# Static / Final

## МОДИФИКАТОР final

Модификатор **final** в Java играет важную роль в обеспечении безопасности и читаемости кода, позволяя разработчикам создавать переменные, которые могут быть инициализированы только один раз, и константы, значения которых не предполагается изменять после их определения.

Если переменная объявлена как **final**, но без инициализации, её значение должно быть установлено впоследствии, но только один раз. После этого попытки изменить значение такой переменной приведут к ошибке компиляции. Это полезно, когда значение переменной известно только во время выполнения программы, но после его определения оно не должно изменяться:

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

Модификатор **final** также используется для создания **констант**, когда финальной переменной присваивается значение непосредственно при объявлении. Такое значение не может быть изменено в дальнейшем:

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

Использование констант делает код более безопасным и предсказуемым, так как гарантирует, что важные значения не будут изменены в процессе выполнения программы. Также это делает код более понятным и легким для анализа.

## final и ССЫЛОЧНЫЕ ТИПЫ

В Java, использование модификатора **final** с ссылочными типами данных, такими как массивы или объекты, имеет свои особенности. Понимание этих особенностей помогает правильно применять **final** для обеспечения безопасности и надежности кода.

Когда модификатор **final** применяется к переменной массива, это означает, что сама ссылка на массив становится константой, и её нельзя изменить после инициализации. Однако это не ограничивает возможности изменения содержимого массива:

```
// 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;
```

Это означает, что **final** гарантирует неизменность ссылки на массив, но не его содержимого. Аналогичные правила применяются к объектам. Объявление объекта с модификатором **final** запрещает изменение ссылки на объект, но не запрещает изменять состояние самого объекта через его методы:

```
// polo ссылается на объект
Car polo = new Car(45, 6);
// мы можем изменить состояние объекта
polo.refuel(30);

// изменение ссылки - ошибка компиляции
Car polo = new Car(10, 2);
```

# СТАТИЧЕСКИЕ ПОЛЯ И ОБРАЩЕНИЕ К НИМ

Статические поля классов в Java играют роль глобальных переменных для всех экземпляров этого класса. Они предоставляют уникальное пространство в памяти, значение которого разделяется между всеми объектами класса, что позволяет управлять общими данными на уровне 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;
        }
    }
    // …
}
```

В классе `Car` статическое поле `totalFuelConsumed` используется для отслеживания общего количества топлива, потраченного всеми автомобилями. Поскольку это поле статическое, оно существует независимо от любых объектов класса `Car` и их состояния.

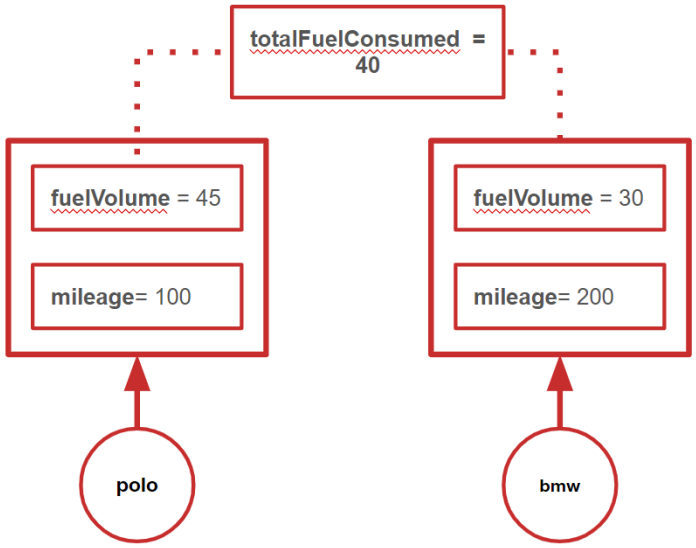
Статические поля доступны для чтения и изменения двумя способами:

- Доступ к статическому полю возможен напрямую **через имя класса**, что является предпочтительным способом, так как это подчеркивает статическую природу поля.
- Хотя доступ к статическому полю также возможен **через экземпляр класса**, такой подход может вводить в заблуждение, поскольку создается впечатление, что значение поля уникально для каждого объекта, что не соответствует действительности.

```
// 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;
```



# СТАТИЧЕСКИЙ ИНИЦИАЛИЗАТОР

**Статический инициализатор** в Java — это специальный блок кода, который используется для инициализации статических полей класса. Этот блок выполняется автоматически один раз, когда класс загружается виртуальной машиной Java, еще до создания любых экземпляров класса и до вызова любых статических методов.

Статические инициализаторы идеально подходят для выполнения сложной логики инициализации, которая не может быть выполнена в момент объявления статического поля. Это может включать заполнение массивов, коллекций, установку сложных зависимостей между статическими данными или выполнение кода, необходимого для настройки окружения класса.

```
class ArrayUtil {
   static int[] randomArray;

   static {
      // инициализация массива
      // случайными числами
   }

   // …
}
```

В этом примере статический блок инициализации используется для заполнения массива **`randomArray`** случайными числами. Блок выполняется один раз, при первом обращении к классу **`ArrayUtil`**, обеспечивая, что массив будет инициализирован до использования.

# СТАТИЧЕСКИЕ МЕТОДЫ

**Статические методы в Java** — это методы, принадлежащие классу, а не отдельным экземплярам или объектам этого класса. Они могут быть вызваны непосредственно через имя класса, даже без создания объектов класса. Эти методы предназначены для выполнения операций, которые не зависят от состояния конкретных объектов класса.

Статические методы выполняют операции, не требующие данных или состояния какого-либо конкретного объекта класса. Примером может служить класс **`Math`**, в котором методы, такие как **`Math.sqrt()`**, являются статическими. Часто статические методы используются для создания утилитных методов, предоставляющих общие служебные функции, такие как обработка строк, операции с файлами и математические вычисления. Статические методы могут читать и модифицировать статические поля класса. Это позволяет им управлять данными, общими для всех объектов класса.

В статических методах можно обращаться только к другим статическим методам и статическим полям класса напрямую. Для доступа к нестатическим полям или методам необходим экземпляр класса.

```
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]);
        }
    }
}
```

В этом примере **`printElements`** — статический метод, который выводит элементы статического массива **`randomArray`**. Этот метод может быть вызван без создания экземпляра **`ArrayUtil`**.

# КОНСТАНТЫ И МАГИЧЕСКИЕ ЧИСЛА

**Глобальные константы** в программировании играют ключевую роль в улучшении читаемости, поддерживаемости и безопасности кода. Они предотвращают использование **магических чисел** — литералов, значение которых без контекста трудно понять или вспомнить. Вместо непосредственного вставления чисел в код, используются именованные константы, которые объясняют смысл этих значений.

Магические числа — это прямое указание числовых значений в коде, которые могут вызвать путаницу или затруднить понимание кода для другого разработчика или даже для автора кода спустя некоторое время. Например:

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

В этом коде **5** — магическое число. Оно указывает на минимальное количество лет службы, необходимое для начисления бонуса, но это значение не объясняется непосредственно в коде. Определение глобальных констант позволяет ясно указать назначение таких числовых значений:

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

Теперь код с использованием этой константы становится более читаемым и понятным:

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