

Статические члены класса

Финальные переменные

НАШИ ПРАВИЛА



Включенная камера



Вопросы по поднятой руке



Не перебиваем друг друга



Все вопросы, не связанные с тематикой курса (орг-вопросы и т. д.), должны быть направлены куратору



Подготовьте свое рабочее окружение для возможной демонстрации экрана (закройте лишние соцсети и прочие приложения)

ЦЕЛЬ

**Научиться использовать статические члены класса и
финальные переменные**

ПЛАН ЗАНЯТИЯ

МОДИФИКАТОР `final`

`final` И ССЫЛОЧНЫЕ ТИПЫ

СТАТИЧЕСКИЕ ПОЛЯ

ОБРАЩЕНИЕ К СТАТИЧЕСКИМ
ПОЛЯМ

СТАТИЧЕСКИЙ ИНИЦИАЛИЗАТОР

СТАТИЧЕСКИЕ МЕТОДЫ,
КОНСТАНТЫ И МАГИЧЕСКИЕ
ЧИСЛА

МОДИФИКАТОР final

Модификатор **final** позволяет создавать как переменные, которые могут быть инициализированы только один раз, так и константы.

В случае, если мы объявляем переменную с **final** без указания какого-либо значения, то в будущем мы можем проинициализировать эту переменную. При этом изменить значение этой переменной далее невозможно:

```
final int value;  
// значение переменной задается  
// во время работы приложения  
value = scanner.nextInt();  
// ошибка компиляции  
value = 5;
```

Также мы имеем возможность создать **константу**, присвоив финальной переменной значение непосредственно в коде:

```
final int value = 5  
// ошибка компиляции  
value = 10;
```

Такие переменные гарантируют, что их значения не изменятся в коде далее, что повышает читаемость кода и гарантирует непреднамеренное изменение какого-либо важного значения.

СТАВИМ ПЛЮС, ЕСЛИ ПОНЯТНО



Модификатор `final`



Необходимость его применения

final И ССЫЛОЧНЫЕ ТИПЫ

Следует помнить об особенностях работы final с **ССЫЛОЧНЫМИ** типами, например с массивами. Применение этого модификатора не запрещает изменять сам массив, на который ссылается переменная. При этом изменить саму ссылку на массив невозможно:

// a ссылается на массив

```
final int[] a = {4, 2, 10, 11};
```

// мы можем изменить значение по индексу

```
a[1] = 777;
```

// изменение ссылки - ошибка компиляции

```
a = null;
```

Аналогичным образом final взаимодействует с объектными переменными

// polo ссылается на объект

```
final Car polo = new Car(45, 6);
```

// мы можем изменить состояние объекта

```
polo.refuel(30);
```

// изменение ссылки - ошибка компиляции

```
polo = new Car(10, 2);
```

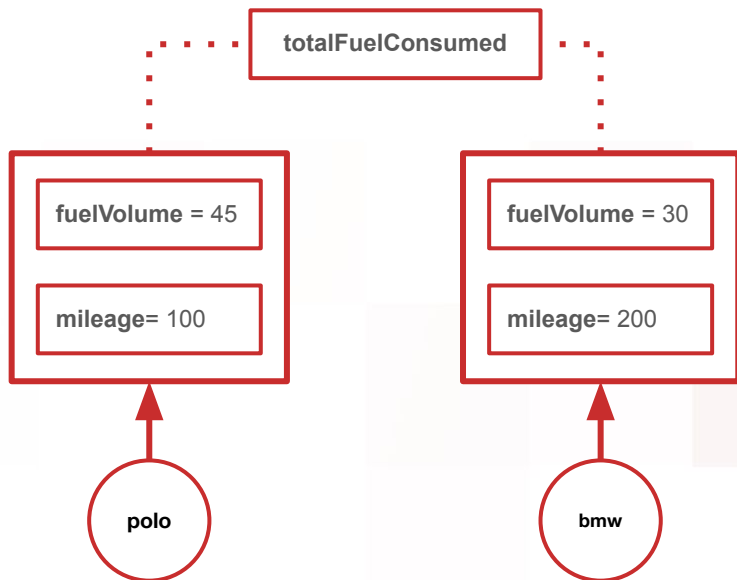
СТАВИМ ПЛЮС, ЕСЛИ ПОНЯТНО



Особенности final при использовании
ссылочных типов

СТАТИЧЕСКИЕ ПОЛЯ

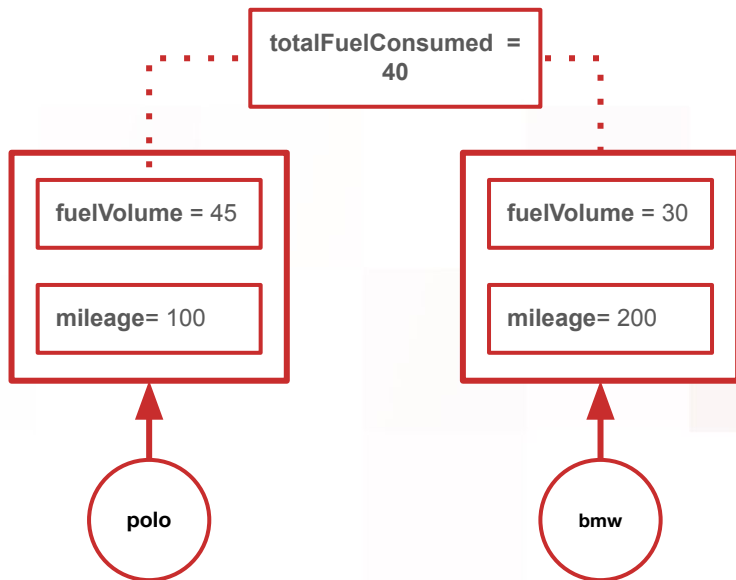
Статическое поле класса представляет собой глобальную переменную, общую для всех объектов, созданных на основе какого-либо класса



```
class Car {  
  
    // ...  
    // используем public только для проверки  
    public static int totalFuelConsumed = 0;  
  
    public boolean go(double kilometers) {  
        // ...  
  
        if (fuelVolume >= fuelNeeded) {  
            // ...  
  
            // Обновляем объем затраченного топлива  
            totalFuelConsumed += fuelNeeded;  
  
            return true;  
        } else {  
            return false;  
        }  
    }  
    // ...  
}
```

ОБРАЩЕНИЕ К СТАТИЧЕСКИМ ПОЛЯМ

Значение статического поля можно получить и задать непосредственно **по названию класса**, а также из любого **его объекта** (чего в реальной разработке никогда не делают)



// обращение к полю через имя класса до создания объектов

```
Car.totalFuelConsumed = 15;
```

// создание объектов класса

```
Car polo = new Car(45, 6);
```

```
Car bmw = new Car(68, 6.5);
```

// изменение глобального статического поля через имена переменных

```
polo.totalFuelConsumed = 30;
```

```
bmw.totalFuelConsumed = 40;
```

СТАВИМ ПЛЮС, ЕСЛИ ПОНЯТНО

■
Статическое поле как глобальная переменная

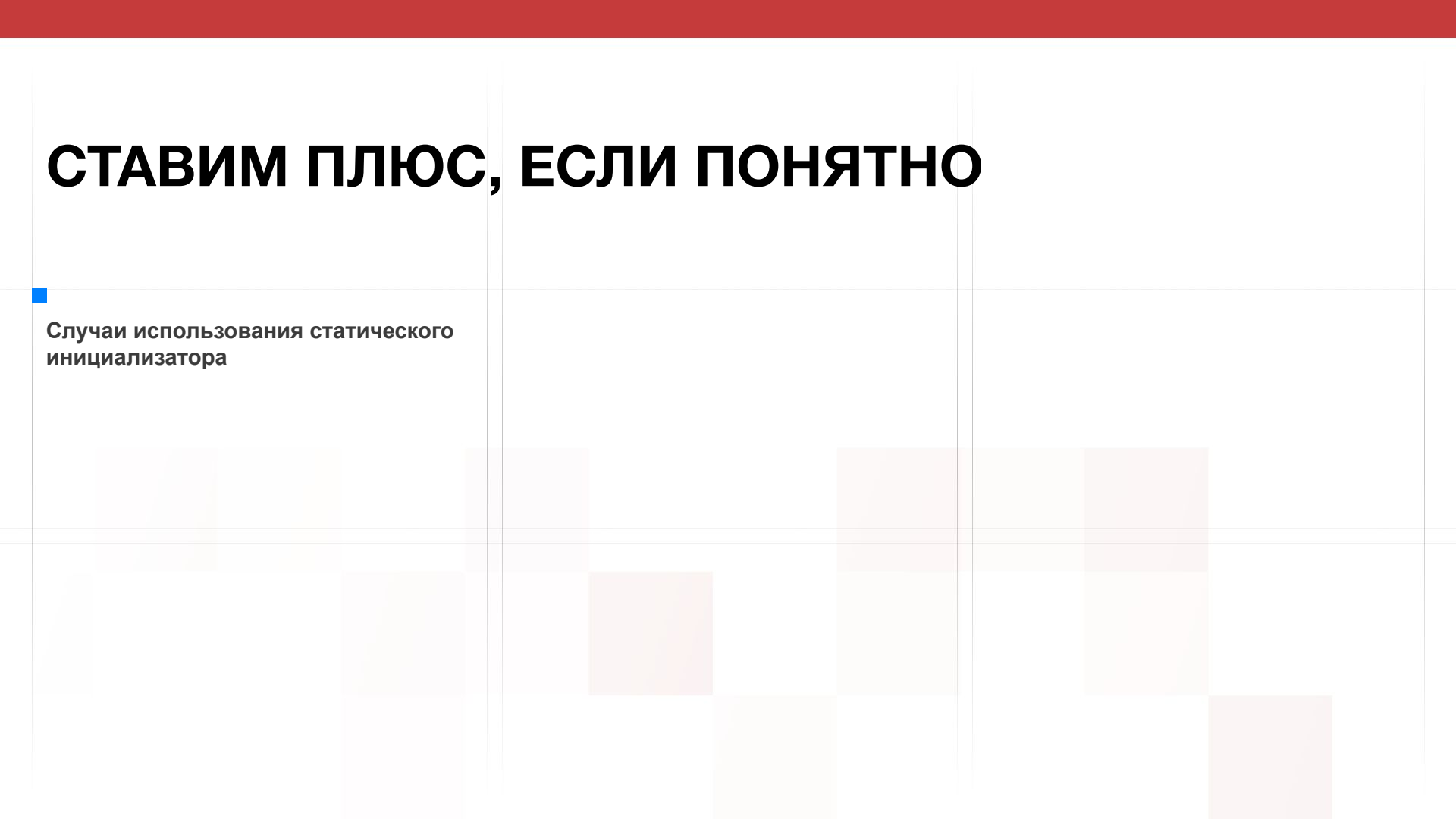
■
Обращение к статическим полям

СТАТИЧЕСКИЙ ИНИЦИАЛИЗАТОР

Статический инициализатор, или статический блок инициализации используется для инициализации статических полей класса. Он выполняется один раз, когда класс загружается в память, до создания любых объектов класса. Статические инициализаторы особенно полезны при сложной логике инициализации поля.

```
class ArrayUtil {  
    static int[] randomArray;  
  
    static {  
        // инициализация массива  
        // случайными числами  
    }  
  
    // ...  
}
```

СТАВИМ ПЛЮС, ЕСЛИ ПОНЯТНО



Случаи использования статического инициализатора

СТАТИЧЕСКИЕ МЕТОДЫ

Статические методы могут быть вызваны непосредственно из класса до создания объектов.

В статических методах описывают операции, которые не зависят от наличия или состояния конкретного объекта (например, статические методы класса `Math`)

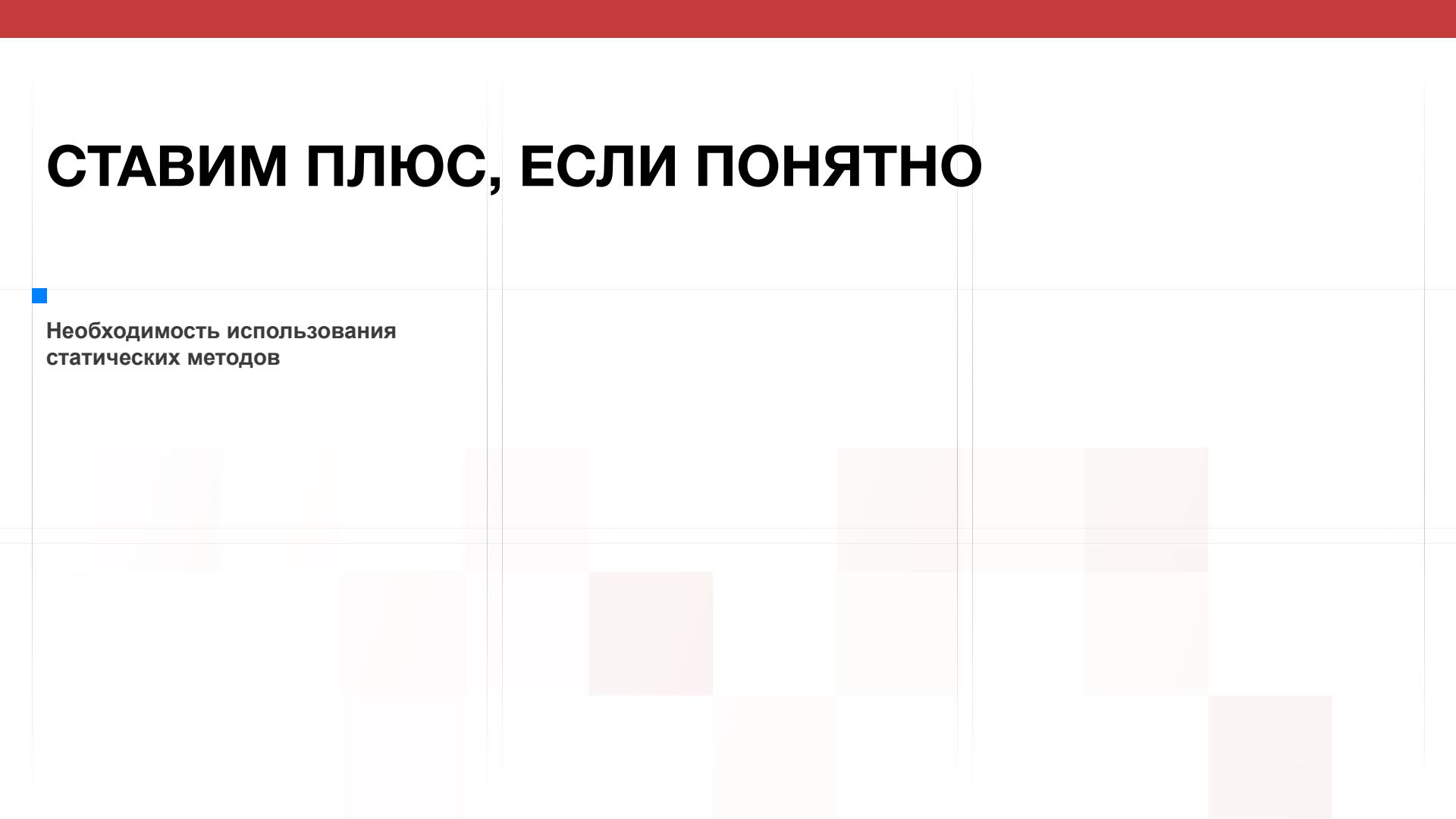
Также с помощью статических методов реализуют утилитные методы для работы с числами, файлами, строками и т.д.

Статические методы могут манипулировать статическими переменными

Статические методы могут обращаться ТОЛЬКО с другим статическим методом или статическим полем

```
class ArrayUtil {  
    static int[] randomArray;  
  
    static {  
        // инициализация массива  
        // случайными числами  
    }  
  
    // Статический метод для вывода всех элементов  
    // массива  
    public static void printElements() {  
        for (int i = 0; i < randomArray[i].length; i++) {  
            System.out.println(randomArray[i]);  
        }  
    }  
}
```

СТАВИМ ПЛЮС, ЕСЛИ ПОНЯТНО



Необходимость использования
статических методов

КОНСТАНТЫ И МАГИЧЕСКИЕ ЧИСЛА

Глобальные константы позволяют избежать использования магических чисел, чье значение не очевидно из контекста использования

```
if (employee.getYearsOfService() > 5) {  
    // начисление бонусов сотруднику  
}
```

Использование константы в данном случае повышает читаемость кода:

```
class CompanyPolicy {  
    public static final int MIN_YEARS_FOR_BONUS = 5;  
}
```

```
if (employee.getYearsOfService() >  
    CompanyPolicy.MIN_YEARS_FOR_BONUS) {  
    // Начисление бонусов сотруднику  
}
```


СТАВИМ ПЛЮС, ЕСЛИ ПОНЯТНО

The background features a light gray grid. A small blue square is located on the left side, aligned with the text 'Необходимость использования констант'. Below this, there is a pattern of larger, semi-transparent squares in various shades of pink, peach, and cream, arranged in a staggered, geometric fashion.

Необходимость использования
констант

ПОИГРАЕМ ;)

■ Статические поля, инициализаторы

■ Статический метод

■ Модификатор `final`

■ Магические числа

■ Константы

ДОМАШНЕЕ ЗАДАНИЕ

Вариант 1:

CurrencyConverter - курсы валют определяются при каждом запуске программы и хранятся в приватном статическом поле, которое инициализируется значениями в статическом инициализаторе. При всем этом поле курсов валют является final. То есть должно быть так:

```
class CurrencyConverter {  
    private final static double[] currencyRates;  
    static {  
        // currencyRates = ....  
    }  
}
```

Вариант 2:

У всех покемонов появляется дополнительный объем HP, он хранится в статическом поле. Задавать значение additionalHP надо в статическом инициализаторе (из пользовательского ввода).

```
private final static double additionalHP; // actualMaxHP = maxHP + additionalHP
```

Вариант 3:

1. Для домашки 17 вариант 2 добавить всем живым организмам приватное поле chromosomNumber, инициализировать его в статическом инициализаторе через пользовательский ввод. Использовать это поле в toString метод
- ```
private final static int chromosomNumber;
```



# **Ваша новая IT-профессия – Ваш новый уровень жизни**

Программирование с нуля в  
немецкой школе AIT TR GmbH