5. class Object

Самый главный класс языка Java, который является родителем всех классов, создаваемых разработчиками или уже существующих, — это `class Object` .

Хоть в объявлениях классов это и не указано, все они неявно наследуют класс `Object` и реализуют его методы. Некоторые из которых можно переопределить.

Методы класса Object в Java

- `protected Object clone()` создает новый объект (клон) на основе существующего.
- `public boolean equals(Object obj)` определяет, равен ли один объект другому.
- `protected void finalize()` вызывается сборщиком мусора перед удалением неиспользуемого объекта. Со сборщиками вы познакомитесь в будущих уроках.
- `public final Class<?> getClass()` получает класс объекта во время выполнения.
- `public int hashCode()` возвращает хеш-код объекта.
- `public String toString()` возвращает строковое представление объекта.
- `public final void notify()` возобновляет исполнение потока, ожидающего вызывающий объект. Это один из механизмов многопоточного программирования, с которым вы познакомитесь в будущем.
- `public final void notifyAll()` возобновляет исполнение всех потоков, ожидающих вызывающий объект. Это один из механизмов многопоточного программирования, с которым вы познакомитесь в будущем.

`public final void wait()` ожидает другого потока исполнения. Это один из механизмов многопоточного программирования, с которым вы познакомитесь в будущем.

`public final void wait(long timeout)` ожидает другого потока исполнения. Это один из механизмов многопоточного программирования, с которым вы познакомитесь в будущем.

`public final void wait(long timeout, int nanos)` ожидает другого потока исполнения. Это один из механизмов многопоточного программирования, с которым вы познакомитесь в будущем.

Далее рассмотрим более подробно некоторые из них.

Методы equals() и hashCode()

Метод `equals()` сравнивает объекты между собой и возвращает результат типа `boolean`. Данный метод необходимо переопределять в своих классах. По умолчанию этот метод сравнивает объекты через оператор `==`, и мы можем получить `true`, только если обе ссылки указывают на один и тот же объект.

Реализовывать данный метод в своих классах нужно вместе с другим методом класса `Object` — `hashCode()`, который возвращает хеш-код объекта (уникальный числовой код любого объекта).

Между методами `equals()` и `hashCode()` существует контракт:

Если два объекта равны (т. е. метод `equals()` возвращает `true`), у них должен быть одинаковый хеш-код.

Если метод `hashCode()` вызывается несколько раз на одном и том же объекте, каждый раз он должен возвращать одно и то же число.

Хеш-код может быть одинаковым у двух разных объектов.

Последний пункт связан с тем, что хеш-код ограничен 32 битами. При этом количество создаваемых объектов не ограничено ничем, кроме объема памяти, доступного приложению.

Переопределить оба метода можно автоматически в среде разработки или вручную, следуя следующему примеру:

```
import java.util.Objects;

class Book {

    private String name;
    private String authorName;

    ... // Геттеры и сеттеры


    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Book book = (Book) o;
        return Objects.equals(name, book.name) && Objects.equals(authorName, book
    }

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

Метод toString()

Для корректной работы программы зачастую необходимо использовать строковое представление того или иного объекта.

Для этой цели в Java в классе

`Object` есть метод `toString()`. Переопределять данный метод в Java принято в собственных классах, чтобы строковое представление содержало в себе все необходимые данные.

```
public class Human {
    private String name;
    private int age;

    ...
    @Override
    public String toString() {
        return "Human{"
            + "name='" + name + '\''
            + ", age=" + age
            + '}';
    }
}
```

Если мы передаем в параметр метода `System.out.println()` экземпляр класса, к нему по умолчанию неявно вызывается метод `toString()`.

```
public class HumanTest {
    public static void main(String[] args) {
        Human human = new Human("Смирнов Роман Сергеевич", 23);
        System.out.println(human);
    }
}
```

Результат выполнения программы будет такой:

```
Human{name='Смирнов Роман Сергеевич', age=23}
```