
Яндекс

Яндекс

Зачем учить Java и как делать это эффективно

Затепякин Михаил, разработчик Яндекс.Маркета

Почему стоит учить джаву?

Java — один из самых распространенных языков

Oct 2019	Oct 2018	Change	Programming Language	Ratings	Change
1	1		Java	16.884%	-0.92%
2	2		C	16.180%	+0.80%
3	4	▲	Python	9.089%	+1.93%
4	3	▼	C++	6.229%	-1.36%
5	6	▲	C#	3.860%	+0.37%
6	5	▼	Visual Basic .NET	3.745%	-2.14%
7	8	▲	JavaScript	2.076%	-0.20%

Предназначение языков программирования

Разные языки программирования решают разные задачи

- Python — язык с минималистичным синтаксисом, идеален для написания скриптов
- C++ — полный контроль над исполняемым кодом

Какие задачи решает Java?

Java — язык для разработки больших систем

- Упор на читаемость, простоту кода и его эффективность
- Объектно-ориентированный язык
- Автоматический сборщик мусора
- Развитые фреймворки
- Огромное количество библиотек для работы со всем, что может пригодиться на бекенде

Основные скилы начинающего джависта

- Java core
- Dependency Injection фреймворк
- Архитектуры и паттерны проектирования
- SQL / ORM (для бекенда)

Java core

- Обязательно учим одну из последних версий Java
- Внимательно смотрим на Java Stream API, var e.t.c.
- На собеседованиях любят спрашивать про Exceptions, итераторы и прочие вещи, которые на первый взгляд кажутся неважными

Структуры данных

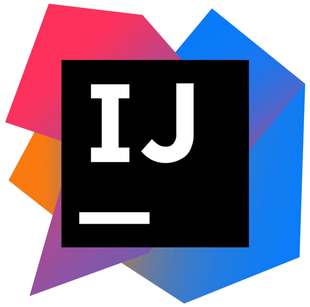
- Бывают разные реализации одних и тех же типов данных, например, HashMap и TreeMap()
- Здорово знать, как работают структуры внутри, например, что такое бакет в HashMap, а не просто их асимптотики
- Обратить внимание на деревья и графы, которых практически нет в production-коде, но есть на собеседованиях

Без вспомогательных систем никуда

- Maven / Gradle

- Git

- IntelliJ Idea



SQL

SQL нужен вообще на любом бекенде, независимо от языка

- SELECT, JOIN, Subqueries https://sqlzoo.net
- Ключи, индексы, нормализация https://habr.com/ru/post/193136

ORM (JPA, Hibernate e.t.c.)

```
private static final String FIND_BY_USER_IDS_QUERY = ""
        + "SELECT "
        + "  id, "
        + "  name "
        + "FROM info.users "
        + "WHERE id IN (:userIds)";

private static final RowMapper<User> USER_ROW_MAPPER = (rs, rowNum) ->
        new User(rs.getInt("id"), rs.getString("name"));

public List<User> findByUserIds(Collection<Integer> userIds) {
    var params = new MapSqlParameterSource("userIds", userIds);
    return postgresJdbcTemplate.query(
            FIND_BY_USER_IDS_QUERY,
            params,
            USER_ROW_MAPPER
    );
}
```

```
List<User> findAllByIdIn(Collection<Integer> userIds);
```

Spring

- Без Dependency Injection фреймворков создавать большие проекты на Java фактически невозможно
- Spring Boot поднимает серверное приложение «из коробки»
- Позволяет писать методы REST API с помощью одной аннотации
- Упрощает написание тестов

Тестирование

Без тестов серьёзная разработка невозможна

- JUnit 5
- Mockito

Паттерны проектирования

Шаблоны для решения типичных проблем

- https://refactoring.guru
- У каждого паттерна существует множество реализаций
- Поначалу получится применять буквально пару паттернов вроде Builder и Singletone
- Без практики знание паттернов бесполезно

Зачем нужны паттерны?

```
public class User {

    private final int id;
    private final String name;

    public User(int id, String name) {
        this.id = id;
        this.name = name;
    }
}
```

```
User user = new User(7, "Bond");
```

```
public class User {

    private int id;
    private String name;

    public void setId(int id) {
        this.id = id;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

```
User user = new User();
user.setId(7);
user.setName("Bond");
```

Пример реализации паттерна Builder

```java
public class User {

    private final int id;
    private final String name;

    private User(Builder builder) {
        this.id = builder.id;
        this.name = builder.name;
    }

    public static class Builder {
        private int id;
        private String name;

        public Builder id(int id) {
            this.id = id;
            return this;
        }

        public Builder name(String name) {
            this.name = name;
            return this;
        }

        public User build() { return new User(this); }
    }

}
```

```java
User user = new User.Builder()
        .id(7)
        .name("Bond")
        .build();
```

```java
User user = new User.Builder()
        .name("Bond")
        .build();
```

```java
User user = new User.Builder()
        .id(7)
        .build();
```

Пример реализации паттерна Builder

```java
public class User {

    private final int id;
    private final String name;

    private User(Builder builder) {
        this.id = builder.id;
        this.name = builder.name;
    }

    public static class Builder {
        private Integer id;
        private String name;

        public Builder id(Integer id) {
            this.id = id;
            return this;
        }

        public Builder name(String name) {
            this.name = name;
            return this;
        }

        public User build() { return new User(this); }
    }

}
```

```java
User user = new User.Builder()
        .id(7)
        .name("Bond")
        .build();
```

```java
User user = new User.Builder()
        .name("Bond")
        .build();
```

```java
User user = new User.Builder()
        .id(7)
        .build();
```

Пример реализации паттерна Builder

```
public class User {

    private final int id;
    private final String name;

    private User(Builder builder) {
        this.id = Objects.requireNonNull(builder.id, "id");
        this.name = Objects.requireNonNull(builder.name, "name");
    }

    public static class Builder {
        private Integer id;
        private String name;

        public Builder id(Integer id) {
            this.id = id;
            return this;
        }

        public Builder name(String name) {
            this.name = name;
            return this;
        }

        public User build() { return new User(this); }
    }

}
```

```
User user = new User.Builder()
        .id(7)
        .name("Bond")
        .build();
```

```
User user = new User.Builder()
        .id(7)
        .build();
```

```
User user = new User.Builder()
        .id(null)
        .name("Bond")
        .build();
```

Пример реализации паттерна Builder

```
public class User {

    private final int id;
    private final String name;

    private User(Builder builder) {
        this.id = Objects.requireNonNull(builder.id, "id");
        this.name = Objects.requireNonNull(builder.name, "name");
    }

    public static class Builder {
        private Integer id;
        private String name;

        public Builder id(int id) {
            this.id = id;
            return this;
        }

        public Builder name(String name) {
            this.name = name;
            return this;
        }

        public User build() { return new User(this); }
    }

}
```

```
User user = new User.Builder()
        .id(7)
        .name("Bond")
        .build();
```

```
User user = new User.Builder()
        .id(7)
        .build();
```

```
User user = new User.Builder()
        .id(null)
        .name("Bond")
        .build();
```

Пример реализации паттерна Builder с читами

```java
import lombok.Builder;

@Builder
public class User {

    private int id;
    private String name;

}
```

```java
User user = User.builder()
        .id(7)
        .name("Bond")
        .build();
```

Project Lombok

Spice up your Java

Архитектурные паттерны

Single Responsibility Principle

Пишем полноценный проект

- Работа с базой данных
- REST API
- Взаимодействие со сторонними сервисами с помощью API

Пример небольшого серверного приложения

Серверное приложение для организации мероприятий

- Авторизация через VK с использованием OAuth 2.0
- Получение аватарок и информации о пользователях из VK
- Сохранение информации о пользователях и мероприятиях в базе данных
- Сохранение картинок и файлов
- REST API для получения данных пользователями
- Unit-тесты всех методов с нетривиальной логикой

Clean Code + Effective Java

- DRY — Don’t Repeat Yourself
- KISS — Keep It Simple, Stupid
- YAGNI — You Ain’t Gonna Need It

- S — Single responsibility principle
- O — Open/closed principle
- L — Liskov substitution principle
- I — Interface segregation principle
- D — Dependency inversion principle

Про собеседования

На собеседованиях проверяют знание языка. Как правило, спрашивают следующие вещи

- Умение придумывать и реализовывать простые алгоритмы
- Применение разных структур данных и знание их асимптотик
- Exception handling
- Collections & Generics
- Java Stream API
- SQL (для бекендеров)
- HTTP

Пример успешного кейса изучения Java

Чтобы выучить джаву с нуля, нужно совсем немного времени

- На первом курсе изучал C# core — получил понимание основ ООП
- На втором курсе изучал Java core — научился писать рабочий код
- В качестве курсовой взял back-end сервер на Java Spring Boot
- Прошёл на собеседование в Яндекс — за пару дней выучил SQL
- Попал на стажировку — получил опыт enterprise разработки
- Остался после стажировки — пишу production-код в Яндекс.Маркете

Полезные ссылки

- Java 8. Руководство для начинающих. Герберт Шилдт
- Структуры данных - https://habr.com/ru/post/128017
- SQL - https://sqlzoo.net
- Нормализация баз данных - https://habr.com/ru/post/193136
- Паттерны проектирования - https://refactoring.guru
- Design Patterns (GoF)
- Clean Code
- Effective Java

Яндекс

Спасибо

Затепякин Михаил

Разработчик Яндекс.Маркета

ogonek@yandex-team.ru @ne_ogonek @ne_ogonek