IT's TINKOFF

# Пишем код когда пишем код: Source generator’ы

Олег Сафонов

# План

## 01 Проблематика

- Контекст
- Постановка задачи

## 02 Реализация

- Варианты решения
- Сравнение подходов

## 03 Source Generator

- Проблемы и решения
- Интересные ошибки

## 04 Выводы

- Когда следует использовать
- Преимущества

# Олег Сафонов

Ведущий разработчик Tinkoff

**12 лет**

Опыт

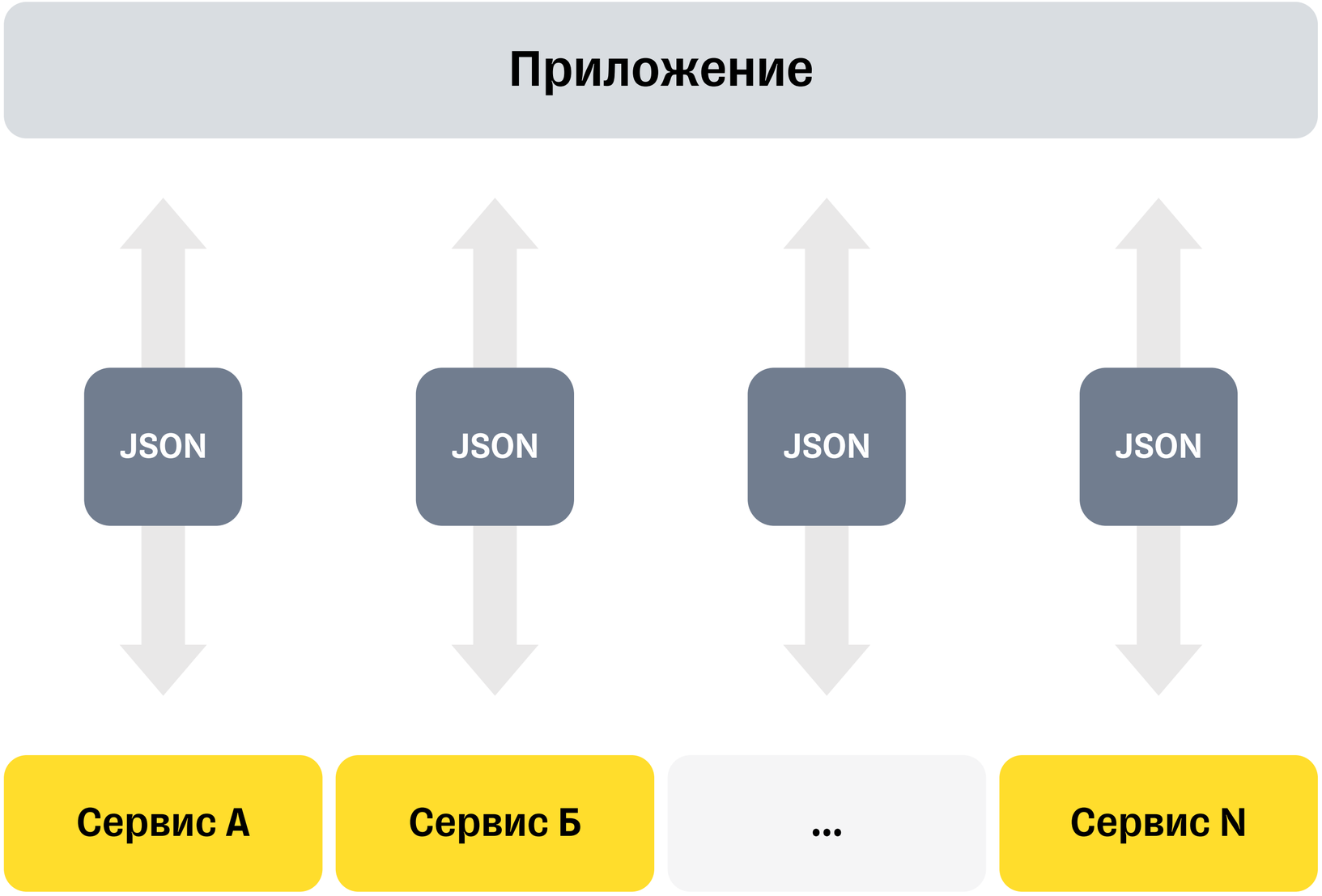
Профильное образование MSIT-SE

IT's TINKOFF

# Проблематика

- Контекст
- Постановка задачи

# Предыстория



# Решение

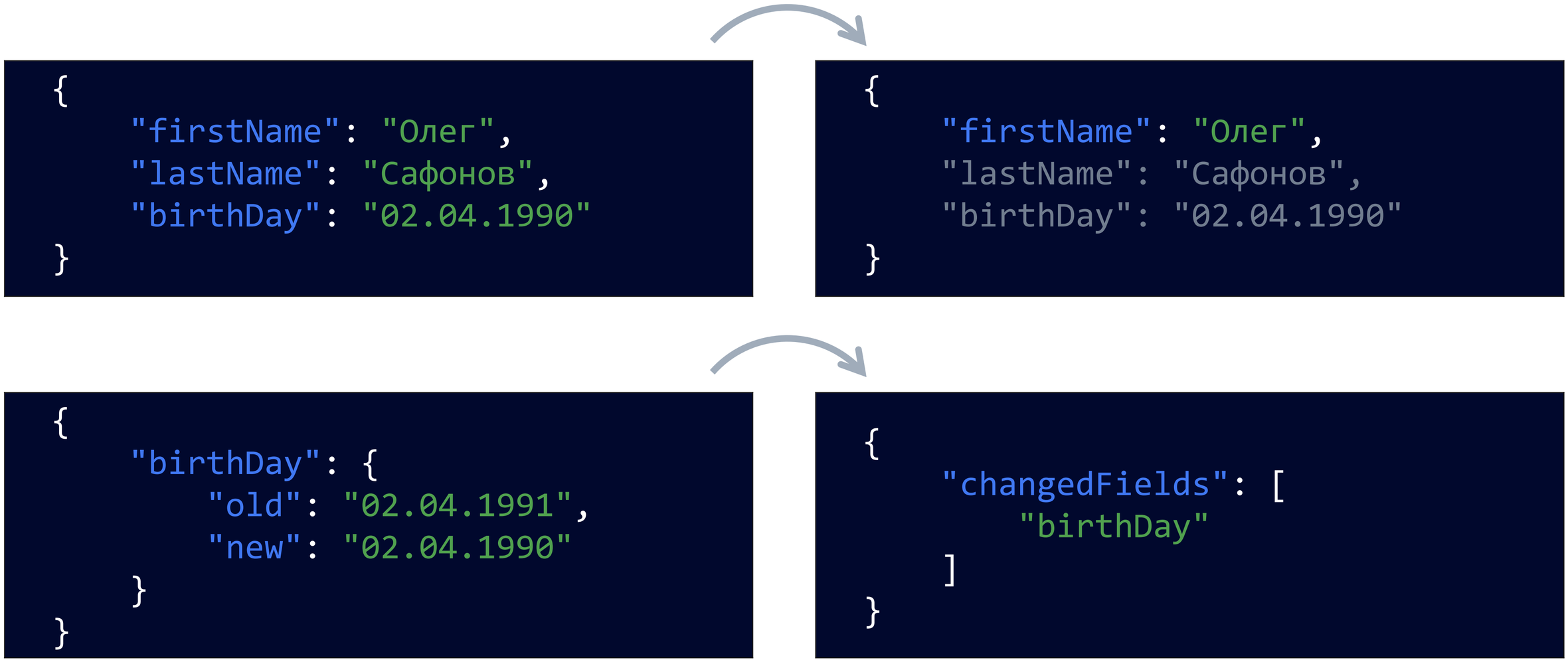
01 Логируем данные перед отправкой

02 Отправляем данные во внешнюю систему

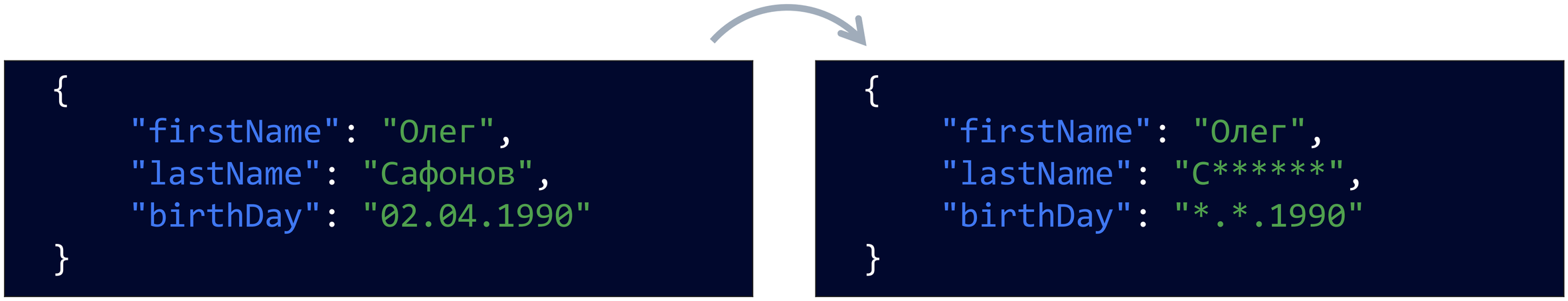
03 Логируем то, что получили в ответ

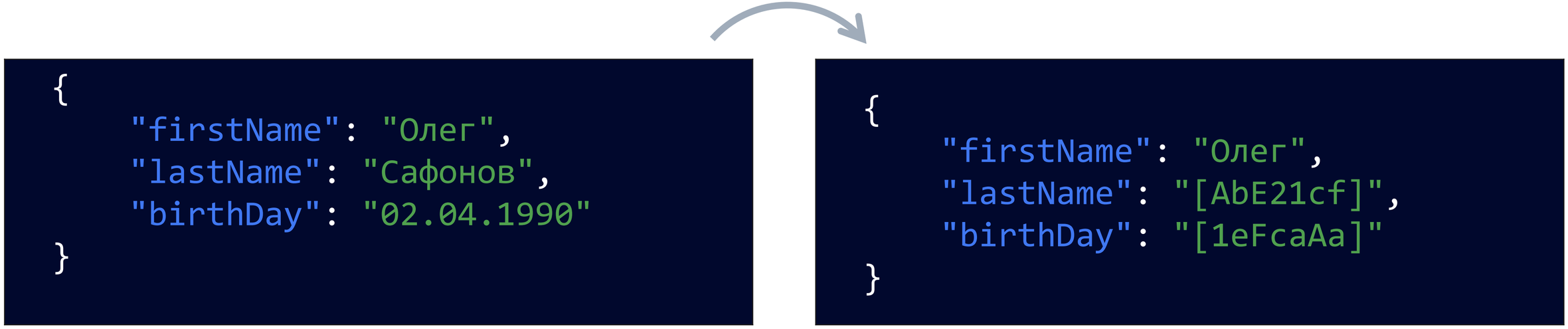
04 Логируем разницу между тем, что отправили и тем, что получили в ответ

# Ограничения — чувствительные данные



# Маскирование



# Цель

```
public interface Contact
{
    MaskedDifference Compare<T>(T first, T second);

    MaskedResult Mask<T>(T input);
}
```

# Требования

## Функциональные

- Логирование с маскированием чувствительных данных
- Сравнение сущностей с результатом в маскированном виде

## Нефункциональные

- Гарантия, что ничего лишнего не попадёт в лог при изменении структуры
- Поведение по умолчанию для разных типов
- Удобная настройка
- Минимум правок при изменении структуры
- Ошибки компиляции при неправильной настройке
- Безопасность (type safety)
- Производительность

IT's TINKOFF

# Реализация

- Варианты решения
- Сравнение подходов

# Рефлексия

# Варианты

### Впилиться в существующую библиотеку

- ❌ Впилиться после сравнения
- ❌ Впилиться в процесс сравнения
- ❌ Впилиться до сравнения

### Впилиться в логи

- ❌ Использовать пути в json
- ❌ Использовать регулярки\префиксы\суффиксы

### Написать свою библиотеку

- ❌ Форкнуть существующую
- ✅ Написать свою

# Пример

```
public MaskedDifference Compare<T>(T first, T second) => GetDiff(first, second);

private MaskedDifference GetDiff(object first, object second)
{
    // if (first.GetType() != second.GetType())
    foreach (var property in input.GetType().GetProperties())
    {
        var maskStrategy = GetMaskStrategy(property);
        result[property.Name] = Compare(property, first, second, maskStrategy);
    }

    return result;
}
```

# Простой код

```
private Difference Compare<T>(IEnumerable<T> first, IEnumerable<T> second);

private Difference Compare<T>(T? first, T? second);

private Difference Compare(string? first, string? second);
private Difference Compare(int? first, int? second);
private Difference Compare(bool? first, bool? second);
// ...
private Difference Compare(byte? first, byte? second);
```

# Удобная настройка

```
public class Contact
{
    [MaskFirstName]
    public string FirstName { get; set; }

    [MaskLastName]
    public string LastName { get; set; }

    [MaskBirthDay]
    public DateOnly BirthDay { get; set; }
}
```

# Итого

## Плюсы

- Мало кода
- Универсальный метод
- Удобная настройка атрибутами
- Минимум правок при изменении структуры
- Поведение по умолчанию для разных типов

## Минусы

- Нельзя посмотреть итоговый код
- Производительность (можно компилировать выражение)
- На этапе компиляции нельзя проверить корректность настроек (можно Debug.Assert)
- Ошибки при выполнении

# Пишем код руками

# Пример

```
public MaskedDifference Compare(Contact? first, Contact? second)
{
    var result = new MaskedDifference();

    if (first?.LastName != second?.LastName)
    {
        result[nameof(Contact.LastName)] = new Difference(
            MaskLastName(first?.LastName),
            MaskLastName(second?.LastName)
        );
    }
    // Сравнение других свойств
    return result;
}
```

# Пример

```
public MaskedDifference Compare(Contact? first, Contact? second)
{
    var result = new MaskedDifference();

    if (first?.Id != second?.Id)
    {
        result[nameof(Contact.Id)] = new Difference(first?.Id, second?.Id);
    }

    // Сравнение других свойств
    return result;
}
```

# Итого

## Плюсы

- Можно посмотреть код
- Производительность
- На этапе компиляции можно проверить корректность настроек
- Больше проверок на этапе компиляции

## Минусы

- На каждую сущность требуется свой метод
- Трудозатраты
- Неудобная настройка
- При изменении структуры никакие новые свойства\сущности не попадут в лог
- Единственное, что можно пойти не так - что то не замаскируется

# Генерация кода

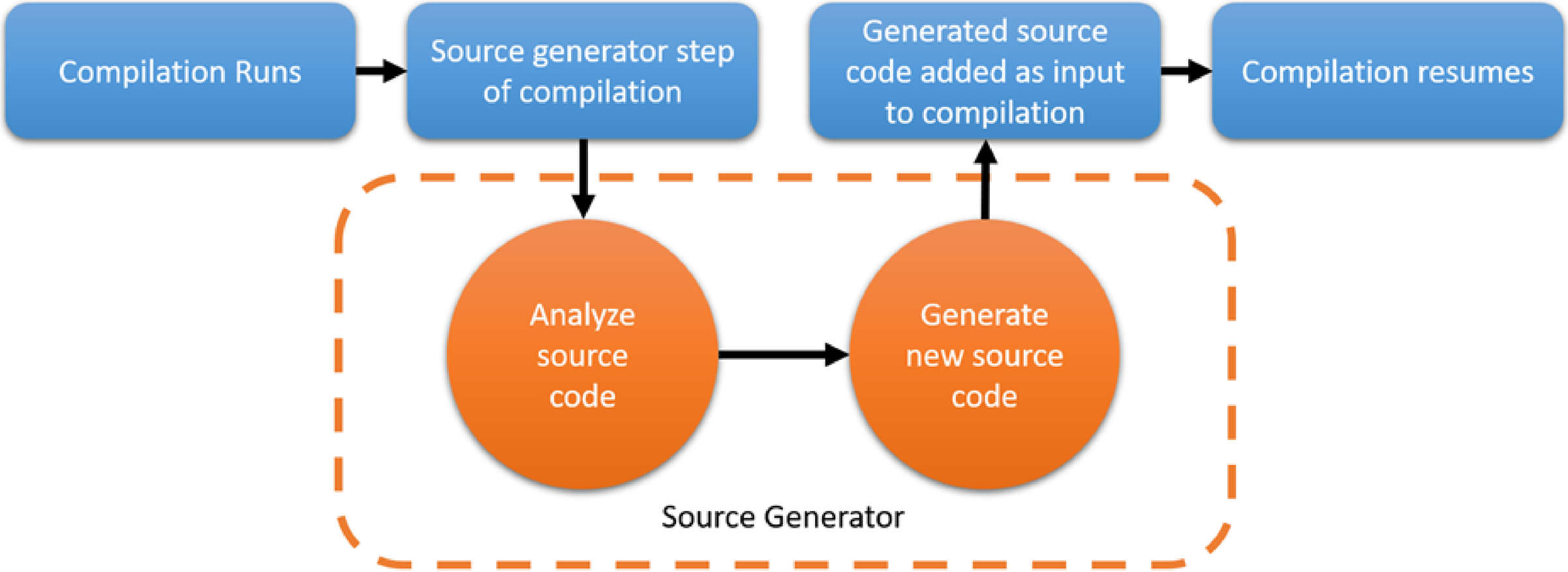
# Варианты

## T4 шаблоны

- ❌ Запускать руками
- ❌ Интегрировать в CI\CD

## Использовать Source Generator

- ✔ Генерировать код при билде






https://learn.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/source-generators-overview

# Генератор – пример кода

```
            sourceCode.Append(@$”
public partial class {comparableClass.Name}Comparer
{
    public MaskedDifference Compare({comparableClass.Name} first, {comparableClass.Name} second)
    {
        var result = new MaskedDifference();“;

        foreach (var property in GetProperties(comparableClass))
        {
            var maskStrategy = GetMaskStrategy(property);”);

            sourceCode.Append(@$”
        if (first.{property.Name} != second.{property.Name})
        {
            result[property.Name] = new Difference(
                        Mask(first.{property.Name}, maskStrategy),
                        Mask(second.{property.Name}, maskStrategy)
                    );
        }”);
    }

            sourceCode.Append(@$”
        return result;
    }
}”);
```

# Генератор – пример кода

```
[GeneratedComparer(typeof(Contact))]
public partial class ContactComparer
{
}

public partial class ContactComparer
{
    public MaskedDifference Compare(Contact? first, Contact? second)
    {
        // if (first is null || second is null) ...
        var result = new MaskedDifference();

        if (first.LastName != second.LastName)
        {
            result[nameof(Contact.LastName)] = new Difference(
                MaskLastName(first.LastName),
                MaskLastName(second.LastName)
            );
        }

        // Сравнение других свойств

        return result;
    }
}
```

# Итого

## Плюсы

- Универсальный метод - пишем алгоритм
- Можно посмотреть код
- Удобная настройка
- Производительность
- На этапе компиляции можно проверить корректность настроек
- Ошибки в compile time
- Минимум правок при изменении структуры

## Минусы

# Сравнение подходов

| | Рефлексия | Код | Source Generator |
|---|---|---|---|
| Удобство настройки - работа с атрибутами | ✓ | ✗ | ✓ |
| Автоматическое обновление кода | ✓ | ✗ | ✓ |
| Compile time проверки | ✗ | ✗ | ✓ |
| Безопасность (с точки зрения проверок, типов и т.д.) | ✗ | ✓ | ✓ |
| Производительность | ✗ | ✓ | ✓ |
| Доступ к коду | ✗ | ✓ | ✓ |
| Подверженность ошибкам | ✓ | ✗ | ✓ |

IT's TINKOFF

# Выбранное решение

- Проблемы и решения
- Интересные ошибки

# Проблемы и решения

## Первый запуск

# Проблемы и решения

## Первый запуск

- [x] Найти человека, имеющего опыт с source generator’ами

# Проблемы и решения

## Первый запуск

- [x] Найти человека, имеющего опыт с source generator’ами

## Поддержка IDE

# Проблемы и решения

## Первый запуск

- [x] Найти человека, имеющего опыт с source generator’ами

## Поддержка IDE

- [x] Отладить разработку

# Проблемы и решения

## Первый запуск

- [x] Найти человека, имеющего опыт с source generator’ами

## Поддержка IDE

- [x] Отладить разработку

## Тестирование и масштабирование

# Проблемы и решения

## Первый запуск

- [x] Найти человека, имеющего опыт с source generator’ами

## Поддержка IDE

- [x] Отладить разработку

## Тестирование и масштабирование

- [x] Запускать на реальном проекте

# Проблемы и решения

## Первый запуск

- [x] Найти человека, имеющего опыт с source generator’ами

## Поддержка IDE

- [x] Отладить разработку

## Тестирование и масштабирование

- [x] Запускать на реальном проекте
- [x] Разобраться с тестированием

# Проблемы и решения

## Первый запуск

- [x] Найти человека, имеющего опыт с source generator’ами

## Поддержка IDE

- [x] Отладить разработку

## Тестирование и масштабирование

- [x] Запускать на реальном проекте
- [x] Разобраться с тестированием

## Ошибки

# Проблемы и решения

## Первый запуск

- [x] Найти человека, имеющего опыт с source generator’ами

## Поддержка IDE

- [x] Отладить разработку

## Тестирование и масштабирование

- [x] Запускать на реальном проекте
- [x] Разобраться с тестированием

## Ошибки

- [x] Использовать диагностики

# Проблемы и решения

## Первый запуск

- ✅ Найти человека, имеющего опыт с source generator’ами

## Поддержка IDE

- ✅ Отладить разработку

## Тестирование и масштабирование

- ✅ Запускать на реальном проекте
- ✅ Разобраться с тестированием

## Ошибки

- ✅ Использовать диагностики

## Новые абстракции

# Проблемы и решения

## Первый запуск

- ✓ Найти человека, имеющего опыт с source generator’ами

## Поддержка IDE

- ✓ Отладить разработку

## Тестирование и масштабирование

- ✓ Запускать на реальном проекте
- ✓ Разобраться с тестированием

## Ошибки

- ✓ Использовать диагностики

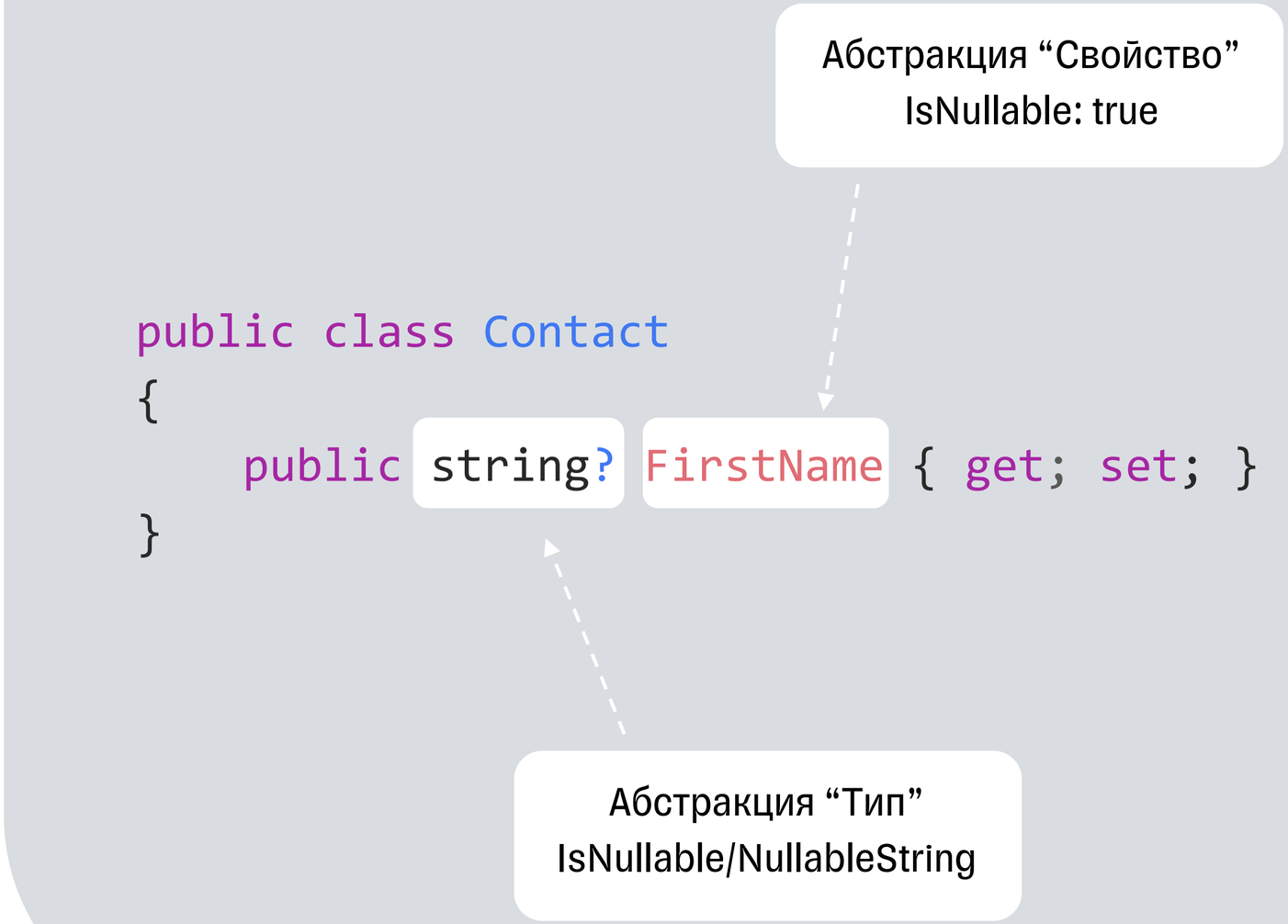
## Новые абстракции

- ✓ Сначала написать код руками

# Интересные ошибки

# Абстракция потекла

**Является ли nullability атрибутом свойства?**



# Абстракция потекла — 2

**А коллекция?**
**Относится к типу или к свойству?**

```
public class Contact
{
    public string[] Tags { get; set; }
}
```

# Абстракция потекла — 3

❌ Абстракция “Свойство”
IsCollection: true

```
public class Contact
{
    public double[][][] Points { get; set; }
}
```

✔ Абстракция “Тип”
ArrayType \ ItemType = ArrayType

# Неприятное событие

```
foreach (var property in publicProperties)
{
    sourceCode.Append($"public {property.Type} {property.Name} {{ get; set; }}");
}
```

# Неприятное событие — 2

```
foreach (var property in publicProperties)
{
    sourceCode.Append($"public {property.Type} {property.Name} {{ get; set; }}");
}

public class InputObject
{
    public string @event { get; set; }
}


public class OutputObject
{
    public string event { get; set; } // Compilation Error
}
```

# We need to go deeper

```
var className = maskableType.Name;
var @namespace = maskableType.ContainingNamespace.ToString();

sourceCode.Append($"var instance = new {@namespace}.{className}();");
```

# We need to go deeper — 2

```
var className = maskableType.Name;
var @namespace = maskableType.ContainingNamespace.ToString();

sourceCode.Append($"var instance = new {@namespace}.{className}();");

namespace Test

public class OuterClass
{
    public class InnerClass { // ... }
}

// ...
var instance = new Test.InnerClass(); // Compilation Error
```

# Порядок прежде всего

```
// return value switch
// {
//     MyType myTypeInstance => myTypeInstance.Do()
// }

foreach (var (type, fullName) in classDescriptions)
{
    sourceCode.Append($"{type} {fullName} => {f}.Do(),");
}
```

# Порядок прежде всего — 2

```
// return value switch
// {
//     MyType myTypeInstance => myTypeInstance.Do()
// }

foreach (var (type, fullName) in classDescriptions)
{
    sourceCode.Append($"{type} {fullName} => {f}.Do(),");
}

return value switch
{
    object objectValue => objectValue.GetHashCode(),
    string stringValue => stringValue.Length, // Compilation Error
    _ => throw new Exception()
};
```

# Пространство и время имён

```
context.AddSource($"{classDescription.Name}Builder.g.cs", sourceCode);
```

# Пространство и время имён — 2

```
context.AddSource($"{classDescription.Name}Builder.g.cs", sourceCode);

namespace Test
{
    public class MyClass
    {
        // ...
    }
}

namespace Test2
{
    public class MyClass
    {
        // ...
    }
}
```

# Сравнение подходов

| | Рефлексия | Код | Source Generator |
|---|---|---|---|
| Простота использования | ✓ | ✗ | ✓ |
| Простота написания кода | ✓ ✗ | ✓ | ✗ |
| Масштабирование | ✓ | ✓ | ✗ |
| Тестирование | ✗ | ✓ | ✗ |
| Поддержка | ✓ ✗ | ✓ | ✗ |
| Дебаг | ✓ ✗ | ✓ | ✗ |

* Нужно время

IT's TINKOFF

# Source generator

- Когда следует использовать source generator
- Преимущества source generator’ов

# Source Generator

## Когда использовать

- Простой алгоритм
- Много однообразного кода (если проблема не в архитектуре)
- Работа с атрибутами
- Требования к производительности

## Когда не использовать

- Сложный алгоритм
- Объём кода генератора близок к итоговому
- Требования не стабилизировались

## Преимущества

- Типобезопасность
- Декларативный подход
- Можно смотреть написанный код
- Низкое влияния на билд проекта (кроме правки связанного кода)
- Проверки на этапе компиляции

# Зачем?

**Новый опыт**

**Качественно новый уровень продукта**

# Пример лога

```
{
    "Deal": {
        "Details": {
            "Summary": {
                "RegistrationRegion": "16 => 86",
                "AddressConfirmed": "null => true“
            },
            "Contacts": [{
                "Details": {
                    "Email": "a*******z@mail.ru => a*******z@gmail.com"
                },
                "Id[Id]": "1234567"
            }]
        }
    }
}
```

# Вопросы


IT's
TINKOFF

# Ссылки

Андрей Дятлов — Source Generator 2.0
https://www.youtube.com/watch?v=p6rxDJwQpoI

Андрей Дятлов — Source Generators в действии
https://www.youtube.com/watch?v=6QWZds35rGs

Список библиотек на source generator’ах
https://github.com/amis92/csharp-source-generators

IT's TINKOFF

# Backup slides