Лекция №3

Основы java, часть 2

Бобряков Дмитрий
Senior разработчик

Меня хорошо видно и слышно?

Проверить, идет ли запись

Ставим «+», если все хорошо
«-», если есть проблемы

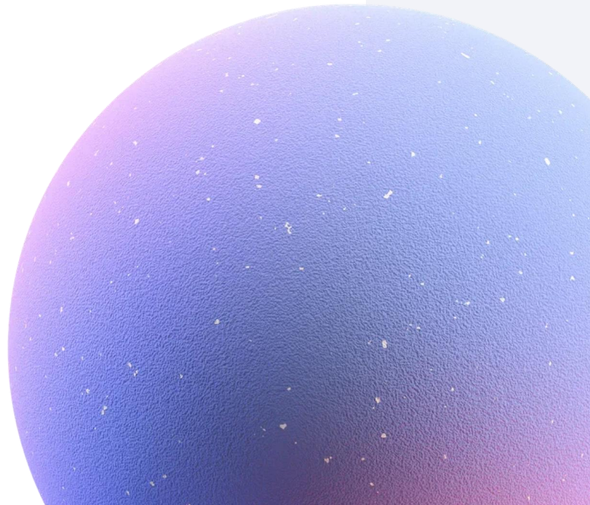
Дмитрий Бобряков

Senior разработчик

- 7 лет в IT
- Senior-разработчик в BigData Streaming
- Experience: NetCracker, AlphaBank, MWS
- Java-ментор в MWS
- Преподаватель в МФТИ

Содержание лекции

- Record
- Sealed class
- Массивы
- String
- Exceptions
- OOP



Record

```
public record RecordCar(
        String brand,
        int year,
        String color) implements Vehicle {

    @Override
    public void print() {
    }
}
```

- → is a restricted form of a class
- → It’s ideal for "plain data carriers"
- → for classes that contain data not meant to be altered
- → have only the most fundamental methods such as constructors and accessors

- → final
- → private
- → full constructor
- → getters
- → ~~setters~~
- → equals & hashCode
- → toString
- → ~~extends~~

Sealed class

```
public abstract sealed class SealedCar permits
Toyota {
}


public final class Toyota extends SealedCar {
}
```

Array

Эти фундаментальные структуры кажутся простыми, но содержат множество нюансов, которые критически важны для высоконагруженных систем.

Массивы и строки — это кирпичики, из которых строятся практически все Java-приложения.

Array

```
// Различные способы объявления и инициализации
int[] numbers = new int[10];                    // Все элементы 0

String[] names = new String[5];                 // Все элементы null

boolean[] flags = {true, false, true};          // Инициализация при объявлении
```

Array

```
 int[] arr = new int[5];
      // Длина массива фиксирована и доступна через
поле length

System.out.println("Длина: " + arr.length); // 5

arr[0] = 10; // Индексация с 0
arr[4] = 50; // Последний элемент

// Попытка доступа к arr[5] выбросит
ArrayIndexOutOfBoundsException
```

Array

```
 // Двумерный массив (массив массивов)
int[][] matrix = new int[3][4];

matrix[0][0] = 1;

matrix[2][3] = 12;
```

Array

```
// Инициализация при объявлении

        int[][] predefined = {
            {1, 2, 3},
            {4, 5},
            {6, 7, 8, 9}
        };
```

Array

```
 int[] numbers = {3, 1, 4, 1, 5, 9, 2, 6};

        // Сортировка
Arrays.sort(numbers); // Быстрая сортировка Dual-Pivot Quicksort

        // Бинарный поиск (только на отсортированных массивах)
int index = Arrays.binarySearch(numbers, 5);

        // Сравнение массивов
int[] copy = Arrays.copyOf(numbers, numbers.length);
boolean equal = Arrays.equals(numbers, copy); // true
```

Array

```
        // Заполнение
int[] filled = new int[10];
Arrays.fill(filled, 42); // Все элементы становятся
42

        // Глубокие операции для многомерных массивов
int[][] matrix1 = {{1, 2}, {3, 4}};
int[][] matrix2 = {{1, 2}, {3, 4}};
boolean deepEqual = Arrays.deepEquals(matrix1,
matrix2); // true

        // Стримы из массивов (Java 8+)
int sum = Arrays.stream(numbers).sum();
```

String

Строки в Java — это иммутабельные объекты на основе char[]

String

```
String str = "Hello";

Иммутабельность означает:
1. Поля final
2. Нет сеттеров
3. Все операции возвращают новые объекты
```

String

```
String str = "  Hello World  ";

        // Базовые операции
int length = str.length(); // 15 (включая пробелы)
char ch = str.charAt(1);   // ' '
String substr = str.substring(6, 11); // "World"

        // Поиск
int index = str.indexOf("World"); // 6
boolean contains = str.contains("Hello"); // true
```

String

```
        // Модификации (возвращают новые строки)
String trimmed = str.trim(); // "Hello World"
String upper = str.toUpperCase(); // "  HELLO WORLD  "
String replaced = str.replace("World", "Java"); // "  Hello
Java  "

        // Разделение и склейка
String[] parts = "a,b,c".split(","); // ["a", "b", "c"]
String joined = String.join("-", "a", "b", "c"); // "a-b-c"
```

String

```
char[] chars = {'H', 'e', 'l', 'l', 'o'};

String str1 = new String(chars); // "Hello"
String str2 = String.valueOf(chars); // "Hello"
```

String

```
        // Обратная конвертация
char[] chars2 = str1.toCharArray();

        // Работа с отдельными символами
for (char c : str1.toCharArray()) {
     System.out.println(c);
}
```

Exceptions

Исключения — это не просто ошибки, это механизм коммуникации между различными слоями приложения.

Основные принципы:
- Исключения должны быть информативными и "действующими"
- Обработка на том уровне, где можно предпринять осмысленные действия
- Не использовать исключения для контроля потока выполнения
- Логгировать на том уровне, где исключение возникает впервые

Checked vs Unchecked исключения

```
// Checked исключения - должны быть обработаны или объявлены
public void readFile() throws IOException {
    // Компилятор заставляет обработать это исключение
}
```

Checked vs Unchecked исключения

```
// Unchecked исключения (RuntimeException) - не обязательны для обработки
public void divide(int a, int b) {
    if (b == 0) {
        throw new IllegalArgumentException("Divisor cannot be zero");
    }
}
```

Error vs Exception

```
// Error - системные ошибки, которые обычно не должны перехватываться
public class CriticalSystemError extends Error {
    public CriticalSystemError(String message) {
        super(message);
    }
}
```

Error vs Exception

```
// Exception - базовый класс для исключений приложения
public class BusinessException extends Exception {
    public BusinessException(String message) {
        super(message);
    }
}
```

Обработка исключений

```
public class DatabaseService {
    public void processTransaction(Connection conn) {
        try {
            // Код, который может бросить исключение
            startTransaction(conn);
            executeBusinessLogic(conn);
            commitTransaction(conn);
        } catch (SQLException e) {
            // Специфичная обработка SQL ошибок
            rollbackTransaction(conn);
            logger.error("Database error during transaction", e);
            throw new TransactionException("Transaction failed", e);
        } catch (BusinessException e) {
            // Обработка бизнес-логики ошибок
            rollbackTransaction(conn);
            logger.warn("Business rule violation", e);
            throw e;
        } finally {
            // Всегда выполняется (даже при return или исключении)
            closeResources(conn);
        }
    }
}
```

Обработка исключений

```
// Новый стиль (автоматическое закрытие)
public void modernStyle() {
    try (Connection conn = dataSource.getConnection();
         PreparedStatement stmt = conn.prepareStatement("SELECT
* FROM users");
         ResultSet rs = stmt.executeQuery()) {
...

} catch (SQLException e) {
        throw new DataAccessException("Database operation
failed", e);
    }
    // Ресурсы автоматически закрываются в обратном порядке
}
```

Обработка исключений

```
public void handleMultipleExceptions() {
    try {
        processData();
    } catch (IOException | SQLException | NetworkException
e) {
        // Общая обработка для разных типов исключений
        logger.error("Operation failed", e);
        throw new ApplicationException("Failed to process
data", e);
    }
}
```

Создание исключений

```
// Бизнес-исключение с дополнительным контекстом

public class PaymentException extends Exception {

    private final String transactionId;
    private final BigDecimal amount;
    private final String reasonCode;

}
```

Создание исключений

```
// Бизнес-исключение с дополнительным контекстом

public class PaymentException extends Exception {

    private final String transactionId;
    private final BigDecimal amount;
    private final String reasonCode;

}
```

ООР

Представьте, что вы описываете мир вокруг себя в программе. ООП — это способ думать о программе как о наборе объектов, которые взаимодействуют друг с другом.

Простая аналогия:
- Класс — это чертеж дома (описание)
- Объект — это реальный дом, построенный по чертежу
- Методы — это то, что дом может делать (открыть дверь, включить свет)
- Поля — это характеристики дома (цвет, количество окон)

ООР

Основная философская идея ООП заключается в том, что программные системы должны моделировать реальный мир через объекты, которые сочетают в себе состояние (данные) и поведение (методы).
Такой подход позволяет создавать более интуитивно понятные и поддерживаемые системы.

ООР

Три кита ООП:

- Инкапсуляция
- Наследование
- Полиморфизм

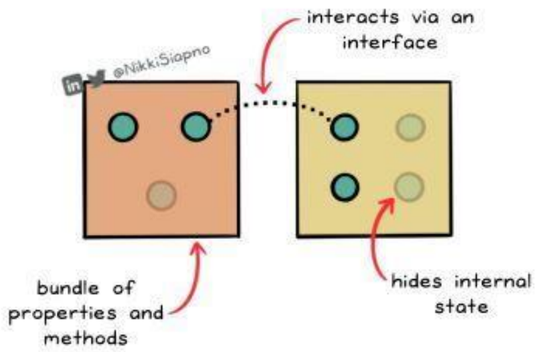
Дополнительно выделяют:

- Абстракция

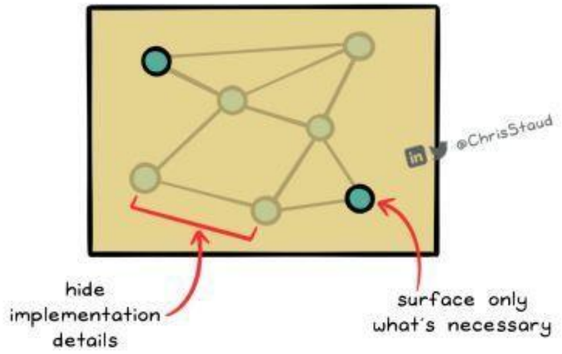
Principles of OOP

by levelupcoding.co

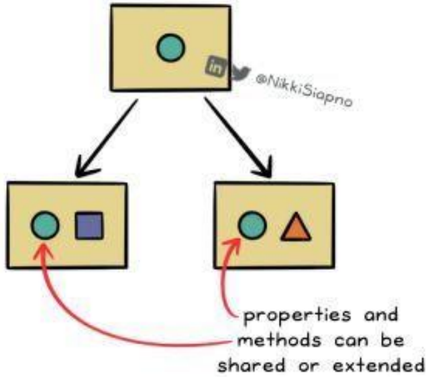
Encapsulation



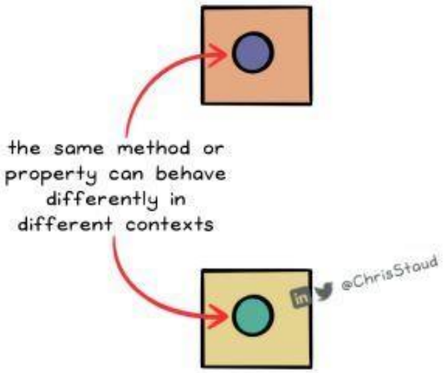
Abstraction



Inheritance



Polymorphism



Let me know you like it

@NikkiSiapno @ChrisStaud

Share it with your friends

Инкапсуляция

```
class Book {
    private String title;
    private String author;
    private int year;
}

class Library {
    private Book[] books;
    private int count;
}
```

Инкапсуляция (сокрытие)

Modifier	Class	Package	Subclasses	World
public	+	+	+	+
protected	+	+	+	-
default	+	+	-	-
private	+	-	-	-

Наследование

```
class Shape {
    String color;

    void draw() {
        System.out.println("Рисую фигуру цвета " +
color);
    }
}
```

Наследование

```
class Circle extends Shape {
    double radius;

    @Override
    void draw() {
        System.out.println("Рисую круг радиусом "
+ radius + " цвета " + color);
    }
}
```

Наследование

```
class Rectangle extends Shape {
    double width;
    double height;

    @Override
    void draw() {
        System.out.println("Рисую прямоугольник " +
width + "x" + height + " цвета " + color);
    }
}
```

Полиморфизм
Абстрактный класс и интерфейс

```
public class Car extends AbstractCar implements Vehicle {

    @Override
    public void print() {
        System.out.println("Бренд: " + this.brand);
        System.out.println("Год выпуск: " + this.year);
        System.out.println("Цвет: " + this.color);
    }
```

Полиморфизм
Абстрактный класс и интерфейс

Три столпа ООП не существуют изолированно — они образуют взаимосвязанную систему:

- Инкапсуляция создает границы объектов
- Наследование устанавливает отношения между объектами
- Полиморфизм обеспечивает гибкое взаимодействие через эти отношения

ООР в распределенных системах

В микросервисной архитектуре принципы ООП находят новое применение:

- Инкапсуляция реализуется через границы сервисов
- Полиморфизм проявляется в различных реализациях сервисов
- Наследование заменяется композицией сервисов

ООР

ООП — это крайне сложно! Но главное понять основные идеи:

1. Класс — это чертеж, объект — конкретная вещь
2. Инкапсуляция — прячем данные, даем методы для работы с ними
3. Наследование — создаем новые классы на основе существующих
4. Полиморфизм — разные объекты могут делать одно и то же по-разному

MTC True Tech

СПАСИБО ЗА ВНИМАНИЕ

Дмитрий Бобряков

Senior big data developer

dmitrybobryakov@gmail.com

@DmitryBobryakov