# 10 заметок о модификаторе Static в Java

Модификатор `static` в Java напрямую связан с классом, если поле статично, значит оно принадлежит классу, если метод статичный, аналогично — он принадлежит классу. Исходя из этого, можно обращаться к статическому методу или полю используя имя класса. Например, если поле `count` статично в классе `Counter`, значит, вы можете обратиться к переменной запросом вида: `Counter.count`.

Конечно, следует учитывать модификаторы доступа. Например, поля `private` доступны только внутри класса, в котором они объявлены. Поля `protected` доступны всем классам внутри пакета (*package*), а также всем классам-наследникам вне пакета. Для более подробной информации ознакомьтесь со статьей "private vs protected vs public". Предположим, существует статический метод `increment()` в классе `Counter`, задачей которого является инкрементирование счётчика `count`. Для вызова данного метода можно использовать обращение вида `Counter.increment()`. Нет необходимости создавать экземпляр класса `Counter` для доступа к статическому полю или методу. Это фундаментальное отличие между статическими и НЕ статическими объектами (членами класса). Важное замечание. Не забывайте, что статические члены класса напрямую принадлежат классу, а не его экземпляру. То есть, значение статической переменной `count` будет одинаковое для всех объектов типа `Counter`. В этой статье мы рассмотрим основополагающие аспекты применения модификатора `static` в Java, а также некоторые особенности, которые помогут понять ключевые концепции программирования.

## Что должен знать каждый программист о модификаторе Static в Java.

В этом разделе мы рассмотрим основные моменты использования статических методов, полей и классов. Начнём с переменных.

1. Вы НЕ можете получить доступ к НЕ статическим членам класса, внутри статического контекста, как вариант, метода или блока. Результатом компиляции приведенного ниже кода будет ошибка:

```
1. public class Counter{
2. private int count;
3. public static void main(String args[]){
4.    System.out.println(count); //compile time error
   }}
```

   Это одна из наиболее распространённых ошибок допускаемых программистами Java, особенно новичками. Так как метод `main` статичный, а переменная `count` нет, в этом случае метод `println`, внутри метода `main` выбросит "Compile time error".

2. В отличие от локальных переменных, статические поля и методы НЕ потокобезопасны (Thread-safe) в Java. На практике это одна из наиболее частых причин возникновения проблем связанных с безопасностью мультипоточного программирования. Учитывая что каждый экземпляр класса имеет одну и ту же копию статической переменной, то такая переменная нуждается в защите — «залочивании» классом. Поэтому при использовании статических переменных, убедитесь, что они должным образом синхронизированы (synchronized), во избежание проблем, например таких как «состояние гонки» (race condition).

3. Статические методы имеют преимущество в применении, т.к. отсутствует необходимость каждый раз создавать новый объект для доступа к таким методам. Статический метод можно вызвать, используя тип класса, в котором эти методы описаны. Именно поэтому, подобные методы как нельзя лучше подходят в качестве методов-фабрик (`factory`), и методов-утилит (`utility`). Класс `java.lang.Math` — замечательный пример, в котором почти все методы статичны, по этой же причине классы-утилиты в Java финализированы (`final`).

4. Другим важным моментом является то, что вы НЕ можете переопределять (`Override`) статические методы. Если вы объявите такой же метод в классе-наследнике (`subclass`), т.е. метод с таким же именем и сигнатурой, вы лишь «спрячете» метод суперкласса (`superclass`) вместо переопределения. Это явление известно как сокрытие методов (`hiding methods`). Это означает, что при обращении к статическому методу, который объявлен как в родительском, так и в дочернем классе, во время компиляции всегда будет вызван метод исходя из типа переменной. В отличие от переопределения, такие методы не будут выполнены во время работы программы. Рассмотрим пример:

```
1. class Vehicle{
2.      public static void  kmToMiles(int km){
3.           System.out.println("Внутри родительского
   класса/статического метода");
4.      } }
5.
6. class Car extends Vehicle{
7.      public static void  kmToMiles(int km){
8.           System.out.println("Внутри дочернего класса/статического
   метода ");
9.      } }
10.
11. public class Demo{
12.    public static void main(String args[]){
13.       Vehicle v = new Car();
14.        v.kmToMiles(10);
    }}
```

**Вывод в консоль:**

***Внутри родительского класса/статического метода***

Код наглядно демонстрирует: несмотря на то, что объект имеет тип `Car`, вызван статический метод из класса `Vehicle`, т.к. произошло обращение к