Класс Object

Привет!
В этом материале вы познакомитесь с очень важным классом языка программирования Java — Object.

Объектно ориентированное программирование в языке Java устроено таким образом, что все классы унаследованы от Object. Даже если вы создаёте новый класс, он тоже будет унаследован от класса Object. Что это значит? И зачем это нужно?

Это значит, что все классы, а следовательно, и их объекты будут вести себя одинаково благодаря методам, которые наследуются от Object.

Давайте разберём все методы класса Object. Разделим их на группы.

Первая группа — это методы, относящиеся к сравнению и копированию объектов:

- `clone()` — создаёт и возвращает копию объекта (как именно он работает и в каких случаях его можно использовать, рассмотрим позже);
- `equals(Object obj)` — сравнивает текущий объект, у которого он вызывается, с объектом, который передан ему в качестве параметра, и возвращает либо true, либо false;
- `hashCode()` — возвращает целое число. Если метод не переопредёлен, то это число рассчитывается виртуальной машиной Java, чтобы упростить сравнение объектов. Возвращаемое число в этом случае будет одним и тем же для одинаковых объектов в процессе выполнения кода. У разных объектов это число тоже может быть одинаковым.

Как работают методы `equals()` и `hashCode()`, мы уже разбирали ранее. Теперь посмотрим на их исходный код в Object.

Для начала найдём в этом классе метод `hashCode()` (переходим в класс Object, нажимаем на Ctrl + F, в строке поиска вводим: hashCode).

Обратите внимание: реализация метода скрыта, он помечен как native. Это означает, что метод реализован не на Java, а на языке C или C++, на котором написана Java-машина:

```
@IntrinsicCandidate
public native int hashCode();
```

Теперь найдём метод `equals()`. Вот он:

```
public boolean equals(Object obj) {
    return (this == obj);
}
```

Мы используем Java версии 17. Здесь код `equals()` немного отличается от кода этого метода в предыдущих версиях. В Java версии 17 просто сравнивается ссылка. Если ссылка на переданный объект (в параметрах) равна ссылке на текущий объект (переменной `this`), это значит, что перед нами один и тот же объект. То есть метод не гарантирует корректности сравнения.

Если `equals()` возвращает true — объекты точно одинаковые, если метод возвращает false — это не означает, что они неодинаковые.

Проверим это. Откроем класс Main, создадим два одинаковых автобуса и сравним их.

```
public class Main {
    public static void main(String[] args) {
        Bus bus1 = new Bus(0.001);
        Bus bus2 = new Bus(0.001);
        System.out.println(bus1.equals(bus2));
    }
}
```

Поскольку объекты разные, результат:

```
false
```

Однако результат false всё же не гарантия того, что объекты неодинаковые.

Напишем так:

```
public class Main {
    public static void main(String[] args) {
        Bus bus1 = new Bus(0.001);
        Bus bus2 = bus1;
        System.out.println(bus1.equals(bus2));
    }
}
```

Только в этом случае при запуске в Java версии 17 метод `equals()` выдаст true. Чтобы сравнивать автобусы, мы должны переопределить этот метод в классе Bus.

Вернём код обратно:

```
Bus bus1 = new Bus(0.001);
Bus bus2 = new Bus(0.001);
System.out.println(bus1.equals(bus2));
```

Перейдём в класс Bus. И создадим здесь метод `equals()`.

Начнём писать слово: equ… Появится подсказка ввода. Примем её. Среда разработки предложит ответить на несколько вопросов, чтобы она сама могла сгенерировать метод. Оставим ответы по умолчанию и посмотрим, что произойдёт (нажимаем Next, Next, Finish).

```
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Bus bus = (Bus) o;
    return Double.compare(bus.tankFullnessRate,
tankFullnessRate) == 0 && Double.compare(bus.consumptionRate,
consumptionRate) == 0;
```

```
}

@Override
public int hashCode() {
    return Objects.hash(tankFullnessRate, consumptionRate);
}
```

Проверим сгенерированный код. Обратите внимание: среда разработки создала код и метода `equals()`, и метода `hashCode()`.

Сначала происходит проверка равенства ссылок, так же как и в методе `equals()` класса Object.

```
if (this == o) return true;
```

Затем проверяется, не равен ли переданный в качестве параметра объект (объект **o**) значению null:

```
if (o == null
```

После этого среда разработки проверяет, к каким классам относятся объекты (текущий и переданный в качестве параметра).

```
|| getClass() != o.getClass()
```

Очевидно, что если сравниваемые объекты относятся к разным классам, то это разные объекты.

Затем происходит приведение переданного объекта к классу Bus:

```
Bus bus = (Bus) o;
```

После чего сравниваются все переменные, которые есть в обоих объектах.

```
bus.tankFullnessRate, tankFullnessRate)
bus.consumptionRate, consumptionRate)
```

Именно здесь объекты сравниваются по существу.

```
return Double.compare(bus.tankFullnessRate, tankFullnessRate)
== 0 && Double.compare(bus.consumptionRate,
consumptionRate) == 0;
```

Метод `hashCode()` переопределён.

```
@Override
public int hashCode() {
    return Objects.hash(tankFullnessRate, consumptionRate);
}
```

И сравнение объектов реализовано очень просто. В нём с помощью метода `hash()` класса Objects считается hash от всех переменных текущего объекта. То есть если два объекта идентичны по переменным `tankFullnessRate`, `consumptionRate`, то и hashCode у них будет одинаковым.

При этом может быть так, что другой набор данных объекта вернёт тот же hashCode, но сами объекты будут разными.

Закрепим. Метод `hashCode()` работает следующим образом: если hashCode объектов равны, эти объекты, возможно, одинаковые. Если же hashCode объектов различны, то объекты точно разные.

Следующий метод класса Object — `finalize()`. Он с версии Java 9 признан устаревшим (deprecated), и использовать его больше не рекомендуется. Тем не менее на рынке, особенно в крупных компаниях, обновление версий Java происходит довольно медленно — вы ещё можете столкнуться с `finalize()` в работе.

Этот метод вызывается, когда Java-машина удаляет из памяти объекты. При наследовании `finalize()` можно было бы использовать для освобождения ресурсов или удаления того, что стало ненужным после удаления объекта.

Ещё один метод класса Object — `getClass()`. Он возвращает класс текущего объекта. Классы в Java тоже представлены объектами (Class).

Вернёмся к методу `equals()`, который был автоматически сгенерирован в классе Bus:

```
@Override
public boolean equals(Object o) {
   if (this == o) return true;
   if (o == null || getClass() != o.getClass()) return false;
   Bus bus = (Bus) o;
   return Double.compare(bus.tankFullnessRate,
tankFullnessRate) == 0
           && Double.compare(bus.consumptionRate,
consumptionRate) == 0;
}
```

Мы видим, что для сравнения классов текущего и переданного ему в качестве параметра объектов как раз используется этот метод.

Попробуем применить этот метод к объекту класса Bus. Создадим объект класса Class. Назовём его `objectClass`:

```
Bus bus = new Bus(0.001);
Class objectClass = bus.getClass();
```

И затем выведем, к примеру, имя класса:

```
System.out.println(objectClass.getName());
```

Получаем имя класса — Bus.

Таким образом, метод `getClass()` универсален для всех объектов и не нуждается в переопределении.

Метод класса Object, с которым вы уже наверняка сталкивались, — `toString()`. Он возвращает строковое представление объекта и может быть вызван у любого объекта, когда тот необходимо превратить в строку. Например,

когда нужно передать объект в качестве параметра конструкции `System.out.println()`.

Посмотрим, что этот метод возвращает по умолчанию.

Есть объект класса Bus:

```
Bus bus = new Bus(0.001);
```

Попробуем представить его в виде строки:

```
System.out.println(bus);
```

Это будет то же самое, как если мы напишем: `(bus.toString);`

В ответе увидим:

```
Bus@eda1cf72
```

Это, на первый взгляд, нечто непонятное.

На самом деле всё просто. Здесь указано имя класса, затем собака, затем hash объекта в шестнадцатеричном представлении. Текстовое представление мы можем сформировать сами:

```
System.out.println(bus.getClass().getName() + "@" +
        Integer.toHexString(bus.hashCode()));
```

Это то, что возвращает метод `toString()` по умолчанию. Если нужно, чтобы объекты класса имели более адекватное строковое представление, следует переопределить метод.

Сделаем это в классе Bus. В конце напишем toString и нажмём Enter. Среда разработки сразу предложит переменные, которые нужно вывести в строковом представлении объекта. Нажмём ОК. И получим почти полноценный метод `toString()`:

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

Испытаем его:

```
Bus{tankFullnessRate=0.0, consumptionRate=0.001}
```

Среда разработки подсвечивает метод серым цветом:

```
System.out.println(bus.toString());
```

Это говорит о том, что метод здесь не нужен — он вызывается автоматически.

Теперь легко посмотреть параметры каждого автобуса. Это удобно не только при написании кода, но и при его отладке.

С отладкой кода детально вы познакомитесь позже. Сейчас отметим, что, когда вы отлаживаете код, вы можете приостанавливать программу на любой строке. В этот момент можно просматривать значения всех переменных, которые были инициализированы. Все объекты отображаются в виде строк. Если строковые представления сделаны адекватно, то и отлаживать код становится проще.

Это, конечно, не означает, что вам нужно во всех классах переопределять метод `toString()`, но теперь вы знаете, для чего он нужен и в каких случаях его переопределение может быть важно.

Ещё одна группа методов — методы для синхронизации потоков в многопоточных приложениях. Всего их пять. Первые два:

- `notify();`
- `notifyAll()`.

Три других — это методы `wait()`:

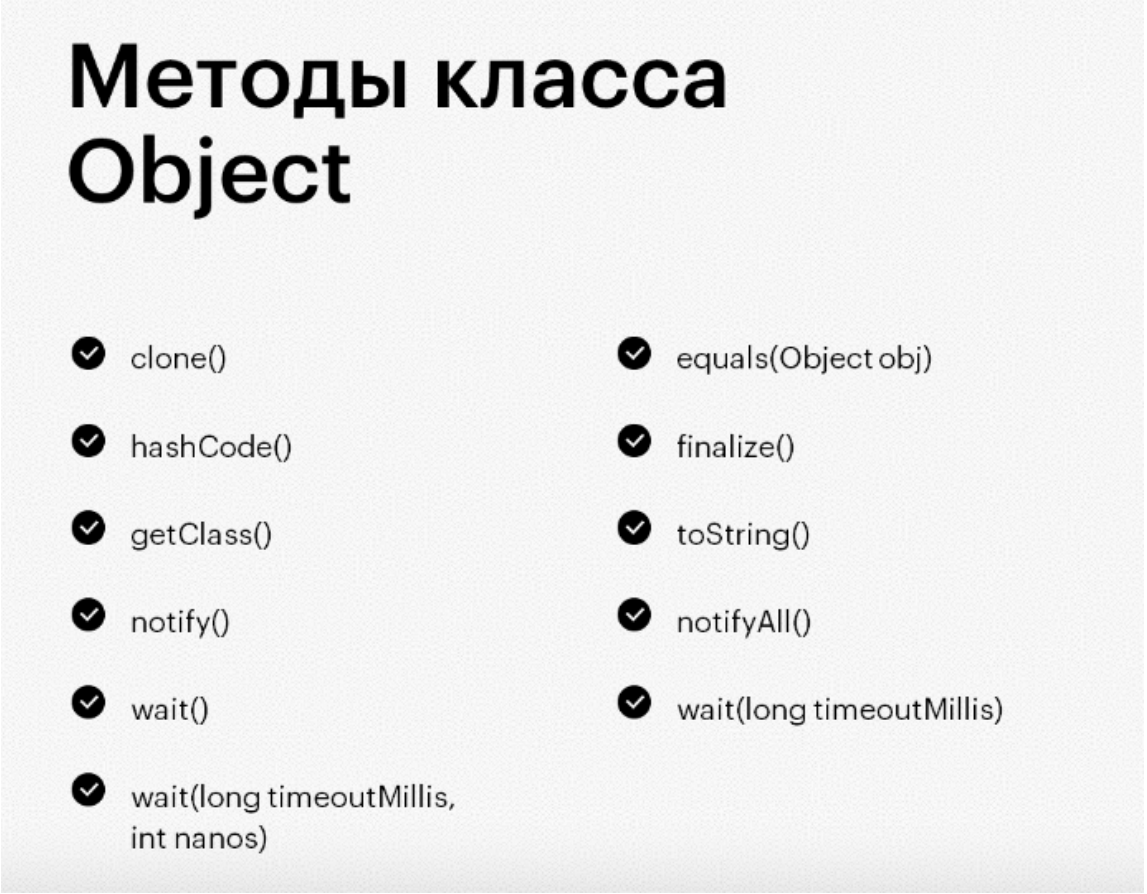
- `wait();`
- `wait(long timeoutMillis);`
- `wait(long timeoutMillis, int nanos).`

Первый метод `wait()` без параметров, во втором указывается количество миллисекунд, в третьем — количество милли- и наносекунд.

Как работают эти методы, для чего используются и как правильно применяются, вы разберёте в следующих модулях.

Всего у класса Object 11 методов.

Итак, вы узнали, что все классы в Java наследуются от класса Object, который имеет 11 методов. И соответственно, эти методы есть и у остальных классов в Java.



Изображение — Skillbox