СТАТИЧЕСКИЕ ЭЛЕМЕНТЫ. СПЕЦИФИКАТОР FINAL. ВЛОЖЕННЫЕ И ВНУТРЕННИЕ КЛАССЫ

Кроме обычных методов и полей класс может иметь статические поля, методы, константы и инициализаторы. Например, главный класс программы имеет метод main, который является статическим:

```
public static void main(String[] args) {

}
```

Для объявления статических переменных, констант, методов и инициализаторов перед их объявлением указывается ключевое слово **static**.

Статические поля

При создании объектов класса для каждого объекта создается своя копия нестатических обычных полей. А статические поля являются общими для всего класса. Поэтому они могут использоваться без создания объектов класса.

Например, создадим статическую переменную:

```
public class Program{

    public static void main(String[] args) {

        Person tom = new Person();
        Person bob = new Person();

        tom.displayId();    // Id = 1
        bob.displayId();    // Id = 2
        System.out.println(Person.counter); // 3

        // изменяем Person.counter
        Person.counter = 8;

        Person sam = new Person();
        sam.displayId();    // Id = 8
    }
}
class Person{

    private int id;
    static int counter=1;

    Person(){
        id = counter++;
```

```
    }
    public void displayId(){

        System.out.printf("Id: %d \n", id);
    }
}
```

Класс Person содержит статическую переменную counter, которая увеличивается в конструкторе и ее значение присваивается переменной id. То есть при создании каждого нового объекта Person эта переменная будет увеличиваться, поэтому у каждого нового объекта Person значение поля id будет на 1 больше чем у предыдущего.

Так как переменная counter статическая, то мы можем обратиться к ней в программе по имени класса:

```
System.out.println(Person.counter); // получаем значение
Person.counter = 8;                 // изменяем значение
```

Консольный вывод программы:

```
Id = 1
Id = 2
3
Id = 8
```

Статические константы

Также статическими бывают константы, которые являются общими для всего класса.

```
public class Program{

    public static void main(String[] args) {

        double radius = 60;
        System.out.printf("Radisu: %f \n", radius);            // 60
        System.out.printf("Area: %f \n", Math.PI * radius);    // 188,4
    }
}
class Math{
    public static final double PI = 3.14;
}
```

Стоит отметить, что на протяжении всех предыдущих тем уже активно использовались статические константы. В частности, в выражении:

```
1    System.out.println("hello");
```

out как раз представляет статическую константу класса System. Поэтому обращение к ней идет без создания объекта класса System.

Статические инициализаторы

Статические инициализаторы предназначены для инициализации статических переменных, либо для выполнения таких действий, которые выполняются при создании самого первого объекта. Например, определим статический инициализатор:

```
public class Program{

    public static void main(String[] args) {

        Person tom = new Person();
        Person bob = new Person();

        tom.displayId();    // Id = 105
        bob.displayId();    // Id = 106
    }
}
class Person{

    private int id;
    static int counter;

    static{
        counter = 105;
        System.out.println("Static initializer");
    }
    Person(){
        id=counter++;
        System.out.println("Constructor");
    }
    public void displayId(){

        System.out.printf("Id: %d \n", id);
    }
}
```

Статический инициализатор определяется как обычный, только перед ним ставится ключевое слово static. В данном случае в статическом инициализаторе мы устанавливаем начальное значение статического поля counter и выводим на консоль сообщение.

В самой программе создаются два объекта класса Person. Поэтому консольный вывод будет выглядеть следующим образом:

```
Static initializer
Constructor
Constructor
Id: 105
Id: 106
```

Стоит учитывать, что вызов статического инициализатора производится после загрузки класса и фактически до создания самого первого объекта класса.

Статические методы

Статические методы также относятся ко всему классу в целом. Например, в примере выше статическая переменная counter была доступна извне, и мы могли изменить ее значение вне класса Person. Сделаем ее недоступной для изменения извне, но доступной для чтения. Для этого используем статический метод:

```
public class Program{

    public static void main(String[] args) {

        Person.displayCounter();    // Counter: 1

        Person tom = new Person();
        Person bob = new Person();

        Person.displayCounter();    // Counter: 3
    }
}
class Person{

    private int id;
    private static int counter = 1;

    Person(){
```

```
        id = counter++;
    }
    // статический метод
    public static void displayCounter(){

        System.out.printf("Counter: %d \n", counter);
    }
    public void displayId(){

        System.out.printf("Id: %d \n", id);
    }
}
```

Теперь статическая переменная недоступна извне, она приватная. А ее значение выводится с помощью статического метода displayCounter. Для обращения к статическому методу используется имя класса: Person.displayCounter().

При использовании статических методов надо учитывать ограничения: в статических методах мы можем вызывать только другие статические методы и использовать только статические переменные.

Вообще методы определяются как статические, когда методы не затрагиют состояние объекта, то есть его нестатические поля и константы, и для вызова метода нет смысла создавать экземпляр класса. Например:

```
public class Program{

    public static void main(String[] args) {

        System.out.println(Operation.sum(45, 23));          // 68
        System.out.println(Operation.subtract(45, 23));     // 22
        System.out.println(Operation.multiply(4, 23));      // 92
    }
}
class Operation{

    static int sum(int x, int y){
        return x + y;
    }
    static int subtract(int x, int y){
        return x - y;
    }
    static int multiply(int x, int y){
        return x * y;
    }
}
```

В данном случае для методов sum, subtract, multiply не имеет значения, какой именно экземпляр класса Operation используется. Эти методы работают

только с параметрами, не затрагивая состояние класса. Поэтому их можно определить как статические.