ЛЕКЦИЯ 5

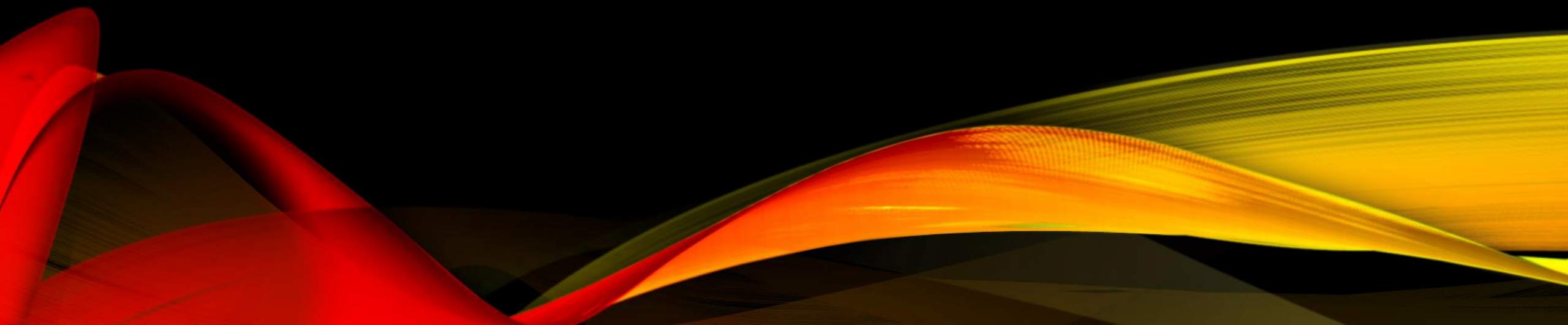
- Модуль 3
- 24.01.2024
- События

ЦЕЛИ ЛЕКЦИИ

- Разобраться с шаблоном проектирования «Наблюдатель»
- Изучить механизм событий (events) в C#

ШАБЛОН НАБЛЮДАТЕЛЬ

Observer



ШАБЛОНЫ ПРОЕКТИРОВАНИЯ

- **Шаблон / паттерн проектирования** [design pattern] – это общего вида решение задачи, которая часто возникает при программировании
- Паттерн проектирования именует, абстрагирует и идентифицирует ключевые аспекты структуры общего решения, которые и позволяют применить его для создания повторно используемого дизайна

ОПОВЕЩЕНИЕ ОБЪЕКТОВ О СОБЫТИЯХ

При написании программ возникает сценарий, когда одни объекты должны получать оповещения от других

При этом:

- Постоянный опрос объекта-издателя о новостях/обновлениях осуществлять неудобно
- Отправлять информацию всем потенциальным объектам-подписчикам – неподходящее решение
- Типы объектов-издателей и объектов-подписчиков должны быть независимы друг от друга

Необходим механизм, позволяющий оповещать о происходящем только те объекты, который действительно нуждаются в получении информации

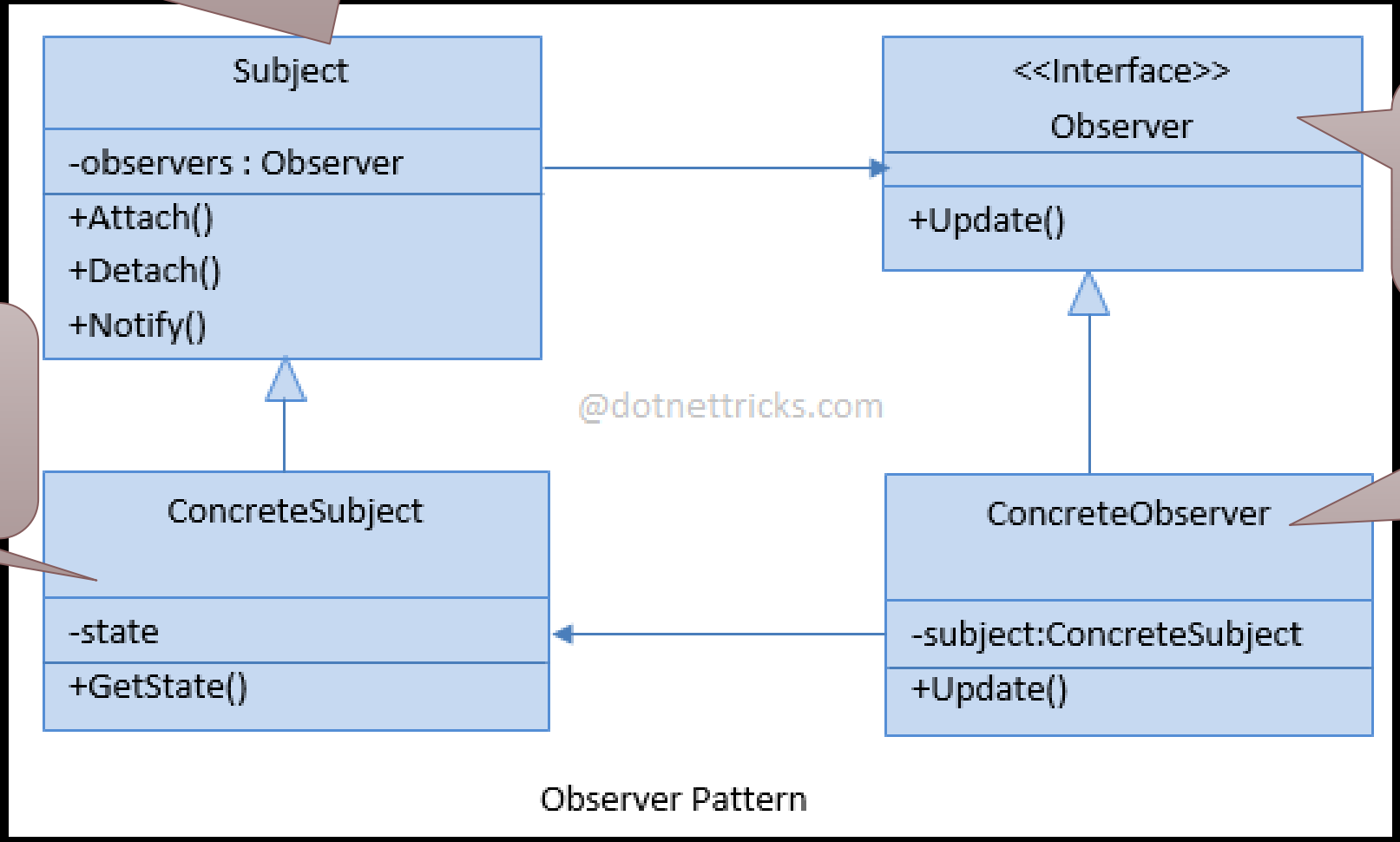
ШАБЛОН НАБЛЮДАТЕЛЬ

Определяет зависимость типа «один ко многим» между объектами так, чтобы при изменении состояния одного объекта все зависящие от него оповещались об этом и обновлялись автоматически

Мотивация: уменьшить связанность класса с его зависимостями путем уничтожения связи инициатора некоторого события с его обработчиками

ШАБЛОН НАБЛЮДАТЕЛЬ

Объект, за которым происходит наблюдение. Определяет методы подключения и отключения наблюдателей



Интерфейс наблюдателя

Конкретный тип наблюдаемого объекта

Реализует интерфейс наблюдателя

https://www.dotnettricks.com/learn/designpatterns/observer-design-pattern-c-sharp

ПРИМЕР РЕАЛИЗАЦИИ ШАБЛОНА НАБЛЮДАТЕЛЬ

Пример реализации «каноничного» шаблона Наблюдатель на C#.

Будем наблюдать за состоянием ConcreteSubject, состоящим из одной целочисленной переменной, при помощи наблюдателя ConcereteObserver, который настраивает своё состояние относительно состояния ConcreteSubject (остаток от деления на 10).

Код примера: https://replit.com/@olgamaksimenkova/ObserverExample

Такая реализация на C# избыточна и не принята, более традиционные способы посмотрим далее, данный пример используется только для демонстрации

РЕАЛИЗАЦИЯ ШАБЛОНА НАБЛЮДАТЕЛЬ

Средствами .NET шаблон Наблюдатель может быть реализован с помощью:

- Делегатов (через обратные вызовы)
- Событий (events)
- Специальных интерфейсов-наблюдателей
- Библиотечных интерфейсов IObserver / IObservable

РЕАЛИЗАЦИЯ ШАБЛОНА НАБЛЮДАТЕЛЬ В C#

Использование делегата (callback)

Использование события (event)

ТЕРМИНОЛОГИЯ

- Издатель (publisher) – тип, экземпляры которого генерируют события (raising an event). Позволяет другим объектам подписываться на рассылку событий и отписываться от неё
- Подписчик (subscriber) – тип, экземпляры которого способны подписываться на рассылку событий и обрабатывать связанные с ней данные
- Обработчик события (event handler) – метод обработки события. Как правило, используется объектом-подписчиком
- Данные события (event arguments) – информация, рассылаемая издателем всем подписчикам при возникновении события

ПОСТАНОВКА ЗАДАЧИ

Издатель

- Позволяет подписываться и отписываться на рассылку событий
  - Принимает циклически консольные команды до ввода команды «exit» (завершение работы). Если принята команда «notify», все подписчики получают текущее время в качестве информации

Подписчик

- Имеет собственное имя (строку)
- Подписывается на рассылку событий – имеет метод, соответствующий формату передаваемого сообщения
  - В момент получения сообщения от издателя обрабатывает его – выводит текст со своим именем и содержащиеся в сообщении данные

РЕШЕНИЕ С ИСПОЛЬЗОВАНИЕ ДЕЛЕГАТОВ: ПОЧЕМУ

Т.к. для передаваемое событие содержит данные определённого формата, а оповещение сводится к вызову реакции на событие со стороны подписчиков, подходящим решением задачи могут быть делегат-типы:

- Сигнатура делегат-типа задаёт контракт передаваемых данных между издателем и подписчиком
- Возможность делегатов хранить ссылки на несколько методов позволяет сохранять всех подписчиков в одном списке вызовов

ИСПОЛЬЗОВАНИЕ ДЕЛЕГАТОВ - ИЗДАТЕЛЬ

```
public class Publisher
{
    public Action<DateTime> notificationAppeared;

    public void HandleCommands()
    {
        string command;
        do
        {
            command = Console.ReadLine();
            if (command == "notify" && notificationAppeared != null)
            {
                notificationAppeared(DateTime.Now);
            }
        } while (command != "exit");
    }
}
```

1) Делегат, на который подписываются подписчики.

2) Циклическая обработка команд – при вводе notify всем подписчикам (если такие есть) будет разослано время возникновения события.

ИСПОЛЬЗОВАНИЕ ДЕЛЕГАТОВ - ПОДПИСЧИК

1) Имя подписчика.

```
public class Subscriber
{
    public string Name { get; init; }

    public Subscriber(string name) => Name = name;

    public void NotificationEventHandler(DateTime eventArgs)
        => Console.WriteLine($"{Name}: received notification " +
                             $"on {eventArgs.ToShortDateString()}");
}
```

2) Метод соответствующий сигнатуре события издателя для обработки сообщений.

ИСПОЛЬЗОВАНИЕ ДЕЛЕГАТОВ - ОСНОВНАЯ ПРОГРАММА

```
class Program
{
    static void Main()
    {
        Publisher publisher = new Publisher();
        Subscriber subscriber = new Subscriber("Alex");
        publisher.notificationAppeared += subscriber.NotificationEventHandler;
        publisher.HandleCommands();
        // Код на строке ниже не скомпилируется для события:
        // publisher.notificationAppeared = null;
    }
}
```

Подписка объекта-подписчика на издателя осуществляется аналогичным образом.

Полный код примера: https://replit.com/@olgamaksimenkova/ObserverImplDelegate

РЕШЕНИЕ С ИСПОЛЬЗОВАНИЕ ДЕЛЕГАТОВ: КОГДА

Рекомендуется использовать когда:

- Наблюдатель должен присутствовать обязательно
- Наблюдаемый объект уведомляет наблюдатель и ожидает некоторого результата

Не рекомендуется использовать когда:

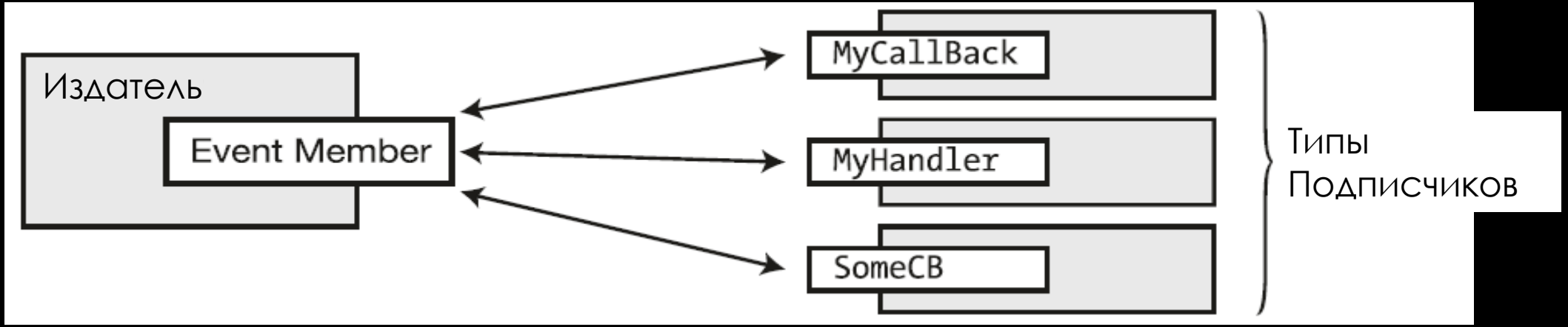
- При росте числа делегатов
  - Рекомендуется выделять именованную зависимость или интерфейс наблюдателя
- Есть повторно используемые компоненты
  - Рекомендуется пользоваться событиями
- При передаче потока событий через делегат
  - Им проще управлять через IObservable

ПРОБЛЕМЫ РЕШЕНИЯ С ИСПОЛЬЗОВАНИЕМ ДЕЛЕГАТОВ

- Вызов делегата издателя может быть осуществлён произвольным кодом без ограничений
- Подписчики могут напрямую заменять список вызовов делегата через операцию присваивания (=)
- Произвольный код может полностью очистить список подписчиков, присвоив делегату null

Требуется инкапсулировать (защитить) делегат так, чтобы можно было контролировать доступ к нему

СХЕМА ВЗАИМОДЕЙСТВИЯ ИЗДАТЕЛЯ И ПОДПИСЧИКОВ

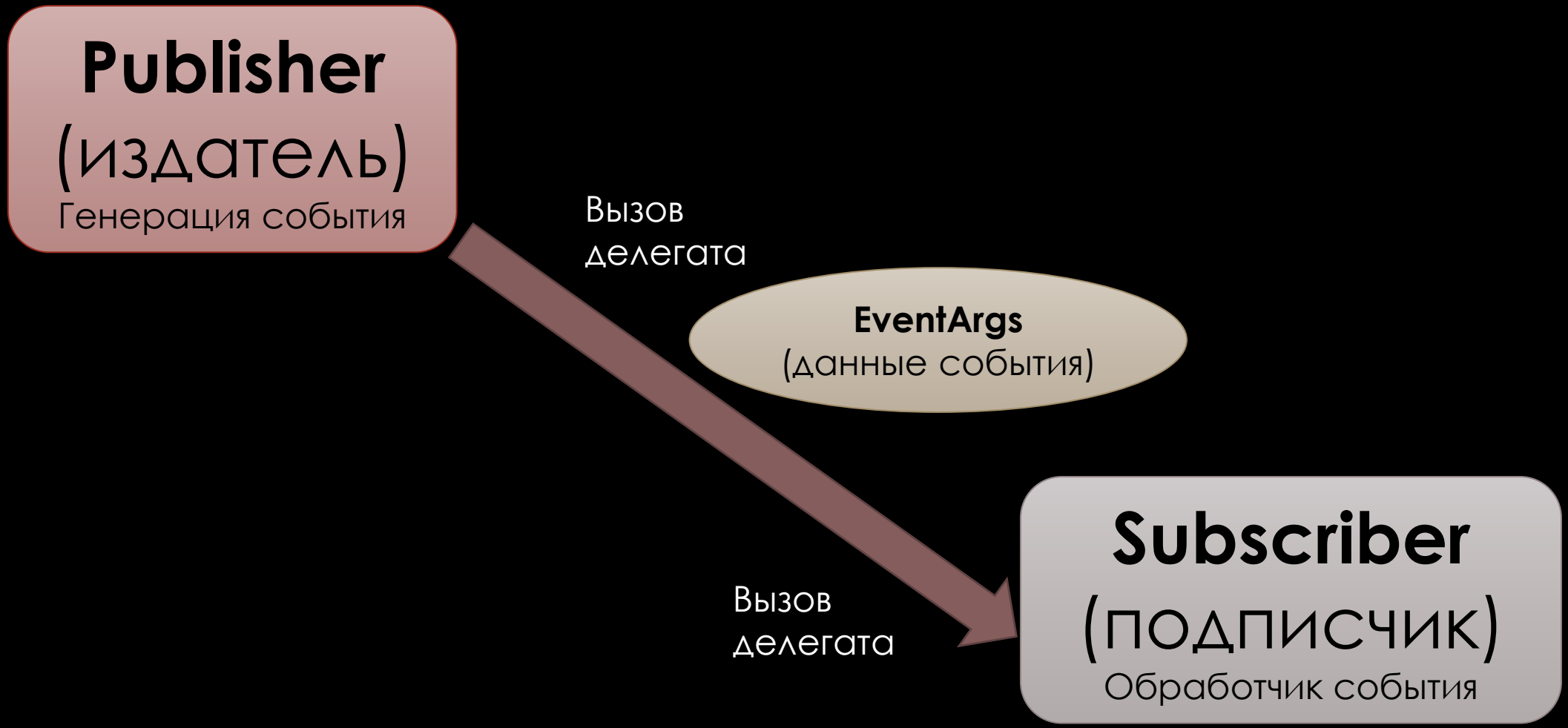


Издатель определяет событие (как член класса)

Подписчики определяют методы-обработчики, которые должны быть вызваны при возникновении события, и подписывает их на событие издателя

Издатель инициирует возникновение события и, как следствие, вызываются все обработчики подписчиков события

УПРОЩЁННАЯ СХЕМА ЗАДАЧИ



КЛЮЧЕВОЕ СЛОВО EVENT

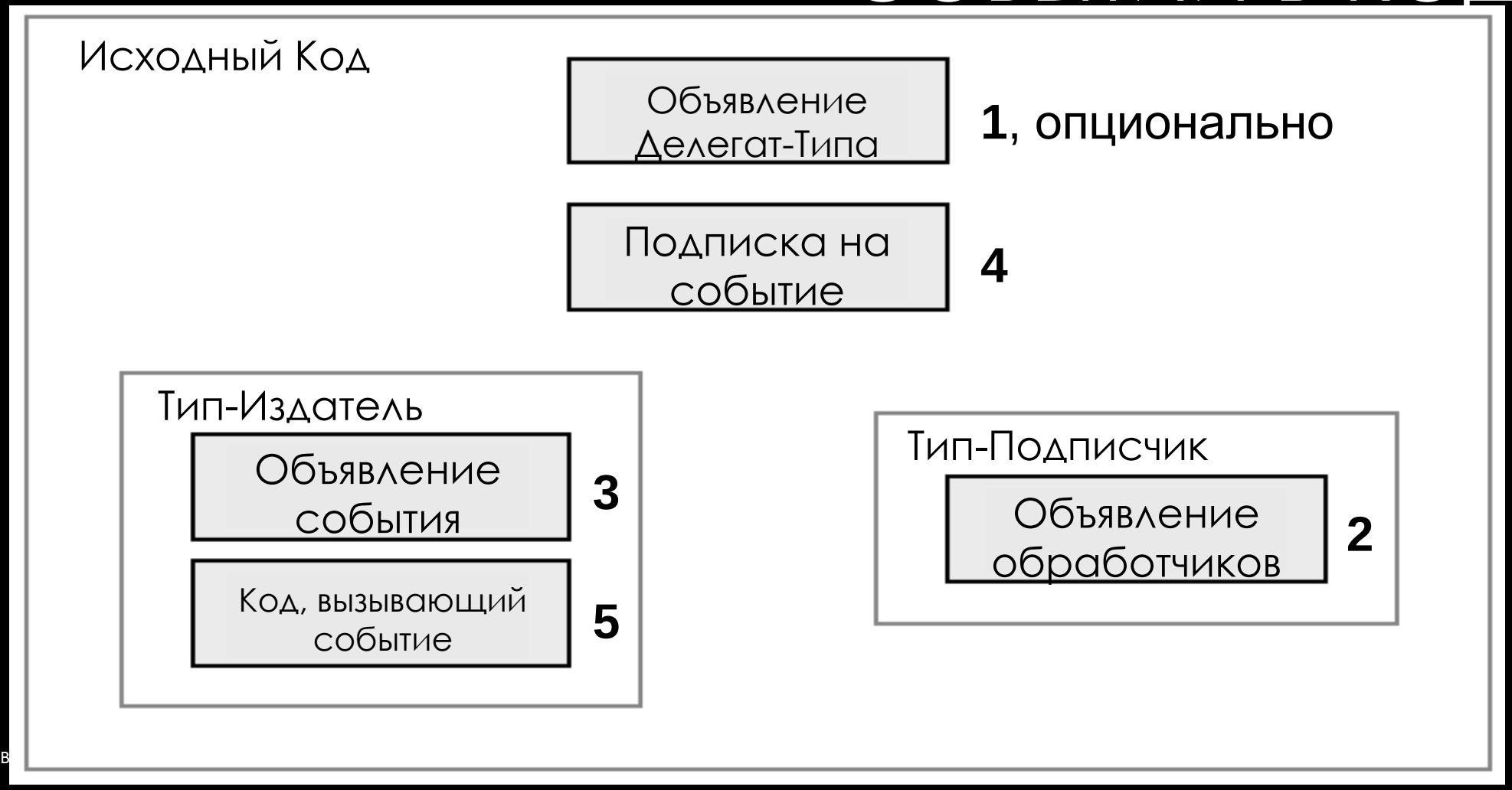
Для упрощения реализации сценариев обработки событий было добавлено ключевое слово `event`, которое позволяет объявить инкапсулированное поле делегат-типа с использованием следующего синтаксиса:

- `[Модификаторы] event <Делегат-Тип> <Идентификатор>;`

Объявление поля-события вводит следующие ограничения:

- события могут быть вызваны только внутри того типа, в котором они объявлены (у наследников доступа тоже нет!)
- для подписчиков извне доступны только операции подписки/отписки (+= и -=)

СХЕМА: ИСПОЛЬЗОВАНИЕ СОБЫТИЙ В КОДЕ



РЕШЕНИЕ С СОБЫТИЕМ: ИЗДАТЕЛЬ

```
public class Publisher
{
    public event Action<DateTime> notificationAppeared;

    public void HandleCommands()
    {
        string command;
        do
        {
            command = Console.ReadLine();
            if (command == "notify" && notificationAppeared != null)
            {
                notificationAppeared(DateTime.Now);
            }
        } while (command != "exit");
    }
}
```

Поле делегат типа теперь объявляется как событие, что добавляет необходимую инкапсуляцию.

Публикация события синтаксически не отличается от вызова делегата

РЕШЕНИЕ С СОБЫТИЕМ: ПОДПИСЧИК

```
public class Subscriber
{
    public string Name { get; init; }

    public Subscriber(string name) => Name = name;

    public void NotificationEventHandler(DateTime eventArgs)
        => Console.WriteLine($"{Name}: received notification " +
                             $"on {eventArgs.ToShortDateString()}");
}
```

Метод-обработчик с той же сигнатурой, т. к. событие основывается на том же делегат-типе

РЕШЕНИЕ С СОБЫТИЕМ ОСНОВНАЯ ПРОГРАММА

```
class Program
{
    static void Main()
    {
        Publisher publisher = new Publisher();
        Subscriber subscriber = new Subscriber("Alex");
        publisher.notificationAppeared += subscriber.NotificationEventHandler;
        // Метод-обработчик подписчика вызывается на каждый ввод строки "notify".
        publisher.HandleCommands();
    }
}
```

Подписка объекта-подписчика на издателя.

РЕШЕНИЕ С ИСПОЛЬЗОВАНИЕМ СОБЫТИЙ: КОГДА

Рекомендуется использовать когда:

- Для повторно используемых компонентов
- При необходимости информировать большое количество наблюдателей, от которых не ожидаются ответные действия
- При реализации pull-модели получения данных наблюдателем

Не рекомендуется использовать когда:

- Наблюдаемому объект нужно получать от наблюдателей некоторый результат

Pull-модель взаимодействия позволяет передавать наблюдателям минимальный объем данных и возлагает на них решение о том, что им требуется для обработки события

СХЕМА: СОБЫТИЕ КАК ИНКАПСУЛИРОВАННЫЙ ДЕЛЕГАТ

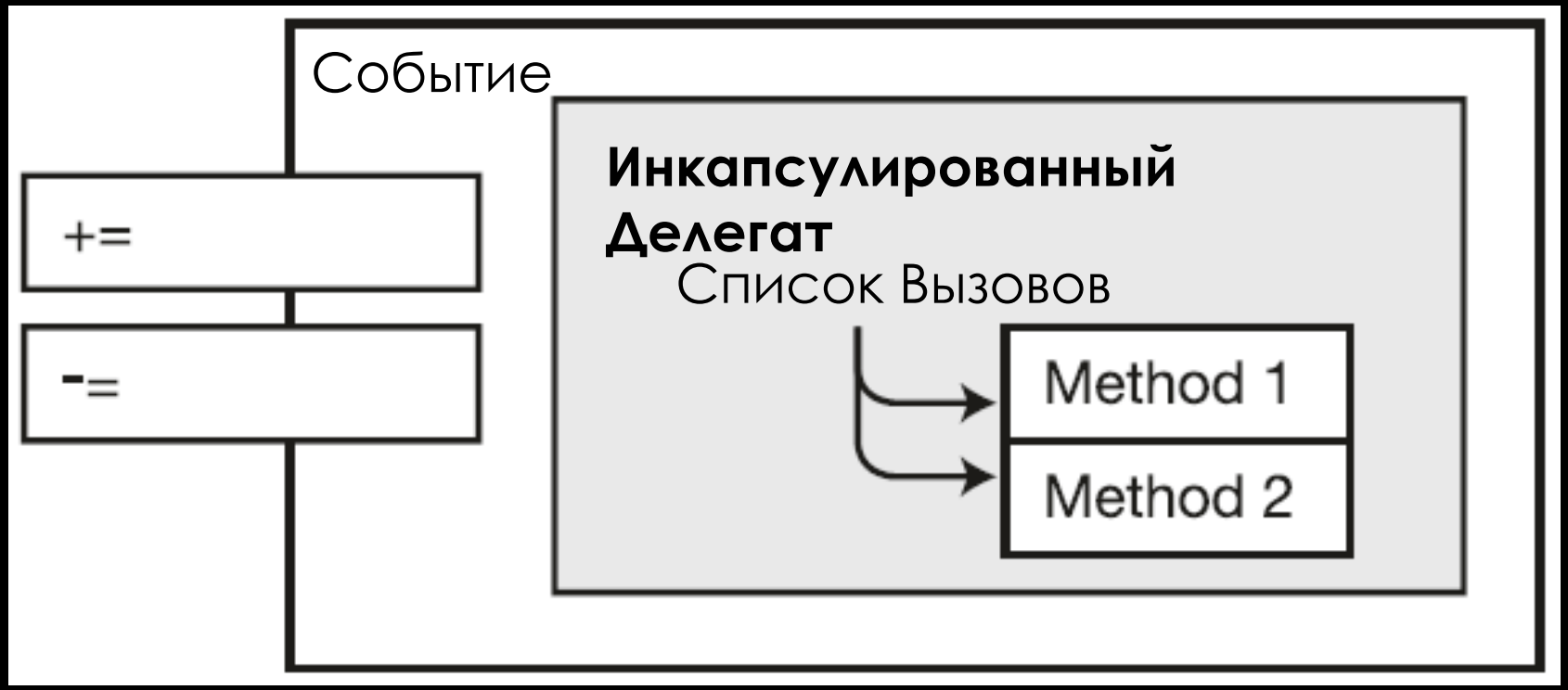
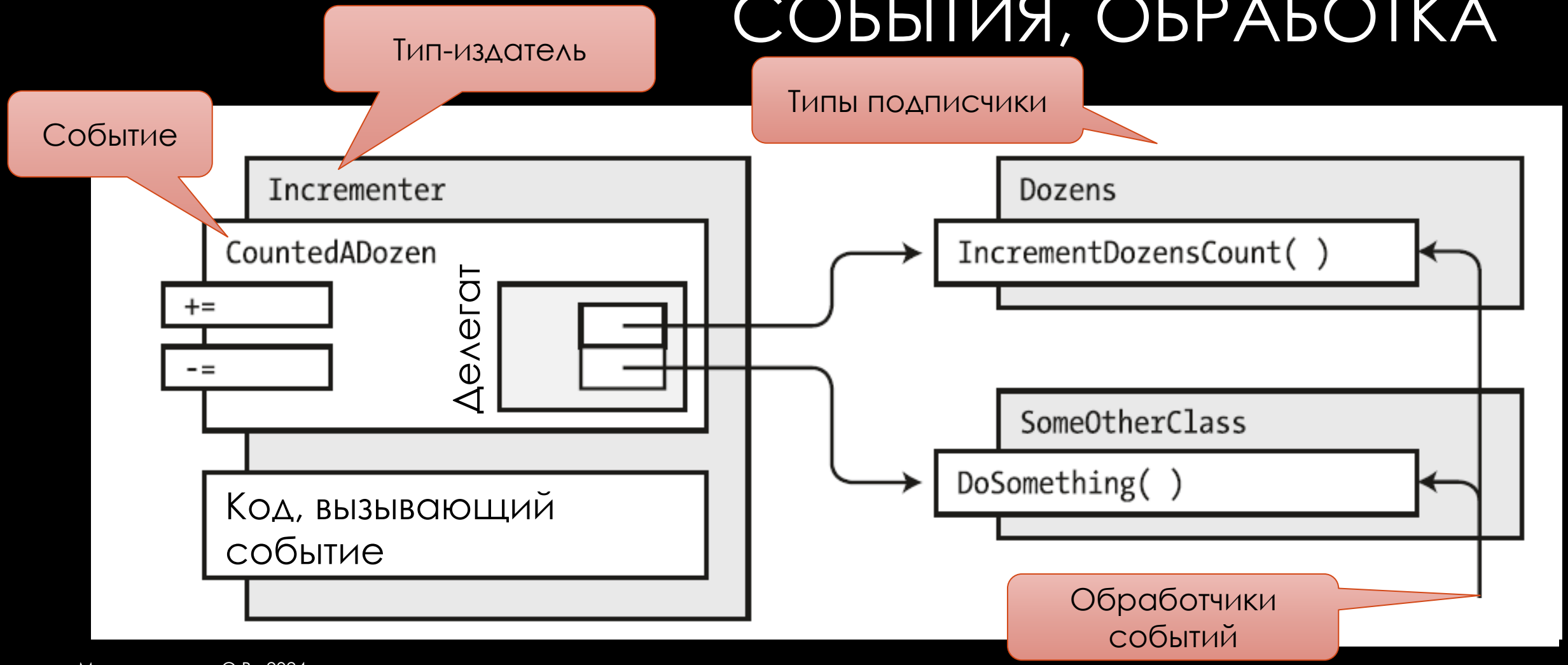


СХЕМА: ВОЗНИКНОВЕНИЯ СОБЫТИЯ, ОБРАБОТКА



ССЫЛКИ С ИСТОЧНИКАМИ ПО СОБЫТИЯМ

- Тепляков С. Паттерны проектирования на платформе .NET. – СПб.: Питер, 2015. – 320 с.
- https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/event
- https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/events/
- Обзорная информация по событиям: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/events/
- Ключевое слово event, допустимые модификаторы:
  https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/event
- Подписка на события; методы доступа add и remove, их явная реализация:
  https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/events/how-to-subscribe-to-and-unsubscribe-from-events
  https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/add
  https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/remove
  https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/events/how-to-implement-custom-event-accessors
- Стандартный шаблон событий .NET:
- https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/events/how-to-publish-events-that-conform-to-net-framework-guidelines
- Вызов событий базового типа из типов наследников:
  https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/events/how-to-raise-base-class-events-in-derived-classes
- Проблема использования виртуальных событий: https://pvs-studio.com/en/blog/posts/csharp/0453/
- Реализация интерфейсов с событиями:
  https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/events/how-to-implement-interface-events