



Curso - Java All in One 48

⌚ Created	@October 16, 2024 2:29 PM
🏷️ Tags	back-end curso livre
☑️ Concluido	<input type="checkbox"/>
👤 Created by	D Douglas Leite

Complete Java, Spring, and Microservices course
Java developer in 48 hours; technically, it will take months to master.
00:00:00 Core Java
 https://youtu.be/4XTsAAHW_Tc?si=u4LndjaX6yMPanbE



▼ Java Core

▼ Java Basics -

▼ Aula 1 - Hello World

Complete Java, Spring, and Microservices course
Java developer in 48 hours; technically, it will take months to master.
 https://www.youtube.com/watch?v=4XTsAAHW_Tc&t=801s



Hello World compilado no Terminal com comandos \$javac e \$java

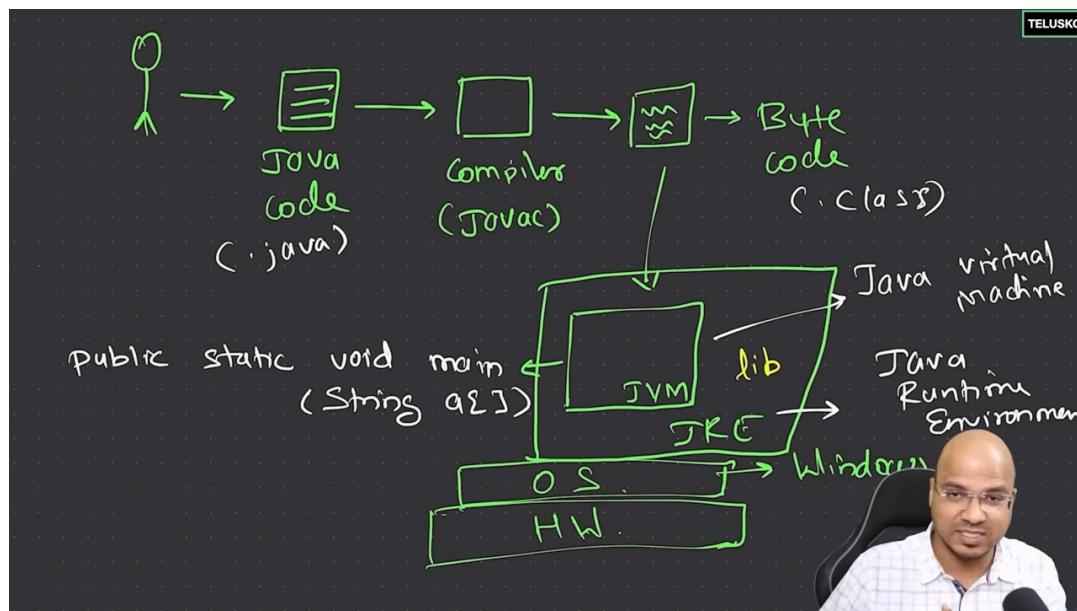


```
douglas:~/code$ cat HelloWorld.java
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
douglas:~/code$ javac HelloWorld.java
douglas:~/code$ java HelloWorld
Hello World!
```

▼ Aula 2 - Estrutura de Compilação e Execução

Complete Java, Spring, and Microservices course
Java developer in 48 hours; technically, it will take months
to master.

▶ https://www.youtube.com/watch?v=4XTsAAHW_Tc&t=1475s



▼ Aula 3 - Variáveis

Complete Java, Spring, and Microservices course
Java developer in 48 hours; technically, it will take months
to master.

▶ https://www.youtube.com/watch?v=4XTsAAHW_Tc&t=2194s



```
douglas:~/code$ cat Variables.java
public class Variables {
    public static void main(String[] args) {
        int num1 = 3;
        int num2 = 5;
        int result = num1 + num2;
        System.out.println(result);
    }
}
douglas:~/code$ javac Variables.java
douglas:~/code$ java Variables
8
```

▼ Aula 4 - Operações Matemáticas

Complete Java, Spring, and Microservices course
Java developer in 48 hours; technically, it will take months
to master.

▶ https://www.youtube.com/watch?v=4XTsAAHW_Tc&t=4640s



```
● ● ●
```

```
public class MathOperations {
    public static void main(String[] args) {
        int a = 7;
        int b = 5;

        int sum = a + b; // 12
        int subtraction = a - b; // 2
        int multiplication = a * b; // 35

        int quotient = a / b; // 1
        int remainder = a % b; // 2
        float division = (float) a / b; // 1.4

        int i = 1;
        int post_increment = i++; // post_increment = 1, i = 2
        int pre_increment = ++i; // pre_increment = 3, i = 3
    }
}
```

▼ Aula 5 - Comparações Lógicas

Complete Java, Spring, and Microservices course

Java developer in 48 hours; technically, it will take months to master.

▶ https://www.youtube.com/watch?v=4XTsAAHW_Tc&t=5252s



```

public class LogicComparisons {
    public static void main(String[] args) {
        int big = 10;
        int small = 1;
        int equal = 1;

        boolean greaterThan = small > big; // false
        greaterThan = big > small; // true
        greaterThan = small > equal; // false

        boolean lesserThan = small < big; // true
        lesserThan = big < small; // false
        lesserThan = small < equal; // false

        boolean equals = small == big; // false
        equals = small == equal; // true

        boolean different = small != big; // true
        different = small != equal; // false

        boolean greaterOrEqual = small >= big; // false
        greaterOrEqual = big >= small; // true
        greaterOrEqual = small >= equal; // true

        boolean lesserOrEqual = small <= big; // true
        lesserOrEqual = big <= small; // false
        lesserOrEqual = small <= equal; // true
    }
}

```

▼ Aula 6 - Operações Lógicas

Complete Java, Spring, and Microservices course
 Java developer in 48 hours; technically, it will take months
 to master.

▶ https://www.youtube.com/watch?v=4XTsAAHW_Tc&t=5735s



```

public class LogicOperations {
    public static void main(String[] args) {
        boolean trueValue = true;
        boolean falseValue = false;

        // AND (&&) operations
        boolean andOperation = trueValue && trueValue; // true
        andOperation = trueValue && falseValue; // false
        andOperation = falseValue && trueValue; // false
        andOperation = falseValue && falseValue; // false

        // OR (||) operations
        boolean orOperation = trueValue || trueValue; // true
        orOperation = trueValue || falseValue; // true
        orOperation = falseValue || trueValue; // true
        orOperation = falseValue || falseValue; // false

        // NOT (!) operations
        boolean notOperation = !trueValue; // false
        notOperation = !falseValue; // true
    }
}

```

01:54:20 — 28/10

▼ Aula 7 - If Else

Complete Java, Spring, and Microservices course
Java developer in 48 hours; technically, it will take months to master.

 https://www.youtube.com/watch?v=4XTsAAHW_Tc&t=6407s



```

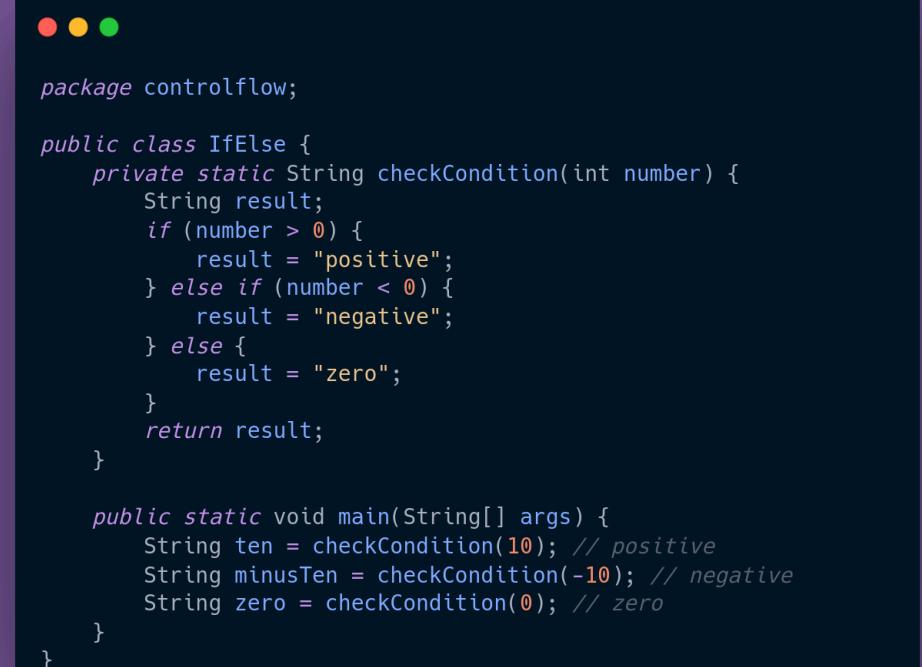
package controlflow;

public class IfElse {
    private static String checkCondition(int number) {
        String result;
        if (number > 0) {
            result = "positive";
        } else if (number < 0) {

```

```
        result = "negative";
    } else {
        result = "zero";
    }
    return result;
}

public static void main(String[] args) {
    String ten = checkCondition(10); // positive
    String minusTen = checkCondition(-10); // negative
    String zero = checkCondition(0); // zero
}
```



The screenshot shows a Java code editor with a dark theme. At the top, there are three colored circular icons: red, yellow, and green. The code itself is identical to the one above, with syntax highlighting for keywords like `package`, `private`, `if`, and `else`. The code editor has a clean, modern look with a white background and a dark border.

```
package controlflow;

public class IfElse {
    private static String checkCondition(int number) {
        String result;
        if (number > 0) {
            result = "positive";
        } else if (number < 0) {
            result = "negative";
        } else {
            result = "zero";
        }
        return result;
    }

    public static void main(String[] args) {
        String ten = checkCondition(10); // positive
        String minusTen = checkCondition(-10); // negative
        String zero = checkCondition(0); // zero
    }
}
```

▼ Aula 8 - Ternário

Complete Java, Spring, and Microservices course
Java developer in 48 hours; technically, it will take months to master.

▶ https://www.youtube.com/watch?v=4XTsAAHW_Tc&t=7518s



```
package controlflow;

public class Ternary {
    public static void main(String[] args) {
        String ten = (10 > 0) ? "positive" : "negative"; // positive
        String minusTen = (-10 > 0) ? "positive" : "negative"; // negative
        String zero = (0 == 0) ? "zero" : "non-zero"; // zero
    }
}
```

▼ Aula 9 - Switch Case

Complete Java, Spring, and Microservices course
Java developer in 48 hours; technically, it will take months to master.

▶ https://www.youtube.com/watch?v=4XTsAAHW_Tc&t=7791s



```
package controlflow;

public class SwitchCase {

    public static void main(String[] args) {
        int dayOfWeek = 3; // Let's say 1 = Monday, 2 = Tuesday, ..., 7 = Sunday
        String dayName;

        switch (dayOfWeek) {
            case 1:
                dayName = "Monday";
                break;
            case 2:
                dayName = "Tuesday";
                break;
        }
    }
}
```

```
        case 3:  
            dayName = "Wednesday";  
            break;  
        case 4:  
            dayName = "Thursday";  
            break;  
        case 5:  
            dayName = "Friday";  
            break;  
        case 6:  
            dayName = "Saturday";  
            break;  
        case 7:  
            dayName = "Sunday";  
            break;  
        default:  
            dayName = "Invalid day";  
            break;  
    }  
  
    String dayType;  
    switch (dayOfWeek) {  
        case 1:  
        case 2:  
        case 3:  
        case 4:  
        case 5:  
            dayType = "Weekday";  
            break;  
        case 6:  
        case 7:  
            dayType = "Weekend";  
            break;  
        default:  
            dayType = "Invalid day";  
            break;  
    }  
}
```

```
}
```

▼ Aula 10 - While

Complete Java, Spring, and Microservices course
Java developer in 48 hours; technically, it will take months
to master.

▶ https://www.youtube.com/watch?v=4XTsAAHW_Tc&t=8472s



```
package loops;

public class While {
    public static void main(String[] args) {
        String phrase = "I'm smaller";

        while (!phrase.isEmpty()) {
            System.out.println(phrase);

            int lastCharIndex = phrase.length() - 1;
            phrase = phrase.substring(0, lastCharIndex);
        }
    }
}

/*
 * Output:
 * I'm smaller
 * I'm smalle
 * I'm small
 * I'm smal
 * I'm sma
 * I'm sm
 * I'm s
 * I'm
 * I'm
 * I'
 * I
 */
```

▼ Aula 11 - Do While

Complete Java, Spring, and Microservices course
Java developer in 48 hours; technically, it will take months to master.

▶ https://www.youtube.com/watch?v=4XTsAAHW_Tc&t=9215s



```
package loops;

public class DoWhile {

    private static void doWhileCounter(int counter) {
        do {
            if (counter == 0) {
                System.out.println("Já é zero...");
            } else {
                System.out.println(counter + "...");
                counter--;
            }
        } while (counter > 0);
    }

    public static void main(String[] args) {
        doWhileCounter(3); // 3... 2... 1...
        doWhileCounter(0); // Já é zero...
    }
}
```

▼ Aula 12 - For

Complete Java, Spring, and Microservices course
Java developer in 48 hours; technically, it will take months to master.

▶ https://www.youtube.com/watch?v=4XTsAAHW_Tc&t=9408s



```
package loops;

public class For {
    public static void main(String[] args) {
```

```
for (int i = 1; i < 5; i++) {  
    System.out.println("Comente " + i + " vezes!");  
}  
/*  
 * Comente 1 vezes!  
 * Comente 2 vezes!  
 * Comente 3 vezes!  
 * Comente 4 vezes!  
 */  
  
for (int i = 4; i > 0; i--) {  
    System.out.println("Comente " + i + " vezes!");  
}  
/*  
 * Comente 4 vezes!  
 * Comente 3 vezes!  
 * Comente 2 vezes!  
 * Comente 1 vezes!  
 */  
}  
}
```

▼ POO
▼ POO Theory

Complete Java, Spring, and Microservices course
Java developer in 48 hours; technically, it will take months to master.

▶ https://www.youtube.com/watch?v=4XTsAAHW_Tc&t=10308s



Object Oriented Programming

Um Objeto possui Propriedades (dados, informações, coisas que o definem ou que ele sabe) e Comportamentos (coisas que faz, que pode fazer, que sabe fazer).

Uma Classe é uma planta (blueprint) de um objeto. Nós escrevemos a Classe como essa blueprint e a JVM, quando compila o código, utilizará essa Classe para criar os Objetos.

Com Minecraft

Objetos - Steve (personagem), Creeper, Árvore

X

Classes - Personagem, Creeper, Árvore

Propriedades:

- Steve - vida, saciedade
- Creeper - vida, tempo para explodir
- Árvore - número de blocos, tipo de árvore

Comportamentos:

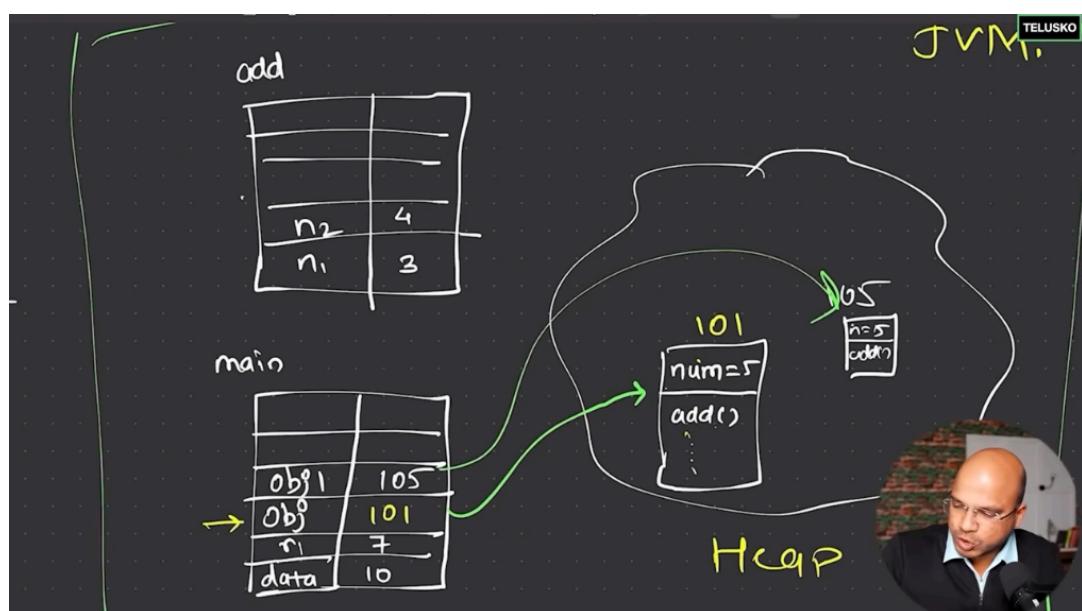
- Steve - andar, bater
- Creeper - andar, explodir
- Árvore - quebrar (retorna madeira), criar (varia no número de blocos e tipo de árvore)

▼ Methods

- Métodos sem retorno (void)
- Métodos com retorno
- Métodos com argumento
- Métodos com mais argumentos
- Métodos com o mesmo nome (sobrecarga, ele usou o exemplo de soma com 2 ou mais argumentos)

▼ Variáveis (Stack X Heap)

- Variáveis locais X Variáveis da Instância (propriedades)
- Variáveis locais são armazenadas em stacks específicas de funções (escopo da função)
- Objetos (Obj obj = new Obj()) não são objetos, são variáveis de referência, que apontam para um objeto na heap mempry
- Variáveis de instância - propriedades - são armazenadas na heap
- VARIÁVEIS LOCAIS FICAM NA STACK DA FUNÇÃO, LOCAIS
- INSTÂNCIAS DE OBJETOS FICAM NA HEAP E SÃO UMA STACK PER SI COM VARIÁVEIS E OS MÉTODO
- Tudo isso fica dentro da JVM!!
-



- Mudanças em um objeto não afetam o outro objeto da mesma classe

▼ Array

Complete Java, Spring, and Microservices course

Java developer in 48 hours; technically, it will take months to master.

https://www.youtube.com/watch?v=4XTsAAHW_Tc&t=13692s



Definição

- Conjunto de dados do mesmo tipo, melhor do que ter várias variáveis “num1, num2, num3, ...”;
- Possível visualizar tudo com um loop;
- Index (começa no 0)
- Por padrão, inicializa com 0 em todos os valores

```
int grades[] = {1, 2, 3, 4};  
int grades[] = new int[4];  
  
for(int i = 0; i< 4; i++){  
    System.out.println(grades[i]);  
}
```

MultiDimensional Array (Array 2D, 3D, nD)

- Conjunto de arrays, melhor do que vários arrays, matriz!
- Possível visualizar com nested loops (for do for)

```
int studentGrades[][] = new int[3][4]; // 3 students, each with 4 grades = Matrix  
studentGrades[1][2] = 4;  
  
for(int i = 0; i<3; i++){  
    for(int j = 0; j<4; j++){  
        System.out.print(studentGrades[i][j]);  
    }  
    System.out.println();  
}
```

Random function

- Retorna valor double, teria que ser convertido
-

```
int random = (int)(Math.random() * 10);  
  
for(int i = 0; i<3; i++){  
    for(int j = 0; j<4; j++){  
        studentGrades[i][j] = (int)(Math.random() * 10);;
```

```
    }  
}
```

Foreach

- Possível acessar os itens de um array um a um sem usar o índice
- “Nova” estrutura de repetição
-

```
for(int grade: grades){  
    System.out.println(grade);  
}  
  
for(int grades[] : studentGrades){  
    for(int grade: grades){  
        System.out.println(grade);  
    }  
}
```

Jagged Array - Dynamic Size na segunda dimensão

- Array de arrays de tamanho indefinido
-

```
int nums[][] = new int[3][];  
  
nums[0] = new int[3];  
nums[1] = new int[4];  
nums[2] = new int[5];  
  
for(int i = 0; i<3; i++){  
    for(int j = 0; j<nums[i].length; j++){  
        System.out.println(nums[i][j]);  
    }  
}
```

Drawbacks of Arrays

- O tamanho é fixo, impossível aumentar o tamanho do array (se for necessário, vai ter que criar um array novo);
- Pesquisa envolve iterar por todo o array;

- Precisa que todos os elementos sejam o mesmo tipo (pode usar Object, mas não é ideal);

Arrays of Objects

- É possível criar um Array de objetos, da mesma forma que é possível criar Arrays de tipos primitivos
-

```
Student students[] = new Student[3]; // não cria 3 estudantes
//cria um Array que pode armazenar 3 estudantes
```

```
for(Student student:students){
    System.out.println(student); // imprime algo como Student@7ad041f3
}
```

▼ Strings

Complete Java, Spring, and Microservices course
 Java developer in 48 hours; technically, it will take months to master.
https://www.youtube.com/watch?v=4XTsAAHW_Tc&t=16507s



Propósito

- A princípio, serve o mesmo propósito de um array de char
- Define com texto em aspas duplas ("string")
- Não é um tipo primitivo, é uma classe! Por isso "String" e não "string"
- Por isso, tem atributos e comportamentos que facilitam a manipulação
- Vira uma variável de referência na Stack;
- Cria o objeto com o texto na Heap;

```
String name = "String";
String name = new String("String");
```

Comportamento

- O comportamento mais utilizado da String é a concatenação

```

String firstName = "Douglas";
String lastName = " Rocha";

String fullName = firstName + lastName; // Douglas Rocha

String fullName2 = firstName.concat(lastName); // Douglas Rocha

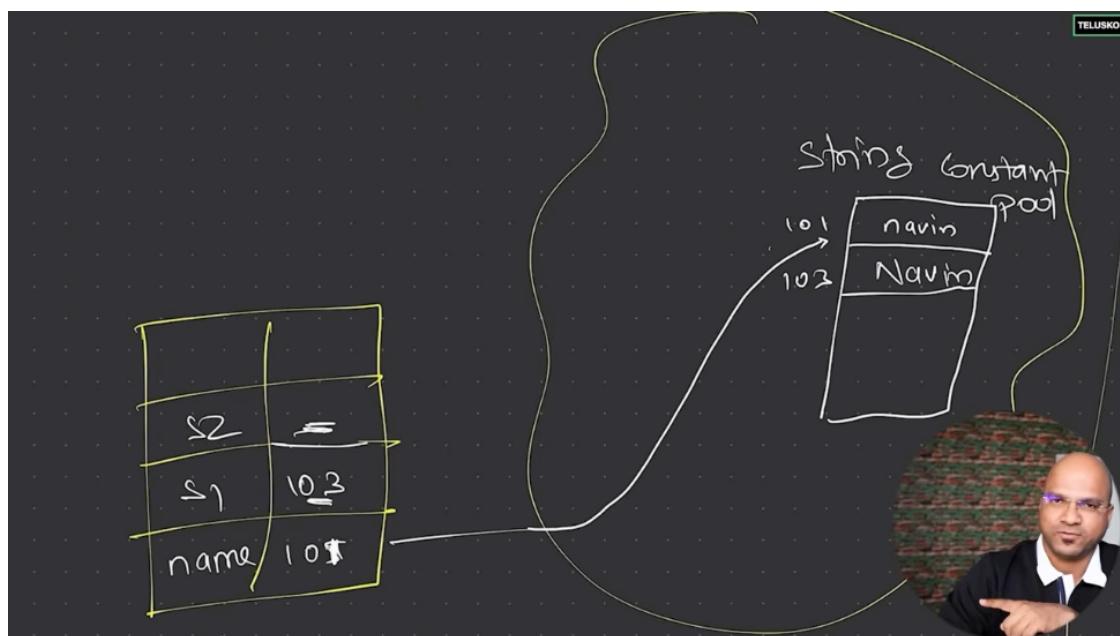
```

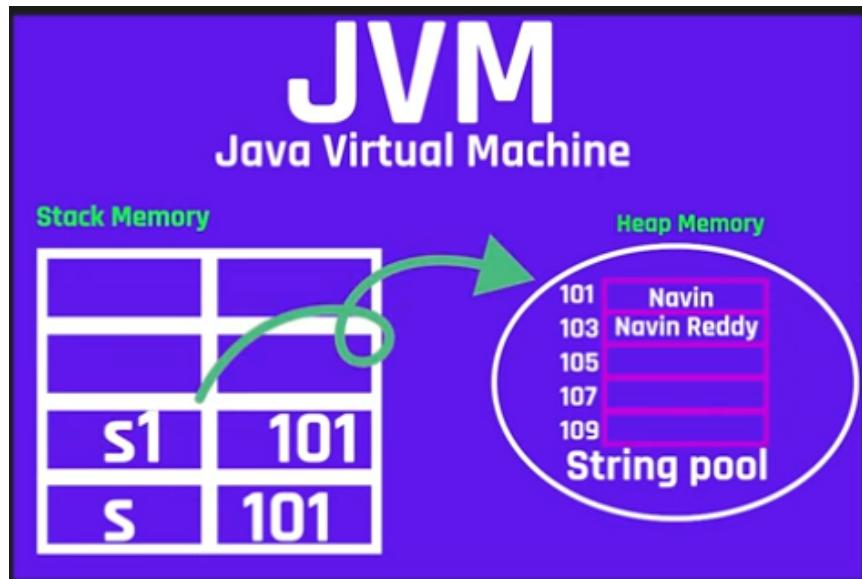
- CharAt - caractér na posição (indexado a partir do 0)
- Quando você altera uma String, não altera o conteúdo na Heap - String Constant Pool

```

String n1 = "Douglas";
String n2 = "Douglas";
// ambas apontam para o mesmo endereço na Heap,
// em um espaço especial chamado
// String Constant Pool, onde todas as Strings são salvas de forma cte
// e cada nova variável busca uma string já existente lá e aponta para ela

```





- Operações na String não alteram o objeto na Heap, na Constant Pool, criam um novo:

```
String name = "Douglas"; // endereço 0x00
name = name + " Rocha"; // endereço 0xFF ("Douglas" permanece no 0x00)
```

- Só quem apaga as Strings na Pool é o Garbage Collector

String Buffer

- É uma forma de ter uma String mutável
- Criado com uma capacidade mínima de 16 chars e preserva esse espaço para alocação futura caso o tamanho inicial seja maior

```
StringBuffer sb = new StringBuffer();
sb.capacity() // 16
sb = new StringBuffer("Douglas");
sb.capacity() // 23 (16 + 7)
```

- Permite mutabilidade de algumas formas

```
public class AulaString {
    public static void main(String[] args) {
        StringBuffer sb = new StringBuffer();
        System.out.println(sb.capacity()); // 16

        sb = new StringBuffer("Douglas");
        System.out.println(sb.capacity()); // 23
```

```

sb.append(" Rocha");
System.out.println(sb); // Douglas Rocha
System.out.println(sb.capacity()); // 23

sb.append(" Figueiredo");
sb.append(" Leite");
System.out.println(sb); // Douglas Rocha Figueiredo Leite
System.out.println(sb.capacity()); // 48
System.out.println(sb.length()); // 29

sb.insert(0, "Prof. ");
System.out.println(sb); // Prof. Douglas Rocha Figueiredo Leite
System.out.println(sb.capacity()); // 48

sb.deleteCharAt(25);
System.out.println(sb); // Prof. Douglas Rocha Figueredo Leite
System.out.println(sb.capacity()); // 48

sb.ensureCapacity(100);
System.out.println(sb); // Prof. Douglas Rocha Figueredo Leite
System.out.println(sb.capacity()); // 100
}

}

```

String Builder

- Idêntico, em funcionalidade, ao Buffer, mas é Thread Safe (ou o contrário, tentar não mencionar);

▼ Static

Complete Java, Spring, and Microservices course
 Java developer in 48 hours; technically, it will take months to master.

https://www.youtube.com/watch?v=4XTsAAHW_Tc&t=17663s



- Static é uma keyword essencial do Java, aparece a partir do HelloWorld na main
- Propriedades e Comportamentos da Classe, não do Objeto

Variável Estática

- Valor comum a todos os objetos da classe
- Pode ser acessado sem um objeto

```
Classe {  
    public static int VALOR = 12;  
}  
  
System.out.println(Classe.VALOR); // 12
```

- Útil para constantes, remover Magic Numbers

Bloco Estático

- Trecho de código executado apenas uma vez, referente à Classe, e cujos resultados influenciam todos os objetos
- A Classe “carrega” antes dos Objetos serem Instanciados, então o bloco é executado antes de qualquer criação de objetos
- Quem “carrega” é um componente da JVM chamado “Class Loader”
- Se não houver nenhuma criação de objeto, não carrega a classe e portanto o trecho não é executado.
- É possível utilizar “Class.forName()” com o nome da Class para fazer o load sem criar um objeto;

```
public class StaticBlock {  
    static int VALOR = 12;  
  
    static {  
        VALOR = VALOR * 2;  
        System.out.println("No bloco");  
    }  
  
    StaticBlock() {  
        System.out.println("No construtor");  
        VALOR = VALOR * 2;  
    }  
}  
  
public static void main(String[] args) throws ClassNotFoundException {  
    System.out.println(StaticBlock.VALOR);
```

```

Class.forName("StaticBlock");
System.out.println(StaticBlock.VALOR);

StaticBlock staticBlock = new StaticBlock();
System.out.println(StaticBlock.VALOR);
}

```

Método Estático

- O método estático tem o mesmo princípio da variável estática: pertence à Classe e não ao Objeto, portanto pode ser chamada a partir da Classe sem instanciar um objeto.
- Não pode acessar variáveis de instância porque não pertence a uma instância.

```

public class StaticMethod {
    static String name = "Static";

    static void seyHello() {
        System.out.println("Hello, " + name);
    }
}

public static void main(String[] args) {
    StaticMethod.seyHello();
}

```

- É possível acessar variáveis de instância caso a instância seja passada como argumento

Método Main

- O método Main é o primeiro método estático visto
- Ele é o ponto inicial da aplicação, onde tudo começa
- Se ele não fosse estático, a classe em que está (App ou HelloWorld) teria que ser instanciada para que ela fosse chamada
- Contudo, não teria como o objeto ser instanciado sem que o código começasse (e ele começa na main)
- Por isso, main é um método estático

▼ POO 2

Modificadores de Visibilidade - Encapsulamento

- Vem do conceito de cápsula, isolado do mundo externo
- Envolve modificadores de visibilidade
- E.g.: uma pessoa tem dados visíveis, públicos (aparência) e dados invisíveis, privados (nome, senha)

```
public class Humano {  
  
    public String corDosOlhos;  
    public String roupa;  
  
    private String senhaDoBanco;  
}
```

- Atributos privados são visíveis, acessíveis apenas dentro da própria classe, enquanto atributos públicos são visíveis por qualquer um

```
public class Demo {  
    public static void main(String[] args) {  
        Humano eu = new Humano();  
        eu.corDosOlhos = "preto";  
        eu.roupa = "calça jeans";  
  
        System.out.println(eu.corDosOlhos);  
        System.out.println(eu.roupa);  
    }  
}
```

- Contudo, atributos públicos não são apenas visíveis, sendo acessíveis eles também podem ser alterados do lado de fora (ninguém deveria mudar sua roupa ou a cor dos seus olhos)
- Portanto, o padrão é que todos os atributos de uma classe sejam privados (há raras exceções):

```
public class Humano {  
  
    private String corDosOlhos;  
    private String roupa;
```

```
    private String senhaDoBanco;  
}
```

Getters e Setters

- O problema em deixar todos os atributos privados, é que não faz sentido armazenar esses dados no software se eles não podem ser acessados e compartilhados
- A solução é o uso de métodos chamados de “Getters” e “Setters” para acessar e alterar os dados, mas apenas conforme a “permissão” do usuário.

```
public class Humano {  
  
    private String corDosOlhos = "preto";  
    private String roupa = "calça jeans";  
  
    private String senhaDoBanco;  
  
    public String getCorDosOlhos() {  
        return corDosOlhos;  
    }  
  
    public String getRoupa() {  
        return roupa;  
    }  
}  
  
public class Demo {  
    public static void main(String[] args) {  
        Humano eu = new Humano();  
  
        System.out.println(eu.getCorDosOlhos());  
        System.out.println(eu.getRoupa());  
    }  
}
```

- Nesse caso, as informações de corDosOlhos e roupa podem ser vistas de fora, mas não podem ser alteradas. Já a senha do banco não pode nem ser vista.

- Existe também o caso de querermos poder alterar algo de fora, como um apelido:

```

public class Humano {

    private String corDosOlhos = "preto";
    private String roupa = "calça jeans";
    private String apelido;

    private String senhaDoBanco;

    public String getCorDosOlhos() {
        return corDosOlhos;
    }

    public String getRoupa() {
        return roupa;
    }

    public String getApelido() {
        return apelido;
    }

    public void setApelido(String apelidoDado) {
        apelido = apelidoDado;
    }
}

public class Demo {
    public static void main(String[] args) {
        Humano eu = new Humano();

        System.out.println(eu.getCorDosOlhos());
        System.out.println(eu.getRoupa());

        eu.setApelido("professor");
        System.out.println(eu.getApelido());
    }
}

```

- Nesse caso utilizamos o método set com o parâmetro apelido para determinar qual o apelido da pessoa.

- Qual a diferença de ter um get e um set ou o atributo ser público: normalmente só é uma prática melhor, mas é possível que o objeto faça o controle das alterações ou visualizações que ele permite ou não

```
public void setApelido(String apelidoDado) {
    if (apelidoDado == "professor") {
        apelido = apelidoDado;
    } else if (apelidoDado == "bobão") {
        System.out.println("Não gostei desse apelido!");
    }
}
```

This

- Em alguns momentos, eu quero, dentro de um objeto, fazer referência a ele mesmo
- O exemplo mais comum é no próprio setter, quando eu quero utilizar o nome do parâmetro igual ao nome do campo

```
public void setApelido(String apelido){
    apelido = apelido;
}
```

- Esse código não funciona. Ele compila, mas nele estamos pegando o valor da variável local (parâmetro) apelido e atribuindo a ela mesma (ou seja, não estamos fazendo nada).
- Para fazer referência ao próprio objeto, e poder trocar o valor do atributo apelido daquele objeto, fazemos:

```
public void setApelido(String apelido){
    this.apelido = apelido;
}
```

Constructor

- É a função que constrói o objeto:

```
Humano eu = new Humano();
```

- O padrão do construtor é não ter nenhum parâmetro, nesse caso ele é chamado de Construtor Padrão (ou Default Constructor), e ele existe para

qualquer classe que não tenha um construtor definido

- É possível definir, para uma classe, mais de um Construtor. Assim, podemos ter um construtor com parâmetros:

```
public Humano(String corDosOlhos, String roupa) {  
    this.corDosOlhos = corDosOlhos;  
    this.roupa = roupa;  
}
```

- Quando definimos, perdemos o construtor padrão (a não ser que nós mesmos o criemos)

```
public class Demo {  
    public static void main(String[] args) {  
        Humano eu = new Humano("azul", "terno e gravata");  
  
        System.out.println(eu.getCorDosOlhos());  
        System.out.println(eu.getRoupa());  
    }  
}
```

▼ Convenções de Nomenclatura

- Existem 4 tipos mais comuns de convenção de nomenclatura:
 - CamelCase (como um camelo): meuBiscoitoOreo
 - Pascal Case: MeuBiscoitoOreo
 - Snake Case: meu_biscoito_oreo
 - Kebab Case: meu-biscoito-orei
- As linguagens de programação, em geral, escolhem uma ou mais convenções para serem utilizadas.
- No Java, nomes de variável e método são escritos com CamelCase
- Nomes de Classe são escritos com PascalCase
- Nomes de constantes (que ainda não vimos) são SnakeCase em caixa alta (MEU_BISCOITO_OREO)
- O propósito da convenção é legibilidade: para que a próxima pessoa que for ler seu código consiga compreender o que é cada coisa

▼ Herança

Conceito

- Diferença entre "ter" e "ser"
 - Um iPhone É um Mobile, É um Computador
 - Um iPhone TEM uma Tela, TEM um Processador
- Herança diz respeito ao SER
- No exemplo do Minecraft, um BlocoDeMadeira É um Bloco, assim como um BlocoDePedra, um BlocoDeVidro, etc;
- Na POO, a Classe Bloco seria uma Classe Pai de BlocoDeMadeira, BlocoDePedra, etc...
- Assim como na herança da vida real, tudo o que um pai tem, seu filho também tem
- A ideia é que o filho tenha todas as mesmas coisas (métodos e atributos) e também adicione novas propriedades específicas
- O FILHO É MAIS ESPECÍFICO DO QUE O PAI
- MULTI-LEVEL
- Não existe multipla herança (um filho com vários pais)

Override

- Um filho pode ter um método com o mesmo nome do seu pai, e quando for chamado será executado o do filho
- Isso se chama Override ou sobrescrever
-

```
public class A {  
    void show(){  
        System.out.println("A");  
    }  
}  
  
public class B extends A{  
    void show(){  
        System.out.println("B");  
    }  
}  
  
main(){  
    A a = new A();
```

```
B b = new B();  
  
    a.show(); // A  
    b.show(); // B  
}
```

▼ Packages

- O sistema de arquivos é organizado por arquivos/diretórios que separam por contexto
- No Java, já que teremos muitos arquivos, é normal termos uma hierarquia de organização em diretórios
- Um diretório, no Java, representa um pacote (package)
-

```
package monsters;  
  
public class Creeper{}  
  
public class Spider{}
```

- Na hora de utilizá-los em outra classe, precisamos importá-los

```
import monsters.Creeper;  
import monsters.Spider;
```

- É possível importar todas as classes de um pacote com *

```
import monsters.*;
```

- É possível também colocar diretórios dentro de diretórios, para melhorar a organização

```
import mobs.monsters.*;
```

- Nesse caso, não seria possível importar mobs.* e acessar as classes de monsters
- Para se preparar para o compartilhamento do projeto e/ou dos pacotes, é comum que o projeto inteiro fique dentro de um pacote estruturado segundo o endereço invertido como

```
package com.douglas.minecraft; // minecraft.douglas.com ao contrário
```

▼ Modificadores de Visibilidade

- **Public** - acessível de qualquer lugar
- **Private** - acessível apenas na mesma classe
- **Nada (default ou private protected)** - pode ser acessível no mesmo pacote
- **Protected** - pode ser acessado pelos filhos (herança) e no pacote
-



TELUSKO

	Private	Protected	Public	Default
Same class	Yes	Yes	Yes	Yes
Same package subclass	NO	Yes	Yes	Yes
Same package non-subclass	NO	Yes	Yes	Yes
Different package subclass	NO	Yes	Yes	NO
Different package non-subclass	NO	NO	Yes	NO

▼ Polymorphism

- Vários comportamentos
- É possível que o comportamento seja definido no tempo de compilação é Compile Time Polymorphism, se for definido em tempo de execução, é Runtime Polymorphism
- Compile Time normalmente é definido como Overloading, pois o comportamento é definido pelo número de parâmetros que já existe na compilação
- RunTime normalmente é definido como Overriding, porque o tipo de comportamento depende da classe mais específica que só é definida na execução
- Quando compila, não temos certeza de qual método será utilizado

```

public class A{
    public void show(){
        System.out.println("A show");
    }

    public void scream(){
        System.out.println("A scream");
    }
}

public class B extends A {
    public void show(){
        System.out.println("B show");
    }
}

public static void main(){
    A obj = new A();
    obj.show(); // A show
    obj.scream(); // A scream

    obj = new B();
    obj.show(); // B show
    obj.scream(); // A scream
}

```

- O comportamento é chamado "Dynamic Method Dispatch"

▼ Final

- Pode ser variável, método ou classe
- Na variável, cria uma constante, um valor que não pode ser substituído
- Na classe, faz com que a classe não possa ser herdada, extendida
- No método, faz com que o método não possa ser sobreescrito por herdeiros

▼ Class Object

- `toString`, `equals`, `hash`, etc

```

public class A{} // public class A extends Object{}

public class B{ // public class B extends Object{}

```

```

public String toString(){
    return "Oi";
}

}

public static void main(){
    A a = new A();
    // Métodos de Object
    a.toString(); // className + @ + hashCode(em hex)
    a.hashCode();

    B b = new B();
    b.toString(); // Oi

    boolean iguais = a.equals(b); // a.hash == b.hash
}

```

▼ Wrapper Classes

- Todas as classes do Java herdam do Object, mas nem todos os dados heram, porque nem todos são classes
- A exceção são os dados primitivos: int, char, boolean
- Existem, contudo, Wrapper Classes: Integer, Character e Double, que representam os mesmos dados primitivos mas herdam de Object e tem outras funcionalidades
- Autoboxing e Autounboxing

```

// boxing é colocar o dado primitivo dentro da classe:
Integer num = new Integer(2);

// autoboxing é quando isso é feito de forma automática:
Integer number = 3;

// unboxing é tirar o dado primitivo da classe:
int num2 = num.intValue();

// autounboxing é quando isso é feito de forma automatica
int number2 = number;

```

- Também tem métodos estáticos como "Integer.parseInt", por exemplo

▼ Abstract

- Métodos abstratos são apenas definidos, mas não implementados
- A ideia é que sejam definidos por filhos da classe que o definiu
- Apenas classes abstratas podem ter métodos abstratos
- Classes abstratas não podem ser instanciadas, não existem objetos abstratos
- Classes abstratas podem ter métodos “normais”, implementados, que serão herdados pelos filhos
- Não é obrigatório ter métodos abstrados na classe abstrata;
- Na realidade, a ideia de uma classe abstrata é ser um conceito, algo apenas para ser herdado
- Os filhos, a não ser que também sejam abstratos, obrigatoriamente precisam implementar todos os métodos abstratos
- O contrário de classe abstrata é classe concreta

▼ Inner Class

- Classe que será utilizada apenas para um propósito único e dentro de uma outra classe

```
public class Line{  
    List<Point> points;  
  
    public class Point{  
        int x;  
        int y;  
  
        public void move(int x, int y){  
            this.x = x;  
            this.y = y;  
        }  
    }  
}
```

- Como a classe não é abstrata, só é possível acessá-la através de um objeto:
-

```
public static void main(){  
    Line line = new Line();
```

```
    Line.Point point = line.new Point();
    point.move(0, 0);
}
```

- Se a classe for estática, contudo, é possível criá-la sem um objeto

```
public class Line{
    List<Point> points;

    public static class Point{
        int x;
        int y;

        public void move(int x, int y){
            this.x = x;
            this.y = y;
        }
    }
}

public static void main(){
    Line.Point point = new Line.Point();
    point.move(0, 0);
}
```

Anonymous Inner Class

- É possível criar classes internas anônimas
- Funcionam mais ou menos como classes definidas “inline”:
- Só existem “uma vez”, nem geram arquivo .class

```
public class A{
    public void show(){
        System.out.println("A");
    }
}

public class Demo{
    public static void main(){
        A a = new A();
```

```

a.show(); // A

A b = new A(){
    public void show(){
        System.out.println("B");
    }
}
b.show(); // B

A a2 = new A();
a2.show(); // A
}
}

```

- Nesse caso, a classe anônima criada “herda” de A
- É possível utilizar para instanciar uma classe abstrata, implementando inline

▼ Interface

- Facilitam o design do código, diminuindo o acoplamento
- Não possuem atributos, não podem ser instanciados e não são herdados
- São implementados!
- Não pode implementar nenhuma função, apenas defini-las
- Fora isso, são semelhantes a classes abstratas
- Qualquer atributo criado é, por default, final e static (não precisa usar os keywords, mas tem que ser inicializados)
- Os atributos não são herdados
- Quando alguma classe implementar, é obrigada a implementar os métodos
- Classes podem implementar mais de uma interface
- Interfaces podem herdar (extends) de outras interfaces

Normal Interface

- Se tiver mais do que 2 métodos na interface, é considerada uma “Normal Interface”

```

public interface Mob{
    void andar();
}

```

```
    void nascer();  
}
```

Functional Interface | SAM (Single Abstract Method Interface)

- Quando tem apenas 1 método, uma interface de apenas um método

Marker Interface

- Quando não tem nenhum método!
- Serve para serialização e desserialização: salvar o estado de um objeto na memória persistente (HD) e depois recuperá-lo; por exemplo
- A Interface “Serializable” serve apenas para informar ao compilador que aquela classe pode passar por esse processo (não é o único caso)

▼ Enum

- Ele usou o exemplo do HTTP

```
enum Status{  
    Running, Failed, Pending, Success;  
}  
  
public main(){  
    Status s = Status.Running;  
}
```

- No Java, o Enum é uma classe e cada opção é um objeto dessa classe
- Útil para constantes que podem ser utilizadas em vários locais, melhor do que strings ou números que podem ser digitados errado
- Cada Enum tem um id “embutido”

```
int a = Status.Running.ordinal();
```

- A Classe é capaz de listar todos os enums:

```
Status[] all = Status.values();
```

- Quando impresso, o toString retorna o nome:

```
System.out.println(Status.Running); // Running
```

- Pode ser usado na tomada de decisão com if-else ou switch:

```
switch(s){
    case Running:
        System.out.println("All good");
        break;
    case Failed:
        System.out.println("Something Wrong!");
        break;
    // ...
}
```

- Todo Enum herda de java.lang.Enum
- Enum não se envolve com herança, mas pode ter construtor, atributos e métodos

```
public enum HttpStatusCode{
    OK(200), NOT_FOUND(404), INTERNAL_SERVER_ERROR(500);

    private int code;

    private HttpStatusCode(int code){
        this.code = code;
    }

    public int getCode(){
        return this.code;
    }
}
```

▼ Annotations

- São informações dadas ao compilador, que normalmente não alteram o funcionamento do código
- Muitas vezes, ajudam mais o programador do que o programa
- Iniciam com um @

```
class A {
    public void show(){
        System.out.println("a");
    }
}
```

```

public void scream(){
    System.out.println("A!");
}

}

class B extends A {
    public void showSmall(){ // não dá erro
        System.out.println("b");
    }

    @Override
    public void screamBig(){ // dá erro, porque não tem screamBig no pai
        System.out.println("B!");
    }
}

public void main(){
    A b = new B();

    b.show(); // a
    b.showSmall(); // b

}

```

▼ Functional Interface e Lambda

- Quando tem apenas 1 método, uma interface de apenas um método
- Propõe-se a reduzir a verbosidade instanciando uma classe anônima com uma lambda

```

@FunctionalInterface
interface Adds{
    public int add(int a, int b);
}

public void main(){
    Adds obj = new Adds(){
        public int add(int a, int b){
            return a + b;
        }
    }
}

```

```
A obj2 = (a, b) → a+b;  
}
```

▼ Exceptions

- 3 tipos de Erro: CompileTimeError, RuntimeError, LogicalError
- CompiletimeError impede a compilação, é acusado antes de executar. Erros de digitação, por exemplo

```
// CompiletimeError, o compilador acusa  
System.out.printNewLine("oi");
```

- LogicalError não tem uma exceção, não quebra o software, mas é um comportamento errado, **um bug**

RuntimeError

- RuntimeError gera exceções, um erro que quebra o software enquanto ele executa
- E.g. quando tentamos acessar um arquivo que antes existia, mas agora não existe mais
- Quando ocorre um RuntimeError, acontece uma Exception, que pode ser tratada
- É mais interessante checar antes do que tratar a exceção, por exemplo verificando se um dividendo é 0 antes de deixar a divisão acontecer
- A forma de “capturar” uma exceção para que ela não cause o encerramento do código é com try-catch:
-

```
public static void main(){  
    int i = 0;  
    int j = 2;  
  
    try{  
        System.out.println(j / i);  
    } catch (Exception e){  
        System.out.println("Erro na divisão!" + e);  
    }  
}
```

```
        System.out.println("Não interrompeu o software...");  
    }
```

- Existem tipos específicos de exceção. No caso da divisão por zero, é a `ArithmaticException`, no caso, por exemplo, de acessar um índice inexistente de um array é o `ArrayIndexOutOfBoundsException`
- Todas essas opções herdam de `Exception`, por isso é possível capturar todas capturando `Exception`
- É possível capturar apenas um tipo de exceção, se preferível, e é possível ter mais de uma cláusula de captura:

```
try{  
    // código que pode dar ArithmaticException ou ArrayIndexOutOfBoundsException  
} catch (ArithmaticException e) {  
    System.out.println("Erro de aritmética");  
} catch (ArrayIndexOutOfBoundsException e) {  
    System.out.println("Fique nos limites");  
} catch (Exception e) { // trata qualquer outra exceção  
    System.out.println("Exceção desconhecida: " + e);  
}
```

- A melhor prática nesse caso, contudo, não é esperar pelas exceções, e sim checar antes

Throwable & Error

- Podem ser exceções ou erros
- Exceções podem ser capturadas e tratadas, e devem ser capturadas e tratadas
- Erros não podem ser tratados, como `ThreadDeath`, `IOError`, `OutOfMemory`, etc;

Checked X Unchecked

- As exceções de `Runtime` (como as vistas) são `unchecked`, que o compilador não nota em tempo de compilação
- As demais, que também herdam de `Exception`, são `checked`, o compilador avisa e obriga a capturar (não necessariamente a tratar)

```
// Esse código apenas captura e relata o erro  
try{
```

```

        int result = 2 / x;
    } catch (ArithmetricException e) {
        System.out.println("Erro de aritmética");
    }

    // Esse código trata o erro
    try{
        int result = 2 / x;
    } catch (ArithmetricException e) {
        int result
        System.out.println("Erro de aritmética");
    }

```

Throw

- Em alguns casos, você quer lançar ou perpetuar uma exceção
- Normalmente a ideia de lançar uma exceção é uma má prática, mas perpetuá-la pode fazer sentido se a função/classe responsável por tratá-la é diferente da função/classe responsável por capturá-la

```

public int divide(int x, int y){
    int result;
    try{
        result = x / y;
    } catch (ArithmetricException e){
        System.out.println("Tentativa de dividir por zero!");
        throw e;
    }

    return result;
}

public void calc(int x, int y, char operation){
    try{
        if(operation == '/'){
            divide(2, 0);
        }
    } catch (ArithmetricException e) {
        System.out.println("Erro na operação, reiniciando calculadora...");
        restart();
    }
}

```

```
    }  
}
```

Throws

- Throws ≠ throw
- Throw é uma ação, throws é uma afirmação
- Throws é uma keyword para informar ao compilador que um método pode lançar uma exceção "checked" e por isso ela deve ser capturada e tratada

```
public int divide(int x, int y) throws ArithmeticException{  
    int result;  
    result = x / y;  
    return result;  
}  
  
public void calc(int x, int y, char operation){  
    try{  
        if(operation == '/'){\br/>            divide(2, 0);  
        }  
    } catch (ArithmeticException e) {  
        System.out.println("Erro na operação, reiniciando calculadora...");  
        restart();  
    }  
}
```

- Este não é o melhor exemplo porque ArithmeticException é uma RuntimeException e, portanto, não é uma Checked Exception, o que significa que nada do que foi feito é feito pro obrigação do compilador, mas em casos de exceções como IOException, que ocorre quando lidamos com arquivos, é possível passar a responsabilidade de capturar e tratar a exceção para a função que chama

Custom Exception

- É possível criar Exceções, basta criar uma classe que herde de Exception (ou de algum outro filho de Exception)

```
public class HerobrineException extends Exception{  
    public HerobrineException(){  
        super("Herobrine identificado, reinicie o jogo!");
```

```
    }  
}
```

▼ Input

System.in.read()

- out é um objeto do tipo PrintStream estático e pertencente à classe System, com algumas funções de impressão:

```
System.out.println("Hello!");
```

- in é um objeto do tipo InputStream, estático e pertencente à classe System

```
int inputCharASCII = System.in.read(); // throws IOException
```

- A função retorna o valor na tabela ASCII do caractere inserido

```
int inputCharASCII = System.in.read();  
System.out.println(inputCharASCII);  
  
// 0 → 48  
// 1 → 49  
// A → 65
```

- O problema é que só lê um char

BufferedReader

- É uma classe que permite a leitura de mais texto:

```
InputStream is = System.in;  
InputStreamReader in = new InputStreamReader(is);  
BufferedReader bf = new BufferedReader(in);  
  
String text = bf.readLine();
```

- Lê até uma quebra de linha
- O benefício do BufferedReader é que ele recebe o Stream de Input como argumento, então pode receber outra fonte de dados, como um arquivo
- Outro problema do BufferedReader, além da verbosidade, é que ele é um recurso, então tem que ser fechado:

```
bf.close();
```

Scanner

- Scanner é a versão mais recente, eficiente e pouco verbosa do Java

```
Scanner sc = new Scanner(System.in); // também pode ler de várias fontes  
sc.nextLine(); // lê nova linha  
sc.nextInt(); // lê um inteiro
```

▼ Resources (Finally e Try-With)

Finally

- A cláusula Finally, num bloco try-catch, serve para executar um comando independente se houver ou não uma exceção:

```
try{  
    result = x / y;  
} catch (ArithmaticException e){  
    System.out.println("Erro aritmético!");  
} finally {  
    System.out.println("Tchau tchau...");  
}
```

- No exemplo, "tchau tchau" acontece independente de ter ocorrido ou não a exceção. Na prática, é o mesmo de fazer:

```
try{  
    result = x / y;  
    System.out.println("Tchau tchau...");  
} catch (ArithmaticException e){  
    System.out.println("Erro aritmético!");  
    System.out.println("Tchau tchau...");  
}
```

- Mas isso feriria a não repetição do código

Resources

- Chamamos algo de recurso (resource) quando se conecta com o mundo exterior (e, portanto, pode lançar exceções de IO)

- Qualquer classe que faça leitura de input é um resource
- Qualquer classe que se conecte com a rede é um recurso
- Classes que se conectam com o banco de dados ou arquivos são recursos
- Todo recurso tem que ser fechado antes do encerramento do código, se não pode gerar **vazamento de recursos**, o que enche a memória RAM da máquina
- Então o código padrão de um recurso é:

```
InputStream is = System.in;
InputStreamReader in = new InputStreamReader(is);
BufferedReader bf = new BufferedReader(in);

String text = bf.readLine();
System.out.println(text);

bf.close();
```

- Contudo, precisamos capturar e tratar a IOException gerada pela função `readLine`

```
try{
    InputStream is = System.in;
    InputStreamReader in = new InputStreamReader(is);
    BufferedReader bf = new BufferedReader(in);

    String text = bf.readLine();
    System.out.println(text);

    bf.close();
} catch(IOException e){
    System.out.println("Erro de IO: " + e);
}
```

- O problema desse código é: se a exceção acontece na função “`readLine`”, nós capturamos a exceção, mas nunca fechamos o recurso

Finally para fechar o Resource

- A solução é utilizar o `finally` para fechar o recurso

```

try{
    InputStream is = System.in;
    InputStreamReader in = new InputStreamReader(is);
    BufferedReader bf = new BufferedReader(in);

    String text = bf.readLine();
    System.out.println(text);
} catch(IOException e){
    System.out.println("Erro de IO: " + e);
} finally {
    bf.close();
}

```

- Desta forma, podemos garantir que o recurso será fechado
- Ainda é possível remover o “catch”, de forma que a exceção será propagada e capturada por alguma função/classe superior, mas manter o finally para fechar o recurso

```

public void readFromTerminal() throws IOException {
    try{
        InputStream is = System.in;
        InputStreamReader in = new InputStreamReader(is);
        BufferedReader bf = new BufferedReader(in);

        String text = bf.readLine();
        System.out.println(text);
    } finally {
        bf.close();
    }
}

```

- Esse código lançará a exceção para que alguém capture (ou o código encerre), mas antes disso fará o fechamento do recurso

Try-with

- Algumas linguagens possuem keywords como “with” para se referir que um certo bloco de código executará com um recurso aberto, mas esse recurso será automaticamente fechado ao final do bloco
- O Java implementa essa funcionalidade dentro do próprio try:

```

try(BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    BufferedReader bf = new BufferedReader(br)) {
    String text = bf.readLine();
    System.out.println(text);
} catch(IOException e){
    System.out.println("Erro de IO: " + e);
}

```

- Essa estrutura garante o fechamento do recurso independente da exceção
- O finally ainda é interessante para quando usamos mais de um recurso dentro de um bloco try

▼ Threads

- O sistema operacional é capaz de executar múltiplas aplicações ao mesmo tempo: navegador (com várias abas), jogo, IDE, música
- Contudo, a maioria dos computadores tem apenas uma CPU (ou uma CPU com apenas 1 core)
- Portanto, os sistemas operacionais precisam ter uma capacidade de alternar rapidamente entre várias coisas, dedicando o processador a cada uma delas por pequenos intervalos de tempo de forma que, para o usuário, todas funcionem “simultaneamente”
- Essa capacidade é chamada de multi-threading, a possibilidade de ter várias threads, linhas de execução ocorrendo “simultaneamente”, com o sistema operacional como responsável por orquestrar todas elas
- Podemos (e devemos) fazer uso dessa capacidade do sistema operacional em nosso software
- Para isso, podemos separar tarefas não relacionadas (como abas de um navegador que não interagem entre si) em threads diferentes, para que elas executem de forma “simultânea”
- Isso é especialmente importante quando alguma das Threads faz uma operação demorada, permitindo que as outras atuem enquanto isso e não fiquem paradas esperando
- Ainda mais importante quando essa operação demorada não depende do processador, como esperar uma entrada do usuário, uma impressão na tela ou uma leitura de disco, que não ocupam o processador mas demoram tempo, sem multithread o software ficaria ocioso, parado, enquanto espera o resultado
- Em um jogo (Minecraft, por exemplo) cada um dos Mobs é controlado por uma thread, a geração do mundo é controlada por outra thread, o seu

personagem é controlado por outra thread

- Isso se chama paralelismo!

Execução Sequencial

```
public class Hi{  
    public void hi10(){  
        for(int i = 0; i < 10; i++){  
            System.out.println("hi");  
        }  
    }  
}  
  
public class Hello{  
    public void hello10(){  
        for(int i = 0; i < 10; i++){  
            System.out.println("hello");  
        }  
    }  
}  
  
public class Demo{  
  
    public static void main(){  
        Hi hi = new Hi();  
        Hello hello = new Hello();  
  
        hi.hi10();  
        hello.hello10();  
    }  
    // Resultado: hi hi hi... hello hello hello ...  
}
```

- Neste exemplo, a execução é sequencial (ou síncrona), porque a execução da função "hello10" só começa depois que a "hi10" termina, e portanto só temos "hello" depois de todos os "hi"

Execução Paralela

```
public class Hi extends Thread {  
    public void run(){  
        for(int i = 0; i < 10; i++){  
    }
```

```

        System.out.println("hi");
    }
}
}

public class Hello extends Thread {
    public void run(){
        for(int i = 0; i < 10; i++){
            System.out.println("hello");
        }
    }
}

public class Demo{
    public static void main(){
        Hi hi = new Hi();
        Hello hello = new Hello();

        hi.start(); // executa a função run de hi
        hello.start(); // executa a função run de hello
    }
    // Resultado: hi hello hi hi hello hi hello hello...
}

```

- Se a execução ainda parecer sequencial, é porque a execução da run de hi é rápida o suficiente para não dar tempo de o OS trocar quem está usando a CPU
- Testando com 100 ao invés de 10, é possível ver mais alternância entre his e hellos
- Outra forma de forçar a alternância é ordenando que as threads “durmam” após uma impressão
- Quando uma thread dorme, a CPU é imediatamente passada para outra thread

```

public class Hi extends Thread throws InterruptedException {
    public void run(){
        for(int i = 0; i < 10; i++){
            System.out.println("hi");
            Thread.sleep(2);
        }
    }
}

```

```

    }
}

public class Hello extends Thread throws InterruptedException {
    public void run(){
        for(int i = 0; i < 10; i++){
            System.out.println("hello");
            Thread.sleep(2);
        }
    }
}

public class Demo{

    public static void main() throws InterruptedException {
        Hi hi = new Hi();
        Hello hello = new Hello();

        hi.start(); // executa a função run de hi
        hello.start(); // executa a função run de hello
    }
    // Resultado: hi hello hi hello hi hello...
}

```

Runnable

- Um problema de usar classes que herdam de Thread, é que isso nos impede de fazer as classes herarem alguma outra coisa (já que só podemos ter uma herança).
- Uma solução é utilizar a interface Runnable
- Em seguida precisamos criar um objeto Thread que receba uma classe Runnable como parâmetro, cuja função run ele vai executar

```

public class Hi implements Runnable throws InterruptedException {
    public void run(){
        for(int i = 0; i < 10; i++){
            System.out.println("hi");
            Thread.sleep(2);
        }
    }
}

```

```

public class Hello implements Runnable throws InterruptedException {
    public void run(){
        for(int i = 0; i < 10; i++){
            System.out.println("hello");
            Thread.sleep(2);
        }
    }
}

public class Demo{

    public static void main() throws InterruptedException {
        Hi hi = new Hi();
        Hello hello = new Hello();

        Thread t1 = new Thread(hi);
        Thread t2 = new Thread(hello);

        t1.start();
        t2.start();
    }
    // Resultado: hi hello hi hello hi hello...
}

```

Lambda

- Podemos criar as classes como funções anônimas

```

public class Demo{

    public static void main() throws InterruptedException {
        Runnable hi = new Runnable(){
            public void run(){
                for(int i = 0; i < 10; i++){
                    System.out.println("hi");
                }
            }
        };
        Runnable hello = new Runnable(){
            public void run(){

```

```

        for(int i = 0; i < 10; i++){
            System.out.println("hi");
        }
    };
}

Thread t1 = new Thread(hi);
Thread t2 = new Thread(hello);

t1.start();
t2.start();
}
// Resultado: hi hello hi hello hi hello...
}

```

- Além disso, podemos usar uma função lambda já que Runnable é uma FunctionalInterface ou SAM

```

public class Demo{

    public static void main() throws InterruptedException {
        Runnable hi = () -> {
            for(int i = 0; i < 10; i++){
                System.out.println("hi");
            }
        };

        Runnable hello = () -> {
            for(int i = 0; i < 10; i++){
                System.out.println("hi");
            }
        };

        Thread t1 = new Thread(hi);
        Thread t2 = new Thread(hello);

        t1.start();
        t2.start();
    }
    // Resultado: hi hello hi hello hi hello...
}

```

- Ainda é possível reduzir ainda mais o código se não atribuirmos as funções lambda às variáveis Runnable

```
public class Demo {
    public static void main(String[] args) {

        Thread t1 = new Thread(() -> {
            for (int i = 0; i < 10; i++) {
                System.out.println("hi");
            }
        });

        Thread t2 = new Thread(() -> {
            for (int i = 0; i < 10; i++) {
                System.out.println("hello");
            }
        });

        t1.start();
        t2.start();
    }
    // Resultado: hi hello hi hello hi hello...
}
```

Race Condition

- Race condition ou condição de corrida, quando duas threads tentam acessar o mesmo recurso (não necessariamente resource, pode ser uma variável)

```
public class Counter {
    public int counter;
    public void increment() {
        counter = counter + 1;
    }
}

public class TestRunnable {
    public static void main(String[] args) {
        Counter counter = new Counter();

        Thread t1 = new Thread(() -> {
```

```

        for (int i = 0; i < 1000; i++) {
            counter.increment();
        }
    });

    t1.start();

    System.out.println(counter.counter); // 123

    System.out.println(counter.counter); // 925

    System.out.println(counter.counter); // 1000
}
}

```

- main também é uma Thread, então enquanto t1 está incrementando counter, main retoma o processador e faz a impressão, mesmo que a thread não tenha terminado
- É possível esperar as threads encerrarem com o comando join

```

public class Counter {
    public int counter;
    public void increment() {
        counter = counter + 1;
    }
}

public class TestRunnable {
    public static void main(String[] args) throws InterruptedException {
        Counter counter = new Counter();

        Thread t1 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) {
                counter.increment();
            }
        });
    }

    t1.start();

    System.out.println(counter.counter); // 123
}

```

```

        t1.join();

        System.out.println(counter.counter); // 1000
    }
}

```

- Contudo, ainda temos um problema se utilizarmos duas threads

```

public class Counter {
    public int counter;
    public void increment() {
        counter = counter + 1;
    }
}

public class TestRunnable {
    public static void main(String[] args) throws InterruptedException {
        Counter counter = new Counter();

        Thread t1 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) {
                counter.increment();
            }
        });

        Thread t2 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) {
                counter.increment();
            }
        });

        t1.start();
        t2.start();

        t1.join();
        t2.join();

        System.out.println(counter.counter); // resultado 1659
    }
}

```

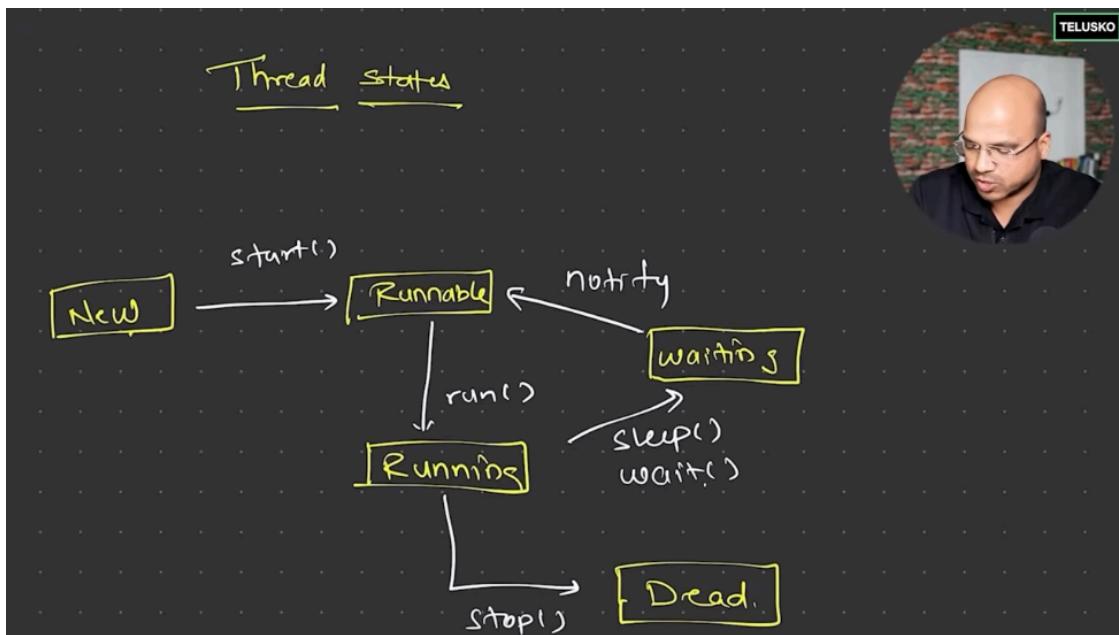
- Isso ocorre porque, em algum momento, t1 tentou incrementar o valor, ela leu o valor (por exemplo) 1200, calculou a soma 1201 e preparou-se para armazenar o valor de volta na variável
- Contudo, em algum momento durante essa execução, t2 pegou o processador, leu 1200, calculou a soma 1201 e armazenou o valor
- Ou até, possivelmente, fez isso várias vezes, chegando até o valor 1412 (exemplo)
- Só depois de todos esses incrementos, t1 pegou de novo o processador, mas ela estava no meio do processamento e apenas armazenou o valor 1201, perdendo todos os dados
- No Java, a forma de resolver esse problema, é garantir que a função "increment" é sincronizada

```
public class Counter {
    public int counter;
    public synchronized void increment() {
        counter = counter + 1;
    }
}
```

- Na prática, o Java está utilizando semáforos internos para impedir o acesso à função e à variável quando alguma thread estiver usando, fazendo o processamento da increment ser **atômico**

Thread States

- As threads tem alguns estados possíveis e uma sequência



- A diferença entre runnable e running é que runnable ela está aguardando a liberação da CPU, running ela está literalmente ocupando a CPU
- O OS também pode fazê-la voltar de running para runnable ao tirar a CPU dela

▼ Collection API

- Collection API é o conceito das coleções do Java, as classes que permitem e facilitam a manipulação de mais de um item de uma mesma classe
- Collection é uma interface que algumas coleções implementam
- Collections é uma classe com algumas funções estáticas úteis
- As coleções tem sobre os Arrays as vantagens de não apenas armazenarem os dados, mas oferecerem facilidades e métodos que os arrays não tem
- Além disso, as coleções são diferentes **Estruturas de Dados**
- Finalmente, as coleções são dinâmicas, ou seja, o tamanho é variável, diferente dos arrays
- API significa Application Programming Interface, que se refere às formas de acessar essas funcionalidades

Collection Interface

- Pertence ao pacote `java.util`
- Tem várias funções úteis para manipulação de coleções, muitas para implementar como uma classe anônima

- Tem muitas implementações de vários tipos de Estruturas de Dados como ArrayList, LinkedList, Queue, HashSet, etc.
- A maioria das coleções não exige um tipo específico de dado e por isso utiliza **Generics** para determinar o tipo de dado que será armazenado (só pode ser classe, não dado primitivo)

ArrayList

```
List<Integer> nums = new ArrayList<Integer>();
nums.add(6);
nums.add(7);
nums.add(8);

System.out.println(nums); // [6,7,8]

System.out.println(nums.get(0)); // 6

nums.forEach(n → System.out.println(n)); // 6 7 8
```

- Ao invés de referenciar a ArrayList como ArrayList, podemos referenciar como List, que tem todos os acessos necessários
- Pode ser considerado uma das coleções mais básicas, pois é basicamente um array dinâmico com mais funcionalidades, mas não representa uma estrutura de dados mais diferenciada

Set

- O Set difere da Lista pois não permite repetição de valores (não gera erro, apenas remove replicatas)
- Além disso, o Set não mantém a ordem em que os itens foram inseridos e, por isso, não permite acesso por índice

```
Set<Integer> nums = new HashSet<Integer>();
nums.add(7);
nums.add(7);
nums.add(8);

System.out.println(nums); // [8,7]

// Não tem índice
// System.out.println(nums.get(0));
```

```
for(int n: nums){  
    System.out.println(n);  
}
```

- Se usarmos o TreeSet, os valores ficarão ordenados, mas ainda não permitem o acesso por índice

```
Set<Integer> nums = new TreeSet<Integer>();  
nums.add(9);  
nums.add(7);  
nums.add(2);  
  
System.out.println(nums); // [2, 7, 9]  
  
for(int n: nums){  
    System.out.println(n);  
}
```

Map

- O Map não implementa a interface Collection, mas faz parte da CollectionAPI
- Map é uma coleção de pares chave-valor como, por exemplo, nome e nota
- Também não mantém índice (fora da chave) e ordem
- Também não permite chaves repetidas, se inserirmos duas vezes há uma atualização do valor

```
import java.util.HashMap;  
import java.util.Map;  
  
public class MapTest {  
    public static void main(String[] args) {  
        Map<String, Integer> notas = new HashMap<>();  
        notas.put("Douglas", 10);  
        notas.put("Pedro", 8);  
        notas.put("Maria", 9);  
  
        System.out.println(notas); // {Douglas=10, Pedro=8, Maria=9}  
  
        System.out.println(notas.get("Douglas")); // 10  
  
        for (String name : notas.keySet()) {
```

```

        System.out.println(notas.get(name)); // 10 8 9
    }

    notas.put("Pedro", 6); // Possível atualizar o valor de uma chave

    System.out.println(notas); // {Douglas=10, Pedro=6, Maria=9}
}
}

```

- Hashtable é identico ao HashMap mas é “synchronized”, ou seja, atômico para operações com múltiplas threads

Sorting

- Em alguns casos, podemos ter dados que não estão (ou não sabemos se estão) ordenados corretamente
- A classe Collections tem funções que nos ajudam com isso:

```

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class SortTest {
    public static void main(String[] args) {
        List<Integer> nums = new ArrayList<>();
        nums.add(6);
        nums.add(3);
        nums.add(9);

        System.out.println(nums); // [6, 3, 9]

        Collections.sort(nums);

        System.out.println(nums); // [3, 6, 9]
    }
}

```

- É possível alterar a forma de comparação como, por exemplo, comparar apenas pelo último dígito de um número
- Para isso, usamos uma instância da classe Comparator (normalmente anônima) que define a função compare que determinará, entre 2 itens, qual é o maior

```

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

public class SortTest {
    public static void main(String[] args) {
        List<Integer> nums = new ArrayList<>();
        nums.add(10009);
        nums.add(90002);
        nums.add(30003);

        System.out.println(nums); // [10009, 90002, 30003]

        Collections.sort(nums);
        System.out.println(nums); // [10009, 30003, 90002]

        Comparator<Integer> comparator = new Comparator<Integer>() {
            @Override
            public int compare(Integer arg0, Integer arg1) {
                if (arg0 % 10 > arg1 % 10) {
                    return 1;
                } else {
                    return -1;
                }
            }
        };

        Collections.sort(nums, comparator);
        System.out.println(nums); // [90002, 30003, 10009]
    }
}

```

Comparable e Comparator

- Para podermos determinar como uma ordenação vai acontecer usamos o Comparator
- Queremos agora testar com uma classe

```

public class Student {
    int age;
    String name;

    public Student(int age, String name) {
        this.age = age;
        this.name = name;
    }

    public String toString() {
        return "Student [age=" + age + ", name = " + name + "]";
    }
}

```

- Sem tentamos criar o Comparator conseguimos

```

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

public class Demo {
    public static void main(String[] args) {

        Comparator<Student> comparator = new Comparator<Student>() {
            public int compare(Student i, Student j) {
                return i.age - j.age;
            }
        };

        List<Student> students = new ArrayList<>();
        students.add(new Student(25, "Douglas"));
        students.add(new Student(24, "Tayla"));
        students.add(new Student(18, "Pedro"));
        students.add(new Student(35, "Maria"));

        for (Student student : students) {
            System.out.println(student);
        }

        Collections.sort(students);
    }
}

```

```
    }
}
```

- Mas há outra forma, de inserir isso dentro da classe, com o Comparable

```
public class Student implements Comparable<Student> {
    int age;
    String name;

    public Student(int age, String name) {
        this.age = age;
        this.name = name;
    }

    public String toString() {
        return "Student [age=" + age + ", name = " + name + "]";
    }

    @Override
    public int compareTo(Student other) {
        return 0;
    }
}
```

- Agora o código funciona!
- Contudo, não precisamos do Comparator, porque podemos implementar a mesma lógica dentro do compareTo

```
@Override
public int compareTo(Student other) {
    return this.age - other.age;
}
```

- E podemos apagar o Comparator

```
public class Demo {
    public static void main(String[] args) {
        List<Student> students = new ArrayList<>();
```

```

students.add(new Student(25, "Douglas"));
students.add(new Student(24, "Tayla"));
students.add(new Student(18, "Pedro"));
students.add(new Student(35, "Maria"));

Collections.sort(students);

for (Student student : students) {
    System.out.println(student);
}
}
}

```

- Podemos utilizar o Comparator E o Comparable. Podemos usar o Comparable SEM o Comparator. Podemos usar o Comparator SEM o Comparable

Lambda Comparator

- Esse nome não existe, mas achei uma forma fácil de pontuar a existência disso:

```

Comparator<Student> comparator = new Comparator<Student>() {
    public int compare(Student i, Student j) {
        return i.age - j.age;
    }
};

Comparator<Student> comparator = (i, j) → i.age - j.age;

```

▼ Streams

- Exemplo sem Stream

```

public static void main(String[] args) {

    List<Integer> nums = Arrays.asList(4, 5, 7, 3, 6);
    List<Integer> result = new ArrayList<>();

    for (int n : nums) {
        if (n % 2 == 0) {
            n = n * 2;
            result.add(n);
        }
    }
}

```

```
    }  
}  
  
System.out.println(result);  
}
```

- Podemos utilizar a Stream API para facilitar processos como esse
- Stream é uma Interface
- A Lista tem um método que a transforma numa Stream

```
Stream<Integer> s = nums.stream();
```

- Assim como a lista, a stream tem o método foreach

```
s.forEach(n → System.out.println(n)); // 4 5 7 3 6
```

- Contudo, essa operação não pode ser feita duas vezes, dando o erro: IllegalStateException: stream has already been operated upon or closed
- Contudo, uma stream pode retornar outra stream, como num filtro, por exemplo

```
Stream<Integer> s1 = nums.stream(); // 4 5 7 3 6  
Stream<Integer> s2 = s1.filter(n → n%2==0); // 4 6  
Stream<Integer> s3 = s2.map(n → n*2); // 8 12
```

- A função filter retorna uma stream apenas com os valores em que a lambda retorna true
- A função map retorna uma stream com o resultado da operação

```
s3.forEach(n → System.out.println(n)); // 8 12
```

- A função forEach não retorna uma stream, e por isso encerra a possibilidade de uso dessa Stream

```
// assumindo que não fizemos o forEach  
Integer result = s3.reduce(0, (c,e) → c+e); // 20
```

- Não vimos ainda o que reduce faz em detalhe, mas aparentemente opera sobre a Stream dando um resultado do tipo T (generic da Stream)

- O benefício de que uma função da stream retorna uma stream é a possibilidade de montar como um pipeline:

```
int result = nums.stream()
    .filter(n → n%2==2)
    .map(n → n*2)
    .reduce(0, (c,e) → c+e);

result; // 20
```

Filter

- O método Filter recebe algo que implemente a interface Predicate
- A interface Predicate é uma FunctionalInterface que tem apenas uma função test, que retorna um booleano
- Se o retorno dessa função for true, o filter mantém o valor, do contrário, ele remove da Stream (ou, no caso, não inclui na nova Stream)

Map

- O método map recebe algo que implemente a interface Function<T,R>
- A interface Function é uma FunctionalInterface que tem apenas uma função apply que recebe T e retorna R

```
Function<Integer, Integer> foo = Function<Integer, Integer>(){
    public Integer apply(Integer n){
        return n*2;
    }
}
```

```
Function<Integer, Integer> foo = n → n * 2;
```

- Ao passar apenas a lambda, o método map infere os tipos de T e R (que não precisam ser iguais)

Reduce

- Normalmente, quando fazemos uma adição ($1 + 2 + 3 + 4$), fazemos uma adição de cada vez, ou uma adição cumulatória ($((1+2) + 3) + 4$)
- No método reduce, pensamos que o "c" é o "carry", o último resultado, e "e" é o "element" a ser somado a seguir

- Além disso, nessa soma, assumimos que o valor inicial é 0, mas poderia ser outro
- Na função reduce, passamos primeiro o argumento do valor inicial, e depois passamos a lambda que vai operar sobre o carry e o element

```
.reduce(0, (c,e) → c + e); // soma os elementos partindo de 0
```

Sorted

- Outro método stream interessante, que não foi usado no exemplo é o .sorted()
- Se não receber argumento, o .sorted vai fazer a ordenação baseado no compareTo interno da classe

ParallelStream

- Algumas operações podem ser agilizadas caso utilizemos o multithread para realizá-las, como filter e map
- Outras não são tão ajudadas, como o sorted ou o reduce

```
nums.parallelStream()
    .filter(n → n%2==0) // beneficia do paralelismo
    .map(n → n*2) // beneficia do paralelismo
    .reduce(0, (c,e) → c + e); // não se beneficia do paralelismo
```

▼ Sealed Classes

- Classes Abstratas são feitas para serem herdadas
- Classes concretas herdam, e classes concretas finais não podem ser herdadas
- As classes seladas (ou Sealed Class) podem determinar quais subclasses podem herdar ou implementar elas
- Usam a keyword nova “sealed”

```
sealed class SmartPhone permits iPhone, GooglePixel, Samsung{}
```

```
sealed class iPhone extends SmartPhone {}
```

```
non-sealed class GooglePixel extends SmartPhone {}
```

```
final class Samsung extends SmartPhone {}
```

```
class NokiaTijoão extends SmartPhone {} // causa erro!!
```

- As herdeiras tem que ser sealed, non-sealed ou final
- No exemplo, iPhone é sealed, então ela pode permitir ser herdada por outras classes

```
sealed class iPhone extends SmartPhone permits iPhoneX, iPhone14 {}
```

- GooglePixel pode ser herdado por qualquer classe, por ser non-sealed

```
class Pixel2 extends GooglePixel {}
```

- Todos os exemplos foram com classes mas também podem ser utilizados para interfaces, a única diferença é que o filho não pode ser final

▼ Records

- Tenta ajudar com "Data Carrier Classes", ou Data Transfer Objects

```
record Alien (int id, String name) {}
```

- Gera automaticamente o construtor com todos os parametros, o `toString` e `equals` com igualdade de parâmetros
- Não gera construtor default
- Não gera getters com o nome `get`, mas com o nome do atributo como `".name()"`
- É possível definir tudo mais do que for necessário dentro do record
- Todos os atributos são, por padrão, **finais e privados**
- Caso necessário alterar o construtor, é possível criar um "construtor compacto"

```
record Alien (int id, String name) {  
    public Alien { // sem argumentos  
        if (id == 0) {  
            throw new IllegalArgumentException();  
        }  
        // não precisa atribuir os valores  
    }  
}
```

- Não pode herdar de nada, mas pode implementar interfaces

▼ JUnit5 / Junit Jupiter

▼ Conceito

- Teste é feito após a devolução, normalmente pela equipe de QA
- Teste Unitário é feito pelo próprio desenvolvedor, dentro do ciclo de vida do desenvolvimento, logo após o desenvolvimento de uma aplicação (ou antes, no TDD)
- Uma função / método / feature pode ser considerada uma unidade a ser testada
- Normalmente, a menor unidade é uma função
- Com TDD, é possível fazer primeiro o teste e depois a unidade

▼ Teste Unitário sem Framework

```
public class Calc{
    public int divide(int a, int b){
        return a / b;
    }
}

public static void main(){
    Calc c = new Calc();

    int result = c.divide(10, 5);

    if(result == 2){
        System.out.println("test passed");
    } else {
        System.out.println("test failed");
    }
}
```

- Passos percorridos:
 - preparação
 - providenciar o input
 - executar
 - providenciar o output esperado

- fazer “assertions” de igualdade
- ter um feedback para o desenvolvedor

▼ Maven

- Para poder utilizar o JUnit5, que é uma biblioteca/framework externo e não inerente ao Java, é necessário fazer a importação/instalação da dependência
- Os build assistants (Maven, Gradle, etc) são essenciais para facilitar o gerenciamento dessas dependências
- Para criar um projeto Maven, normalmente as IDE dão apoio, mas basta criar a estrutura e o arquivo “pom.xml”
- Para incluir o JUnit, basta incluir uma tag <dependencies> no pom.xml, pesquisar as informações de dependência do JUnit 5 ou Jupiter e incluí-las, o resultado será esse:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apac
<modelVersion>4.0.0</modelVersion>

<groupId>com.telusko.junit</groupId>
<artifactId>first</artifactId>
<version>1.0-SNAPSHOT</version>

<properties>
  <maven.compiler.source>17</maven.compiler.source>
  <maven.compiler.target>17</maven.compiler.target>
</properties>

<dependencies>
  <!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-api -->
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>5.11.3</version>
    <scope>test</scope>
  </dependency>
</dependencies>

</project>
```

▼ JUnit5

- Não retrocompatível com JUnit4, mas tem uma API Vintage que permite algo próximo disso
- A API padrão do JUnit 5 é o Jupiter, mas é possível aumentar a API com as Extensions
- Os testes unitários são, normalmente, organizados em arquivos com o nome da classe testada + Test
- Eles devem ser postos no mesmo pacote que a classe, mas ao invés da pasta main, ficam na pasta test
- Neste exemplo, o código ficará em src/main/java/com/telusko/junit, enquanto os testes ficarão em src/test/java/com/telusko/junit

```
package com.telusko.junit;

public class Calc {
    public int divide(int a, int b) {
        return a / b;
    }
}
```

```
package com.telusko.junit;

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class CalcTest {
    @Test
    public void test1() {
        Calc c = new Calc();

        assertEquals(2, c.divide(10, 5));
    }
}
```

- A IDE normalmente facilita a execução dos testes, mas é possível executá-los com "mvn test"

@ Test

- A Annotation @ Test é aplicada sobre métodos que serão testados
- Na hora de importar, deve ser importado de org.junit.jupiter.api, não de org.junit
- Os métodos com test não precisam ser public, podem ser default ou protected também
- Se alguma das asserções falhar, o teste falhou, do contrário ele falha (ou seja, um teste vazio passa)
- É possível criar múltiplos testes num mesmo arquivo, e nesse caso é importante esclarecer o que está sendo testado com o nome do método

```
package com.telusko.junit;

import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

public class CalcTest {
    @Test
    public void testDivision_with10and5() {
        Calc c = new Calc();

        assertEquals(2, c.divide(10, 5));
    }

    @Test
    public void testDivision_with3and2() {
        Calc c = new Calc();

        assertEquals(1, c.divide(3, 2));
    }
}
```

- Nesse caso, é interessante ter uma única instância da classe sendo testada (quando possível)

```
package com.telusko.junit;

import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;
```

```

public class CalcTest {

    Calc c = new Calc();

    @Test
    public void testDivision_with10and5() {
        assertEquals(2, c.divide(10, 5));
    }

    @Test
    public void testDivision_with3and2() {
        assertEquals(1, c.divide(3, 2));
    }
}

```

- Os métodos de assertion do JUnit5 permitem incluir mensagens para quando haja algum erro:

```

@Test
public void testDivision_with3and2() {
    assertEquals(1, c.divide(3, 2), "Divisão deu errado com valor racional");
}

```

- Além disso, é possível utilizar uma lambda com a implementation supplier que usa functional interface, que aumenta levemente a performance pois a String não é “instanciada” ou “evaluated” caso o teste passe

```

@Test
public void testDivision_with3and2() {
    assertEquals(1, c.divide(3, 2),
        () -> "Divisão deu errado com valor racional"
    );
}

```

Métodos Assert

- Para verificar se dois elementos são iguais ou diferentes:

```

assertEquals(expected, actual);
assertNotEquals(expected, actual);

```

- Para verificar se uma afirmação (statement) é verdadeiro ou falso:

```
assertTrue(statement);
assertFalse(statement);
```

- Para verificar se uma chamada de método ou trecho de código dispara ou não uma exceção:

```
assertThrows(null.metodo());
assertDoesNotThrow(obj.method());
```

- Para comparar dois arrays (não precisam ter a mesma ordem):

```
assertArrayEquals(expected, actual);
// não tem negativa
```

- Para testar o tempo de execução de um método

```
assertTimeout(duration, obj.method());
// duration normalmente em Duration.ofMillis(...)
```

Annotations

- Além do @ Test, existem outras Annotations relevantes
- O BeforeEach anota um método que será executado antes de todos os testes

```
@BeforeEach
void inicializar(){
    Obj obj = new Obj();
    System.out.println("Executado antes do teste");
}
```

- Ao contrário, o AfterEach vem depois de todos os testes

```
@AfterEach
void finalizar(){
    resource.close();
    System.out.println("Executando depois do teste");
}
```

- `BeforeAll`, ao invés de executar antes de cada teste, executa uma vez antes de todos os testes. O método anotado tem que ser `static`

```
@BeforeAll
static void antesDeTudo(){
    System.out.println("Executado antes de todos os testes");
}
```

- `AfterAll` executa apenas uma vez depois de todos os testes serem executados. O método também tem que ser `static`

```
@AfterAll
static void depoisDeTudo(){
    System.out.println("Executado depois de todos os testes");
}
```

- O conceito das annotations nos permite inferir (e é o caso) que é criada uma instância da classe de teste para cada teste que é executado. É possível alterar esse comportamento com a annotation `@TestInstance`

```
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
public class Test {
```

- Nesse caso, podemos criar `@BeforeAll` e `@AfterAll` com métodos não estáticos
- Esse seria o fluxo de execução de vários testes com as annotations e com `TestInstance` padrão:

```
package com.telusko.junit;

import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class SequenceTest {

    SequenceTest() {
        System.out.println("\nConstrutor - Antes de cada teste");
    }
}
```

```
@BeforeAll
static void antesDeTudo() {
    System.out.println("BeforeAll - Antes de todos os testes");
}

@BeforeEach
void antesDeCada() {
    System.out.println("BeforeEach - Antes de cada teste");
}

@Test
void test1() {
    System.out.println("Teste 1");
}

@Test
void test2() {
    System.out.println("Teste 2");
}

@Test
void test3() {
    System.out.println("Teste 3");
}

@AfterEach
void depoisDeCada() {
    System.out.println("AfterEach - Depois de cada teste\n");
}

@AfterAll
static void depoisDeTudo() {
    System.out.println("AfterAll - Depois de todos os testes");
}

/*
 * BeforeAll - Antes de todos os testes
 *
 * Construtor - Antes de cada teste
 * BeforeEach - Antes de cada teste
 * Teste 1
```

```

    * AfterEach - Depois de cada teste
    *
    *
    * Construtor - Antes de cada teste
    * BeforeEach - Antes de cada teste
    * Teste 2
    * AfterEach - Depois de cada teste
    *
    *
    * Construtor - Antes de cada teste
    * BeforeEach - Antes de cada teste
    * Teste 3
    * AfterEach - Depois de cada teste
    *
    * AfterAll - Depois de todos os testes
    */
}

}

```

▼ Test Driver Development (TDD)

- Uma forma comum de desenvolver uma aplicação ou feature maior e mais crítica, é começar pelos testes e seguir o passo a passo do TDD
 1. **Primeiro** crie a assinatura do método, mas sem nenhum código dentro (não necessariamente começa assim, mas foi o que o curso disse) apenas o necessário para o código compilar

```

public class Shape {
    public double calcSquareArea(double side) {
        return 0;
    }
}

```

2. **Segundo**, desenvolva o teste com o comportamento esperado e o execute, esperando que falhe

```

public class ShapeTest {
    Shape shape = new Shape();

    @Test
    void testCalcQuareArea_withSide10() {
        assertEquals(144, shape.calcSquareArea(12));
    }
}

```

```
    }  
}
```

3. **Terceiro**, corrija o código para que o teste passe

```
public class Shape {  
    public double calcSquareArea(double side) {  
        return side * side;  
    }  
}
```

4. **Finalmente**, teste outros casos de uso e, quando necessário (isto é, quando o teste falhar) ajuste o código conforme necessário

▼ Data Structures And Algorithms (DSA)

▼ Conceito e Motivação

- Dados primitivos estão presentes em todas as linguagens de programação: inteiro, float, char, booleano, etc;
- Estruturas de dados vêm para armazenar mais do que uma unidade daquele dado
- O estudo das Estrutura de Dados pretende armazenar, alterar e acessar os dados de forma eficiente quanto a **Tempo e Espaço**
- Existem alguns **algoritmos** que operam sobre essas estruturas de dados de forma mais eficiente
- A ideia do estudo de DSA é, finalmente, melhorar a experiência do usuário selecionado **quando usar o quê**
- A ideia também é reduzir os custos com poder computacional, especialmente para empresas que realizam processamento aaS
- Finalmente, o conhecimento de DSA é interessante para demonstrar conhecimento para grandes empresas

▼ Abstract Data Types

- Diferem dos tipos primitivos de dados quanto à complexidade e por poderem ser definidos pelo desenvolvedor
- Podem armazenar mais de um dado primitivo
- São objetos
- Podem ser chamados de User Data Type ou Complex Data Type
- Definem um conjunto de operações específicos

- Um array é um tipo novo de dado, pois é um conjunto de dados de um certo tipo, define suas formas de leitura, busca, adição, deleção, etc
- Qualquer tipo de dado, em especial quando armazena muitos dados e pode implementar um conjunto específico de operações, pode ser chamado **Abstract Data Type**
- Arrays, Queues, Stacks, etc, são conceitos, que são implementados de formas diferentes em cada linguagem, são **Abstract Data Type**
- Podemos criar outros além deles

▼ Arrays

- Sequencia contigua de dados com um único nome
- Ao programar uma variável “int a = 5”, armazenamos o dado em um espaço de 2 bytes na memória e armazenamos o endereço em a;
- Ao programar um array “int nums[5]” temos 10 bytes na memória e o endereço em nums é do primeiro item, ao usar o índice, incrementamos 2 bytes no endereço
- As operações possíveis no Array são:
 - Leitura, usando o índice “nums[2] → 2”, é rápida pois utilizamos o índice
 - Busca, exige comparação com todos os valores, lenta pois temos que comparar com n itens
 - Inserção no final é fácil, basta alocar mais espaço ao final do array, mas inserção no meio é mais lento pois temos que mover todos os itens depois do local um endereço depois, muito lento quando ocorre no meio
 - Deleção é idêntico à inserção, no final é rápido mas no meio precisamos mover tudo o que sobrou um endereço antes
- É importante se atentar para o tempo gasto com cada operação no Array
- Não pode ser calculado por tempo, pois processadores podem mudar, mas por número de iterações
- Um bom algoritmo precisa ser eficiente quanto ao tempo e quanto ao espaço

Algoritmos de Busca

▼ Linear Search (Busca Linear)

- Começamos com um array de 5 valores (5, 7, 9, 12, 17) que está ordenado
- A suposição é que queremos buscar o valor 12
- A busca linear se trata de analisar os valores 1 por 1 e conferir se é o valor 12

- O número de passos necessário depende diretamente do número de valores

```
for(int i = 0; i<array.length(); i++){
    if(array[i]==target){
        return i;
    }
}
```

- O número de iterações caso o valor seja o último ou não exista é o número de itens

▼ Binary Search (Busca Binária)

- Começamos com um array de 7 valores (5, 6, 7, 8, 9, 11, 13, 17) que está ordenado
- Queremos achar o 8
- O nome é binário pois dividimos o array em 2
- Se são 7 valores, acessaremos primeiro o quarto valor, no índice 3

```
int i = roundDown(array.length()/2);
```

- Como sabemos que o array está ordenado, se o valor no meio é menor do que o nosso, olhamos apenas para o que está depois dele. Se for maior, olhamos apenas para o que vem antes. Se for igual achamos.
- Nesse caso, o valor no meio é 9, 8 é menor do que nove, então olharemos para a primeira parte do array
- Em seguida, repetimos o procedimento. É importante notar que i não recebe mais a metade do array, mas é calculado assim:

```
i = roundDown((firstIndex+lastIndex)/2);
```

- Na segunda iteração, teremos o array 5, 6, 8, e vamos comparar o 8 com o valor no meio, que é 6
- Como 6 é menor, olharemos para o que vem a seguir
- Calculando o meio de novo, entre os índices 1 e 3, achamos 2, que é o índice do 8
- Ao compararmos, vemos que é o valor real

```
int left = 0;
int right = array.length() -1;
```

```

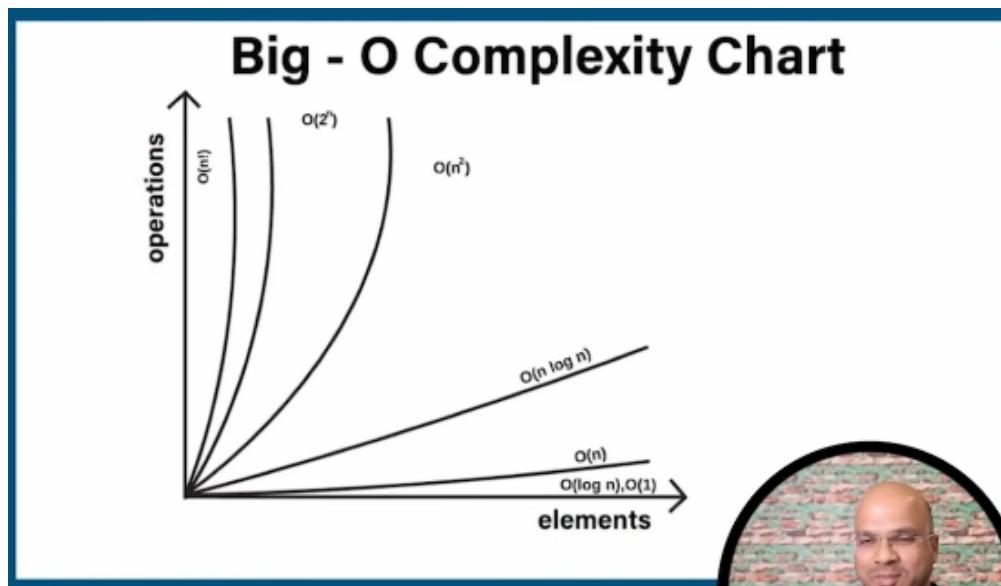
while (left<=right){
    mid = (left + right) /2;
    if array[mid] == target{
        return mid;
    } else if (array[mid] < target){
        left = mid + 1;
    } else if (array[mid] > target){
        right = mid - 1;
    }
}

```

- O resultado é que reduzimos significativamente o número de execução
- Por exemplo, com um array de 1000 elementos, na primeira comparação já eliminamos 500 comparações, 500 iterações

▼ Time Complexity (Big O)

- A notação Big O é utilizada para representar a complexidade temporal de um algoritmo
 - $O(1)$ é tempo constante,
 - $O(\log n)$ é tempo logarítmico,
 - $O(n)$ é tempo linear
 - $O(n \log n)$ linearithmic time
 - $O(n^2)$ é tempo quadrático
 - $O(2^n)$ é tempo exponencial
 - $O(n!)$ é tempo fatorial
 -



- Em um array, quando acessamos um dado, o tempo necessário para essa operação é constante, sempre demora o mesmo tempo, independente do tamanho do array e do índice. Nesse caso, a operação é $O(1)$
- Na busca linear, se temos n elementos e o que estamos buscando está no final ou não está (pior caso), fazemos n iterações. Portanto, a operação é $O(n)$
- Na busca binária, quando tivemos 7 elementos, precisamos de 3 iterações, mas se tivermos 14, precisaremos de só mais 1, indo para 4. Isso quer dizer que não é $O(n)$, mas também não é $O(1)$. Nesse caso, quando está entre os dois, o mais provável é ser $O(\log n)$
- É possível provar, ao invés de inferir que é $O(\log n)$, mas não vale o trabalho
-

Algoritmos de Ordenação

▼ Conceito e Motivação

- A ideia de estudar vários algoritmos diferentes de ordenação é encontrar alguns cujo tempo de execução não cresça exponencialmente com o número de dados
- A ordenação é importante em muitos momentos, e um deles é na busca: a busca binária é muito mais rápida do que a linear, mas depende de um conjunto ordenado
- Algumas opções:
 - Bubble Sort
 - Selection Sort

- Insertion Sort
- Merge Sort
- Quick Sort
- Counting Sort
- Radix Sort
- Heap Sort
- Bucket Sort

▼ Bubble Sort

- Bubble Sort não é eficiente, mas é simples de compreender
- Ex: 8, 6, 9, 2, 4, 5, alvo: 2, 4, 5, 6, 8, 9
- No Bubble Sort você cria uma bolha, pega o maior elemento e o faz “subir” para a última posição
- No exemplo, pegaríamos o 8 e comparariamos com 6, como 8 é maior, trocaríamos os dois
 - 6, 8, 9, 2, 4, 5
- Em seguida, continuariamos comparando de dois em dois.
- O próximo passo seria comparar 8 e 9, como 9 é maior, não trocamos. Como 9 é maior do que 2, contudo, trocamos
 - 6, 8, 2, 9, 4, 5
 - 6, 8, 2, 4, 9, 5
 - 6, 8, 2, 4, 5, 9
- Após chegar na última posição, confirmamos que o maior elemento está no topo. Subiu como uma “bolha”
- Agora precisamos repetir todo esse processo n vezes até que todos os números estejam nas suas devidas posições
- A próxima iteração, por exemplo, seria:
 - 6, 8, 2, 4, 5, 9
 - 6, 2, 8, 4, 5, 9
 - 6, 2, 4, 8, 5, 9
 - 6, 2, 4, 5, 8, 9
 - 6, 2, 4, 5, 8, 9

- É possível armazenar a posição do último valor que já subiu, para reduzir o tamanho das iterações seguintes. Contudo, ainda assim se percebe ser um algoritmo lento.
- Como são 2 laços (um externo e um interno), a complexidade é quadrática: **$O(n^2)$**

```

int nums[] = {6, 5, 2, 8, 9, 4};

for(int i = 0; i < nums.length ; i++){
    for(int j = 0; j < nums.length -i -1 ; j++){
        if(nums[j] > nums[j+1]){
            int temp = nums[j];
            nums[j] = nums[j+1];
            nums[j+1] = temp;
        }
    }
}

// 2, 4, 5, 6, 8, 9

```

▼ Selection Sort

- Além da complexidade quadrática, o Bubble Sort tem péssima performance devido à necessidade de trocar dois valores
- O Selection Sort também tem complexidade quadrática (**$O(n^2)$**) , mas não faz esse “swap” em todos os elementos, diminuindo o tempo gasto (e melhorando a performance quanto à memória)
- Ex: 8, 3, 2, 5, 7, 6
- O Selection Sort encontra primeiro quem é o valor mínimo do array (no caso 2) e o troca com o primeiro número
 - 2, 3, 8, 5, 7, 6
- Na segunda iteração, ele vai descobrir que o menor valor é o 3 e não vai alterar
- Em seguida, o menor é 5, então tem que ser trocado com o 8, e assim por diante
 - 2, 3, 5, 8, 7, 6
 - 2, 3, 5, 6, 7, 8
 - 2, 3, 5, 6, 7, 8

- 2, 3, 5, 6, 7, 8
- É possível inverter a lógica e encontrar o maior valor para colocá-lo no final, ao invés de usar o menor
- O problema dessa estratégia é o custo de encontrar o maior valor
- A forma padrão de fazer isso no selection sort é olhar para todos os números, comparando qual o maior entre 2, e guardando sempre o maior e o índice dele

```

public class SelectionSort {
    public static void main(String[] args) {
        int nums[] = { 6, 5, 2, 8, 9, 4 };

        for (int i : nums) {
            System.out.print(i + " ");
        }

        // Usando o valor máximo
        for (int i = nums.length - 1; i > 0 - 1; i--) {
            int maxIndex = 0;
            for (int j = 0; j < i; j++) {
                if (nums[j] > nums[maxIndex]) {
                    maxIndex = j;
                }
            }
            int temp = nums[i];
            nums[i] = nums[maxIndex];
            nums[maxIndex] = temp;

        }

        System.out.println();
        for (int i : nums) {
            System.out.print(i + " "); // 2, 4, 5, 6, 8, 9
        }
    }
}

```

▼ Insertion Sort

- É semelhante aos últimos dois algoritmos, mas, novamente, diminui o custo da modificação

- Ao invés de swap, aqui é feito o “shift”
- Ex: 5, 3, 4, 2, 1
- Não iniciamos no primeiro elemento, como nos outros
- Na verdade, no Insertion Sorte, precisamos considerar que já processamos um trecho do array e outro não, por isso, iniciamos considerando que o primeiro elemento (no caso 5) já foi processado, e olhamos para o segundo
- Olhando para o segundo elemento (3), primeiro o separamos numa variável temporária
 - 3, 5, 4, 2, 1
- Na iteração seguinte, guardamos o 4 numa variável temporária e comparamos com o 5
- Como o 5 é maior, movemos o cinco para a direita e comparamos com o 3
- Como o 3 é menor, inserimos o 4 onde o 5 estava
 - 3, 4, 5, 2, 1
- A análise do 2 é mais interessante
- Separamos o 2, numa variável temporária, então começamos as comparações
 - 3, 4, 5, _, 1 | 2
 - 3, 4, _, 5, 1 | 2
 - 3, _, 4, 5, 1 | 2
 - _, 3, 4, 5, 1 | 2
 - 2, 3, 4, 5, 1
- Finalmente com o 1 a ordenação é finalizada

```
public class InsertionSort {
    public static void main(String[] args) {
        int nums[] = { 6, 5, 2, 8, 9, 4 };

        for (int i : nums) {
            System.out.print(i + " ");
        }

        for (int i = 0; i < nums.length; i++) {
```

```

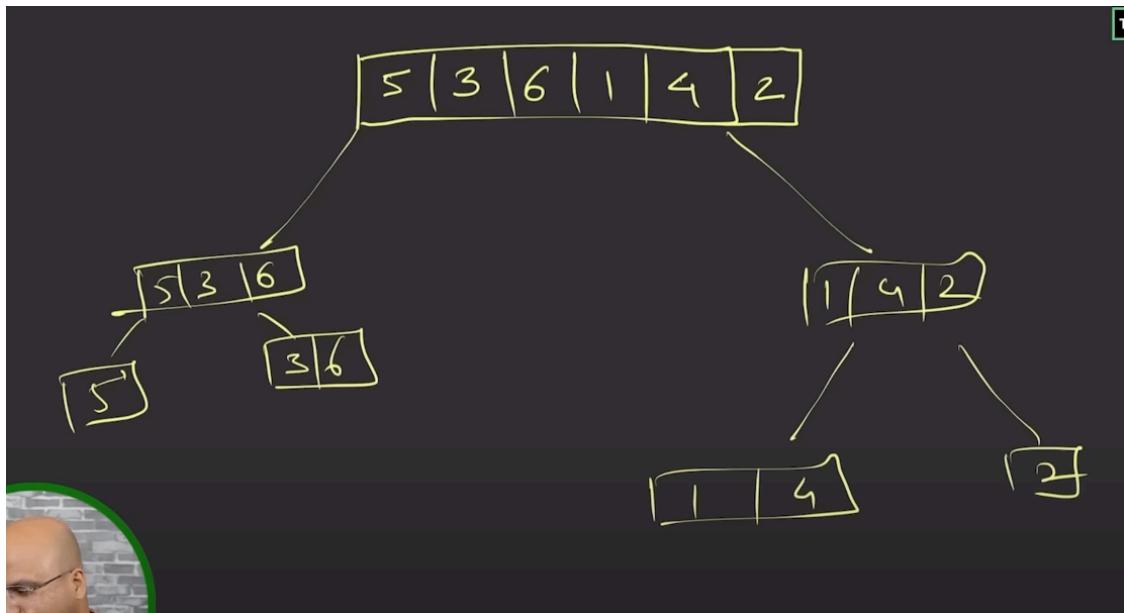
int key = nums[i];
int j = i - 1;
while (j >= 0 && nums[j] > key) {
    nums[j + 1] = nums[j];
    j--;
}
nums[j + 1] = key;
}

System.out.println();
for (int i : nums) {
    System.out.print(i + " "); // 2, 4, 5, 6, 8, 9
}
}
}
}

```

▼ Quick Sort

- Até agora os algoritmos eram **O(n^2)**, eles reduzem o tempo em cada iteração mas ainda crescem quadraticamente com o tamanho dos dados
- Quick Sort é **O($n \log n$)** no melhor caso e no caso médio, apesar de poder chegar a **O(n^2)** no pior caso.
- A primeira abordagem do Quick Sort é “Dividir e Conquistar”, reduzir o problema em problemas menores e resolver um de cada vez
- A segunda é a Recursão. Podemos usar recursão nos outros métodos, mas nesse é necessário
- A terceira é a ideia de Pivot, um ponto central de equilíbrio no problema (no caso, no array), a partir do qual começaremos a resolver o problema
- Finalmente, o Quick Sort também usa o conceito de árvore
- Ex: 5, 3, 6, 1, 4, 2
- Primeiro passo é dividir e conquistar:
- 5, 3, 6 | 1, 4, 2
- 5 | 3, 6 | 1, 4 | 2
- O resultado é uma espécie de árvore



- Depois disso faremos a recursão, ordenando cada passo de uma vez
- Essa divisão, contudo, não é a correta, e sim apenas um exemplo
- Da forma correta, primeiro precisamos identificar o pivot, um ponto no array no local correto, que já esteja na posição certa

Escolhendo o Pivô

- O padrão é escolher o último valor, nesse caso o 2, e depois posicioná-lo
- As outras variáveis relevantes são o i (inicia como -1), o j (inicia como 0), o l (de low, inicia no começo do array) e o h (de high, inicia no fim do array)
- Primeiro buscamos pelo array até achar um valor que seja menor que o pivô, no caso 1
- Então incrementamos i (anda pra posição 0) e trocamos o valor encontrado com o valor em i
 - 1, 3, 6, 5, 4, 2
- Continuamos buscando para ver se achamos outro valor que seja menor do que o pivô, nesse exemplo não achamos
- Concluida a busca, trocamos o valor em i+1 com o valor em high (nesse caso, 2, o pivô)
 - 1, 2, 6, 5, 4, 3
- Todos os valores à esquerda do pivô são menores do que ele, e todos à direita são maiores. Portanto, agora ele está na posição correta

Dividindo para Conquistar

- Sabendo, agora, que o pivô está na posição correta, podemos fazer a divisão
- Teremos 2 arrays: tudo antes do pivô e tudo depois do pivô, já que ele já está ordenado
 - $1 | 6, 5, 4, 3$
- Nesse exemplo, o 1 é um elemento só, então já está ordenado e na posição correta, não precisamos nos preocupar com ele
- Vamos agora realizar o Quick Sort no segundo array:
 - $6, 5, 4, 3$

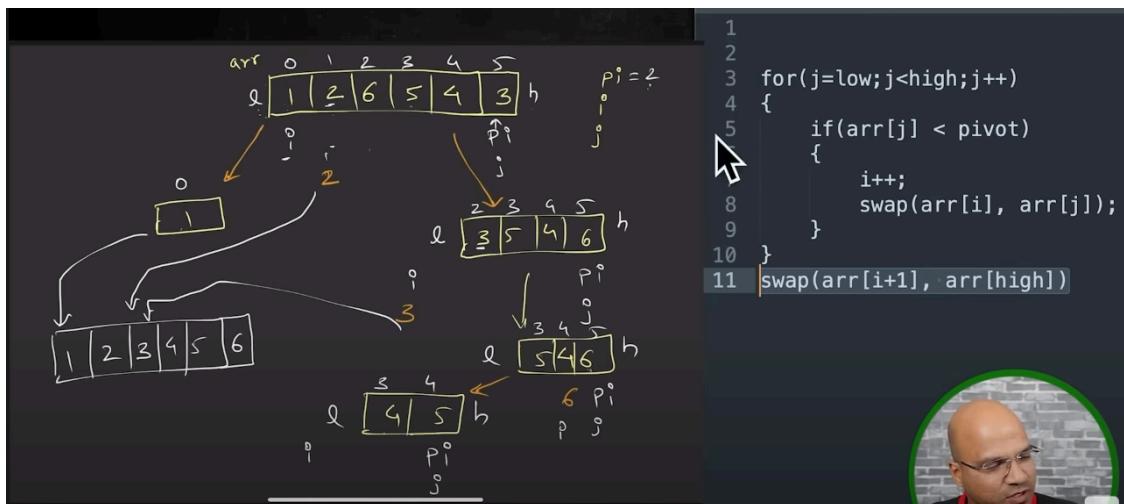
Recursão

- Aqui entra a recursão: vamos chamar novamente a função quickSort sobre esse segundo array, e fazer exatamente o mesmo procedimento
- O pivô é o último número, 3; $i = -1$, $j = 0$, $l = \text{começo do array}$, $h = \text{final do array}$
- Comparamos todos os números com ele até achar algum menor, não tem
- Depois de todas as comparações, trocamos $i+1$ com $high$ e temos:
 - $3, 5, 4, 6$
- O pivô está ordenado, e portanto podemos separar novamente
- Como o pivô está no começo, só temos o array da direita, onde executamos mais um passo
 - $5, 4, 6$
- O pivô é 6
- Comparamos com 5, que é menor
- Incrementamos i (vira 0) e trocamos 5 com quem está em i - já está no lugar
- Comparamos com 4, também é menor, mas ao incrementarmos i vira 1 e já está no lugar de novo
- Novamente, então, trocamos quem está em $i+1$ (no caso, 2) com o pivô. Mais uma vez, já está no lugar
- No final das contas, essa operação não mexeu no array
 - $5, 4, 6$
- Mas é coerente porque, de fato, se o pivô é 6, ele já está no lugar correto, no final
- Agora reduzimos novamente para apenas o array da esquerda

- 5, 4
- O pivô é o 4
- Comparamos 5 com 4, não é maior, encerramos as comparações
- Trocamos i+1 com o pivô
 - 4, 5

União dos Elementos

- No final das contas, ficamos com uma árvore de vários arrays
- A mágica é que, na verdade, nunca criamos um novo array, a forma correta é utilizar o mesmo array, apenas limitando as extremidades
- Dessa forma, todos os elementos finalmente estão ordenados em suas posições corretas



Código

```

public class QuickSort {

    public static void main(String[] args) {
        int nums[] = { 6, 5, 2, 8, 9, 4 };

        for (int i : nums) {
            System.out.print(i + " ");
        }

        quickSort(nums, 0, nums.length - 1);

        System.out.println();
    }
}

```

```

        for (int i : nums) {
            System.out.print(i + " ");
        }
    }

public static void quickSort(int[] nums, int low, int high) {
    if (low < high) {
        int pivot = partition(nums, low, high);

        quickSort(nums, low, pivot - 1);
        quickSort(nums, pivot + 1, high);
    }
}

// Função que encontra o pivô, esse é o nome padrão
public static int partition(int[] nums, int low, int high) {
    int pivot = nums[high];
    int i = low - 1;
    int swapTemp;

    for (int j = low; j < high; j++) {
        if (nums[j] < pivot) {
            i++;
            swapTemp = nums[i];
            nums[i] = nums[j];
            nums[j] = swapTemp;
        }
    }

    swapTemp = nums[i + 1];
    nums[i + 1] = nums[high];
    nums[high] = swapTemp;

    return i + 1;
}
}

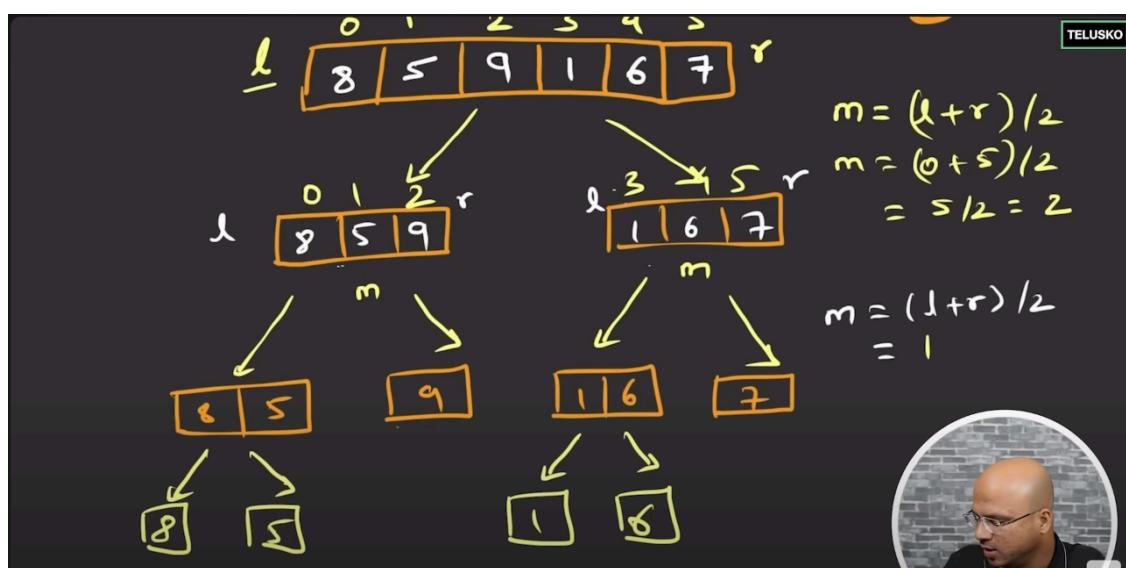
```

▼ Merge Sort

- Assim como Quick Sort, tem uma complexidade de $n \log n$
- Também aborda com Dividir e Conquistar
- Ex: 8, 5, 9, 1, 6, 7

Divisão

- Diferente do Quick Sort, não precisamos identificar um pivô
- Dividimos no meio do array (se for ímpar, tanto faz deixar na esquerda ou na direita)
 - $8, 5, 9 | 1, 6, 7$
- Seguimos dividindo no meio
 - $8, 5 | 9 | 1, 6 | 7$
 - $8 | 5 | 9 | 1 | 6 | 7$
- O resultado é uma espécie de árvore

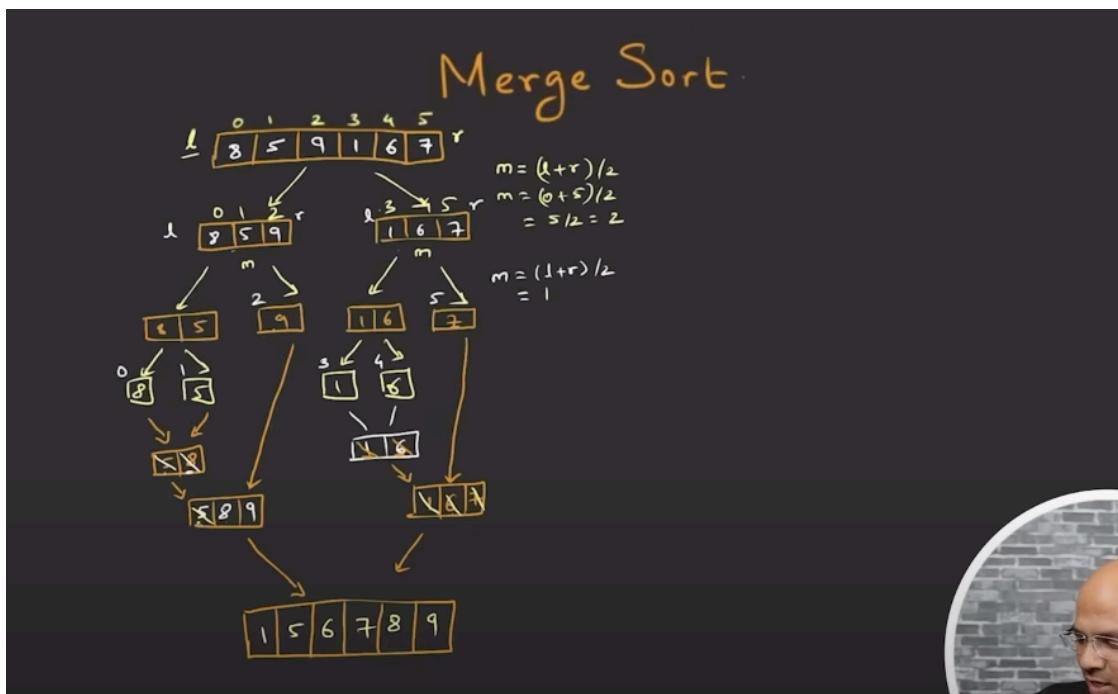


- Na prática, não criamos novos arrays, então podemos manter os índices
- A ideia é que, cada uma das folhas da árvore já está ordenada, então o trabalho é fazer o "merge" mantendo a ordenação

Merge

- Ao unir 8 e 5, colocamos o menor primeiro e o maior depois
 - $5, 8 | 9 | 1 | 6 | 7$
- Em seguida, quando temos o 9, fazemos o merge colocando o 9 na posição correta comparando 2 a 2
 - $5, 8, 9 | 1 | 6 | 7$
- O 1, o 6 e o 7 estão de outro lado da árvore, temos que seguir a sequência e ordem que está nela pois representa como funcionará na recursão
- Então, vamos primeiro unir 1 e 6

- $5, 8, 9 | 1, 6 | 7$
- E depois incluimos o 7
 - $5, 8, 9 | 1, 6, 7$
- Agora a forma de fazer o merge fica mais clara, como funciona a comparação 2 a 2
- Olharemos para o primeiro elemento de cada array e, comparando, incluiremos no “novo array” o menor
 - $5 > 1 \rightarrow 1, -, -, -, -, -$
 - $5 < 6 \rightarrow 1, 5, -, -, -, -$
 - $8 > 6 \rightarrow 1, 5, 6, -, -, -$
 - $8 > 7 \rightarrow 1, 5, 6, 7, -, -$
- Como o segundo array acabou, podemos apenas incluir o resto dos elementos na ordem em que estão
 - $1, 5, 6, 7, 8, 9$



```
public class MergeSort {
    public static void main(String[] args) {
        int nums[] = { 6, 5, 2, 8, 9, 4 };

        for (int i : nums) {
            System.out.print(i + " ");
        }
    }
}
```

```

}

mergeSort(nums, 0, nums.length - 1);

System.out.println();
for (int i : nums) {
    System.out.print(i + " ");
}
}

private static void mergeSort(int[] nums, int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2;
        mergeSort(nums, left, mid);
        mergeSort(nums, mid + 1, right);

        merge(nums, left, mid, right);
    }
}

private static void merge(int[] nums, int left, int mid, int right) {
    int leftNums[] = new int[mid - left + 1];
    int rightNums[] = new int[right - mid];

    for (int i = 0; i < leftNums.length; i++) {
        leftNums[i] = nums[left + i];
    }

    for (int i = 0; i < rightNums.length; i++) {
        rightNums[i] = nums[mid + 1 + i];
    }

    int i = 0;
    int j = 0;
    int k = left;

    while (i < leftNums.length && j < rightNums.length) {
        if (leftNums[i] <= rightNums[j]) {
            nums[k] = leftNums[i];
            i++;
        } else {
            nums[k] = rightNums[j];
        }
    }
}

```

```

        j++;
    }
    k++;
}
}

while (i < leftNums.length) {
    nums[k] = leftNums[i];
    i++;
    k++;
}

while (j < rightNums.length) {
    nums[k] = rightNums[j];
    j++;
    k++;
}

}
}

```

Estruturas de Dados

▼ Linked List (Lista Encadeada)

- Um problema importante do array é que, depois que você o cria, ele tem um tamanho fixo, que não pode crescer nem diminuir
- Ex: 12, 6, 8, 3
- Podemos armazenar esses dados num array, mas não poderíamos inserir um novo
- A solução é utilizar cada dado com uma estrutura de dado e referência
- Esse dado é chamado de **nó**, e cada um tem duas informações: o valor armazenado e o endereço do próximo nó
- O primeiro nó é chamado de **Head**
- Supondo que esse seja o valor e o endereço na memória de cada dado:
 - 12 na memória 512
 - 6 na memória 326
 - 8 na 101
 - 3 na 202

- Então a estrutura do nó seria:
 - 12 | 326
 - 6 | 101
 - 8 | 202
 - 3 | null
- Onde uma referência a null indica o final da lista
- Essas referências são chamadas de Link (ou encadeamento, no português)

Inserção

- Para inserir um novo nó no final, precisamos apenas criá-lo
 - 7 | null (na memória 126)
- E armazenar o endereço no último nó da lista
 - 3 | 126
- Para inserir o valor no começo apenas criamos um nó, apontamos sua referência para a Head e mudamos a Head para ser ele
 - Head == 512
 - Node <= 5 | 512 (na memória 306)
 - Head <= 306
- Para inserir no meio, criamos o nó com uma memória. Pegamos o elemento que virá antes e mudamos sua referência para o novo endereço. Pegamos a antiga referência do anterior e inserimos no novo nó
 - Node <= 9 | 101 (na memória 506)
 - 6 | 101 <= 6 | 506
- Importante notar que os endereços são “aleatórios”, isto é, não precisamos que eles sejam contíguos
- É possível que as listas sejam “double linked” ou duplamente encadeadas, caso os nós tenham referência do próximo E do anterior
- A linked list é mais lenta do que o array, mas tem tamanho dinâmico
- A busca na lista encadeada é mais complicada, pois para saber qual o item no terceiro índice eu preciso ir de nó em nó até o terceiro item

Implementação

```
public class LinkedList {  
    Node head;  
  
    public class Node {  
        int data;  
        Node next;  
  
        public Node(int data, Node next) {  
            this.data = data;  
            this.next = next;  
        }  
    };  
  
    public static void main(String[] args) {  
        LinkedList list = new LinkedList();  
  
        list.show(); // Lista vazia  
  
        list.insert(3);  
        list.insert(4);  
        list.insert(6);  
  
        list.show(); // 3 → 4 → 6 → NULL  
  
        list.insertAt(5, 2);  
  
        list.show(); // 3 → 4 → 5 → 6 → NULL  
  
        list.insertStart(2);  
        list.insertStart(1);  
  
        list.show(); // 1 → 2 → 3 → 4 → 5 → 6 → NULL  
  
        list.deleteAt(2);  
  
        list.show(); // 1 → 2 → 4 → 5 → 6 → NULL  
  
    }  
  
    public void show() {  
        if (head == null) {  
            System.out.println("Lista vazia");  
        }  
    }  
}
```

```

        return;
    }

Node cur = head;
do {
    System.out.print(cur.data + " → ");
    cur = cur.next;
} while (cur != null);
System.out.println("NULL");
}

// insere no final
public void insert(int data) {
    Node newNode = new Node(data, null);

    if (head == null) {
        head = newNode;
        return;
    }

    Node cur = head;
    while (cur.next != null) {
        cur = cur.next;
    }

    cur.next = newNode;
}

public void insertAt(int data, int index) {
    Node newNode = new Node(data, null);
    int counter = 0;

    if (head == null) {
        head = newNode;
        return;
    }

    if (index == 0) {
        newNode.next = head;
        head = newNode;
        return;
    }
}

```

```

        Node cur = head;
        while (counter != (index - 1) && cur.next != null) {
            cur = cur.next;
            counter++;
        }
        newNode.next = cur.next;
        cur.next = newNode;
    }

    public void insertStart(int data) {
        Node newNode = new Node(data, head);
        head = newNode;
    }

    public void deleteAt(int index) {
        if (head == null) {
            return;
        }

        Node cur = head;
        int counter = 0;
        while (counter < (index - 1) && cur.next != null) {
            counter++;
            cur = cur.next;
        }

        if (counter < (index - 1)) {
            return;
        }

        cur.next = cur.next.next;
    }
}

```

▼ Stack (Pilha)

- Stack é uma ADT, um Abstract Data Type, ou seja, implementa funções como adição, remoção e busca (push, pop e peek)
- Stack é **LIFO - Last In First Out**, o último elemento inserido é o primeiro a sair
- É como uma lista de livros, para tirar o de baixo temos que tirar todos os que estão em cima

- Portanto, a inserção só pode ser no topo, a remoção só pode ser do topo e mesmo a função peek, de busca, só podemos olhar o item no topo
- É possível implementar com array, listas dinâmicas, listas encadeadas, etc
- Um detalhe da implementação em arrays é que, como não removemos elementos e o tamanho é fixo, é importante manter uma variável top que armazene o topo da lista

```

package structures;

public class Stack {
    int nums[] = new int[5];
    int top = -1;

    public static void main(String[] args) {
        Stack s = new Stack();

        s.push(2);

        s.show();

        s.push(6);
        s.push(9);

        s.show();

        s.pop();

        s.show();
    }

    public void show() {
        for (int i = nums.length - 1; i >= 0; i--) {
            String isTop = i == top ? " ← top" : "";
            System.out.println(nums[i] + isTop);
        }
        System.out.println();
    }

    public void push(int data) {
        if (top < 4) {
            top++;
        }
    }
}

```

```

        nums[top] = data;
        return;
    }
    System.out.println("Overflow");
}

public int pop() {
    if (top >= 0) {

        int topEl = nums[top];
        nums[top] = 0;
        top--;
        return topEl;
    }
    System.out.println("Underflow");
    return -1;
}

}

```

▼ Queue (Fila)

- A fila também é uma ADT
- Diferente da Stack, ela trabalha com **LIFO - Last In First Out**
- Semelhante à fila do banco: quem chegou primeiro sai primeiro
- Pode ser implementado com diversas estruturas de dados: arrays, linked list, etc
- Os termos para inserção e remoção são diferentes da stack: enqueue e dequeue
- A implementação a seguir não é a melhor, a forma correta é utilizando uma fila circular para não exceder os limites do array, mas ela explica o conceito:

```

package structures;

public class Queue {
    int queue[] = new int[5];
    int size = 0;
    int front = 0;
    int rear = 0;

    public static void main(String[] args) {

```

```
Queue q = new Queue();

q.enQueue(3);
q.show() // 3 0 0 0 0

q.enQueue(4);
q.enQueue(5);
q.enQueue(6);

q.deQueue();

q.show() // 0 4 5 6 0
}

public void enQueue(int data) {
    if (rear < 5) {
        queue[rear] = data;
        rear++;
        size++;
    } else {
        System.out.println("Overflow");
    }
}

public int deQueue() {
    if (front <= rear) {
        int ret = queue[front];
        queue[front] = 0;
        front++;
        size--;
        return ret;
    }
    return -1;
}

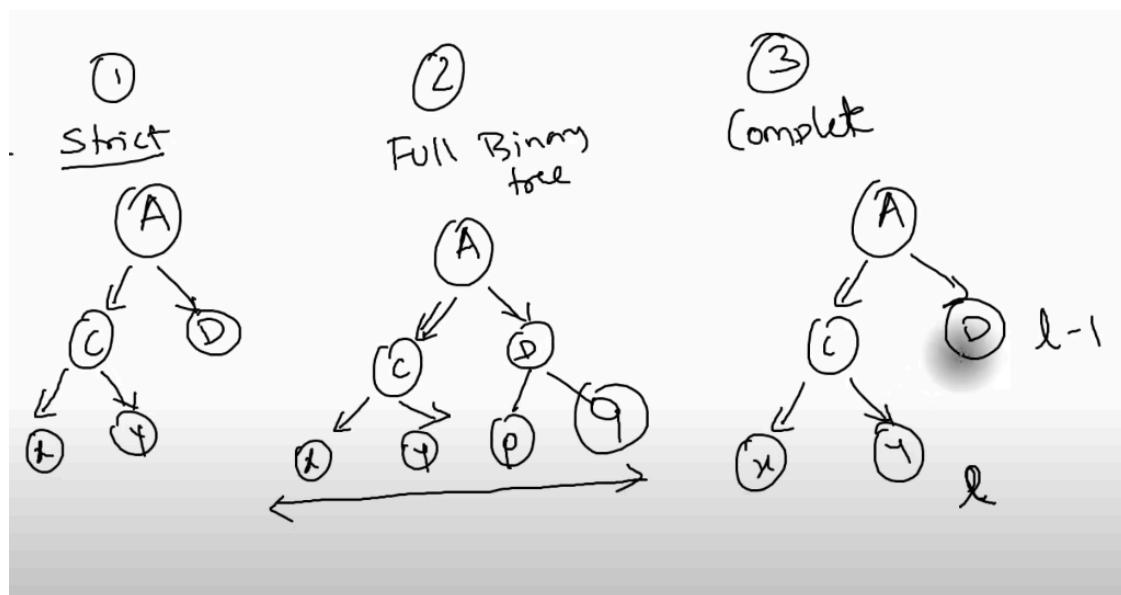
public void show() {
    for (int i : queue) {
        System.out.print(i + " ");
    }
    System.out.println();
}
}
```

▼ Tree (Árvore)

- A árvore também é uma ADT, muito semelhante à Linked List no sentido que um nó terá referência para outro nó
- A diferença é que um nó pode ter mais de uma referência, a vários nós diferentes
- Utilizamos alguns nomes que fazem referência a árvore:
 - Cada item é um Nó (Node)
 - Cada ligação ou referência é uma Borda (Edge)
 - O primeiro item, mais "acima" é a Raiz (Root)
 - Os últimos items, que não fazem referência a ninguém, são Folhas (Leaf)
- É também possível utilizar termos genéticos como Pai, Filho, etc

Árvore Binária (Binary Tree)

- A árvore binária é um tipo mais específico de árvore, em que cada nó só pode ter, no máximo, duas referências (ou cada pai, dois filhos)
- Uma **Strict Binary Tree** é uma árvore binária em que cada nó tem 0 ou 2 filhos, não pode ter um só
- Uma **Full Binary Tree** é uma árvore binária em que todos os nós folha estão no mesmo nível
- Uma **Complete Binary Tree** é uma árvore binária em que todos os nós folhas estão no mesmo nível ou, no máximo, 1 nível acima



- Dois conceitos relevantes também são:

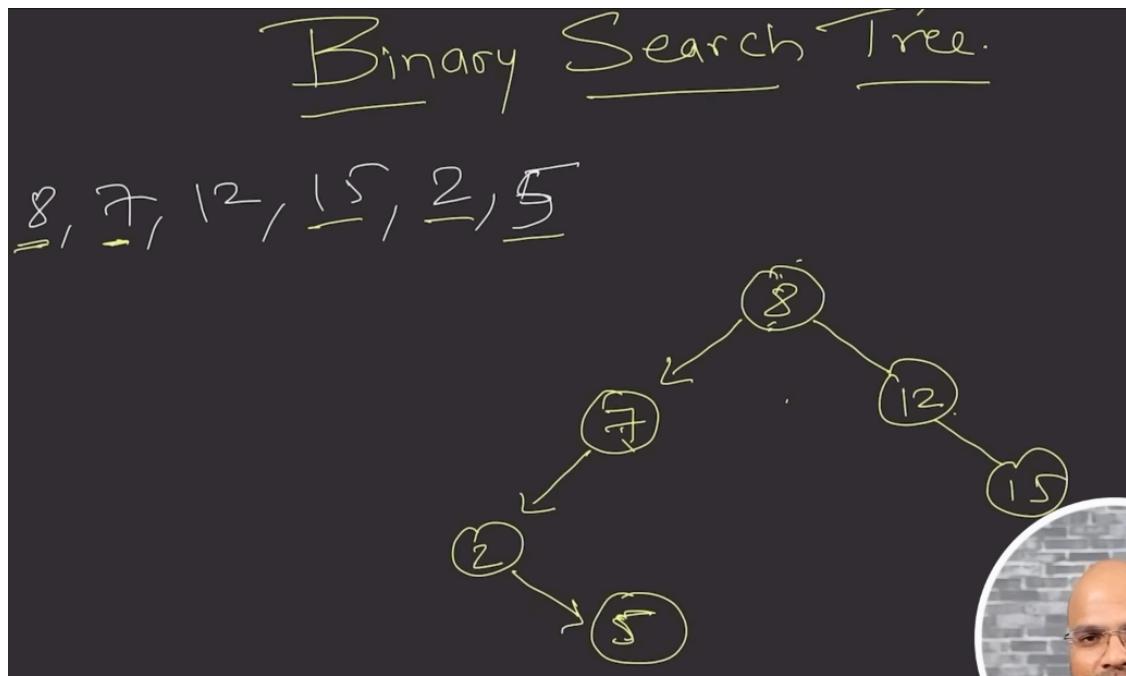
- **Depth:** a profundidade um nó é o número de bordas entre ele e a raiz
- **Height:** a altura de um nó é o número de bordas entre ele e seu “filho mais distante” ou a última folha do seu galho
- **Tree Height:** a altura de uma árvore é a altura da raiz

Traverse

- A ideia do Traverse é percorrer a árvore, normalmente com o intuito de visualizar a árvore
- Existem 3 formas de percorrer a árvore binária. Para explicar, usaremos o exemplo da árvore com raiz 8, filho às esquerda 7 e filho à direita 12
 - In Order - da esquerda para a direita, então 7, 8, 12 (resulta em valores ordenados)
 - Pre Order - da raiz primeiro, depois da esquerda para a direita: 8, 7, 12
 - Post Order - primeiro os filhos, da esquerda para a direita, depois o pai: 7, 12, 8

Binary Search Tree

- A árvore de busca binária é uma árvore binária com o propósito de facilitar a busca
- Nesse caso utilizaremos valores inteiros como o valor do nó
- O conceito é que valores menores são adicionados à esquerda e valores maiores à direita. Todos os elementos à esquerda de um nó são menores do que ele, à direita serão maiores
- A ideia conversa bem com a busca binária



```

package structures;

public class BinaryTree {

    Node root;

    public static void main(String[] args) {
        BinaryTree binaryTree = new BinaryTree();

        binaryTree.insert(3);
        binaryTree.insert(2);
        binaryTree.insert(4);

        binaryTree.show();
        /*
         * Data: 2 Left: NULL Right: NULL
         * Data: 3 Left: 2 Right: 4
         * Data: 4 Left: NULL Right: NULL
         */
    }

    public class Node {
        int data;
        Node left;
        Node right;
    }
}
  
```

```

public Node(int data) {
    this.data = data;
}

public String toString() {
    return "Data: " + this.data + " Left: " + (this.left == null ? "NULL"
        : this.left.data) + " Right: " + (this.right == null ? "NULL" : String.value
    )
}

private Node insert(Node parent, int data) {
    if (parent == null) {
        parent = new Node(data);
    } else if (data < parent.data) {
        parent.left = insert(parent.left, data);
    } else if (data > parent.data) {
        parent.right = insert(parent.right, data);
    }
    return parent;
}

public void insert(int data) {
    root = insert(root, data);
}

public void show() {
    if (root != null)
        inorderRecursive(root);
    else
        System.out.println("Árvore vazia");
}

private void inorderRecursive(Node parent) {
    if (parent.left != null) {
        inorderRecursive(parent.left);
    }

    System.out.println(parent);

    if (parent.right != null) {
        inorderRecursive(parent.right);
    }
}

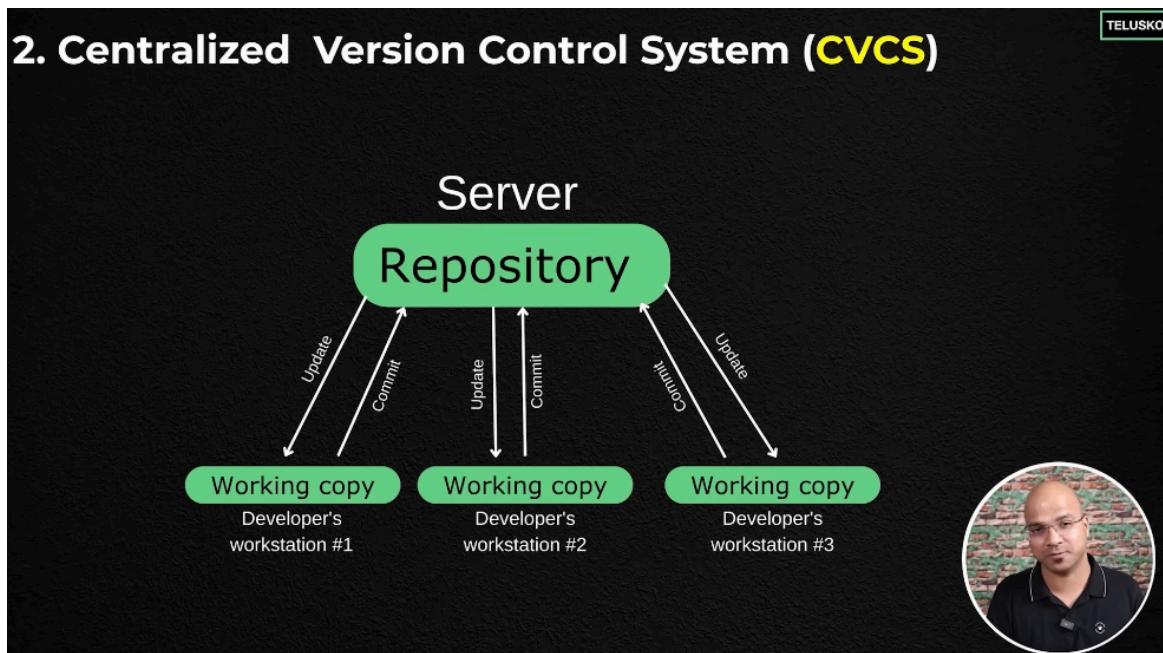
```

```
}
```

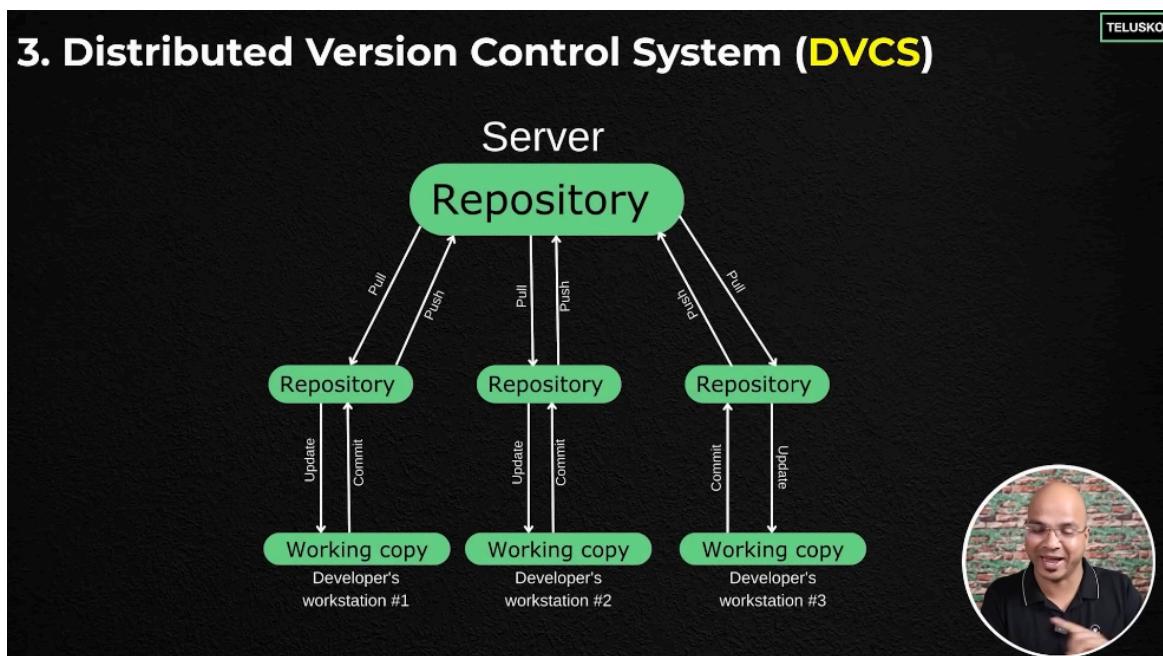
```
}
```

▼ GIT

- Versionamento e Colaboração
- Distributed Version Control System



TELUSKO



TELUSKO



▼ JDBC

- O ponto central do mundo tecnológico são os dados
- Temos que armazenar os dados em um banco de dados
- Ao desenvolver um software em qualquer linguagem, precisamos de uma forma de acessar o banco de dados, precisamos de uma conexão com o banco para poder acessá-lo
- A conexão entre o Java e o banco de dados é o **Java Data Base Connectivity (JDBC)**
- São necessários **7 passos** para conectar o Java ao banco de dados com JDBC
 1. Importar o pacote
 - a. `java.sql.*`
 2. Carregar e registrar o Driver
 - a. o driver depende do banco, no caso de MySQL é `com.mysql.jdbc.driver`, que baixamos de um .jar
 - b. para registrar, usamos `Class.forName("com.mysql.jdbc.driver")`;
 3. Estabelecer a conexão
 - a. Devemos instanciar uma interface `Connection`
 4. Criar o Statement
 - a. Pode ser Normal Statement (ou só Statement), Prepared Statement e Callable Statement
 - b. O Statement permite a executar uma query
 - c. Se temos uma query com variáveis, usamos a PreparedStatement
 - d. Quando vamos executar uma Stored Procedure ou PL (Procedural Language) usamos o CallableStatement
 5. Executar a Query
 6. Processar os Resultados
 7. Fechar a conexão

```
import java.sql.*;

/*
Imaginando uma tabela de estudantes com id e nome
*/
```

```

main(){
    Class.forName("com.mysql.jdbc.Driver");
    Connection con = DriverManager.getConnection(URL, Username, Password);
    Statement st = con.createStatement();
    ResultSet rs = st.executeQuery("SELECT * FROM student");
    while(rs.next()){
        Sysout(rs.getInt(1) + " " + rs.getString(2));
    }
    st.close();
    con.close();
}

```

- No SQL existem 3 tipos principais de query:
 - DDL - na hora de criar uma tabela
 - DML - na hora de inserir dados
 - DQL - na hora de buscar dados
- Para DQL usamos executeQuery, para DML usamos executeUpdate

```

package com.example.jdbc;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;

public class Main {
    public static void main(String[] args) throws Exception {
        // Class.forName("com.mysql.jdbc.Driver");

        String url = "jdbc:mysql://localhost:3306/mysqlteste";
        String uname = "douglas";
        String pass = "123";
        String dql = "SELECT * FROM estudantes";

        Connection con = DriverManager.getConnection(url, uname, pass);
        Statement st = con.createStatement();
        ResultSet rs = st.executeQuery(dql);
    }
}

```

```

while (rs.next()) {
    System.out.println(rs.getInt("id") + ":" + rs.getString("nome") + "(" + rs.get
}

/*
 * 1: João Silva (8.5)
 * 2: Maria Oliveira (9.2)
 * 3: Carlos Santos (7.3)
 */

// Insert sem parametros
String dml = "INSERT INTO estudantes VALUES (4, 'José Roberto', 9.8)";
int count = st.executeUpdate(dml);
System.out.println(count + " rows affected"); // 1 rows affected

// Insert com parâmetros
int id = 5;
String nome = "Joana Elisa";
double nota = 7.4;
String dmlWithParams = "INSERT INTO estudantes VALUES (?,?,?)";
PreparedStatement preparedStatement = con.prepareStatement(dmlWithPara
preparedStatement.setInt(1, id);
preparedStatement.setString(2, nome);
preparedStatement.setDouble(3, nota);
count = preparedStatement.executeUpdate();
System.out.println(count + " rows affected"); // 1 rows affected

rs = st.executeQuery(dql);

while (rs.next()) {
    System.out.println(rs.getInt("id") + ":" + rs.getString("nome") + "(" + rs.get
}

/*
 * 1: João Silva (8.5)
 * 2: Maria Oliveira (9.2)
 * 3: Carlos Santos (7.3)
 * 4: José Roberto (9.8)
 * 5: Joana Elisa (7.4)
 */

```

```
        st.close();
        con.close();
    }
}
```

Com Design Patterns DAO

- A motivação é que queremos utilizar boas práticas na hora de buscar os dados do banco
- Uma boa prática é o uso de **DAO - Data Access Object**, que são classes feitas exclusivamente para se comunicar com o banco, devem replicar identicamente a estrutura da tabela

```
package com.example.jdbc;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;

public class ComDesignPattern {
    public static void run() throws Exception {

        System.out.println();
        System.out.println();

        StudenDAO dao = new StudenDAO();
        dao.connect();

        Student s = dao.getStudent(2);

        System.out.println(s.nome);

        Student novo = new Student();
        novo.id = 7;
        novo.nome = "Rafael Medeiros";
        novo.nota = 6.1;
        dao.addStudent(novo);

        Student buscado = dao.getStudent(7);
```

```

        System.out.println(buscado.nome);

    }

public static class Student {
    int id;
    String nome;
    double nota;
}

public static class StudenDAO {

    Connection con;

    public void connect() throws Exception {
        String url = "jdbc:mysql://localhost:3306/mysqlteste";
        String uname = "douglas";
        String pass = "123";
        con = DriverManager.getConnection(url, uname, pass);
    }

    public Student getStudent(int id) {
        Student s = new Student();
        s.id = id;

        try {
            String query = "SELECT nome, nota FROM estudantes WHERE id=" + id;

            Statement st = con.createStatement();
            ResultSet rs = st.executeQuery(query);

            rs.next();
            s.nome = rs.getString(1);
            s.nota = rs.getDouble(2);

        } catch (Exception e) {
            System.out.println(e);
        }

        return s;
    }
}

```

```

public void addStudent(Student student) {
    try {

        String query = "INSERT INTO estudantes VALUES (?,?,?)";

        PreparedStatement pst = con.prepareStatement(query);
        pst.setInt(1, student.id);
        pst.setString(2, student.nome);
        pst.setDouble(3, student.nota);

        pst.executeUpdate();
    } catch (Exception e) {
        System.out.println("1" + e);
    }
}
}

```

▼ Servlet & JSP

▼ Core

- Na web, o cliente pede uma página para o servidor
- Essa página pode ser estática (pronta) ou dinâmica (criada em runtime)
- Quando fazemos um request de uma página pronta, o arquivo html já está pronto no servidor
- Quando fazemos um request de uma página dinâmica, a página tem que ser criada
- Quem cria a página é uma aplicação chamada “Helper Application” ou “WebContainer” (e.g., Tomcat)
- Esse request vai para um Servlet, uma classe que herda HttpServlet e vai processar o request
- Quem determina, no servidor, qual o Servlet que será chamado é o Deployment Descriptor do Tomcat, um arquivo chamado web.xml
- É possível utilizar Annotations para mapear as requests ao invés do xml

Setup

- O vídeo usa o Eclipse, que em teoria tem um método de setup facilitado específico. Spoiler: o do VSCode é melhor
1. Crie um projeto Maven com o Command Pallette
 2. Escolha o template de webapp
 - a. O resultado nem vai ter uma main, mas vai ter uma pasta WEB-INF com o web.xml e, fora dela, vai ter um index.jsp
 - b. Esse index.jsp ainda não foi abordado no curso, então pode ser substituído por um index.html por motivos didáticos
 3. <https://stackoverflow.com/questions/76013263/how-can-we-create-a-dynamic-web-project-in-vs-code>
 - a. Seguindo os passos do link, instale a extensão Community Server Connectors
 - b. Com a Command Pallette, use Create New Server
 - c. A forma mais fácil é baixar o tomcat direto pelo VSCode, mas é possível baixar por fora, só tomar cuidado porque dá trabalho se deixar num local de difícil permissão (o VSCode vai ter que acessar)
 4. Builde o projeto (do jeito que está serve) com o mvn package
 5. Clique com o botão direito no .war e clique em "Run on Server"
 6. Escolha o Server que acabou de criar
 7. Botão direito no servidor (fica na parte de baixo, na mesma aba do explorador de arquivos) e Server Actions → Show on Browser
 8. Selecione o endereço com o nome do novo projeto
 9. Tcharam

Página Dinâmica

- Um exemplo de página dinâmica que vamos utilizar é uma calculadora:

```
<!DOCTYPE html>
<html lang="en">
<body>
  <h1>Calculator</h1>
  <form action="add">
    Enter the 1st numer: <input type="text" name="num1" /> <br />
    Enter the 2nd numer: <input type="text" name="num2" /> <br />
    <input type="submit" />
  </form>
```

```
</body>
</html>
```

- Ao clicar no submit/enviar, tentamos acessar o endereço:
<http://localhost:8080/servlet/add?num1=1&num2=2>
- Contudo, como ainda não fizemos o endereço add, retorna 404

Criando Servlet

- O primeiro passo (não registrado no vídeo) é incluir o pacote jakarta.servlet (não javax servlet, desde o tomcat 10) no projeto através do pom. Esse é o código:

```
<!-- https://mvnrepository.com/artifact/jakarta.servlet/jakarta.servlet-api -->
<dependency>
    <groupId>jakarta.servlet</groupId>
    <artifactId>jakarta.servlet-api</artifactId>
    <version>6.0.0</version>
    <scope>provided</scope>
</dependency>
```

- Em seguida criamos a classe AddServlet

```
package com.servlet.add;

import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

public class AddServlet extends HttpServlet {
    public void service(HttpServletRequest req, HttpServletResponse res) {
        int i = Integer.parseInt(req.getParameter("num1"));
        int j = Integer.parseInt(req.getParameter("num2"));

        int k = i + j;
        try {
            res.getWriter().println("result is: " + k);
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

- Depois disso, é necessário mapear essa classe no web.xml

```
<web-app>
<servlet>
    <servlet-name>add</servlet-name>
    <servlet-class>com.servlet.add.AddServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>add</servlet-name>
    <url-pattern>/add</url-pattern>
</servlet-mapping>
</web-app>
```

Verbo HTTP e doGet+doPost

- Por padrão, toda vez que não definimos, na tag form do html, o tipo de ação HTTP, o padrão é GET
- Se quisermos utilizar o POST para que não deixar os dados no url, precisamos alterar o método

```
<!DOCTYPE html>
<html lang="en">
    <body>
        <h1>Calculadora</h1>
        <form action="add" method="post">
            Enter the 1st numer: <input type="text" name="num1" /> <br />
            Enter the 2nd numer: <input type="text" name="num2" /> <br />
            <input type="submit" />
        </form>
    </body>
</html>
```

- Dessa forma, quando submetemos o cálculo, tudo funciona normalmente, mas os argumentos não ficam expostos na URL
- É possível, no código Java do Servlet forçar o uso apenas do verbo GET ou POST, mudando o nome da função

```
public void doPost(HttpServletRequest req, HttpServletResponse res) {
```

- Agora o cálculo funciona com POST, mas se tentarmos mandar um GET resulta em erro 404
- Na prática, a função chamada é service, mas internamente ela analisa o que foi implementado na função

HttpServletRequest & HttpServletResponse

- Ao fazer a comunicação entre Cliente e Servidor, é necessário enviar dados em algum formato
- Como estamos usando Java, esse formato é como objetos das classes HttpServletRequest (chamarei de Request) e HttpServletResponse (chamarei de Response)
- A resposta pode ter dados de texto, html, vídeo, etc.
- O Tomcat se responsabiliza por criar os dois objetos e passá-los no método service do HttpServlet

▼ Calling a Servlet from a Servlet

RequestDispatcher: Calling a Servlet from a Servlet

- A ideia aqui é poder chamar outro Servlet de dentro de um Servlet
- Primeiro, vamos voltar a usar o método get

```
public void doGet(HttpServletRequest req, HttpServletResponse res) {
```

```
<form action="add" method="get">
```

- Usaremos o exemplo de, além da adição, fazer o quadrado do número em um outro Servlet. Primeiro vamos apenas chamar o servlet com retorno simples

```
package com.servlet.add;
```

```
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
```

```
public class SquareServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) {
        try {
            res.getWriter().println("Square Servlet here!");
        }
```

```

        } catch (Exception e) {
            System.out.println(e);
        }
    }
}

```

- Podemos chamar um Servlet de dentro do outro por meio de redirecionamento ou com o RequestDispatcher, vamos usar a segunda opção:

```

package com.servlet.add;

import jakarta.servlet.RequestDispatcher;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

public class AddServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) {
        int i = Integer.parseInt(req.getParameter("num1"));
        int j = Integer.parseInt(req.getParameter("num2"));

        int k = i + j;

        RequestDispatcher rd = req.getRequestDispatcher("/square");

        try {
            rd.forward(req, res);
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}

```

- Agora temos que mapear o endereço /square no .xml

```

<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application//EN"

```

```

<web-app>
    <servlet>
        <servlet-name>abc</servlet-name>
        <servlet-class>com.servlet.add.AddServlet</servlet-class>

```

```

</servlet>

<servlet-mapping>
    <servlet-name>abc</servlet-name>
    <url-pattern>/add</url-pattern>
</servlet-mapping>

<servlet>
    <servlet-name>def</servlet-name>
    <servlet-class>com.servlet.add.SquareServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>def</servlet-name>
    <url-pattern>/square</url-pattern>
</servlet-mapping>

</web-app>

```

- Para passar o valor da soma para ser elevado ao quadrado, incluimos o valor no objeto HttpServletRequest

```

package com.servlet.add;

import jakarta.servlet.RequestDispatcher;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

public class AddServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) {
        int i = Integer.parseInt(req.getParameter("num1"));
        int j = Integer.parseInt(req.getParameter("num2"));

        int k = i + j;
        req.setAttribute("sum", k);

        RequestDispatcher rd = req.getRequestDispatcher("/square");

        try {
            rd.forward(req, res);
        }
    }
}

```

```

        } catch (Exception e) {
            System.out.println(e);
        }
    }
}

```

```

package com.servlet.add;

import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

public class SquareServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) {
        int k = (int) req.getAttribute("sum");
        int sq = k * k;
        try {
            res.getWriter().println("Square: " + sq);
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}

```

- RequestDispatcher funciona bem quando os dois endpoints estão dentro do mesmo site, já que para o Client, o processamento do segundo endpoint não será visível

SendRedirect

- Para casos em que é importante que o usuário visualize a mudança de página, não é interessante utilizar o RequestDispatcher
- Para esses casos, é melhor utilizar o sendRedirect, que vai realmente mandar o client para um novo endereço

```

package com.servlet.add;

import jakarta.servlet.RequestDispatcher;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

```

```

public class AddServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) {
        int i = Integer.parseInt(req.getParameter("num1"));
        int j = Integer.parseInt(req.getParameter("num2"));

        int k = i + j;

        res.sendRedirect("square");
    }
}

```

- Dessa forma, removemos o envio de dados, então o Square não pode esperar esse dado

```

package com.servlet.add;

import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

public class SquareServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) {
        try {
            res.getWriter().println("Square called");
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}

```

- Dessa forma, quando trigamos a action do formulário, ao invés de exibir o retorno do square no endereço /add, agora alteramos o endereço para /square
- Como não estamos mais fazendo o “repasse” do request, não podemos enviar nem coletar os dados como Atributos
- A forma a ser feito é, novamente, como parâmetros:

```

package com.servlet.add;

import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;

```

```

import jakarta.servlet.http.HttpServletResponse;

public class SquareServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) {
        int k = Integer.parseInt(req.getParameter("sum"));
        int square = k * k;
        try {
            res.getWriter().println("Square: " + square);
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}

```

- O problema é que, se clicarmos em enviar, ainda dá erro de NullPointerException, pois não há parâmetro algum
- Contudo, com o código dessa forma, se incluirmos o parâmetro diretamente na URL, ele funciona
- Podemos fazer isso dentro da sendRedirect

```

package com.servlet.add;

import java.io.IOException;

import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

public class AddServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws IOException {
        int i = Integer.parseInt(req.getParameter("num1"));
        int j = Integer.parseInt(req.getParameter("num2"));

        int k = i + j;

        res.sendRedirect("square?sum=" + k);
    }
}

```

- Essa é uma de 3 formas de passar dados de um Servlet para o outro utilizando redirecionamento:

1. URL Rewriting (essa forma)
2. Cookies
3. Session Management

HttpSession

- O estado atual é:
 - Ao clicar em submit, enviamos os 2 números para o AddServlet pelo endpoint add
 - O AddServlet soma os valores
 - O AddServlet envia a soma com redirect para o SquareServlet pelo endpoint square
 - O AddServlet envia essa soma como parâmetro na URL
- A forma de passar o dado não é ideal, pois podemos querer repassar esse dado para um outro servlet, passar mais dados, etc
- Precisamos de uma forma de passar dados que atenda a toda uma "**sessão**"
- O Tomcat manterá essa sessão, então se colocarmos o valor na sessão, poderemos acessá-lo em todos os Servlets

```
HttpSession session = req.getSession();
session.setAttribute("sum", k);

res.sendRedirect("square");
```

- A sessão é preservada durante toda a visita ao site, entre várias páginas, ou mesmo outros endereços da mesma aplicação
- Podemos, portanto, buscar o dado no Square Servlet

```
HttpSession session = req.getSession();
int k = (int) session.getAttribute("sum");
```

- Funciona perfeitamente
- Os dados permanecem lá até serem removido, por exemplo, pela própria aplicação:

```
session.removeAttribute("sum");
```

- Isso é comumente utilizado para manter o usuário logado ou, na remoção, para deslogá-lo

Cookies

- Outra alternativa para a passagem de dados que persistem mais do que na URL são os Cookies
- Os Cookies, ao invés da sessão, são persistidos dentro da própria resposta do servidor

```
Cookie cookie = new Cookie("sum", String.valueOf(k));
res.addCookie(cookie);
```

- Na hora de fazer o redirecionamento, contudo, o AddServlet vai passar a resposta para o SquareServlet como sendo o request do SquareServlet
- No request, podemos pegar um array com todos os Cookies e achar o que interessa
- Não podemos, contudo, apenas pegar o primeiro porque já há alguns Cookies padrão
- O primeiro Cookie é algo como:

```
Cookie cookie = req.getCookies()[0];
System.out.println(cookie.getName() + ": " + cookie.getValue());
//JSESSIONID : 73358F8D1143177D038505D5D4031D25
```

- A forma correta, portanto, é buscar pelo nome que demos:

```
package com.servlet.add;

import jakarta.servlet.http.Cookie;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

public class SquareServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) {

        Cookie cookies[] = req.getCookies();
        int k = 0;
        for (Cookie cookie : cookies) {
            if (cookie.getName().equals("sum")) {
```

```

        k = Integer.parseInt(cookie.getValue());
    }
}

int square = k * k;
try {
    res.getWriter().println("Square: " + square);
} catch (Exception e) {
    System.out.println(e);
}
}
}

```

▼ Configuração Externa

ServletContext

- É um objeto útil para determinar alguns valores iniciais para os Servlets
- Podemos incluir esses valores no web.xml
- Eg. incluir meu nome e meu telefone

```

<context-param>
    <param-name>name</param-name>
    <param-value>Douglas</param-value>
</context-param>

<context-param>
    <param-name>phone</param-name>
    <param-value>iPhone</param-value>
</context-param>

```

- No Servlet, vamos dar um passo atrás e utilizar um Servlet básico, apenas para testar o contexto

```

public class BasicServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        PrintWriter out = response.getWriter();
        out.println("Hello World");
    }
}

```

```

<servlet>
    <servlet-name>basic</servlet-name>
    <servlet-class>com.servlet.basic.BasicServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>basic</servlet-name>
    <url-pattern>/basic</url-pattern>
</servlet-mapping>

```

- Podemos buscar o ServletContext com um método do próprio HttpServlet

```

public class BasicServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response) {
        PrintWriter out = response.getWriter();
        out.println("Hello World");

        ServletContext context = getServletContext();
    }
}

```

- Dentro dele, podemos buscar os valores pelas chaves ou nomes que especificamos:

```

ServletContext context = getServletContext();
String name = context.getInitParameter("name");
out.println(name);

String phone = context.getInitParameter("phone");
out.println(phone);

```

- O problema do Context (ou benefício, dependendo do propósito) é que os valores são globais, compartilhados por todas os Servlets daquela aplicação

ServletConfig

- O ServletConfig funciona de forma semelhante, útil como o Context para armazenar credenciais de banco de dados, endereço de API, etc
- Contudo, o Config é específico para cada Servlet

```

<servlet>
    < servlet-name>basic</servlet-name>
    < servlet-class>com.servlet.basic.BasicServlet</servlet-class>

    <init-param>
        <param-name>specific-name</param-name>
        <param-value>Douglas</param-value>
    </init-param>
</servlet>

```

- No código, buscamos por ServletConfig ao invés de ServletContext

```

ServletConfig config = getServletConfig();
String specificValue = config.getInitParameter("specific-name");
out.println(specificValue);

```

▼ Annotations

- Até o momento, temos 3 endpoints: add, square e basic, cada um acionando um Servlet diferente
- Esse mapeamento é feito no web.xml, mas pode ser feito com Annotations
- Podemos apagar ou comentar todo o conteúdo do web.xml
- Mapeamos na classe com a annotation "WebServlet"

```

@WebServlet("/add")
public class AddServlet extends HttpServlet {

    @WebServlet("/square")
    public class SquareServlet extends HttpServlet {

        @WebServlet("/basic")
        public class BasicServlet extends HttpServlet {

```

▼ JSP

- No começo, apagamos o arquivo .jsp e substituimos por um .html
- Isso funciona na raiz, mas em todas as outras telas, resultado da resposta dos Servlets, não temos html algum, apenas texto
- É possível forçar o html dentro do texto:

```

@WebServlet("/basic")
public class BasicServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        PrintWriter out = response.getWriter();

        out.println("<html><body bgcolor='cyan'>");
        out.println("Hello World");
        out.println("</body></html>");
    }
}

```

- E funciona, mas é um completo inferno, e piora gravemente com o crescimento da página de retorno
- Ademais, design é feito por designers, back-end feito pro backends
- JSP permite que, ao invés de escrever html dentro do Java, escrevamos Java dentro do html
- JSP = Java Server Pages
- Vamos criar um arquivo basic.jsp com o mesmo conteúdo, no mesmo local em que agora está o index.html

```

<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>Basic</title>
    </head>
    <body bgcolor="cyan">
        Hello World!
    </body>
</html>

```

- Podemos acessar essa página no endereço /basic.jsp

Conteúdo Dinâmico

- Para testar com conteúdo dinâmico, vamos voltar para o AddServlet (sem o Square)

```

@WebServlet("/add")
public class AddServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws IOException {
        int i = Integer.parseInt(req.getParameter("num1"));
        int j = Integer.parseInt(req.getParameter("num2"));

        int k = i + j;

        PrintWriter out = res.getWriter();
        out.println("Soma: " + k);
    }
}

```

- Vamos criar o add.jsp

```

<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body bgcolor="yellow">

</body>
</html>

```

- Agora, o mesmo código que tínhamos no Servlet, colocaremos aqui, com a única diferença que os objetos request, response e PrintWriter já estão “ímplicitos” no JSP com palavras reservadas
- Além disso, temos que incluir o código dentro de um par de tags específico chamado **Scriptlet** (lembra o Jinja)

```

<html lang="en">
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
</head>
<body bgcolor="yellow">
    <%
        int i = Integer.parseInt(request.getParameter("num1"));

```

```

int j = Integer.parseInt(request.getParameter("num2"));

int k = i + j;

out.println("Soma: " + k);
%>
</body>
</html>

```

- Podemos já remover o Servlet da jogada (dá pra apenas comentar a Annotation)
- Além disso, temos que acessar o add.jsp ao invés de apenas add na action do form, no index.html

```
<form action="add.jsp" method="get">
```

Por que aprender Servlet?

- Com Servlet, precisamos criar uma classe, um objeto, definir um método, buscar os dados, buscar o writer e escrever
- No JSP, temos o request, e o responde e o writer de forma implícita, e precisamos apenas programar dentro do JSP
- Por que Servlets?
- JSP não roda no Tomcat
- Tomcat é um Servlet Container, ele roda Servlets
- Na prática, o código JSP é convertido em Servlets
- São geradas classes com o nome do jsp (eg. add.jsp → addJsp) e o código dentro da tag (**chamada Scriptlet**) é inserido no método service
- Se quisermos criar variáveis de instância ou métodos, não podemos colocar dentro do scriptlet, já que seria criado no service
- Para isso, criamos um trecho chamado “**declaration**” que inclui uma exclamação no final da tag, algo como:

```

<%!
    int instanceVar = 1;
    public String toString(){ //...
%>

```

- Se quisermos importar alguns pacotes, tem uma tag específica também chamada “**directive**”:

```
<%@page import = "java.util.Date, java.util.List" %>
```

- Finalmente, é possível diminuir mais a verbosidade do código com a tag “**expression**”, que coloca todo o conteúdo dentro dela no out.print:

```
<%= k %>
//igual a
out.print(k);
```

- Na prática, portanto, é a mesma coisa, os JSP são apenas uma forma mais fácil de programar
- Contudo, a teoria por trás é a mesma
- Servlets ainda são úteis para quando o processamento não envolve a exibição numa página
- Finalmente, Servlets são código .java, então tem facilidades de autocomplete e intelisense

JSP Directives

- Directives são um tipo de tag do JSP
- Quando queremos usar um pacote, usando <%@ page
- Mas existem outros tipos de tags
- A tag include (<%@ include) permite incluir uma outra página jsp dentro da página jsp em que está a tag
- Existem bibliotecas com extensões dessas tags, que podem ser acessadas como a tag taglib (<%@ taglib)

Objetos Implícitos

- Não precisamos criar alguns dos objetos utilizados comumente dentro dos Servlets quando programamos dentro do JSP
 - request (HttpServletRequest)
 - response (HttpServletResponse)
 - pageContext (PageContext)
 - out (JspWriter ~ PrintWriter)
 - session (HttpSession)

- application (ServletContext)
- config (ServletConfig)
- Page Context não tinha sido citado ainda
- Assim como o ServletContext, contém um conjunto de pares chave-valor que podemos preencher

```
pageContext.setAttribute("name", "Douglas");
```

- A diferença é que, ao invés de poder ser acessada por toda a sessão, o PageContext só pode ser acessado dentro daquela página
- É, contudo, possível, extender algum atributo do PageContext para a sessão

```
pageContext.setAttribute("name", "Douglas",
    PageContext.SESSION_SCOPE);
```

Exception Handling

- É possível utilizar try-catch para capturar exceções

```
<%
try{
    int k = 9/0;
} catch(Exception e){
    out.println("Error " + e.getMessage());
%>
```

- Mas não é uma boa prática passar a mensagem de erro para o usuário
- A solução é criar uma página de erro como, por exemplo, error.jsp

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>Document</title>
    </head>
    <body bgcolor="red">
        Error!
    </body>
</html>
```

```
</body>  
</html>
```

- E podemos fazer referência automática a ela nas outras páginas na tag directive

```
<%@ page errorPage="error.jsp" %>
```

- Não precisamos, nesse caso, fazer o try-catch

```
<%  
    int k = 9/0;  
%>
```

- Além disso, se quisermos, é possível ver, na página de erro, qual é o erro
- Para isso, usamos mais um objeto implícito: exception
- Contudo, o exception só é definido implicitamente em páginas de erro, então precisamos definir na tag page que aquela página (a error.jsp) é de erro

```
<%@ page isErrorPage="true" %>  
  
<html lang="en">  
  <head>  
    <meta charset="UTF-8" />  
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
    <title>Document</title>  
  </head>  
  <body bgcolor="red">  
    Error!  
    <%= exception.getMessage() %>  
  </body>  
</html>
```

▼ JDBC in JSP

- Útil para criar páginas que buscam dados no banco de dados
- Vamos começar criando um novo projeto de webapp com o maven
- Vamos criar a página studentProfile.jsp

```
<!DOCTYPE html>  
<html lang="en">
```

```

<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
</head>
<body>
    1 <br />
    Douglas <br />
    100 <br />
</body>
</html>

```

- Ainda temos dados no banco do exemplo de JDBC

	123 id	A-Z nome	123 nota
1	1	João Silva	8,5
2	2	Maria Oliveira	9,2
3	3	Carlos Santos	7,3
4	4	José Roberto	9,8
5	5	Joana Elisa	7,4
6	6	teste	0
7	7	Rafael Medeiros	6,1

- Temos que buscá-los no banco
- O primeiro passo é importar o pacote jdbc com a tag page

```
<%@ page import="java.sql.*" %>
```

- Em seguida precisamos incluir o driver no projeto

```

<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.33</version>
</dependency>

```

- E incluí-lo com o Class.forName no jsp

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

- Vamos repetir os passos definidos para busca de dados no JDBC

```

<%@ page language="java" import="java.sql.*" errorPage="error.jsp" %>

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <%
      String url = "jdbc:mysql://localhost:3306/mysqlteste";
      String user = "douglas";
      String pass = "123";

      Class.forName("com.mysql.cj.jdbc.Driver");
      Connection con = DriverManager.getConnection(url, user, pass);

      Statement st = con.createStatement();

      ResultSet rs = st.executeQuery("select * from estudantes where id = 1");
      rs.next();
    %>

    Id: <%= rs.getInt(1) %> </br>
    Nome: <%= rs.getString(2) %> </br>
    Nota: <%= rs.getDouble(3) %> </br>
  </body>
</html>

```

▼ MVC com JSP e Servlet

- JSP é uma forma mais rápida e fácil de desenvolver utilizando Servlets
- Contudo, JSP é mais lento do que Servlets
- Além disso, não é interessante colocar a lógica de negócio, persistência, etc, dentro de um arquivo jsp
- Isso representa a separação entre a View (a tela) e a lógica de negócio
- O padrão é que o cliente peça uma tela
 - Ao pedir por uma tela, o servidor vai responder a View, mas a View é apenas o layout (já que tiramos a lógica de negócio de dentro)

- Quem preenche os dados na View é o Controller, que também é quem recebe a solicitação da página e direciona para a página correta
- A forma de passar dados do Controller para o View, como estamos falando de Java, tem que ser um objeto. Esse objeto é o Model
- A diferença entre JSP e HTML para a View é que o JSP permite que a página seja dinâmica em relação aos dados, ao Model que recebe
- O Controller é programado com Servlets, onde todo o processamento é feito
- A View é em JSP, e apenas preenche a tela com os dados do modelo
- O Model é, por padrão, um POJO - Plain Old Java Object, ou seja, apenas dados, sem lógica

N-Tier Architecture

- A boa-prática da Orientação a Objetos é ter classes e objetos que realizem, cada um, apenas uma função
- Portanto, por mais que estejamos já separando a lógica de visualização, os dados e a lógica de negócio, ainda é muita coisa para ficar apenas no Controller
- A Arquitetura N-Tier faz uma separação a mais do que seria o back-end, colocando mais camadas para separar as responsabilidades de processamento
- Ao Controller, fica a responsabilidade apenas de receber as requisições do cliente e responder ao cliente com a view correspondente
- A lógica de negócio de fato, os processamentos, qualquer análise, ficam em um outro objeto chamado **Service**
- Além disso, é comumente necessário acessar o banco de dados, e seria também uma má-prática deixar a lógica de negócio junto com a lógica de acesso ao banco (até porque, podemos eventualmente querer mudar o banco de dados)
- Por isso, é criado mais um objeto, uma camada de acesso ao banco chamada **DAO - Data Access Object** (que, no Spring, é chamado Repository)
- Lembrando que a N-Tier Architecture não é exigência para usar MVC, mas é a melhor prática, especialmente para softwares maiores

▼ JSTL - JSP Standard Tag Library

- O padrão de JSP é escrever Java dentro do HTML
- A ideia do JSTL é converter uma parte do código Java em tags como as do HTML, para facilitar o desenvolvimento especialmente para desenvolvedores

front-end

- Primeiro criaremos um novo projeto com um Servlet que apenas passa um atributo para uma tela usando o RequestDispatcher

```
@WebServlet("/demo")
public class DemoServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response) {
        String name = "Douglas";
        request.setAttribute("name", name);

        try {
            RequestDispatcher rd = request.getRequestDispatcher("display.jsp");
            rd.forward(request, response);
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

- E vamos pegar esse valor na tela:

```
<body>
<%
    String name = request.getAttribute("name").toString();
    out.println(name);
%
</body>
```

- Contudo, o código entre as tags ainda é pouco familiar para quem desenvolve com html
- Podemos utilizar algo chamado Expression Language

Expression Language

- Podemos utilizar a Expression Language dentro do JSP para facilitar algumas funcionalidades

```
 ${label}
```

- Essa sintaxe acessa a o rótulo de algum atributo, ficaria assim

```

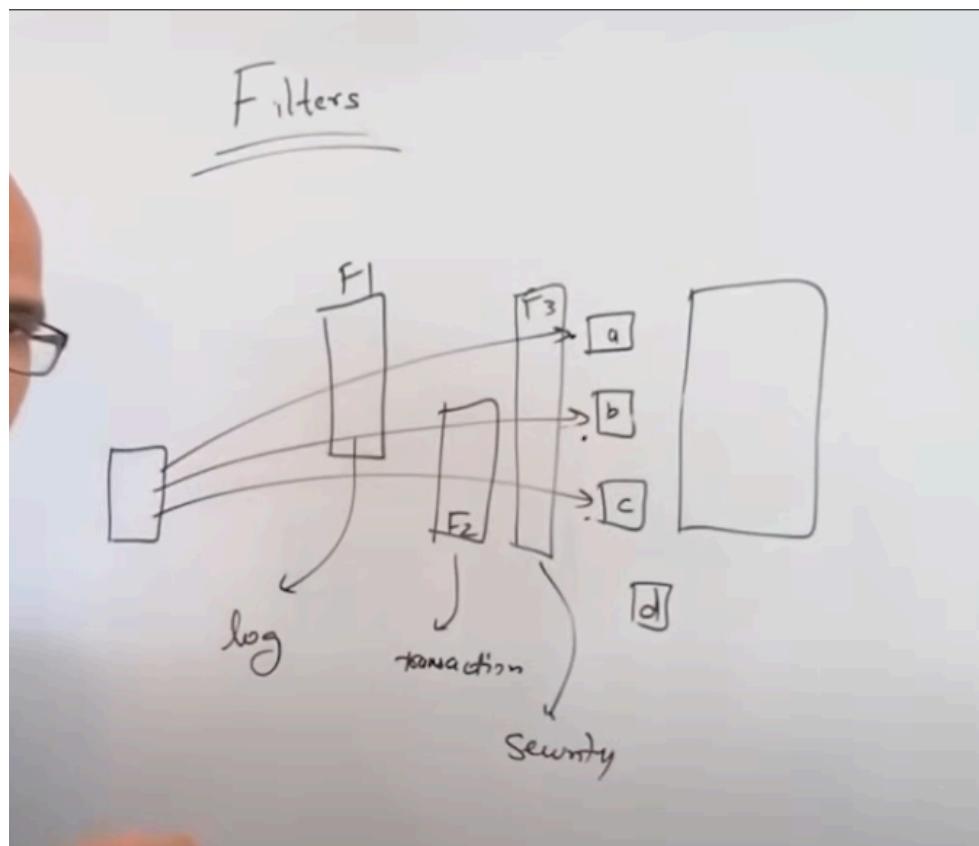
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEnc
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>Document</title>
</head>
<body>
${name}
</body>
</html>

```

- Contudo, não funcionou por nada nesse mundo, então vamos pular esse tema

▼ Filters

- A ideia dos filtros é que eles podem ser aplicados a vários Servlets diferentes
- Podendo ser aplicados dessa forma, eles são úteis para criar funcionalidades que podem ser compartilhadas entre múltiplos Servlets como Log, Segurança, etc.



- Eles ficam entre o Cliente e o Servidor, e interceptam a mensagem antes que ela chegue no Servlet (quase um Middleware do Express)
- Em casos onde há múltiplos filtros antes de um Servlet, precisamos “chain” os filtros, mas eles não saberão disso, apenas o Tomcat saberá
- A configuração dos filtros é feita no web.xml

NÃO TANKEI CONTINUAR, DOUGLAS DO FUTURO POR GENTILEZA VOLTE DE ONDE EU PAREI, IMPOSSÍVEL

▼ Hibernate

▼ Core

- Ao aprender uma nova tecnologia, precisamos responder 4 perguntas: Why, What, Implementation, Advance Concepts
- Falaremos de tudo
- Hibernate é um dos melhores (o melhor) ORM (Object Relational Mapping) de Java
- É feito para acessar o banco de dados junto com o JDBC, mapeando classes e objetos para tabelas e linhas
- A ideia é não ter que escrever as queries SQL
- Hibernate também define as tabelas

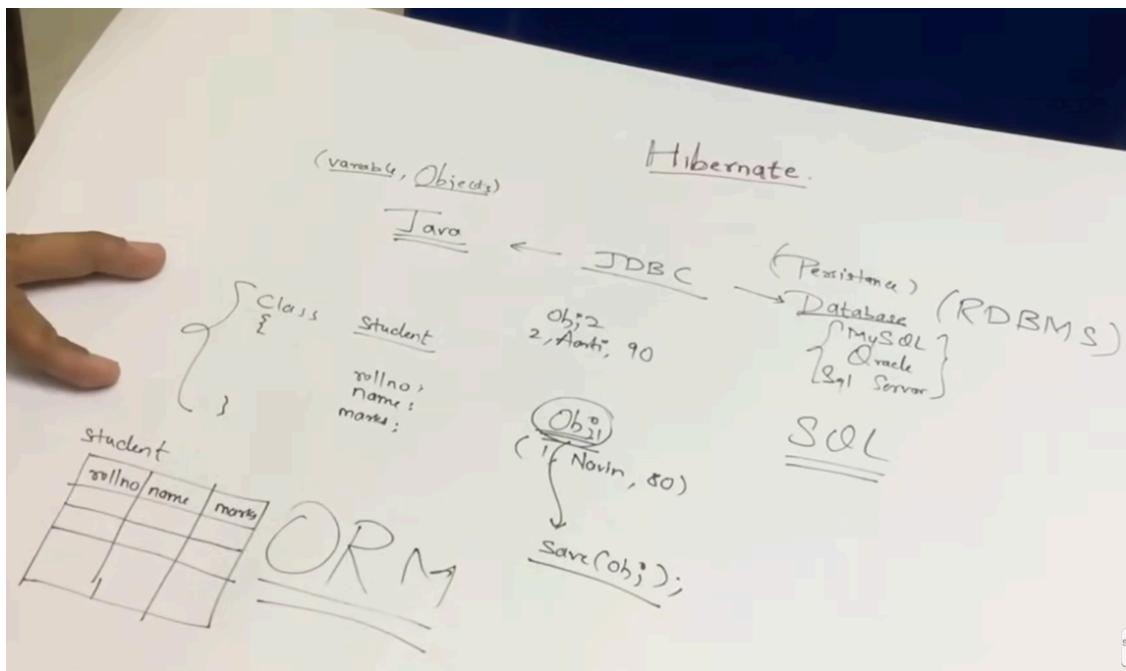
Prerequisites

- O primeiro pré-requisito é saber Java (ok)
- O segundo é SQL (ok)
- O terceiro é JDBC (ok)

Teoria

- No Java, os dados podem ser primitivos (variáveis) ou objetos (instâncias), esses dados são **transientes**, temporários
- Esses dados devem ser **persistidos** num banco de dados
- A conexão entre o Java e o banco de dados é feita com o JDBC
- Por padrão, mesmo com JDBC, precisamos utilizar SQL para acessar e manipular o banco de dados
- O Hibernate vem para remover a necessidade de utilizar o SQL, mapeando os objetos para o banco

- Esse mapeamento é chamado de Object Relational Mapping (ORM)
- Ele é possível pois a estrutura de POO é semelhante à estrutura de classes: cada classe tem uma série de atributos, cada tabela tem uma série de colunas, eles são mapeados 1 a 1



- Além do Hibernate tem o iBatis, o TopLink (ou tinha, em 2000 e bolinha)
- Com eles, usamos apenas um método "save" e salvamos o objeto no banco de dados
- O método save pertence a uma classe Session, que criamos com uma Session Factory
- Podemos configurar o Session Factory com XML, Java Configuration, etc

Prática

- Precisamos de um banco para começar, mas podemos começar com ele vazio
- Vamos também começar com um projeto Maven
- E dentro dele usaremos uma classe Alien

```
package com.example;

public class Alien {
    private int aid;
    private String aname;
```

```
    private String color;  
}
```

- Ele cria os getters e setters na mão, eu coloquei o lombok pq Deus o livre
- Isso é um POJO
- Agora vamos criar um Alien na main

```
package com.example;  
  
public final class App {  
  
    public static void main(String[] args) {  
        Alien douglas = new Alien();  
        douglas.setAid(101);  
        douglas.setAname("Douglas");  
        douglas.setColor("Blue");  
    }  
}
```

- Com JDBC, normalmente seguimos os mesmos 7 passos sempre
- A ideia do Hibernate é usar um método save que já salve isso no banco
- Precisamos incluir o Hibernate no pom e o driver do mysql

```
<!-- https://mvnrepository.com/artifact/org.hibernate.orm/hibernate-core -->  
<dependency>  
    <groupId>org.hibernate.orm</groupId>  
    <artifactId>hibernate-core</artifactId>  
    <version>6.5.2.Final</version>  
</dependency>  
  
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->  
<dependency>  
    <groupId>mysql</groupId>  
    <artifactId>mysql-connector-java</artifactId>  
    <version>8.0.33</version>  
</dependency>
```

- Agora podemos utilizar o Session, que é uma interface, utilizando o SessionFactory, que acessamos através do Configuration

```
Configuration configuration = new Configuration();
SessionFactory sessionFactory = configuration.buildSessionFactory();
```

```
Session session = sessionFactory.openSession();
session.save(douglas);
```

- Contudo, ainda não podemos rodar esse código pois precisamos conectar com o banco
- Além disso, esse save é deprecated
- Precisamos criar um arquivo hibernate.cfg.xml (ele usou o Eclipse, eu usei o GPT)

```
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration File DTD 3.0//EN"
 "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <!-- Configurações de conexão com o banco -->
    <property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
    <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/mydb</property>
    <property name="hibernate.connection.username">douglas</property>
    <property name="hibernate.connection.password">123</property>

    <!-- Dialet do Hibernate para MySQL -->
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

    <!-- Mostrar comandos SQL no console -->
    <property name="hibernate.show_sql">true</property>
    <property name="hibernate.format_sql">true</property>

    <!-- Configurações de tabela -->
    <property name="hibernate.hbm2ddl.auto">update</property>

    <!-- Classe mapeada -->
    <mapping class="com.example.Alien" />
  </session-factory>
</hibernate-configuration>
```

- E acessar o arquivo no código

```
Configuration configuration = new Configuration().configure();
```

- Se não usarmos o nome padrão (usamos) precisamos colocar o nome do arquivo na função
- Agora precisamos informar ao hibernate que Alien é uma entidade com a anotação entity e dizer qual é a PK com a annotation id

```
package com.example;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import lombok.Data;

@Data
@Entity
public class Alien {
    @Id
    private int aid;
    private String aname;
    private String color;
}
```

- E avisar no Configuration que estamos usando a classe anotada

```
Configuration configuration = new Configuration().configure().addAnnotatedClass(Alien.class);
```

- Se rodarmos o código agora, a tabela já é criada no banco!
- Mas, os dados ainda não estão lá, porque precisamos colocar a operação de salvar numa transaction

```
Transaction transaction = session.beginTransaction();
session.save(douglas);
transaction.commit();
```

- No vídeo, ele não conseguiu criar a tabela porque não tinha essa tag no xml (que o GPT colocou para nós)

```
<property name="hibernate.hbm2ddl.auto">update</property>
```

Service Registry

- O método save está deprecated ainda, no vídeo é o buildSessionFactory
- A forma “atual” de fazer isso é com o ServiceRegistry, como segue:

```
Configuration configuration = new Configuration().configure().addAnnotatedClass(Alien.class);
ServiceRegistry registry = new StandardServiceRegistryBuilder() //.
    .applySettings(configuration.getProperties())
    .build();
SessionFactory sessionFactory = configuration.buildSessionFactory(registry);
```

- Não resolveu o save, e não está idêntico ao curso
- Mas tudo continua funcionando

▼ Annotations

- Com isso, já é possível utilizar o Hibernate, criando tabelas e persistindo os dados
- Contudo, ainda é possível fazer mais alguns ajustes e configurações utilizando Annotations
- Podemos mudar o nome da tabela com o parâmetro name na annotation Entity

```
@Entity(name = "alien_table")
public class Alien {
```

- O problema de fazer isso é que estamos também afetando o nome da entidade, não apenas da tabela
- Por isso a melhor forma de fazer é com a annotation **@Table**

```
@Entity
@Table(name = "alien_table")
public class Alien {
```

- Também podemos mudar o nome de uma coluna com a **Column**

```
@Column(name = "alien_name")
private String aname;
```

- Posso também excluir um dos campos de ser persistido, fazendo ser um dado apenas existente na instância, de forma transiente, com a annotation **Transient**

```
@Transient
private String color;
```

▼ Fetching Data

- Já aprendemos a criar e adicionar valores, agora vamos buscar os dados
- Vamos começar removendo as Annotations

```
package com.example;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import lombok.Data;

@Data
@Entity
public class Alien {
    @Id
    private int aid;
    private String aname;
    private String color;
}
```

- O código volta a funcionar normalmente
- Não vamos mais criar um objeto, mas sim buscar um objeto pelo id 101

```
public final class App {
    public static void main(String[] args) {
        Configuration configuration = new Configuration().configure().addAnnotatedClass(Alien.class);
        ServiceRegistry registry = new StandardServiceRegistryBuilder() //
            .applySettings(configuration.getProperties())
            .build();
        SessionFactory sessionFactory = configuration.buildSessionFactory(registry);

        Session session = sessionFactory.openSession();

        Transaction transaction = session.beginTransaction();
        Alien douglas = (Alien) session.get(Alien.class, 101);
        transaction.commit();

        System.out.println(douglas);
        // Alien(aid=101, aname=Douglas, color=Blue)
    }
}
```

▼ Embeddable Object

- Queremos que o nome do Alien tenha 3 partes: first, middle e last
- Criaremos uma classe AlienName

```
@Data  
@AllArgsConstructor  
public class AlienName {  
    private String firstName;  
    private String middleName;  
    private String lastName;  
}
```

- Vamos alterar o nome na classe Alien

```
package com.example;  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.Id;  
import lombok.Data;  
  
@Data  
@Entity  
@AllArgsConstructor  
public class Alien {  
    @Id  
    private int aid;  
    private AlienName aname;  
    private String color;  
}
```

- Vamos criar um novo alien e salvá-lo

```
public final class App {  
    public static void main(String[] args) {  
        Alien alien = new Alien(104, new AlienName("Shiva", "Catarina", "Rocha"),  
  
        Configuration configuration = new Configuration().configure().addAnnotatedClass(Alien.class);  
        ServiceRegistry registry = new StandardServiceRegistryBuilder() //  
            .applySettings(configuration.getProperties())  
            .build();  
        SessionFactory sessionFactory = configuration.buildSessionFactory(regist
```

```

Session session = sessionFactory.openSession();

Transaction transaction = session.beginTransaction();
session.save(alien);
transaction.commit();
}
}

```

- Recebemos um erro por termos alterado a estrutura da tabela
- Vamos alterar o xml para create onde está update, e ele irá recriar tudo

```
<property name="hibernate.hbm2ddl.auto">create</property>
```

- Agora deveria funcionar, mas não funciona pois não reconhece o AlienName
- Precisamos colocar alguma annotation no AlienName para ser reconhecido pelo Hibernate
- Se quisermos outra tabela, podemos criar como Entity e ajustar de acordo
- Mas, para podermos ter todos os dados dentro da mesma tabela, podemos usar Embeddable

```

@Embeddable
public class AlienName {

```

Problema do Construtor

- Esse problema não acontece no curso porque ele sempre usa o construtor default e seta tudo, aqui como usamos o AllArgs deu problema
- Incluindo o NoArgsConstructor nas duas classes, tudo volta a funcionar
- No banco, fica assim:

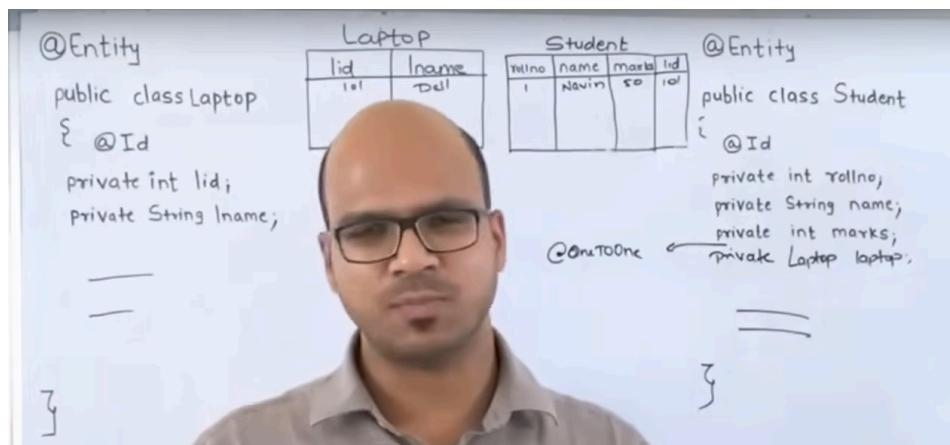
	aid	A-Z color	firstName	lastName	middleName
1	104	Brown	Shiva	Rocha	Catarina

▼ Relacionamentos

Teoria

- Nos bancos de dados, precisamos estabelecer relacionamentos

- Se temos uma classe departamento e outra professor, precisamos conseguir expressar, no banco, quais professores pertencem a cada relacionamento
- Nesse caso, teríamos uma relação de um pra muitos
- Ou, se um professor puder ter mais de um departamento, é uma relação de muitos para muitos
- Precisamos fazer esse mapeamento no Java com Hibernate
- Usaremos um exemplo de Student e Laptop, em que cada Student tem um Laptop



- Focamos na necessidade de colocar a Annotation `@OneToOne`
- Se tivessemos, contudo, uma lista de Laptop em cada Student, em cima do atributo lista, teríamos um `@OneToMany` representando o relacionamento
- Na tabela, contudo, não poderíamos ter uma lista dentro de um campo do Student
- Nesse caso, cria-se uma nova tabela `Student_Laptop` que mapeará os relacionamentos com `rollno` e `lid`

Laptop

lid	lname
101	Dell
102	HP
103	Macbook

Student

rollno	name	marks
1	Navin	50
2	Aarti	56

student_laptop

rollno	lid
1	101
1	102
2	103

- Isso resolve o problema, mas não é a forma mais eficiente
- A forma mais eficiente é alterar a tabela de Laptop para informar de quem é o laptop

`@Entity`

```
public class Laptop
{
    @Id
    private int lid;
    private String lname;
    @ManyToOne
    private Student stud;
```

—

}]

id	lid	lname
1	101	Dell
1	102	HP
2	103	Macbook

- Deixamos o OneToMany no Student, mas também criamos o ManyToOne no Laptop
- A diferença é que, no Student, alteramos a annotation para

```
@OneToMany(mappedBy = "stud")
```

- Sendo stud o nome da variável Student em Laptop
- A outra possibilidade é que múltiplos estudantes possam acessar múltiplos laptops, nesse caso, seria uma relação ManyToMany, e cada classe precisaria ter uma lista da outra
- Novamente, não utilizando o "(mappedBy...)”, o próprio Hibernate forçará a criação de 2 tabelas para estabelecer o relacionamento
- É necessário, contudo, pelo menos 1 tabela de relacionamento (e.g. laptop_student), resultante de manter o “mappedBy”

Prática

- Diferente do Telusko, usarei o mesmo código da prática com JDBC, usando Aliens ao invés de Students, e a segunda classe será Planet

```

public class Planet {
    @Id
    private int pid;
    private String name;
    private String color;
}

public final class App {
    public static void main(String[] args) {
        Alien alien = new Alien(105, new AlienName("Pandora", "Maria", "Rocha"),

        Planet planet = new Planet(101, "SpazioScenarium", "Gray");

        Configuration configuration = new Configuration().configure().addAnnotatedClass(
            .addAnnotatedClass(Planet.class);
        ServiceRegistry registry = new StandardServiceRegistryBuilder() //
            .applySettings(configuration.getProperties())
            .build();
        SessionFactory sessionFactory = configuration.buildSessionFactory(registry);

        Session session = sessionFactory.openSession();

        Transaction transaction = session.beginTransaction();
        session.save(alien);
        session.save(planet);
        transaction.commit();
    }
}

```

- É necessário que a Configuration seja informada das duas classes anotadas (péssimo) e podemos salvar as duas
- Nessa situação, contudo, ainda não há relação
- Se tentarmos apenas colocar o planeta no Alien

```

public class Alien {
    @Id
    private int aid;
    private AlienName aname;
    private String color;
    private Planet planet;
}

```

```
Alien alien = new Alien(
    105,
    new AlienName("Pandora", "Maria", "Rocha"),
    "Black&White",
    planet);
```

- Ao executar recebemos o erro de que “Could not determine recommended JdbcType for Planet”
- É necessário anotar, dentro do Alien, a anotação OneToOne

```
public class Alien {
    @Id
    private int aid;
    private AlienName aname;
    private String color;
    @OneToOne
    private Planet planet;
}
```

	aid	planet_pid	color	firstName	lastName	middleName
1	105	101	Black&White	Pandora	Rocha	Maria

- PS: o hibernate está no modo create, por isso ele é capaz de fazer as alterações com facilidade, mas perde os dados
- Mas e se eu quiser agora que um Alien possa ter múltiplos planetas? Necessário usar uma lista na classe e alterar a anotação

```
public class Alien {
    @Id
    private int aid;
    private AlienName aname;
    private String color;
    @OneToMany
    private List<Planet> planets;
}
```

- Nessa situação, contudo, é criada uma tabela Alien_Planet com os relacionamentos

	Alien_aid	planets_pid
1	105	101
2	105	102

- Mas, como é apenas 1 para muitos, não haveria necessidade de criar uma tabela inteira para isso, o próprio planeta poderia saber quais Aliens os possuem

```
public class Planet {
    @Id
    private int pid;
    private String name;
    private String color;
    @ManyToOne
    private Alien alien;
}
```

```
public static void main(String[] args) {
    Planet planet1 = new Planet(101, "SpazioScenarium", "Gray", null);
    Planet planet2 = new Planet(102, "MansaDoLenario", "Redish", null);

    Alien alien = new Alien(
        105,
        new AlienName("Pandora", "Maria", "Rocha"),
        "Black&White",
        List.of(planet1, planet2));

    planet1.setAlien(alien);
    planet2.setAlien(alien);
}
```

- Agora o planeta sabe qual o Alien dele
- Mas, a tabela de relacionamento ainda existe, mesmo não precisando mais existir
- Para isso usamos o mappedBy na tabela que tem a lista

```
public class Alien {
    @Id
    private int aid;
    private AlienName aname;
```

```

    private String color;
    @OneToMany(mappedBy = "planet")
    private List<Planet> planets;
}

```

- Por algum motivo, nesse estágio, foi necessário dar drop no banco
- Agora, sim, não temos mais a tabela intermediária

	alien_aid	pid	color	name
1	105	101	Gray	SpazioScenarium
2	105	102	Redish	MansaoDoLenario

- E se, agora, cada planeta tiver muitos aliens?

```

public class Alien {
    @Id
    private int aid;
    private AlienName aname;
    private String color;
    @ManyToMany
    private List<Planet> planets;
}

public class Planet {
    @Id
    private int pid;
    private String name;
    private String color;
    @ManyToMany
    private List<Alien> aliens;
}

public static void main(String[] args) {
    Planet planet1 = new Planet(101, "SpazioScenarium", "Gray", null);
    Planet planet2 = new Planet(102, "MansaoDoLenario", "Redish", null);

    Alien alien1 = new Alien(
        105,
        new AlienName("Pandora", "Maria", "Rocha"),
        "Black&White",
        List.of(planet1, planet2));
    Alien alien2 = new Alien(

```

```

108,
new AlienName("Shiva", "Catarina", "Rocha"),
"Brown",
List.of(planet1, planet2));

planet1.setAliens(List.of(alien1, alien2));
planet2.setAliens(List.of(alien1, alien2));

Configuration configuration = new Configuration().configure().addAnnotatedClass(Planet.class);
ServiceRegistry registry = new StandardServiceRegistryBuilder() //.
    .applySettings(configuration.getProperties())
    .build();
SessionFactory sessionFactory = configuration.buildSessionFactory(registry);

Session session = sessionFactory.openSession();

Transaction transaction = session.beginTransaction();
session.save(planet1);
session.save(planet2);
session.save(alien1);
session.save(alien2);
transaction.commit();
}

```

- O código funciona! Mas cria tabelas demais:

The screenshot shows two windows from MySQL Workbench. The top window is titled 'A-Z Tables_in_mysqlteste' and lists five tables: Alien, Alien_Planet, Planet, and Planet_Alien. The 'Alien' table is currently selected. The bottom window is titled 'Alien_Planet 1' and displays the data for the 'Alien_Planet' table. The table has three columns: Grade, Alien_aid, and planets_pid. The data is as follows:

Grade	Alien_aid	planets_pid
1	105	101
2	105	102
3	108	101
4	108	102

Grade	123 Planet_pid	123 aliens_aid
Texto		
1	101	105
2	101	108
3	102	105
4	102	108

- A solução é escolher um dos dois e atribuir o mapeamento a ele colocando o mappedBy no outro

```
public class Alien {
    @Id
    private int aid;
    private AlienName aname;
    private String color;
    @ManyToMany(mappedBy = "aliens")
    private List<Planet> planets;
}
```

- E agora temos apenas 1 tabela fazendo o relacionamento

Grade	123 aliens_aid	123 planets_pid
Texto		
1	105	101
2	108	101
3	105	102
4	108	102

INCLUSÃO POSTERIOR: descobri isso no desenvolvimento de uma feature no MLFF

- É possível determinar os nomes das tabelas criadas no banco para mapear relacionamentos ManyToMany:

```
@ManyToMany
@JoinTable(
    name = "usuario_grupo",
    joinColumns = @JoinColumn(name = "id_usuario"), //
```

```

        inverseJoinColumns = @JoinColumn(name = "id_grupo")
    )
private List<GrupoUsuarioEntity> grupos;

```

- Dessa forma, a tabela de mapeamento criada será usuario_grupo e as colunas de relacionamento serão id_usuario e id_grupo, mesmo que o padrão de nomenclatura automática do Hibernate seja diferente

▼ FetchType

- O FetchType pode ser Eager (apressado) ou Lazy (preguiçoso)
- Usaremos o mesmo exemplo de Aliens e Planets

Planet 1

SELECT * from Planet p				Insira uma expressão SQL para filtrar os resultados (use Ctrl+Espaço)
Grade	pid	color	name	
1	101	Gray	SpazioScenarium	
2	102	Redish	MansaoDoLenario	

Alien 1

SELECT * from Alien a						Insira uma expressão SQL para filtrar os resultados (use Ctrl+Espaço)
Grade	aid	color	firstName	lastName	middleName	
1	105	Black&White	Pandora	Rocha	Maria	
2	108	Brown	Shiva	Rocha	Catarina	

Planet_Alien 1

SELECT * from Planet_Alien			Insira uma expressão SQL para filtrar os resultados (use Ctrl+Espaço)
Grade	aliens_aid	planets_pid	
1	105	101	
2	108	101	
3	105	102	
4	108	102	

- Vamos tentar dar o fetch em Pandora
- Mas antes, vamos mudar a configuração do Hibernate para update, assim não precisamos criar todos os dados sempre

```
<property name="hibernate.hbm2ddl.auto">update</property>
```

```

public static void main(String[] args) {
    Configuration configuration = new Configuration().configure().addAnnotatedClass(Planet.class);
    ServiceRegistry registry = new StandardServiceRegistryBuilder() //
        .applySettings(configuration.getProperties())
        .build();
    SessionFactory sessionFactory = configuration.buildSessionFactory(registry);

    Session session = sessionFactory.openSession();
    Transaction transaction = session.beginTransaction();

    Alien pandora = session.get(Alien.class, "105");

    transaction.commit();
}

```

- O código funciona, e mesmo sem imprimir nada ou debugar, podemos ver a query feita pelo log:

```

Hibernate:
select
    a1_0.aid,
    a1_0.firstName,
    a1_0.lastName,
    a1_0.middleName,
    a1_0.color
from
    Alien a1_0
where
    a1_0.aid=?

```

- Ele não buscou os planetas!
- Se tentarmos acessar os planetas, contudo, depois de imprimir o nome do Alien, entendemos o comportamento pelo log

```

Alien pandora = session.get(Alien.class, "105");
System.out.println(pandora.getAname());

List<Planet> planets = pandora.getPlanets();
for (Planet planet : planets) {
    System.out.println(planet.getName());
}

```

```
}
```

Hibernate:

```
select
    a1_0.aid,
    a1_0.firstName,
    a1_0.lastName,
    a1_0.middleName,
    a1_0.color
from
    Alien a1_0
where
    a1_0.aid=?
```

AlienName(firstName=Pandora, middleName=Maria, lastName=Rocha)

Hibernate:

```
select
    p1_0.aliens_aid,
    p1_1.pid,
    p1_1.color,
    p1_1.name
from
    Planet_Alien p1_0
join
    Planet p1_1
    on p1_1.pid=p1_0.planets_pid
where
    p1_0.aliens_aid=?
```

SpazioScenarium

MansaoDoLenario

- Claramente, o código busca o Alien sem os planetas e, apenas quando é necessário, vai atrás de buscar os planetas no banco
- Esse comportamento é um **FetchType.LAZY**, porque só faz o que tem que fazer quando é necessário
- Há benefícios nesse modo, e por isso é o padrão. Contudo, é possível alterar a forma de FetchType para EAGER

```
public class Alien {
    @Id
    private int aid;
    private AlienName aname;
```

```

private String color;
@ManyToMany(mappedBy = "aliens", fetch = FetchType.EAGER)
private List<Planet> planets;
}

```

- Agora ele deve buscar tudo de uma vez

Hibernate:

```

select
    a1_0.aid,
    a1_0.firstName,
    a1_0.lastName,
    a1_0.middleName,
    a1_0.color,
    p1_0.aliens_aid,
    p1_1.pid,
    p1_1.color,
    p1_1.name
from
    Alien a1_0
left join
    Planet_Alien p1_0
        on a1_0.aid=p1_0.aliens_aid
left join
    Planet p1_1
        on p1_1.pid=p1_0.planets_pid
where
    a1_0.aid=?
AlienName(firstName=Pandora, middleName=Maria, lastName=Rocha)
SpazioScenarium
MansaDoLenario

```

- A ideia do Lazy é que, às vezes, você pode não precisar daquele dado, e buscá-lo desnecessariamente é muito custoso

▼ Hibernate Caching

Teoria

- Há algumas vantagens em usar Hibernate ao invés de JDBC
- Uma delas é o ORM, mas não é a única
- O Hibernate consegue fazer Caching dos dados, para não buscá-los multiplas vezes

- Para buscar algum dado, utilizamos a Transaction que é parte da Session
- Na prática, o Cliente pede os dados pro Servidor que pede para o Banco
- Podem ser buscados muitos dados e com frequência, então os dados buscados no banco (dentro de uma mesma Session) são armazenados em Cache no Servidor
- Nessa mesma Session, os dados são armazenados como First Level Cache
- Contudo, se um outro usuário usar outra Session para buscar os mesmos dados, o Hibernate não usaria o First Level Cache
- Seria possível, nesse caso, utilizar Second Level Cache, para facilitar o acesso, mas o Hibernate não faz isso
- Algumas bibliotecas permitem isso: ehcache, os e swam, mas ehcache é a mais comum (ou era na época do curso)

ehcache

- Para utilizar o ehcache para Second Level Cache, há alguns passos
- O primeiro, obviamente, é inclui-lo no pom.xml. (ele também fala da necessidade de instalar o .jar)
- Depois disso, temos que ajustar a configuração no hibernate.cfg.xml para utilizar o ehcache como Second Level Cache
- Em seguida, atualizamos as Entitys com anotações como Cachable (para autorizar o Caching) e Cache (para determinar a estratégia de Cache)

Prática

Hibernate - First Level Cache

- Level1 Cache funciona na mesma Session, em Transactions diferentes
- Usaremos ainda o exemplo de Aliens e Planets usado acima, alterando apenas para excluir os planetas no tostring do alien

```
public class Alien {
    @Id
    private int aid;
    private AlienName aname;
    private String color;
    @ToString.Exclude
    @ManyToMany(mappedBy = "aliens", fetch = FetchType.EAGER)
```

```
    private List<Planet> planets;  
}
```

- Vamos buscar o Alien Pandora 2 vezes seguidas:

```
public static void main(String[] args) {  
    Configuration configuration = new Configuration().configure().addAnnotatedClass(Planet.class);  
    ServiceRegistry registry = new StandardServiceRegistryBuilder() //  
        .applySettings(configuration.getProperties())  
        .build();  
    SessionFactory sessionFactory = configuration.buildSessionFactory(registry);  
  
    Session session = sessionFactory.openSession();  
    Transaction transaction = session.beginTransaction();  
  
    Alien pandora = session.get(Alien.class, "105");  
    System.out.println(pandora.toString());  
  
    pandora = session.get(Alien.class, "105");  
    System.out.println(pandora.toString());  
  
    transaction.commit();  
}
```

Hibernate:

```
select  
    a1_0.aid,  
    a1_0.firstName,  
    a1_0.lastName,  
    a1_0.middleName,  
    a1_0.color,  
    p1_0.aliens_aid,  
    p1_1.pid,  
    p1_1.color,  
    p1_1.name  
from  
    Alien a1_0  
left join  
    Planet_Alien p1_0  
        on a1_0.aid=p1_0.aliens_aid  
left join
```

```

    Planet p1_1
        on p1_1.pid=p1_0.planets_pid
    where
        a1_0.aid=?
Alien(aid=105, fname=AlienName(firstName=Pandora, middleName=Maria, last
Alien(aid=105, fname=AlienName(firstName=Pandora, middleName=Maria, last

```

- Vemos que fizemos a mesma busca 2 vezes e, graças ao First Level Cache, ele não fez a query duas vezes, pois os dados estavam armazenados
- Contudo, se fizermos isso em outra Session, não funciona

```

Session session1 = sessionFactory.openSession();
Transaction transaction1 = session1.beginTransaction();

```

```

Alien pandora = session1.get(Alien.class, "105");
System.out.println(pandora.toString());

```

```

transaction1.commit();
session1.close();

```

```

Session session2 = sessionFactory.openSession();
Transaction transaction2 = session2.beginTransaction();

```

```

pandora = session2.get(Alien.class, "105");
System.out.println(pandora.toString());

```

```

transaction2.commit();
session2.close();

```

Hibernate:

```

select
    a1_0.aid,
    a1_0.firstName,
    a1_0.lastName,
    a1_0.middleName,
    a1_0.color,
    p1_0.aliens_aid,
    p1_1.pid,
    p1_1.color,
    p1_1.name
from
    Alien a1_0

```

```

left join
    Planet_Alien p1_0
        on a1_0.aid=p1_0.aliens_aid
left join
    Planet p1_1
        on p1_1.pid=p1_0.planets_pid
where
    a1_0.aid=?
Alien(aid=105, fname=AlienName(firstName=Pandora, middleName=Maria, last
Hibernate:
    select
        a1_0.aid,
        a1_0.firstName,
        a1_0.lastName,
        a1_0.middleName,
        a1_0.color,
        p1_0.aliens_aid,
        p1_1.pid,
        p1_1.color,
        p1_1.name
    from
        Alien a1_0
    left join
        Planet_Alien p1_0
            on a1_0.aid=p1_0.aliens_aid
    left join
        Planet p1_1
            on p1_1.pid=p1_0.planets_pid
    where
        a1_0.aid=?
Alien(aid=105, fname=AlienName(firstName=Pandora, middleName=Maria, last

```

- Ele faz duas Queries!
- A solução é Second Level Cache, utilizaremos ehcache para isso

ehcache - Second Level Cache

Não rolou...

▼ Hibernate Query Language (HQL)

- Por que? Se temos o Orm
- Quando usamos o .save, estamos salvando toda a entidade no banco

- Mas, quando usamos get, temos que passar a classe e o id
- E se eu quiser baseado numa condição? Se quiser um join?
- Para isso podemos utilizar HQL, que é próximo de SQL
- Parece com a linguagem do JPA:

SQL:

```
select column from table
select * from table
```

HQL:

```
select property from Class
from Class
```

- Podemos fazer todas as operações, mas utilizando a classe e a propriedade ao invés da tabela e a coluna
- Outra diferença é que, com JDBC, quando fazemos uma query, recebemos um ResultSet que funciona como um iterator e temos que fazer o casting
- Com HQL, quando fazemos a query, recebemos uma lista da própria classe

```
ResultSet rs= st.executeQuery(sql);
while(rs.next())
{
}
```



```
List<Student>st= query.list();
for(Student s: st)
{
    System.out.println (s);
}
```

- Podemos utilizar nativeQuery, se quisermos

Prática

- Para a prática, usaremos uma versão mais simples do Alien

```
public class Alien {
    @Id
    private int aid;
    private String aname;
    private String color;
}
```

- Vamos criar vários de uma vez

```

public static void main(String[] args) {
    Configuration configuration = new Configuration().configure().addAnnotatedClass(Planet.class);
    ServiceRegistry registry = new StandardServiceRegistryBuilder() //
        .applySettings(configuration.getProperties())
        .build();
    SessionFactory sessionFactory = configuration.buildSessionFactory(registry);

    Session session = sessionFactory.openSession();
    Transaction transaction = session.beginTransaction();

    for (int i = 1; i <= 50; i++) {
        Alien a = new Alien(i, "Alien" + i, "Blue");
        session.save(a);
    }

    transaction.commit();
}

```

id	aid	aname	color
1	1	Alien1	Blue
2	2	Alien2	Blue
3	3	Alien3	Blue
4	4	Alien4	Blue
5	5	Alien5	Blue
6	6	Alien6	Blue
7	7	Alien7	Blue
8	8	Alien8	Blue
9	9	Alien9	Blue
10	10	Alien10	Blue
11	11	Alien11	Blue
12	12	Alien12	Blue
13	13	Alien13	Blue

- Agora vamos imprimir todos os valores

```

Query q = session.createQuery("from Alien");
List<Alien> aliens = q.list();
for (Alien alien : aliens) {
    System.out.println(alien);
}

```

Alien(aid=1, aname=Alien1, color=Blue)

```
Alien(aid=2, fname=Alien2, color=Blue)
Alien(aid=3, fname=Alien3, color=Blue)
Alien(aid=4, fname=Alien4, color=Blue)
Alien(aid=5, fname=Alien5, color=Blue)
Alien(aid=6, fname=Alien6, color=Blue)
Alien(aid=7, fname=Alien7, color=Blue)
Alien(aid=8, fname=Alien8, color=Blue)
Alien(aid=9, fname=Alien9, color=Blue)
Alien(aid=10, fname=Alien10, color=Blue)

...
```

- Muito mais fácil do que com Result Set
- Agora queremos apenas com id maior do que 45

```
Query q = session.createQuery("from Alien where aid > 45");
List<Alien> aliens = q.list();
for (Alien alien : aliens) {
    System.out.println(alien);
}
```

```
Alien(aid=46, fname=Alien46, color=Blue)
Alien(aid=47, fname=Alien47, color=Blue)
Alien(aid=48, fname=Alien48, color=Blue)
Alien(aid=49, fname=Alien49, color=Blue)
Alien(aid=50, fname=Alien50, color=Blue)
```

- É possível buscar apenas um dado também, mas nesse caso precisamos fazer o cast

```
Query q = session.createQuery("from Alien where aid = 45");
Alien alien = (Alien) q.uniqueResult();
System.out.println(alien);
```

```
Alien(aid=45, fname=Alien45, color=Blue)
```

- Contudo, não podemos buscar apenas uma parte dos dados assim

```
Query q = session.createQuery("select aid, color from Alien where aid = 45");
Alien alien = (Alien) q.uniqueResult();
System.out.println(alien);
```

```
Exception in thread "main" java.lang.ClassCastException: class [Ljava.lang.Object; at com.example.App.main(App.java:26)
```

- A solução é utilizar um Array de objetos

```
Query q = session.createQuery("select aid, color from Alien where aid = 45");  
Object[] alien = (Object[]) q.uniqueResult();
```

```
for (Object o : alien) {  
    System.out.println(o);  
}
```

```
45  
Blue
```

- Podemos usar uma lista de objetos para trazer vários desses registros

```
Query q = session.createQuery("select aid, color from Alien");  
List<Object[]> aliens = q.list();
```

```
for (Object[] alien : aliens) {  
    System.out.println(alien[0] + " | " + alien[1]);  
}
```

```
1 | Blue  
2 | Blue  
3 | Blue  
4 | Blue  
5 | Blue  
6 | Blue  
7 | Blue  
...
```

- É possível utilizar **Aliases** no HQL também

```
Query q = session.createQuery("select a.aid, a.color from Alien a");  
List<Object[]> aliens = q.list();
```

- É possível utilizar **Agregate Functions** também:

```
Query q = session.createQuery("select sum(aid) from Alien a");  
Long sum = (Long) q.uniqueResult();
```

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

1275

- Ainda é possível utilizar variáveis do código utilizando algo semelhante ao **Prepared Statement**

```
int desiredId = 33;
```

```
Query q = session.createQuery("from Alien where aid = :desiredId");
q.setParameter("desiredId", desiredId);
Alien alien = (Alien) q.uniqueResult();
```

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

```
Alien(aid=33, aname=Alien33, color=Blue)
```

- HQL é muito parecido com SQL. Contudo, se for necessário usar SQL ao invés de HQL, é possível utilizando **createSQLQuery** (ou ao menos era, não consegui reproduzir)

```
SQLQuery query = session.createSQLQuery("select * from student where marks>60");
query.addEntity(Student.class);
List<Student> students = query.list();

for(Student o : students)
{
    System.out.println(o);
}

session.getTransaction().commit();
```

▼ Persistance Life Cycle

Teoria

- Tudo no Java tem um início no new e um fim quando é coletado pelo Garbage Collector, multiplas vezes quando o passamos para Null
- Tudo que acontece entre o começo e o fim é chamado de Transient State
- Se os dados do Transient State forem perdidos, não podem ser recuperados
- Para poder armazenar esse estado transiente, salvamos ou persistimos no banco de dados
- Uma característica do Hibernate é que, mesmo após o save, dentro de uma mesma sessão ou transaction, alterações feitas ao objeto ainda assim são persistidas

```
Alien a = new Alien(52, "Zé", "Branco");
session.save(a);
a.setColor("Preto");

transaction.commit();
```

Hibernate:

```
insert
into
    Alien
    (aname, color, aid)
values
    (?, ?, ?)
```

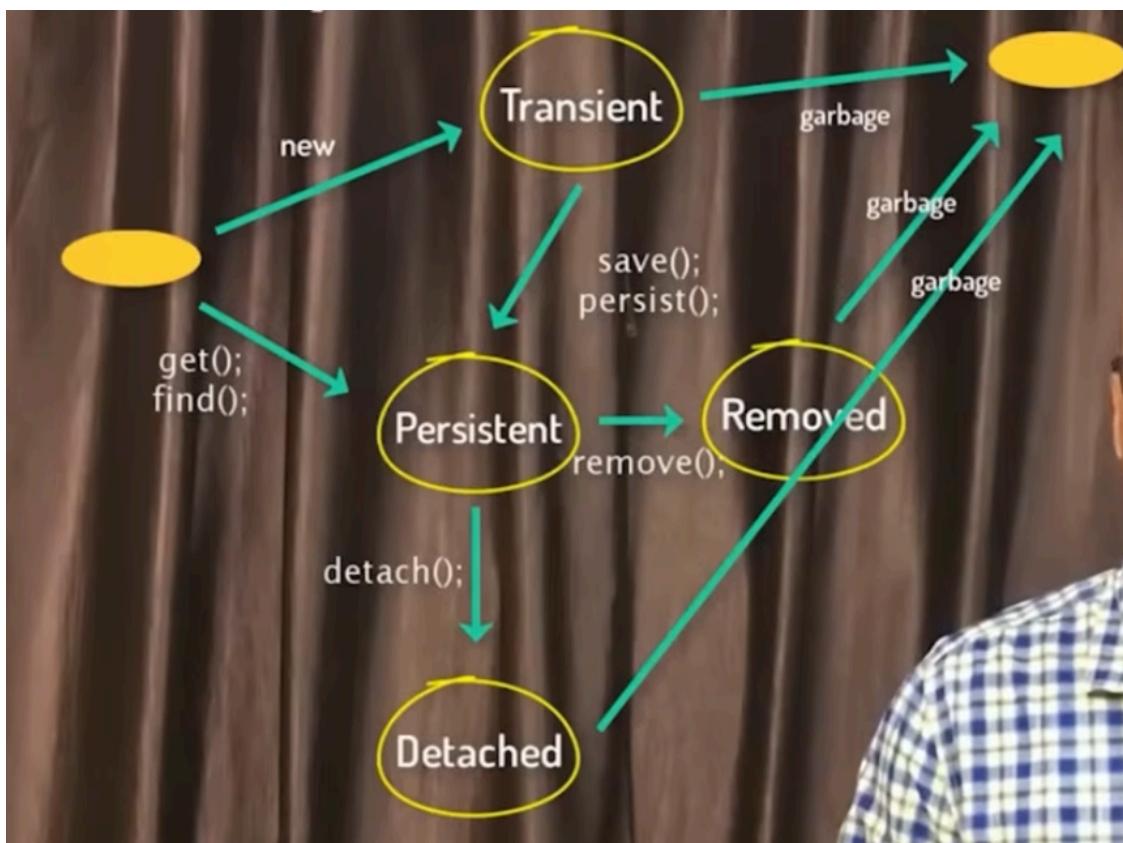
Hibernate:

```
update
    Alien
set
    aname=?,
    color=?
where
    aid=?
```

id	aid	aname	color
1	52	Zé	Preto

- Nesse estado, após ter sido salvo mas dentro da mesma sessão, o dado é chamado de **Persistent State**
- Isso deixa de ser o estado quando ele passa para o **Detached State**

- A transição para esse estado acontece ao final da sessão ou se utilizarmos o comando **detach()**
- Além desses, temos o **Removed State**, que ocorre quando utilizamos o comando **remove()**. O dado deixa de existir no banco mas continua existindo no Java
- Quando buscamos um dado do banco, ele vai diretamente para o estado **Persistent**



Prática

- Continuaremos utilizando os Aliens
- Quando criamos um objeto, ele está em Transient State, o que significa que ao sair do escopo ou encerrar o software, esses dados serão perdidos

```
// Transient State
Alien a = new Alien(53, "Jão", "Amarelo");
```

- A partir do momento em que salvo o dado, ele passa para o Persistent State

```
// Persistent State
```

```
session.save(a);
```

- Nesse estado, se eu fizer uma alteração antes do commit, a alteração é refletida no banco

```
// Persistent State  
session.save(a);  
  
a.setColor("Preto");  
  
transaction.commit();
```

- Pode parecer que, na verdade, o objeto é persistido apenas uma vez, mas não é o caso. Se olharmos para o log, vemos que são feitas duas queries:

Hibernate:

```
insert  
into  
Alien  
(aname, color, aid)  
values  
(?, ?, ?)
```

Hibernate:

```
update  
Alien  
set  
aname=?,  
color=?  
where  
aid=?
```

- Se quisermos, por algum motivo, que alteração exista somente no software e não seja persistida no banco, devemos fazê-lo após o commit

```
// Transient State  
Alien a = new Alien(56, "Jão", "Amarelo");  
  
// Persistent State  
session.save(a);  
  
transaction.commit();
```

```
// Detached State  
a.setColor("Preto")
```

	123 aid	A-Z aname	A-Z color
1	56	Jão	Amarelo

- Finalmente, podemos remover o dado do banco com remove, mas sem perdê-lo no software

```
// Transient State  
Alien a = new Alien(57, "Jão", "Amarelo");  
  
// Persistent State  
session.save(a);  
  
// Removed State  
session.remove(a);  
  
transaction.commit();  
  
System.out.println(a);
```

```
Alien(aid=57, aname=Jão, color=Amarelo)
```

- O dado deixa de existir na tabela

▼ Get X Load

- `get()` e `load()` são dois métodos de busca de dados no banco
- Normalmente os dois funcionam da mesma forma:

```
Alien a = session.get(Alien.class, 56);
System.out.println(a);

Alien(aid=56, aname=Jão, color=Amarelo)
```

```
Alien a = session.load(Alien.class, 56);
System.out.println(a);

Alien(aid=56, aname=Jão, color=Amarelo)
```

- A diferença é que, na prática, `load` apenas busca o dado quando é utilizado, como se fosse um `FetchType Lazy`

```
Alien a = session.load(Alien.class, 56);
System.out.println("Load");
System.out.println(a);
```

```
Load
Hibernate:
    select
        a1_0.aid,
        a1_0.aname,
        a1_0.color
    from
        Alien a1_0
    where
        a1_0.aid=?
Alien(aid=56, aname=Jão, color=Amarelo)
```

- Ele retorna, em primeira instância, uma espécie de **Proxy** do objeto
- Uma utilidade desse proxy indicada pelo Telusko é quando queremos criar, por exemplo, um planeta onde colocaríamos esse Alien. Não precisamos de nenhum dado dele apenas para criar o planeta

- Por oferecer um Proxy, o load também é útil para identificar objetos não existentes, porque o get retorna null

```
Alien a = session.get(Alien.class, 57);
System.out.println(a);
System.out.println(a.getColor());

null
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "com.
at com.example.App.main(App.java:27)
```

```
Alien a = session.load(Alien.class, 57);
System.out.println(a);
System.out.println(a.getColor());

Exception in thread "main" org.hibernate.ObjectNotFoundException: No row wit
at org.hibernate.boot.internal.StandardEntityNotFoundDelegate.handleEntit
at org.hibernate.proxy.AbstractLazyInitializer.checkTargetState(AbstractLa
at org.hibernate.proxy.AbstractLazyInitializer.initialize(AbstractLazyInitializ
at org.hibernate.proxy.AbstractLazyInitializer.getImplementation(AbstractLa
at org.hibernate.proxy.pojo.bytebuddy.ByteBuddyInterceptor.intercept(Byte
at org.hibernate.proxy.ProxyConfiguration$InterceptorDispatcher.intercept(
at com.example.Alien$HibernateProxy$uzTjwnCA.toString(Unknown Source)
at java.base/java.lang.String.valueOf(String.java:4220)
at java.base/java.io.PrintStream.println(PrintStream.java:1047)
at com.example.App.main(App.java:26)
```

- O interessante é que o load estoura uma exception específica que pode ser capturada e identificada mais facilmente
- Normalmente, ainda assim, o Get é mais utilizado

▼ JPA - Java Persistence API

Teoria

- Tudo é sobre dados
- Dados são processados por software
- Software roda na RAM, então quando o software encerrar, os dados nele são perdidos
- A solução é persistir esses dados num banco de dados (ou outra forma)
- No Java, a forma básica de acessar e interagir com bancos de dados é utilizando o JDBC
- O problema do JDBC é a falta de transparência
- ORM é um conceito que vem para aumentar a transparência, relacionando objetos e tabelas do banco de forma “automatizada”
- O Hibernate nos permite utilizar o ORM no Java
- No JDBC, para salvar um dado, precisamos especificá-lo de forma explícita, o ORM permite salvar o objeto inteiro no banco
- O Hibernate aplica ORM de uma forma, outra ferramenta (iBatis ou TopLink) aplica de outra forma, o que deixa o software acoplado
- A ideia do JPA é criar mais uma camada de encapsulamento, mantendo uma mesma especificação para todas as ferramentas de ORM
- É mais fácil, se usarmos JPA, alternar entre ferramentas
- JPA não isenta a necessidade do Hibernate ou outra ferramenta parecida, pois JPA é apenas a especificação, mas precisa da ferramenta para comunicar com o banco
- JPA não é a mesma coisa de SpringBootJPA

Prática

- Para a prática dessa etapa, criaremos um novo projeto Maven
- Utilizaremos, contudo, o mesmo banco de antes, com Aliens

	123 aid	A-Z aname	A-Z color
1	1	Alien1	Blue
2	2	Alien2	Blue
3	3	Alien3	Blue
4	4	Alien4	Blue
5	5	Alien5	Blue
6	6	Alien6	Blue

- Para representar a tabela, criaremos um objeto Alien.java

```
@Data  
public class Alien {  
    private int aid;  
    private String aname;  
    private String color;  
}
```

- Nossa primeira missão será buscar algum Alien do banco
- No JPA, utilizaremos o método `find()`

```
public static void main(String[] args) {  
    Alien a = find(Alien.class, 4);  
}
```

- O método, contudo, não funciona sem prévia configuração
- Algo necessário é criar um `EntityManager`, uma classe responsável por gerenciar todas as entidades que usaremos com o JPA
- Mas, mesmo para importar o Manager, precisamos incluir a dependência do Hibernate e o MySQL
- Agora já é possível instanciar o `EntityManager`, mas por ser uma interface criaremos com uma Factory. O código será mais ou menos assim:

```
public static void main(String[] args) {  
    EntityManagerFactory emf = Persistence.createEntityManagerFactory("");  
    EntityManager em = emf.createEntityManager();  
  
    Alien a = em.find(Alien.class, 4);  
  
    System.out.println(a);  
}
```

- Contudo, esse código ainda não tem nenhuma forma de saber qual o banco que queremos acessar
- Criaremos um arquivo de configuração em `src/main/resources/META-INF` com o nome de `persistence.xml`

```
<?xml version="1.0" encoding="UTF-8"?>  
  
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_1_0.xsd"
<persistence-unit name="pu">
    <properties>
        <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
        <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/alien"/>
        <property name="javax.persistence.jdbc.user" value="douglas"/>
        <property name="javax.persistence.jdbc.password" value="123"/>
    </properties>
</persistence-unit>
</persistence>
```

Tem um erro nesse arquivo, ou pelo menos a IDE acusa, mas funciona, confia

- E precisamos linkar a “persistence-unit” ao EntityManager

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("pu");
```

- Ainda assim, o código não funciona, porque ele não sabe que o Alien é uma entidade. Para isso usaremos uma Annotation de Entity e ID

```
@Entity
public class Alien {
    @Id
    private int aid;
    private String aname;
    private String color;
}
```

- Agora o código funciona!!

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("pu");
EntityManager em = emf.createEntityManager();
```

```
Alien a = em.find(Alien.class, 4);
```

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

```
Alien(aid=4, aname=Alien4, color=Blue)
```

- Em seguida, vamos tentar salvar um novo Alien

```
Alien a = new Alien(60, "Doug", "White");
```

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("pu");  
EntityManager em = emf.createEntityManager();
```

```
em.persist(a);
```

- O código executa! Mas o Alien não aparece no banco
- Isso acontece porque faltou a parte do gerenciamento de sessions e transactions, que fazemos assim:

```
Alien a = new Alien(60, "Doug", "White");
```

```
em.getTransaction().begin();  
em.persist(a);  
em.getTransaction().commit();
```

- Agora o dado foi criado!

Conclusão

- A sintaxe do JPA é muito semelhante à do Hibernate
- A vantagem não é de ser melhor do que Hibernate, mas de ser independente da plataforma
- Se mudarmos a ferramente de ORM, podemos manter o mesmo código, a mesma estrutura, pois estamos deixando tudo encapsulado com a API

▼ RESTful Web Service

Complete Java, Spring, and Microservices course

Java developer in 48 hours; technically, it will take months to master.

► https://youtu.be/4XTsAAHW_Tc?si=mWqP-7Ngs901eKiV&t=121780



▼ REST API

REST

- Quando aprendemos a programar, criamos softwares que rodam no navegador, no terminal ou, no máximo em alguma GUI da máquina
- Para criarmos softwares que busquem dados de algum lugar ou mandem dados para algum lugar, precisamos permitir comunicação via internet
- A maioria dos dados fica armazenada em servidores em algum lugar do planeta
- A comunicação com esses dados é feita através de Request e Response
- Essa comunicação é feita através de APIs
- No caso do Java, dentro do servidor, haverá um Servlet e o cliente fará um request para receber dados
- Até agora, estudamos a possibilidade de o Servlet retornar uma página HTML, uma tela para ser renderizada no navegador
- No caso de uma API, contudo, o Servlet pode retornar dados ao invés de design, em formato XML ou JSON
- A ideia de uma Rest API é que, no servidor, teremos um objeto com informações relevantes e, pela API, enviaremos o estado desse objeto em formato XML ou JSON
- Por isso REST = Representational State Transfer
- É relevante pois sabemos que não enviamos um objeto, mas seu estado!
- Por isso também, quando buscamos algum recurso, algum dado, no servidor, não deve nos interessar qual é o processamento que acontece lá dentro
- REST veio para substituir o SOAP

CRUD

- Nos softwares, costumeiramente fazemos CRUDs → Create, Read, Update e Delete sobre os conjuntos de dados e **recursos** que temos

- Os métodos HTTP, referentes ao REST, que mapeam as operações do CRUD são:
 - Create → Post
 - Read → Get
 - Update → Put
 - Delete → Delete

jersey & Spring

- Existem duas formas principais de implementar uma API rest com Java, que são jersey e Spring
- O Spring vem com mais coisa, mas veremos os dois

▼ Jersey

Prática

- Continuaremos usando os Aliens
- No Java, os Aliens serão objetos:

```
public class Alien {
    private Integer id;
    private String name;
    private Integer points;
}

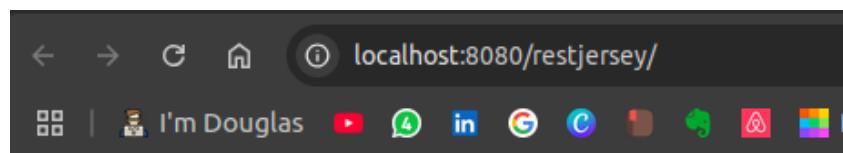
public static void main(String[] args) {
    Alien doug = new Alien();
    doug.id = 1;
    doug.name = "Douglas";
    doug.points = 100;
}
```

- No processo de comunicação com REST API, precisamos transformar esse objeto em um JSON:

```
{
  id: 1,
  name: "Douglas",
```

```
    points: 100  
}
```

- E precisamos também fazer o envio ou recebimento dos dados desse tipo.
- Para isso, usaremos inicialmente o Jersey
- O Jersey é necessário pois REST é apenas um conceito, Jersey é um framework que cuida da estrutura de comunicação REST
- O próprio Maven é capaz de criar um projeto Jersey
- O funcionamento da comunicação lembra os Servlets, mas é mais fácil de utilizar
- **Para iniciar** crie um projeto Maven (eu usei a Extension Java e não a Maven) `jersey-quickstart-webapp`
- Depois, sirva sua aplicação da mesma forma que faz com um Servlet, usando Tomcat : <https://www.linkedin.com/pulse/subindo-um-servidor-local-com-tomcat-e-java-visual-studio-rocha-japmf/?trackingId=9IHL1TZRQzWHwivSA0P9Dw%3D%3D>
- Chegamos a esse ponto:



Jersey RESTful Web Application!

[Jersey resource](#)

Visit the [Project Jersey website](#) for more information on Jersey!

- Temos esse arquivo Java:

```
package com.example;  
  
import javax.ws.rs.GET;  
import javax.ws.rs.Path;  
import javax.ws.rs.Produces;  
  
/**  
 * Example resource class hosted at the URI path "/myresource"
```

```

*/
@Path("/myresource")
public class MyResource {

    /**
     * Method processing HTTP GET requests, producing "text/plain" MIME media
     * type.
     *
     * @return String that will be send back as a response of type "text/plain".
     */
    @GET
    @Produces("text/plain")
    public String getIt() {
        return "Hi there!";
    }
}

```

- O do Telusko usa MediaType.TEXT_PLAIN, é melhor e vamos usar:

```

@GET
@Produces(MediaType.TEXT_PLAIN)
public String getIt() {
    return "Hi there!";
}

```

- O Telusko estava usando o jersey 2.26-b06 e deu erro, mudou para 2.26-b03 e funcionou. A minha é 1.19.4 e apenas mudar para a 2.26-b03 não funcionou... o GPT resolveu

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"
    modelVersion="4.0.0">
    <groupId>com.example</groupId>
    <artifactId>restjersey</artifactId>
    <packaging>war</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>restjersey Jersey Webapp</name>

    <properties>
        <jersey.version>2.26-b03</jersey.version>
        <maven.compiler.source>1.8</maven.compiler.source>
    
```

```
<maven.compiler.target>1.8</maven.compiler.target>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

<build>
    <finalName>restjersey</finalName>
    <plugins>
        <!-- Compiler Plugin -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.1</version>
            <configuration>
                <source>${maven.compiler.source}</source>
                <target>${maven.compiler.target}</target>
            </configuration>
        </plugin>
        <!-- WAR Plugin -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-war-plugin</artifactId>
            <version>3.3.2</version>
            <configuration>
                <failOnMissingWebXml>false</failOnMissingWebXml>
            </configuration>
        </plugin>
        <!-- Embedded Jetty Plugin for Testing -->
        <plugin>
            <groupId>org.eclipse.jetty</groupId>
            <artifactId>jetty-maven-plugin</artifactId>
            <version>9.4.44.v20210927</version>
            <configuration>
                <webApp>
                    <contextPath>/</contextPath>
                    <war>${project.build.directory}/${project.build.finalName}.war</war>
                </webApp>
                <stopKey>stop</stopKey>
                <stopPort>9999</stopPort>
                <scanIntervalSeconds>10</scanIntervalSeconds>
            </configuration>
        </plugin>
    </plugins>
```

```

</build>

<dependencies>
    <!-- Jersey Server -->
    <dependency>
        <groupId>org.glassfish.jersey.core</groupId>
        <artifactId>jersey-server</artifactId>
        <version>${jersey.version}</version>
    </dependency>
    <!-- Jersey Servlet -->
    <dependency>
        <groupId>org.glassfish.jersey.containers</groupId>
        <artifactId>jersey-container-servlet</artifactId>
        <version>${jersey.version}</version>
    </dependency>
    <!-- Jersey JSON Processing -->
    <dependency>
        <groupId>org.glassfish.jersey.media</groupId>
        <artifactId>jersey-media-json-processing</artifactId>
        <version>${jersey.version}</version>
    </dependency>
    <!-- JUnit for Testing -->
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.13.2</version>
        <scope>test</scope>
    </dependency>
    <!-- Jersey Test Framework -->
    <dependency>
        <groupId>org.glassfish.jersey.test-framework.providers</groupId>
        <artifactId>jersey-test-framework-provider-jetty</artifactId>
        <version>${jersey.version}</version>
        <scope>test</scope>
    </dependency>
</dependencies>
</project>

```

- Continuamos recebendo aquele HTML (que vem de um JSP) no endereço
- Se clicarmos em “Jersey Resource” ou acessando “/myresource” na URL, recebemos um 404. O texto esperado “Hi there” especificado na função também não aparece em lugar algum

- Precisamos observar o arquivo web.xml em src/main/webapp/WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This web.xml file is not required when using Servlet 3.0 container,
     see implementation details http://jersey.java.net/nonav/documentation/latest
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi=-
<servlet>
    <servlet-name>Jersey Web Application</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</ser
    <init-param>
        <param-name>com.sun.jersey.config.property.packages</param-name>
        <param-value>com.example</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>Jersey Web Application</servlet-name>
    <url-pattern>/webresources/*</url-pattern>
</servlet-mapping>
</web-app>
```

- Indica o ServletContainer como o servlet que estamos utilizando
- Mas também aponta que o url a ser usado é /webresources/
- No do Telusko funcionou, no meu não

Parou de funcionar aqui:

https://youtu.be/4XTsAAHW_Tc?si=XXTIBg260cs4hi7M&t=123611

Vou passar pro Spring...

▼ Spring - Introdução com SpringBoot



Telusko, nesse vídeo imenso, começa sobre Spring aqui:

https://youtu.be/4XTsAAHW_Tc?si=MuZB34b-AqvtjdlQ&t=131060

Pensei em seguir esse curso:

https://www.youtube.com/watch?v=-Fe0zk-F4OA&ab_channel=Telusko

Mas vou tentar seguir com o primeiro...

- O nome Spring, sem nada acompanhando, entende-se referir ao Spring Framework
- Ja houve Struts, EJB e JPA, mas o Spring une todos
- Spring foca em POJO e Services
- Spring é a combinação de muitos projetos que fazem muitas coisas diferentes:
 - Dependency INjection - Spring Core
 - Web - Spring MVC
 - REST - Spring RES
 - Security
 - JPA
- Spring possui extensa documentação:

Home

Level up your Java code and explore what Spring can do for you.



<https://spring.io/>

Spring Framework Documentation :: Spring Framework

IoC Container, Events, Resources, i18n,
Validation, Data Binding, Type Conversion, SpEL, AOP, AOT.

<https://docs.spring.io/spring-framework/reference/index.html>

- Não há muita necessidade de livros

Prerequisites

- Tem que saber Java
- Database - JDBC
- Maven
- Servlet e JSP é importante, pois Spring transformará as coisas pra Servlets para rodar em TOMCAT
- ORM - Hibernate e JPA
- Internet Protocol
- LDAP \ OAuth

Software Requirements

- Spring oferece “Spring Tool Suite” que é uma IDE em cima do Eclipse específica para Spring
- Atualmente ao pesquisar mostra as extensões de outras IDEs como VSCode
- Depende de JDK > 8
- Usando Spring na versão do Telusko, é exigido que o Tomcat seja baixado separadamente, mas o STS já traz uma versão pronta

Criando um Projeto Maven e Incluindo Spring

- Ele começa criando um workspace no STS
- Ele está fazendo com Maven, copiando e colando as dependências... vamos lá
- Primeiro criamos um projeto Maven vazio:

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apach
<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>
<artifactId>spring</artifactId>
<version>1.0-SNAPSHOT</version>

<name>spring</name>
<!-- FIXME change it to the project's website -->
<url>http://www.example.com</url>

<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
</properties>

<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
```

```
<version>4.11</version>
<scope>test</scope>
</dependency>
</dependencies>

</project>
```

- E adicionamos o Spring Context
- Peguei o 5.1.3 porque é o mesmo que ele usou

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.1.3.RELEASE</version>
</dependency>
```

- Teoricamente funciona, mas ele não rodou nada

Creating Starter Project - Spring Boot

- Uma das ideias do Spring é diminuir o Overhead de configuração que boa parte dos frameworks traz
- SpringBoot usa essa ideia de “Convention Over Configuration”, de usar coisas mais prontas conforme convenção
- O abençoado começa criando um projeto com Spring Boot no STS, vou criar com Spring Initializr no VSCode
- Ele usou a versão 2.1.2, eu fui na 3.4.3 mesmo
- Temos muitas dependências, mas ele não escolheu nenhuma. Escolherei Lombok



Beneficio do VSCode com initializer é que ele já abre uma nova janela

- Assim o Boot cria a estrutura completa do projeto com essa main:

```
package com.example.spring;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication  
public class Application {  
  
    public static void main(String[] args) {  
        SpringApplication.run(Application.class, args);  
    }  
}
```

- Nesse ponto, já podemos executar o projeto:



Aqui ele volta e, depois de 1 minuto de tela branca, o vídeo volta para a parte de Dependency Injection

Dependency Injection

- Esse exemplo exige um projeto criado com SpringBoot
 - Ele explicou que uma empresa que faz LapTop escolhe algum HD e coloca no notebook, não necessariamente cria do nada.
 - A ideia é que vamos pedir um objeto ao Spring e ele providenciará (conforme configurado), ao invés do próprio objeto Laptop criar o objeto HD
 - Começaremos criando uma referência a um objeto Alien e a definição da classe Alien

```
public class Alien {  
    public void code() {  
        System.out.println("Coding...");  
    }  
}
```

```
@SpringBootApplication  
public class Application {
```

```
public static void main(String[] args) {  
    SpringApplication.run(Application.class, args);  
  
    Alien alien;  
  
    alien.code();  
}  
}
```

- Esse código não funciona porque o objeto alien não foi criado. Uma solução é:

```
Alien alien = new Alien();  
alien.code();
```

- Mas o problema é que nesse caso estamos utilizando o new, estamos criando o Alien, não é essa a ideia
- A capacidade de DI do Spring é a de armazenar um conjunto de classes num contexto de aplicação e, quando solicitamos uma classe Alien, o próprio Spring cuida do preenchimento
- Para que isso funcione primeiro precisamos recuperar o contexto do método run

```
ApplicationContext context = SpringApplication.run(Application.class, args);
```

- Esse objeto context tem um método getBean que buscará um objeto daquela classe

```
Alien alien = context.getBean(Alien.class);
```

- Contudo, se rodarmos, vemos um erro:

```
Exception in thread "main" org.springframework.beans.factory.NoSuchBeanDef  
at org.springframework.beans.factory.support.DefaultListableBeanFactory.  
at org.springframework.beans.factory.support.DefaultListableBeanFactory.  
at org.springframework.context.support.AbstractApplicationContext.getBean  
at com.example.spring.Application.main(Application.java:13)
```

- “No qualified bean” significa que não existe um Bean do tipo Alien. Isso ocorre porque a classe existe, mas não foi passada como Bean para o Spring, fazemos isso com uma annotation Component:

```

@Component
public class Alien {
    public void code() {
        System.out.println("Coding...");
    }
}

```

- Agora o software roda:



```

:: Spring Boot ::          (v3.4.3)

2025-03-12T08:56:14.399-03:00 INFO 3326131 --- [main] com.example.spring.Application      : Starting Application using Java 17.0.14 with PID
3326131 (/home/douglasleite/code/Douglas/estudos/java_telusko/Spring/spring/target/classes started by douglasleite in /home/douglasleite/code/Douglas/estudos/java_telusko/Spring/spring)
2025-03-12T08:56:14.401-03:00 INFO 3326131 --- [main] com.example.spring.Application      : No active profile set, falling back to 1 default
profile: "default"
2025-03-12T08:56:14.700-03:00 INFO 3326131 --- [main] com.example.spring.Application      : Started Application in 0.57 seconds (process runn
ing for 0.823)
Coding...

```

- O Spring, quando vê uma classe anotada com Component, se entende como responsável pela criação, gerenciamento e injeção dos objetos daquela classe

Autowire

- O objeto Alien não tem nenhum atributo. Criaremos um objeto Laptop que o Alien terá para programar.

```

public class Laptop {
    public void compile() {
        System.out.println("Compiling...");
    }
}

@Component
public class Alien {
    private Laptop lap;

    public void code() {
        System.out.println("Coding...");
        lap.compile();
    }
}

public static void main(String[] args) {

```

```
ApplicationContext context = SpringApplication.run(Application.class, args);

Alien alien = context.getBean(Alien.class);
alien.code();
}
```

- O código compila, pois confia na injeção de dependência do Spring, mas é claro que um erro vai acontecer pois em momento algum indicamos de onde vem esse Laptop pro Alien

Coding...

```
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "com.
at com.example.spring.Alien.code(Alien.java:11)
at com.example.spring.Application.main(Application.java:14)
```

- Alguém tem que instanciar o Laptop. Poderíamos fazer assim:

```
private Laptop lap = new Laptop();
```

- Mas, novamente, queremos evitar o uso de “new”.
- A primeira solução vinda à mente é colocar um Component no Laptop, assim o Spring cuidará dele

```
@Component
public class Laptop {
    public void compile() {
        System.out.println("Compiling...");
    }
}
```

- Não está errado, mas não resolve. O código continua dando o mesmo erro.
- Isso porque, ao contrário da Main, em momento algum estamos pedindo o Bean do tipo Laptop para o contexto. E nem temos acesso direto ao contexto dentro do Alien para fazê-lo.
- A solução é a annotation **Autowired**

```
@Component
public class Alien {
    @Autowired
    private Laptop lap;
```

```
public void code() {  
    System.out.println("Coding...");  
    lap.compile();  
}  
}
```

- Como é o Spring que cria o Alien, com essa Annotation dizemos que ele deve incluir um atributo Laptop com um Bean Laptop que ele também criou:

```
2025-03-12T09:04:14.546-03:00 INFO 3332078 --- [spring] [main] com  
Coding...  
Compiling...
```

▼ Spring Framework



Então, era pra tudo ficar dentro de Spring, mas seria muita coisa.

Na parte de introdução ele criou um projeto Maven com Spring, mas não usou.

Depois criou um com SpringBoot e demonstrou o básico de Injeção de Dependência

Agora ele deve ir mais a fundo sem o SpringBoot

- Uma das maiores e principais facilidades do Spring é esconder muitas coisas do usuário e fazer por trás
- Agora, vamos fazer as coisas “na mão”, sem pedir que o SpringBoot faça tudo por trás
- Usaremos Spring Framework, em que temos que especificar tudo, sem confiar apenas no Spring Boot
- Criaremos um projeto Spring com Maven (é o mesmo que foi feito antes)
- Podemos criar com as facilidades padrões do VSCode, como feito, de fato, no item acima
- Antes mesmo de incluir o Spring, podemos rodar e vemos um “Hello World!”
- Vamos também criar um objeto Alien

```

public class Alien {
    public void code() {
        System.out.println("Coding...");
    }
}

public class App {
    public static void main(String[] args) {
        Alien alien;
        alien.code();
    }
}

```

- Novamente, o código não funciona pois não criamos o objeto. Não queremos usar o new
- A ideia do Spring é resolver isso por nós. Sabemos que para isso devemos utilizar o método getBean, mas não temos, aqui, o acesso direto ao contexto
- Na verdade, nem temos a dependência do Spring com o método ainda. Vamos adicionar

```

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apach
<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>
<artifactId>spring</artifactId>
<version>1.0-SNAPSHOT</version>

<name>spring</name>
<!-- FIXME change it to the project's website →
<url>http://www.example.com</url>

<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>

```

```

<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.11</version>
        <scope>test</scope>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-context
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.1.3.RELEASE</version>
    </dependency>

</dependencies>

</project>

```

- No SpringBoot, temos fácil acesso ao ApplicationContext que tem o método getBean, aqui teremos que seguir mais alguns passos.

BeanFactory

- Primeiro vamos criar uma BeanFactory com XmlBeanFactory

```

public static void main(String[] args) {
    BeanFactory beanFactory = new XmlBeanFactory(resource);

    Alien alien = beanFactory.getBean("alien");
}

```

- O código indica que teremos um recurso (arquivo) XML que servirá de configuração. Nesse arquivo teremos como indicar que o bean "alien" é da classe Alien
- O próximo passo, portanto, é criar o arquivo xml "spring.xml" na raiz (ao lado do pom):

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://w

```

```
http://www.springframework.org/schema/context http://www.springfr  
/>
```

- Assim ele não vai funcionar, mas já podemos ajustar o código para usá-lo

```
public class App {  
    public static void main(String[] args) {  
        BeanFactory beanFactory = new XmlBeanFactory(new FileSystemResource("beans.xml"));  
  
        Alien alien = (Alien) beanFactory.getBean("alien");  
    }  
}
```

```
Exception in thread "main" org.springframework.beans.factory.NoSuchBeanDefin  
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.  
    at org.springframework.beans.factory.support.AbstractBeanFactory.getMe  
    at org.springframework.beans.factory.support.AbstractBeanFactory.doGet  
    at org.springframework.beans.factory.support.AbstractBeanFactory.getBean  
    at com.example.App.main(App.java:15)
```

- O código está correto, roda, mas acusa o erro de não ter o bean alien definido. Temos que especificar isso no xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
       xmlns:context="http://www.springframework.org/schema/context"  
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://w  
                           http://www.springframework.org/schema/context http://www.springfr  
/>  
  
<bean id="alien" class="com.example.Alien"></bean>  
  
</beans>
```

- Agora podemos buscar esse Alien e chamar a função code e veremos resultado:

```
public static void main(String[] args) {  
    BeanFactory beanFactory = new XmlBeanFactory(new FileSystemResource("beans.xml"));  
    Alien alien = (Alien) beanFactory.getBean("alien");  
    System.out.println(alien.code());
```

```
Alien alien = (Alien) beanFactory.getBean("alien");
alien.code();
}
```

Coding...

ApplicationContext

- Começamos utilizando o BeanFactory, o que funciona, mas o XmlBeanFactory está deprecado
- Além disso, desde que o Spring começou a fazer mais do que apenas injeção de dependência, o objeto ApplicationContext passou a ser utilizado

```
public static void main(String[] args) {
    // BeanFactory beanFactory = new XmlBeanFactory(new
    // FileSystemResource("spring.xml"));

    ApplicationContext context = new ClassPathXmlApplicationContext("spring.x

    Alien alien = (Alien) context.getBean("alien");
    alien.code();
}
```

- Usaremos o ClassPathXmlApplicationContext para encontrar o arquivo pelo nome, mas se rodarmos o código assim recebemos um erro de FileNotFoundException, pois ele não está no Classpath
- Movemos então o arquivo spring.xml para src/main/java (sem entrar no com/example) e agora o código funciona novamente

Spring Container

- Vamos entender o que acontece por trás disso tudo criando um construtor na classe Alien

```
public class Alien {
    public Alien() {
        System.out.println("Alien Object Created...");
    }

    public void code() {
        System.out.println("Coding...");
    }
}
```

```
    }  
}
```

- Rodando agora, recebemos isso no terminal:

```
Alien Object Created...  
Coding...
```

- Naturalmente, ele cria o objeto e depois chama conforme fizemos. Contudo, se eu comentar a instancia de Alien, ainda assim o Spring o cria

```
public static void main(String[] args) {  
    ApplicationContext context = new ClassPathXmlApplicationContext("spring.x  
  
    // Alien alien = (Alien) context.getBean("alien");  
    // alien.code();  
}
```

```
Alien Object Created...
```

- Isso acontece porque, quando criamos o ApplicationContext, criamos todos os objetos descritos no spring.xml, pois o Spring entende que eventualmente o usaremos.
- O Context na verdade criará um SpringContainer, que ficará dentro da JVM com os Beans, que são as classes que definimos no spring.xml
- Se definirmos vários objetos inúteis no spring.xml, o Spring criará todos eles
- Quando, depois, usamos o método "getBean", o SpringContainer nos devolve o objeto daquela classe que criamos
- Como isso interage com atributos?
- Vamos criar um atributo idade no Alien.

```
public class Alien {  
    public int age;
```

- E vamos acessá-lo na Main

```
public static void main(String[] args) {  
    ApplicationContext context = new ClassPathXmlApplicationContext("spring.x  
  
    Alien alien = (Alien) context.getBean("alien");
```

```
    alien.code();
    System.out.println(alien.age);
}
```

Alien Object Created...

Coding...

0

- Por default, o valor de um inteiro não inicializado é 0. Pode alterá-lo antes de verificar

```
Alien alien = (Alien) context.getBean("alien");
alien.code();
alien.age = 15;
System.out.println(alien.age);
```

Alien Object Created...

Coding...

15

- Agora vamos instanciar um segundo alien e ver sua idade

```
Alien alien = (Alien) context.getBean("alien");
alien.code();
alien.age = 15;
System.out.println(alien.age);
```

```
Alien alien2 = (Alien) context.getBean("alien");
System.out.println(alien2.age);
```

Alien Object Created...

Coding...

15

15

- Ao contrário do esperado, a idade de ambos foi alterada para 15

- Isso acontece porque, nas duas vezes, pedimos um objeto de id alien, ele entrega o mesmo objeto já que ele tem o objeto
- Por isso, por padrão, os Beans no Spring são **SingletonBeans**

Singletor x Prototype

- Em alguma circunstância, é possível que você queira que seus Beans não seja singleton
- Vamos fazer a alteração no spring.xml

```
<bean id="alien" class="com.example.Alien" scope="singleton"></bean>
```

- Por padrão, quando criamos os Beans no xml, o atributo scope tem o valor singleton. Ou seja, se pedirmos 10 vezes por Alien, receberemos sempre o mesmo
- A outra opção de scope é "prototype"

```
<bean id="alien" class="com.example.Alien" scope="prototype"></bean>
```

- Agora, sempre que pedirmos um Alien para o SpringContainer, ele criará um novo:

```
Alien alien = (Alien) context.getBean("alien");
alien.code();
alien.age = 15;
System.out.println(alien.age);
```

```
Alien alien2 = (Alien) context.getBean("alien");
System.out.println(alien2.age);
```

Alien Object Created...

Coding...

15

Alien Object Created...

0

- Agora a idade está zerada.
- Contudo, nesse caso, se não instanciarmos nenhum Alien, nenhum será criado:

```

public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext("spring.xml");

    // Alien alien = (Alien) context.getBean("alien");
    // alien.code();
    // alien.age = 15;
    // System.out.println(alien.age);

    // Alien alien2 = (Alien) context.getBean("alien");
    // System.out.println(alien2.age);
}

// nada

```

- Existem outras opções de escopo mas não abordaremos

Setter Injection

- No nosso Alien, temos a idade publica, mas deveríamos utilizar private e métodos de acesso

```

@Data
public class Alien {
    private int age;

```

- Agora temos que ajustar no main e mudar o escopo para singleton

```

public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext("spring.xml");

    Alien alien = (Alien) context.getBean("alien");
    alien.code();
    System.out.println(alien.getAge());
}

<bean id="alien" class="com.example.Alien"></bean>

```

- Rodando, vemos 0. Mas queremos criar um valor padrão pra idade
- Uma forma de fazer isso é definindo no atributo

```
public class Alien {  
    private int age = 10;
```

- Podemos também criar com construtor ou sempre dar um set
- Mas outra opção é colocar esse valor por default no spring, usando a tag property

```
<bean id="alien" class="com.example.Alien">  
    <property name="age" value="10"/>  
</bean>
```

- Rodando agora, mesmo depois de tirar a atribuição na classe, retorna 10
- Isso só é possível utilizando o método setAge, porque após criar o objeto, o SpringContainer chama esse método com o valor definido
- Esse é o erro quando voltamos para o public e tentamos rodar:

```
mar 13, 2025 9:05:27 AM org.springframework.context.support.AbstractApplicationEventPublisher publishEvent  
ADVERTÊNCIA: Exception encountered during context initialization - cancelling refresh attempt  
Exception in thread "main" org.springframework.beans.factory.BeanCreationException:  
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:537)  
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:501)  
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:479)  
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:459)  
    at org.springframework.beans.factory.support.AbstractBeanFactory.lambda$doGetBean$0(AbstractBeanFactory.java:317)  
    at org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleton(DefaultSingletonBeanRegistry.java:234)  
    at org.springframework.beans.factory.support.AbstractBeanFactory doGetBean(AbstractBeanFactory.java:311)  
    at org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBeanFactory.java:267)  
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.preInstantiateSingletons(DefaultListableBeanFactory.java:883)  
    at org.springframework.context.support.AbstractApplicationContext.finishBeanFactoryInitialization(AbstractApplicationContext.java:877)  
    at org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:552)  
    at org.springframework.context.support.ClassPathXmlApplicationContext.<init>(ClassPathXmlApplicationContext.java:139)  
    at org.springframework.context.support.ClassPathXmlApplicationContext.<init>(ClassPathXmlApplicationContext.java:125)  
    at com.example.App.main(App.java:12)
```

```
Caused by: org.springframework.beans.NotWritablePropertyException: Invalid property 'age' of bean class [com.example.Alien]: Bean property 'age' is not writable or has an invalid setter method. Does the parameter type match the return type or is the get method annotated? If your bean has a custom getter for this property, you may need to apply the @Transient annotation to the field or the get method.  
    at org.springframework.beans.BeanWrapperImpl.createNotWritablePropertyException(BeanWrapperImpl.java:145)  
    at org.springframework.beans.AbstractNestablePropertyAccessor.setPropertyValue(AbstractNestablePropertyAccessor.java:215)  
    at org.springframework.beans.AbstractNestablePropertyAccessor.setPropertyValue(AbstractNestablePropertyAccessor.java:196)  
    at org.springframework.beans.AbstractPropertyAccessor.setPropertyValue(AbstractPropertyAccessor.java:111)  
    at org.springframework.beans.AbstractPropertyAccessor.setPropertyValue(AbstractPropertyAccessor.java:105)
```

```
at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:524)
... 13 more
```

- Também é possível confirmar isso colocando um print no setAge, mas não faremos isso
- Um detalhe importante é que o Spring usará o nome na property para gerar o nome do setter: age → setAge. Mas não tem nada a ver com o nome do atributo. Se deixarmos o atributo como age mas mudarmos o setter, o Spring quebra, mas se mudarmos o nome do atributo não acontece nada

```
public class Alien {
    public int alalala;

    public void setAge(int a) {
        this.alalala = a;
    }

    // 10
```

- A situação muda de figura, contudo, se estamos falando de dados não primitivos, de referência

Reference Attribute

- Vamos colocar um valor de referência, ou seja, um objeto não primitivo no Alien
- Vamos criar uma classe Laptop e colocá-la no Alien e usar no método code

```
public class Laptop {
    public void compile() {
        System.out.println("Compiling...");
    }
}

@Data
public class Alien {
    private int age;
    private Laptop laptop;

    public Alien() {
        System.out.println("Alien Object Created...");
    }
}
```

```

public void code() {
    System.out.println("Coding...");
    laptop.compile();
}
}

```

- Rodando, recebemos um NullPointerException, porque em momento algum informamos o Spring de que deveria incluir o Laptop
- Para que o Spring possa inclui-la, primeiro precisamos criar o bean de Laptop e incluí-lo como propriedade no ALien, mas de forma diferente

```

<bean id="alien" class="com.example.Alien">
    <property name="age" value="10"></property>
    <property name="laptop" ref="laptop"></property>
</bean>

<bean id="laptop" class="com.example.Laptop"></bean>

```

- Rodando agora, temos o resultado esperado

```

public void code() {
    System.out.println("Coding...");
    laptop.compile();
}

```

Coding...
Compiling...

Constructor Injection

- Até agora aprendemos como preencher as propriedades utilizando setters com Spring
- Como podemos fazer isso com construtor?

```

public Alien(int age) {
    System.out.println("Alien Object Created...");
    this.age = age;
}

```

- Mas quem roda o construtor é o SpringContainer

- Se rodarmos assim, recebemos esse erro:

```

mar 13, 2025 9:23:42 AM org.springframework.context.support.AbstractApplication
ADVERTÊNCIA: Exception encountered during context initialization - cancelling
Exception in thread "main" org.springframework.beans.factory.BeanCreationEx
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory
    at org.springframework.beans.factory.support.AbstractBeanFactory.lambda$doGetBe
    at org.springframework.beans.factory.support.DefaultSingletonBeanRegister
    at org.springframework.beans.factory.support.AbstractBeanFactory doGetB
    at org.springframework.beans.factory.support.AbstractBeanFactory.getBean
    at org.springframework.beans.factory.support.DefaultListableBeanFactory
    at org.springframework.context.support.AbstractApplicationContext.finishB
    at org.springframework.context.support.AbstractApplicationContext.refresh
    at org.springframework.context.support.ClassPathXmlApplicationContext.<
    at org.springframework.context.support.ClassPathXmlApplicationContext.<
    at com.example.App.main(App.java:12)

Caused by: org.springframework.beans.BeanInstantiationException: Failed to in
    at org.springframework.beans.factory.support.SimpleInstantiationStrategy.
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanF
    ... 13 more

Caused by: java.lang.NoSuchMethodException: com.example.Alien.<init>()
    at java.lang.Class.getConstructor0(Class.java:3082)
    at java.lang.Class.getDeclaredConstructor(Class.java:2178)
    at org.springframework.beans.factory.support.SimpleInstantiationStrategy.
    ... 14 more

```

- Reclamando que não foi encontrado um construtor default (Alien()), porque o nosso construtor é parametrizado
- Temos que fazer o ajuste no xml para passar o valor no construtor

```

<bean id="alien" class="com.example.Alien">
    <!-- <property name="age" value="10"></property> -->
    <constructor-arg value="12"></constructor-arg>
    <property name="laptop" ref="laptop"></property>
</bean>

```

- Ao rodar, vemos 12

- Não colocamos referência do parâmetro na tag porque funciona como construtor, mas precisamos incluir os itens na ordem correta
- Podemos construir com referência também, mudando de value para ref
- Uma boa forma de decidir entre construtor e setter é se o atributo é essencial ou opcional

Autowire

- Vamos voltar ao uso dos setters e properties:

```
public Alien() {
    System.out.println("Alien Object Created...");
}

<bean id="alien" class="com.example.Alien">
    <property name="age" value="10"></property>
    <property name="laptop" ref="laptop"></property>
</bean>
```

- Aliens atualmente precisam de um Laptop para programar, mas eles podem programar em qualquer tipo de computador. Criaremos uma interface e o Desktop:

```
public interface IComputer {
    public void compile();
}

public class Laptop implements IComputer {
    @Override
    public void compile() {
        System.out.println("Compiling on Laptop...");
    }
}

public class Desktop implements IComputer {
    @Override
    public void compile() {
        System.out.println("Compiling on Desktop...");
    }
}
```

- Agora temos que ajustar no Alien para ter qualquer computador

```

@Data
public class Alien {
    private int age;
    private IComputer computer;

<bean id="alien" class="com.example.Alien">
    <property name="age" value="10"></property>
    <property name="computer" ref="laptop"></property>
</bean>

```

- Se rodarmos

```

public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext("spring.xml");

    Alien alien = (Alien) context.getBean("alien");
    alien.code();
    System.out.println(alien.getAge());
}

---
Alien Object Created...
Coding...
Compiling on Laptop...
10

```

- Isso porque está conectando a propriedade computer com laptop, que é um IComputer
- Podemos agora criar um Bean Desktop

```

<bean id="alien" class="com.example.Alien">
    <property name="age" value="10"></property>
    <property name="computer" ref="desktop"></property>
</bean>

<bean id="laptop" class="com.example.Laptop"></bean>

<bean id="desktop" class="com.example.Desktop"></bean>

```

```
---
Alien Object Created...
Coding...
Compiling on Desktop...
10
```

- Vamos agora testar mudar o id de laptop para com e passar para o Alien:

```
<bean id="alien" class="com.example.Alien">
    <property name="age" value="10"/></property>
    <property name="computer" ref="computer"/></property>
</bean>

<bean id="computer" class="com.example.Laptop"/></bean>

<bean id="desktop" class="com.example.Desktop"/></bean>
```

- Ao rodar, funciona, volta para o Laptop
- Outra opção é, ao invés de definir exatamente qual queremos, podemos usar, na tag bean, a propriedade **autowire**, que fará o Spring procurar pelo Bean de acordo com o método que escolhermos

```
<bean id="alien" class="com.example.Alien" autowire="byName">
    <property name="age" value="10"/></property>
    <!-- <property name="computer" ref="computer"/></property> →
</bean>

<bean id="computer" class="com.example.Laptop"/></bean>

<bean id="desktop" class="com.example.Desktop"/></bean>
```

- Como usamos “**byName**”, ele usará o nome “computer”, conforme está no bean Laptop e na classe. Roda
- Agora vamos voltar para o desktop e comentar o laptop. Agora não temos mais o mesmo nome, então não podemos usar byName, vamos usar **byType**

```
<bean id="alien" class="com.example.Alien" autowire="byType">
    <property name="age" value="10"/></property>
    <!-- <property name="computer" ref="computer"/></property> →
</bean>
```

```
<!-- <bean id="computer" class="com.example.Laptop"></bean> -->  
  
<bean id="desktop" class="com.example.Desktop"></bean>
```

- O código roda com Desktop
- E se tivermos os dois?

```
<bean id="alien" class="com.example.Alien" autowire="byType">  
  <property name="age" value="10"></property>  
  <!-- <property name="computer" ref="computer"></property> -->  
</bean>  
  
<bean id="computer" class="com.example.Laptop"></bean>  
  
<bean id="desktop" class="com.example.Desktop"></bean>
```

- Ambos são aptos por tipo. Se rodarmos vemos esse erro:

```
Caused by: org.springframework.beans.factory.NoUniqueBeanDefinitionException  
    at org.springframework.beans.factory.config.DependencyDescriptor.resolveCandidate(De  
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.selectBe  
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.resolveB  
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.  
... 13 more
```

Primary

- Como resolver isso? Temos 2 classes do tipo Computer, e o Spring não consegue decidir qual usar
- Naturalmente, é possível utilizar os métodos anteriores, escolhendo pelo nome ou especificando a propriedade
- Mas outra solução é colocar um atributo "primary" no que queremos usar:

```
<bean id="alien" class="com.example.Alien" autowire="byType">  
  <property name="age" value="10"></property>  
  <!-- <property name="computer" ref="computer"></property> -->  
</bean>  
  
<bean id="computer" class="com.example.Laptop" primary="true"></bean>
```

```
<bean id="desktop" class="com.example.Desktop"></bean>
```

- O código roda com Laptop

▼ Spring Boot JDBC

- Para conectar o código ao banco no Java, usamos JDBC
- Usando JDBC diretamente, teremos que fazer muitas coisas manuais para atingir um código limpo e performático
- Um exemplo é a **Connection Pool** que cria um conjunto de conexões com o banco que podem ser reutilizadas, diminuindo o gasto de performance com a criação de conexões
- Existe algo chamado JdbcTemplate que nos ajuda com o encapsulamento de alguns passos do JDBC e melhoria performática, o Spring Boot JDBC traz isso
- Vamos criar um projeto novo no VSCode
- Usaremos como dependência apenas o Lombok, a JDBC API e uma conexão com o banco, que para esse exemplo será a H2, um banco em memória pré-configurado que facilitará o uso nesse exemplo



Se quisermos utilizar um banco que não o h2, faremos as configurações no application.properties

- O código já roda, e o H2 está presente, mas não fazemos nada com ele
- Vamos criar uma pasta/package "model" e, dentro, criar a classe que representará a tabela Alien

```
@Data  
@Component  
@Scope("prototype")  
public class Alien {  
    private int id;  
    private String name;  
    private String tech;  
}
```

- Usamos o Component para que o próprio Spring lide com a criação do Alien, mas com Scope prototype para que não aja como singleton
- Vamos buscar o Alien no Context

```
ApplicationContext context = SpringApplication.run(SpringJdbcApplication.class);

Alien a1 = context.getBean(Alien.class);
a1.setId(111);
a1.setName("Douglas");
a1.setTech("Java");

System.out.println(a1);

// Alien(id=111, name=Douglas, tech=Java)
```

- Agora queremos inserir esse Alien no banco
- Podemos, se formos usar JDBC diretamente, escrever todos os 7 passos do JDBC na Main
- Isso é ruim por ser muito código, por não ser bem replicável e por estar na main
- Ao invés disso, começaremos criando o AlienRepository no package repo

```

@Repository
public class AlienRepository {
    public void save(Alien alien) {
        System.out.println("Added");
    }

    public List<Alien> findAll() {
        return new ArrayList<>();
    }
}

```

- A Annotation Repository é basicamente igual à Component, mas traz mais clareza ao código separando repository, service, controller, etc
- Naturalmente, ela não interage com o banco ainda, mas podemos chama-la na main

```

public static void main(String[] args) {
    ApplicationContext context = SpringApplication.run(SpringJdbcApplication.class);

    Alien a1 = context.getBean(Alien.class);
    a1.setId(111);
    a1.setName("Douglas");
    a1.setTech("Java");

    System.out.println(a1);

    AlienRepository repository = context.getBean(AlienRepository.class);
    repository.save(a1);

    System.out.println(repository.findAll());
}

---
// Alien(id=111, name=Douglas, tech=Java)
// Added
// []

```

JDBC Template

- Para podermos começar a realmente fazer alterações no banco, vamos alterar o código do Alien Repository
- Vamos acessar o banco usando um JDBC Template

```
@Repository
@Data
public class AlienRepository {

    @Autowired
    private JdbcTemplate template;
```

- Naturalmente, o Spring cuida da construção, configuração e injeção do objeto
- Agora usaremos o template para salvar o Alien
- No JDBC, utilizavamos executeQuery para buscar dados e executeUpdate quando queríamos fazer alguma alteração no banco
- Aqui os nomes são query e update apenas, mas seguem o mesmo princípio. Portanto, usaremos update

```
public void save(Alien alien) {
    template.update("insert into alien (id, name, tech) values (?, ?, ?)", alien.getId());
    System.out.println("Added");
}
```

- Se rodarmos agora, contudo, receberemos um erro:

```
Caused by: org.h2.jdbc.JdbcSQLException: Table "ALIEN" not found;
insert into alien (id, name, tech) values (?, ?, ?) [42104-232]
        at org.h2.message.DbException.getJdbcSQLException(DbException.java:51)
        at org.h2.message.DbException.getJdbcSQLException(DbException.java:47)
        at org.h2.message.DbException.get(DbException.java:223)
        at org.h2.message.DbException.get(DbException.java:199)
        at org.h2.command.Parser.getTableOrViewNotFoundDbException(Parser.java:1521)
        at org.h2.command.Parser.getTableOrViewNotFoundDbException(Parser.java:1521)
        at org.h2.command.Parser.readTableOrView(Parser.java:7895)
        at org.h2.command.Parser.readTableOrView(Parser.java:7857)
        at org.h2.command.Parser.parseInsert(Parser.java:1521)
        at org.h2.command.Parser.parsePrepared(Parser.java:698)
        at org.h2.command.Parser.parse(Parser.java:581)
        at org.h2.command.Parser.parse(Parser.java:556)
```

```
at org.h2.command.Parser.prepareCommand(Parser.java:484)
at org.h2.engine.SessionLocal.prepareLocal(SessionLocal.java:645)
at org.h2.engine.SessionLocal.prepareCommand(SessionLocal.java:561)
at org.h2.jdbc.JdbcConnection.prepareStatement(JdbcConnection.java:116)
at org.h2.jdbc.JdbcPreparedStatement.<init>(JdbcPreparedStatement.java:54)
at org.h2.jdbc.JdbcConnection.prepareStatement(JdbcConnection.java:319)
at com.zaxxer.hikari.pool.ProxyConnection.prepareStatement(ProxyConnection.java:242)
at com.zaxxer.hikari.pool.HikariProxyConnection.prepareStatement(HikariProxyConnection.java:111)
at org.springframework.jdbc.core.JdbcTemplate$SimplePreparedStatementCreator.createPreparedStatement(JdbcTemplate.java:940)
at org.springframework.jdbc.core.JdbcTemplate.execute(JdbcTemplate.java:607)
... 16 more
```

- Isso ocorre porque, em momento algum, criamos o schema do banco de dados, muito menos o populamos. Faremos isso utilizando arquivos .sql na pasta resources
- O primeiro será o schema.sql

```
CREATE TABLE
alien (
    id int PRIMARY KEY,
    name varchar(32),
    tech varchar(32)
);
```

- O segundo será o data.sql, que criará dados iniciais

```
INSERT INTO
alien (id, name, tech)
VALUES
(101, 'Douglas', 'Java'),
(102, 'Tayla', 'Law'),
(103, 'Shiva', 'JS'),
(104, 'Pandora', 'Go');
```

- Rodando o código agora, temos o resultado esperado: a mensagem Added e o código sem erros.
- A lista de aliens ainda está vazia, pois estamos apenas retornando um array vazio
- Vamos ajustar o método findAll do Repository para fazer uma query na tabela
- O método query recebe a query como String e uma de 3 opções

- ResultSetExtractor
- RowCallbackHandler
- RowMapper
- No exemplo, usaremos o RowMapper, que é uma FunctionalInterface

```

public List<Alien> findAll() {

    RowMapper<Alien> mapper = new RowMapper<Alien>() {
        @Override
        public Alien mapRow(ResultSet resultSet, int rowNum) throws SQLException {
            Alien a = new Alien();
            a.setId(resultSet.getInt(1));
            a.setName(resultSet.getString(2));
            a.setTech(resultSet.getString(3));

            return a;
        }
    };

    List<Alien> aliens = template.query("select * from alien", mapper);
    return aliens;
}

// Alien(id=111, name=Douglas, tech=Java)
// Added
// [Alien(id=101, name=Douglas, tech=Java), Alien(id=102, name=Tayla, tech=L

```

- Naturalmente, podemos reduzir a FunctionalInterface a uma Lambda

```

public List<Alien> findAll() {

    List<Alien> aliens = template.query("select * from alien",
        (resultSet, rowNum) -> {
            Alien a = new Alien();
            a.setId(resultSet.getInt(1));
            a.setName(resultSet.getString(2));
            a.setTech(resultSet.getString(3));

            return a;
        });
}

```

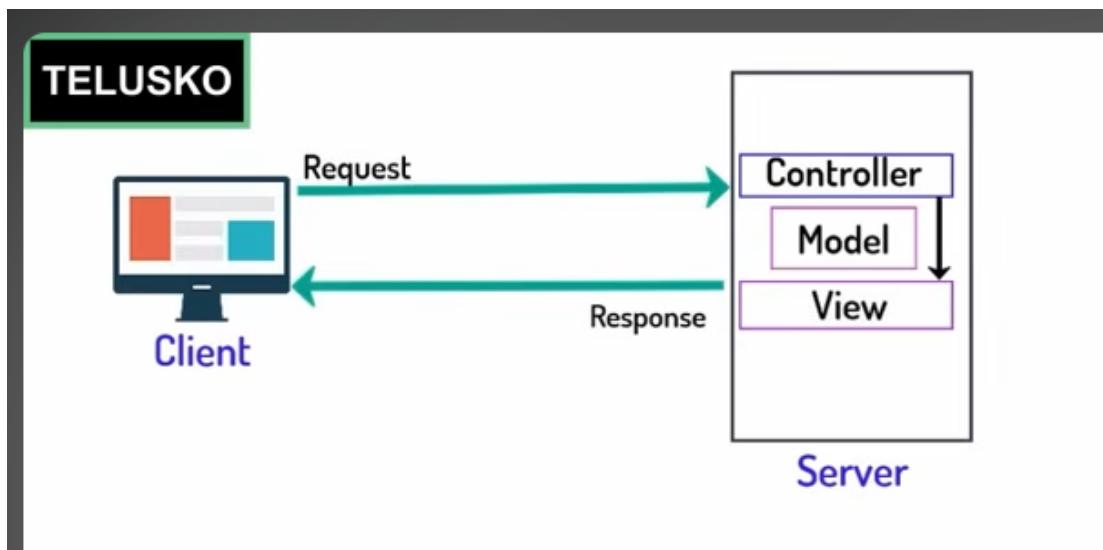
```
return aliens;  
}
```

- Não dá para usar um construtor porque o Alien está como Component

▼ Spring MVC

Teoria

- Antigamente fazíamos a configuração do Spring MVC pela configuração dele mesmo
- Hoje, Spring Boot faz tudo ser bem mais fácil
- Spring MVC se baseia na arquitetura MVC (que já vimos na parte de Servlet)



- Uma diferença é que, no caso dos Servlets, tínhamos que criar um Servlet para cada caso, para cada requisição, e isso ficava confuso
- A solução foi criar múltiplos Controllers, mas apenas um "Front Controller", chamado Dispatcher Servlet, que receberia todas as requisições e direcionaria para o Servlet correto
- O Spring MVC cuida dessa parte

Prática

- É possível iniciar com uma aplicação Spring MVC ou uma aplicação Spring Boot com MVC. A configuração sem o Boot é muito grande, então faremos primeiro com Boot e depois sem

▼ Spring Boot MVC

- Vamos criar um aplicativo com Spring Initializr com
 - Lombok
 - Spring Web
- O pom terá essas dependencias:

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

- Já é possível rodar e ver funcionando. A primeira diferença é que o software não morre, ele continua rodando:

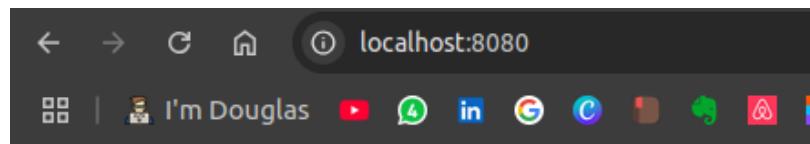


:: Spring Boot :: (v3.4.3)

```
2025-03-18T19:01:51.514-03:00 INFO 2399521 --- [mvc-boot] [main]
2025-03-18T19:01:51.517-03:00 INFO 2399521 --- [mvc-boot] [main]
2025-03-18T19:01:52.442-03:00 INFO 2399521 --- [mvc-boot] [main]
2025-03-18T19:01:52.456-03:00 INFO 2399521 --- [mvc-boot] [main]
2025-03-18T19:01:52.456-03:00 INFO 2399521 --- [mvc-boot] [main]
```

```
2025-03-18T19:01:52.500-03:00 INFO 2399521 --- [mvc-boot] [main]
2025-03-18T19:01:52.502-03:00 INFO 2399521 --- [mvc-boot] [main]
2025-03-18T19:01:52.965-03:00 INFO 2399521 --- [mvc-boot] [main]
2025-03-18T19:01:52.983-03:00 INFO 2399521 --- [mvc-boot] [main]
```

- Já podemos nos atentar ao fato de que o Spring subiu um próprio servidor Tomcat na porta 8080



Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Tue Mar 18 19:03:07 BRT 2025

There was an unexpected error (type=Not Found, status=404).

- Diferente do exemplo com servlets e tomcat manual, não existe uma página index.jsp padrão, recebemos um erro.
- Para vermos algo criaremos, dentro de src/main, mas fora de java, uma página webapp com um arquivo index.jsp dentro

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <title>Oi</title>
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link rel="stylesheet" type="text/css" media="screen" href="main.css" />
    <script src="main.js"></script>
  </head>
  <body>
    Welcome to Douglas!
  </body>
</html>
```

- Mesmo atualizando a página, nada muda

Criando o Controller

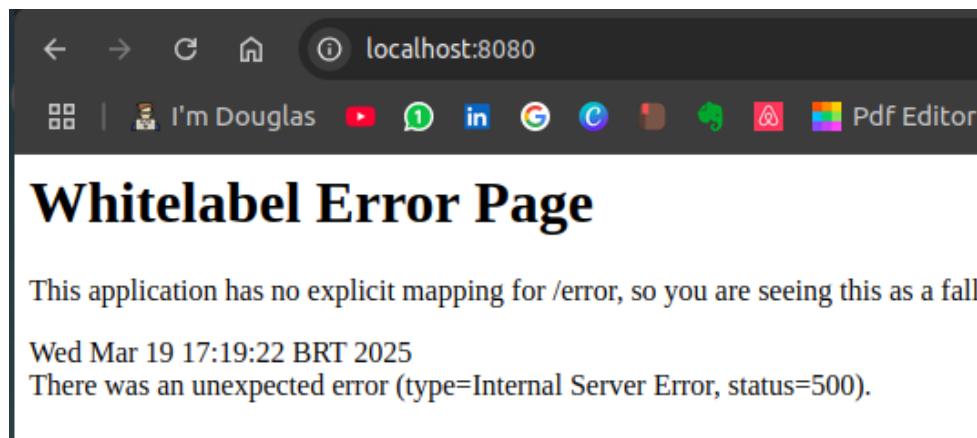
- Criaremos um Controller chamado `HomeController.java` com a anotação `Controller`

```
@Controller
public class HomeController {
}
```

- Sem Spring Boot, precisaríamos criar uma configuração para atribuir esse Controller a um caminho
- Agora precisamos de um método que será chamado quando eu requisitar a página home

```
@RequestMapping("/")
public void home() {
    System.out.println("home page requested");
}
```

- A annotation serve para indicar que, quando for acessado o endereço /, essa função deve ser chamada
- Se executarmos o código e tentarmos acessar o endereço ainda recebemos um erro, mas um erro diferente



- Além disso, estouro um erro no terminal, mas junto com a mensagem esperada:

```
home page requested
2025-03-19T17:19:22.983-03:00 ERROR 129156 --- [mvc-boot] [nio-8080-exec-1]
jakarta.servlet.ServletException: Circular view path []: would dispatch back t
```

```
at org.springframework.web.servlet.view.InternalResourceView.prepare  
at org.springframework.web.servlet.view.InternalResourceView.renderM
```

- Ou seja, funcionou! Mas está faltando o redirecionamento para a página JSP
- O primeiro problema é que o Spring MVC não foi feito para criar sites, ou seja, não é a forma “padrão” de retornar telas como JSP
- O Spring, na verdade, foi feito para ser um servidor REST, que retorne dados
- Contudo, é possível retornar um JSP usando a dependencia Jasper

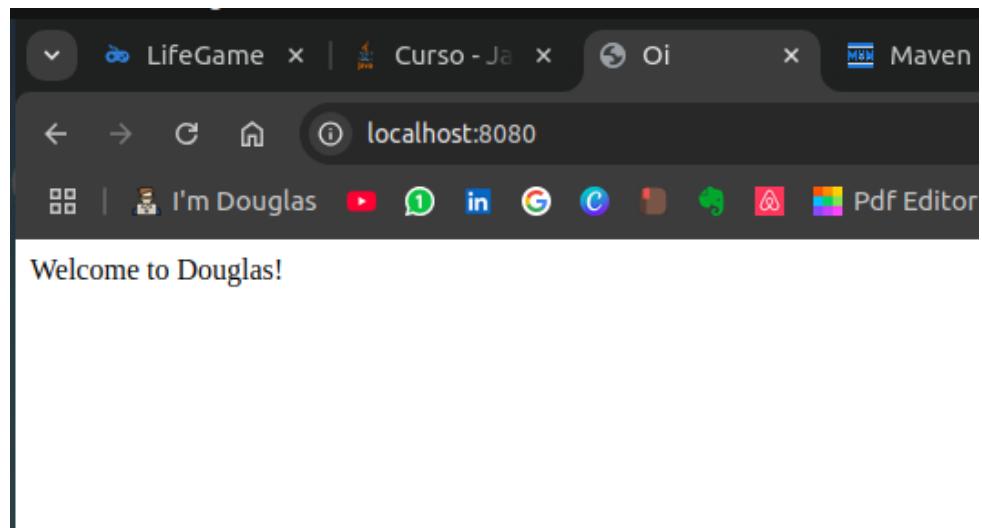
```
<!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-jasper -->  
<dependency>  
    <groupId>org.apache.tomcat</groupId>  
    <artifactId>tomcat-jasper</artifactId>  
    <version>9.0.4</version>  
</dependency>
```



No curso ele só joga isso no pom e funciona, aqui deu erro, então coloquei isso com ajuda do GPT:

```
<!-- Tomcat Embedded Jasper para JSP -->  
<dependency>  
    <groupId>org.apache.tomcat.embed</groupId>  
    <artifactId>tomcat-embed-jasper</artifactId>  
</dependency>  
  
<!-- JSTL (para usar tags padrão em JSPs) -->  
<dependency>  
    <groupId>jakarta.servlet.jsp.jstl</groupId>  
    <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>  
    <version>3.0.0</version>  
</dependency>  
<dependency>  
    <groupId>org.glassfish.web</groupId>  
    <artifactId>jakarta.servlet.jsp.jstl</artifactId>  
    <version>3.0.1</version>  
</dependency>
```

- Agora vemos a tela que criamos

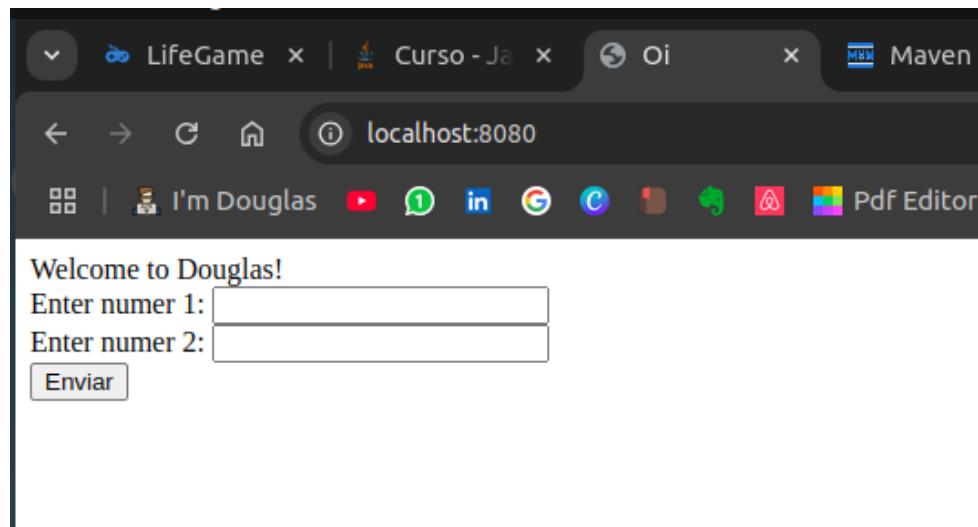


User Input

- Se agora quisermos receber dados do usuário
- Vamos receber 2 números e retornar a soma
- Portanto, na home, criaremos um form com os números para adição

```
<body>
    Welcome to Douglas!

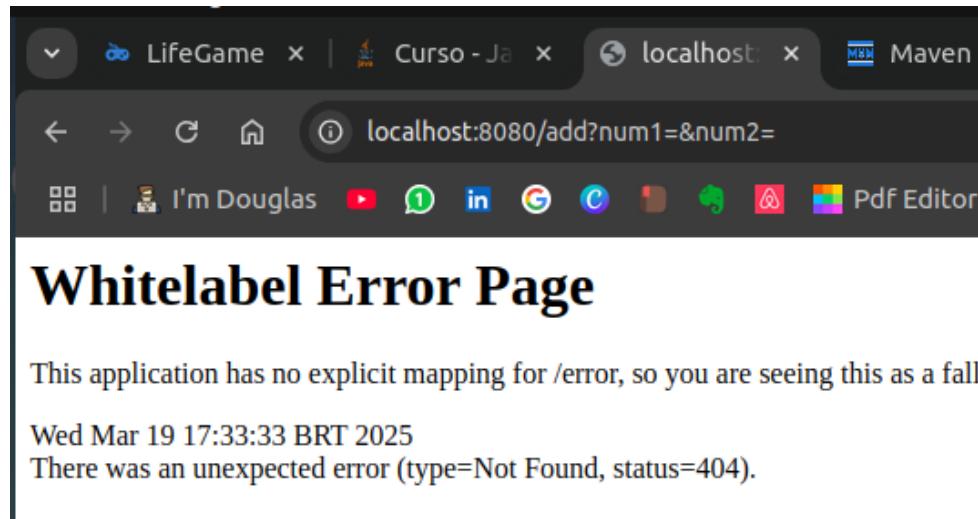
    <form action="add">
        Enter numer 1: <input type="text" name="num1" /><br />
        Enter numer 2: <input type="text" name="num2" /><br />
        <input type="submit" />
    </form>
</body>
```



- Interessante que apenas mudar o jsp não resolve, temos que atualizar o Spring, como funciona com CSS do Vaadin
- Temos que criar a função que vai de fato realizar a adição quando chamada no form

```
@RequestMapping("/add")
public String add() {
    return "";
}
```

- Agora quando clicamos em "Enviar" vemos isso:



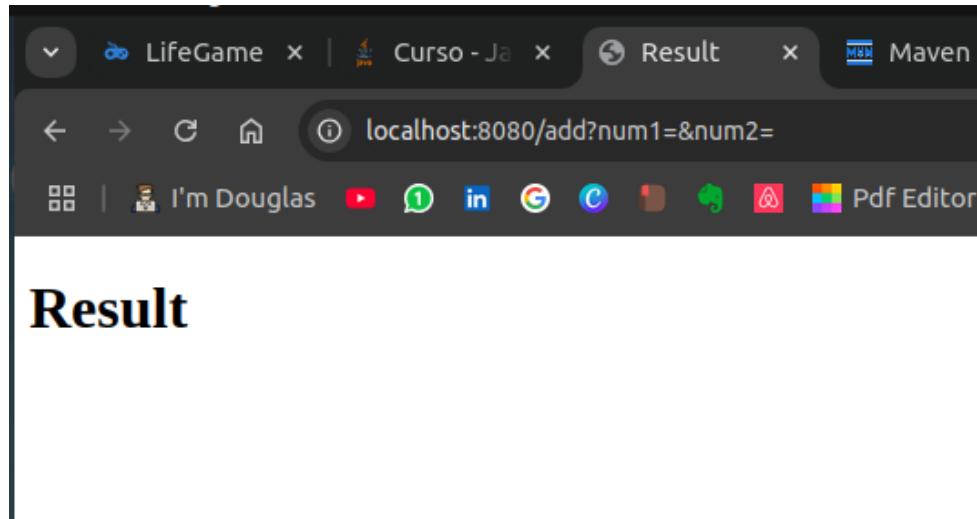
- Precisamos criar um result.jsp para retornar o resultado

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Result</title>
</head>
<body>
    <h1>Result</h1>
</body>
</html>
```

- E vamos retornar essa página no método add

```
@RequestMapping("/add")
public String add() {
    return "result.jsp";
}
```



- Mas não estamos usando os valores da tela anterior
- Para isso, como fazíamos com Servlets, acessaremos o `HttpServletRequest` no método add
- Além disso, precisamos passar o resultado de alguma forma para a tela final. Usaremos a sessão

```
@RequestMapping("/add")
public String add(HttpServletRequest request) {
    int i = Integer.parseInt(request.getParameter("num1"));
    int j = Integer.parseInt(request.getParameter("num2"));
```

```

int k = i + j;

HttpSession session = request.getSession();
session.setAttribute("num3", k);

return "result.jsp";
}

```

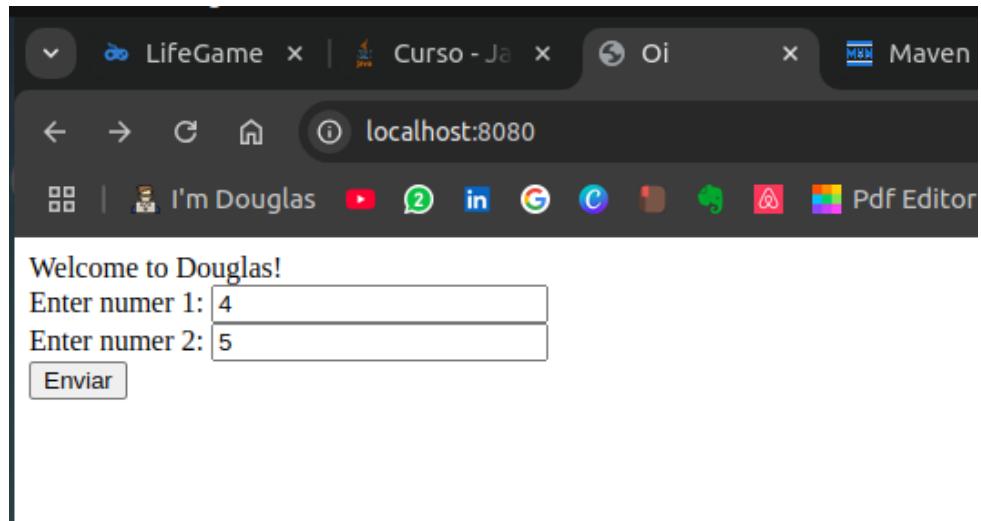
- Agora vamos acessar o valor no jsp e recuperar o valor

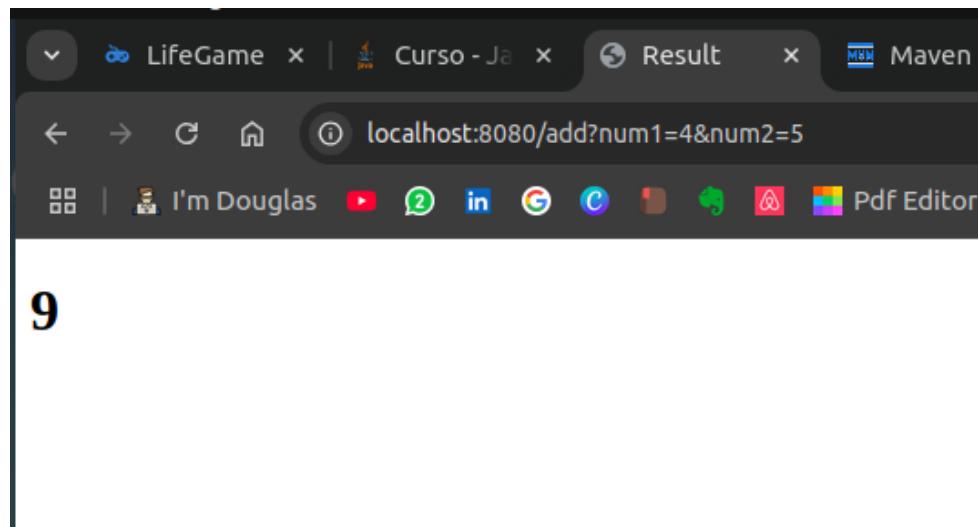
```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>Result</title>
</head>
<body>
<h1>${num3}</h1>
</body>
</html>

```

- Agora se, na primeira tela, preenchermos os valores e clicarmos enviar:

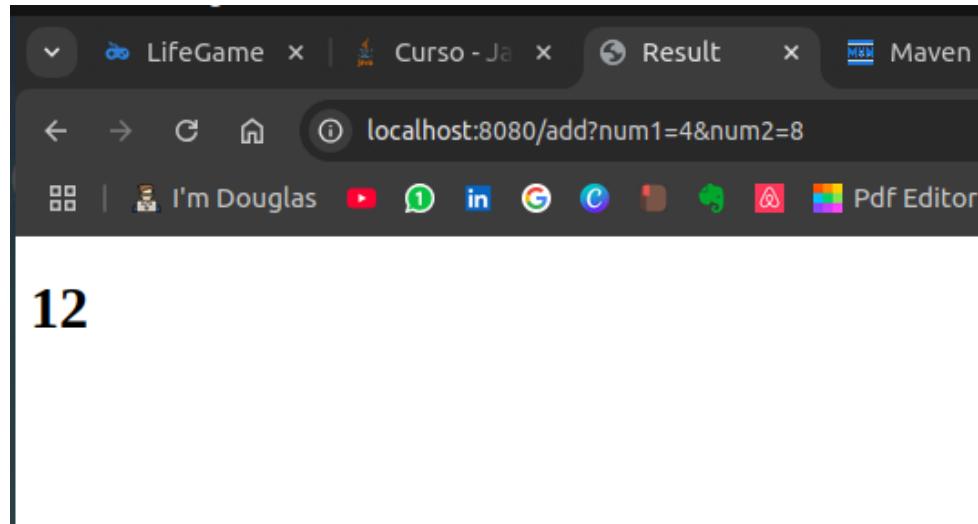




RequestParam

- No Spring, existe uma forma mais fácil de recuperar os parâmetros passados na URL, os RequestParams

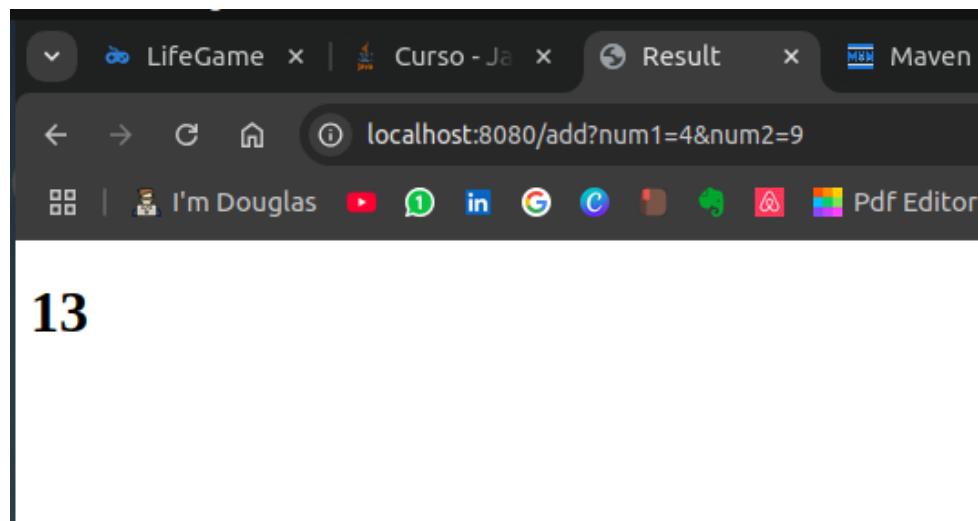
```
@RequestMapping("/add")
public String add(@RequestParam("num1") int i, @RequestParam("num2") int j) {
    int k = i + j;
    HttpSession session = request.getSession();
    session.setAttribute("num3", k);
    return "result.jsp";
}
```



- Interessante que pareceu demorar um tiquinho mais... mas o curso não fala nada sobre
- Além disso, é possível que o Spring já faça a injeção do HttpSession, dispensando o HttpServletRequest

```
@RequestMapping("/add")
public String add(@RequestParam("num1") int i, @RequestParam("num2") i
int k = i + j;
session.setAttribute("num3", k);

return "result.jsp";
}
```



- O Session, contudo, ainda incomoda

ModelAndView

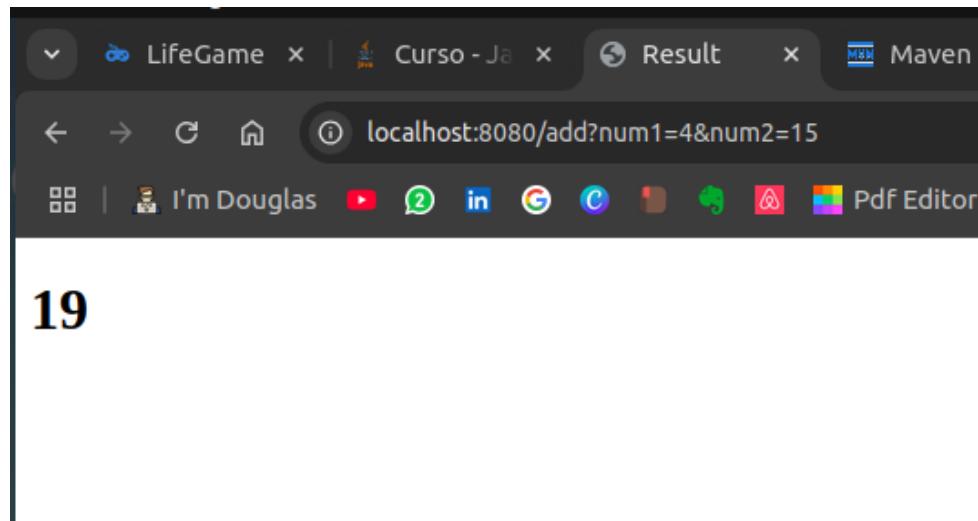
- Model tem dados, o nosso dado é o resultado "k", precisamos passar esse modelo para a View
- O Spring MVC tem um objeto ModelAndView, que comporta tanto o modelo (os dados) quanto a tela (o JSP) e abstrai a comunicação entre os dois

```
@RequestMapping("/add")
public ModelAndView add(@RequestParam("num1") int i, @RequestParam(
int k = i + j;

ModelAndView modelAndView = new ModelAndView();
```

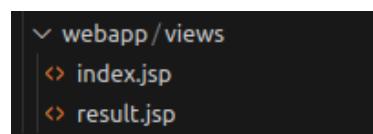
```
modelAndView.setViewName("result.jsp");
modelAndView.addObject("num3", k);

return modelAndView;
}
```



Prefix and Suffix

- O problema a ser abordado agora é a extensão .jsp, pois podemos no futuro usar outra tecnologia
- Além disso, todas as telas estão na pasta webapp, queremos alterar isso
- Vamos criar a pasta views e colocar ambas as telas dentro



- Também tiraremos o .jsp das telas no código

```
@Controller
public class HomeController {
    @RequestMapping("/")
    public String home() {
        return "index";
    }

    @RequestMapping("/add")
    public ModelAndView add(@RequestParam("num1") int i, @RequestParam(
```

```

int k = i + j;

ModelAndView modelAndView = new ModelAndView();
modelAndView.setViewName("result");
modelAndView.addObject("num3", k);

return modelAndView;
}
}

```

- Voltamos a ver o erro 404 na página
- Para informarmos o Spring onde os arquivos estão e qual a extensão, usaremos uma configuração no application.properties

```

spring.application.name=mvc-boot
spring.mvc.view.prefix=/views/
spring.mvc.view.suffix=.jsp

```

Model and ModelMap

- Já começamos a utilizar o ModelAndView para unir o objeto e a view
- Mas será que é a melhor opção de retornar a tela e os dados?
- Existe uma opção de manter a forma de enviar os dados de forma encapsulada mas separar da tela usando a classe model, que terá uma referência no método

```

@RequestMapping("/add")
public String add(Model model, @RequestParam("num1") int i, @RequestPa
    int k = i + j;
    model.addAttribute("num3", k);

    return "result";
}

```

- Além do model, existe o ModelMap, que substitui perfeitamente o Model:

```

@RequestMapping("/add")
public String add(ModelMap model, @RequestParam("num1") int i, @Reque
    int k = i + j;
    model.addAttribute("num3", k);

```

```
        return "result";
    }
```

- O resultado é absolutamente o mesmo, mas o ModelMap usa um mapa de atributos (sei lá a diferença)

ModelAttribute

- Até agora, alimentamos o modelo com números, vamos tentar com um Alien

```
@Data
public class Alien {
    private int id;
    private String name;
}
```

- Tentaremos criar um Alien a partir da tela, no index.jsp

```
<form action="addAlien">
    Enter Alien Id: <input type="text" name="id" /><br />
    Enter Alien Name: <input type="text" name="name" /><br />
    <input type="submit" />
</form>
```

```
@RequestMapping("/addAlien")
public String addAlien(Model model, @RequestParam int id, @RequestPara

    Alien a = new Alien();
    a.setId(id);
    a.setName(name);

    model.addAttribute("alien", a);

    return "result";
}
```

```
<body>
    <h1>${alien}</h1>
</body>
```

Welcome to Douglas!

Enter Alien Id:

Enter Alien Name:

Alien(id=1, name=Douglas)



No curso ele manteve como @RequestParam("id"), mas funcionou direitinho não usar a String

- Funcionou! Mas ainda não está da forma ideal. Se tivermos vários parâmetros todos estarão na URL
- Seria bom poder receber dessa forma:

```
@RequestMapping("/addAlien")
public String addAlien(Model model, Alien alien) {
    model.addAttribute("alien", a);

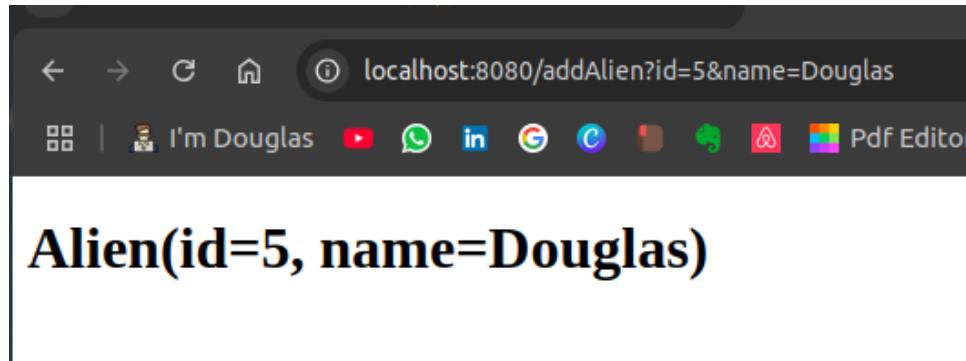
    return "result";
}
```

- Para isso, usaremos a annotation ModelAttribute

```
@RequestMapping("/addAlien")
public String addAlien(Model model, @ModelAttribute Alien alien) {
    model.addAttribute("alien", alien);
```

```
        return "result";
    }
```

- Como o nome dos atributos que estão sendo enviados no form é igual aos atributos de Alien, ele é capaz de preencher sozinho e o código já funciona



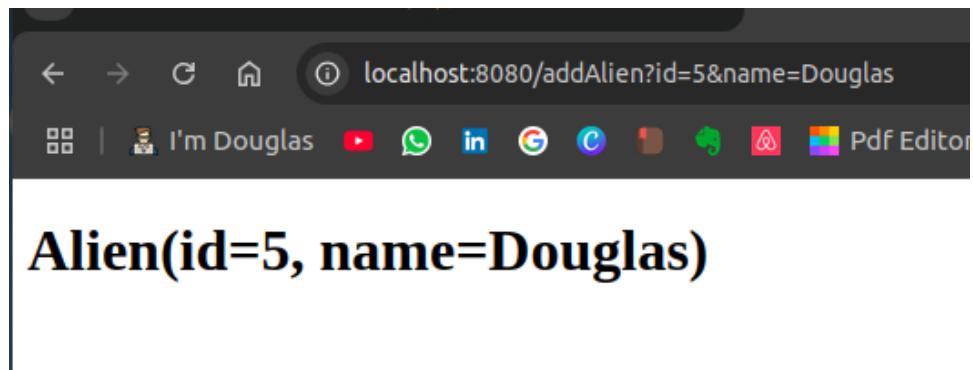
A URL fica igual?? É só pra facilitar a assinatura do método mesmo

- É possível ainda, se utilizarmos o mesmo nome no parâmetro e no JSP, remover o model

```
@RequestMapping("/addAlien")
public String addAlien(@ModelAttribute Alien alien) {
    return "result";
}
```

- Tudo continua funcionando
- Ainda é possível remover a anotação completamente, mas não é uma boa prática

```
@RequestMapping("/addAlien")
public String addAlien(Alien alien) {
    return "result";
}
```



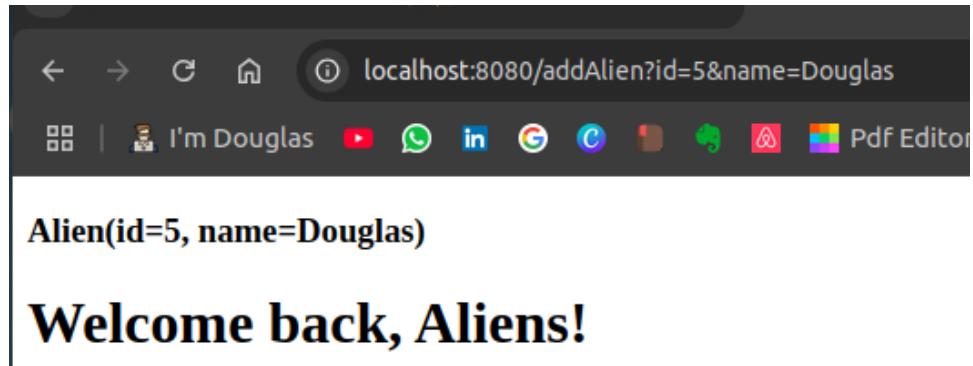
- Agora queremos dar um olá para os Aliens, mas com um valor configurável:

```
<body>
<h3>${alien}</h3>

<h1>Welcome back, ${name}!</h1>
</body>
```

- Os métodos de Request não são o lugar ideal para fazer isso, mas podemos criar um método a parte

```
@ModelAttribute
public void modelData(Model m) {
    m.addAttribute("name", "Aliens");
}
```



- Dessa forma, o atributo “Aliens” sempre será passado, independente de qual Request seja chamado

▼ Spring MVC (Sem Boot)

Complete Java, Spring, and Microservices course
Java developer in 48 hours; technically, it will take months
to master.

https://youtu.be/4XTsAAHW_Tc?si=5kLer08JW5lfEba-&t=141294

ALL IN ONE!

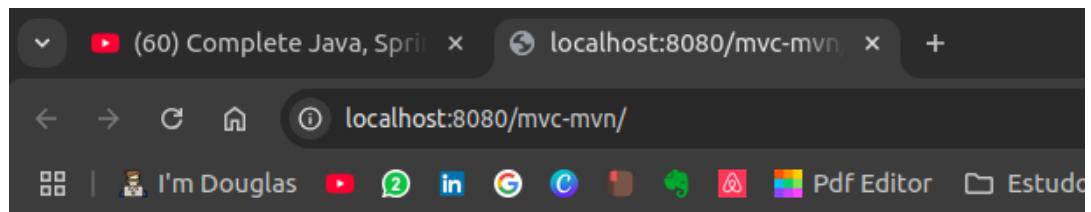
Hours

Java

Spring

Microservices

- Até agora fizemos proveito do SpringBoot para fazer toda a configuração necessária do Spring MVC
- Agora, faremos tudo de forma manual
- Para isso, criaremos um projeto maven, ao invés de Spring
- Usaremos o arquétipo de webapp do Maven
- Já temos o index.jsp, mas diferente do Spring, não tem o Tomcat pronto
- Se ainda tiver o Tomcat usado em outros tópicos, basta rodar o mvn package, achar o war no target e “Run on Server”. Se der errado, abre a aba “Servers” embaixo só pra ele lembrar que tem
- Já é acessível:



Hello World!

- Agora vamos buscar o código que já criamos na etapa de MVC com Boot e jogar aqui ajustando os pacotes

```
src/main
└── java/com/douglas/mvc_mvn
    ├── Alien.java
    ├── HomeController.java
    └── MvcMvnApplication.java
        └── webapp
            └── WEB-INF
                └── web.xml
            └── index.jsp
```

- Naturalmente o código apresenta erros pois não temos a dependência do Spring no pom (por algum motivo não acusou os erros no VSCode, precisei atualizar o pom pro Java 17 pra ele olhar de novo e acusar os erros)
- Vamos buscar o Spring Mvc no maven repository

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.1.4.RELEASE</version>
</dependency>
```

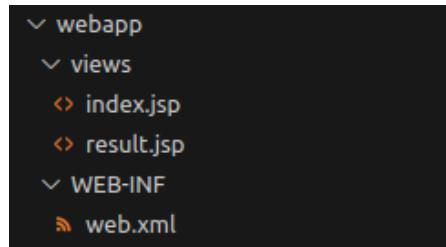
- Também adicionarei o Lombok

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.24</version>
    <scope>provided</scope>
</dependency>

<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <configuration>
        <annotationProcessorPaths>
            <path>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
            </path>
        </annotationProcessorPaths>
    </configuration>
</plugin>
```

- Agora não tem mais erros nem no HomeController nem no Alien, mas ainda tem erros no MvcMvnApplication, mas como não estamos usando SpringBoot não precisamos dele, podemos deletar
- Em seguida, para que o código funcione corretamente, precisamos das telas copiadas do último projeto

- Vamos também apagar a index.jsp que já existe



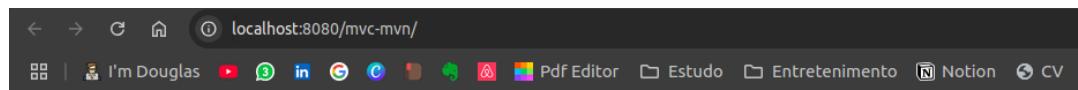
- Podemos agora rodar o código (package e restart server), mas recebemos apenas uma tela com 404, já que ele buscou a tela onde o index estava
- Isso por si já está errado, já que o correto seria o request ser direcionado para o Controller
- Faremos essas configurações no web.xml
- Podemos apagar o nome que está lá
- Vamos adicionar o servlet e mapear os requests para o servlet
- O servlet para onde queremos que os requests sejam mapeados é o DispatcherServlet, do Spring MVC, que podemos acessar

```

<web-app>
  <servlet>
    <servlet-name>douglas</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</serv
  </servlet>

  <servlet-mapping>
    <servlet-name>douglas</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
  
```

- Precisamos também configurar o DispatcherServlet para mais uma vez direcionar o request para o HomeController
- Se tentarmos rodar:



HTTP Status 500 – Internal Server Error

Type Exception Report

Message Error instantiating servlet class [org.springframework.web.servlet.DispatcherServlet]

Description The server encountered an unexpected condition that prevented it from fulfilling the request.

Exception

```
jakarta.servlet.ServletException: Error instantiating servlet class [org.springframework.web.servlet.DispatcherServlet]
    org.apache.catalina.authenticator.AuthenticatorBase.invoke(AuthenticatorBase.java:482)
    org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:93)
    org.apache.catalina.valves.AbstractAccessLogValve.invoke(AbstractAccessLogValve.java:676)
    org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:341)
    org.apache.coyote.http11.Http11Processor.service(Http11Processor.java:389)
    org.apache.coyote.AbstractProcessorLight.process(AbstractProcessorLight.java:63)
    org.apache.coyote.AbstractProtocol$ConnectionHandler.process(AbstractProtocol.java:894)
    org.apache.tomcat.util.net.NioEndpoint$SocketProcessor.doRun(NioEndpoint.java:1742)
    org.apache.tomcat.util.net.SocketProcessorBase.run(SocketProcessorBase.java:52)
    org.apache.tomcat.util.threads.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
    org.apache.tomcat.util.threads.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:636)
    org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)
    java.base/java.lang.Thread.run(Thread.java:840)
```

Root Cause

```
java.lang.NoClassDefFoundError: javax/servlet/http/HttpServlet
    java.base/java.lang.ClassLoader.defineClass1(Native Method)
    java.base/java.lang.ClassLoader.defineClass(ClassLoader.java:1017)
    java.base/java.security.SecureClassLoader.defineClass(SecureClassLoader.java:150)
    org.apache.catalina.loader.WebappClassLoaderBase.findClassInternal(WebappClassLoaderBase.java:2293)
    org.apache.catalina.loader.WebappClassLoaderBase.findClass(WebappClassLoaderBase.java:756)
    org.apache.catalina.loader.WebappClassLoaderBase.loadClass(WebappClassLoaderBase.java:1254)
    org.apache.catalina.loader.WebappClassLoaderBase.loadClass(WebappClassLoaderBase.java:1121)
    java.base/java.lang.ClassLoader.defineClass1(Native Method)
    java.base/java.lang.ClassLoader.defineClass(ClassLoader.java:1017)
    java.base/java.security.SecureClassLoader.defineClass(SecureClassLoader.java:150)
```

- Nessa mensagem não dá pra ver, mas no do Telusko mostra que não foi encontrado um arquivo "telusko-servlet.xml"
- Criaremos, dentro de WEB-INF, um douglas-servlet.xml
- Começaremos com a tag beans

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans http://www.springframework.org/schema/context http://www.springframework.org/schema/context http://www.springframework.org/schema/mvc http://www.springfrir
```

```
</beans>
```

- Depois teremos que definir o contexto, o pacote base
- Depois temos que informar que estamos usando anotações

```
<context:component-scan base-package="com.douglas"></context:component-scan>
<context:annotation-config></context:annotation-config>
```

- Com isso, se tivessemos o caminho completo e a extensão no Controller, já conseguiríamos executar o código. Mas vamos configurar isso também

```
<bean class="org.springframework.web.servlet.view.InternalResourceViewFactoryBean">
    <property name="prefix" value="/views/"></property>
    <property name="suffix" value=".jsp"></property>
</bean>
```

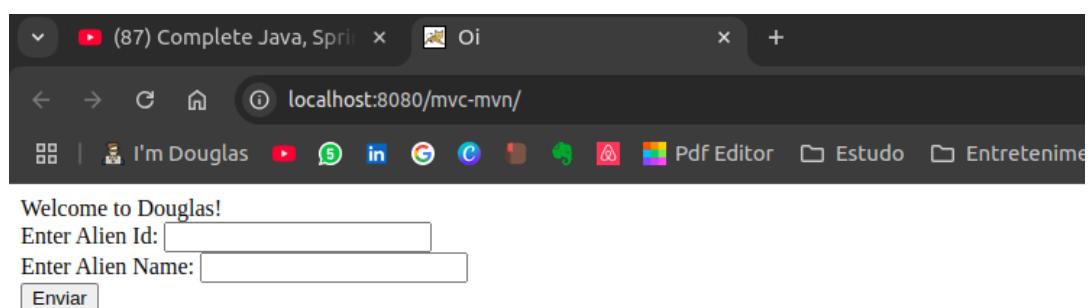
- Contudo, ainda recebemos 404. Um problema possível que o Telusko abordou foi a necessidade de permitir o uso de JSTL
- No momento, o primeiro request dá 500, porque bate no DispatcherServlet, depois de dar f5 fico recebendo 404. O dele, no f5, funciona

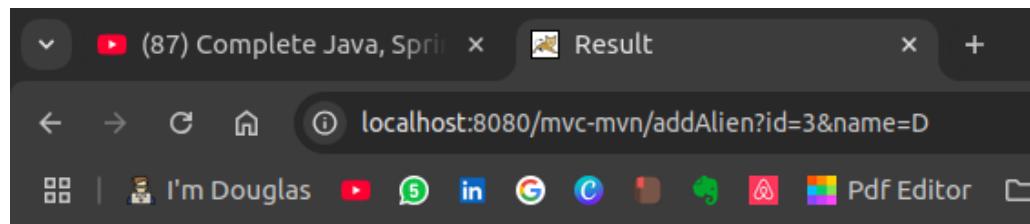


Depois de alguns vai e vens com GPT, identifiquei que era um problema de versão:

O Spring 5.1.4 não roda no Tomcat 11 e nem com Java 17, então fiz o downgrade do Tomcat pra 9.0.41 e do Java pra 11

Depois disso, a tela reclamou do JSTL, foi só tirar que funcionou!





#{alien}

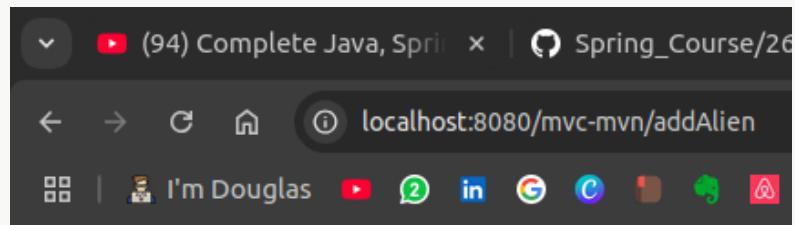
Welcome back, #{name}!

- Não deu jeito de funcionar 100%. Tentei um bocado de coisa, inclusive com o GPT, mas não deu jeito



Depois de um tempo, já na parte de ORM, identifiquei o que estava faltando, e era o trecho que o Telusko coloca no topo de todo arquivo .jsp. Copiei no repositório e funcionou!

```
<%@ page language="java" contentType="text/html; charset=ISO-8
pageEncoding="ISO-8859-1" isELIgnored="false"%>
```

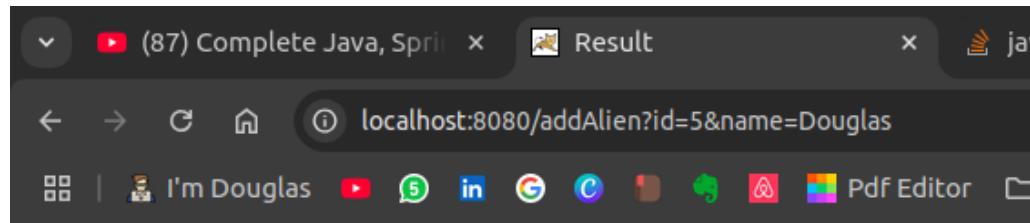


Alien(id=5, name=Douglas)

Welcome back, Aliens!

▼ Spring Boot MVC (part 2)

- Por algum motivo ele voltou pro Boot. Não sei qual mas vai me ajudar também, então vamos lá



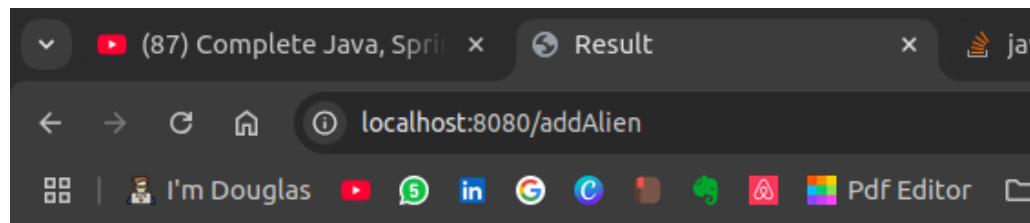
Alien(id=5, name=Douglas)

Welcome back, Aliens!

Post Mapping

- Quando passamos os dados na URL, estamos usando um método GET. Portanto, temos utilizado GET, que é a forma errada
- Vamos reajustar o método no action do inex.jsp

```
<form action="addAlien" method="post">
    Enter Alien Id: <input type="text" name="id" /><br />
    Enter Alien Name: <input type="text" name="name" /><br />
    <input type="submit" />
</form>
```



Alien(id=6, name=Douglas)

Welcome back, Aliens!

- Os dados somem da URL
- Já no Controller, conseguimos utilizar o mesmo método pois no RequestMapping não definimos o method, por isso ele atende a ambos.

Para restringir:

```
@RequestMapping(value = "/addAlien", method = RequestMethod.POST)
public String addAlien(Alien alien) {
    return "result";
}
```

Ou

```
@PostMapping(value = "/addAlien")
public String addAlien(Alien alien) {
    return "result";
}
```

Get Mapping

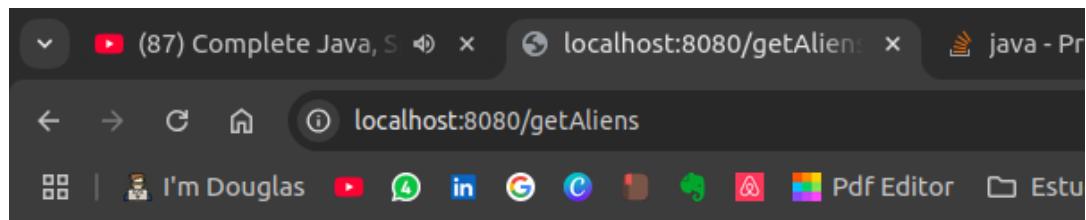
- Agora sim criaremos um Get para buscar todos os Aliens com getAliens
- A primeira coisa será adicionar um AllArgs ao Alien

```
@Data
@AllArgsConstructor
public class Alien {
    private int id;
    private String name;
}
```

- Agora vamos criar uma lista de Aliens e retorna-la como toString

```
@GetMapping("/getAliens")
public String getAlien() {
    List<Alien> aliens = Arrays.asList(new Alien(1, "Douglas"), new Alien(2, "T
        return aliens.toString();
}
```

- Ao rodar:



Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Tue Mar 25 18:34:18 BRT 2025

There was an unexpected error (type=Not Found, status=404).

JSP file [/views/[Alien(id=1, name=Douglas), Alien(id=2, name=Tayla)].jsp] not found

- O erro é facilmente percebido: o Controller tenta achar uma página com esse nome e não consegue
- Vamos então criar uma página showAliens.jsp

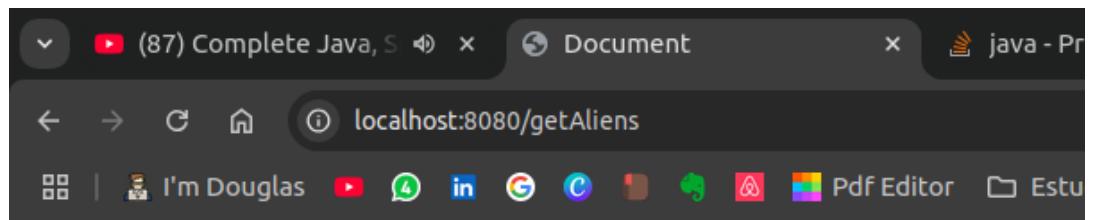
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    ${result}
  </body>
</html>
```

- E passaremos o dado pelo model

```
@GetMapping("/getAliens")
public String getAlien(Model m) {
    List<Alien> aliens = Arrays.asList(new Alien(1, "Douglas"), new Alien(2, "Tayla"));
    m.addAttribute("result", aliens);

    return "showAliens";
}
```

- Agora funciona!



- Agora tentaremos buscar um dado específico, mas com ORM!

▼ Spring ORM Theory

- Para aprender o Spring ORM Theory, usaremos os mesmos projetos do Spring MVC

Teoria

- Até agora já vemos o Spring Core, que trabalha com injeção de dependência
- Vimos também o Spring JDBC, que facilita comunicação com banco de dados
- Vimos ainda o Spring MVC, que nos permite criar o Front Controller
- Também usamos, de certa forma, o Spring REST
- O Spring JDBC, apesar de ser mais fácil do que o JDBC padrão, ainda traz algumas dificuldades
- Dificuldades essas que serão aliviadas utilizando um ORM (Object Relational Mapping) e JPA (Java Persistence API) com Hibernate e Spring ORM

Spring Hibernate Configuration

- Voltaremos, agora, para o projeto mvc_mvn, que não usa Spring Boot para as configurações
- Adicionaremos algumas dependências
- A primeira será o Hibernate Core

```
<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.4.1.Final</version>
</dependency>
```

- Em seguida, a dependência que será o meio termo entre Hibernate e Spring, o Spring ORM

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-orm -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
```

```
<version>5.1.4.RELEASE</version>
</dependency>
```

- Existe também a parte de Transações, que o Spring pode cuidar para nós com Spring Tx ou Spring Transaction

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-tx -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>5.1.4.RELEASE</version>
</dependency>
```

- Ainda precisaremos comunicar com o banco, no nosso caso MySQL, e para isso precisamos do MySQL Connector

```
<!-- https://mvnrepository.com/artifact/com.mysql/mysql-connector-j -->
<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <version>8.0.33</version>
</dependency>
```

- Finalmente, usaremos uma dependência para gerenciar o Connection Pool chamada C3P0

```
<!-- https://mvnrepository.com/artifact/com.mchange/c3p0 -->
<dependency>
    <groupId>com.mchange</groupId>
    <artifactId>c3p0</artifactId>
    <version>0.9.5.3</version>
</dependency>
```

- Agora, com todas as dependências, vamos configurar o Hibernate
- Podemos fazer isso diretamente no douglas-servlet.xml
- Primeiro vamos ter que adicionar o spring tx nos beans

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
```

```
xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.
    http://www.springframework.org/schema/context http://www.springfr
    http://www.springframework.org/schema/mvc http://www.springfr
    http://www.springframework.org/schema/tx http://www.springfr
```

- Em seguida, incluiremos algumas configurações (vamos só colar) esse é o resultado:

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.
        http://www.springframework.org/schema/context http://www.springfr
        http://www.springframework.org/schema/mvc http://www.springfr
        http://www.springframework.org/schema/tx http://www.springfr

<context:component-scan base-package="com.douglas.mvc_mvn"></contex
<context:annotation-config></context:annotation-config>

<bean class="org.springframework.web.servlet.view.InternalResourceViewRes
    <!-- <property name="viewClass"
        value="org.springframework.web.servlet.view.JstlView"></property> -->
    <property name="prefix" value="/views/"></property>
    <property name="suffix" value=".jsp"></property>
</bean>

<bean id="myDataSource" class="com.mchange.v2.c3p0.ComboPooledDataS
    <property name="driverClass" value="com.mysql.jdbc.Driver"></property>
    <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/telusko"><p
    <property name="user" value="root"></property>
    <property name="password" value="12345678"></property>

    <property name="minPoolSize" value="5" />
    <property name="maxPoolSize" value="10" />
    <property name="maxIdleTime" value="30000" />
</bean>
```

```

<bean id="sessionFactory" class="org.springframework.orm.hibernate5.Local
<property name="dataSource" ref="myDataSource" />
<property name="packagesToScan" value="com.telusko.springmvc.model" /
<property name="hibernateProperties">
<props>
<prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
<prop key="hibernate.show_sql">true</prop>
</props>
</property>
</bean>

<bean id="myTransactionManager"
      class="org.springframework.orm.hibernate5.HibernateTransactionManager">
<property name="sessionFactory" ref="sessionFactory" />
</bean>

<tx:annotation-driven transaction-manager="myTransactionManager" />

<bean class="org.springframework.web.servlet.view.InternalResourceViewRes
<property name="prefix" value="/view/" /></property>
<property name="suffix" value=".jsp" /></property>
</bean>

</beans>

```



O código do Telusko no git usa **packageToScan** ao invés de **packagesToScan**. Deu uma mensagem de erro reclamando e eu troquei, funcionou direitinho.

- Algumas dessas configurações foram alteradas como jdbcUrl, user e password

```

<property name="jdbcUrl" value="jdbc:mysql://localhost:3306/mysqlteste"></
<property name="user" value="douglas" /></property>
<property name="password" value="123" /></property>

```

- Além do próprio packagesToScan

```
<property name="packagesToScan" value="com.douglas.mvc_mvn" />
```

- Rodando o código, funciona, mas não estamos acessando o banco ainda

MySQL and DAO

- Já temos um MySQL rodando com Docker, então pularemos essa etapa do video
- Por algum motivo, o banco está sem tabelas, então criaremos junto com ele:

```
CREATE TABLE alien (id int, name varchar(30));
```

```
INSERT INTO alien VALUES (101, 'Douglas'), (102, 'Tayla');
```

```
SELECT * FROM alien;
```

		id	name
		int	varchar(30)
	>	101	Douglas
	>	102	Tayla



Vou testar a extensão Database Connection do VSCode ao invés do DBeaver para testar

- Agora precisamos de uma camada DAO (Data Access Object) para conectar com o banco de dados
- Criaremos o AlienDAO.java para isso

```
package com.douglas.mvc_mvn;
```

```
public class AlienDAO {
```

```
}
```

- Nossa primeira missão é buscar todos os dados, então criaremos o método para fazer isso
- Começando no DAO, precisamos de uma sessão para buscar os dados, e criaremos essa sessão com o SessionFactory, que injetaremos com Autowired

```
@Autowired
private SessionFactory sessionFactory;

public List<Alien> getAliens() {
    Session session = sessionFactory.getCurrentSession();
}
```

- Esse Autowire, sem Spring Boot, não é automático, mas na verdade criamos um bean sessionFactory na nossa etapa de configuração:

```
<bean id="sessionFactory" class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
    <property name="dataSource" ref="myDataSource" />
    <property name="packagesToScan" value="com.telusko.springmvc.model" />
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
            <prop key="hibernate.show_sql">true</prop>
        </props>
    </property>
</bean>
```

- Em seguida, criaremos uma query na session e transformaremos em uma lista para retornar

```
public List<Alien> getAliens() {
    Session session = sessionFactory.getCurrentSession();
    return session.createQuery("from Alien", Alien.class).list();
}
```

- Agora criaremos o método no Controller para buscar todos os Aliens
- Para isso precisamos adicionar um @Component no DAO para injetá-lo no Controller

```
@Component
public class AlienDAO {
```

```

@.Autowired
private SessionFactory sessionFactory;

public List<Alien> getAliens() {
    Session session = sessionFactory.getCurrentSession();
    return session.createQuery("from Alien", Alien.class).list();
}

@Controller
public class HomeController {

    @Autowired
    private AlienDAO alienDAO;

    @RequestMapping("/")
    public String home() {
        return "index";
    }

    @PostMapping(value = "/addAlien")
    public String addAlien(@ModelAttribute Alien alien) {
        return "result";
    }

    @GetMapping("getAliens")
    public String getAliens(Model m) {
        m.addAttribute("result", alienDAO.getAliens().toString());

        return "showAliens";
    }

    @ModelAttribute
    public void modelData(Model m) {
        m.addAttribute("name", "Aliens");
    }
}

```

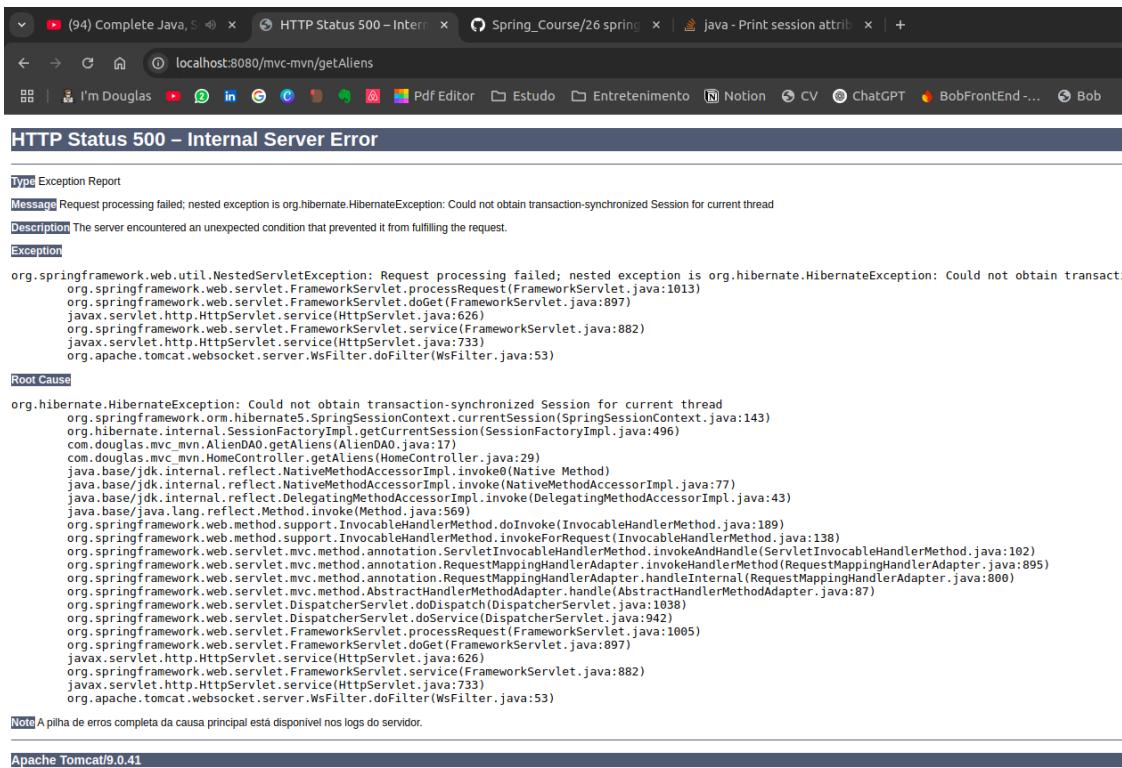
- Temos ainda que criar o jsp showAliens

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1" isELIgnored="false"%>
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>Show Aliens</title>
</head>
<body>
${result}
</body>
</html>

```

- Rodando o código...



The screenshot shows a browser window with the URL `localhost:8080/mvc-mvn/getAliens`. The page title is "HTTP Status 500 – Internal Server Error". The error message is as follows:

Type: Exception Report
Message: Request processing failed; nested exception is org.hibernate.HibernateException: Could not obtain transaction-synchronized Session for current thread
Description: The server encountered an unexpected condition that prevented it from fulfilling the request.
Exception:

```

org.springframework.web.util.NestedServletException: Request processing failed; nested exception is org.hibernate.HibernateException: Could not obtain transaction-synchronized Session for current thread
    org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:1013)
    org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:897)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:626)
    org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:882)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:733)
    org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:53)

```

Root Cause

```

org.hibernate.HibernateException: Could not obtain transaction-synchronized Session for current thread
    org.springframework.orm.hibernate5.SpringSessionContext.currentSession(SpringSessionContext.java:143)
    org.hibernate.internal.SessionFactoryImpl.getCurrentSession(SessionFactoryImpl.java:496)
    com.douglas.mvc.mvn.AlienDAO.getAliens(AlienDAO.java:17)
    com.douglas.mvc.mvn.HomeController.getAliens(HomeController.java:29)
    java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:77)
    java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    java.base/java.lang.reflect.Method.invoke(Method.java:569)
    org.springframework.web.method.support.InvocableHandlerMethod.doInvoke(InvocableHandlerMethod.java:189)
    org.springframework.web.method.support.InvocableHandlerMethod.invokeForRequest(InvocableHandlerMethod.java:138)
    org.springframework.web.servlet.mvc.method.annotation.ServletInvocableHandlerMethod.invokeAndHandle(ServletInvocableHandlerMethod.java:102)
    org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.invokeHandlerMethod(RequestMappingHandlerAdapter.java:895)
    org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.handleInternal(RequestMappingHandlerAdapter.java:800)
    org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.java:1038)
    org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.java:942)
    org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:1005)
    org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:897)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:626)
    org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:882)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:733)
    org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:53)

```

Note: A pilha de erros completa da causa principal está disponível nos logs do servidor.

Apache Tomcat/9.0.41



Pelo menos é o mesmo erro dele...

- O problema é que tentamos fazer direto com a sessão, mas precisamos também lidar com as transações

- Normalmente teríamos que iniciar e commitar a transaction, mas o SpringTx resolve isso para nós

```
package com.douglas.mvc_mvn;

import java.util.List;

import javax.transaction.Transactional;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class AlienDAO {

    @Autowired
    private SessionFactory sessionFactory;

    @Transactional
    public List<Alien> getAliens() {
        Session session = sessionFactory.getCurrentSession();
        return session.createQuery("from Alien", Alien.class).list();
    }
}
```

```

HTTP Status 500 – Internal Server Error

Type Exception Report
Message Request processing failed; nested exception is java.lang.IllegalArgumentException: org.hibernate.hql.internal.ast.QuerySyntaxException: Alien is not mapped [from Alien]
Description The server encountered an unexpected condition that prevented it from fulfilling the request.
Exception
org.springframework.web.util.NestedServletException: Request processing failed; nested exception is java.lang.IllegalArgumentException: org.hibernate.hql.internal.ast.QuerySyntaxException: Alien is not mapped [from Alien]
	org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:1013)
	org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:897)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:626)
	org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:882)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:733)
	org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:53)

Root Cause
java.lang.IllegalArgumentException: org.hibernate.hql.internal.ast.QuerySyntaxException: Alien is not mapped [from Alien]
	org.hibernate.internal.ExceptionConverterImpl.convert(ExceptionConverterImpl.java:138)
	org.hibernate.internal.ExceptionConverterImpl.convert(ExceptionConverterImpl.java:181)
	org.hibernate.internal.ExceptionConverterImpl.convert(ExceptionConverterImpl.java:188)
	org.hibernate.internal.AbstractSharedSessionContract.createQuery(AbstractSharedSessionContract.java:713)
	org.hibernate.internal.AbstractSharedSessionContract.createQuery(AbstractSharedSessionContract.java:729)
	org.hibernate.internal.AbstractSharedSessionContract.createQuery(AbstractSharedSessionContract.java:103)
	com.douglas.mvc.mvn.AlienDAO.getAliens(AlienDAO.java:21)
	com.douglas.mvc.mvn.AlienDAO$fastClassBySpringGLIB$5fa45993.invoke(<generated>)
	org.springframework.cglib.proxy.MethodProxy.invoke(MethodProxy.java:218)
	org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.invokeJoinpoint(CglibAopProxy.java:749)
	org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:163)
	org.springframework.transaction.interceptor.TransactionAspectSupport.invokeWithinTransaction(TransactionAspectSupport.java:294)
	org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java:98)
	org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:186)
	org.springframework.aop.framework.CglibAopProxy$DynamicAdvisedInterceptor.intercept(CglibAopProxy.java:688)
	com.douglas.mvc.mvn.AlienDAO$enhancerBySpringGLIB$5c64e773.getAliens(<generated>)
	com.douglas.mvc.mvn.HomeController.getAliens(HomeController.java:29)
	java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
	java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:77)
	java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
	java.base/java.lang.reflect.Method.invoke(Method.java:569)
	org.springframework.web.method.support.InvocableHandlerMethod.doInvoke(InvocableHandlerMethod.java:189)
	org.springframework.web.method.support.InvocableHandlerMethod.invokeForRequest(InvocableHandlerMethod.java:138)
	org.springframework.web.servlet.mvc.method.annotation.ServletInvocableHandlerMethod.invokeAndHandle(ServletInvocableHandlerMethod.java:102)
	org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.invokeHandlerMethod(RequestMappingHandlerAdapter.java:895)
	org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.handleInternal(RequestMappingHandlerAdapter.java:800)
	org.springframework.web.servlet.mvc.method.AbstractHandlerMethodAdapter.handle(AbstractHandlerMethodAdapter.java:87)

```

- O código roda com outro erro agora: Alien is not mapped
- Isso ocorre porque o Alien não está mapeado no código como uma Entity e com o nome da Table ajustado

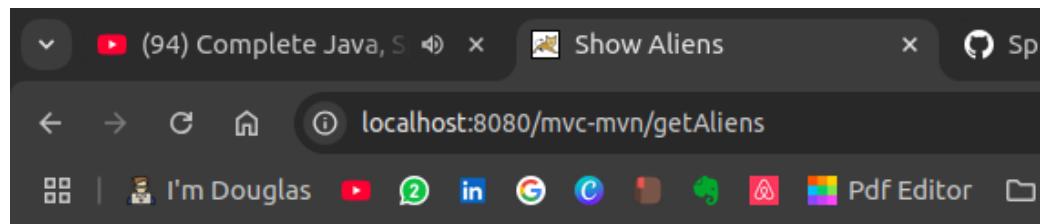


O Telusko não teve que adicionar o Table

```

@Data
@Entity
@Table(name = "alien")
public class Alien {
    @Id
    private int id;
    private String name;
}

```



- Conseguimos visualizar a query no log do console

```
2023-05-30 10:54:40.924 INFORMAÇÕES [http-nio-8080-exec-8] org.springframework.web.servlet.DispatcherServlet: select alien0_.id as id1_0_, alien0_.name as name2_0_ from alien alien0_
```

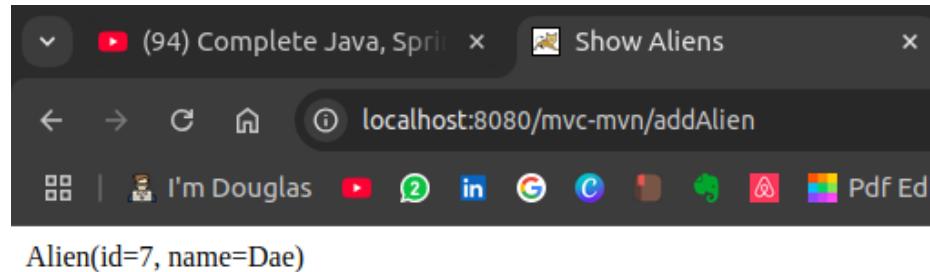
Add and Fetch

- Agora queremos adicionar e buscar um Alien ao banco de dados
- Primeiro vamos adicionar um Alien no banco, usando o método addAlien do Controller

```
@PostMapping(value = "/addAlien")
public String addAlien(@ModelAttribute Alien alien) {
    return "result";
}
```

- A primeira alteração será utilizar o mesmo jsp, showAliens, já que ele apenas joga o "result" na tela

```
@PostMapping(value = "/addAlien")
public String addAlien(@ModelAttribute("result") Alien alien) {
    return "showAliens";
}
```



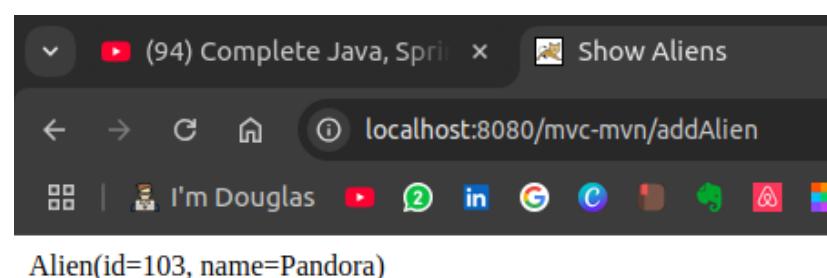
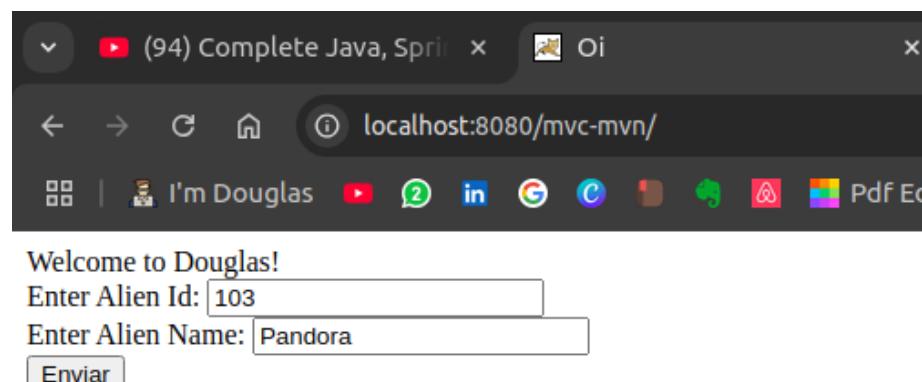
- Agora teremos que criar o método de addAlien no DAO

```
@Transactional  
public void addAlien(Alien alien){  
    Session session = sessionFactory.getCurrentSession();  
    session.save(alien);  
}
```

- E chamá-lo no Controller

```
@PostMapping(value = "/addAlien")  
public String addAlien(@ModelAttribute("result") Alien alien) {  
    alienDAO.addAlien(alien);  
  
    return "showAliens";  
}
```

- O código roda, vamos criar um Alien novo



```
20 Mai 2023 10:01:58.127 ADVERTENCIA [http-nio-8080-exec-  
Hibernate: insert into alien (name, id) values (?, ?)
```

		id	name
		int	varchar(30)
	>	101	Douglas
	>	102	Tayla
	>	103	Pandora

- Funcionou!
- Agora vamos tentar buscar o Alien específico
- Adicionaremos no index.jsp um segundo form para buscar o alien ao invés de criá-lo

```

<body>
    Welcome to Douglas!

    <form action="addAlien" method="post">
        Enter Alien Id: <input type="text" name="id" /><br />
        Enter Alien Name: <input type="text" name="name" /><br />
        <input type="submit" />
    </form>

    <hr />

    <form action="getAlien" method="get">
        Enter existing Alien Id: <input type="text" name="id" /><br />
        <input type="submit" />
    </form>
</body>

```

- Criaremos um novo método getAlien no Controller que receberá o Id

```

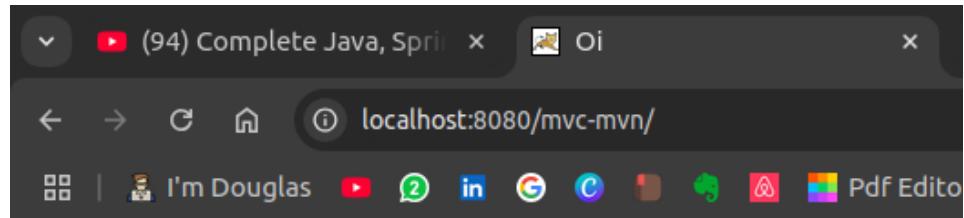
@GetMapping("getAlien")
public String getAlien(@RequestParam int id, Model m) {
    m.addAttribute("result", alienDAO.getAlien(id));

    return "showAliens";
}

```

- O código está com erro pois não há ainda o getAlien no DAO, vamos criar

```
@Transactional  
public Alien getAlien(int id) {  
    Session session = sessionFactory.getCurrentSession();  
    return session.get(Alien.class, id);  
}
```

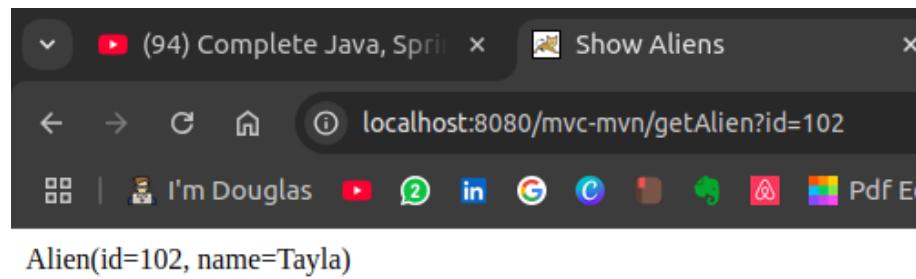


Welcome to Douglas!

Enter Alien Id:

Enter Alien Name:

Enter existing Alien Id:



▼ Spring Data JPA

- Até agora conectamos o Spring no banco com Spring ORM, mas ainda tivemos que adicionar várias dependências de forma manual
- Além disso tivemos que escrever muitas configurações
- A ideia do Spring Boot é cuidar da configuração por nós, então vamos refazer o que fizemos anteriormente com Spring Boot inicialmente, colocando as configurações essenciais no properties
- Usaremos a aplicação que estavamos utilizando com Spring Boot MVC
- Nela já temos o getAliens retornando uma lista estática, mas vamos fazer isso agora com Spring Boot
- Precisaremos adicionar ainda algumas dependências MySQL e Spring Data, que facilitará muito o acesso ao banco

```
<!-- https://mvnrepository.com/artifact/com.mysql/mysql-connector-j -->
<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <version>8.0.33</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
    <version>2.1.2.RELEASE</version>
</dependency>
```

- A diferença entre Spring com e sem Boot é que, sem Boot, nós mesmos fazemos a configuração, então se não fizermos nada acontece
- No Boot, apenas de existir o Spring Data, ele assume que precisa haver um banco de dados. E como não criamos, dá erro ao rodar:

```
Error starting ApplicationContext. To display the condition evaluation report re-r
2025-03-26T10:47:59.820-03:00 ERROR 890353 --- [mvc-boot] [ restartedMa
```

```
*****
APPLICATION FAILED TO START
*****
```

Description:

Failed to configure a DataSource: 'url' attribute is not specified and no embedde

Reason: Failed to determine a suitable driver class

Action:

Consider the following:

If you want an embedded database (H2, HSQL or Derby), please put it on the classpath. If you have database settings to be loaded from a particular profile you may

- Temos então que adicionar algumas poucas configurações no application.properties

```
spring.application.name=mvc-boot
spring.mvc.view.prefix=/views/
spring.mvc.view.suffix=.jsp

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

spring.datasource.url=jdbc:mysql://localhost:3306/mysqlteste
spring.datasource.username=douglas
spring.datasource.password=123
```



O dialect foi diferente do Telusko por causa da diferença na versão do Hibernate por causa da diferença na versão do Spring

- Rodando agora, o código roda normalmente, mas ainda não acessa o banco
- Vamos criar, novamente, o DAO, e ajustar o Alien para ser uma Entity
- Na verdade, no SpringBoot, ao invés de uma classe DAO, criaremos uma interface Repository, que herdará da JpaRepository, que tem template da classe e o Id

```
package com.douglas.mvc_boot;

import org.springframework.data.jpa.repository.JpaRepository;

public interface AlienRepository extends JpaRepository<Alien, Integer> {
```

- Devido ao DevTools, o código já atualiza e acusa um erro, porque o Alien não está mapeado, mas resolveremos isso depois. Além disso, ainda não nos preocuparemos em criar a classe que implementará essa interface
- Vamos instanciar a interface com autowired no Controller

```
@Controller
public class HomeController {

    @Autowired
    AlienRepository repository;
```

- E fazer o mapeamento do Alien

```
@Data
@AllArgsConstructor
@Entity
public class Alien {
    @Id
    private int id;
    private String name;
}
```

- Agora o código não apresenta mais nenhuma mensagem de erro
- Em seguida, vamos buscar os dados do banco no Controller
- Começaremos pela getAliens, removendo a lista estática e buscando no repositório

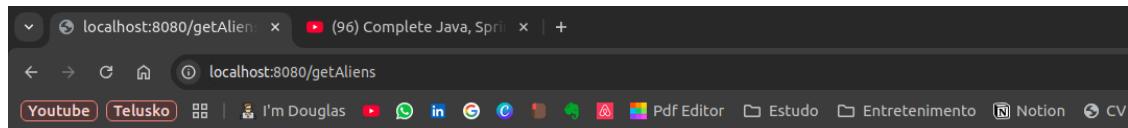
```
@GetMapping("/getAliens")
public String getAlien(Model m) {
    m.addAttribute("result", repository.findAll());
```

```

        return "showAliens";
    }

```

- O método findAll não foi definido por nós, mas no JpaRepository
- Se rodarmos o código:



Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

```

Wed Mar 26 18:21:54 BRT 2025
There was an unexpected error (type=Internal Server Error, status=500).
No default constructor for entity 'com.douglas.mvc_boot.Alien'
org.springframework.orm.jpa.JpaSystemException: No default constructor for entity 'com.douglas.mvc_boot.Alien'
at org.springframework.orm.jpa.vendor.HibernateJpaDialect.convertHibernateAccessException(HibernateJpaDialect.java:341)
at org.springframework.orm.jpa.vendor.HibernateJpaDialect.translateExceptionIfPossible(HibernateJpaDialect.java:241)

```

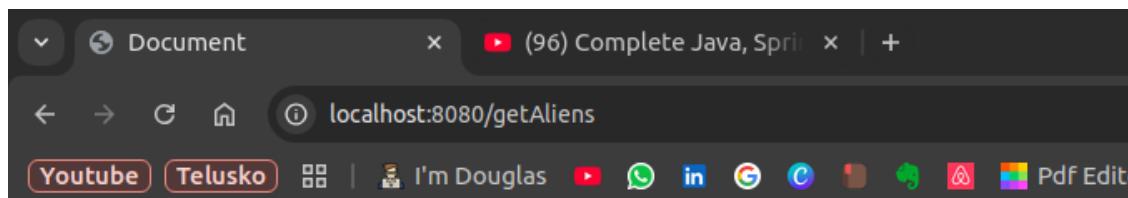
- Pois o SpringJPA precisa que exista um construtor default (ou NoArgs)

```

@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
public class Alien {
    @Id
    private int id;
    private String name;
}

```

- Feito esse ajuste, o código roda



- O estranho é que estamos utilizando uma interface como Repository. Interface não implementa método, não é criada, não pode executar nada. Mas, o código funciona.

- Essa é a mágica do Spring Data JPA, ele acha a interface e cria os Beans necessários que implementam o CRUD com a classe que colocamos no Generic
- No final das contas, escrevemos infinitamente menos código, e menos código significa menos bugs e menos erros

GetAlien e AddAlien

- Vamos implementar agora o getAlien e addAlien com Spring JPA

```

@PostMapping("/addAlien")
public String addAlien(@ModelAttribute("result") Alien alien) {
    repository.save(alien);

    return "showAliens";
}

@GetMapping("/getAlien")
public String getAlien(@RequestParam int id, Model m) {

    m.addAttribute("result", repository.findById(id));

    return "showAliens";
}

```

- Sim... simples assim. E funciona. Precisamos apenas ajustar o jsp:

```

<body>
    Welcome to Douglas!

    <form action="addAlien" method="post">
        Enter Alien Id: <input type="text" name="id" /><br />
        Enter Alien Name: <input type="text" name="name" /><br />
        <input type="submit" />
    </form>

    <hr />

    <form action="getAlien" method="get">
        Enter Alien Id: <input type="text" name="id" /><br />
        <input type="submit" />
    </form>

```

```
</form>  
</body>
```

Welcome to Douglas!

Enter Alien Id:

Enter Alien Name:

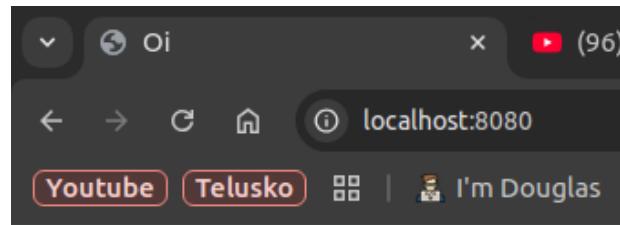
Enter Alien Id:

Alien(id=104, name=Shiva)

alien

Search Results

		id	name
		int	varchar(30)
	>	101	Douglas
	>	102	Tayla
	>	103	Pandora
	>	104	Shiva



Welcome to Douglas!

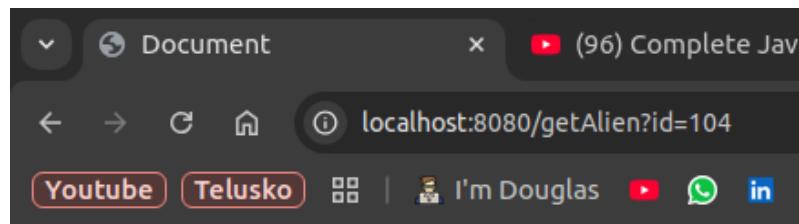
Enter Alien Id:

Enter Alien Name:

Enviar

Enter Alien Id:

Enviar



Optional[Alien(id=104, name=Shiva)]

Query DSL

- Podemos também querer buscar um dado por outra coluna além do id, como no nosso caso o nome
- Vamos começar a fazer isso partindo do Controller

```
@GetMapping("getAlienByName")
public String getAlienByName(@RequestParam String name, Model m) {
    m.addAttribute("result", repository.findByName(name)) ;

    return "showAliens";
}
```

- Temos um erro, pois o método não existe, mas podemos definir esse método na interface

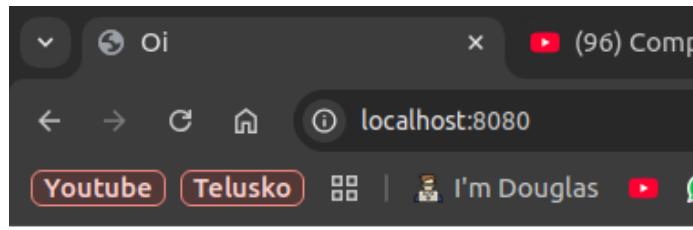
```
public interface AlienRepository extends JpaRepository<Alien, Integer> {
```

```
    List<Alien> findByName(String name);  
}
```

- Precisamos que seja uma lista pois, como não é chave primária, podemos ter mais de um alien com o mesmo nome
- Agora precisamos apenas criar um form para o nome

```
<hr />
```

```
<form action="getAlienByName" method="get">  
    Enter Alien Name: <input type="text" name="name" /><br />  
    <input type="submit" />  
</form>  
</body>
```



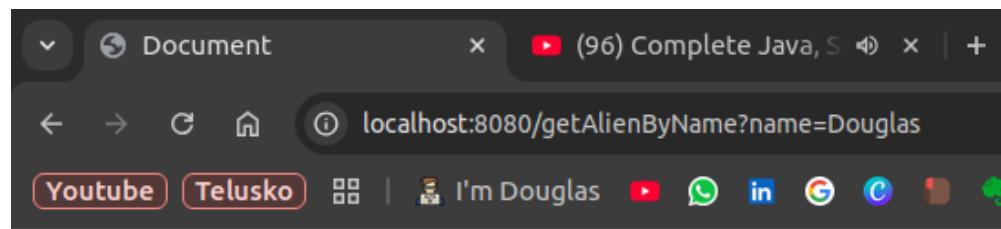
Welcome to Douglas!

Enter Alien Id:
Enter Alien Name:

Enter Alien Id:

Enter Alien Name:

Douglas



[Alien(id=101, name=Douglas)]

- Novamente, não tivemos que criar o método, nem implementá-lo, mesmo não sendo um dos métodos básicos do JpaRepository e utilizando um

atributo específico

- Isso acontece porque seguimos a DSL - Domain Specific Language, e o próprio Spring JPA, quando criou o Bean que implementa essa interface, implementou esse método
- Isso são as Query DSL
- Naturalmente, elas não são mágicas, mas seguem algumas regras, entre elas a necessidade de usar Camel Case, de usar os nomes exatamente como estão na classe, etc
- É possível, contudo, fazer muitas coisas com DSL, inclusive AND e OR

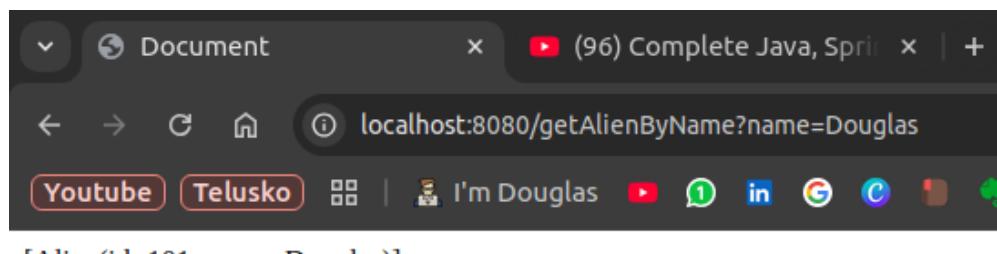
Query Annotation

- Query DSL são muito poderosas, mas não são onipotentes
- Em alguns momentos, podemos querer buscar dados com uma query "normal"
- Criaremos um novo método para buscar pelo nome, ao invés de usar o com DSL, mas anotaremos com a Annotation Query e a Query desejada

```
@Query("from Alien where name= :alienName")
List<Alien> findUsingTheName(@Param("alienName") String name);
```

```
@GetMapping("getAlienByName")
public String getAlienByName(@RequestParam String name, Model m) {
    m.addAttribute("result", repository.findUsingTheName(name));

    return "showAliens";
}
```



- Ainda é possível, nessa query com HQL utilizar algo como Like
-

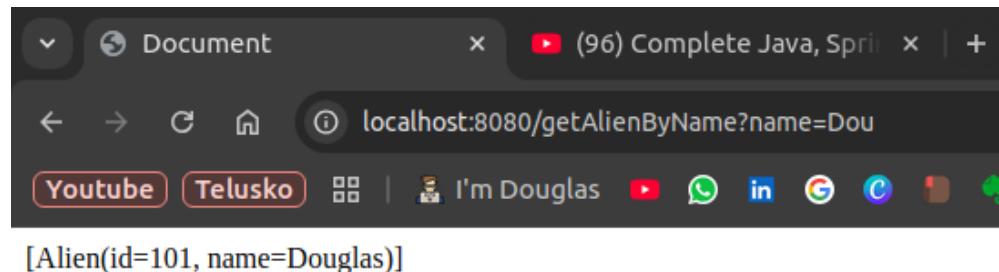
```

public interface AlienRepository extends JpaRepository<Alien, Integer> {

    // List<Alien> findByName(String name);

    @Query("from Alien where name like %:alienName%")
    List<Alien> findUsingTheName(@Param("alienName") String name);
}

```



▼ REST API with Spring

REST

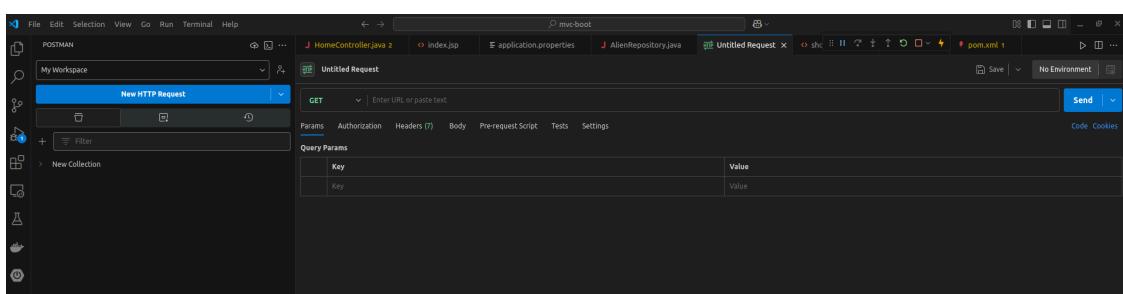
- No conceito de Client e Server, são duas entidades diferentes, o Client consome dados e o Server gera ou retorna dados
- No começo, o Client era o ser humano que sempre queria ver dados bonitos, como HTML
- Atualmente, o Cliente também costuma ser uma máquina, então não precisa que os dados sejam bonitos, pelo contrário, quer de fácil acesso como JSON ou XML
- Agora também teremos mais opções de actions, ou endpoints, e por isso usamos "nomes" nos endpoints (e.g. www.rest.com/alien ao invés de www.rest.com/getAlien)
- Nisso temos **REST Web Services ou Web APIs**, mas o REST também aplica mais algumas restrições
- **Stateless:** requisições REST são Stateless, ou seja, o Servidor não analisa se aquele usuário chamou o serviço instantes antes. Se for necessário ter esse

tipo de informação, teremos que usar um Token dentro da requisição

- **VERBS:** como dito, os endpoints não serão verbos, e sim nomes, diferenciaremos as operações com os verbos que são os métodos: **GET, POST, PUT, DELETE**

Postman

- Até agora, comunicamos com o servidor pelo navegador, porque sempre retornamos telas
- Uma forma mais visualmente eficiente de fazer essa comunicação é com um API Development Environment, como Postman
- Já que eu uso Linux, usarei a extensão do VSCode, já está funcionando



GetMapping

- Usaremos o mesmo projeto de antes, com alguns ajustes
- Vamos começar buscando todos os aliens
- Criaremos um AlienController para fazer tudo "do começo"

```
@Controller  
public class AlienController {  
    @Autowired  
    AlienRepository alienRepository;  
  
}
```

- Ao criar o getMapping, não usaremos mais verbos, e sim apenas o nome "aliens". Além disso, não retornaremos mais um JSP

```
@GetMapping("aliens")  
public String getAliens() {  
    return alienRepository.findAll().toString();  
}
```

- Se rodarmos a aplicação e executarmos o get no Postman:

```

GET http://localhost:8080/aliens
{
  "timestamp": "2025-03-26T22:20:56.302+00:00",
  "status": 404,
  "error": "Not Found",
  "message": "JSP file [/views/[Alien(id=101, name=Douglas), Alien(id=102, name=Tayla), Alien(id=103, name=Pandora), Alien(id=104, name=Shiva)].jsp] not found",
  "path": "/aliens"
}
  
```

- Até conseguimos ver a lista, mas com uma mensagem de erro
- Isso ocorre porque o MVC entende por padrão que o retorno de uma chamada API é o caminho de um jsp, isso além de termos configurado o prefix e o suffix para Strings conforme isso
- Precisamos adicionar uma annotation ResponseBody no método

```

@GetMapping("aliens")
@ResponseBody
public String getAliens() {
    return alienRepository.findAll().toString();
}
  
```

```

GET http://localhost:8080/aliens
[
  {"id": 101, "name": "Douglas"}, {"id": 102, "name": "Tayla"}, {"id": 103, "name": "Pandora"}, {"id": 104, "name": "Shiva"}
]
  
```

- O problema é que queríamos um JSON, não uma String
- Para isso, basta mudar o tipo de retorno do método

```

@GetMapping("aliens")
@ResponseBody
public List<Alien> getAliens() {
    return alienRepository.findAll();
}
  
```

```

GET      | http://localhost:8080/aliens
Params   Authorization Headers (7) Body
Body    Cookies Headers (5) Test Results
Pretty Raw Preview JSON ▾
1  [
2   {
3     "id": 101,
4     "name": "Douglas"
5   },
6   {
7     "id": 102,
8     "name": "Tayla"
9   },
10  {
11    "id": 103,

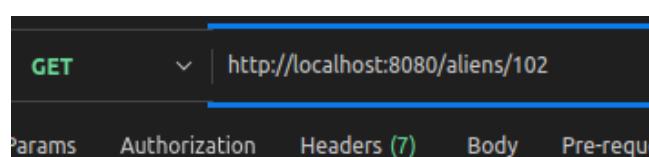
```

Jackson

- Não estamos fazendo nenhuma conversão direta dos objetos para o JSON
- O Spring faz isso de forma “automática” utilizando Jackson
- Isso também permite que o Spring receba um objeto Java e transforme em JSON ou o contrário

Get 1

- Agora vamos buscar um único dado pelo id
- O padrão, na URI, como fizemos antes, é ter o ID na URI
- Podemos ter como queryParam antes, mas podemos colocar depois da barra, assim:



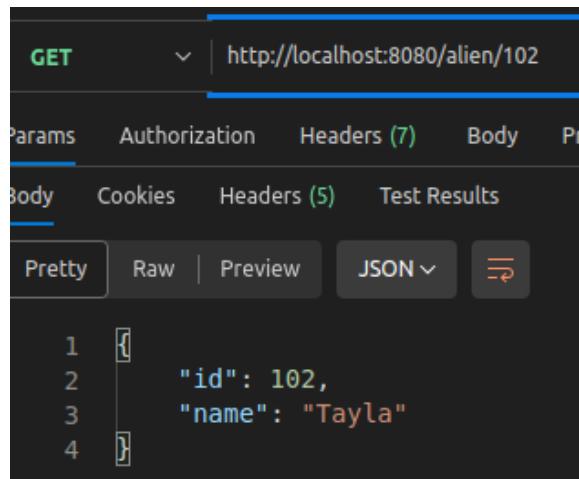
- Se rodarmos não funciona pois temos que mapear esse método, e usaremos o parâmetro no Mapping e com a annotation PathVariable

```

@.GetMapping("alien/{id}")
@ResponseBody
public Alien getAlien(@PathVariable("id") int id) {

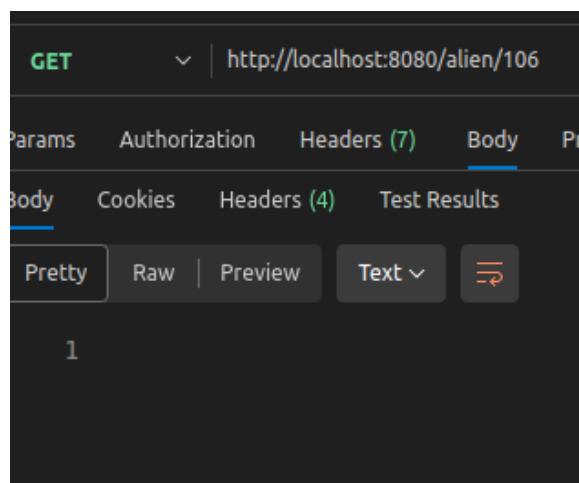
```

```
        return alienRepository.findById(id).orElse(null);  
    }
```



A screenshot of the Postman application interface. The top bar shows a GET request to `http://localhost:8080/alien/102`. The 'Body' tab is selected, and the 'Pretty' option is chosen. The response body is displayed as:

```
1  [  
2      "id": 102,  
3      "name": "Tayla"  
4 ]
```



A screenshot of the Postman application interface. The top bar shows a GET request to `http://localhost:8080/alien/106`. The 'Body' tab is selected, and the 'Text' option is chosen. The response body is displayed as:

```
1
```

Post

- Como faremos para adicionar um dado? Sem ter um form com todos os dados
- Vamos mudar o verbo de GET para POST no Postman
- Além disso, usaremos o mesmo nome “alien”, mas com um verbo diferente
- Finalmente, os dados serão inseridos no Body, ao invés da URL

The screenshot shows the Postman interface with a POST request to `http://localhost:8080/alien/`. The 'Body' tab is selected, showing 'form-data' selected. The table below contains two fields: 'id' with value '105' and 'name' with value 'Tiquito'. Other tabs like 'Params', 'Headers', and 'Tests' are visible at the top.

Key	Value
id	105
name	Tiquito
Key	Value

- Rodando vemos um erro pois não existe esse endpoint no código. Vamos criá-lo

```
@PostMapping("alien")
@ResponseBody
public Alien addAlien(Alien alien) {
    alienRepository.save(alien);

    return alien;
}
```



Por algum motivo, no nosso não funcionou sem o `ResponseBody`, no dele sim

The screenshot shows the Postman interface with a POST request to `localhost:8080/alien`. The 'Body' tab is selected, showing 'form-data' selected. The table below contains two fields: 'id' with value '105' and 'name' with value 'Tiquito'. The 'Pretty' tab is selected in the bottom navigation bar, and the JSON response is displayed as:

```
1 ↴ []
2   "id": 105,
3   "name": "Tiquito"
4 ]
```

		id	name
		int	varchar(30)
	>	101	Douglas
	>	102	Tayla
	>	103	Pandora
	>	104	Shiva
	>	105	Tiquito

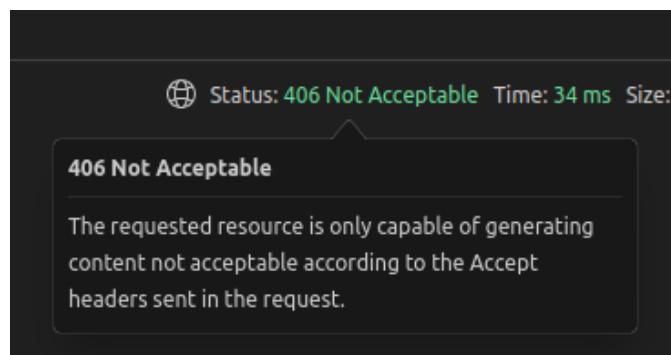
XML

- Podemos, através do Header, exigir um retorno como XML

The screenshot shows the Postman interface with the 'Headers' tab selected. There is one header entry: 'Accept' with the value 'application/xml'. Other tabs like 'Params', 'Authorization', 'Body', and 'Tests' are visible at the top.

Key	Value
Accept	application/xml
Key	Value

- Se rodarmos algum método como, por exemplo, o aliens, recebemos vazio e um erro de Not Acceptable



- Isso ocorre porque, por padrão, o Jackson lida apenas com JSON, e não com XML. Para isso precisaremos adicionar o Jackson XML

Jackson XML

- Content Negotiation é quando o cliente negocia com o servidor qual o formato do dado que virá
- Tentamos fazer isso no Postman mas o código não estava preparado

- Incluiremos para isso a dependencia do Jackson XML

```
<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.dataformat/jacks
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.18.2</version>
</dependency>
```

- Agora podemos tentar o request novamente e funciona

The screenshot shows a Postman request configuration for a GET request to `localhost:8080/aliens`. The `Headers` tab is selected, containing the entry `Accept: application/xml`. The `Body` tab is selected, showing a `Pretty` representation of the XML response. The response body is:

```

1  <List>
2   <item>
3     <id>101</id>
4     <name>Douglas</name>
5   </item>
6   <item>
7     <id>102</id>
8     <name>Tayla</name>
9   </item>
10  <item>
11    <id>103</id>
12    <name>Pandora</name>
13  </item>
14  <item>
15    <id>104</id>
16    <name>Shiva</name>
17  </item>
18  <item>

```

- É possível ainda restringir, do lado do servidor, o formato de dado que receberemos, ou ainda o que aceitamos receber
- Para responder apenas XML:

```
@GetMapping(path = "aliens", produces = { "application/xml" })
@ResponseBody
public List<Alien> getAliens() {
    return alienRepository.findAll();
}
```

```

GET | localhost:8080/aliens
Params Authorization Headers (7) Body Pre-request Script Tests Settings
Headers (7 hidden)
Key Value
Key Value

Body Cookies Headers (5) Test Results Status: 200 OK
Pretty Raw Preview XML ↗

1 <List>
2   <item>
3     <id>101</id>
4     <name>Douglas</name>
5   </item>
6   <item>
7     <id>102</id>
8     <name>Tayla</name>
9   </item>
10  <item>
11    <id>103</id>
12    <name>Bender</name>
13  </item>

PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
(set 'hibernate.transaction.jta.platform' to enable JTA platform integration)
2025-03-27T14:45:16.797-03:00 INFO 2403209 --- [mvc-boot] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManager for persistence unit 'default'
2025-03-27T14:45:16.989-03:00 INFO 2403209 --- [mvc-boot] o.s.d.j.r.query.QueryEnhancerFactory : Hibernate is in classpath; HQL parser will be used.

```

- Isso porque não exigimos nenhum tipo de dados
- Podemos tentar exigir um json:

```

GET | localhost:8080/aliens
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Headers (7 hidden)
Key Value
Accept application/json
Key Value

Body Cookies Headers (6) Test Results Status: 406 Not Acceptable
Pretty Raw Preview JSON ↗

```

- Voltamos a receber 406
- É possível determinar também o que podemos receber em casos de POST
- Primeiro tentaremos enviar um XML para o servidor
- O primeiro passo é incluir o campo Content-Type nos Headers como XML

```

POST | localhost:8080/aliens
Params Authorization Headers (11) Body Pre-request Script Tests Settings
Headers (9 hidden)
Key Value
Accept application/xml
Content-Type application/xml
Key Value

Body Cookies Headers (6) Test Results Status: 406 Not Acceptable
Pretty Raw Preview JSON ↗

```

- Em seguida preencheremos os dados, mas no modo raw ao invés de form-data. E inicialmente como um Json

POST | localhost:8080/aliens

Params | Authorization | Headers (11) | **Body** | Pre-request Script | Tests | Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON**

```

1 {
2   "id": "106",
3   "name": "Proximo"
4 }
```

Body | Cookies | Headers (5) | Test Results

Pretty | Raw | Preview | XML |

```

1 <Alien>
2   <id>0</id>
3   <name/>
4 </Alien>
```

- Ocorreu um erro na hora de criar o recurso no banco
- Isso acontece porque, quando enviamos como form, o Postman é capaz de discretizar melhor que aquele é o Alien, mas quando mandamos o dado precisamos deixar mais claro para o Java que aquilo virá no Request Body:

```

@PostMapping("/alien")
@ResponseBody
public Alien addAlien(@RequestBody Alien alien) {
    alienRepository.save(alien);

    return alien;
}
```

- Contudo, se rodarmos agora, recebemos um erro de formatação, porque definimos o Header como XML e enviamos um JSON. Se alterarmos o Content-type para JSON, funciona:

The screenshot shows the Postman interface. In the 'Headers' section, there is a table with one row containing 'Content-Type' and 'application/json'. Below the headers, the 'Body' tab is selected, showing a raw XML payload:

```

1 <Alien>
2   <id>106</id>
3   <name>Proximo</name>
4 </Alien>

```

- Mas o objetivo é impedir o envio no formato JSON, faremos isso com o atributo consumes
- O atributo pode receber uma lista de String, assim como o produces, mas também pode receber Enums ao invés de String

```

@PostMapping(path = "/alien", consumes = { MediaType.APPLICATION_XML_VALUE })
@ResponseBody
public Alien addAlien(@RequestBody Alien alien) {
    alienRepository.save(alien);

    return alien;
}

```

- Se agora tentarmos enviar um JSON:

The screenshot shows the Postman interface with a JSON request body in the 'Body' tab:

```

1 [
2   ...
3   ...
4 ]

```

The response status is 415 Unsupported Media Type. The response body is an XML error message:

```

1 <Map>
2   <timestamp>2025-03-27T18:06:13.273+00:00</timestamp>
3   <status>415</status>
4   <error>Unsupported Media Type</error>
5   <trace>org.springframework.web.HttpMediaTypeNotSupportedException: Content-Type 'application/json' is not supported
6       at org.springframework.web.servlet.mvc.method.RequestMappingInfoHandlerMapping.handleNoMatch(RequestMappingInfoHandlerMapping.java:282)
7       at org.springframework.web.servlet.handler.AbstractHandlerMethodMapping.lookupHandlerMethod(AbstractHandlerMethodMapping.java:441)

```

- Mas podemos alterar para enviar um XML

Params Authorization Headers (11) Body Pre-request Script Tests Settings

Headers @ 9 hidden

	Key	Value
<input checked="" type="checkbox"/>	Accept	application/xml
<input checked="" type="checkbox"/>	Content-Type	application/xml
	Key	Value

POST localhost:8080/alien

Params Authorization Headers (11) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL XML

```
1 <Alien>
2 ...<id>107</id>
3 ...<name>Outro</name>
4 </Alien>
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview XML

```
1 <Alien>
2     <id>107</id>
3     <name>Outro</name>
4 </Alien>
```

Status: 200 OK

- E tudo funciona!

▼ Java Project

- Finalmente criaremos um projeto completo

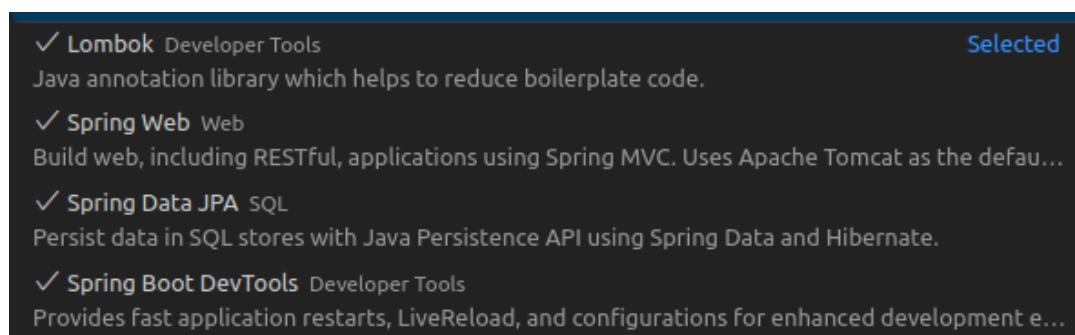
Definição

- A ideia será gerenciar produtos
 - A ideia é que, no escritório do Telusko, há alguns itens que ele usa com frequência, mas outros que usa com muito pouca frequência
 - Criaremos uma aplicação de console para rastrear produtos, para saber se tem e quantos tem
 - (quase um sistema de estoque)
 - Quando pensamos em POO ou em bancos relacionais, precisamos desenhar nosso software com base em Entidades (Objetos, Tabelas) e Relacionamentos

- Classes (Entidades, Objetos, Tabelas) tem Comportamentos e Atributos
- A classe base desse software é o **Product**
- E os atributos serão
 - name
 - type
 - place
 - warranty

Prática

- Começaremos criando um projeto Spring chamado ProductManager



Aqui percebi que o Telusko na verdade está retomando todo o conceito com esse projeto, ao invés de avançar e expandir.

Ele na verdade vai começar com um projeto Java que só tem a Main e vai salvando os dados numa lista

Depois ele vai incluir o banco de dados (não me pergunte como) e vai salvar lá

Por último ele vai criar o projeto com Spring MVC

Me parece ótimo se eu não tivesse gasto os últimos meses com o resto do curso, vou pular pro próximo

▼ Spring Boot + MongoDB + React Project

- Agora sim!
- Criaremos um projeto com Spring Boot, MongoDB e React, fullstack mesmo
- O React já vai estar pronto (e é bem básico na real)
- O vídeo assume que já sabemos o básico de Spring Boot e Mongo DB, e o que vai acrescentar é mais o fluxo do projeto e Mongo DB Atlas
- Como eu não sei muito Mongo com Java, vai ser muito bom! Bora!

▼ Definição

- A ideia é uma aplicação de contratação

Get Hired or Hire people for free!

[HIRE TALENT](#)

[GET JOB NOW](#)



Subscribe Telusko

[HOME](#)

Search...

developer
Description: Software engineer who can work on enterprise projects using spring boot and mongodb and react

Years of Experience: 1 years
Skills : java . jee . spring . springboot . microservices .

java developer
Description: Senior Software engineer with java skillset expert in core java and concurrency

Years of Experience: 5 years
Skills : java . jee . spring . springboot . microservices . threads .

Architect
Description: Associate consultant architecture of software, Agile, Devops

Senior Architect
Description: Senior Associate consultant for Mumbai Location

[HOME](#)

developer

Mobile Developer
Description: IOS Developer worked with Swift, expert in design

Years of Experience: 7 years
Skills : Java . Swift . React .

Mobile Developer
Description: Android Developer worked with Java and Kotlin, expert in design

Years of Experience: 5 years
Skills : Java . Kotlin . React .

developer
Description: Software engineer who can work on enterprise projects using spring boot and mongodb and react

Years of Experience: 1 years
Skills : java . jee . spring . springboot . microservices .

java developer
Description: Senior Software engineer with java skillset expert in core java and concurrency

Years of Experience: 5 years
Skills : java . jee . spring . springboot . microservices . threads .

[HOME](#)

threads

java developer
Description: Senior Software engineer with java skillset expert in core java and concurrency

Years of Experience: 5 years
Skills : java . jee . spring . springboot . microservices . threads .



Subscribe Telusko

[HOME](#)

mumbai

Senior Architect
Description: Senior Associate consultant for Mumbai Location

Years of Experience: 15 years
Skills : java . jee . spring . springboot . microservices .

Web Developer
Description: Asp.net c# mongodb work in mumbai

Years of Experience: 6 years
Skills : Python . Java .



Subscribe Telusko

- **Search Feature!**

Create New Post

Job-profile*
Blockchain Developer

Years of Experience*
2

Job-desc*
Ethereum developer

Please mention required skills



Job-desc*
Ethereum developers who knows how to make dapps.

Please mention required skills

- Javascript
- Java
- Python
- Django
- Rust

SUBMIT



Por que NoSQL?

- No NoSQL, ao invés de tabelas temos coleções, e ao invés de linhas temos documentos, isso facilita a inclusão de dados não estruturados, de replicação, etc
- É bom também por não precisarmos instalar nada na máquina, porque usaremos o MongoDB na nuvem com Atlas

▼ MongoDB Atlas

- Começaremos criando o banco no MongoDB Atlas
- Podemos acessar o site e logar/registrar
- Então vamos criar um novo projeto

Atlas Douglas's Org... Access Manager Billing

MDB_EDU Data Services Charts

Search for a Project... Address not added. You will not be able to connect to databases from this address.

MDB_EDU Project 0

View All Projects + New Project

Clusters myAtlasClusterEDU Connect Edit configuration

DOUGLAS'S ORG - 2023-11-12 > PROJECTS Create a Project

Name Your Project Add Members

Name Your Project Project names have to be unique within the organization (and other restrictions). HiringProjectTelusko

Add Tags (Optional) Use tags to efficiently label and categorize your projects. A project can have a maximum of 50 tags. You can modify tags for the project later. Learn more

Key	Value	Actions
Select a key or enter your own	Select a value or enter your own	Delete
+ Add tag		0 TAGS

Cancel Next

The screenshot shows the MongoDB Atlas interface. At the top, there's a navigation bar with 'Atlas', 'Douglas's Org...', 'Access Manager', and 'Billing'. Below it, a sidebar has 'MDB_EDU' selected under 'Data Services'. The main area shows a search bar and a list of projects. A modal window is open for creating a new project, titled 'Create a Project'. It has tabs for 'Name Your Project' and 'Add Members', with 'Name Your Project' currently active. A text input field contains 'HiringProjectTelusko'. Below it is a section for 'Add Tags (Optional)' with a table for managing tags. At the bottom right of the modal are 'Cancel' and 'Next' buttons.

- Em seguida criaremos um novo Cluster

Deploy your cluster

Use a template below or set up advanced configuration options. You can also edit these configuration options once the cluster is created.

Template	Price	Description
M10	\$0.12/hour	Dedicated cluster for development environments and low-traffic applications. Storage: 10 GB, RAM: 2 GB, vCPUs: 2 vCPUs
Flex	From \$0.011/hour Up to \$30/month	For application development and testing, with on-demand burst capacity for unpredictable traffic. Storage: 5 GB, RAM: Shared, vCPUs: Shared
Free		For learning and exploring MongoDB in a cloud environment. Storage: 512 MB, RAM: Shared, vCPUs: Shared

Free forever! Your free cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

Configurations

Name
You cannot change the name once the cluster is created.

Provider

AWS Google Cloud Azure

Region

São Paulo (sa-east-1) ★

★ Recommended i Low carbon emissions i

Tag (optional)
Create your first tag to categorize and label your resources; more tags can be added later. [Learn more](#).

Select or enter key : Select or enter value

Quick setup

Automate security setup i

Preload sample dataset i

Connect to telusko

1 Set up connection security 2 Choose a connection method 3 Connect

You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. [Read more](#)

1. Add a connection IP address

Your current IP address (189.26.114.130) has been added to enable local connectivity. Only an IP address you add to your Access List will be able to connect to your project's clusters. Add more later in [Network Access](#).

2. Create a database user

This first user will have [atlasAdmin](#) permissions for this project.

We autogenerated a username and password. You can use this or create your own.

i You'll need your database user's credentials in the next step. Copy the database user password.

Username	Password
douglas	1234 <small>HIDE</small> <input type="button" value="Copy"/>

- Clicamos em Create Database User e, em seguida, ele nos pede para determinar a conexão, mas o Telusko não fez isso ainda então vamos deixar para depois também (só fechar)

- Em seguida temos que criar um banco, não temos ainda, vamos fazer isso via "Add Data"

Overview

The screenshot shows the MongoDB Atlas 'Clusters' overview for the 'telusko' cluster. At the top, there are buttons for 'Connect' and 'Edit configuration'. Below these are two main sections: 'Add data' (with a file icon) and 'Load sample data' (with a database icon). To the right of these are 'Data modeling templates' (with a tree icon) and a '+ Add Tag' button. A green header bar at the top says 'Overview'.

- Agora Create Database

DOUGLAS'S ORG - 2023-11-12 > HIRINGPROJECTTELUSKO
← Back

Add Data Options

The screenshot shows the 'Add Data Options' page with the 'Create Dataset' tab selected. There are four tabs in total: 'Create Dataset' (selected), 'Load Sample Data', 'Migrate Data', and 'Import File'. Below the tabs are three cards: 'Create Database on Atlas' (with a database icon), 'Create via Application' (with a gear icon), and 'Create via MongoShell' (with a terminal icon). Each card has a 'VIEW INSTRUCTIONS' button.

[← Back](#)

Add Data Options

[Create Dataset](#)[Load Sample Data](#)[Migrate Data](#)[Import File](#)[← Create Database on Atlas](#)**Database name**

Enter a name for your new database

telusko

Collection name

Enter a name for your new collection

JobPost

[Learn more about database and collection naming ↗](#)Capped collection [i](#)Time series collection [i](#)

Insert JSON document to Collection (optional)

[SHOW ME AN EXAMPLE](#)

1

[Create Database](#) 

- Não Pode passar um tempinho carregando, se acontecer dá f5

DOUGLAS'S ORG - 2023-11-12 > HIRINGPROJECTTELUSKO > DATABASES

 telusko

VERSION 8.0.5 REGION AWS Sao Paulo (sa-east-1)

Overview	Real Time	Metrics	Collections	Atlas Search	Query Insights	Performance Advisor	Online Archive	Cmd Line Tools	Infrastructure As Code
DATABASES: 1	COLLECTIONS: 1		+ Create Database						
<div style="border: 1px solid #ccc; padding: 5px;"> + Create Database  Search Namespaces telusko JobPost Find Indexes Schema Anti-Patterns Aggregation Search Indexes Filter <input type="text" value="Type a query: { field: 'value' }"/> Reset Apply Options QUERY RESULTS: 0 </div>									

- Ainda não temos nenhum dado, vamos inserir usando INSERT DOCUMENT com um JSON do repo do Telusko

<https://github.com/navinreddy20/SpringMongoDB>

[jsondata.txt](#)

Insert Document

To collection JobPost

VIEW { } ⚙

```
1  [
2  {
3      "_id" : "1",
4      "desc": "Software engineer who can work on enterprise",
5      "exp": 1,
6      "profile": "developer",
7      "techs": [
8          "java","jee","spring","springbot","microservices"
9      ]
10 },
11 {
12     "_id" : "2",
13     "desc": "Senior Software engineer with java skillset",
14     "exp": 5,
15     "profile": "java developer",
16     "techs": [
17         "java","jee","spring","springbot","microservices"
18     ]
19 },
20 {
21     "_id" : "3",
22     "desc": "Associate consultant architecture of software",
23     "exp": 10,
24     "profile": "Architect",
25     "techs": [
26         "java","jee","spring","springbot","microservices"
27     ]
28 },
29 {
30     "_id" : "4",
31     "desc": "Senior Associate consultant for Mumbai Location",
32     "exp": 15,
33     "profile": "Senior Architect",
34     "techs": [
35         "java","jee","spring","springboot","microservices"
36     ]
37 },
38 {
39     "_id" : "5",
40     "desc": "Manager for onsite projects in US and UK",
41     "exp": 15,
42     "profile": "Senior Manager",
43     "techs": [
```

Cancel Insert

- Os dados são formatados como JSON, facilitando a compreensão, visualização e importação no Java

▼ Spring Project

- Para o backend, criaremos um projeto Spring com o Initializr com as seguintes dependencias
 - Lombok
 - Spring DevTools
 - Spring Web
 - Spring Data MongoDB
- Se tentarmos rodar o projeto, ele roda, mas exibe erros relacionados ao Mongo
- Adicionaremos as dependencias do Swagger

```
<!-- https://mvnrepository.com/artifact/io.springfox/springfox-swagger-ui -->
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>

<!-- https://mvnrepository.com/artifact/io.springfox/springfox-swagger2 -->
<dependency>
    <groupId>io.springfox</groupId>
```

```

<artifactId>springfox-swagger2</artifactId>
<version>2.9.2</version>
</dependency>

```

MongoDB Config

- Para que a aplicação comunique com o banco, precisamos configurar o application.properties
- Vamos pegar o uri na GUI do Mongo, em Overview → Connect

DOUGLAS'S ORG - 2023-11-12 > HIRINOPROJECTTELUSKO > DATABASES
telusko

VERSION: 8.0.6 REGION: AWS Sao Paulo (sa-east-1) CLUSTER TIER: M0 Sandbox (General)

Overview Real Time Metrics Collections Atlas Search Query Insights Performance Advisor Online Archive Cmd Line Tools Infrastructure As Code

SANDBOX NODES REPLICAS SET

Connect Configuration ...

TAOS

Use tags to efficiently label and categorize your clusters. Any tags you apply will display here.
Learn more about tagging.

ADD TAG

REGION: Sao Paulo (sa-east-1)

This is a Shared Tier Cluster
If you need a database that's better for high-performance production applications, upgrade to a dedicated cluster.

Operations R: 0 W: 100.0%

- Connect with application (talvez tenha que autorizar o IP, se ainda não tiver feito)

Connect to telusko

1 Set up connection security 2 Choose a connection method 3 Connect

✓ Current IP Address (189.26.114.130/32) added! Visit [Network Access](#) to modify your settings.

Connect to your application

 Drivers Access your Atlas data using MongoDB's native drivers (e.g. Node.js, Go, etc.) >

Access your data through tools

 Compass Explore, modify, and visualize your data with MongoDB's GUI >

 Shell Quickly add & update data using MongoDB's Javascript command-line interface >

 MongoDB for VS Code Work with your data in MongoDB directly from your VS Code environment >

 Atlas SQL Easily connect SQL tools to Atlas for data analysis and visualization >

[Go Back](#) [Close](#)

Connect to telusko

Set up connection security ✓ Choose a connection method ✓ Connect 3

Connecting with MongoDB Driver

1. Select your driver and version

We recommend installing and using the latest driver version.

Driver	Version
Java	4.3 or later

2. Install your driver

[View MongoDB Java Driver installation instructions.](#)

3. Add your connection string into your application code

Use this connection string in your application

View full code sample

```
mongodb+srv://douglas:<db_password>@telusko.0p0eal8.mongodb.net/?retryWrites=true&w=majority&appName=telusko
```

Replace `<db_password>` with the password for the `douglas` database user. Ensure any option params are [URL encoded](#).

RESOURCES

[Get started with the Java Driver](#) [Java Starter Sample App](#) [Access your Database Users](#) [Troubleshoot Connections](#)

[Go Back](#) [Done](#)

- Usaremos a versão 4.3 pois o Telusko também usou
- Em seguida é só copiar a URI
- Além da URI precisaremos também do nome

```
spring.application.name=mongo
spring.data.mongodb.uri=mongodb+srv://douglas:1234@telusko.0p0eal8.m
spring.data.mongodb.database=telusko
```

Swagger API

- Agora vamos habilitar a API do Swagger

- Basta colocar a annotation EnableSwagger2 no Application e criar alguns Beans lá:

```

@SpringBootApplication
@EnableSwagger2
public class MongoApplication {

    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2).select()
            .apis(RequestHandlerSelectors.withClassAnnotation(RestController))
            .useDefaultResponseMessages(false);
    }

    @Bean
    public ApiInfo apiInfo() {
        final ApiInfoBuilder builder = new ApiInfoBuilder();
        return builder.build();
    }

    public static void main(String[] args) {
        SpringApplication.run(MongoApplication.class, args);
    }
}

```

Controller

- Vamos criar um Controller para poder comunicar com o software

```

@RestController
public class JobPostController {
}

```

- O primeiro endpoint que vamos mapear é a raiz, que vamos direcionar para a UI do Swagger

```

@RestController
public class JobPostController {
    @RequestMapping(value = "/")
    public void redirect(HttpServletRequest response) throws IOException {
        response.sendRedirect("/swagger-ui.html");
    }
}

```

```
    }  
}
```



Nesse momento descobri que o Spring 3.4.4 não é compatível com Swagger (por algum motivo). Vamos voltar a versão do Spring para 2.7. com esse pom

▼ Pom

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"
  <modelVersion>4.0.0</modelVersion>

<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.7.18</version> <!-- Downgrade para Spring Boot
  <relativePath />
</parent>

<groupId>com.douglas</groupId>
<artifactId>mongo</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>mongo</name>
<description>Demo project for Spring Boot</description>

<properties>
  <java.version>17</java.version>
</properties>

<dependencies>
  <!-- MongoDB -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb</artifactId>
  </dependency>

  <!-- Web -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
```

```

<!-- DevTools -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>

<!-- Lombok -->
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>

<!-- Test -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>

<!-- Swagger (Springfox) -->
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-boot-starter</artifactId>
    <version>3.0.0</version> <!-- Atualizado para versão mai
</dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
                <annotationProcessorPaths>
                    <path>
                        <groupId>org.projectlombok</groupId>
                        <artifactId>lombok</artifactId>
                    </path>
                </annotationProcessorPaths>
            </configuration>
        </plugin>
    </plugins>
</build>

```

```

        </configuration>
    </plugin>
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <configuration>
        <excludes>
            <exclude>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
            </exclude>
        </excludes>
    </configuration>
</plugin>
</plugins>
</build>
</project>

```

Mesmo com esse POM, um bocado de coisa começou a quebrar. Decidimos pular o Swagger.

- Removido o Swagger, voltamos a isso:

```

@RestController
@RequestMapping("/")
public class JobPostController {

}

```

- Vamos apenas retornar uma lista de JobPosts
- Vamos começar criando o Pojo do JobPost

```

@Data
public class JobPost {
    private String profile;
    private String desc;
    private int exp;
    private String techs[];
}

```

- Agora vamos criar o método que os retorna no Controller

```
@RestController  
 @RequestMapping("/")  
 public class JobPostController {  
     @GetMapping("allJobs")  
     public List<JobPost> getAllPosts() {  
         return null;  
     }  
 }
```

- Para podermos buscar os dados no banco, vamos criar o JobPostRepository

```
public interface JobPostRepository extends MongoRepository<JobPost, Stri
```

- Agora vamos injetá-lo no Controller e usá-lo para buscar os dados

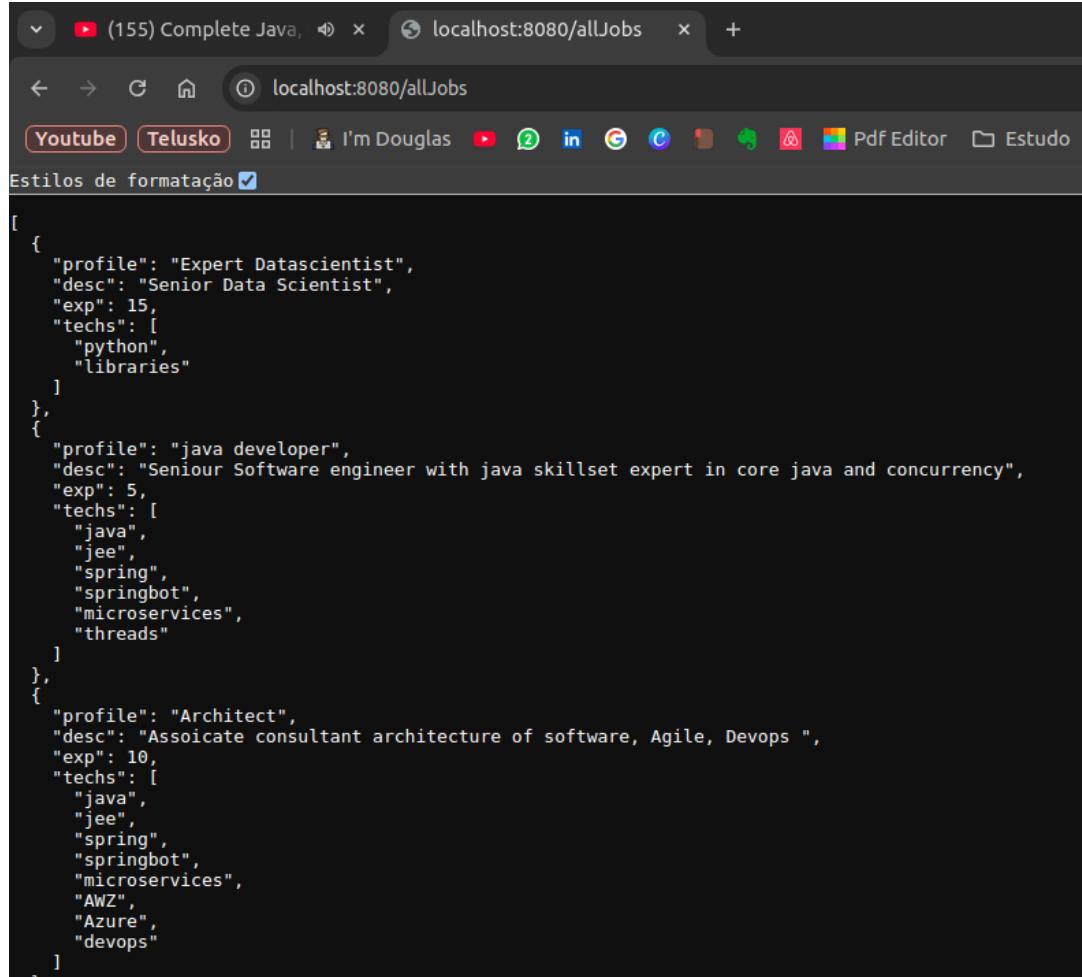
```
@RestController  
 @RequestMapping("/")  
 public class JobPostController {  
  
     @Autowired  
     JobPostRepository repository;  
  
     @GetMapping("allJobs")  
     public List<JobPost> getAllPosts() {  
         return repository.findAll();  
     }  
 }
```

- Se rodarmos o código, funciona, mas retorna uma lista vazia
- Isso ocorre pois não fizemos o mapping do Pojo para Table - ou melhor, nesse caso, Document

```
@Data  
 @Document(collection = "JobPost")  
 public class JobPost {  
     private String profile;  
     private String desc;  
     private int exp;
```

```
    private String techs[];  
}
```

- Rodando agora já funciona:



A screenshot of a web browser window titled '(155) Complete Java, localhost:8080/allJobs'. The browser's address bar shows 'localhost:8080/allJobs'. Below the address bar, there are several icons for various websites like YouTube, Telusko, and Google. The main content area displays a JSON array of three job posts. Each job post has fields: profile, desc, exp, and techs.

```
[  
  {  
    "profile": "Expert Datascientist",  
    "desc": "Senior Data Scientist",  
    "exp": 15,  
    "techs": [  
      "python",  
      "libraries"  
    ]  
  },  
  {  
    "profile": "java developer",  
    "desc": "Senior Software engineer with java skillset expert in core java and concurrency",  
    "exp": 5,  
    "techs": [  
      "java",  
      "jee",  
      "spring",  
      "springboot",  
      "microservices",  
      "threads"  
    ]  
  },  
  {  
    "profile": "Architect",  
    "desc": "Associate consultant architecture of software, Agile, Devops ",  
    "exp": 10,  
    "techs": [  
      "java",  
      "jee",  
      "spring",  
      "springboot",  
      "microservices",  
      "AWS",  
      "Azure",  
      "devops"  
    ]  
  }]
```

Post Method

- Agora vamos tentar criar um dado
- Como não estamos usando o Swagger, vamos utilizar o Postman para a requisição
- Mas antes vamos criar o método no Controller

```
@PostMapping("addJob")  
public JobPost postMethodName(@RequestBody JobPost jobPost) {  
    jobPost = repository.save(jobPost);  
  
    return jobPost;  
}
```

A screenshot of the Postman application interface. The top bar shows 'POST' and the URL 'localhost:8080/addJob'. Below the URL are tabs for 'Params', 'Authorization', 'Headers (9)', 'Body', and 'Pre-request'. The 'Body' tab is selected and has a green dot next to it. Below the tabs, there are radio buttons for 'none', 'form-data', 'x-www-form-urlencoded', and 'raw'. The 'raw' option is selected and has a blue dot next to it. The body content is a JSON object with line numbers from 1 to 10:

```
1 {  
2   "profile": "Java Expert",  
3   "desc": "Expert with Java",  
4   "exp": 15,  
5   "techs": [  
6     "java",  
7     "spring",  
8     "JEE"  
9   ]  
10 }
```

- Funcionou! Podemos confirmar através do outro endpoint



The screenshot shows a browser window with the URL `localhost:8080/allJobs`. The page title is "Estilos de formatação" with a checked checkbox icon. The content displays a JSON array of three job posts:

```
[{"exp": 12, "techs": ["c", "c++", "java", "kubernetes"], "profile": "Junior DataScientist", "desc": "DataScientiest who has end"}, {"exp": 12, "techs": ["python", "numpy", "pandas", "matplotlib"], "profile": "Java Expert", "desc": "Expert with Java"}, {"exp": 15, "techs": ["java", "spring", "JEE"]}]
```

- E deve aparecer também no Atlas

telusko.JobPost

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 2.13KB TOTAL DOCUMENTS: 11 INDEXES TOTAL SIZE: 36KB

[Find](#) [Indexes](#) [Schema Anti-Patterns \(0\)](#) [Aggregation](#) [Search In](#)

Generate queries from natural language in Compass ↗

Filter ↗ Type a query: { field: 'value' }

```

exp : 1
profile : "developer"
▶ techs : Array (5)

```

```

_id: "6"
desc : "Devops Engineer worked on kubernetes, ansible, docker"
exp : 12
profile : "development"
▶ techs : Array (4)

```

```

_id: "8"
desc : "DataScientiest who has end-to-end project experience"
exp : 12
profile : "Junior DataScientist"
▶ techs : Array (4)

```

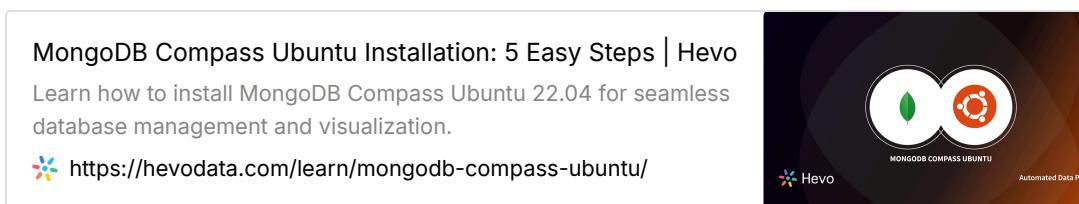
```

_id: ObjectId('67eda88c64414c4d32e03b7f')
profile : "Java Expert"
desc : "Expert with Java"
exp : 15
▶ techs : Array (3)
_class : "com.douglas.mongo.model.JobPost"

```

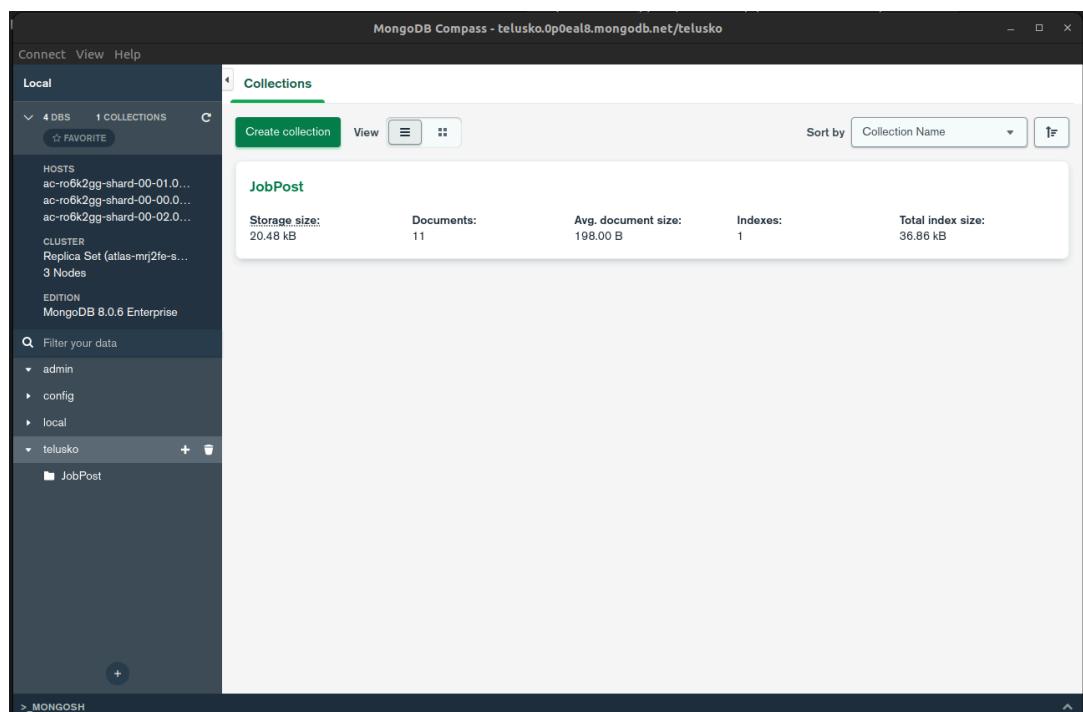
▼ Spring Compass e Atlas Search

- Atualmente estamos acessando o Mongo DB Atlas pela Web, mas vamos instalar o Compass para acessar pelo Desktop
- Se estivessemos usando o MongoDB on-premise, só seria possível acessá-lo com o Compass, não pela Web, mas o Compass serve tanto para on-premise quanto cloud
- Ele baixa pelo site normalmente, provavelmente porque está no Windows ou MacOS, eu vou seguir esse tutorial aqui:



MongoDB Compass Ubuntu Installation: 5 Easy Steps | Hevo
Learn how to install MongoDB Compass Ubuntu 22.04 for seamless database management and visualization.
<https://hevodata.com/learn/mongodb-compass-ubuntu/>

- Assim que o software estiver instalado, podemos acessar o banco pela mesma URI



- Se não tivermos pego o URI para a application ainda, podemos ir em Connect e testar com a opção pro Compass
- É interessante reparar que temos 3 hosts, porque o Mongo DB cria réplicas automaticamente

Atlas Search

- Para agilizar buscas por texto, é possível criar Indíces de busca
- Criaremos através do Mongo Atlas, partindo da nossa coleção

The screenshot shows the MongoDB Atlas interface. At the top, there are tabs for Overview, Real Time, Metrics, Collections (which is highlighted), Atlas Search, Query Insights, and Performance Advisor. Below the tabs, it says 'DATABASES: 1 COLLECTIONS: 1'. There are buttons for '+ Create Database' and 'Search Namespaces'. Under the 'telusko' database, there is a 'JobPost' collection. The 'JobPost' page displays storage information (Storage Size: 36KB, Logical Data Size: 2.13KB, Total Documents: 11, Indexes Total Size: 36KB), search options (Find, Indexes, Schema Anti-Patterns, Aggregation, Search Indexes), and a natural language query generator. A 'Filter' button is also present.

- Podemos usar tudo padrão e ir dando next até chegar nessa tela:

The screenshot shows the MongoDB Atlas Atlas Search interface. At the top, there are tabs for Overview, Real Time, Metrics, Collections, and Atlas Search (which is highlighted). It shows the 'telusko' database and 'JobPost' collection. The 'JobPost' index details page includes fields like Version (8.0.6), Region (AWS São Paulo (sa-east-1)), Cluster (M0 Sandbox (General)), and a table for index configuration. The table has columns: Database, Collection, Index Name, Status, Queryable, Type, Index Fields, Documents, Size, and Actions. One entry is shown: telusko, JobPost, default, PENDING, search, [dynamic], 0 (0%) indexed of 11, 0B, with a 'QUERY' button.

- Uma informação relevante é o campo Index Fields, que significa que todos os campos estão sendo considerados para o índice
- Em casos de bancos muito grandes e/ou querer buscar apenas por alguns campos, é possível alterar isso
- Após algum tempo, o campo Documents indicará que todos os documentos foram mapeados pelo índice
- Agora vamos testar
- Podemos clicar, ao lado do campo em que está descrito o índice, no botão Query

The screenshot shows the MongoDB Atlas Search interface. At the top, it displays 'VERSION 8.0.6' and 'REGION AWS Sao'. Below the header, there are tabs for 'Metrics', 'Collections', 'Atlas Search' (which is selected), 'Query Insights', 'Performance Advisor', 'Online Archive', and 'Cmd Line Tools'. A sub-header 'TRYABLE' is visible above the search area. The main section is titled 'Search Tester' with the instruction: 'Enter a term below to run a simple \$search query on these sample documents from telusko.JobPost using your search index and see the top 10 documents that are returned ranked by score. To edit \$search query or to run \$searchMeta, click "Edit Query" button. To run queries with stages after \$search, visit the aggregation pipeline.' A search input field contains 'kubernetes', with a 'Search' button and an 'Edit Query' button next to it. Below the input field, it says 'This search took 0.01 seconds'. The results section shows one document with a green background: 'SCORE: 1.9799914360046387 _id: "6"' followed by the document's fields: 'desc : "Devops Engineer worked on kubernetes, ansible, docker"', 'exp : 12', and 'profile : "development"'. There is also a small 'Edit' icon next to the document ID.

Client Side Field Encryption

- Mongo DB Atlas ainda tem uma funcionalidade de criptografar dados antes de chegar no banco de dados
- O Telusko só mencionou e não vai mostrar, mas é interessante

Search via Java

- Agora vamos fazer uma busca usando o índice pelo Java
- Além da busca pelo índice, também faremos algumas etapas de agregação:
 - vamos ordenar de forma decrescente por tempo de experiência
 - e vamos limitar a apenas 5 resultados
- Vamos criar um novo método no Controller para isso

```
@GetMapping("job/{searchText}")
public List<JobPost> searchJob(@PathVariable String searchText) {
    return repository.findAll();
}
```

- O código roda mas retorna tudo
- Vamos criar um novo repositório chamado SearchRepository



Ele fez uma query toda estranha. Se tiver que ser assim nem vale a pena usar MongoDB. Vou encerrar esse projeto por aqui.

▼ Spring AOP

- AOP significa Aspect Oriented Programming
- Ele está usando o projeto com os Aliens com Boot
- Toda aplicação possui algumas características que quase sempre estarão presentes
 - Logging

- Security
- Etc...
- Vamos incluir um log no método de Get com xml usando sysout

```
@GetMapping(path = "aliens", produces = { "application/xml" })
@ResponseBody
public List<Alien> getAliens() {
    System.out.println("getAliens method called");
    return alienRepository.findAll();
}
```

- Podemos, por exemplo, precisar de segurança nesse trecho também

```
@GetMapping(path = "aliens", produces = { "application/xml" })
@ResponseBody
public List<Alien> getAliens() {
    System.out.println("getAliens method called");
    // check if user exists
    // check if user has access
    // ...
    return alienRepository.findAll();
}
```

- Mas nada disso faz parte da lógica de negocio, não diz respeito ao que o usuário espera
- A prática correta, nesse caso, é extrair essa lógica
- Vamos remover tudo isso

```
@GetMapping(path = "aliens", produces = { "application/xml" })
@ResponseBody
public List<Alien> getAliens() {
    return alienRepository.findAll();
}
```

- E criar uma classe à parte chamada LoggingAspect para cuidar do log e passaremos o log para ela

```
public class LoggingAspect {
    public void log() {
        System.out.println("getAliens method called");
```

```
}
```

- Como faríamos agora o método get chamar esse método?
- Se chamarmos a função no meio do método, cometemos o mesmo erro, é o mesmo número de linhas de código
- Então como fazer?
- Esse problema é chamado de **Cross Cutting Concerns**
- O problema (concern) do logging é presente em todos os métodos (cross cutting)
- Para solucionar isso, o Spring usa o AOP

AOP Concepts

- *Aspect*: um aspect é uma classe que terá todos os métodos de um certo "concern"
- *Join point*: é o ponto em que o método (e.g. getAliens) chama o aspecto ("logging")
- *Advice*: o advice é a ação tomada no joint point, no nosso caso o log será o advice
- *Pointcut*: é um padrão que fará a conexão entre o aspecto e o joint point
- *Weaving*: é a conexão entre o aspect e o joint point, pode ser feita na compilação ou em runtime, com Spring ocorre em runtime
- *Types of advice*:
 - *Before advice*: "quero chamar o log antes do método"
 - *After advice*: "quero chamar o log depois do método"
 - *Around advice*: "quero chamar o log durante o método"

Implementation

- Para usar AOP, precisamos adicionar algumas dependências
- Com Maven, teríamos que incluí-las, mas no Spring Boot ele já inclui
- Para transformar nossa classe num aspect usaremos uma annotation e usaremos component para que o Spring o crie

```
@Aspect  
@Component  
public class LoggingAspect {
```

```
public void log() {  
    System.out.println("getAliens method called");  
}  
}
```

- Já temos um aspect, nosso Joint point será o get, mas precisamos criar o pointcut primeiro, determinando o tipo de advice. Aqui faremos com que o log venha antes do método getAliens

```
@Before("execution(public * com.douglas.mvc_boot.AlienController.getAliens())")  
public void log() {  
    System.out.println("getAliens method called");  
}
```

- Tivemos que incluir todo o caminho do nome do método para que o aspect o encontrasse, mas se agora fizermos o get pelo Postman, vemos o log no terminal

```
2025-04-07T10:02:14.005-03:00  
tcherServlet'  
2025-04-07T10:02:14.065-03:00  
0 ms  
getAliens method called
```

- O interessante é que o Controller não tem ideia da existência desse Aspect

Logger

- Até agora estamos utilizando o sysout, mas usaremos Logger agora.

```
public class LoggingAspect {  
  
    private static final Logger LOGGER = LoggerFactory.getLogger(LoggingAspect.class);
```

- E usaremos o Logger ao invés do sysout

```
private static final Logger LOGGER = LoggerFactory.getLogger(LoggingAspect.class);  
  
@Before("execution(public * com.douglas.mvc_boot.AlienController.getAliens())")  
public void log() {  
    LOGGER.info("getAliens method called");  
}
```

```
2025-04-07T10:05:31.780-03:00 INFO 67826 --- [mvc-boot] [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : Initializing servlet 'dispatcherServlet'
2025-04-07T10:05:31.781-03:00 INFO 67826 --- [mvc-boot] [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2025-04-07T10:05:31.783-03:00 INFO 67826 --- [mvc-boot] [nio-8080-exec-2] com.douglas.mvc_boot.LoggingAspect : getAliens method called
```

- Esse log é do tipo INFO. Podemos reduzir o nível dos logs com a configuração no properties, mas manteremos como info
- Podemos também salvar num arquivo externo usando o properties

```
logging.level.root=info
logging.file.name=app.log
```

```
2025-04-07T10:07:49.849-03:00 INFO 67826 --- [mvc-boot] [restartedMain] o.s.b.a.e.EmbeddedDatabaseBuilder : Database JDBC URL [Connecting through datasource 'HikariDataSource' (Hikari)]
2025-04-07T10:07:49.850-03:00 INFO 67826 --- [mvc-boot] [restartedMain] o.s.b.a.e.EmbeddedDatabaseBuilder : Database driver: undefined/unknown
2025-04-07T10:07:49.908-03:00 INFO 67826 --- [mvc-boot] [restartedMain] o.s.b.a.e.EmbeddedDatabaseBuilder : Database version: 9.2
2025-04-07T10:07:49.911-03:00 INFO 67826 --- [mvc-boot] [restartedMain] o.s.b.a.e.EmbeddedDatabaseBuilder : Autocommit mode: undefined/unknown
2025-04-07T10:07:49.944-03:00 INFO 67826 --- [mvc-boot] [restartedMain] o.s.b.a.e.EmbeddedDatabaseBuilder : Isolation level: undefined/unknown
2025-04-07T10:07:49.945-03:00 INFO 67826 --- [mvc-boot] [restartedMain] o.s.b.a.e.EmbeddedDatabaseBuilder : Minimum pool size: undefined/unknown
2025-04-07T10:07:49.945-03:00 INFO 67826 --- [mvc-boot] [restartedMain] o.s.b.a.e.EmbeddedDatabaseBuilder : Maximum pool size: undefined/unknown
2025-04-07T10:07:49.980-03:00 INFO 67826 --- [mvc-boot] [restartedMain] o.s.b.a.e.EmbeddedDatabaseBuilder : Database JDBC URL [Connecting through datasource 'HikariDataSource' (Hikari)]
2025-04-07T10:07:49.987-03:00 INFO 67826 --- [mvc-boot] [restartedMain] o.s.b.a.e.EmbeddedDatabaseBuilder : Database driver: undefined/unknown
2025-04-07T10:07:49.987-03:00 INFO 67826 --- [mvc-boot] [restartedMain] o.s.b.a.e.EmbeddedDatabaseBuilder : Database version: 9.2
2025-04-07T10:07:49.989-03:00 INFO 67826 --- [mvc-boot] [restartedMain] o.s.b.a.e.EmbeddedDatabaseBuilder : Autocommit mode: undefined/unknown
2025-04-07T10:07:49.990-03:00 INFO 67826 --- [mvc-boot] [restartedMain] o.s.b.a.e.EmbeddedDatabaseBuilder : Isolation level: undefined/unknown
2025-04-07T10:07:49.994-03:00 INFO 67826 --- [mvc-boot] [restartedMain] o.s.b.a.e.EmbeddedDatabaseBuilder : Minimum pool size: undefined/unknown
2025-04-07T10:07:49.996-03:00 WARN 67826 --- [mvc-boot] [restartedMain] o.s.b.a.e.EmbeddedDatabaseBuilder : Maximum pool size: undefined/unknown
2025-04-07T10:07:49.996-03:00 INFO 67826 --- [mvc-boot] [restartedMain] o.s.b.a.e.EmbeddedDatabaseBuilder : Database JDBC URL [Connecting through datasource 'HikariDataSource' (Hikari)]
2025-04-07T10:07:50.013-03:00 INFO 67826 --- [mvc-boot] [restartedMain] o.s.b.a.e.EmbeddedDatabaseBuilder : Database driver: undefined/unknown
2025-04-07T10:07:50.013-03:00 INFO 67826 --- [mvc-boot] [restartedMain] o.s.b.a.e.EmbeddedDatabaseBuilder : Database version: 9.2
2025-04-07T10:07:50.048-03:00 WARN 67826 --- [mvc-boot] [restartedMain] o.s.b.a.e.EmbeddedDatabaseBuilder : Autocommit mode: undefined/unknown
2025-04-07T10:07:50.051-03:00 INFO 67826 --- [mvc-boot] [restartedMain] o.s.b.a.e.EmbeddedDatabaseBuilder : Isolation level: undefined/unknown
2025-04-07T10:07:50.093-03:00 INFO 67826 --- [mvc-boot] [restartedMain] o.s.b.a.e.EmbeddedDatabaseBuilder : Minimum pool size: undefined/unknown
2025-04-07T10:07:50.097-03:00 INFO 67826 --- [mvc-boot] [restartedMain] o.s.b.a.e.EmbeddedDatabaseBuilder : Maximum pool size: undefined/unknown
```

```
2025-04-07T10:07:50.099-03:00 INFO 67826 --- [mvc-boot] [restartedMain] c  
2025-04-07T10:07:50.100-03:00 INFO 67826 --- [mvc-boot] [restartedMain] .C  
2025-04-07T10:07:56.753-03:00 INFO 67826 --- [mvc-boot] [http-nio-8080-e  
2025-04-07T10:07:56.754-03:00 INFO 67826 --- [mvc-boot] [http-nio-8080-e  
2025-04-07T10:07:56.754-03:00 INFO 67826 --- [mvc-boot] [http-nio-8080-e  
2025-04-07T10:07:56.755-03:00 INFO 67826 --- [mvc-boot] [http-nio-8080-e
```

- O arquivo é criado na raiz

After

- Agora testaremos um after advice com um log após o getAliens
- Criaremos um sysout dentro do método só para visualizar a diferença

```
@Aspect  
@Component  
public class LoggingAspect {  
  
    private static final Logger LOGGER = LoggerFactory.getLogger(LoggingAspect.class);  
  
    @Before("execution(public * com.douglas.mvc_boot.AlienController.getAliens())")  
    public void logBefore() {  
        LOGGER.info("getAliens method called");  
    }  
  
    @After("execution(public * com.douglas.mvc_boot.AlienController.getAliens())")  
    public void logAfter() {  
        LOGGER.info("getAliens method executed");  
    }  
}  
  
@GetMapping(path = "aliens", produces = { "application/xml" })  
@ResponseBody  
public List<Alien> getAliens() {  
    System.out.println("During call");  
    return alienRepository.findAll();  
}  
  
getAliens method called  
During call  
getAliens method executed
```

- O after advice pode ser subdividido em 3 tipos:
 - After Finally: que é o padrão, chamado após o método encerrar independente do que ocorrer
 - After returning: que ocorre depois de o método retornar com sucesso
 - After throwing: que ocorre caso o método lance uma exceção
- Vamos tentar o finally com uma exceção no método

```
@GetMapping(path = "aliens", produces = { "application/xml" })
@ResponseBody
public List<Alien> getAliens() {
    int i = 9 / 0;
    return alienRepository.findAll();
}
```

getAliens method executed

- Recebemos um erro gigantesco depois do log, mas ele não deixou de existir.

After Returning / Throwing

- Agora vamos executar o log apenas quando for uma execução com sucesso
- Precisamos apenas retornar o After para AfterReturning

```
@AfterReturning("execution(public * com.douglas.mvc_boot.AlienController.get")
public void logAfter() {
    LOGGER.info("getAliens method executed");
}
```

```
2025-04-09T08:53:38.415+03:00  INFO 1428154 --- [mvc-boot] [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2025-04-09T08:53:38.454+03:00  INFO 1428154 --- [mvc-boot] [nio-8080-exec-2] com.douglas.mvc.boot.LoggingAspect : getAliens method called
2025-04-09T08:53:38.459+03:00 ERROR 1428154 --- [mvc-boot] [nio-8080-exec-2] o.a.c.c.C.[.].{dispatcherServlet} : Servlet.service() for servlet [dispatcherServlet] in context with path [] threw exception [Request processing failed: java.lang.ArithmetricException: / by zero] with root cause
java.lang.ArithmetricException: / by zero
at com.douglas.mvc.boot.AlienController.getAliens(AlienController.java:22) ~[classes/:na]
at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method) ~[na:na]
at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:77) ~[na:na]
at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43) ~[na:na]
```

- Agora não vemos mais o log de executed
- Por outro lado, podemos adicionar um log que só aparece quando há um erro:

```
@AfterThrowing("execution(public * com.douglas.mvc_boot.AlienController.get")
public void logAfterThrowing() {
```

```

    LOGGER.info("getAliens method threw error");
}

```

```

t'
2025-04-08T08:55:33.172-03:00 INFO 1428154 --- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet      : Initializing Servlet 'dispatcherServlet'
2025-04-08T08:55:33.173-03:00 INFO 1428154 --- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet      : Completed initialization in 1 ms
2025-04-08T08:55:33.174-03:00 INFO 1428154 --- [nio-8080-exec-2] com.douglas.mvc.boot.LoggingAspect       : getAliens method called
2025-04-08T08:55:33.174-03:00 INFO 1428154 --- [nio-8080-exec-2] com.douglas.mvc.boot.LoggingAspect       : getAliens method threw error
2025-04-08T08:55:33.175-03:00 ERROR 1428154 --- [nio-8080-exec-2] o.a.c.c.C.[.].[/dispatcherServlet]        : Servlet.service() for servlet [dispatcherServlet] in context with path [] threw exception [Request processing failed: java.lang.ArithmetricException: / by zero] with root cause
java.lang.ArithmetricException: / by zero
at com.douglas.mvc.boot.AlienController.getAliens(AlienController.java:22) ~[classes/:na]
at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method) ~[na:na]

```

- Finalmente, podemos remover o erro na chamada do método e receberemos apenas o AfterReturning:

```

@GetMapping(path = "aliens", produces = { "application/xml" })
@ResponseBody
public List<Alien> getAliens() {
    return alienRepository.findAll();
}

```

```

t'
2025-04-08T08:56:22.041-03:00 INFO 1428154 --- [nio-8080-exec-2] c.douglas.mvc_boot.MvcBootApplication   : Started MvcBootApplication in 0.676 seconds (process running for 186.397)
2025-04-08T08:56:22.042-03:00 INFO 1428154 --- [nio-8080-exec-2] ConditionEvaluationDeltaLoggingListener : Condition evaluation unchanged
2025-04-08T08:56:26.255-03:00 INFO 1428154 --- [nio-8080-exec-2] o.a.c.c.C.[Tomcat].[localhost].[/]      : Initializing Spring DispatcherServlet 'dispatcherServlet'
2025-04-08T08:56:26.255-03:00 INFO 1428154 --- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet      : Initializing Servlet 'dispatcherServlet'
2025-04-08T08:56:26.256-03:00 INFO 1428154 --- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet      : Completed initialization in 1 ms
2025-04-08T08:56:26.256-03:00 INFO 1428154 --- [nio-8080-exec-2] com.douglas.mvc.boot.LoggingAspect       : getAliens method called
2025-04-08T08:56:26.311-03:00 INFO 1428154 --- [nio-8080-exec-2] com.douglas.mvc.boot.LoggingAspect       : getAliens method executed

```

▼ Spring Security

- Spring Security é o padrão para proteção de aplicações backend com Spring
- Em geral, as principais informações que precisamos proteger são as informações de usuários, necessárias em Login e Logout, por exemplo
- Além de acessar, temos que permitir que alguns usuários vejam algumas telas ou endpoints e outras não
- Precisamos também implementar algum tipo de criptografia nas senhas. Atualmente o algoritmo Bcrypt é o mais seguro para senhas
- No código, temos que checar em todo endpoint se o usuário e a senha estão certos, podemos tentar usar LDAP ou OAuth2 para autenticação

- Finalmente, em casa etapa, precisamos avaliar a autorização dos usuários
- A ideia do Spring Security é que ele facilita a maior parte desse processo, exigindo apenas alguma configuração

Prática

- Implementaremos Spring Security em uma aplicação Web
- Criaremos um projeto com Spring Boot do zero. Usaremos esses pacotes:
 - Lombok
 - DevTools
 - Spring Web
 - Spring JPA
 - Spring Security
- Começaremos criando um controller para poder acessar algo

```
@Controller // se for RestController não retorna tela
@RequestMapping("/")
public class HomeController {
    @GetMapping("/")
    public String home(@RequestParam String param) {
        return "home.jsp";
    }
}
```

- Para nosso home.jsp, vamos criar uma pasta webapp em main com nosso jsp

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>Document</title>
    </head>
    <body>
        <h1>Hello Aliens</h1>
    </body>
</html>
```

- Para servirmos o JSP precisamos do Tomcat Jasper

```
<!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-jasper -->
<dependency>
    <groupId>org.apache.tomcat</groupId>
    <artifactId>tomcat-jasper</artifactId>
    <version>10.1.39</version>
</dependency>
```

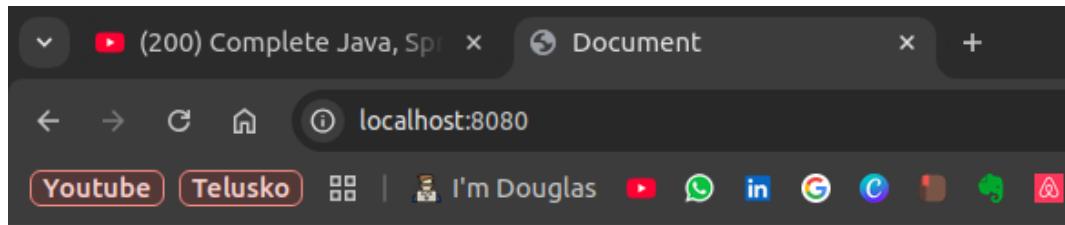


Nessa hora, tentando rodar o código, ele quebra. Isso ocorreu porque o Telusko deixou para incluir JPA e Security depois. Faremos o mesmo, pois sei que depois vai travar com security não configurado também.

Vamos tirar o JPA e o Security do pom.

```
<!-- <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency> -->

<!-- <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-test</artifactId>
        <scope>test</scope>
    </dependency> -->
```



Hello Aliens

- Em seguida, queremos que seja necessário algum nível de segurança para home
- Faremos algumas coisas para isso:
 1. Toda vez que um usuário não autenticado acessar, temos que redirecionar pra tela de login
 2. Quando ele tentar logar, precisamos que a autenticação funcione
 3. Precisamos manter a sessão
- Tudo isso é factível apenas com a inclusão da dependência do Spring Security. Faremos a inclusão e vamos tentar acessar de novo

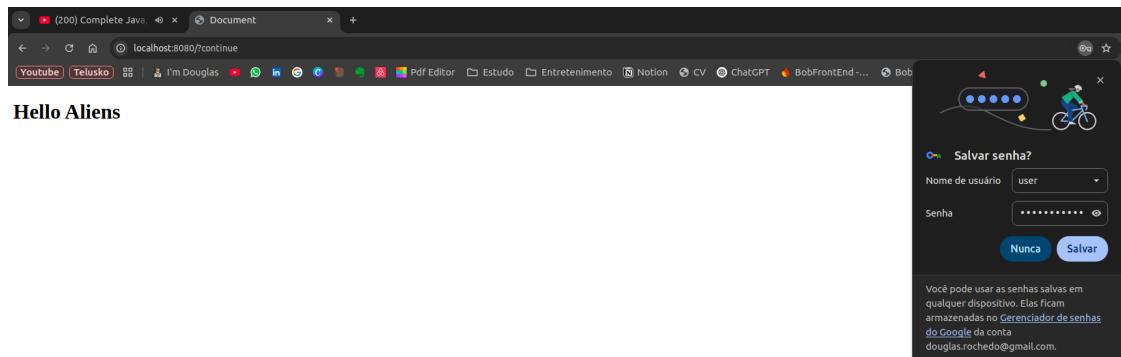
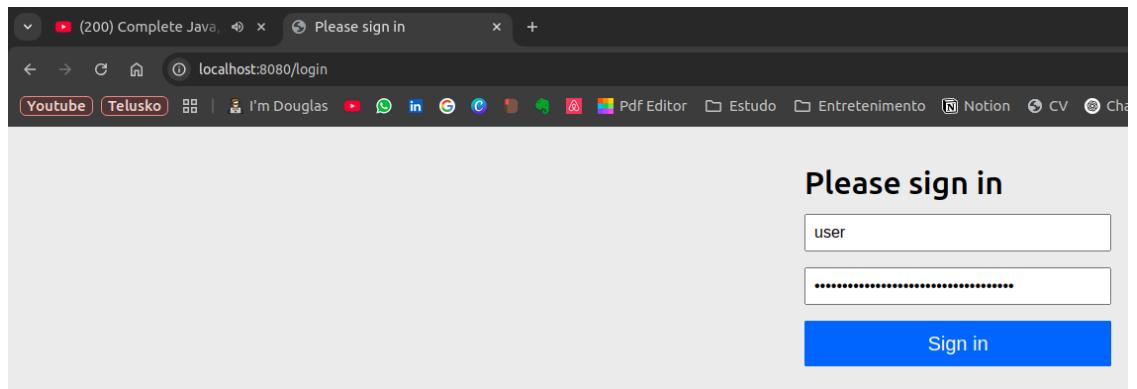
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

The screenshot shows a browser window with the URL 'localhost:8080/login'. The page title is 'Please sign in'. It contains two input fields: 'Username' and 'Password', and a blue 'Sign in' button.

- A tela de Login é gerada automaticamente pelo Spring Security! E já ocorre o redirecionamento por não estarmos logados

- Para logar aqui, usaremos o usuário padrão “user”. A senha é impressa no console

```
2025-04-08T17:42:01.653-03:00  INFO 1969252 --- [security] [  restartedMain] o.s.c.ServletWebServerApplicationContext : Initializing Spring embedded
context
2025-04-08T17:42:01.653-03:00  INFO 1969252 --- [security] [  restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext
initialization completed in 975 ms
2025-04-08T17:42:01.984-03:00  WARN 1969252 --- [security] [  restartedMain] .s.s.UserDetailsServiceAutoConfiguration :
Using generated security password: cf89ab70-bd8f-4dc8-92d6-c8ae16c7ad2c
This generated password is for development use only. Your security configuration must be updated before running your application in production.
2025-04-08T17:42:02.008-03:00  INFO 1969252 --- [security] [  restartedMain] r$InitializeUserDetailsServiceManager : Global AuthenticationManager
initialized with UserDetailsService bean with name inMemoryUserDetailsService
2025-04-08T17:42:02.187-03:00  INFO 1969252 --- [security] [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer      : LiveReload server is runn
ing on port 35729
```



- É possível alterarmos o usuário e a senha em memória, assim como é possível também armazenar pares de usuário e senha no banco de dados para que usuários reais se registrem

Definindo Credenciais na Memória

- Para definirmos usuário e senha diferentes do padrão, criaremos um arquivo de configuração



Para seguir adiante, precisamos voltar a versão do Spring para 2.6.14 porque o WebSecurityConfigurerAdapter estava deprecado

▼ Pom resultante

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.6.14</version> <!-- Versão compatível com WebSecurityConfigurerAdapter -->
    <relativePath /> <!-- lookup parent from repository -->
  </parent>

  <groupId>com.douglas</groupId>
  <artifactId>security</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>security</name>
  <description>Demo project for Spring Boot</description>

  <properties>
    <java.version>11</java.version> <!-- Java 11 é o default recomendado -->
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
    </dependency>
```

```

        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>

    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-test</artifactId>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>org.apache.tomcat</groupId>
        <artifactId>tomcat-jasper</artifactId>
        <version>9.0.85</version> <!-- Versão compatível com Spring Boot 2.5.0 -->
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
                <annotationProcessorPaths>
                    <path>
                        <groupId>org.projectlombok</groupId>
                        <artifactId>lombok</artifactId>
                    </path>
                </annotationProcessorPaths>
            </configuration>
        </plugin>
    </plugins>

```

```

</plugin>
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <configuration>
        <excludes>
            <exclude>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
            </exclude>
        </excludes>
    </configuration>
</plugin>
</plugins>
</build>
</project>

```

```

@Configuration
@EnableWebSecurity
public class AppSecurityConfig extends WebSecurityConfigurerAdapter {

}

```

- Para poder alterar as credenciais, vamos sobrescrever um dos métodos dessa classe pai

```

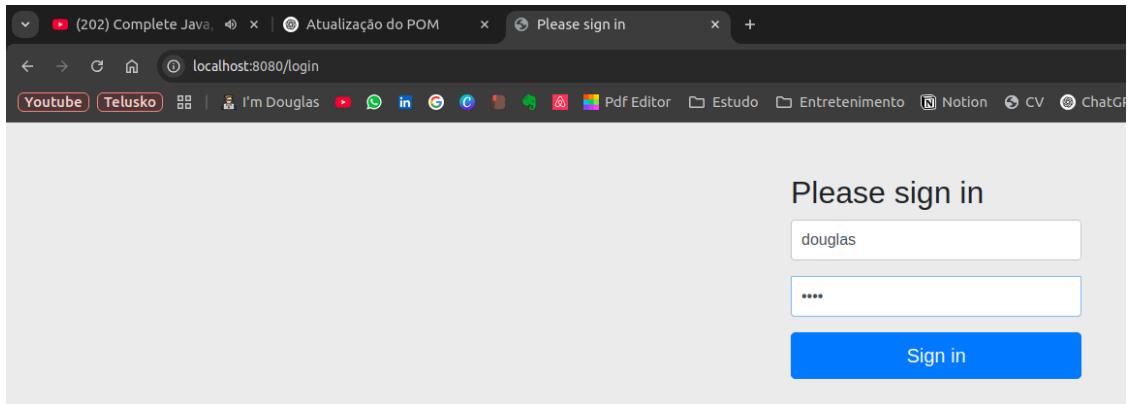
@Bean
@Override
protected UserDetailsService userDetailsService() {
    List<UserDetails> users = new ArrayList<>();
    users.add(User.withDefaultPasswordEncoder().username("douglas").password("123456").roles("USER"));

    return new InMemoryUserDetailsManager(users);
}

```

- Nesse código, estamos criando o usuário com todos os dados no código, o que está errado. Além disso, estamos utilizando o "DefaultPasswordEncoder" que está deprecado.
- Um detalhe importante é o "roles(USER)", que servirá para a etapa de autorização

- Já podemos reiniciar a aplicação e não vemos a senha no console



- Nossas credenciais novas já funcionam

Definindo Credenciais no Banco

- Agora queremos ter essas mesmas credenciais, mas armazenadas no banco de dados
- Primeiro podemos comentar todo o método criado anteriormente
- Ao invés dele, vamos criar um método que retorna um provedor de autenticação que comunica com o banco, isto é, um DAO

```
@Bean
public AuthenticationProvider authProvider() {
    DaoAuthenticationProvider provider = new DaoAuthenticationProvider();

    return provider;
}

// @Bean
// @Override
// protected UserDetailsService userDetailsService() {
//     List<UserDetails> users = new ArrayList<>();
//     users.add(User.withDefaultPasswordEncoder().username("douglas").passw

//     return new InMemoryUserDetailsManager(users);
// }
```

- Naturalmente, isso ainda não funciona pois não conectamos nenhum banco ainda
- Vamos adicionar dependências para o JPA e o conector, usaremos MySQL

```

<!-- https://mvnrepository.com/artifact/com.mysql/mysql-connector-j -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.46</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-data-jpa -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

```

- Faremos também a configuração do acesso no properties

```

spring.datasource.url=jdbc:mysql://localhost:3306/mysqlteste
spring.datasource.username=douglas
spring.datasource.password=123
spring.datasource.driver-class-name=com.mysql.jdbc.Driver

```

- Agora precisamos criar nossa tabela de usuários no banco e popular com alguns exemplos

```

CREATE TABLE user(
    id int NOT NULL PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(255),
    password VARCHAR(255)
);

```

The screenshot shows the MySQL Workbench interface with the 'user' table selected. The table has three columns: 'id' (primary key, auto-increment, int), 'username' (varchar(255)), and 'password' (varchar(255)). There are two rows of data: one with id 101, username 'admin', and password 'admin'; and another with id 102, username 'dev', and password 'dev'.

		* id	username	password
		int	varchar(255)	varchar(255)
	>	101	admin	admin
	>	102	dev	dev

- Finalmente, modelaremos a tabela no código

```

@Data
@Entity
public class UserEntity {
    @Id

```

```
private long id;  
private String username;  
private String password;  
}
```

- Agora podemos voltar para o Security Config, definindo o service e o mecanismo de encoding da senha

```
@Configuration  
@EnableWebSecurity  
public class AppSecurityConfig extends WebSecurityConfigurerAdapter {  
    @Autowired  
    private UserDetailsService userDetailsService;  
  
    @Bean  
    public AuthenticationProvider authProvider() {  
        DaoAuthenticationProvider provider = new DaoAuthenticationProvider();  
        provider.setUserDetailsService(userDetailsService);  
        provider.setPasswordEncoder(NoOpPasswordEncoder.getInstance());  
        return provider;  
    }  
  
}
```

- Para esse exemplo, estamos utilizando um Encoder que na verdade não faz nada, mas apenas para fins didáticos
- Precisamos agora definir um UserDetailsService que saiba que precisa acessar os dados no banco

```
@Service  
public class MyUserDetailsService implements UserDetailsService{  
  
    @Override  
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
        // TODO Auto-generated method stub  
        throw new UnsupportedOperationException("Unimplemented method 'loadUserByUsername'");  
    }  
  
}
```

- Esse código deverá buscar o usuário no banco e ser usado pelo authProvider

- Para que ele possa fazer isso, precisamos também criar o UserRepository

```
public interface UserRepository extends JpaRepository<UserEntity, Long> {  
}
```

- Precisamos criar aqui um método que busque o usuário pelo nome, conforme esperado pelo Details Service

```
public interface UserRepository extends JpaRepository<UserEntity, Long> {  
    UserEntity findByUsername(String username);  
}
```

- Agora, voltando para o Details Service

```
@Override  
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
    UserEntity userEntity = userRepository.findByUsername(username);  
  
    if (userEntity == null) {  
        throw new UsernameNotFoundException("User 404");  
    }  
  
    return null;  
}
```

- Estamos buscando a informação do usuário e lançando um erro caso não seja encontrado
- Contudo, ainda não retornamos nada, porque precisamos retornar um UserDetails
- Só que UserDetails é uma interface, não uma classe
- Vamos criar, portanto, mais um objeto UserPrincipal, que implemente UserDetails - Principal aqui significa o usuário atual

```
package com.douglas.security;  
  
import java.util.Collection;  
  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.userdetails.UserDetails;
```

```
public class UserPrincipal implements UserDetails {

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException("Unimplemented method 'getAuthorities'")
    }

    @Override
    public String getPassword() {
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException("Unimplemented method 'getPassword()")
    }

    @Override
    public String getUsername() {
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException("Unimplemented method 'getUsername()")
    }

    @Override
    public boolean isAccountNonExpired() {
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException("Unimplemented method 'isAccountNonExpired()")
    }

    @Override
    public boolean isAccountNonLocked() {
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException("Unimplemented method 'isAccountNonLocked()")
    }

    @Override
    public boolean isCredentialsNonExpired() {
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException("Unimplemented method 'isCredentialsNonExpired()")
    }

    @Override
    public boolean isEnabled() {
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException("Unimplemented method 'isEnabled()")
    }
}
```

```
}
```

```
}
```

- A interface já nos obriga a implementar vários métodos por padrão. Todos os que retornam "boolean" vamos fazer retornar true por enquanto

```
@Override  
public boolean isAccountNonExpired() {  
    return true;  
}  
  
@Override  
public boolean isAccountNonLocked() {  
    return true;  
}  
  
@Override  
public boolean isCredentialsNonExpired() {  
    return true;  
}  
  
@Override  
public boolean isEnabled() {  
    return true;  
}
```

- Para implementar os métodos de getUsername e getPassword
- Para podermos acessar essas informações, precisamos ter acesso ao UserEntity dentro dessa classe

```
@AllArgsConstructor  
public class UserPrincipal implements UserDetails {  
  
    private UserEntity userEntity;  
  
    @Override  
    public Collection<? extends GrantedAuthority> getAuthorities() {  
        // TODO Auto-generated method stub  
        throw new UnsupportedOperationException("Unimplemented method 'getAuthorities()'")  
    }
```

```
@Override  
public String getPassword() {  
    return userEntity.getPassword();  
}  
  
@Override  
public String getUsername() {  
    return userEntity.getUsername();  
}
```

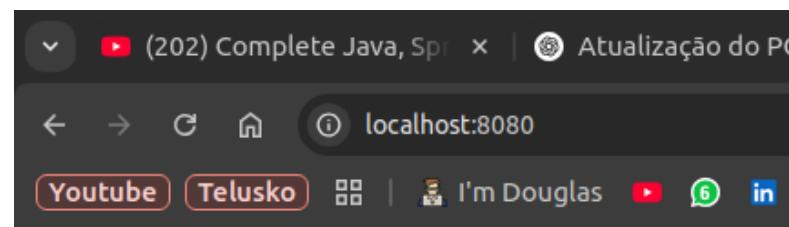
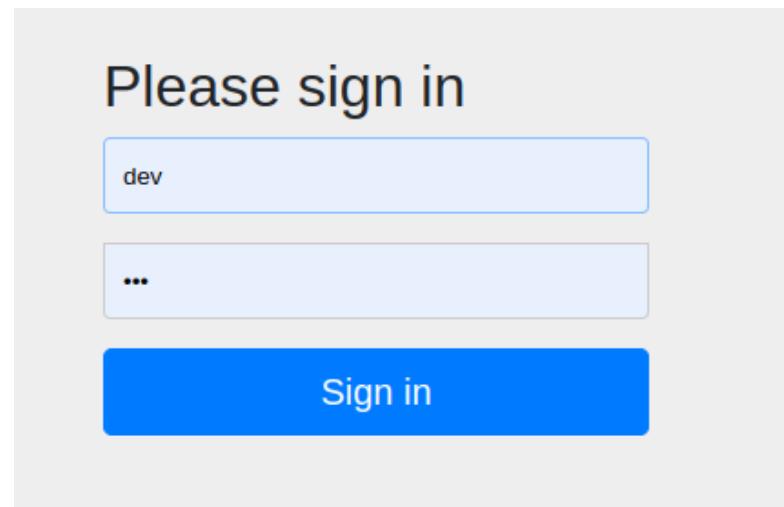
- Fica faltando apenas o getAuthorities. Por enquanto, apenas retornaremos USER

```
@Override  
public Collection<? extends GrantedAuthority> getAuthorities() {  
    return Collections.singleton(new SimpleGrantedAuthority("USER"));  
}
```

- Finalmente, de volta ao Details Service, poderemos criar um UserPrincipal e retorná-lo

```
@Override  
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
    UserEntity userEntity = userRepository.findByUsername(username);  
  
    if (userEntity == null) {  
        throw new UsernameNotFoundException("User 404");  
    }  
  
    return new UserPrincipal(userEntity);  
}
```

- Finalmente podemos rodar o código!
- Recebemos algumas reclamações de SSL no banco mas o código subiu



Hello Aliens

- Conseguimos acessar com as credenciais armazenadas no banco!
- Podemos ainda tentar acessar com uma credencial inexistente



BCrypt Password Encoder

- Um dos erros que comentemos - deliberadamente - no código foi não codificar as senhas, usando o NoOpPasswordEncoder
- Essa não é a forma correta de fazer
- Existem várias formas de criptografar os dados, mas a maioria não é 100% confiável
- Uma forma mais moderna e confiável é o BCrypt, que faz algumas "rodadas" de hashing do conteúdo
- Foi uma forma de criptografia desenvolvida especificamente para senhas
- Para usar BCrypt, podemos apenas alterar no AppSecurityConfig

```
@Bean
public AuthenticationProvider authProvider() {
    DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
    provider.setUserDetailsService(userDetailsService);
    provider.setPasswordEncoder(new BCryptPasswordEncoder());
    return provider;
}
```

- Já funciona
- Contudo, agora nossas senhas armazenadas no banco não vão funcionar, porque estão em formato de texto

- Vamos gerar uma senha com BCrypt usando uma plataforma online e armazenar no banco

URL Slug Generator

Generate Bcrypt password hashes with desired cost option.
Verify/Validate existing Bcrypt hashes.

<https://bcrypt.online/>



Bcrypt Hash
Generator & Verifier

bcrypt.online
esse.tools

```
insert into
`user` (
  id,
  username,
  password
)
values
(
  103,
  "douglas",
  "$2y$10$2UGVhUGtIYUXsqAA3d8eD.AjwmwhO6O0zcawE6pw2rC5JLpyK7u
);
```

	Q	* id	int	username	varchar(255)	password	varchar(255)	▼
	>	101		admin	admin			🔍
	>	102		dev	dev			
	>	103		douglas	\$2y\$10\$2UGVhUGtIYUXsqAA3d8eD.AjwmwhO6O0zcawE6pw2rC5JLpyK7uHS			

- Agora podemos testar com "douglas" e "1234"



Customizando a tela de login

- Agora nosso propósito é termos nossa própria tela de login, ao invés da tela padrão do Spring
- Criaremos uma tela num arquivo login.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1" isELIgnored="false"%>
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>Login</title>
</head>
<body>
<h1>Login</h1>
${SPRING_SECURITY_LAST_EXCEPTION.message}
<form action="login" , method="POST">
<table>
<tr>
<td>User:</td>
<td><input type="text" name="username" value="" /></td>
</tr>

<tr>
<td>Password:</td>
<td><input type="password" name="password" /></td>
</tr>
```

```

<tr>
    <td><input type="submit" name="submit" value="submit" /></td>
</tr>
</table>
</form>
</body>
</html>

```

- Criaremos também uma tela básica de logout

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1" isELIgnored="false"%>
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>Login</title>
    </head>
    <body>
        <h1>Logout</h1>
        <a href="/">home</a>
    </body>
</html>

```

- Agora precisamos configurar nossa aplicação para usar essas telas
- Naturalmente, a configuração será feita no AppSecurityConfig

```

@Override
protected void configure(HttpSecurity http) throws Exception {
    http //
        .csrf()//
        .disable()//
        .authorizeRequests()//
        .antMatchers("login")//
        .permitAll()//
        .anyRequest()//
        .authenticated()//
        .and()
        .formLogin()//
        .loginPage("/login")//
}

```

```

    .permitAll()//
    .and()//
    .logout()//
    .invalidateHttpSession(true)//
    .clearAuthentication(true)//
    .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))//
    .logoutSuccessUrl("/logout-sucess")//
    .permitAll();
}

```

- Aqui definimos para onde queremos que a aplicação direcione o usuário em cada caso
- Agora precisamos combinar isso com os ajustes no Controller

```

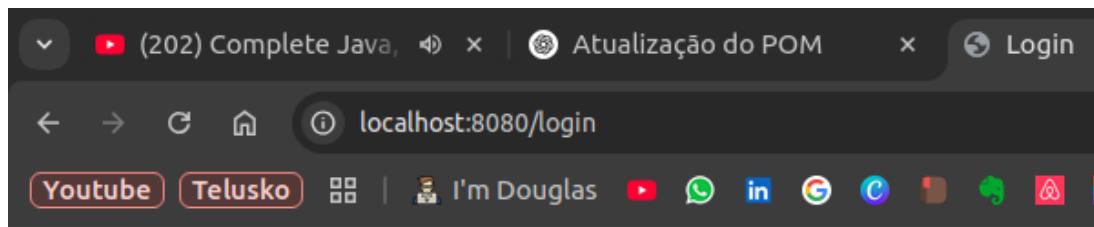
@Controller
@RequestMapping("/")
public class HomeController {
    @GetMapping("/")
    public String home() {
        return "home.jsp";
    }

    @RequestMapping("/login")
    public String loginPage() {
        return "login.jsp";
    }

    @RequestMapping("/logout-success")
    public String logoutPage() {
        return "logout.jsp";
    }
}

```

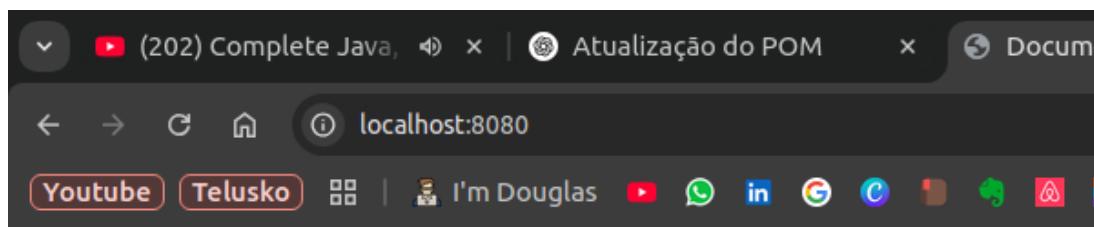
- Agora podemos novamente rodar o código e funciona



Login

User:

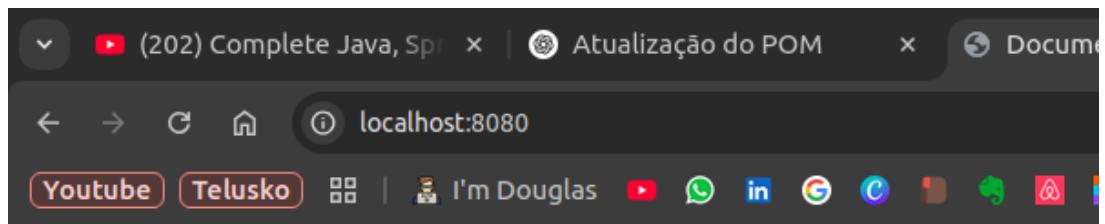
Password:



Hello Aliens

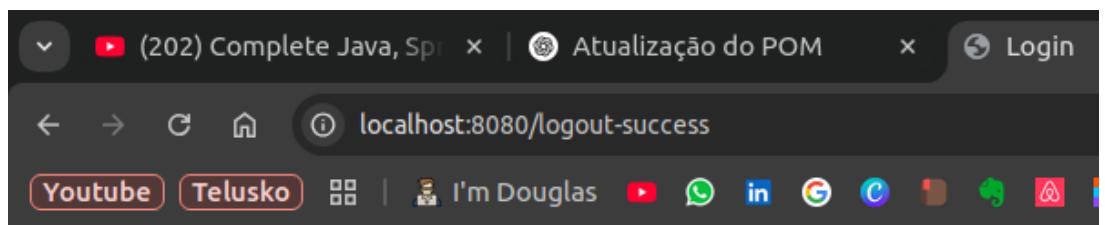
- Vamos adicionar um botão de logout na home

```
<body>
  <h1>Hello Aliens</h1>
  <a href="/logout">logout</a>
</body>
```



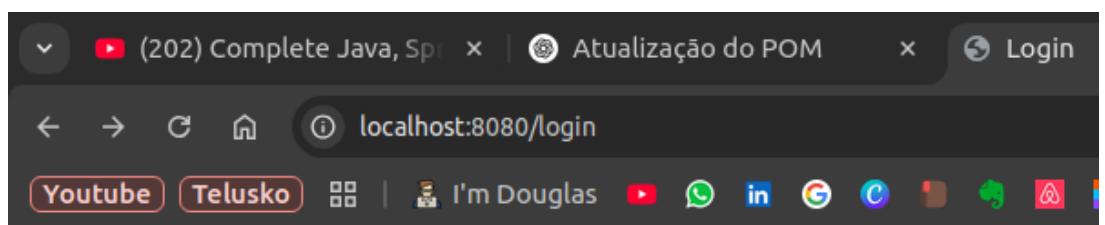
Hello Aliens

[logout](#)



Logout

[home](#)



Login

User:

Password:

OAuth2

- Por mais que tentemos protegê-los, ter dados de usuários em nosso banco é um problema
- Além disso, hoje em dia é comum usarmos redes sociais ou outras contas para acessar plataformas
- Um framework de autorização que toma conta disso por nós é o OAuth2
- Vamos expandir nosso projeto para poder fazer login pelo Google usando OAuth2
- Precisamos antes de qualquer coisa incluir a dependência do OAuth2 autoconfigure

```
<!-- https://mvnrepository.com/artifact/org.springframework.security.oauth.boot
<dependency>
  <groupId>org.springframework.security.oauth.boot</groupId>
  <artifactId>spring-security-oauth2-autoconfigure</artifactId>
  <version>2.1.2.RELEASE</version>
</dependency>
```

- Para habilitar o OAuth2, basta uma annotation no AppSecurityConfig

```
@Configuration
@EnableWebSecurity
@EnableOAuth2Sso
public class AppSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http //
            .csrf()//
            .disable()//
            .authorizeRequests()//
            .antMatchers("login")//
            .permitAll()//
            .anyRequest()//
            .authenticated()//
            .and()
            .formLogin()//
            .loginPage("/login")//
            .permitAll()//
            .and()//
            .logout()//
```

```

.invalidateHttpSession(true)//
.clearAuthentication(true)//
.logoutRequestMatcher(new AntPathRequestMatcher("/logout"))//
.logoutSuccessUrl("/logout-success")//
.permitAll();
}
}

```

- Além da annotation, removemos o UserDetailService, já que usaremos o OAuth
- Também podemos remover uma boa parte da configuração

```

@Override
protected void configure(HttpSecurity http) throws Exception {
    http //
        .csrf()//
        .disable()//
        .authorizeRequests()//
        .antMatchers("login")//
        .permitAll()//
        .anyRequest()//
        .authenticated()//
        .and()//
        .httpBasic();
}

```

- Faremos também algumas configurações no properties

```

security.oauth2.client.client-id=<CLIENT_ID>
security.oauth2.client.client-secret=<CLIENT_SECRET>
security.oauth2.client.access-token-uri=https://www.googleapis.com/oauth2/v
security.oauth2.client.user-authorization-uri=https://accounts.google.com/o/oa
security.oauth2.client.token-name=oauth_token
security.oauth2.client.authentication-scheme=query
security.oauth2.client.client-authentication-scheme=form
security.oauth2.client.scope=profile email

security.oauth2.resource.user-info-uri=https://www.googleapis.com/userinfo/v
security.oauth2.resource.prefer-token-info=false

```

- O Client ID e Client Secret são dados do desenvolvedor, precisamos criar uma conta na GCP para ter acesso ao OAuth2
- Vou usar o projeto do HelloWorldDeploy lá
- Vamos em APIs & Services → Credentials → OAuth consent screen

1 App Information

App name *

The name of the app asking for consent

User support email *

For users to contact you with questions about their consent. [Learn more ↗](#)

[Next](#)

2 Audience

Internal [?](#)

Only available to users within your organization. You will not need to submit your app for verification. [Learn more about user type ↗](#)

External [?](#)

Available to any test user with a Google Account. Your app will start in testing mode and will only be available to users you add to the list of test users. Once your app is ready to push to production, you may need to verify your app. [Learn more about user type ↗](#)

[Next](#)

Audience

3 Contact Information

Email addresses *

douglas.rochedo@gmail.com

These email addresses are for Google to notify you about any changes to your project.

[Next](#)

4 Finish

[Create](#)

[Cancel](#)

Audience

Contact Information

4 Finish

I agree to the [Google API Services: User Data Policy.](#) 

[Continue](#)

[Create](#)

[Cancel](#)

Metrics

You haven't configured any OAuth clients for this project yet.

Create OAuth client

Project Checkin

Client ID for Web application

[Delete](#)

Name *

The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.

i The domains of the URIs you add below will be automatically added to your [OAuth consent screen](#) as [authorized domains](#).

Authorized JavaScript origins [?](#)

For use with requests from a browser

URIs 1 *

[+ Add URI](#)

Authorized redirect URIs [?](#)

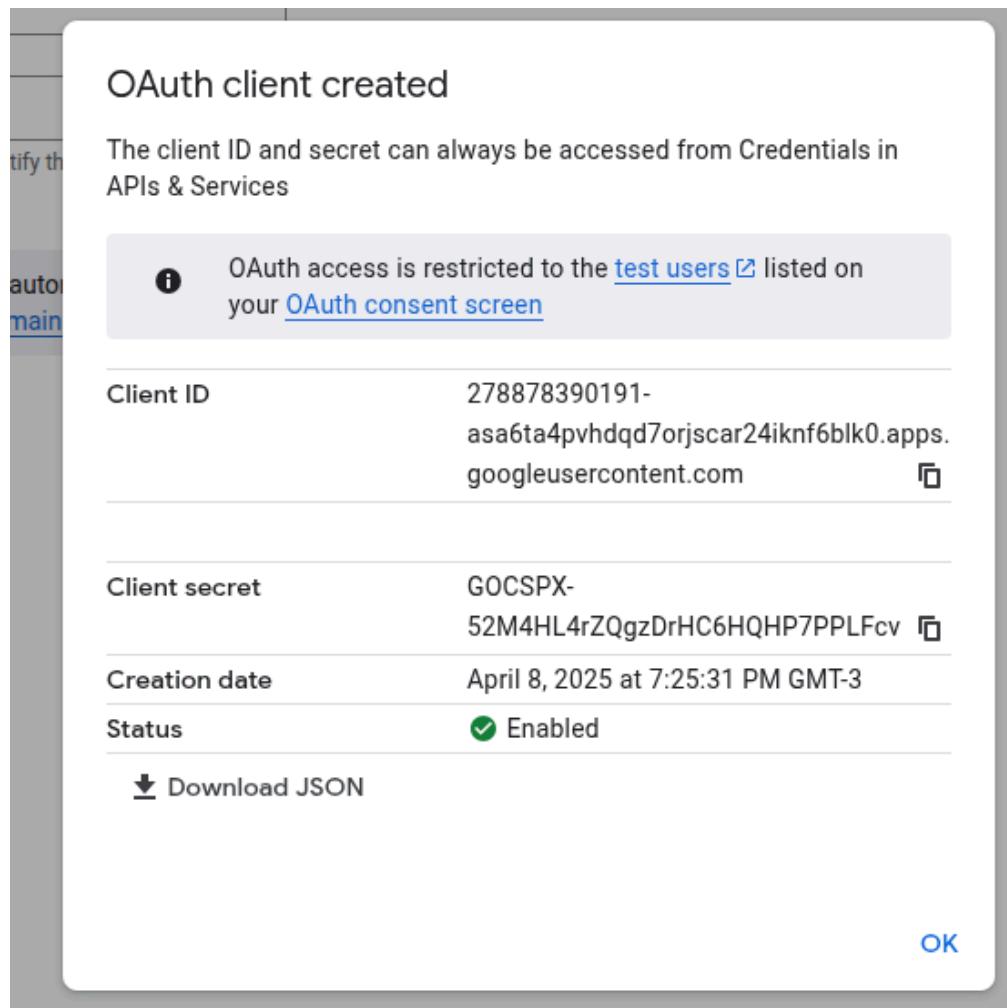
For use with requests from a web server

URIs 1 *

[+ Add URI](#)

Note: It may take 5 minutes to a few hours for settings to take effect

[Save](#) [Cancel](#)

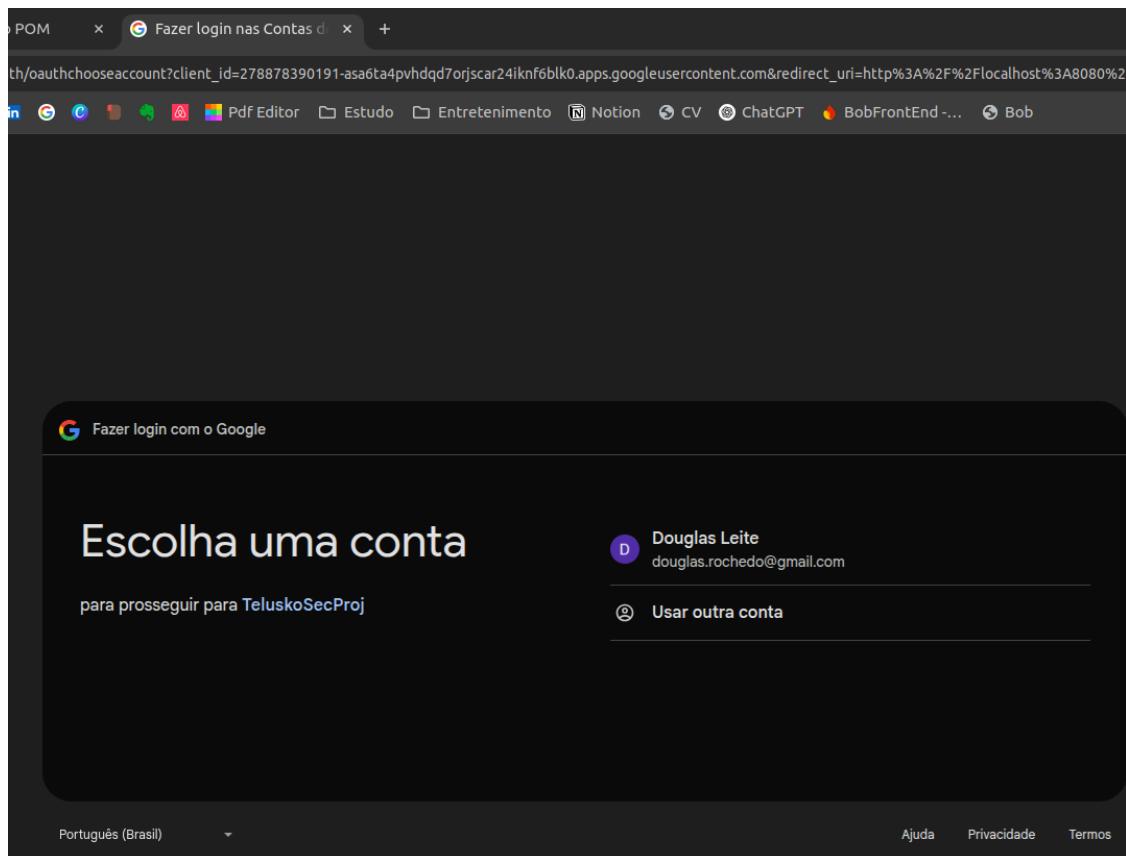


```
security.oauth2.client.client-id=278878390191-asa6ta4pvhdqd7orjscar24iknf6k  
security.oauth2.client.client-secret=GOCSPX-52M4HL4rZQgzDrHC6HQHP7PPI  
security.oauth2.client.access-token-uri=https://www.googleapis.com/oauth2/v  
security.oauth2.client.user-authorization-uri=https://accounts.google.com/o/oa  
security.oauth2.client.token-name=oauth_token  
security.oauth2.client.authentication-scheme=query  
security.oauth2.client.client-authentication-scheme=form  
security.oauth2.client.scope=profile email  
  
security.oauth2.resource.user-info-uri=https://www.googleapis.com/userinfo/v  
security.oauth2.resource.prefer-token-info=false
```

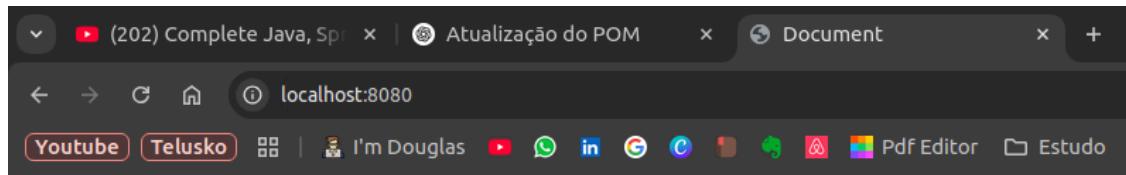
- Fizemos quase tudo o necessário para usar OAuth2
- Falta apenas criarmos um método no Controller para podermos acessar o token

```
@RequestMapping("user")
@ResponseBody
public Principal user(Principal principal) {
    return principal;
}
```

- Se rodarmos agora e acessarmos o endereço 8080



- Vemos a opção de acessar com Google
- Fazendo o Login com Google, temos acesso à aplicação novamente



Hello Aliens

[logout](#)

- E podemos ainda buscar pelo endpoint user que criamos e podemos ver os dados do usuário (no caso os meus hehe)

```
{  
    "authorities": [  
        {  
            "authority": "ROLE_USER"  
        }  
    ],  
    "details": {  
        "remoteAddress": "0:0:0:0:0:0:1",  
        "sessionId": "8ED5908F529CD41DF2FC0249E9D68B6E",  
        "tokenValue": "ya29.a0AZYKNzhN9Xm2s7Kp23s0tP0l5CHMNQc1eE0ISw0gwayg3D8nCUP9A1xeQ_y-mq2AJ2nJc0tmVjpX3XvY1vhIXVSFmbYzX",  
        "tokenType": "Bearer",  
        "decodedDetails": null  
    },  
    "authenticated": true,  
    "userAuthentication": {  
        "authorities": [  
            {  
                "authority": "ROLE_USER"  
            }  
        ],  
        "details": {  
            "id": "102305124665203835743",  
            "email": "douglas.rochedo@gmail.com",  
            "verified_email": true,  
            "name": "Douglas Leite",  
            "given_name": "Douglas",  
            "family_name": "Leite",  
            "picture": "https://lh3.googleusercontent.com/a/ACg8ocK7V0peyTDPuugk22m6J8nnS0FmPBsn-23ffEmx0wtcy4J0QQ=s96-c"  
        },  
        "authenticated": true,  
        "principal": "102305124665203835743",  
        "credentials": "N/A",  
        "name": "102305124665203835743"  
    },  
    "principal": "102305124665203835743",  
    "clientOnly": false,  
    "credentials": "",  
    "oauth2Request": {  
        "clientId": "278878390191-asaa6ta4pvhdqd7orjscar24iknf6blk0.apps.googleusercontent.com",  
        "scope": [],  
        "requestParameters": {  
        },  
        "resourceIds": [],  
        "authorities": [],  
        "approved": true,  
        "refresh": false,  
        "redirectUri": null,  
        "responseTypes": [],  
        "extensions": {}  
    },  
    "refreshTokenRequest": null,  
    "grantType": null  
},  
    "name": "102305124665203835743"  
}
```

▼ Microservices

- Para entender microserviços, voltemos para 1970
- Na época, os computadores existiam para resolver problemas simples, científicos e acadêmicos
- Com o tempo, eles começaram a servir mais para propósitos comerciais
- Hoje em dia, tudo existe e subsiste com a ajuda de softwares, aplicações
- Graças a isso, as aplicações foram ficando cada vez maiores, fazendo várias coisas diferentes



- Quando pensamos na construção de um projeto como esse, separamos o software em alguns módulos e distribuimos para algumas equipes
- Existe também a diferença de plataformas: criamos aplicações para um ambiente específico, seja browser, cloud, mobile, desktop, etc
- Para colocar todos esses serviços em uma aplicação, é um pacote muito grande
- Isso chamamos de Aplicação **Monolito**
- Numa aplicação monolito, temos a vantagem de termos uma aplicação só, menos complexa e com deploy facilitado
- Em contrapartida, temos alguns outros problemas:
 1. Team dependencies → um time depende do outro para tudo, deploy, versionamento, etc.
 2. Escalabilidade → nem todos os serviços precisam escalar juntos. Por mais que consigamos, sim, escalar sistemas monolitos, acabamos escalando serviços desnecessários
 3. Tecnologia → precisamos ter sempre uma única tecnologia em toda a aplicação, sem a possibilidade de diversificar ao linguagem de programação, por exemplo, dependendo do uso

- A ideia, portanto, é separar cada um desses serviços em **Microsserviços**, entidades menores que sejam
 - Self Contained - independentes
 - Deployadas independentemente
 - Escaláveis separadamente



- Microsserviços, contudo, também trazem alguns novos desafios como
 - Comunicação → eles tem que se descobrir, se conhecer e serem capazes de se comunicar de forma equilibrada e balanceada
 - Tolerância → como lidar caso um dos serviços pare?
 - Complexidade → são mais produtos, mais deploys, mais espaço para falha
- Microsserviços trazem mais desafios, mas trazem muitos benefícios também.

Prática

- Começaremos com uma aplicação exemplo Quizapp do Telusko que vamos clonar (vamos não, não tem essa opção)
- Fiquei com preguiça de replicar tudo isso sobre microsserviços. Vou deixar o vídeo tocando e depois vou reproduzir o outro vídeo do cara do Wiggle.
- Mudei de ideia, vai ser daora, vamos em frente
- Ele vai criar cada aplicação com um banco de dados separado
- Vai criar também o API Gateway, para gerenciar a comunicação
- E está usando ferramentas de Discovery

Question Service

- O primeiro serviço será o de questões
- Vamos criá-lo com Spring Initializr usando
 - Maven
 - Java
 - Spring Web, PostgreSQL Driver, Lombok, DevTools, Data JPA, **OpenFeign e Eureka Discovery Client**
- Como usamos Spring Data JPA, o software não roda sem um banco, mas já temos o projeto criado
- Vamos, por enquanto, comentar as dependências de Eureka e OpenFeign

```
<!-- <dependency>
    <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-openfeign</artifactId>
    </dependency> -->
```

- No tutorial, ele copiou do monolito e jogou no serviço
- Como não temos o monolito, vamos copiar olhando pra tela
- Vamos começar criando o esqueleto de cada coisa

```
package com.microservices.question_service.model;

public class Question {
    private Long id;
}

--

package com.microservices.question_service.dao;

import org.springframework.data.jpa.repository.JpaRepository;

import com.microservices.question_service.model.Question;
```

```

public interface QuestionDAO extends JpaRepository<Question, Long> {
}

--


package com.microservices.question_service.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import com.microservices.question_service.dao.QuestionDAO;

@Component
public class QuestionService {
    @Autowired
    private QuestionDAO repository;
}

--


@RestController
@RequestMapping("question")
public class QuestionController {
    @Autowired
    QuestionService service;
}

```

- Vamos primeiro alterar o Controller

```

@RestController
@RequestMapping("question")
public class QuestionController {
    @Autowired
    QuestionService service;

    @GetMapping("allQuestions")
    public ResponseEntity<List<Question>> getAllQuestions() {
        return service.getAllQuestions();
    }

    @GetMapping("category/{category}")
    public ResponseEntity<List<Question>> getQuestionsByCategory(@PathVariable

```

```

        return service.getQuestionsByCategory(category);
    }

    @PostMapping("add")
    public ResponseEntity<String> addQuestion(@RequestBody Question question) {
        return service.addQuestion(question);
    }
}

```

- Eu criei os métodos no Service com Ctrl+.

```

@Component
public class QuestionService {
    @Autowired
    private QuestionDAO repository;

    public ResponseEntity<List<Question>> getAllQuestions() {
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException("Unimplemented method 'getAllQuestions'");
    }

    public ResponseEntity<List<Question>> getQuestionsByCategory(String category) {
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException("Unimplemented method 'getQuestionsByCategory'");
    }

    public ResponseEntity<String> addQuestion(Question question) {
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException("Unimplemented method 'addQuestion'");
    }
}

```

- Agora vamos ajustar o QuestionService

```

@Component
public class QuestionService {
    @Autowired
    private QuestionDAO repository;

    public ResponseEntity<List<Question>> getAllQuestions() {
        try {
            return new ResponseEntity<>(repository.findAll(), HttpStatus.OK);
        }
    }
}

```

```

    } catch (Exception e) {
        e.printStackTrace();
    }

    return new ResponseEntity<>(new ArrayList<>(), HttpStatus.BAD_REQUEST);
}

public ResponseEntity<List<Question>> getQuestionsByCategory(String category) {
    try {
        return new ResponseEntity<>(repository.findByCategory(category), HttpStatus.OK);
    } catch (Exception e) {
        e.printStackTrace();
    }

    return new ResponseEntity<>(new ArrayList<>(), HttpStatus.BAD_REQUEST);
}

public ResponseEntity<String> addQuestion(Question question) {
    repository.save(question);
    return new ResponseEntity<>("success", HttpStatus.CREATED);
}
}

```

- Tive que criar manualmente (pelo menos ajustar o retorno) do findByCategory, mas o repositório está pronto

```

public interface QuestionDAO extends JpaRepository<Question, Long> {

    List<Question> findByCategory(String category);
}

```

- Agora vamos ajustar o próprio Question

```

@Data
@Entity
public class Question {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String questionTitle;
    private String option1;
    private String option2;
}

```

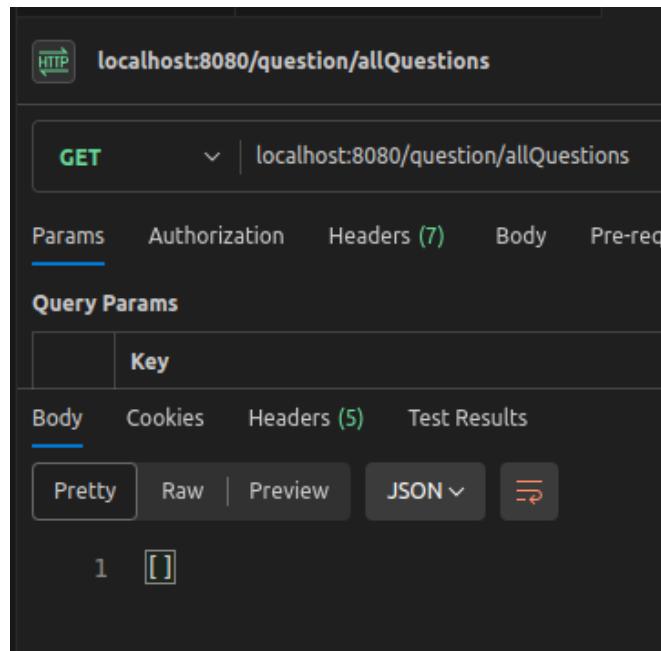
```
private String option3;  
private String option4;  
private String rightAnswer;  
private String difficultyLevel;  
private String category;  
}
```

- Finalmente, precisamos criar o application.properties

```
spring.application.name=question-service
```

```
spring.datasource.driver-class-name=org.postgresql.Driver  
spring.datasource.url=jdbc:postgresql://localhost:5432/questiondb  
spring.datasource.username=sa  
spring.datasource.password=fiscal@123  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

- Agora já conseguimos rodar o código!!
- Vamos testar o post e o get pelo postman



```
{  
    "questionTitle": "Which Java keyword is used to create a subclass?",  
    "option1": "class",  
    "option2": "interface",
```

```
"option3": "extends",
"option4": "implements",
"rightAnswer": "extends",
"difficultyLevel": "Easy",
"category": "Java"
}
```

The screenshot shows the Postman application interface. At the top, there is a header bar with the URL `localhost:8080/question/add`. Below this, a POST request is selected. The 'Body' tab is active, showing a JSON payload. The JSON content is identical to the one shown in the code block above. The 'Pretty' button is highlighted, indicating the current view. At the bottom, a success message '1 success' is displayed.

```
1 {
2     "questionTitle": "Which Java keyword is used to create a subcl
3     "option1": "class",
4     "option2": "interface",
5     "option3": "extends",
6     "option4": "implements",
7     "rightAnswer": "extends",
8     "difficultyLevel": "Easy",
9     "category": "Java"
10 }
```

Body Cookies Headers (5) Test Results
Pretty Raw | Preview Text ▾
1 success

The screenshot shows the Postman interface with a GET request to `localhost:8080/question/allQuestions`. The response body is displayed in JSON format:

```

1  [
2   {
3     "id": 1,
4     "questionTitle": "Which Java keyword is used to create a s",
5     "option1": "class",
6     "option2": "interface",
7     "option3": "extends",
8     "option4": "implements",
9     "rightAnswer": "extends",
10    "difficultyLevel": "Easy",
11    "category": "Java"
12  }
13 ]

```

- Essa estrutura do QuestionService foi desenvolvida pensando na estrutura de monolito do projeto. Ela funciona, mas há algumas integrações a mais que teremos que fazer para que consiga se comunicar de forma externa com os outros serviços
- A primeira funcionalidade que vamos implementar é a de gerar um Quiz, ou seja, de selecionar um conjunto de questões a partir de uma categoria e um número de perguntas
- Começaremos pelo Controller

```

@.GetMapping("generateQuiz")
public ResponseEntity<List<Long>> getQuestionsForQuiz(@RequestParam String
    @RequestParam String numQuestions) {
    return service.getQuestionsForQuiz(category, numQuestions);
}

```

- Criando esse método no service, usaremos um método que ainda não foi implementado, o de encontrar questões aleatórias por categoria

```

public ResponseEntity<List<Long>> getQuestionsForQuiz(String category, String r
    return new ResponseEntity<>(
        repository.findRandomQuestionsByCategory(category, Integer.valueOf(numQu
            .map(q → q.getId()).toList(),
        HttpStatus.OK);
    }
}

```

- Finalmente, vamos implementar a query

```

@Query(value = "SELECT * FROM question q WHERE q.category= :category ORDER BY
List<Question> findRandomQuestionsByCategory(String category, Integer numQ)

```

- Em seguida, vamos implementar outra funcionalidade
- Quando já temos um quiz com uma lista de questões, precisamos buscar essas questões no QuestionService
- Por isso, criaremos um POST que recebe a lista de ids e retorna as questões

```

// Controller
@PostMapping("getQuestions")
public ResponseEntity<List<Question>> getQuestionsById(@RequestBody List<L
    return service.getQuestionsById(questionIdList);
}

// Service
public ResponseEntity<List<Question>> getQuestionsById(List<Long> questionIdl
    List<Question> questions = questionIdList.stream().map(qId → repository.findBy

    return new ResponseEntity<>(questions, HttpStatus.OK);
}

```

- Finalmente, criaremos o método que retorna a pontuação final de um Quiz
- Novamente, devido à quantidade de dados chegando, usaremos um POST, e retornaremos apenas o resultado baseado em uma lista de respostas
- Começaremos criando o POJO da Resposta

```

@Data
@NoArgsConstructor
public class Response {
    private Long questionId;
}

```

```
    private String response;  
}
```

- Em seguida faremos o Controller

```
// getScore  
    @PostMapping("getScore")  
    public ResponseEntity<Integer> getScore(@RequestBody List<Response> responses) {  
        return service.getScore(responses);  
    }
```

- E calcularemos a nota no service

```
public ResponseEntity<Integer> getScore(List<Response> responses) {  
    return new ResponseEntity<>((int) responses.stream().map(res → {  
        Optional<Question> question = repository.findById(res.getQuestionId());  
        return question.isPresent() && question.get().getRightAnswer() == res.getRightAnswer()  
    }).filter(r → r == true).count(), HttpStatus.OK);  
}
```

- Finalmente, criamos tudo. Agora vamos testar!
- Criamos mais 2 questões, para facilitar os testes

```
[  
  {  
    "id": 1,  
    "questionTitle": "Which Java keyword is used to create a subclass?",  
    "option1": "class",  
    "option2": "interface",  
    "option3": "extends",  
    "option4": "implements",  
    "rightAnswer": "extends",  
    "difficultyLevel": "Easy",  
    "category": "Java"  
  },  
  {  
    "id": 2,  
    "questionTitle": "In Java, what is the default value of an uninitialized boolean variable?",  
    "option1": "true",  
    "option2": "false",  
    "option3": "0",  
    "option4": "null"  
  }]
```

```

    "option4": "null",
    "rightAnswer": "false",
    "difficultyLevel": "Hard",
    "category": "Java"
},
{
    "id": 3,
    "questionTitle": "In Java, what is the default value of an uninitialized integer variable?",
    "option1": "true",
    "option2": "false",
    "option3": "0",
    "option4": "null",
    "rightAnswer": "0",
    "difficultyLevel": "Medium",
    "category": "Java"
}
]

```

- Vamos tentar criar um Quiz

GET | localhost:8080/question/generateQuiz?category=Java&numQuestions=2

Params • Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

Key	Value
<input checked="" type="checkbox"/> category	Java
<input checked="" type="checkbox"/> numQuestions	2
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw | Preview JSON

```

1 [
2   3,
3   1
4 ]

```

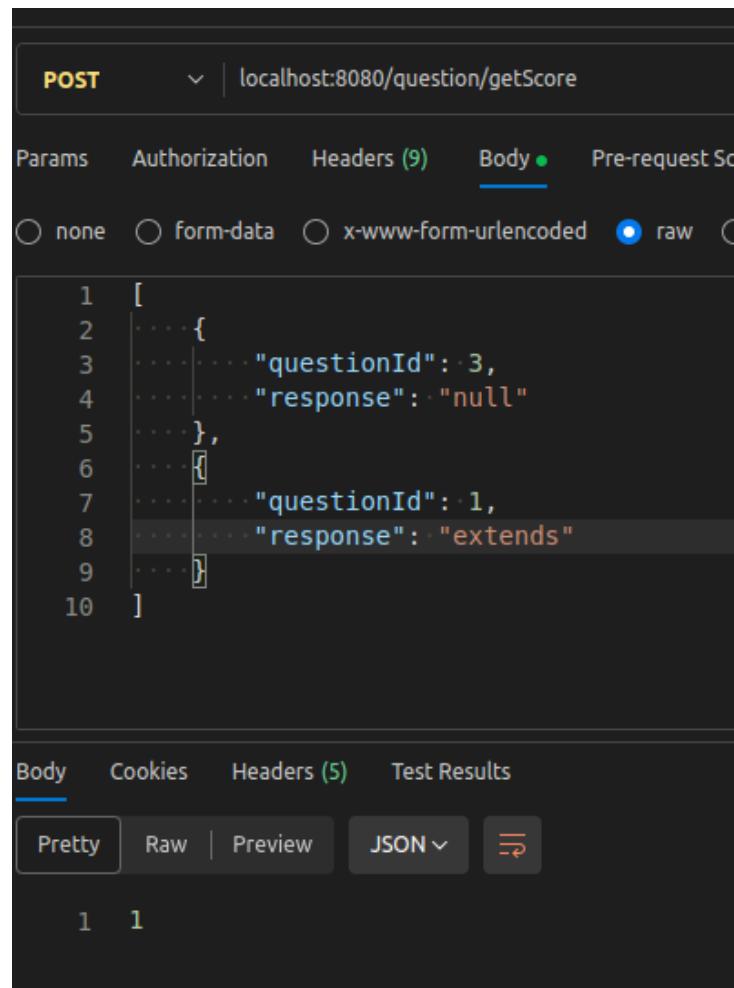
- Vamos tentar buscar essas perguntas

The screenshot shows a Postman interface with a POST request to `localhost:8080/question/getQuestions`. The request body is set to `raw` JSON and contains the following data:

```
1 [3,1]
2
```

```
1 [
2   {
3     "id": 3,
4     "questionTitle": "In Java, what is the default value of an uninitialized integer variable?",
5     "option1": "true",
6     "option2": "false",
7     "option3": "0",
8     "option4": "null",
9     "rightAnswer": "0",
10    "difficultyLevel": "Medium",
11    "category": "Java"
12  },
13  {
14    "id": 1,
15    "questionTitle": "Which Java keyword is used to create a subclass?",
16    "option1": "class",
17    "option2": "interface",
18    "option3": "extends",
19    "option4": "implements",
20    "rightAnswer": "extends",
21    "difficultyLevel": "Easy",
22    "category": "Java"
23  }
24 ]
```

- Agora vamos mandar as respostas e ver o resultado



- Ele explicou como subir várias instâncias do QuestionService pelo IntelliJ, vamos rodar com vários terminais mesmo rs

Quiz Service

- O Quiz Service vai ser um projeto aparte que terá as mesmas dependências do QuestionService, inclusive aquelas que foram comentadas

```
<!-- <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency> -->
```

- O Telusko só copiou e colou tudo do monolito pra cá, eu não tenho o monolito então vamos fazer cada coisa separada
- Começando pelo properties:

```
spring.application.name=quiz_service

spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.url=jdbc:postgresql://localhost:5432/quizdb
spring.datasource.username=sa
spring.datasource.password=fiscal@123
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

- Agora o Quiz propriamente dito

```
public class Quiz {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;

    @ManyToMany
    private List<Question> questions;
}
```

- Já vemos aqui o detalhe de que o Quiz possui uma lista de Questions. Isso porque será necessário que ele tenha acesso às informações da questão
- O Telusko criou um QuestionWrapper que não tem a resposta, e usa ele no Quiz, mas não faz muito sentido para mim...
- Vamos trazer o Question pra cá

```
@Data
@Entity
public class Question {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String questionTitle;
    private String option1;
```

```
private String option2;
private String option3;
private String option4;
private String rightAnswer;
private String difficultyLevel;
private String category;
}
```

- Agora vamos criar o Service, mas não vamos preenche-lo ainda

```
@Service
public class QuizService {

}
```

- Agora vamos para o Controller

```
@RestController
@RequestMapping("quiz")
public class QuizController {

    @Autowired
    private QuizService service;

    @PostMapping("create")
    public ResponseEntity<String> createQuiz(@RequestParam String category, @Re
        @RequestParam String title) {
        return service.createQuiz(category, numQ, title);
    }

    @GetMapping("getQuestions/{id}")
    public ResponseEntity<List<Question>> getQuizQuestions(@PathVariable Long id)
        return service.getQuizQuestions(id);
    }

    @PostMapping("submitResponses/{id}")
    public ResponseEntity<Integer> submitQuizResponses(@PathVariable Long id, @
        return service.calculateResult(id, responses);
    }

}
```

- Já vimos que vamos precisar do modelo de Response também, vamos puxá-lo

```
@Data
@NoArgsConstructor
public class Response {
    private Long questionId;
    private String response;
}
```

- E agora já temos alguns métodos no nosso Service

```
@Service
public class QuizService {

    public ResponseEntity<String> createQuiz(String category, int numQ, String title)
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException("Unimplemented method 'createQu
    }

    public ResponseEntity<List<Question>> getQuizQuestions(Long id) {
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException("Unimplemented method 'getQuizQ
    }

    public ResponseEntity<Integer> calculateResult(Long id, List<Response> respons
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException("Unimplemented method 'calculatel
    }

}
```

- Vamos fazer algumas alterações nele, mas vamos começar criando o repositório de Quiz

```
public interface QuizRepository extends JpaRepository<Quiz, Long> {
}
```

- Agora começaremos as alterações no Service
- A primeira coisa será incluir o repositório

```

@Service
public class QuizService {

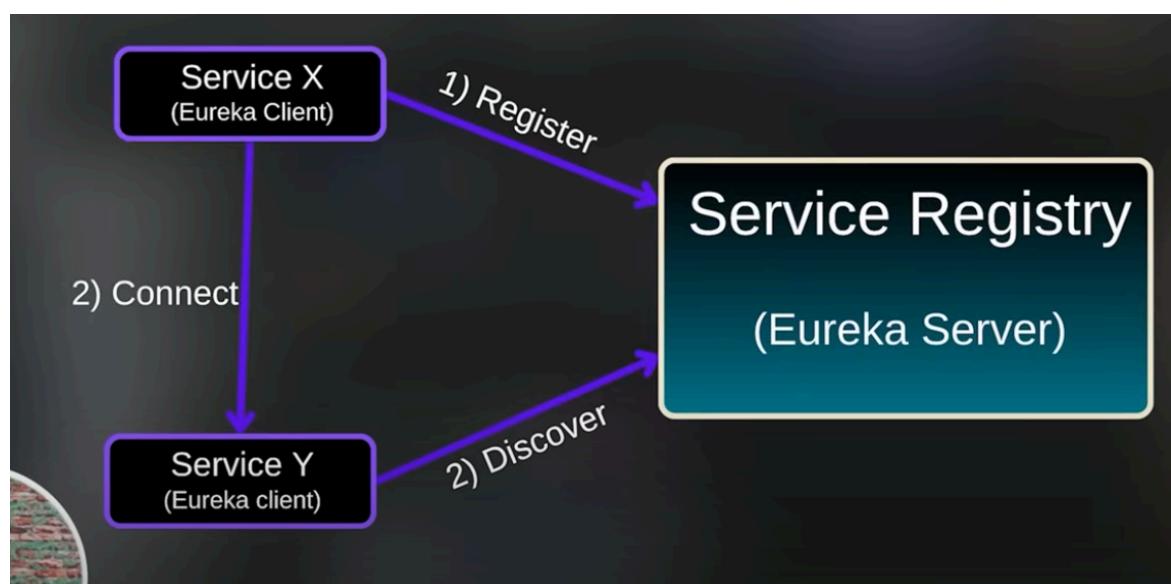
    @Autowired
    private QuizRepository repository;

```

- Começaremos com o `createQuiz`
- Aqui já temos um problema: precisamos das questões. Como faremos isso?

Feign Client & Eureka Service Discovery

- As questões são acessíveis via URL, no endpoint do `QuestionService`
- Acessaremos esse endpoint via `RestTemplate`
- Por padrão, acessaríamos via `localhost:8080/question/template`
- Em projetos mais sérios, poderíamos colocar esse endpoint na configuração
- Contudo, nesse caso, nem isso resolveria, porque estamos tratando de microserviços
- Podemos ter várias instâncias do `QuestionService` rodando em várias portas, em vários servidores diferentes, como lidar com isso?
- Aqui vai entrar o **Feign Client**
- Ele vai executar os requests HTTP, mas de forma mais declarativa
- Também usaremos o **Eureka Service Discovery**, que vai descobrir os endpoints que o Feign pode acessar



- Portanto, nosso primeiro passo será criar o nosso **Netflix Eureka Discovery Server**
- Vamos criar mais um projeto! O service-registry
- Precisaremos apenas de 2 dependências: Spring Web e Eureka Server (vamos colocar também o Dev Tools)
- Para fazer nosso servidor de Discovery, precisamos fazer apenas alguns ajustes no SpringApplication, i.e., apenas a annotation EnableEurekaServer

```
@SpringBootApplication
@EnableEurekaServer
public class ServiceRegistryApplication {

    public static void main(String[] args) {
        SpringApplication.run(ServiceRegistryApplication.class, args);
    }

}
```

- E no properties:

```
spring.application.name=service-registry
server.port=8761

eureka.instance.hostname=localhost
eureka.client.fetch-registry=false
eureka.client.register-with-eureka=false
```

- Essas configurações farão com que o nosso servidor de discovery não se auto-registre e não busque os registros sozinho
- A porta é padrão, mas faz bem ressaltarmos
- Já podemos rodar o software e roda direitinho
- Podemos ainda acessar localhost:8761 no browser e veremos uma tela padrão

The screenshot shows the Spring Eureka dashboard. At the top, it displays the current time as 2025-04-16T16:17:19 -0300 and uptime as 00:08. A warning message in red text states: "EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE." Below this, the "DS Replicas" section shows a table with one row: "No instances available". The "General Info" section contains a table with various system metrics like total-available-memory, num-of-cpus, and server-uptime.

- A informação mais importante para nós no momento é o **Instances currently registered**

This screenshot shows the "Instances currently registered with Eureka" section. It has two tabs at the top: "Application" (which is selected) and "AMIs". Below the tabs, the message "No instances available" is displayed.

- Por enquanto não temos nenhuma, mas em breve isso vai mudar
- Vamos deixar o Eureka rodando e vamos fazer nosso QuestionService se registrar nele
- Para isso, vamos descomentar a dependência do Eureka Client nele

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
<!-- <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency> -->
```

- Apenas com isso já podemos rodar o projeto e ele irá se registrar pela porta padrão do Eureka Server

Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
QUESTION-SERVICE	n/a (1)	(1)	UP (1) - douglasleite-thinkpad:question-service
General Info			

- O Telusko não tinha definido o application name dele, e por isso aparecia com Unknown, mas como nós já tínhamos, funcionou direto
- Podemos ainda rodar novamente a aplicação em outra porta e ver o que acontece
- Vamos rodar via linha de comando para poder alterar a porta

```
/usr/bin/env /usr/lib/jvm/java-17-openjdk-amd64/bin/java @/tmp/cp_7mqh1z1kcc92
```

- E ele também deve se registrar como uma aplicação semelhante no Eureka

Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
QUESTION-SERVICE	n/a (2)	(2)	UP (2) - douglasleite-thinkpad:question-service:8081 , douglasleite-thinkpad:question-se
General Info			

- Agora precisamos fazer com que o QuizService consiga se comunicar com o QuestionService via Eureka Server, para isso usaremos o Feign
- No Quiz, criaremos uma interface Feign, mas antes precisaremos descomentar a dependencia do Feign (aproveitaremos para descomentar o Eureka também)

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

- E agora a interface, em que o principal ponto é a annotation FeignClient, que mencionará qual o serviço que acessaremos

```
@FeignClient("QUESTION-SERVICE")
public interface QuestionFeignInterface {

}
```

- Na classe, precisaremos da definição dos métodos que estão no QuestionController que precisaremos acessar, com todo o endereço do endpoint

```
@FeignClient("QUESTION-SERVICE")
public interface QuestionFeignInterface {

    // generate
    @GetMapping("question/generateQuiz")
    public ResponseEntity<List<Long>> getQuestionsForQuiz(@RequestParam String category,
                                                               @RequestParam Integer numQuestions);

    // getQuestions (questionid)
    @PostMapping("question/getQuestions")
    public ResponseEntity<List<Question>> getQuestionsById(@RequestBody List<Long> questionIdList);

    // getScore
    @PostMapping("question/getScore")
    public ResponseEntity<Integer> getScore(@RequestBody List<Response> responses);
}
```

- Agora, no nosso QuizService, vamos instanciar e utilizar essa interface para buscar os dados que precisamos

```
@Service
public class QuizService {

    @Autowired
    private QuizRepository repository;

    @Auto
    private QuestionFeignInterface questionInterface;

    public ResponseEntity<String> createQuiz(String category, int numQ, String title) {
        List<Long> questionIdList = questionInterface.getQuestionsForQuiz(category, numQ);
        ...
    }
}
```

```
        return null;  
    }  
}
```

- Aqui já identificamos um problema que, no código do QuizService, o Quiz recebe a lista de Questions, mas nosso QuestionService só nos dá a lista de IDs
- Vamos fazer a alteração necessária no Quiz

```
@Data  
@Entity  
public class Quiz {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    private String title;  
  
    @ElementCollection  
    private List<Long> questionIdList;  
}
```

- O ElementCollection nos permitirá armazenar essa lista no banco
- Podemos também remover o Entity do Question

```
@Data  
public class Question {  
    private Long id;  
    private String questionTitle;  
    private String option1;  
    private String option2;  
    private String option3;  
    private String option4;  
    private String rightAnswer;  
    private String difficultyLevel;  
    private String category;  
}
```

- Agora podemos concluir nosso método no service, finalmente

```
public ResponseEntity<String> createQuiz(String category, int numQ, String title) {  
    List<Long> questionIdList = questionInterface.getQuestionsForQuiz(category, n
```

```

Quiz quiz = new Quiz();
quiz.setQuestionIdList(questionIdList);
quiz.setTitle(title);

repository.save(quiz);

return new ResponseEntity<>("Success", HttpStatus.OK);
}

```

- Antes de rodarmos, precisamos apenas fazer algumas configurações
- Primeiro vamos alterar a porta para que não conflite com o QuestionService

```

spring.application.name=quiz_service
server.port=8090

spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.url=jdbc:postgresql://localhost:5432/quizdb
spring.datasource.username=sa
spring.datasource.password=fiscal@123
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

```

- Agora vamos adicionar uma annotation no application

```

@SpringBootApplication
@EnableFeignClients
public class QuizServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(QuizServiceApplication.class, args);
    }

}

```

- Finalmente podemos rodar o código
- O serviço já deve aparecer no Eureka Server

Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
QUESTION-SERVICE	n/a (2)	(2)	UP (2) - douglasleite-thinkpad:question-service:8081 , douglasleite-thinkpad:question-service
QUIZ_SERVICE	n/a (1)	(1)	UP (1) - douglasleite-thinkpad:quiz_service:8090

- E já podemos testar a criação do nosso primeiro Quiz via QuizService com Postman

POST | localhost:8090/quiz/create?category=Java&numQ=2&title=First Quiz

Params • Authorization Headers (8) Body Pre-request Script Tests Settings

• none ○ form-data ○ x-www-form-urlencoded ○ raw ○ binary ○ GraphQL

Body Cookies Headers (5) Test Results

Pretty Raw | Preview Text ↴

1 Success

```
SELECT * FROM quiz LIMIT 100
```

	Q	Search Results	✉ 1	⟳
	Q	* id	title	varchar(255)
	Q	bigint		
	>	1	First Quiz	

```
SELECT * FROM quiz_question_id_list LIMIT 100
```

	Q	Search Results	✉ 1	⟳
	Q	* quiz	question_id_list	bigint
	Q	bigint		
	>	1	2	
	>	1	1	

- Uma dúvida importante é: como ele decidiu qual das instâncias foi chamada?
- Como não colocamos log, não conseguimos nem saber pelo terminal, mas entenderemos isso melhor depois com o load balancer

- Por enquanto vamos finalizar nosso QuizService, começando pelo getQuizQuestions

```
public ResponseEntity<List<Question>> getQuizQuestions(Long id) {
    Quiz quiz = repository.findById(id).get();
    List<Question> questions = questionInterface.getQuestionsById(quiz.getQuestio
        return new ResponseEntity<>(questions, HttpStatus.OK);
}
```

The screenshot shows a Postman interface with a GET request to `localhost:8090/quiz/getQuestions/1`. The response is displayed in a JSONpretty-printed format:

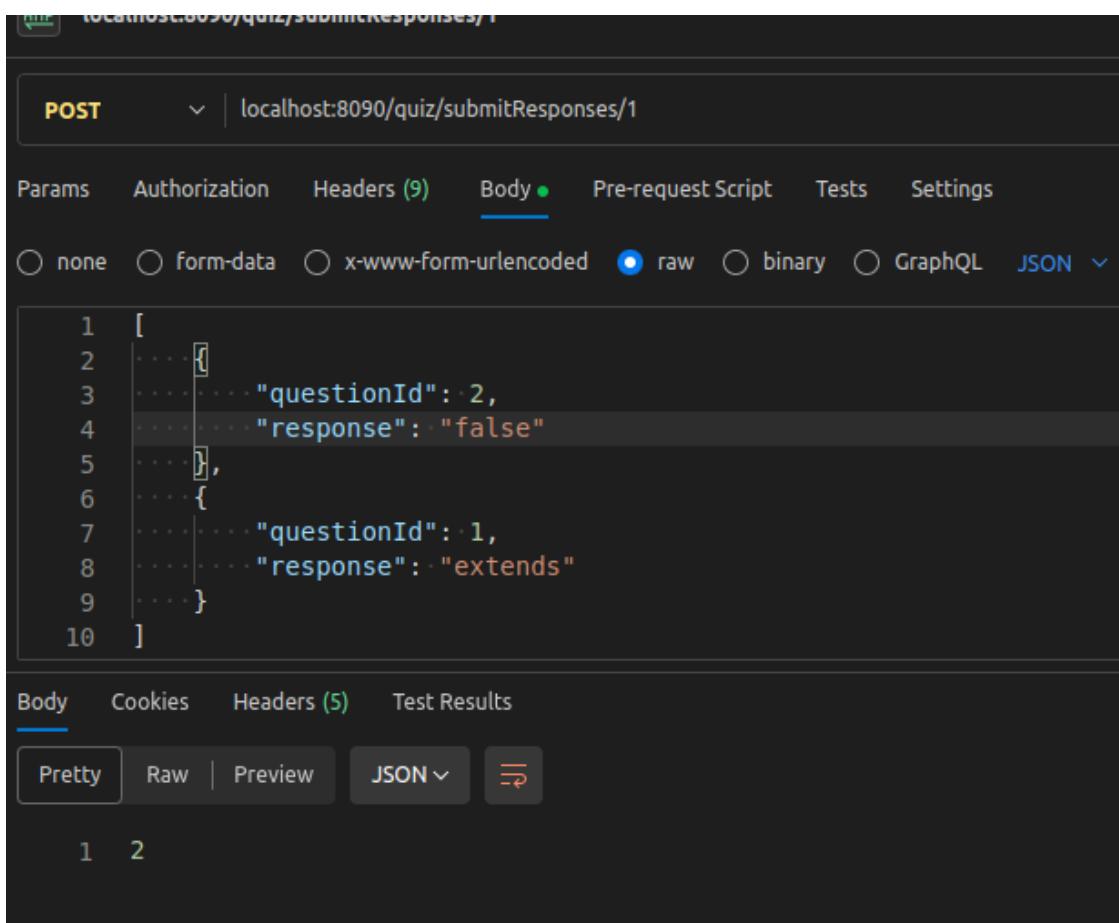
```

1 [
2   {
3     "id": 2,
4     "questionTitle": "In Java, what is the default value of an uninitialized boolean variable?",
5     "option1": "true",
6     "option2": "false",
7     "option3": "0",
8     "option4": "null",
9     "rightAnswer": "false",
10    "difficultyLevel": "Hard",
11    "category": "Java"
12  },
13  {
14    "id": 1,
15    "questionTitle": "Which Java keyword is used to create a subclass?",
16    "option1": "class",
17    "option2": "interface",
18    "option3": "extends",
19    "option4": "implements",
20    "rightAnswer": "extends",
21    "difficultyLevel": "Easy",
22    "category": "Java"
23  }
]

```

- E o calculateResult

```
public ResponseEntity<Integer> calculateResult(Long id, List<Response> responses) {
    Integer score = questionInterface.getScore(responses).getBody();
    return new ResponseEntity<>(score, HttpStatus.OK);
}
```



- É importante ressaltar que, apesar de não termos escrito muito código, esses métodos todos executaram comunicação com o QuestionService

Load Balancer

- A ideia de LoadBalancer é alguém que vai distribuir as requisições para um determinado Microserviço entre as instâncias desse Microserviço
- No nosso caso, o Quiz comunica com o QuestionService, mas temos 2 QuestionServices funcionando, alguém tem que determinar qual das duas instâncias será chamada
- O próprio Eureka Client traz uma dependência de Load Balancer, que fará essa escolha de forma alternante, ou seja, manda um request para um e o próximo pro outro
- Vamos tentar identificar a porta que está sendo utilizada no Question Controller usando uma classe Environment

```

@RestController
@RequestMapping("question")
public class QuestionController {
  @Autowired

```

QuestionService service;

@Autowired
Environment env;

```
// getQuestions (questionid)
@PostMapping("getQuestions")
public ResponseEntity<List<Question>> getQuestionsById(@RequestBody List<L
    System.out.println(env.getProperty("local.server.port"));
    return service.getQuestionsById(questionIdList);
}
```



Nem precisava né... era só olhar qual terminal logou, mas beleza, aprendemos algo novo

- Como tem o DevTools, as duas instâncias atualizaram (inclusive a que rodamos pelo terminal, bom saber)
- Agora podemos testar com o getQuestions do QuizService, e vemos a alternância nos logs

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
nt : DiscoveryClient_QUESTION-SERVICE/douglasleite-thinkpad:question-service:8081: registering service...
2025-04-16T18:05:29.275-03:00 INFO 3410364 --- [question-service] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebSe
rver : Tomcat started on port 8081 (http) with context path '/'
2025-04-16T18:05:29.275-03:00 INFO 3410364 --- [question-service] [ restartedMain] .s.c.n.e.s.EurekaAutoServiceRegistr
ation : Updating port to 8081
2025-04-16T18:05:29.284-03:00 INFO 3410364 --- [question-service] [foReplicator-%d] com.netflix.discovery.DiscoveryClie
nt : DiscoveryClient_QUESTION-SERVICE/douglasleite-thinkpad:question-service:8081 - registration status: 204
2025-04-16T18:05:29.287-03:00 INFO 3410364 --- [question-service] [ restartedMain] c.m.q.QuestionServiceApplication
: Started QuestionServiceApplication in 0.472 seconds (process running for 6125.892)
2025-04-16T18:05:29.288-03:00 INFO 3410364 --- [question-service] [ restartedMain] .ConditionEvaluationDeltaLogginglis
tener : Condition evaluation unchanged
2025-04-16T18:06:49.791-03:00 INFO 3410364 --- [question-service] [nio-8081-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]
: Initializing Spring DispatcherServlet 'dispatcherServlet'
2025-04-16T18:06:49.791-03:00 INFO 3410364 --- [question-service] [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet
: Initializing Servlet 'dispatcherServlet'
2025-04-16T18:06:49.792-03:00 INFO 3410364 --- [question-service] [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet
: Completed initialization in 1 ms
8081
8081
8081
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
2025-04-16T18:05:29.423-03:00 INFO 3407446 --- [question-service] [foReplicator-%d] com.netflix.discovery.DiscoveryClien
t : DiscoveryClient_QUESTION-SERVICE/douglasleite-thinkpad:question-service: registering service...
2025-04-16T18:05:29.424-03:00 INFO 3407446 --- [question-service] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebSe
rver : Tomcat started on port 8080 (http) with context path '/'
2025-04-16T18:05:29.424-03:00 INFO 3407446 --- [question-service] [ restartedMain] .s.c.n.e.s.EurekaAutoServiceRegistr
ation : Updating port to 8080
2025-04-16T18:05:29.431-03:00 INFO 3407446 --- [question-service] [foReplicator-%d] com.netflix.discovery.DiscoveryClie
nt : DiscoveryClient_QUESTION-SERVICE/douglasleite-thinkpad:question-service - registration status: 204
2025-04-16T18:05:29.444-03:00 INFO 3407446 --- [question-service] [ restartedMain] c.m.q.QuestionServiceApplication
: Started QuestionServiceApplication in 0.598 seconds (process running for 6279.821)
2025-04-16T18:05:29.445-03:00 INFO 3407446 --- [question-service] [ restartedMain] .ConditionEvaluationDeltaLogginglis
tener : Condition evaluation unchanged
2025-04-16T18:07:18.833-03:00 INFO 3407446 --- [question-service] [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]
: Initializing Spring DispatcherServlet 'dispatcherServlet'
2025-04-16T18:07:18.834-03:00 INFO 3407446 --- [question-service] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet
: Initializing Servlet 'dispatcherServlet'
2025-04-16T18:07:18.834-03:00 INFO 3407446 --- [question-service] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet
: Completed initialization in 0 ms
8080
8080
```

- No nosso caso, não faz muita diferença qual é chamado porque são idênticos e não há muita carga sobre nenhum dos dois, mas isso é relevante quando temos muita carga

API Gateway

- Até agora, temos apenas 2 microserviços e 3 instâncias ao todo
- Quando pensamos na comunicação entre eles, o Eureka e o Feign resolvem o problema de termos múltiplas instâncias de um único serviço
- Contudo, atualmente quando vamos acessar como usuários, temos que alternar a porta
- Ainda podemos levantar a problemática de que, sendo um usuário num software com autenticação, não é viável que ele tenha que se autenticar com cada microserviço
- Sim, estamos falando de backend, então haveria um frontend que poderia, em tese, conhecer todos os endpoints diferentes, mas mesmo isso não faz muito sentido
- Para isso, vamos criar um **API Gateway**, um serviço que servirá de “entry-point” ou ponto de entrada para o usuário
- Para isso, vamos criar um novo projeto
- As dependências serão apenas o Eureka Discovery Client (pois será um serviço) e Gateway (além, claro do DevTools)



Por algum motivo, quando ele usou o Gateway, era apenas Gateway, mas para funcionar da forma como veremos a seguir, tem que ser o Reactive Gateway

- Apenas o que já temos deve ser suficiente para que o serviço funcione como Gateway, graças à dependência que incluímos
- Faremos apenas algumas configurações no properties para que consiga subir sem conflito de porta e registrando adequadamente no Eureka

```
spring.application.name=api-gateway
server.port=8765
```

- O software já roda e devemos vê-lo no Eureka Server

Instances currently registered with Eureka

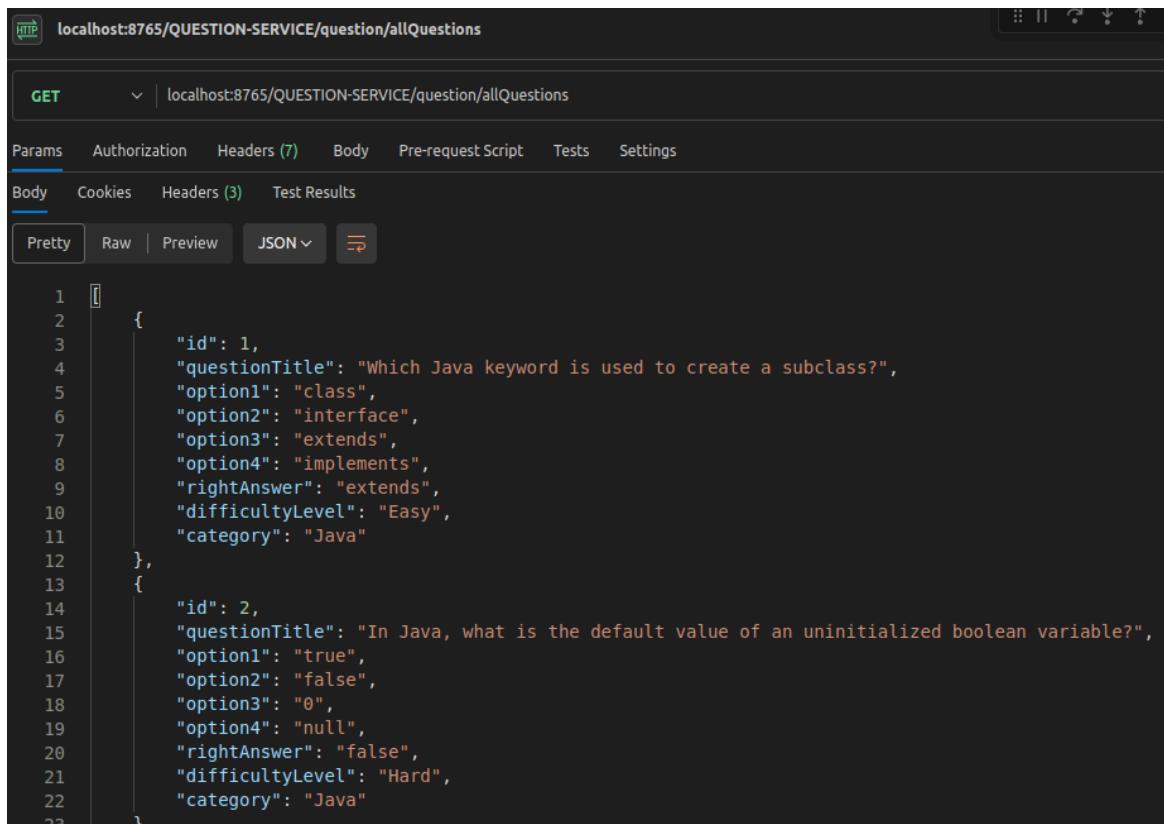
Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - douglasleite-thinkpad:api-gateway:8765
QUESTION-SERVICE	n/a (2)	(2)	UP (2) - douglasleite-thinkpad:question-service:8081 , douglasleite-thinkpad:question-service:8082
QUIZ_SERVICE	n/a (1)	(1)	UP (1) - douglasleite-thinkpad:quiz_service:8090

- A ideia agora é que consigamos acessar os endpoints do QuestionService e QuizService pelo endpoint 8765
- Não tem como o Gateway saber, por exemplo, que deve mapear o localhost:8080/question/allQuestions para localhost:8765/question/allQuestions sem nenhuma configuração prévia
- Contudo, o Gateway apenas pela dependÊncia, já consegue fazer o mapeamento com base no descobrimento feito com o Eureka
- Ou seja, se incluirmos o nome do serviço no endpoint, ele já deve conseguir fazer o resto
- Por padrão, essa funcionalidade fica desligada, precisamos habilitá-la no properties

```
spring.application.name=api-gateway
server.port=8765
```

```
spring.cloud.gateway.discovery.locator.enabled=true
```

- Agora devemos conseguir usá-lo como ponto de entrada:



```
1 [  
2   {  
3     "id": 1,  
4     "questionTitle": "Which Java keyword is used to create a subclass?",  
5     "option1": "class",  
6     "option2": "interface",  
7     "option3": "extends",  
8     "option4": "implements",  
9     "rightAnswer": "extends",  
10    "difficultyLevel": "Easy",  
11    "category": "Java"  
12  },  
13  {  
14    "id": 2,  
15    "questionTitle": "In Java, what is the default value of an uninitialized boolean variable?",  
16    "option1": "true",  
17    "option2": "false",  
18    "option3": "0",  
19    "option4": "null",  
20    "rightAnswer": "false",  
21    "difficultyLevel": "Hard",  
22    "category": "Java"  
23  }]
```



Depois de bater muito a cabeça aqui, percebi que o nome do QuizService não está QUIZ-SERVICE, e sim QUIZ_SERVICE

Parece que não muda nada, mas deu erro na hora de chamar, mesmo com o nome certo

Ele aparentemente até mapeou os endpoints desse cara, mas não conseguiu acessar, depois da pra tentar entender o motivo



Nessa hora, devido aos bugs, descobri essa propriedade que pode ser interessante quando pedimos ao Gateway para descobrir os endpoints sozinhos.

```
logging.level.org.springframework.cloud.gateway=DEBUG
```

Esse é o resultado

▼ Log grande

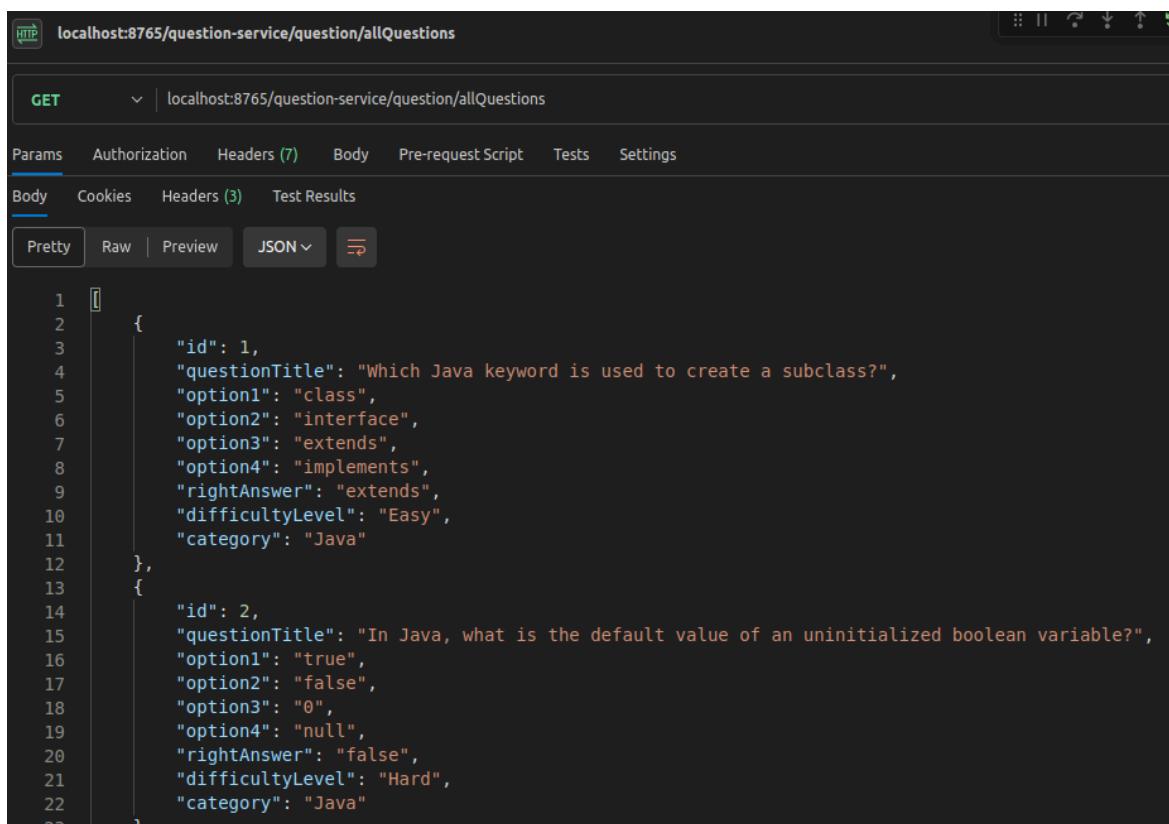
```
2025-04-16T18:45:56.080-03:00 INFO 3553311 --- [api-gateway] [ res
2025-04-16T18:45:56.081-03:00 INFO 3553311 --- [api-gateway] [ res
2025-04-16T18:45:56.206-03:00 INFO 3553311 --- [api-gateway] [ res
2025-04-16T18:45:56.359-03:00 INFO 3553311 --- [api-gateway] [ res
2025-04-16T18:45:56.359-03:00 INFO 3553311 --- [api-gateway] [ res
2025-04-16T18:45:56.359-03:00 INFO 3553311 --- [api-gateway] [ res
2025-04-16T18:45:56.360-03:00 INFO 3553311 --- [api-gateway] [ res
2025-04-16T18:45:56.388-03:00 WARN 3553311 --- [api-gateway] [ re
2025-04-16T18:45:56.406-03:00 INFO 3553311 --- [api-gateway] [ res
2025-04-16T18:45:56.414-03:00 WARN 3553311 --- [api-gateway] [ re
2025-04-16T18:45:56.420-03:00 INFO 3553311 --- [api-gateway] [ res
2025-04-16T18:45:56.422-03:00 INFO 3553311 --- [api-gateway] [ res
2025-04-16T18:45:56.422-03:00 INFO 3553311 --- [api-gateway] [ res
2025-04-16T18:45:56.423-03:00 INFO 3553311 --- [api-gateway] [ res
2025-04-16T18:45:56.444-03:00 INFO 3553311 --- [api-gateway] [ res
2025-04-16T18:45:56.445-03:00 INFO 3553311 --- [api-gateway] [ res
2025-04-16T18:45:56.445-03:00 INFO 3553311 --- [api-gateway] [ res
```

```
2025-04-16T18:45:56.446-03:00 INFO 3553311 --- [api-gateway] [ res
2025-04-16T18:45:56.446-03:00 INFO 3553311 --- [api-gateway] [ res
2025-04-16T18:45:56.446-03:00 INFO 3553311 --- [api-gateway] [ res
2025-04-16T18:45:56.447-03:00 INFO 3553311 --- [api-gateway] [ res
2025-04-16T18:45:56.447-03:00 INFO 3553311 --- [api-gateway] [foRe
2025-04-16T18:45:56.448-03:00 DEBUG 3553311 --- [api-gateway] [ re
2025-04-16T18:45:56.452-03:00 DEBUG 3553311 --- [api-gateway] [ re
2025-04-16T18:45:56.457-03:00 DEBUG 3553311 --- [api-gateway] [ re
2025-04-16T18:45:56.457-03:00 INFO 3553311 --- [api-gateway] [foRe
2025-04-16T18:45:56.458-03:00 DEBUG 3553311 --- [api-gateway] [ re
2025-04-16T18:45:56.458-03:00 DEBUG 3553311 --- [api-gateway] [ re
2025-04-16T18:45:56.459-03:00 DEBUG 3553311 --- [api-gateway] [ re
2025-04-16T18:45:56.459-03:00 DEBUG 3553311 --- [api-gateway] [ re
2025-04-16T18:45:56.460-03:00 DEBUG 3553311 --- [api-gateway] [ re
2025-04-16T18:45:56.460-03:00 DEBUG 3553311 --- [api-gateway] [ re
2025-04-16T18:45:56.462-03:00 INFO 3553311 --- [api-gateway] [ res
2025-04-16T18:45:56.463-03:00 INFO 3553311 --- [api-gateway] [ res
2025-04-16T18:45:56.475-03:00 DEBUG 3553311 --- [api-gateway] [ re
2025-04-16T18:45:56.476-03:00 DEBUG 3553311 --- [api-gateway] [ re
2025-04-16T18:45:56.477-03:00 DEBUG 3553311 --- [api-gateway] [ re
2025-04-16T18:45:56.477-03:00 DEBUG 3553311 --- [api-gateway] [ re
2025-04-16T18:45:56.478-03:00 DEBUG 3553311 --- [api-gateway] [ re
2025-04-16T18:45:56.478-03:00 DEBUG 3553311 --- [api-gateway] [ re
2025-04-16T18:45:56.478-03:00 DEBUG 3553311 --- [api-gateway] [ re
2025-04-16T18:45:56.479-03:00 DEBUG 3553311 --- [api-gateway] [ re
2025-04-16T18:45:56.480-03:00 INFO 3553311 --- [api-gateway] [ res
2025-04-16T18:45:56.482-03:00 INFO 3553311 --- [api-gateway] [ res
```

- Para podermos remover completamente o nome do serviço da URL, são necessárias algumas configurações
- Nesse momento, vamos apenas configurar para que fique em minúsculas, pelo visual da URL
- Usaremos essa configuração

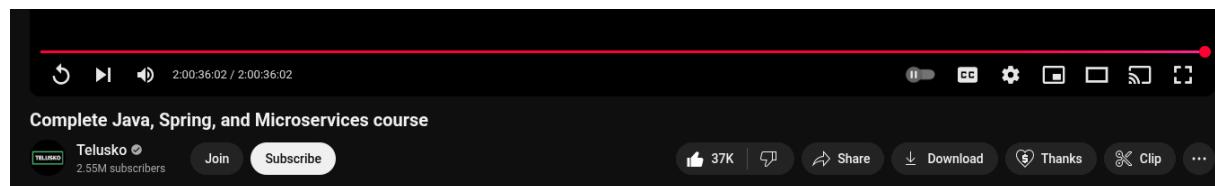
```
spring.application.name=api-gateway
server.port=8765
```

```
spring.cloud.gateway.discovery.locator.enabled=true  
spring.cloud.gateway.discovery.locator.lower-case-service-id=true
```



The screenshot shows a Postman interface with a GET request to `localhost:8765/question-service/question/allQuestions`. The response body is a JSON array containing two objects, each representing a question:

```
[  
  {  
    "id": 1,  
    "questionTitle": "Which Java keyword is used to create a subclass?",  
    "option1": "class",  
    "option2": "interface",  
    "option3": "extends",  
    "option4": "implements",  
    "rightAnswer": "extends",  
    "difficultyLevel": "Easy",  
    "category": "Java"  
  },  
  {  
    "id": 2,  
    "questionTitle": "In Java, what is the default value of an uninitialized boolean variable?",  
    "option1": "true",  
    "option2": "false",  
    "option3": "0",  
    "option4": "null",  
    "rightAnswer": "false",  
    "difficultyLevel": "Hard",  
    "category": "Java"  
  }  
]
```



ACABOOOOOOOUUU !!!!!

Data de finalização 16/04/2025

EXATOS 6 MESES DEPOOOOOIS

182 DIAS SOFRIDOS

Mas muito produtivos

Ótimo curso, fenomenal, conhecimento demaaaaais

Principalmente a parte de Microservices, simplesmente incrível!

A parte sobre Spring Core Framework também foi fenomenal

Consigo tranquilamente criar um curso com tudo isso kkkkk

Muito bom!

ACABEI: https://youtu.be/4XTsAAHW_Tc?si=seZEgpQmLGGDWGAt&t=174962