
OTUS
ОНЛАЙН-ОБРАЗОВАНИЕ
Онлайн-образование

Проверить, идет ли запись!

Меня хорошо видно && слышно?

Ставьте + , если все хорошо
Напишите в чат, если есть проблемы

Делегаты, Event-ы, добавляем асинхронное выполнение

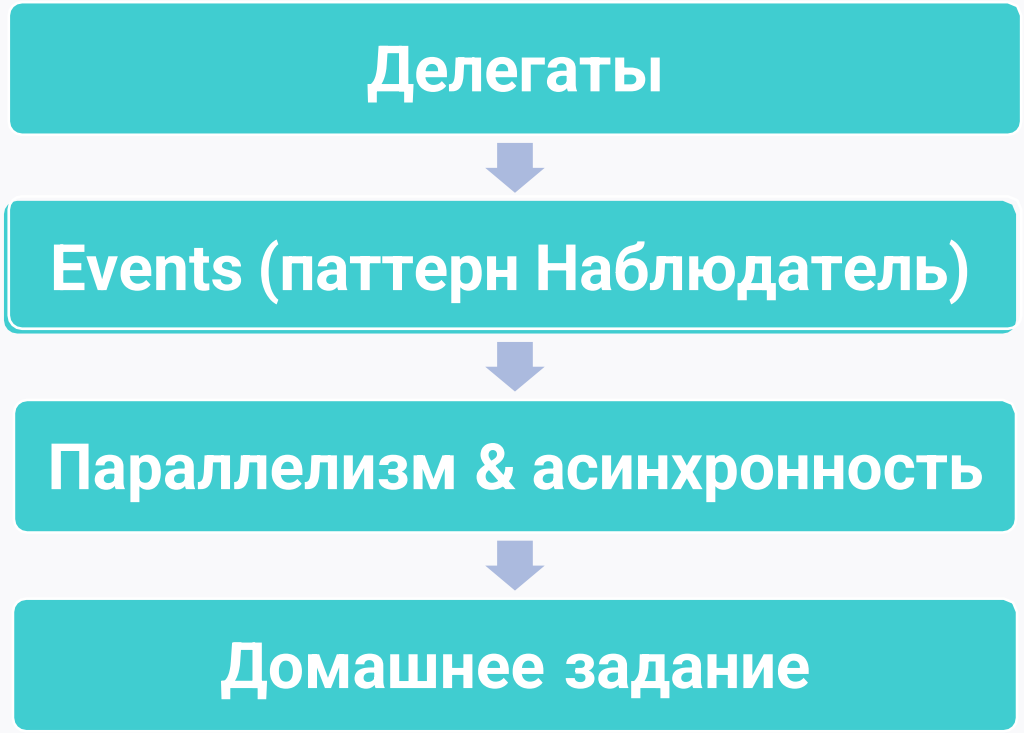
Виктор Дзицкий

TeamLead, Full Stack .Net Developer

SolarLab

Telegram: **@Dzitskiy**

Маршрут вебинара



Делегаты

Что такое делегат?

Что такое делегат?

Делегат это тип, который определяет сигнатуру используемой функции

Похож на указатель на функцию из C++

Делегаты

```
interface ICalculator
{
   T Sum(T a, T b);
   T Mul(T a, T b);
   T Sub(T a, T b);
   T Div(T a, T b);
}
```

Делегаты

```
interface ICalculator
{
   T Operation(T leftOperand,
               T rightOperand,
                function)
}
```

Делегаты

- Объект , ссылающийся на метод(ы)*
- Ссылка позволяет вызвать метод, абстрагируясь от конкретики
- Делегат может быть объявлен вне класса
- По-сути, делегат это подсказка о том, какая сигнатура (типы и количество параметров, тип возвращаемого значения) ожидается

Объявление:

```
delegate int Operation (int i, int j);
```

Использование:

```
void MyFunc(int a, int b, Operation op);
```

Events

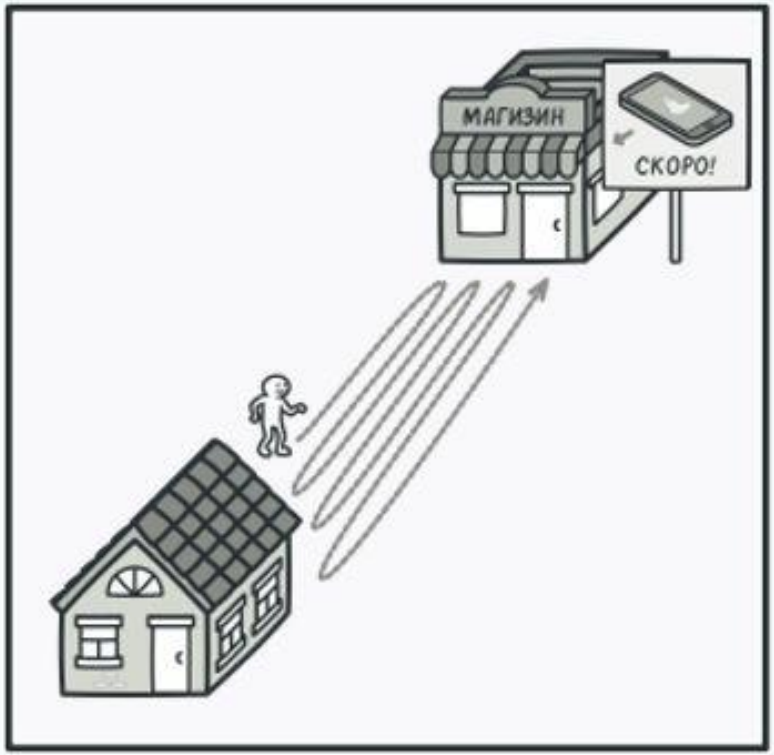
Паттерны (шаблоны) проектирования

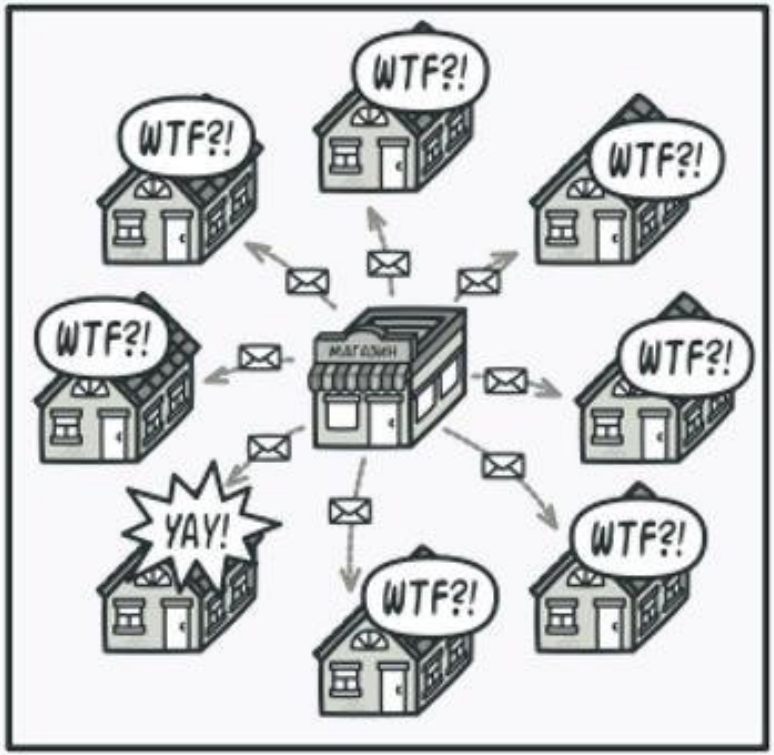
Шаблон проектирования или паттерн (англ. **design pattern**):

повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Поведенческие, структурные, порождающие.

Паттерн “Наблюдатель”





Паттерн “Наблюдатель”

Термины:

- Observer (subscriber) -наблюдатель
- Observable (publisher) -наблюдаемый

Плюсы:

- Помогает избежать лишних запросов (от наблюдателя)
- Помогает избежать лишних рассылок (от наблюдаемого)

Можно реализовать как с использованием Event, так и без него.

Событие (event)

Событие:

- Ситуация, требующая реакции
- Реакция может состоять из одного и более действий

Событие (в C#) -именованный делегат, запускающий в момент возникновения события все подписавшиеся на него методы.

```
public event <НазваниеДелегата> <Имя>;
```

Событие не может существовать вне класса (оно в этом случае бесполезно).

Параллелизм

Процессы и потоки

Операционная система -это менеджер ресурсов

Одна из основных задач ОС -распределение ресурсов между приложениями:

- Оперативная память
- Процессорное время
- I\O (сеть, HDD, шины, например PCI, USB)

Процессы и потоки

В операционной системе работают **процессы**

Процесс -загруженная в оперативную память программа, исполняемая в данный момент

Поток -наименьшая единица исполнения, которая может быть назначена на ядро ЦП

Процессы и потоки

Каждый процесс может иметь >= 1 потока

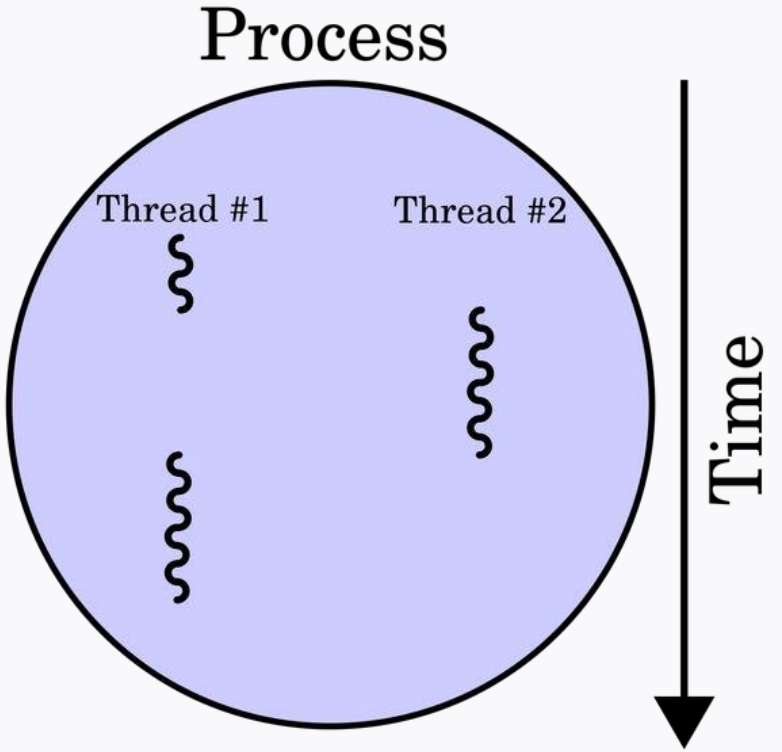
Минимально каждый процесс запускает хотя бы один поток исполнения (главный поток или Main thread)

Процессы и потоки

ОС распределяет ресурсы между потоками (многопоточное выполнение).

Чтобы все процессы в системе могли работать “параллельно” ОС должна давать им доступ к ядрам ЦП.

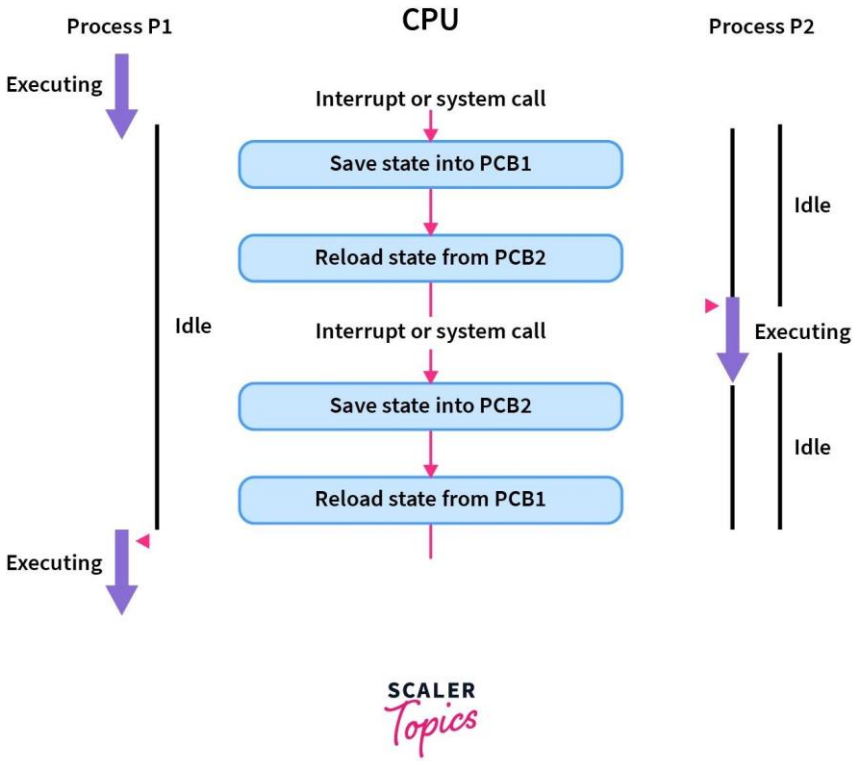
Потоков в ОС как правило больше чем ядер ЦП, следовательно, каждому потоку нужно давать время на выполнение работы.



Процессы и потоки

Переключение контекста (context switch) - смена исполнителя.

Дорогая операция, частая смена контекста = “тормоза”



Блокировка потока

Некоторые операции блокируют поток:

- I\O
- Работа с сетью
- Thread.Sleep

Пока поток заблокирован -он не выполняет никакой работы, ожидая завершения длительной операции.

Блокировка потока

Некоторые действия привязаны к одному потоку, например работать с UI (в WinForms) может только один поток.

Если в этом же потоке запустить “тяжелую” операцию -это приведет к блокировке окна и элементов управления.

Блокировка потока

Создание потока -небыстрая операция

Чаще всего используется пул потоков (ThreadPool) -некоторое количество заранее созданных потоков, которые динамически назначаются на выполнение “тяжелых” задач.

Блокировка потока

- Кол-во потоков <= кол-во ядер ЦП
- Если ЦП на 1 ядро -то уменьшения времени работы за счет разделения добиться сложнее (время на переключение контекста)
- На 2+ ядра -можно увеличить производительность работы программы

Асинхронность

Идея асинхронности заключается в том, что:

- При выполнении блокирующей операции не обязательно блокировать поток
- Можно выполнять ее в фоновом режиме
- В какой-то момент получить результат ее (операции) выполнения

Асинхронность (Task)

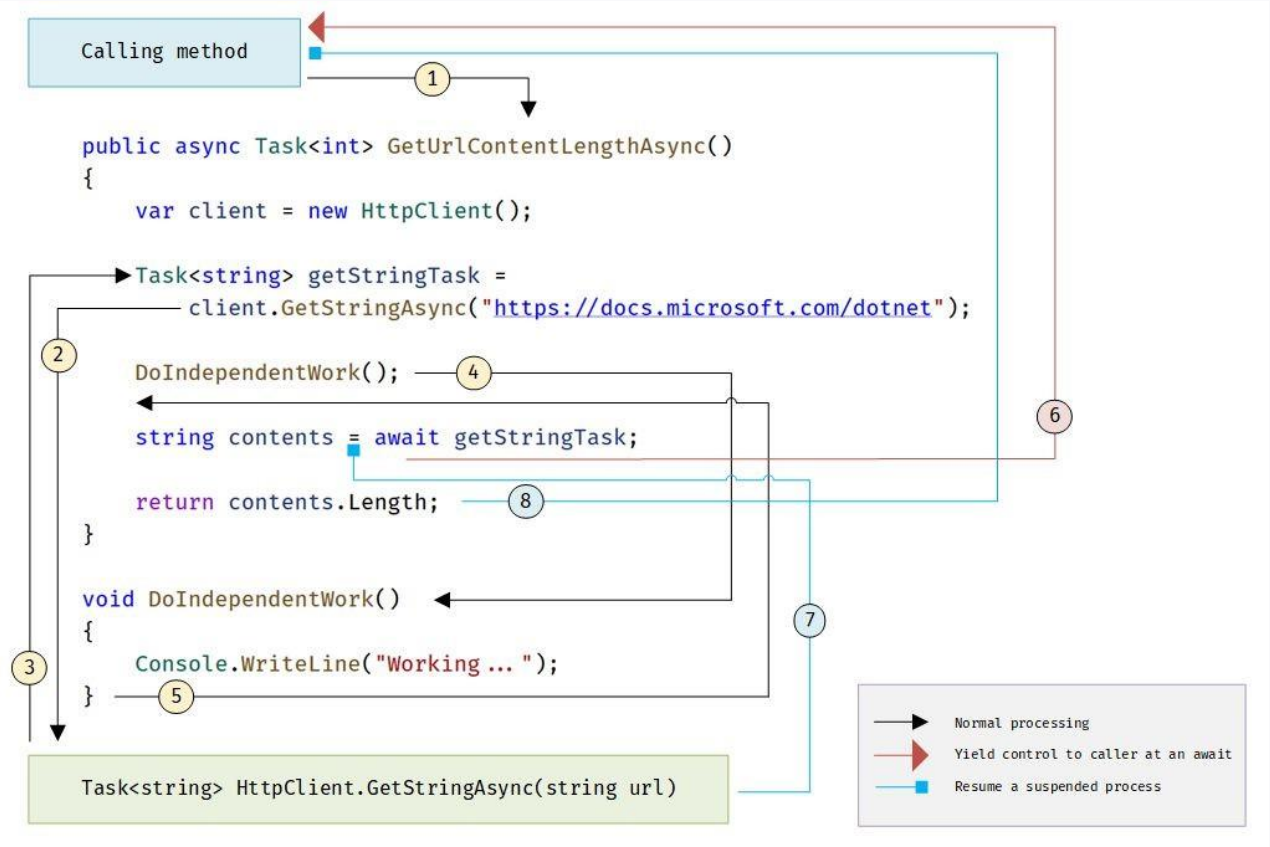
Концепция похожа на потоки, но реализуется на уровне CLR.

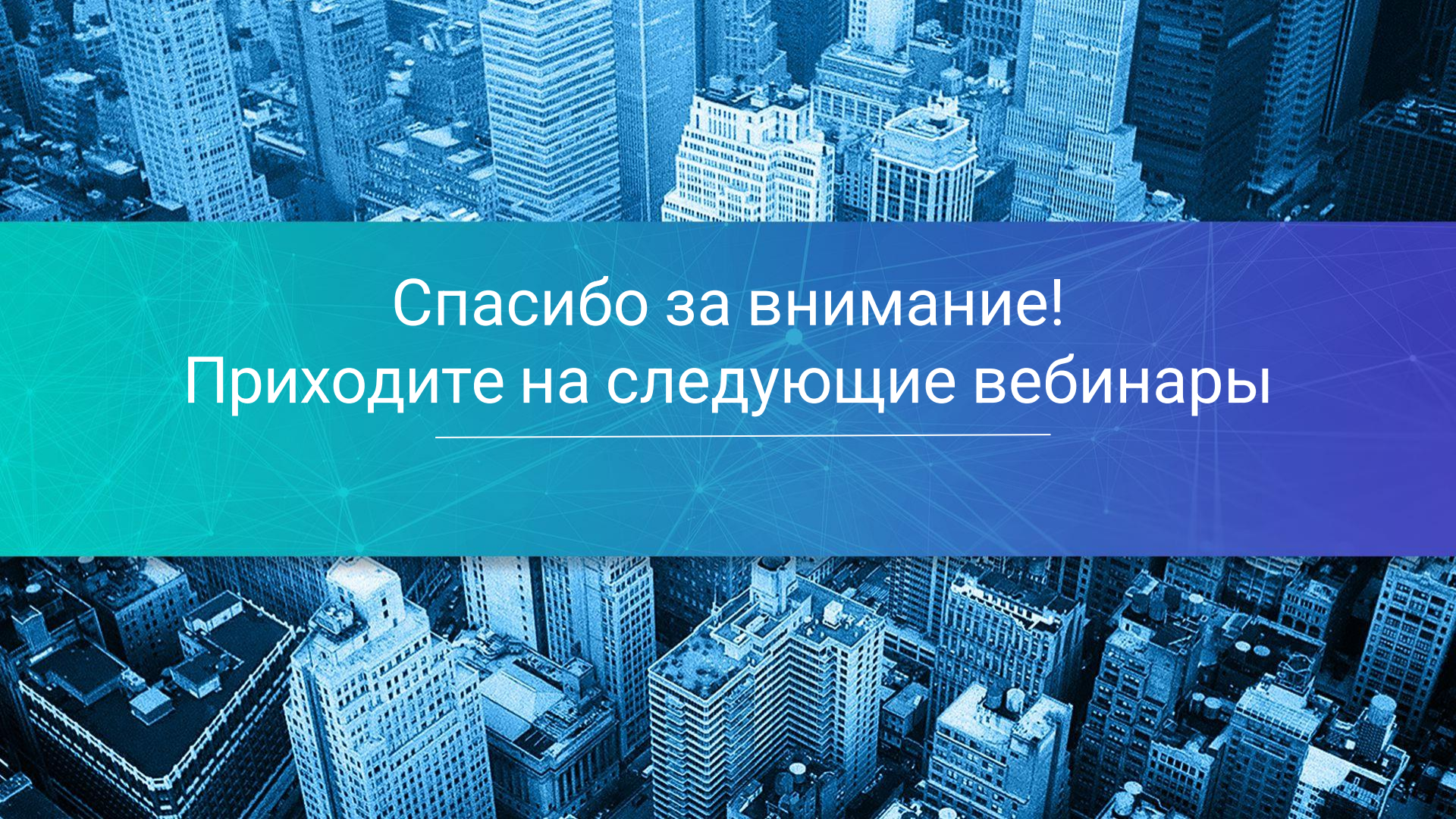
Task - задача, которая имеет возможность отмены, возврата результата.

Task != Thread

Но, Task для запуска использует ThreadPool

Асинхронность





Спасибо за внимание!
Приходите на следующие вебинары