Модификаторы доступа

В этой теме мы разберём модификаторы доступа — специальные ключевые слова, которые используют для установки уровня доступа к классам, их переменным, методам и конструкторам. В Java есть четыре уровня доступа, которые устанавливаются при помощи таких модификаторов.

С двумя из них вы уже хорошо знакомы, но мы дополнительно разберём их на примерах, так как при наследовании классов их работа специфична.

Самый открытый уровень доступа — это уровень public. Его устанавливают с помощью ключевого слова public, которое вы уже много раз видели в коде. Если это ключевое слово стоит перед именем класса, конструктора, переменной, метода или enum, значит, все они доступны в любом другом месте этой программы.

Следующий уровень доступа — protected (от англ. «защищённый»). Если переменные, метод или конструктор помечены этим словом, они доступны только внутри этого же класса или внутри его дочерних классов.

Уровень доступа private (от англ. «приватный», «частный») ограничивает доступ к переменной, методу или конструктору только этим же классом. Из других классов, в том числе классов-наследников, родителей или классов в том же или другом пакете, нет доступа к приватным переменным, методам или конструкторам. Они доступны только из этого же класса.

Последний уровень доступа — package-private. Он ограничивает доступ к классам, методам, переменным и конструкторам текущим пакетом, в котором они находятся.

Все предыдущие уровни доступа устанавливают при помощи одноимённых модификаторов: public, private и protected. А уровень package-private — это

уровень доступа по умолчанию, который работает, если модификатор не установлен.
Теперь посмотрим на практике, как работают эти модификаторы доступа.

С модификатором public всё ясно. Он предоставляет доступ к классам, переменным, методам и конструкторам из любого другого места в коде. С модификатором доступа private тоже всё понятно. К приватным переменным, методам и конструкторам есть доступ только из того же класса. Перейдём к модификатору protected, который ограничивает доступ к переменным, конструкторам и методам извне и позволяет использовать их только в этом же классе и в классах-наследниках. Зачем это нужно?

Существует класс ElectricBus. К примеру, у электробусов, в отличие от обыкновенных автобусов, есть особенность — изнашиваемость аккумулятора. Представим, что мы точно знаем, что она одинакова для всех электробусов. Каждый новый заряд аккумулятора более чем на 10% увеличивает скорость его потребления на одну сотую процента. Вы хотите учесть эту особенность в коде.

Для этого можно создать переменную с модификатором доступа private в классе ElectricBus. Пусть это будет статичная переменная, переменная final (неизменяемая), переменная типа double, которую мы назовём degradationRate.

```
private static final degradationRate = 0.0001;
```

Это будет константа, так как мы точно знаем, что она одинакова для всех электробусов. Теперь учтём её в коде. Для этого переопределим метод.

```
@Override
public final void refuel(double tankRate) {
   super.refuel(tankRate);
}
```

Для начала вызываем родительский метод, а затем увеличиваем consumptionRate, если заряд составил более 10%. Эти 10% мы запишем в качестве константы.

```
private static final double minRateToDegrade = 0.1;
```

```
if (tankRate > minRateToDegrade) {

}
```

Если мы заряжаем электробус больше чем на 10%, то должны увеличить consumptionRate на degradationRate.

```
@Override
public final void refuel(double tankRate) {
    super.refuel(tankRate);
    if (tankRate > minRateToDegrade) {
        consumptionRate += degradationRate;
    }
}
```

Каждый новый заряд аккумулятора более чем на 10% увеличивает скорость его потребления на одну сотую процента. Но при этом возникает проблема.

```
consumptionRate += degradationRate;
```

Мы не можем обратиться к переменной consumptionRate, так как она определена в родительском классе как приватная. Но для реализации нового метода необходим доступ к ней из дочернего класса. Для этого обозначим переменную модификатором доступа protected и уберём final, чтобы эту переменную можно было изменять. Теперь код будет работать.

Как видите, переменная стала доступна в классе-наследнике.

```
protected double consumptionRate;
```

У методов тоже можно устанавливать модификатор доступа protected. Например, существует метод:

```
powerReserve()
```

Его можно сделать protected, если нужно ограничить доступ к нему. Поменяем public на protected.

```
protected int powerReserve() {
    return (int) (tankFullnessRate / consumptionRate);
}
```

При этом обратите внимание на его наследника — электробус:

```
@Override
public int powerReserve() {
    double remainingRate = getTankFullnessRate() -
minimalTankFullnessRate;
    if (remainingRate <= 0) {
        return 0;
    }
    return (int) (remainingRate / getConsumptionRate());
}
```

Доступ к этому методу останется, потому что этот метод в нем переопределён. Но обратите внимание, что он переопределён с модификатором доступа public. Если мы создадим электробус в другом коде, то сможем обратиться к этому методу, потому что он public.

При переопределении методов можно расширять уровень доступа, но сужать нельзя.

Напишем public здесь:

```
public int powerReserve() {
    return (int) (tankFullnessRate / consumptionRate);
}
```

И сделаем protected здесь:

```
@Override
public int powerReserve() {
    double remainingRate = getTankFullnessRate() -
minimalTankFullnessRate;
    if (remainingRate <= 0) {
        return 0;
    }
    return (int) (remainingRate / getConsumptionRate());
}
```

Такой вариант не сработает, так как сужать нельзя. Поэтому возвращаем всё обратно.

Если мы не будем указывать ни один из модификаторов, то доступ к этому классу, его переменной, методу или конструктору будет только из классов, находящихся в том же пакете. Чтобы убедиться в этом, создадим пакет transport. Поместим в него автобусы Bus и ElectricBus.

Теперь, если нужно ограничить доступ к методу powerReserve() извне, но при этом мы хотим пользоваться им внутри пакета, можно убрать у него ключевое слово protected.

```
int powerReserve() {
    return (int) (tankFullnessRate / consumptionRate);
}
```

Здесь тоже можно убрать ключевое слово, если этот метод не должен использовать кто-то снаружи пакета.

```
@Override
int powerReserve() {
    double remainingRate = getTankFullnessRate() -
minimalTankFullnessRate;
    if (remainingRate <= 0) {
        return 0;
    }
    return (int) (remainingRate / getConsumptionRate());
}
```

Теперь его нельзя вызывать из классов, которые находятся вне этого пакета. Давайте проверим это. Мы создаём автобус и пробуем вызвать у него метод powerReserve(). Такой вызов не работает. Метод не является публичным, а значит не будет доступен снаружи пакета. Так работают модификаторы уровней доступа к классам, их переменным, методам и конструкторам.

Подведём итоги

Модификатор доступа public позволяет получить доступ из любого места: из того же класса, из классов в том же пакете и из подклассов, то есть классов-наследников, а также из кода всего приложения.

Модификатор protected ограничивает доступ к конструкторам, методам и переменным из кода снаружи, за исключением кода в дочерних классах и в классах того же пакета.

Доступ по умолчанию, предоставляемый при отсутствии модификатора, есть только у классов того же пакета. Его нет ни у классов-наследников, ни у классов вне этого пакета.

Модификатор private полностью ограничивает доступ к переменным, методам и конструкторам класса. Их можно использовать только из того же класса.

В этой теме вы разобрали, как отличать четыре модификатора доступа в Java. С их помощью можно регулировать уровень доступа к классам, конструкторам, методам и переменным.