Переопределение статических методов

В этом материале разберём работу со статическими методами с учётом наследования классов. Статические методы не умеют переопределяться. Это происходит потому, что методы родительского класса — статические и не являются частью дочернего класса, хотя доступны в нём.

Убедимся в этом на примере. У нас есть класс Bus, его наследник — класс ElectricBus. Создадим в классе Bus статический метод, который будет подсчитывать количество созданных автобусов в соответствующую статическую переменную.

Вначале создадим эту переменную:

```
private static int count;
```

Пропишем в конструкторе увеличение числа автобусов:

```
public Bus(double consumptionRate) {
    this.consumptionRate = consumptionRate;
    count++;
}
```

Теперь создадим метод, который будет возвращать количество автобусов:

```
public static int getCount() {
    return count;
}
```

Посмотрим, как работает метод. Для этого создадим три обычных автобуса и два электробуса, затем выведем их количество в консоль:

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

        ElectricBus eBus1 = new ElectricBus(0.001, 0.1);
        ElectricBus eBus2 = new ElectricBus(0.001, 0.1);
        Bus bus1 = new Bus(0.001);
        Bus bus2 = new Bus(0.001);
        Bus bus3 = new Bus(0.001);

        System.out.println("Количество автобусов: " +
```

```
Bus.getCount());
        System.out.println("Количество электробусов: " +
ElectricBus.getCount());
    }
}
```

У ElectricBus метод доступен, поскольку ElectricBus — наследник класса Bus. Запустим код и посмотрим, как он будет работать:

```
Количество автобусов: 5
Количество электробусов: 5
```

В обоих случаях количество равно пяти, то есть подсчитывается общее количество автобусов и электробусов. Если мы хотим, чтобы количество автобусов и электробусов подсчитывалось отдельно, надо создать подсчёт ещё и в электробусах. Давайте сделаем это. Также создадим приватную статическую переменную:

```
private static int count;
```

Пропишем её увеличение в конструкторе:

```
public ElectricBus(double consumptionRate, double
minimalTankFullnessRate) {
    super(consumptionRate);
    this.minimalTankFullnessRate = minimalTankFullnessRate;
    count++;
}
```

Создадим метод, который будет возвращать количество электробусов:

```
public static int getCount() {
    return count;
}
```

Если попытаться дописать над этим методом аннотацию @Override, то среда разработки подчеркнёт аннотацию и выдаст ошибку Method does not override method from superclass, потому что это статический метод. Это говорит о том, что никакого переопределения статических методов не происходит.

Запустим код и посмотрим, как он будет работать:

```
Количество автобусов: 5
Количество электробусов: 2
```

Видим, что всё считается верно. Количество автобусов равно пяти, а количество электробусов — двум. То есть при создании электробусов у нас прибавляется единица, как в родительском классе, так и в дочернем.

Вывод

Вы увидели, что не существует переопределения статических методов. Если в дочернем классе создать такой же статический метод, как в родительском, то это будет новый отдельный метод.

Код всех классов:

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

        ElectricBus eBus1 = new ElectricBus(0.001, 0.1);
        ElectricBus eBus2 = new ElectricBus(0.001, 0.1);
        Bus bus1 = new Bus(0.001);
        Bus bus2 = new Bus(0.001);
        Bus bus3 = new Bus(0.001);

        System.out.println("Количество автобусов: " +
Bus.getCount());
        System.out.println("Количество электробусов: " +
ElectricBus.getCount());
    }
}
public class Bus {
    private double tankFullnessRate; //насколько заполнен
топливный бак (от нуля до единицы)
    private final double consumptionRate; // расход топлива на
1 км
    /*
    условно, если бак заполнен на 100% (единица), а расход
0,01, то бака хватит ровно на 100 км
     */

    private static int count;

    public Bus(double consumptionRate) {
        this.consumptionRate = consumptionRate;
        count++;
```

```
    }

    public boolean run(int distance) {    // проезд автобуса на
определённое расстояние в километрах
        if (powerReserve() < distance) { // проверяется,
хватит ли топлива на это расстояние
            return false;
        }
        tankFullnessRate -= distance * consumptionRate;
        return true; // из переменной с объёмом топливного
бака вычитается путь в километрах, умноженный на
        // расход топлива на 1 км
    }

    public final void refuel(double tankRate) { // заправка
автобуса
        double total = tankFullnessRate + tankRate; //
насколько заполнен топливный бак плюс количество доливаемого
топлива
        tankFullnessRate = total > 1 ? 1 : total; // если
попытаться заправить больше, чем на 100%, он заполнится
только до 100%
    }

    public int powerReserve() {
       return (int) (tankFullnessRate / consumptionRate); //
на сколько километров хватит оставшегося запаса топлива
    }

    public double getConsumptionRate() { // уровень
потребления топлива
        return consumptionRate;
    }

    public double getTankFullnessRate() { // степень
наполненности бака
        return tankFullnessRate;
    }

    // метод, который возвращает количество созданных
автобусов

    public static int getCount() {
        return count;
    }

    public static void setCount(int count) {
        Bus.count = count;
    }
```

```
    @Override
    public String toString() {
        return "Bus{" +
                "tankFullnessRate=" + tankFullnessRate +
                ", consumptionRate=" + consumptionRate +
                '}';
    }
}
```

```
import java.util.Objects;

public class ElectricBus extends Bus {

    // появился электробус, но аккумулятор, в отличие от
бензобака, не может быть разряжен в ноль

    private final double minimalTankFullnessRate;
    private static int count;

    public ElectricBus(double consumptionRate, double
minimalTankFullnessRate) {
        super(consumptionRate);
        this.minimalTankFullnessRate =
minimalTankFullnessRate;
        count++;
    }

    // в этом классе нужен метод из родительского, но с другим
поведением
    // начинаем вводить powe... появляется подсказка, выбираем
её и получаем переопределённый метод
    @Override
    public int powerReserve() {
        double remainingRate = getTankFullnessRate() -
minimalTankFullnessRate; // оставшийся уровень
        if (remainingRate <= 0) {
            return 0;
        }
        return (int) (remainingRate / getConsumptionRate());
    }

    public static int getCount() {
        return count;
```

```
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true; // проверка на равенство
ссылок
        if (o == null || getClass() != o.getClass()) return
false; // не равен ли переданный объект значению null
        // затем проверяется, к каким классам относятся
объекты (текущий и переданный в качестве параметра)
        ElectricBus that = (ElectricBus) o; // происходит
приведение к классу Bus
        return Double.compare(that.minimalTankFullnessRate,
minimalTankFullnessRate) == 0; // сравниваются все переменные
        // которые есть в этих объектах
    }

    @Override
    public int hashCode() {
        return Objects.hash(minimalTankFullnessRate); //
сравниваются хеши от всех переменных текущего объекта
    }
}
```