нетология

Типы данных в Java: объекты

Филипп
Воронов

Филипп Воронов

Teamlead, VK Group

@Филипп Воронов

План занятия

1. Ссылочные типы данных
2. Структура объекта
3. Методы `Object`
4. Тип данных `String`
5. Тип данных `Enum`
6. Сравнение объектов

Ссылочные типы данных

Типы данных в Java

- примитивные (Primitive Data Types);
- **ссылочные (Reference Types).**

Ссылочные типы

Помимо примитивных типов в Java есть ссылочные типы, которые описываются в java-файлах. Пример создания ссылки на объект:

```
Object example1 = null;
String example2 = null;
```

Где:

- `Object` и `String` — это тип создаваемой ссылки;
- `example1` и `example2` — это имена ссылок, по которым мы сможем дальше обращаться к этим объектам в программе;
- `null` — ключевое слово в Java обозначающее отсутствие, то есть фактически `example1` и `example2` — это ссылки на несуществующие объекты.

На основе этих файлов с помощью специальных конструкторов создаются объекты. Каждый класс по умолчанию является наследником класса `Object`. Исключая примитивные типы, все в Java является объектом.

Пример создания объекта

```
Object myNewObject = new Object();
```

где:

- `Object` — это тип ссылки `myNewObject`;
- `myNewObject` — это имя ссылки на объект, по которой мы можем обращаться к созданному объекту;
- `new` — специальный оператор, вызываемый для создания нового объекта из конструктора;
- `Object()` — специальный метод класса `Object`, называемый конструктор и необходимый для создания объекта и выделения для него памяти.

Пример описания объекта

Любой объект описывается классом, который, в свою очередь, описывается в одноименном java-файле. Пример: `Car.java`

```
package ru.netogoly;

public class Car {

    private String modelName;

    public Car() {
    };

    public Car(String name) {
        this.modelName = name;
    }

    public String getModelName() {
        return this.modelName;
    }
}
```

Имя java-файла должно совпадать с именем класса, задекларированного (описанного) в нем. Класс `Tesla` должен находиться в файле `Tesla.java`.

Описание примера

```java
// пакет текущего класса
package ru.netogoly;

// имя класса (должно совпадать с именем файла Car.java)
public class Car {

    // поле класса (строковое значение)
    private String modelName;

    // первый конструктор класса без параметра (создается всегда, если не указан ни один конструктор)
    public Car() {
    };

    // второй конструктор класса с параметром, чтобы его вызвать, нужно написать new Car("Tesla model S")
    public Car(String name) {

        // сохранение входного строкового параметра в поле modelName, где this — ссылка на текущий объект
        this.modelName = name;
    }

    // описание метода, возвращающего поле modelName
    public String getModelName() {
        return this.modelName;
    }
}
```

Методы класса `Object`

Неважно, какой класс мы написали, каждый созданный из него объект по умолчанию получает следующий список методов:

- `Object clone()` создает новый объект, не отличающийся от клонируемого;
- `boolean equals(Object obj)` определяет, равен ли один объект другому;
- `void finalize()` вызывается перед удалением неиспользуемого объекта;
- `Class<?> getClass()` получает класс объекта во время выполнения;
- `int hashCode()` возвращает хеш-код, связанный с вызывающим объектом;

Методы класса `Object`

- `void notify()` возобновляет выполнение потока, которого ожидает вызывающий объект;
- `void notifyAll()` возобновляет выполнение всех потоков, которых ожидает вызывающий объект;
- `String toString()` возвращает строку, описывающую объект;
- `void wait()` бесконечно долгое время ожидает другой поток выполнения;
- `void wait(long millis)` определенное время ожидает другой поток выполнения;
- `void wait(long millis, int nanos)` определенное время ожидает другой поток выполнения.

Вопрос

Как вы думаете, какой самый наиболее используемый класс в Java?

Ответ

Класс-строка `String`

Тип данных `String`

Тип данных `String`

Самым часто используемым классом практически для любого языка являются строки в Java. Они представлены классом `String`, мы еще не раз к ним вернемся, а сейчас узнаем, как их создавать.

```
String message = "Hello world";
```

Так же строку можно создать как объект через конструктор класса:

```
String message = new String("Hello world");
```

Строки можно складывать:

```
String s1 = "Hello";
String s2 = "world";
String result = s1 + " " + s2; //  "Hello world"
```

Тип данных `String`

Каждая строка — это массив символов, у которого можно получить длину и каждый элемент:

```
String text = "Welcome to netology";
char symbol = text.charAt(0);   // W
int textLength = text.length(); // 19
```

Сравнение строк:

```
String s1 = "Welcome to netology";
String s2 = "Welcome to netology";

// Неправильный способ сравнения строк
System.out.println(s1 == s2);

// Правильный способ сравнения строк
System.out.println(s1.equals(s2));
```

Тип данных `String`

Строки — это ссылочный тип данных, а все ссылочные типы сравниваются только через метод `equals`. Проверка вхождения

```
String text1 = "We are looking";

// проверка вхождения подстроки в строку
text1.contains("looking"); // true

// проверка, с какой подстроки начинается строка
text1.startsWith("We"); // true

// проверка, на какую подстроку заканчивается строка
text1.endsWith("are"); // false
```

Подробнее я рекомендую почитать статью в официальной документации:
https://docs.oracle.com/javase/8/docs/api/java/lang/String.html
Чтобы что-то изучить, нужно прочитать об этом в разных источниках, посмотреть с разных сторон.

Вопрос

Как вы думаете, правильное ли следующее сложение строк (конкатенация)?

```
String value1 = "The lesson";
String value2 = "about";
String value3 = "objects";

String result = value1 + " " + value2 + " " + value3;
```

Ответ

Такое сложение строк допускается, но создает несколько дополнительных объектов в памяти JVM. Чтобы избежать создания дополнительных объектов, нужно использовать специальные классы `StringBuilder` и `StringBuffer`. Пример решения:

```
StringBuilder result = new StringBuilder();
result.append(value1);
result.append(value2);
result.append(value3);

// Для получение результирующей строки нужно обязательно вызвать
метод toString();
result.toString();
```

Тип данных `Enum`

Тип данных `Enum`

`Enum` — это специальный тип перечисления именованных констант. Они могут иметь конструкторы, методы и переменные экземпляра.

Пример:

```
enum Season {
    AUTUMN, WINTER, SPRING, SUMMER;
}

System.out.println(Season.AUTUMN);
```

Обертки над примитивами

Каждый примитивный тип имеет аналог ссылочного типа. Они называются классы-оболочки (wrappers):

```
short – Short
byte – Byte
int – Integer
long – Long
float – Float
double – Double
char – Character
boolean – Boolean
```

Такие классы нужны для расширения функционала примитивных типов, например, для преобразования строки в число `Integer.parseInt("1901");`

Обертки над примитивами

Сравнение объектов Все объекты сравниваются через вызов метода `equals`. Рассмотрим пример:

```
Object a = new Object();
Object b = new Object();
Object c = a; // присваиваем ссылке `c` объект по ссылке `a`

System.out.println(a == b); // сравниваем ссылки на объекты
System.out.println(a.equals(b)); // сравниваем объекты
```

`==` сравнивает ссылки на объекты, а они будут равны только в одном случае, если ссылка ведет на один и тот же объект.

Чему мы научились

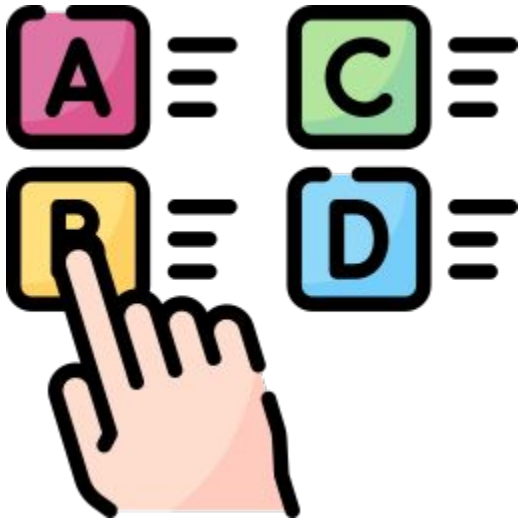
1. Узнали, что такое ссылочные типы;
2. Познакомились со структурой и методами объекта;
3. Научились создавать строки;
4. Поняли, как искать подстроки в строке;
5. Посмотрели, как сравнивать объекты.

Домашнее задание

Закрепите тему сегодняшней лекции — пройдите **квиз**!

В квизе вас ждут:

- пояснения к каждому варианту ответа,
- неограниченное количество попыток.



нетология

Задавайте вопросы и пишите отзыв о лекции!

Филипп Воронов

@Филипп Воронов