную функциональную зависимость
* Не должно быть зависимостей только от части ключа



**Транзитивность**

A < B, B < C – значит, A < C

A = B, B = C – значит A = C

A -> B, B -> C – значит A -> C

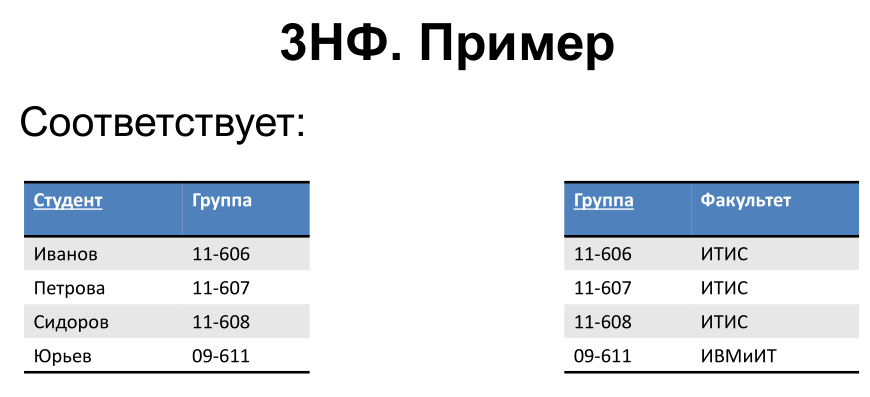
-> -импликация

* Пример нетранзитивного отношения:
  + A ≠ B, B ≠ C - ещё не значит, что А ≠ С

**3НФ**

* 2НФ
* Ни один неключевой атрибут не находится в транзитивной функциональной зависимости от первичного ключа

*Проще:* Выносить в отдельные таблицы все неключевые поля, содержимое которых может относиться к нескольким записям таблицы.



**НФ Бойса-Кодда**

* Усиленная 3НФ
  + Не существует функциональной зависимости, в которой левая часть (детерминант) не является потенциальным ключом отношения
  + В случае одного потенциального ключа совпадает с 3НФ
* 3НФ не совсем подходит, если
* У отношения два или более потенциальных ключа
* Два и более потенциальных ключа – составные
* Потенциальные ключи имеют хотя бы один общий атрибут



### **Structured Query Language (SQL)**

* Декларативный
* Основан на исчислении кортежей (см. теорема Кодда о равномощности
* реляционной алгебры, исчисления кортежей и SQL)
* Имеет диалекты (PL/SQL, Transact-SQL,…)

**SQL**

* Data Definition Language (DDL)
  + create, alter, drop
* Data Manipulation Language (DML)
  + select, insert, update, delete
* Data Control Language (DCL)
  + grant, revoke
* Transaction Control Language (TCL)
  + commit, rollback

### **СУБД**

*ПО для работы с базами данных*

Классификация по способу доступа:

* Файл-серверный:
  + FoxPro, Access
* Клиент-серверный:
  + MS SQL, PostgreSQL, Oracle, MySQL
* Встроенный:
  + SQLite

### **Работа с БД**

* Для взаимодействия приложения с БД необходим провайдер данных (для каждой СУБД свой)
* Также провайдеры разные в зависимости от технологии работы с БД

**Способы работы с БД в C#**

* ADO.NET System.Data.SqlClient
* LINQ to SQL System.Data.Linq
* LINQ to Entities System.Data.Entity

**ADO.NET. Провайдеры**

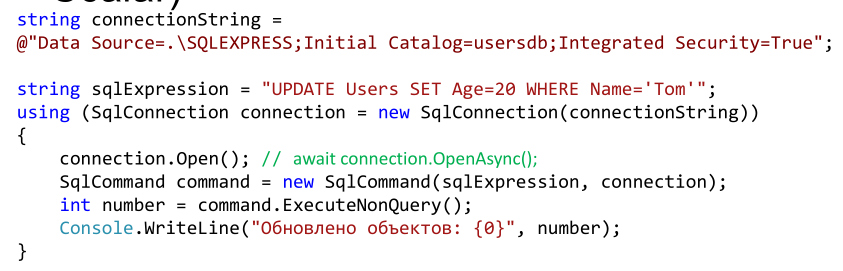
* Встроенные
  + MS SQL Server
  + OLE DB (Предоставляет доступ к некоторым старым версиям MS SQL Server, а также к БД Access, DB2, MySQL и Oracle)
  + ODBC (Провайдер для тех источников данных, для которых нет своих провайдеров)
  + Oracle
  + EntityClient. Провайдер данных для технологии ORM Entity Framework
  + SQL Server Compact 4.0
* Специальные: MySql, Postgresql и т.д.

**ADO.NET. Объекты**

* Connection
* Command
* DataReader
* DataSet
* DataAdapter

**SQL Client**

* Позволяет подключаться к БД и выполнять команды (NonQuery, Reader, Scalar)



### **Строка подключения**

* Зависит от СУБД и провайдера
  + Data Source: указывает на название сервера.
  + Initial Catalog: указывает на название базы данных на сервере
  + Integrated Security: устанавливает проверку подлинности
* Лучше жестко не кодировать, а выносить в конфиги

### **Типичные ошибки**

* Unhandled Exception: System.ArgumentException: Keyword not supported...
* Cannot open database "название базы данных" requested by the login. The login failed. Login failed for user 'название\_пользователя‘
* A network-related or instance-specific error occurred while establishing a connection to SQL Server. The server was not found or was not accessible...

### **Пул подключений**

SqlConnection connection;

connection = new SqlConnection(connectionString);

connection.Open();

Console.WriteLine(connection.ClientConnectionId);

connection.Close();

connection.Open();

Console.WriteLine(connection.ClientConnectionId);

connection.Close();

**SqlCommand**

SqlCommand command = new SqlCommand();

command.CommandText = "SELECT \* FROM Users";

command.Connection = connection;

• ExecuteNonQuery

• ExecuteReader

• ExecuteScalar

**SqlDataReader**

string sqlExpression = "SELECT \* FROM Users";

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

SqlCommand command = new SqlCommand(sqlExpression, connection);

SqlDataReader reader = command.ExecuteReader();

if (reader.HasRows) // если есть данные

{

Console.WriteLine("{0}\t{1}\t{2}", reader.GetName(0), reader.GetName(1), reader.GetName(2));

while (reader.Read()) // построчно считываем данные (можно сразу типизировать)

{

object id = reader.GetValue(0);

object name = reader.GetValue(1);

object age = reader.GetValue(2);

Console.WriteLine("{0} \t{1} \t{2}", id, name, age);

}

}

reader.Close();

}

**Скалярные данные**

string sqlExpression = "SELECT COUNT(\*) FROM Users";

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

SqlCommand command = new SqlCommand(sqlExpression, connection);

object count = command.ExecuteScalar();

command.CommandText = "SELECT MIN(Age) FROM Users";

object minAge = command.ExecuteScalar();

Console.WriteLine("В таблице {0} объектов", count);

Console.WriteLine("Минимальный возраст: {0}", minAge);

}

**Параметризация запросов**

string name = "Tom',10);INSERT INTO Users (Name, Age) VALUES('Hack"; // sql-инъекция

int age = 23;

string sqlExpression = String.Format("INSERT INTO Users (Name, Age) VALUES ('{0}', {1})", name, age);

// чтобы избежать инъекций лучше использовать параметризованные запросы

int age = 23;

string name = "T',10);INSERT INTO Users (Name, Age) VALUES('H";

string sqlExpression = "INSERT INTO Users (Name, Age) VALUES (@name, @age)";

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

SqlCommand command = new SqlCommand(sqlExpression, connection);

SqlParameter nameParam = new SqlParameter("@name", name);

command.Parameters.Add(nameParam);

SqlParameter ageParam = new SqlParameter("@age", age);

command.Parameters.Add(ageParam);

int number = command.ExecuteNonQuery();

Console.WriteLine("Добавлено объектов: {0}", number); // 1

}

### **Транзакции**

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

SqlTransaction transaction = connection.BeginTransaction();

SqlCommand command = connection.CreateCommand();

command.Transaction = transaction;

try

{

// выполняем две отдельные команды

command.CommandText = "INSERT INTO Users (Name, Age) VALUES('Tim', 34)";

command.ExecuteNonQuery();

command.CommandText = "INSERT INTO Users (Name, Age) VALUES('Kat', 31)";

command.ExecuteNonQuery();

transaction.Commit(); // подтверждаем транзакцию }

catch (Exception ex)

{

transaction.Rollback();

}

}

### **Файлы в БД**

command.CommandText = @"INSERT INTO Images VALUES (@FileName, @Title, ImageData)";

command.Parameters.Add("@FileName", SqlDbType.NVarChar, 50);

command.Parameters.Add("@Title", SqlDbType.NVarChar, 50);

command.Parameters.Add("@ImageData", SqlDbType.Image, 1000000);

string filename = @"C:\Pictures\cats.jpg";

string title = "Коты";

string shortFileName = filename.Substring(filename.LastIndexOf('\\')+1); // cats.jpg

byte[] imageData;

using (System.IO.FileStream fs = new System.IO.FileStream(filename, FileMode.Open))

{

imageData = new byte[fs.Length];

fs.Read(imageData, 0, imageData.Length);

}

// передаем данные в команду через параметры

command.Parameters["@FileName"].Value = shortFileName;

command.Parameters["@Title"].Value = title;

command.Parameters["@ImageData"].Value = imageData;

### **Работа с базой данных**

* При работе с базой данных из приложения нам нужно работать с экземплярами классов-сущностей

CREATE TABLE "STUDENTS" (

"ID" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,

"NAME" VARCHAR(30) NOT NULL,

"GROUP\_NAME" VARCHAR(10)

);

public class Student

{

public int Id { get; set; }

public string Name { get; set; }

public string GroupName { get; set; }

}

**Student -> Group**

CREATE TABLE “GROUPS”

“ID” INTEGER NOT NULL PRIMARY KEY …,

“NAME” VARCHAR(10) NOT NULL,

);

CREATE TABLE “STUDENTS” (

“ID” INTEGER NOT NULL PRIMARY KEY …,

“NAME” VARCHAR(30) NOT NULL,

“GROUP\_ID” INTEGER REFERENCES “GROUPS”(ID)

);

class Group

{

public int Id { get; set; }

public string Name { get; set; }

public ICollection<Student> Students { get; set; }

}

class Student

{

public int Id { get; set; }

public string Name { get; set; }

public Group Group { get; set; }

public int GroupId { get; set; }

}

### **Data Access Layer**

* Слой доступа к данным
  + Позволяет инкапсулировать код для работы с БД
  + В итоге слой бизнес-логики становится независимым от реализации методов доступа к БД
* Реализация: DAO / ORM

### **Data Access Object**

* •Абстрактный интерфейс к какому-либо типу базы данных или механизму хранения
* По сути набор методов для извлечения и записи данных из/в БД; преобразование данных к объектам
* В классическом варианте содержит только методы для CRUD операций, получая или передавая в качестве аргумента так называемый DTO (Data Transfer Object)
* Проще в реализации, чем ORM, но менее гибкий

### **Object-relational mapping**

* Сопоставление БД с классами
* Самая популярная технология работы с БД
  + EntityFramework, Dapper, NHibernate
  + JPA, Hibernate
  + Django models
  + Ruby on Rails Active Records, etc

### **Отображение**



### **Entity Framework**

* Рекомендованная ORM для .NET Framework (в том числе Core)
* Сочетает в себе реализацию паттернов Repository и Unit of Work
* Использует LINQ для запросов к БД
* Entity Data Model для сопоставления классов сущностей с таблицами в БД

### **Взаимодействие с БД**

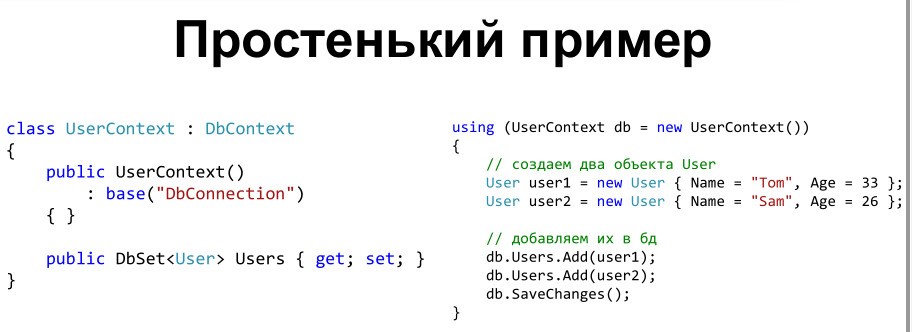
* Database First. Создание классов сущностей по существующей базе
* Model First. По модели данных создается база данных и сущности
* Code First. По классам сущностей генерируется база

**Поддержка связей**

* Поддерживаются связи
  + Один-к-одному
  + Один-ко-многим
  + Многие-ко-многим
    - При подходе CodeFirst автоматически генерируется вспомогательная таблица

### **Основа Entity Framework**

* Классы
  + DbContext – контекст данных для взаимодействия с базой
  + DbModelBuilder – сопоставляет классы с таблицами
  + DbSet/DbSet<TEntity> - наборы сущностей, соответствующие таблицам



### **Миграции**

* Обновление базы данных при изменении моделей и контекста данных
  + Создание, удаление столбцов/таблиц, ключей, индексов, хранимых процедур
* Add-Migration MigrationName
* Update-Database (-script)

ЛЕКЦИЯ 7

# MVC - фреймворк для создания веб-приложений, который реализует шаблон Model-view-controller.

# 

# **Архитектура приложений:**

Метафора архитектуры зданий - ? я думаю, что он имел ввиду схожесть в плане структуры зданий и кода или хз, нигде ничего нет.

Высокоуровневая структура системы

* структура, ответственности, интерфейсы, взаимодействия
* обладающая атрибутами качества   
   - надежность, отказоустойчивость, расширяемость. удобство использования

# Хорошая архитектура:

Эффективность системы

Гибкость системы. Откладывание ключевых решений

Расширяемость системы. YAGNI.  
  
Масштабируемость процесса разработки

Тестируемость

Возможность повторного использования

Хорошо структурированный код

# Паттерны:

Архитектурные паттерны применяются для удовлетворения атрибутам   
качества.

К примеру:

**Layered pattern** - независимые модули, взаимодействуют только со смежными.

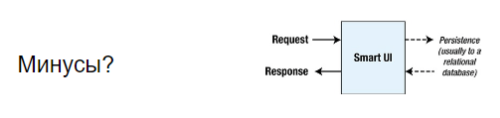
**Broken pattern** - посредник (шина), для общение модулей  
**Client-Server-pattern** - доступ большего числа клиентов к ограниченному числу ресурсов

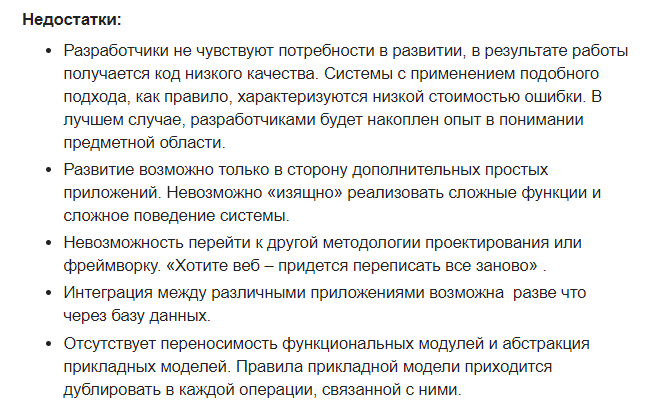
# SMART UI

Это - Интеллектуальный интерфейс пользователя

* Содержит всю логику и работает с данным напрямую
* Логика в обработчике событий UI

SMART UI Минусы:



Минусы.   
 Минусы Минусы  
Минусы Минусы  
 Минусы  
Минусы:   


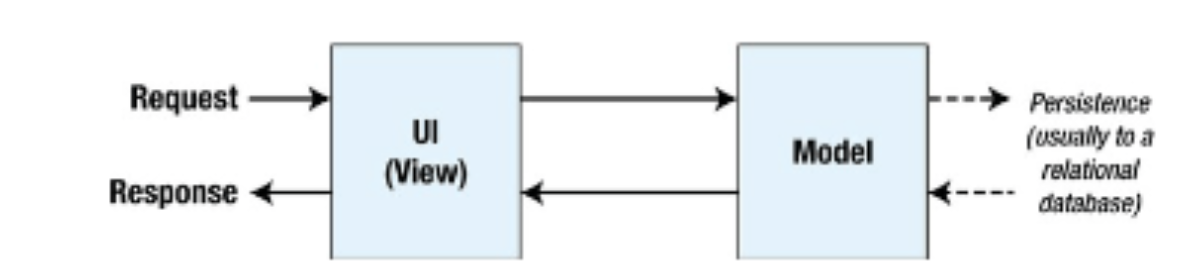
# Бизнес Логика

Это - совокупность правил, принципов.

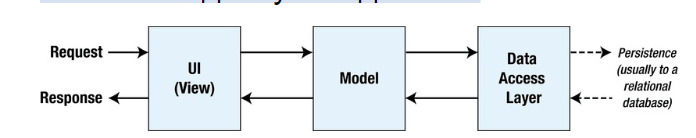
* Enterprice данные и правила работы с ними
* Поведение компоненты
* Реализация модели процессов реального мира в программной коде

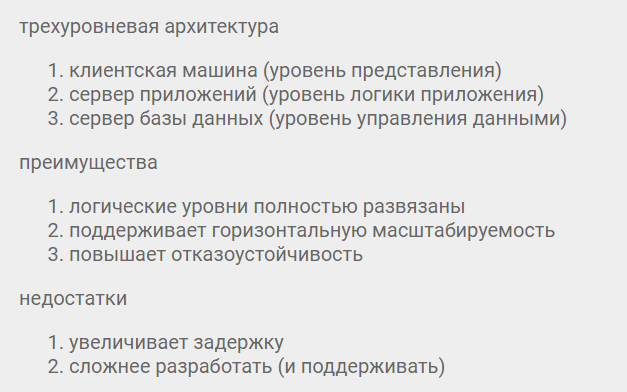
# Model - View

* Отделение модели от представлений в (UI)
* данных и действий от отображения



# Three-tier

Разделение UI, модели предметной области и доступа к данным.   


  
  
Отличие парадигмы от паттерна?

ЭТО ОТКУДА?

#### WebSockets

Это бинарный дуплексный протокол, позволяющий клиенту и серверу общаться на равных. Этот протокол можно применять для игр, чатов и всех тех приложений где вам нужны предельно точные события близкие к реальному времени.

**ЛЕКЦИЯ 8**

**Что такое OSI?**

**OSI** (Open System Interconnection model) - это некоторое взаимодействие различных сетевых устройств. Происходит такое взаимодействие посредством данной сетевой модели. Также есть 7 различных уровней взаимодействия, где каждый уровень выполняет свои какие-то функции при таком взаимодействии.

1. Физический. (*Знать наизусть не обязательно, но на сетях пригодится*)
2. Канальный.
3. Сетевой.
4. Транспортный.
5. Сеансовый.
6. Представления.
7. Прикладной.

Сетевой уровень и выше

•В основном работаем с сетевым, транспортным и прикладным уровнями

•IP –маршрутизация пакетов

•TCP –надежная передача данных по маршруту

•UDP –быстрая передача данных по маршруту без каких-либо гарантий

•HTTP –протокол приложений поверх TCP

**Что такое HTTP?**

**HTTP** — это протокол, позволяющий получать различные ресурсы, например HTML-документы. Основой HTTP является [технология «клиент-сервер»](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80). Протокол HTTP лежит в основе обмена данными в Интернете. В основе самого HTTP лежит HttpListener. (Во второй и третьей лекции это подробно разобрано).

HttpListener:

– Работает с контекстом (Context).

– Принимает запрос (Request).

– Отправляет ответ (Response).

Использование:

* для отправки запроса и приёма ответа используются классы WebRequest и WebResponse
* WebRequest - фабрика HttpWebResponse соответственно

**Socket**

* Предоставляет низкоуровневый интерфейс для приема/передачи данных по сети
* Содержит

–AddressFamily–адреса сокета

–ProtocolType–протокол сокета

–SocketType–тип сокета (Dgram, Stream)

–LocalEndPoint–адрес, по которой сокет принимает значения

–Connected –подключен ли к удалѐнному хосту

–RemoteEndPoint–адрес хоста, к которому подключен сокет

Основные методы:

•Accept - создаѐт сокет

•Bind - связывает с конечной точкой

•Close - закрывает сокет

•Connect - соединяется с удаленным хостом

•Listen - начинает прослушку запросов

•Poll - определяет состояние

•Receive - получает данные

•Send - отправляет данные

•Shutdown - блокирует прием/отправку

Что такое TCP и UDP (примеры сокетов)?

**TCP** и **UDP** это протоколы транспортного уровня (см. OSI), которые в сети отвечают за доставку данных конечному адресату и за правильность данных.

**TCP** и его особенности. Для того, чтобы начать отправку данных конечному адресату (клиенту) необходимо установить соединение между двумя этими хостами. Только после подтверждения начнется отправка данных. После самой отправки нужно ждать еще одного подтверждения получения данных от принимающей стороны. Все данные идут упорядоченно и шанс потери данных минимален. Также возможно получение повторной отправки, если в первый раз что-то потерялось при передаче. Но из-за этих свойств TCP уменьшается скорость отправки данных.

Преимущества TCP:

• Высокая надежность.

• Упорядоченная отправка данных.

• Шанс потери данных минимален.

Недостатки TCP:

• Медленная скорость передачи.

**UDP** - это противоположность TCP во всем. Для передачи данных ему не обязательно устанавливать соединение между отправителем и получателем. Информация передается без предварительной проверки готовности принимающей стороны. Это делает протокол менее надежным – при передаче некоторые фрагменты данных могут теряться. Зато скорость передачи данных по данному транспортному протоколу будет более высокой.

Плюсы и минусы UDP противоположны TCP.

Как реализуется multicast?

**Вещание(multicast)** - это метод передачи данных в сетях, при котором адресатом является является группа клиентов. Но для того, чтобы быть в этой группе нужно подписаться на клиента, который будет раздавать данных в группе. Также передача идет не дальше локальных сетей. (идет ограничение со стороны маршрутизатора).

multicast работает на UDP протоколе, ведь UDP позволяет рассылать одно сообщение группе клиентов-получателей. В отличие TCP, не нужно отправлять сообщение всем по отдельности.

Приведите пример 3 протоколов как минимум для 3 уровней OSI.

Транспортный уровень:

• UDP.

• TCP.

• SCTP.

Сетевой уровень:

• IP.

• ICMP.

• IGMP.

Прикладной:

• HTTP.

• FTP.

• SSH.

Лекция 9 (Многопоточность)

## I/O bound vs CPU bound

* Вычислительные задачи можно разделить на две категории:

– I/O bound - требующие ввода/вывода

– CPU bound - нагружающие процессор

* I/O bound задачи могут блокироваться, позволяя выполнять работу процессам, нуждающимся в процессорной мощности

## Вычислительные потоки

* **Поток или нить (thread)** – блок кода, который может выполняться одновременно с другими потоками
* Потоки – виртуализация процессора в Windows

## Потоки vs процессы

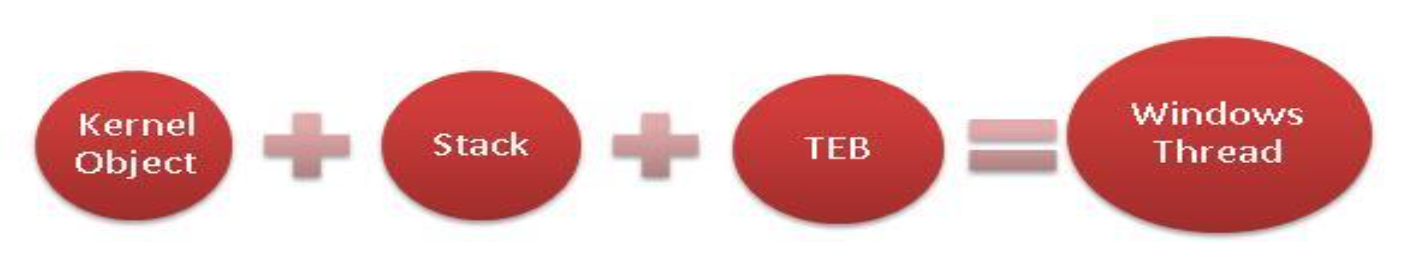
* Поток **!=** Процесс

– у процесса свое адресное пространство, у потоков – разделяемое

– у процесса есть хотя бы один поток (Main)

– 1 процесс может содержать несколько потоков

## Содержимое потока

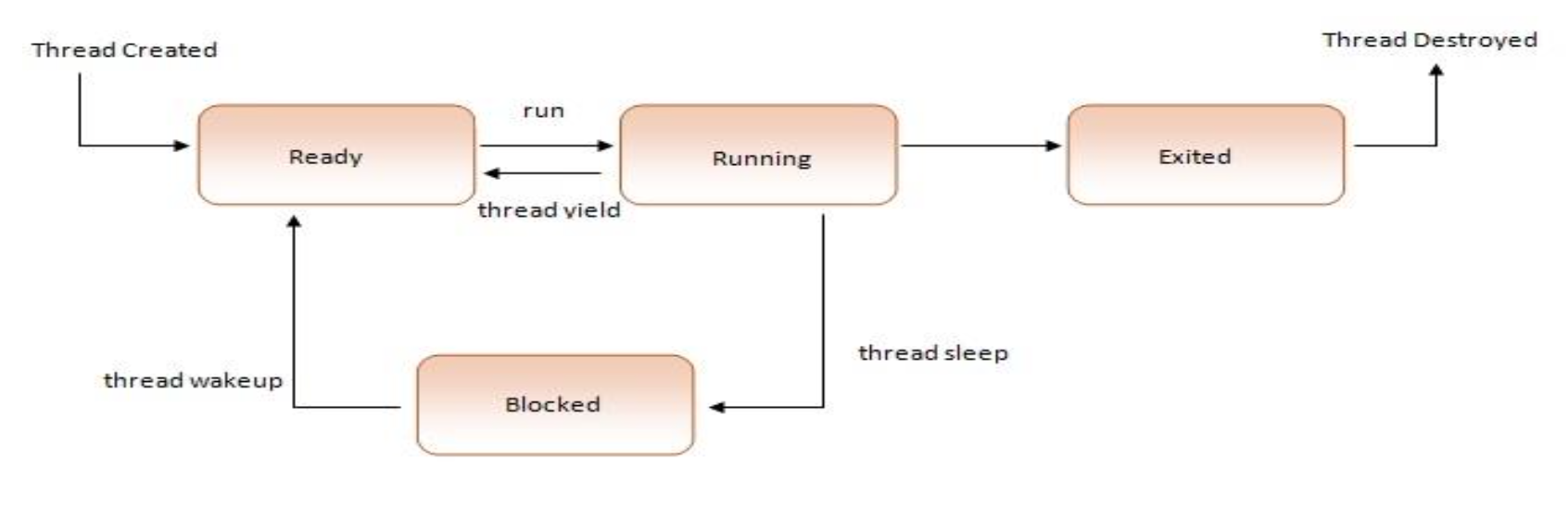


• **Thread kernel object** – структура данных для данных о потоке – регистры процессора, статистика

• **Thread Environment Block (TEB)** – данные для обработки исключений и локальное хранилище потока

• **User-mode Stack & Kernel-mode stack** (для безопасной передачи данных)

## Состояния потока



* **Running** – использует ресурсы ЦП
* **Blocked** – ждет ввода
* **Ready** – готов к запуску (не Блокирован и не Запущен)
* **Exited** – завершен, но не уничтожен

## Потоки в CLR

• Управление всеми потоками осуществляется посредством класса Thread пространства имён System.Threading

• CLR потоки в целом аналогичны Windows- потокам, но есть ряд отличий

## Создание потока

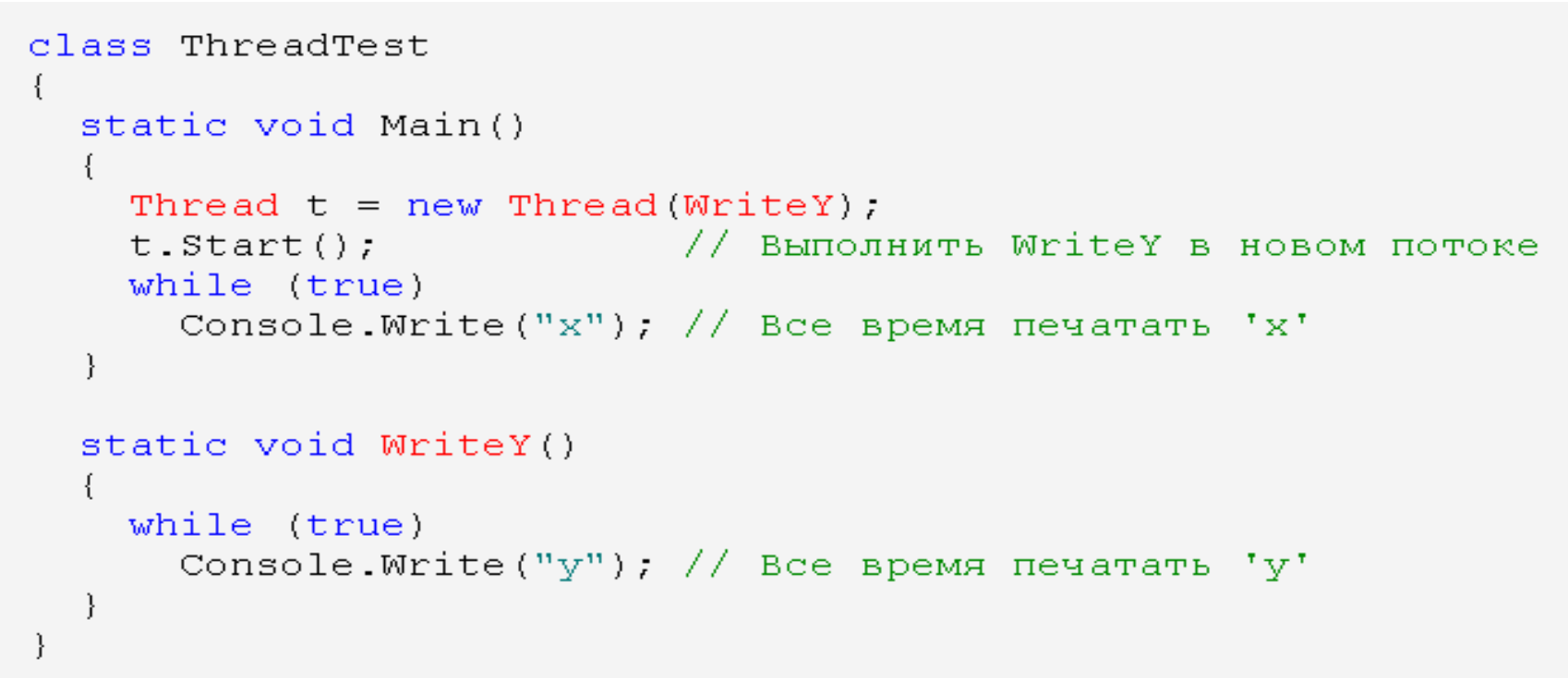
* В основном потоке создаётся экземпляр ThreadStart threadDelegate =

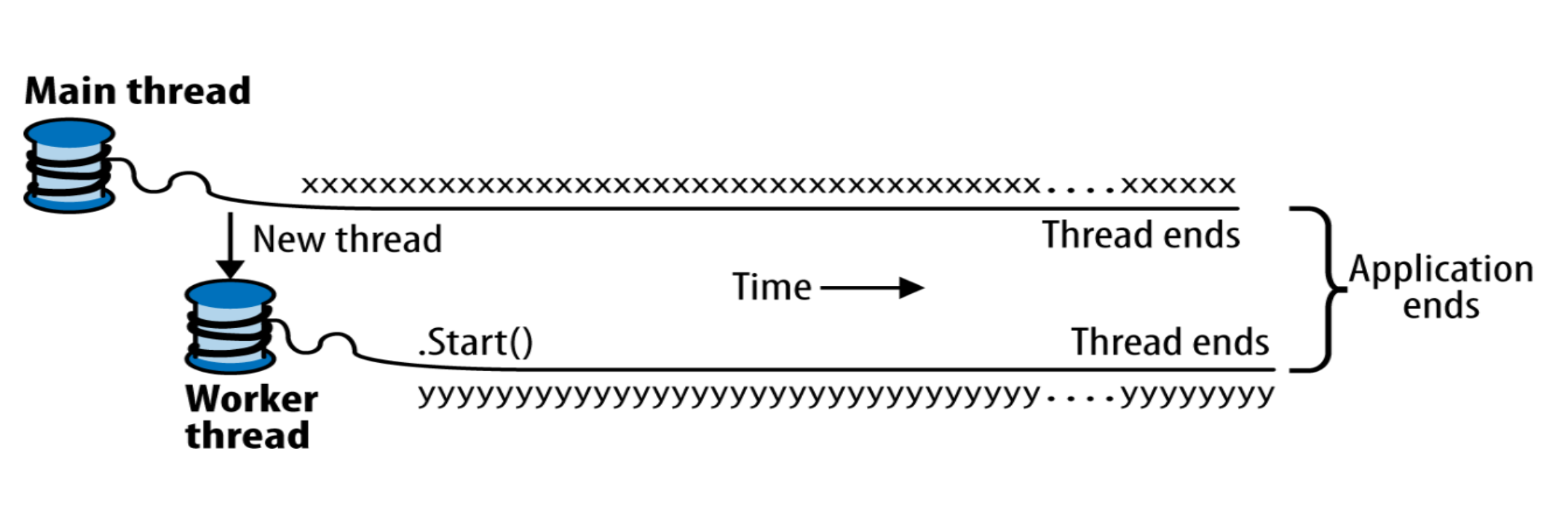
new ThreadStart(Work.DoWork);

Thread thread = new Thread(threadDelegate);

* В конструкторе указывается метод (делегатThreadStart), который будет выполняться в отдельном потоке
* Вызов метода Start у экземпляра создаёт новый поток thread.Start();

## Concurrency (Параллелизм)





## Параллелизм и переключение

**Con** – вместе, **Current** – выполнение

* На многоядерных процессорах потоки могут выполняться одновременно
* Если потоков больше, чем процессоров, происходит переключение

– выделяются кванты времени 20-30мс между переключениями. Слишком

часто плохо, слишком редко тоже

## Что происходит при переключении?

* Значения регистров процессора сохраняются в контексте ядра потока
* Из набора потоков выделяется тот, которому будет передано управление. Если выбранный поток принадлежит другому процессу, ОС переключает адресное пространство
* Значения из контекста ядра потока загружаются в регистры процессора

## GC, отладка и потоки

* GC при работе
  + – приостанавливает все потоки
  + – просматривает стеки
  + – помечает объекты в куче
  + – снова просматривает стеки (обновляя перемещенные объекты)
  + – возобновляет исполнение всех потоков
* При отладке Windows приостанавливает все потоки в каждой точке останова и на каждом шаге отладки

## Сон и передача выполнения

* Можно усыпить (блокировать) текущий поток на  
  указанное время Thread.Sleep(1000);  
  – квант времени немедленно отдаётся следующему  
  потоку из очереди
* Можно указать нулевое время Thread.Sleep(0);
* При вызове Thread.Yield(); произойдёт аналогичная передача выполнения, но только потокам того же процессора

## Воссоединение потоков

* При создании потока происходит разделение выполнения
* Можно воссоединить потоки, заставив одного потока ожидать завершения другого

var thread = new Thread(Operation); thread.Start(5);

...

thread.Join();

## Ожидание с таймаутом

* При ожидании текущий поток приостанавливает работу
* Можно указать таймаут ожидания. По окончанию ожидания вернётся bool: true – поток завершился, false – таймаут

bool result = thread.Join(1000);

Console.WriteLine(result?"Дождались":"Устали ждать");

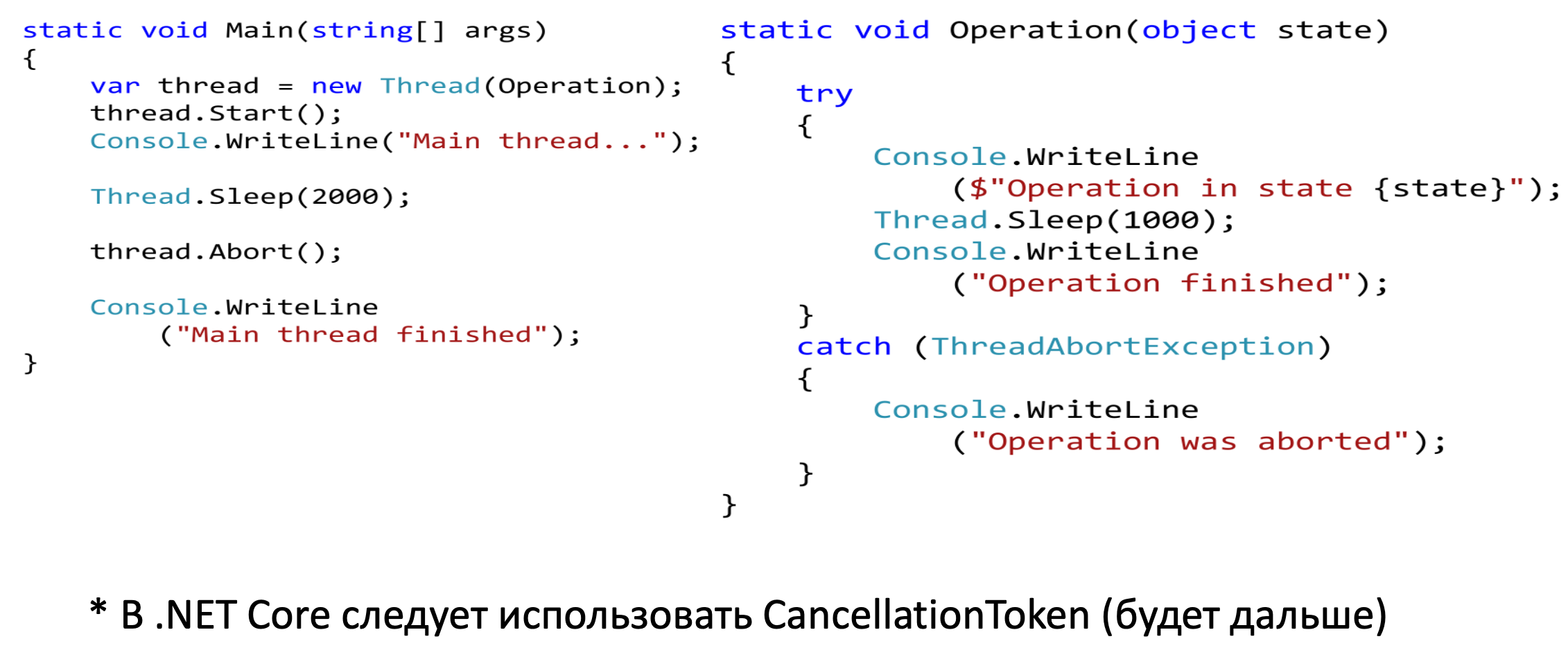
## Завершение потока

* Можно принудительно завершить поток

var thread = new Thread(Operation);

thread.Abort();

* При этом в потоке thread возникнет исключение ThreadAbortException



## Foreground & Background

* Thread.IsBackground
* При завершении всех активных foreground потоков CLR принудительно завершает все запущенные фоновые потоки немедленно без исключений
* Foreground потоки могут быть источником багов

– могут мешать завершению приложения

(пик)

## Приоритеты потоков

* ОС с вытесняющей многозадачностью должна иметь алгоритм выбора порядка и продолжительности исполнения потоков
* В Windows реализуется с помощью относительных приоритетов процессов и потоков

Thread.Priority

* Можно поменять и класс приоритета процесса

(пик)

* Потоки с низким приоритетом хорошо использовать для длительных вычислений
* Потоки с высоким приоритетом большую часть времени находятся в режиме ожидания (Windows Explorer, мгновенная реакция на действия пользователя)

## Обработка исключений

(пик)

* Исключения надо ловить внутри потоков

– При этом обычно в catch сигнализировать другим потокам о проблеме

## Работа с памятью

У каждого потока

* Свой стек
* Но при этом разделяемая куча с потоками того же процесса:

– имеют общий доступ к одному объекту, если у них у всех есть ссылка на него

## Ещё про разделимость

(пик)

## Разделяемые данные – к беде

* Использование разделяемых данных (изменяемых) может приводить к ошибкам
* даже инкремент может привести к ошибкам и к совершенно разному поведению программы
* Для обеспечения потокобезопасности,нужно синхронизировать совместный доступ потоков к ресурсу

## Потокобезопасность

* Код потокобезопасен (thread-safe), если он корректно функционирует при одновременном использовании его в нескольких потоках
* Синхронизация – один из самых грубых способов потокобезопасности, но – снижает производительность

## Исключительная блокировка

* Простой инструмент достижения потокобезопасности – lock
* Блокируется объект ссылочного типа
* Выполнение потока блокируется в ожидании снятия блокировки, дождавшись – продолжает работу

## lock

(пик)

## Кроме lock

• Mutex

* – гарантирует, что только один поток может получить доступ к  
  ресурсу или секции кода.
* – доступен из разных процессов, обеспечивая блокировку на уровне компьютера
* – обычно используется для проверки,что запущена только одна версия программы
* – медленнее lock вразы

• Semaphore

– гарантирует, что не более заданного числа потоков может получить доступ к ресурсу или секции кода.

## Semaphore

(пик)

## User-mode синхронизация

* Не блокирующие конструкции синхронизации

– Volatile – препятствуют оптимизации компилятора и перестановке инструкций, тем самым заставляют всегда перезагружать значение из памяти (а не брать из кэша процессора). Модификатор volatile y полей примитивных и ссылочных типов (а также указателей)

– Interlocked –для атомарного доступа (монопольного захвата) для записи и чтения (int, long). Cтатические методы класса Interlocked: Increment, Decrement, Add, CompareAdd, Exchange

## DEADLOCK

* Исключительное владение ресурсом
* При этом ожидание захвата еще одного ресурса
* Ресурс может освободиться только захватчиком
* Циклическое ожидание

(пик)

## Пул потоков

* Общий для CLR набор готовых для использования потоков

– потоков мало (сколько и процессоров), при необходимости создаются новые

– при простое – уничтожаются

– операции встают в очередь пула и исполняются на доступных потоках из пула

* Варианты использования – Task Parallel Library (TPL)

– ThreadPool.QueueUserWorkItem

– Асинхронные делегаты

– Через BackgroundWorker

Есть ещё непрямые варианты работы с ThreadPool: WCF, Remoting, ASP.NET, Timer, ...Async, Begin..., PLINQ

## Добавление операции в очередь

Метод ThreadPool.QueueUserWorkItem:

public static void Main() {

Console.WriteLine("Main: queuing an asynchronous operation");

ThreadPool.QueueUserWorkItem(ComputeBoundOp, 5);

Console.WriteLine("Main thread: Doing other work here...");

Thread.Sleep(10000); // Имитация другой работы (10 секунд)

}

private static void ComputeBoundOp(object state) {

}

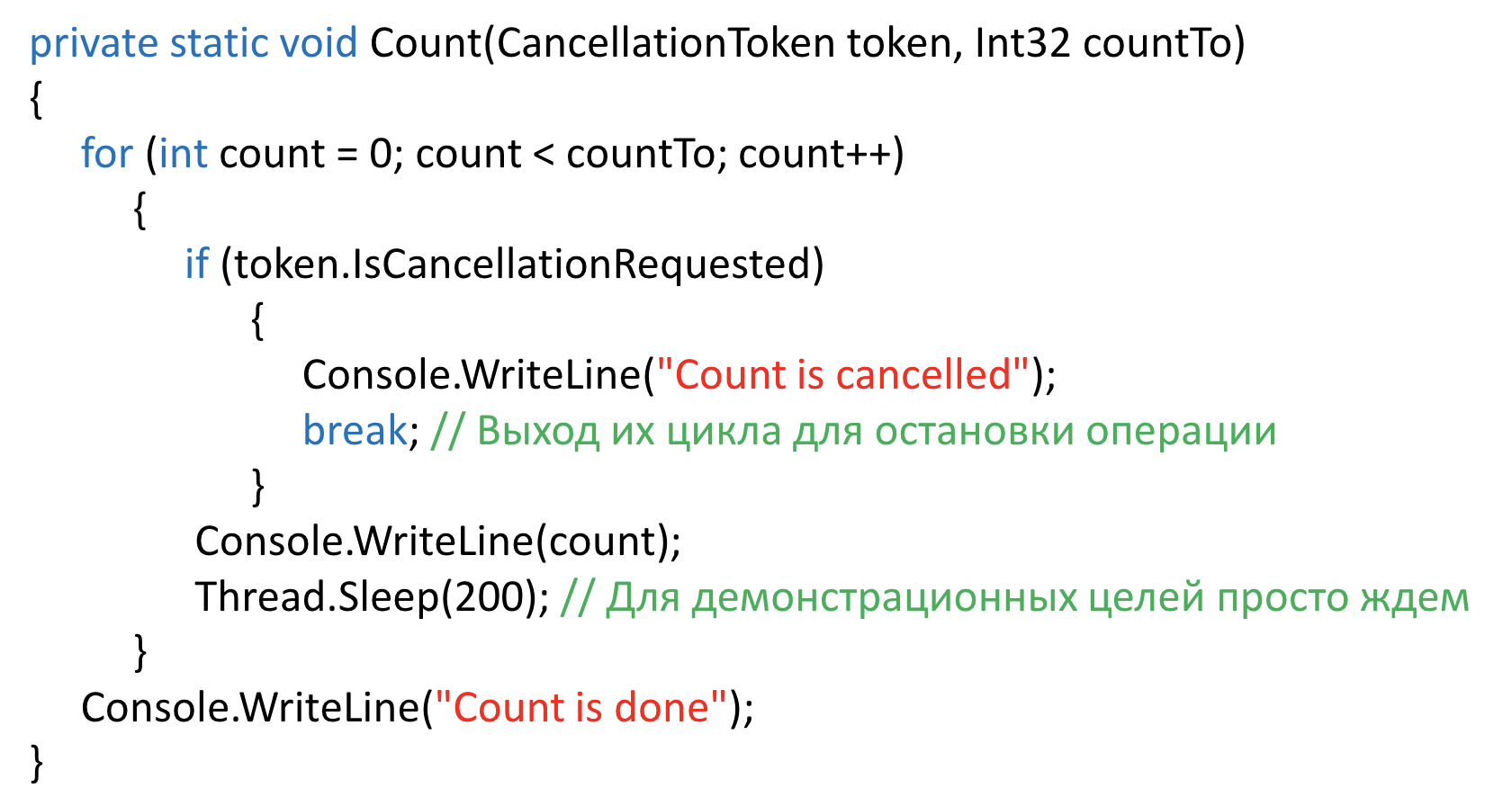
Console.WriteLine("In ComputeBoundOp: state={0}", state);

Thread.Sleep(1000);

## Отмена выполнения

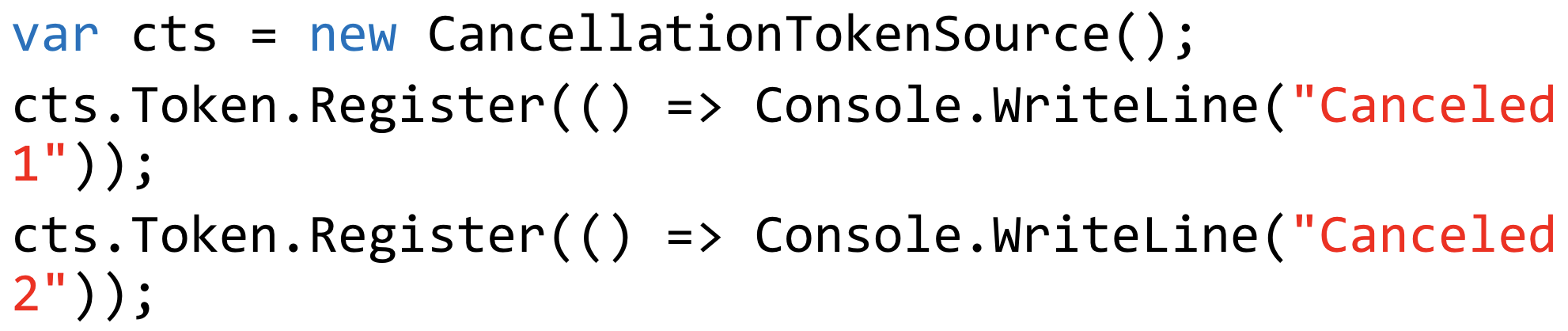
Реализуется с помощью токена:



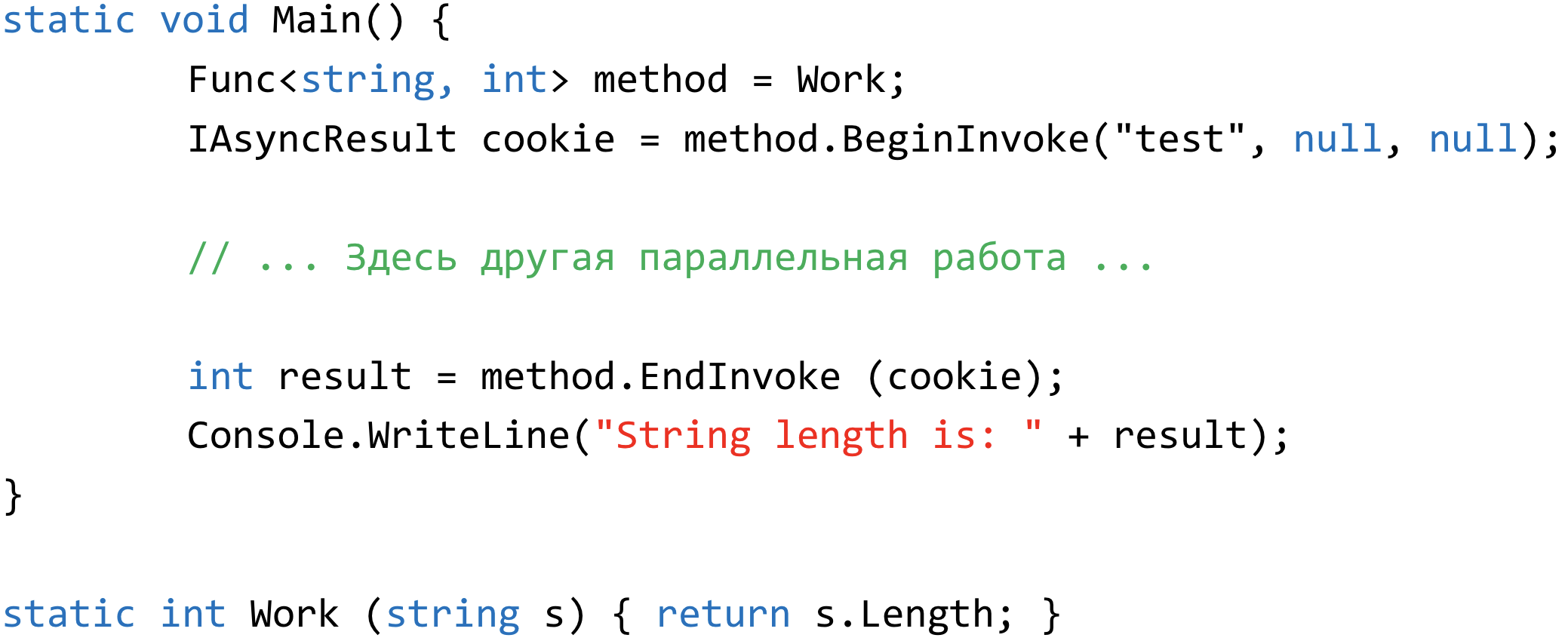


## Callback при отмене

Можно у CancellationTokenSource зарегистрировать несколько callback методов, которые будут вызываться в случае отмены операции (Cancel)



## Асинхронные делегаты



## Асинхронные операции

* Асинхронность

– форма одновременного выполнения

– вызванная операция продолжает работу в фоне, не требуя от

вызвавшего её кода ожидания завершения.

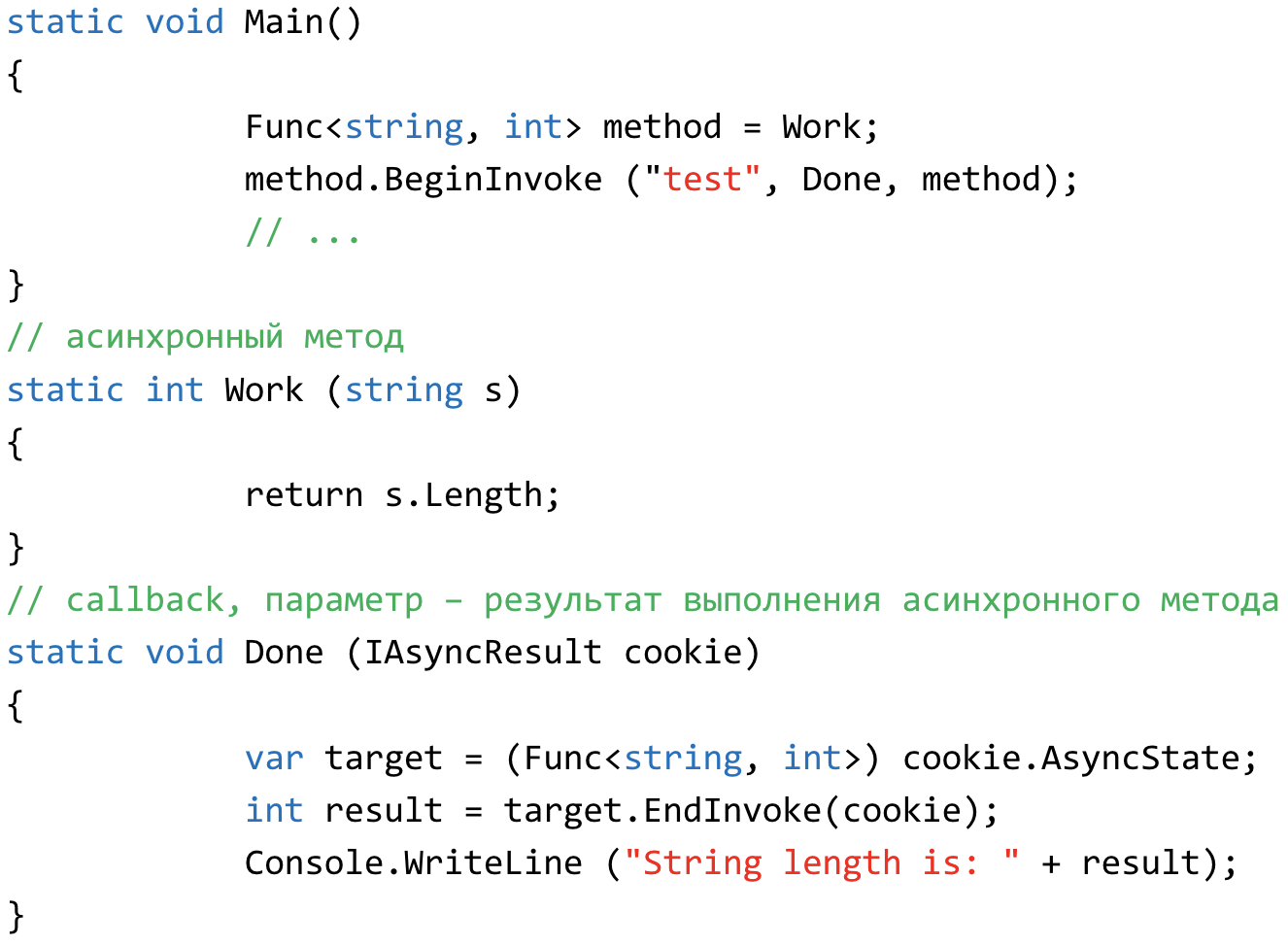
– по завершению, асинхронная операция продолжает работу операции Callback

* CPU bound операции есть смысл выполнять синхронно (в том же потоке)
* I/O bound операции вместо ожидания или блокирования вызывающего потока хорошо выполнять асинхронно
* При получении команды I/O bound операция вызывает Callback метод, позволяя обработать запрос

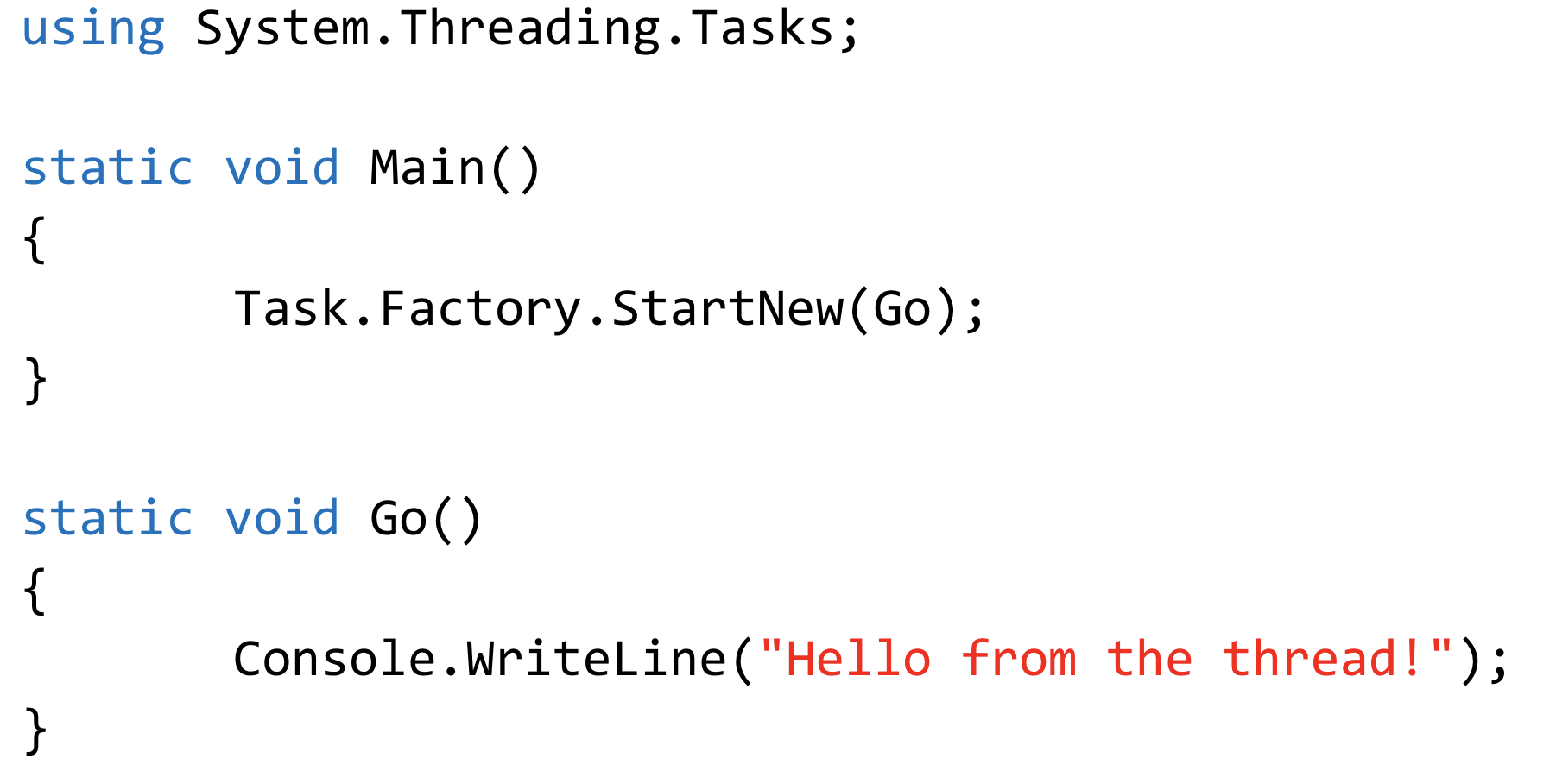
## EndInvoke

* Ожидает выполнения
* Получает значения
* Выбрасывает любое необработанное исключение в вызывающий поток
* Можно не вызывать, особенно если нет возвращаемых значений, но тогда надо ловить исключения

## IAsyncResult



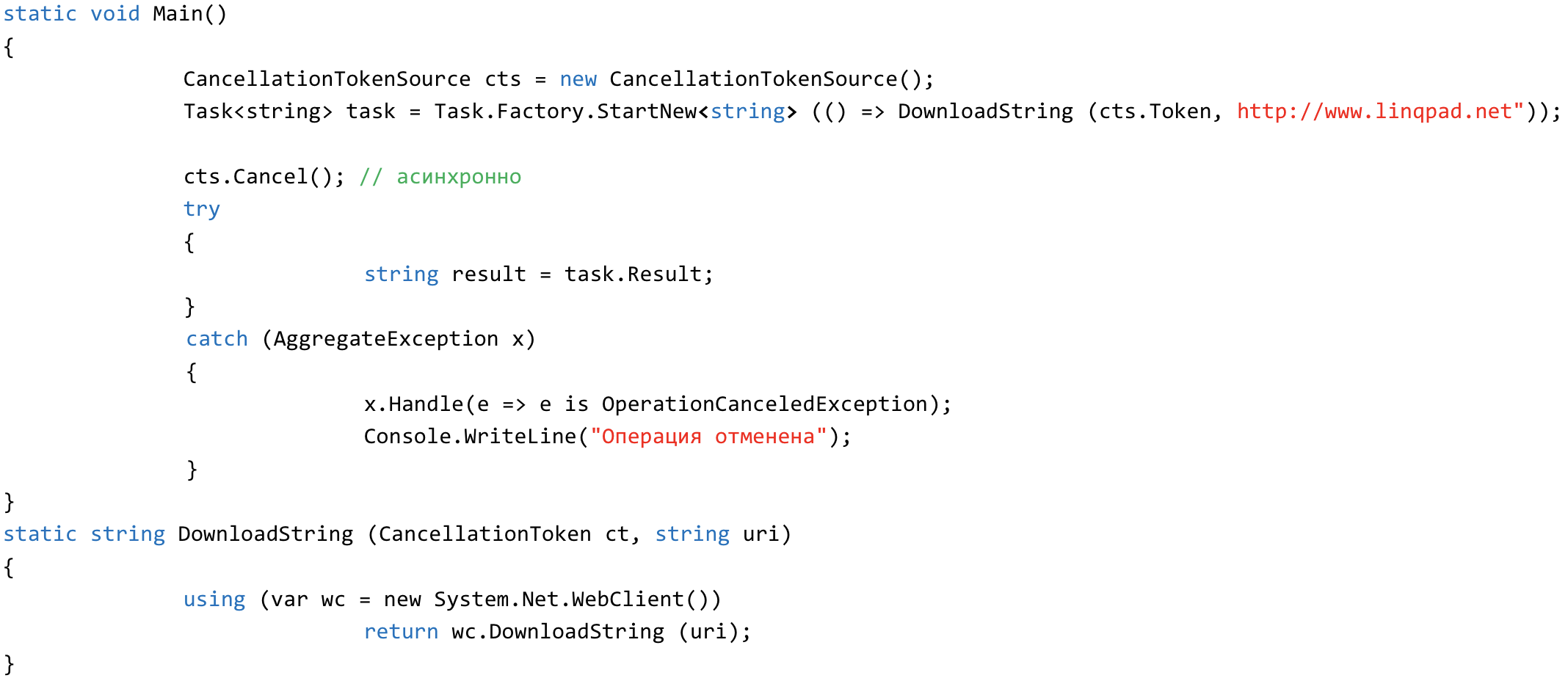
## Использование TPL



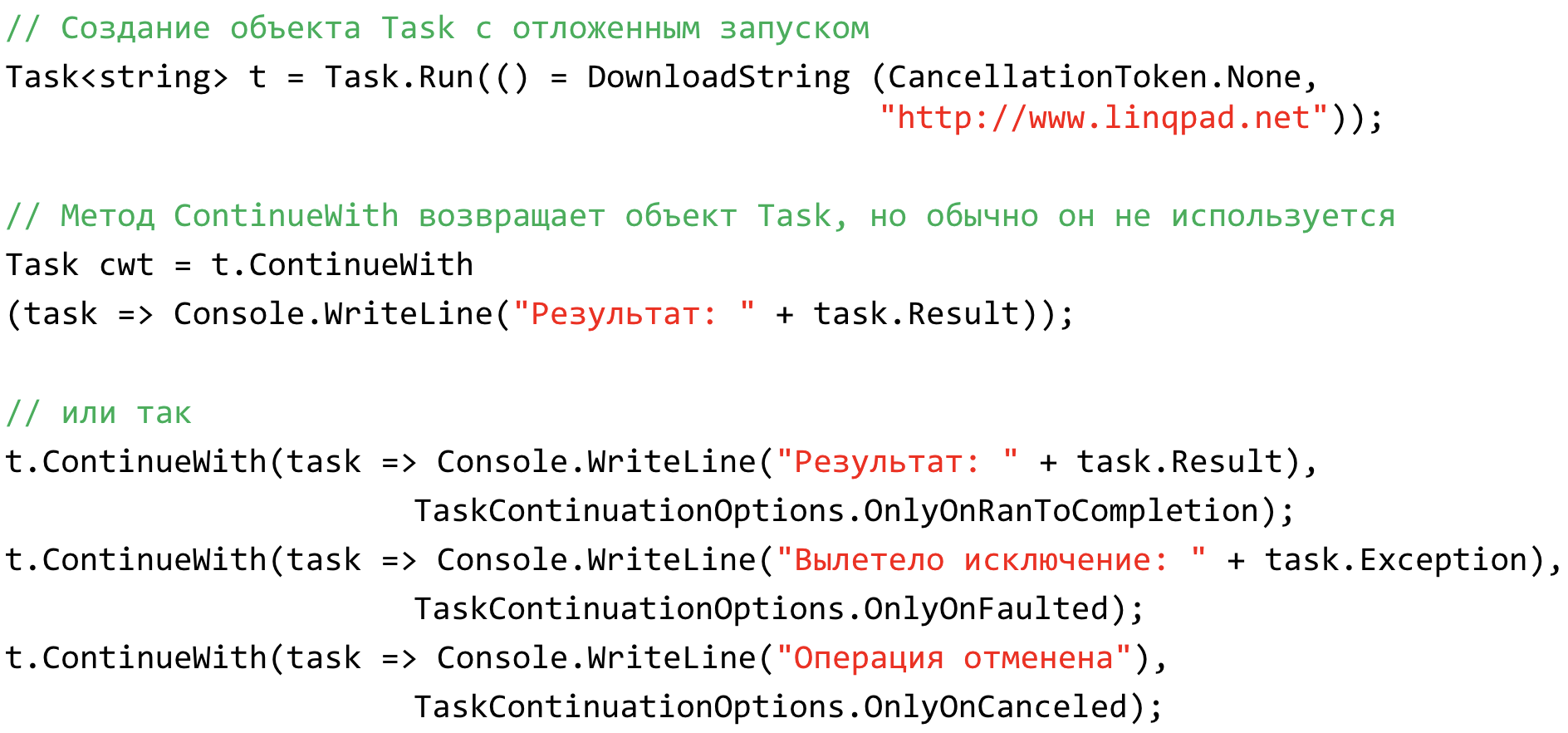
## Task<TResult>



## Отмена Taskа

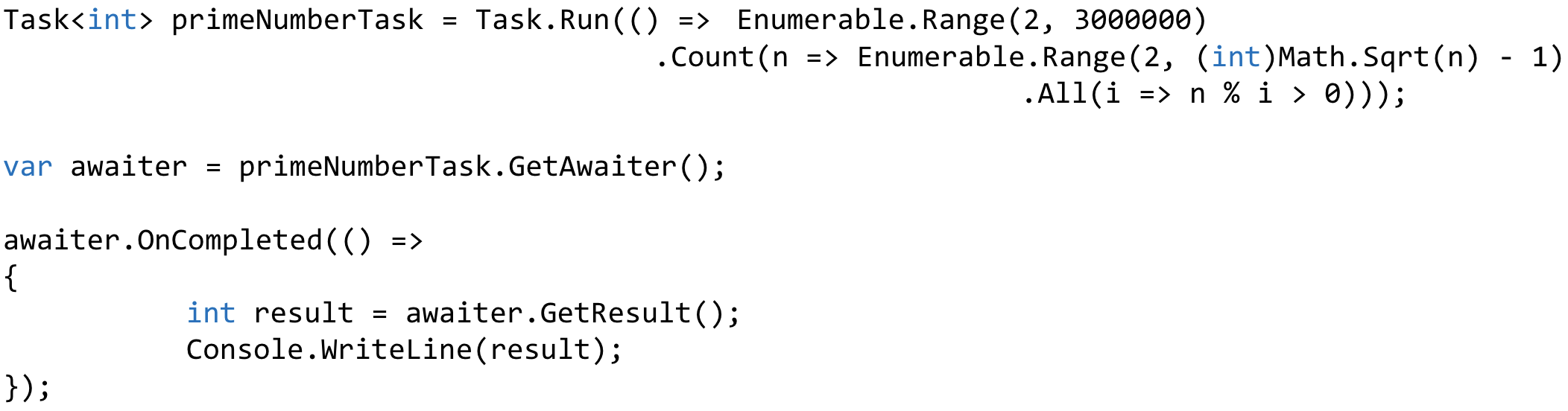


## Запуск нового задания при завершении

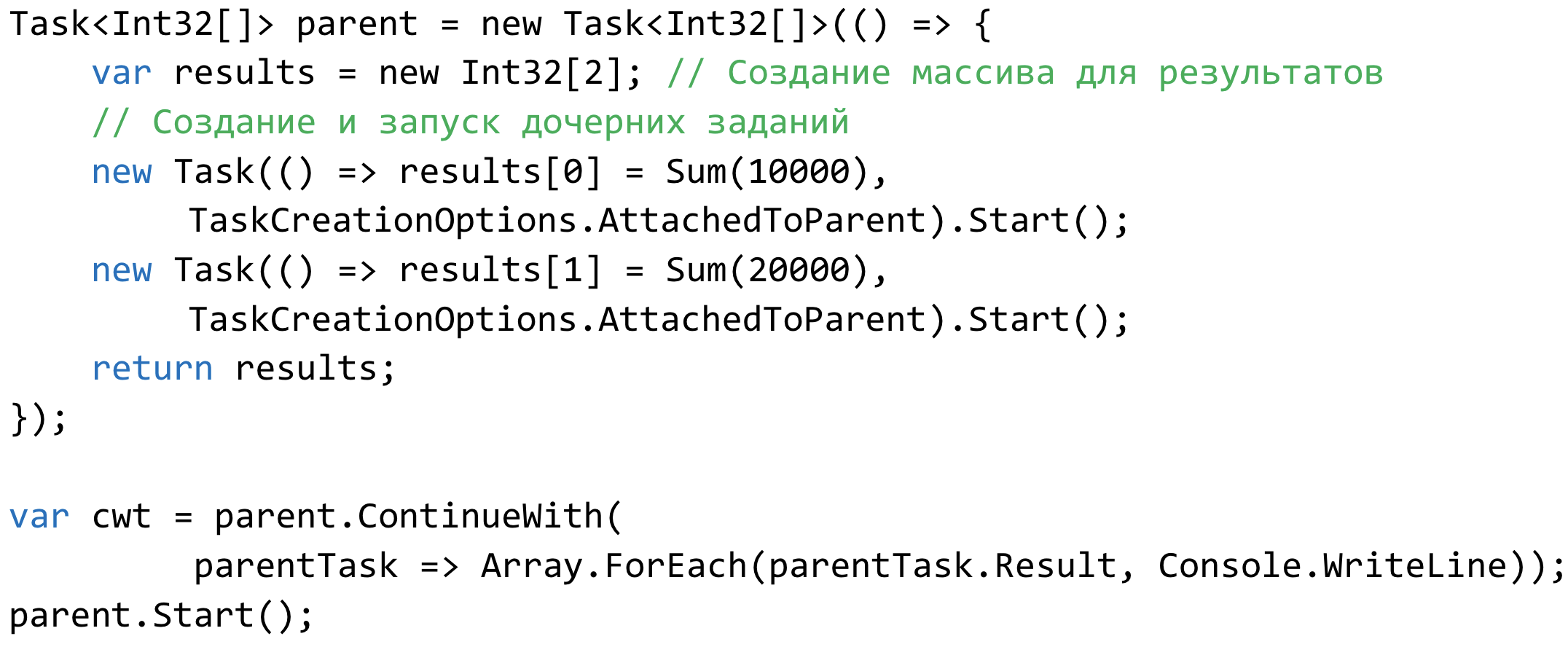


## TaskAwaiter

У задачи Task можно получить объект TaskAwaiter, ожидающий завершения задачи и уведомляющий методы по завершению (OnCompleted) и имеющий доступ к результату (GetResult)

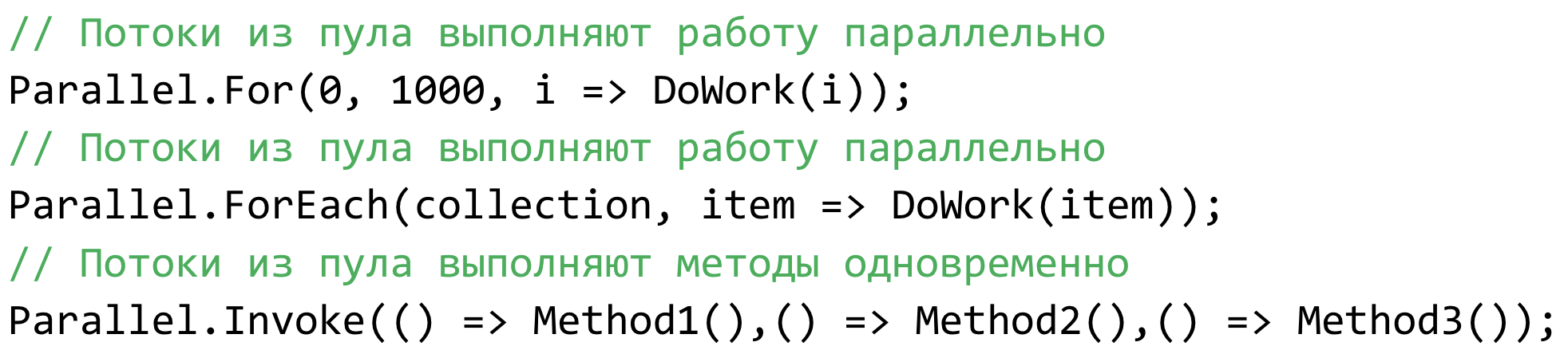


## Вложенные задания



## Класс Parallel

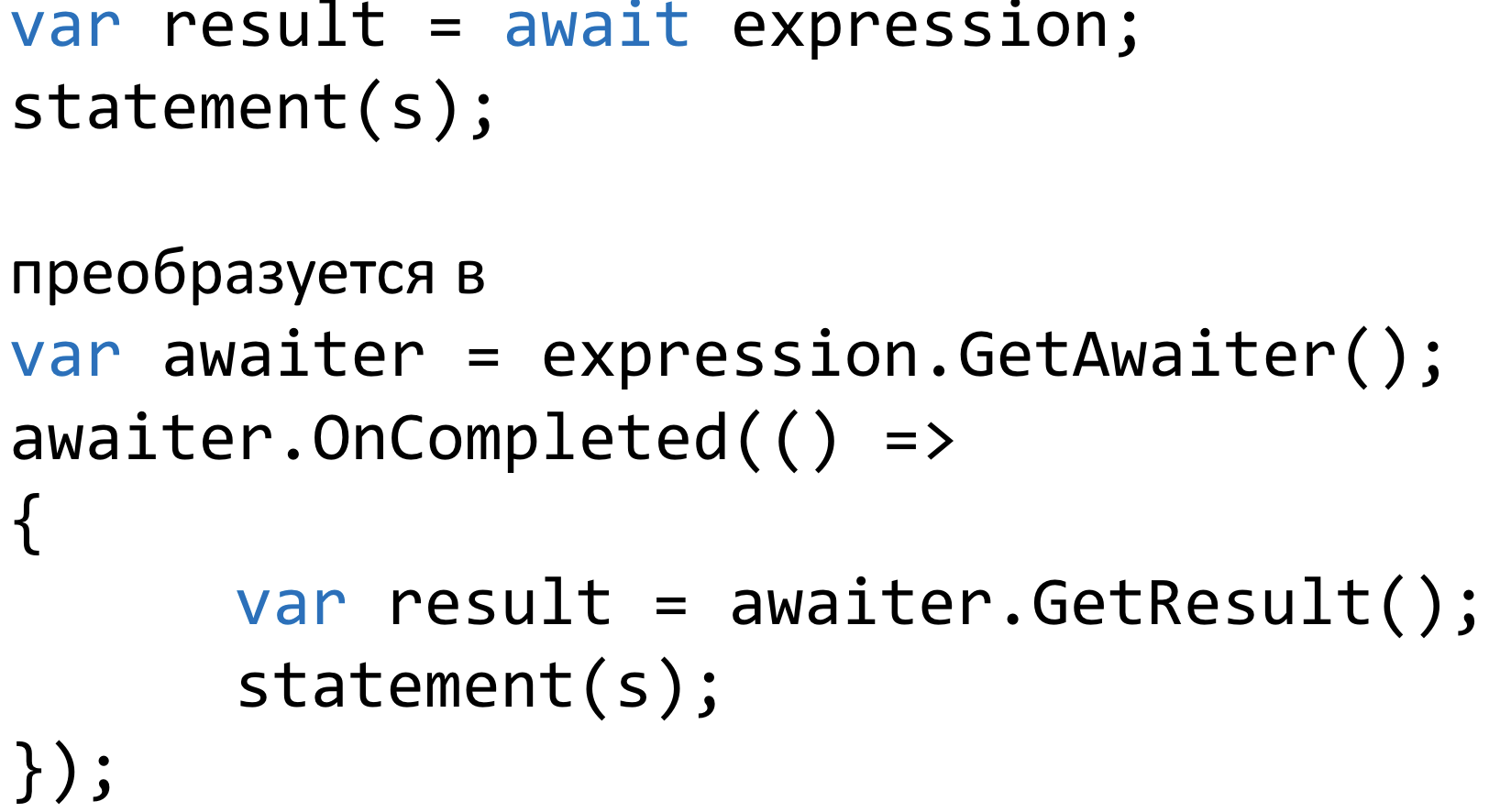
* Ещё сильнее упрощает работу с заданиями, но увеличивает overhead
* Содержит методы For, ForEach, Invoke



## Доступная асинхронность

* Асинхронные операции – ключ к созданию высокопроизводительных масштабируемых приложений, выполняющих операции на небольшом количестве потоков
* CLR предлагает доступную модель для реализации асинхронности  
  – основанную на задачах Task и асинхронных методах (TAP) – по сути, просто синтаксический сахар

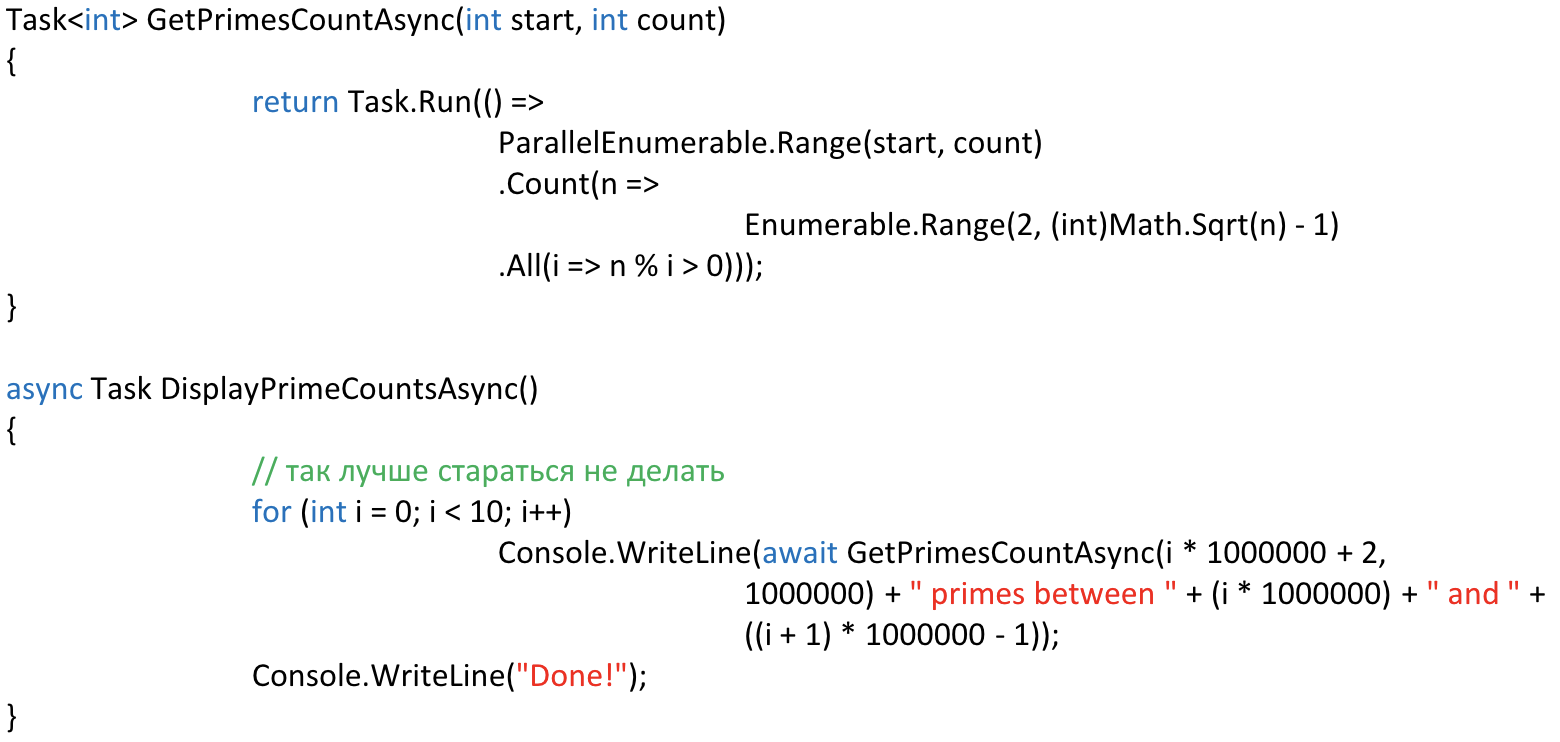
## Асинхронные методы



## Количество простых чисел

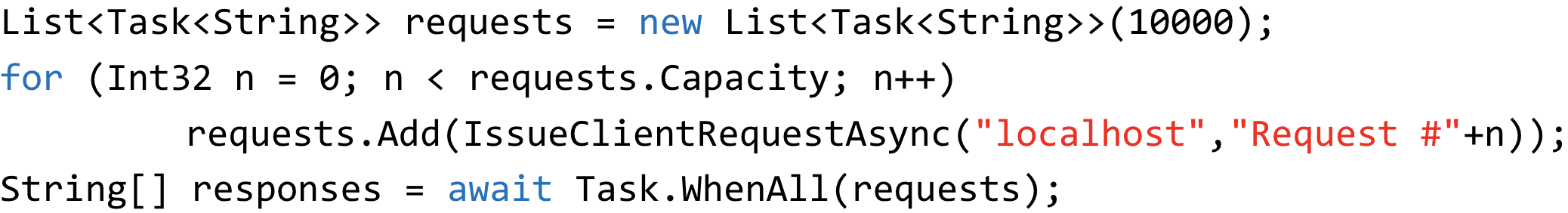


## Асинхронный вариант



## Так правильнее

• Можно группировать операции в коллекцию и вызывать await с продолжением по завершению всех или хотя бы одной операции



## Асинхронные методы

• async, await – синтаксический сахар, упрощающий реализацию асинхронного выполнения операций с помощью заданий(но всё же не совсем)

• Асинхронные методы – конечный автомат (на подобие автоматов последовательностей)