

Desenvolvimento de APIs REST

05 - Relacionamento entre entidades

- Banco de Dados Embedded
- Relacionamentos Embedded
- Exceções Enum
- Herança

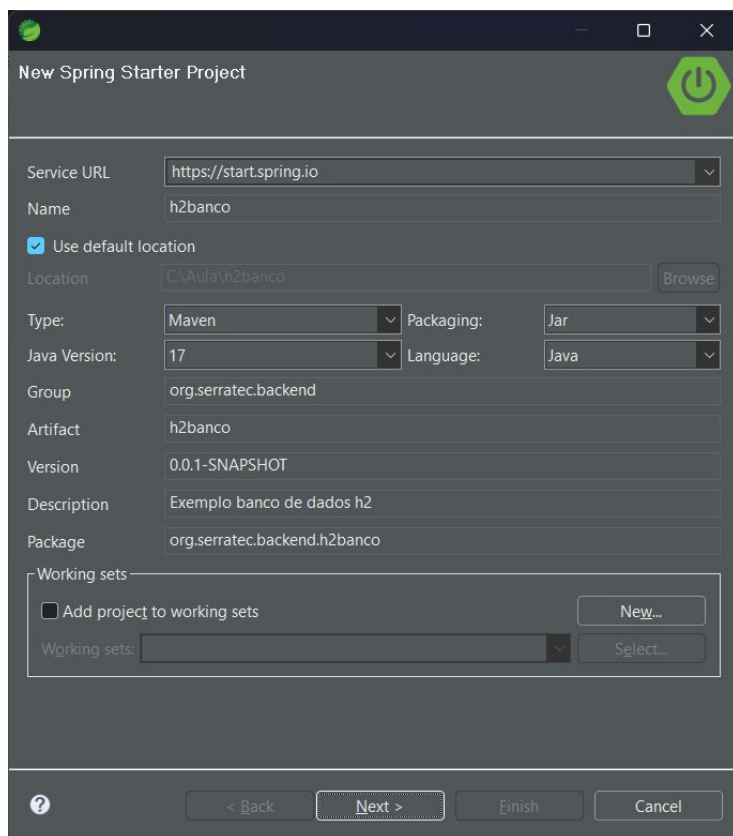


- Controllers
- Mapeamento Objeto Relacional
- Operações CRUD
- Validação
- Exceções

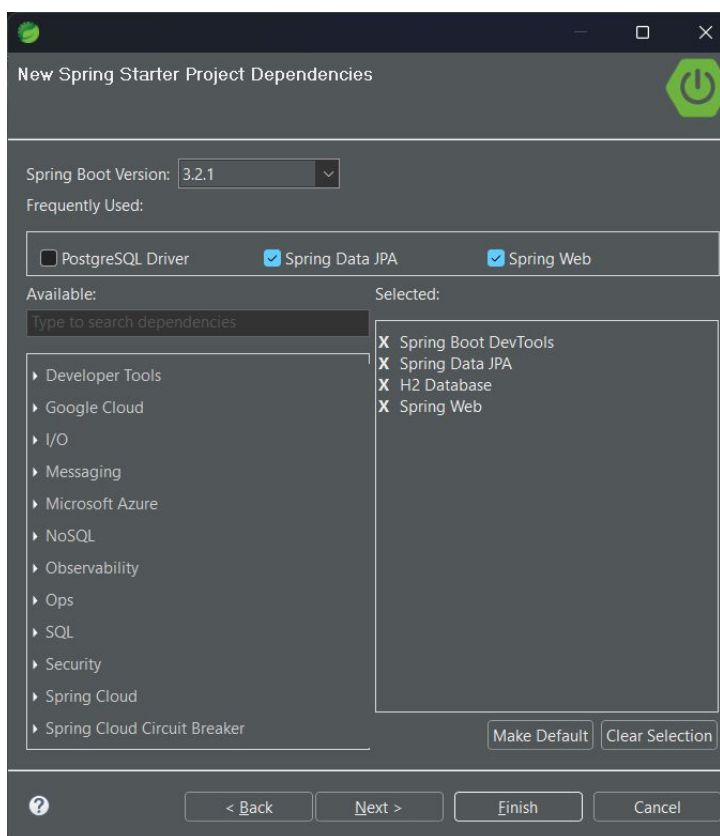
H2 - Banco de dados “Embedded”



É um banco de dados Open Source “embedded” feito inteiramente em Java, não precisa de instalação, pois pode executar somente em memória ou armazenar os dados em uma pasta. É acessível pelo navegador.



The image shows the 'New Spring Starter Project' dialog in an IDE. The 'Service URL' is set to 'https://start.spring.io'. The 'Name' is 'h2banco'. The 'Location' is 'C:\Aula\h2banco'. The 'Type' is 'Maven', 'Packaging' is 'Jar', 'Java Version' is '17', and 'Language' is 'Java'. The 'Group' is 'org.serratec.backend', 'Artifact' is 'h2banco', 'Version' is '0.0.1-SNAPSHOT', and 'Description' is 'Exemplo banco de dados h2'. The 'Package' is 'org.serratec.backend.h2banco'. There are buttons for 'New...' and 'Select...' under 'Working sets'.



The image shows the 'New Spring Starter Project Dependencies' dialog. The 'Spring Boot Version' is '3.2.1'. Under 'Frequently Used', 'Spring Data JPA' and 'Spring Web' are checked. Under 'Available', 'H2 Database' is selected. Under 'Selected', 'Spring Boot DevTools', 'Spring Data JPA', 'H2 Database', and 'Spring Web' are listed. There are buttons for 'Make Default' and 'Clear Selection'.

Dependência do H2 que será inserida na criação do projeto no pom.xml

```
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>
```

H2 - Banco de dados “Embedded”



Vamos criar a classe **Pessoa** no pacote **domain**
Importar as anotações do pacote **jakarta.persistence**

```
@Entity
public class Pessoa {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column
    private String nome;

    @Column
    private String email;

}
```

H2 - Banco de dados em Memória



Precisamos alterar a url no arquivo de propriedades do spring para executar em memória

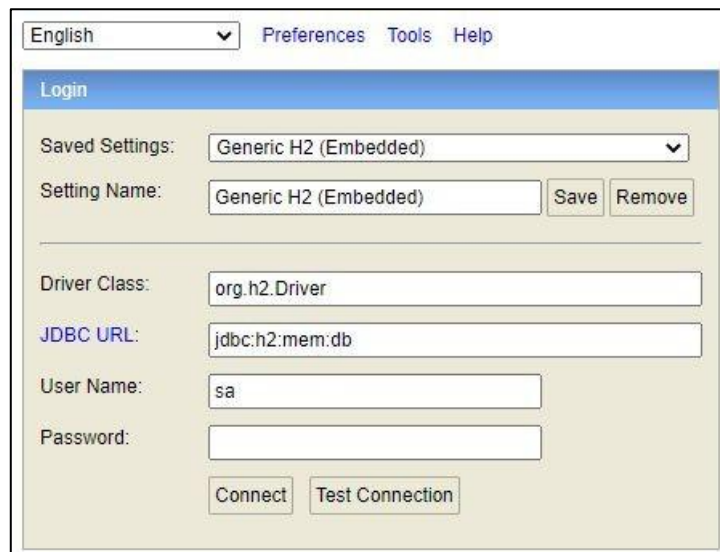
```
spring.h2.console.enabled=true  
spring.datasource.url=jdbc:h2:mem:db  
spring.jpa.hibernate.ddl-auto=create
```

Habilitar o console e url para executar em memória

H2 - Banco de dados em Memória

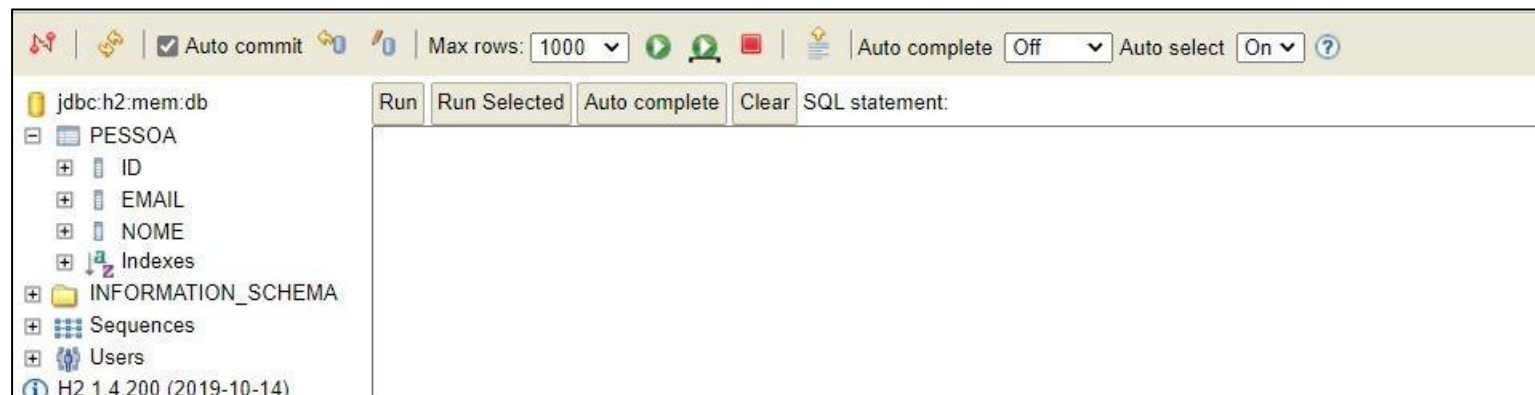


- Executar nossa aplicação
- Acessar <http://localhost:8080/h2-console/>



Verifique a url, se precisar altere o campo **JDBC URL** para **jdbc:h2:mem:db**
User Name: sa
Password:

Após conectar verifique se a tabela foi criada



H2 - Banco de dados em Disco



Precisamos alterar a url no arquivo de propriedades do spring para utilizar o h2 em disco

```
spring.datasource.url=jdbc:h2:file:./h2/banco;DB_CLOSE_DELAY=-1
spring.datasource.username=sa
spring.datasource.password=password
```

Informações sobre a conexão

```
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Criar as tabelas automaticamente e mostrar o sql gerado no console

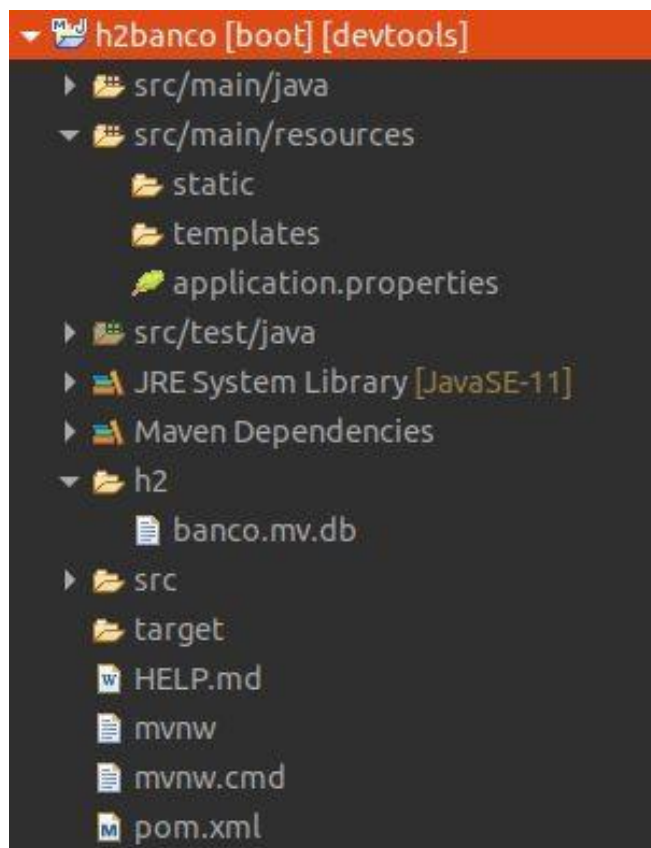
```
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

Habilitar o console e o path do h2

H2 - Banco de dados em Disco



Fazer um “refresh” na estrutura do projeto (F5) a pasta h2 que foi criada deverá aparecer com o nome do arquivo de banco de dados.



Depois basta acessar <http://localhost:8080/h2-console> com os mesmos dados utilizados no arquivo de propriedades (**url, username e password**)

Relacionamento - Embedded



No banco de dados tem situações que colocamos alguns campos na mesma tabela, mas em classes na programação orientada a objetos separamos nosso código para ficar mais organizado. Vamos utilizar as anotações abaixo para demonstrar.

@Embedded - A anotação é utilizada para embutir um tipo em outra entidade.

@Embeddable - Serve para declarar que uma classe será incorporada por outras entidades.

Relacionamento - Embedded



No exemplo abaixo a anotação **@Embedded** informa que a tabela **Veiculo** deverá conter todos os campos de **Caracteristica** no banco de dados.

```
@Entity
@Table(name="veiculo")
public class Veiculo {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank(message="Preencha a placa")
    @Size(max=7)
    @Column(nullable = false, length=7)
    private String placa;

    @NotBlank(message="Preencha a marca")
    @Size(max=30)
    @Column(nullable = false, length=30)
    private String marca;

    @NotBlank(message="Preencha o modelo")
    @Size(max=40)
    @Column(nullable = false, length=40)
    private String modelo;

    @Embedded
    private Caracteristica caracteristica;

    //... gets e sets
}
```

Inserir a classe **Veiculo** com seus **Getters** e **Setters** e as anotações do hibernate e do Bean Validation

Vamos inserir a dependência para utilizarmos o **Bean Validation**

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

Quando adicionamos uma nova dependência, eventualmente ocorrem erros no **pom.xml**. Estes podem ser corrigidos fazendo uma atualização dos repositórios. Clique com o botão direito sobre o projeto **Maven – Update Project** selecione o projeto e antes de clicar em ok, verifique se a opção **Force Update of Snapshot/Releases** está selecionada.

Relacionamento - Embedded



@Embeddable - Estamos informando que a classe **Caracteristica** será incorporada por outra classe.

```
@Embeddable
public class Caracteristica {
    private String renavam;
    private String chassi;
    private Long ano;

    @Enumerated(EnumType.STRING)
    private Categoria categoria;

    private String cor;

    @Enumerated(EnumType.ORDINAL)
    private Combustivel combustivel;

    //... gets e sets
}
```

@Enumerated - Serve para configurar um tipo de enumeração.

(EnumType.String) - Na categoria vai armazenar na tabela o valor da constante do tipo String.

(EnumType.Ordinal) - No combustível vai armazenar o número que representa a opção no Enum.

Inserir os Enums abaixo

```
public enum Categoria {
    HATCH, SEDAN, PICAPE, SUV, CONVERSIVEL, MINIVAN
}
```

```
public enum Combustivel {
    ALCOOL(1, "Álcool"), GASOLINA(2, "Gasolina"),
    DIESEL(3, "Diesel"), FLEX(4, "Flex");
    private Integer codigo;
    private String tipo;

    private Combustivel(Integer codigo, String tipo) {
        this.codigo = codigo;
        this.tipo = tipo;
    }

    public Integer getCodigo() {
        return codigo;
    }

    public String getTipo() {
        return tipo;
    }
}
```

Relacionamento - Embedded



Vamos executar a aplicação e visualizar o console com as informações gerada pela criação da tabela.

```
2022-09-03 19:59:30.347 INFO 1369021 --- [ restartedMain] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2022-09-03 19:59:30.435 INFO 1369021 --- [ restartedMain] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
Hibernate: create table veiculo (id bigint generated by default as identity, ano bigint, categoria varchar(255), chassi varchar(255), combustivel integer, cor varchar(255), renavam varchar(255), marca varchar(255), modelo varchar(255), placa varchar(255))
2022-09-03 19:59:30.962 INFO 1369021 --- [ restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2022-09-03 19:59:30.968 INFO 1369021 --- [ restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
```

Verificando no h2 que todos campos foram adicionados a tabela **veiculo**

The screenshot shows the H2 console interface. On the left, a tree view displays the database schema, including the 'VEICULO' table with columns: ID, ANO, CATEGORIA, CHASSI, COMBUSTIVEL, COR, RENAVAM, MARCA, MODELO, and PLACA. The main area shows the SQL statement 'SELECT * FROM VEICULO;' and the result, which is an empty table with the same columns. The status bar at the bottom indicates '(no rows, 2 ms)'.

Exercício (revisão)



Inserir a interface **VeiculoRepository** herdando de **JpaRepository**

Inserir a classe de **VeiculoController** criando um CRUD para utilização do recursos.

Inserir as anotações necessárias para que esta classe funcione como um controlador.

Inserir as validações para tratamento dos campos obrigatórios na classe **Veiculo**: placa, marca e modelo.

Inserir a classe de **ControllerExceptionHandler** e **ErroResposta** para tratamento das exceções

Exercício (revisão) - Resolução



Inserir a interface **VeiculoRepository** no pacote **repository** herdando de **JpaRepository**

```
@Repository
public interface VeiculoRepository extends JpaRepository<Veiculo, Long>{
}
```

Exercício (revisão) - Resolução



Inserir a classe de **VeiculoController** criando um CRUD para utilização do recursos.

```
@RestController
@RequestMapping("/veiculos")
public class VeiculoController {
    @Autowired
    private VeiculoRepository veiculoRepository;

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public Veiculo inserir(@Valid @RequestBody
        Veiculo veiculo) {
        return veiculoRepository.save(veiculo);
    }

    @GetMapping
    public List<Veiculo> listar() {
        return veiculoRepository.findAll();
    }

    @GetMapping("/{id}")
    public ResponseEntity<Veiculo> buscar(@PathVariable Long id) {
        Optional<Veiculo> veiculo = veiculoRepository.findById(id);
        if (!veiculo.isPresent()) {
            return ResponseEntity.notFound().build();
        }
        return ResponseEntity.ok(veiculo.get());
    }
}
```

```
@DeleteMapping("/{id}")
public ResponseEntity<Void> apagar(@PathVariable Long id) {
    if (!veiculoRepository.existsById(id)) {
        return ResponseEntity.notFound().build();
    }
    veiculoRepository.deleteById(id);
    return ResponseEntity.noContent().build();
}

@PutMapping("/{id}")
public ResponseEntity<Veiculo> alterar(@PathVariable Long id,
    @Valid @RequestBody Veiculo veiculo) {
    if (!veiculoRepository.existsById(id)) {
        return ResponseEntity.notFound().build();
    }
    veiculo.setId(id);
    return ResponseEntity.ok(veiculoRepository.save(veiculo));
}
```

Exercício (revisão) - Resolução



Inserir a classe de **ControllerExceptionHandler** e **ErroResposta** para tratamento das exceções

```
@ControllerAdvice
public class ControllerExceptionHandler extends ResponseEntityExceptionHandler{
    @Override
    protected ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException ex,
        HttpHeaders headers, HttpStatus status, WebRequest request) {

        List<String> erros = new ArrayList<>();
        for(FieldError error: ex.getBindingResult().getFieldErrors()){
            erros.add(error.getField() + ": " + error.getDefaultMessage());
        }
        ErroResposta erroResposta = new ErroResposta(status.value(), "Existem Campos Inválidos", LocalDateTime.now(), erros);
        return super.handleExceptionInternal(ex, erroResposta, headers, status, request);
    }
}
```

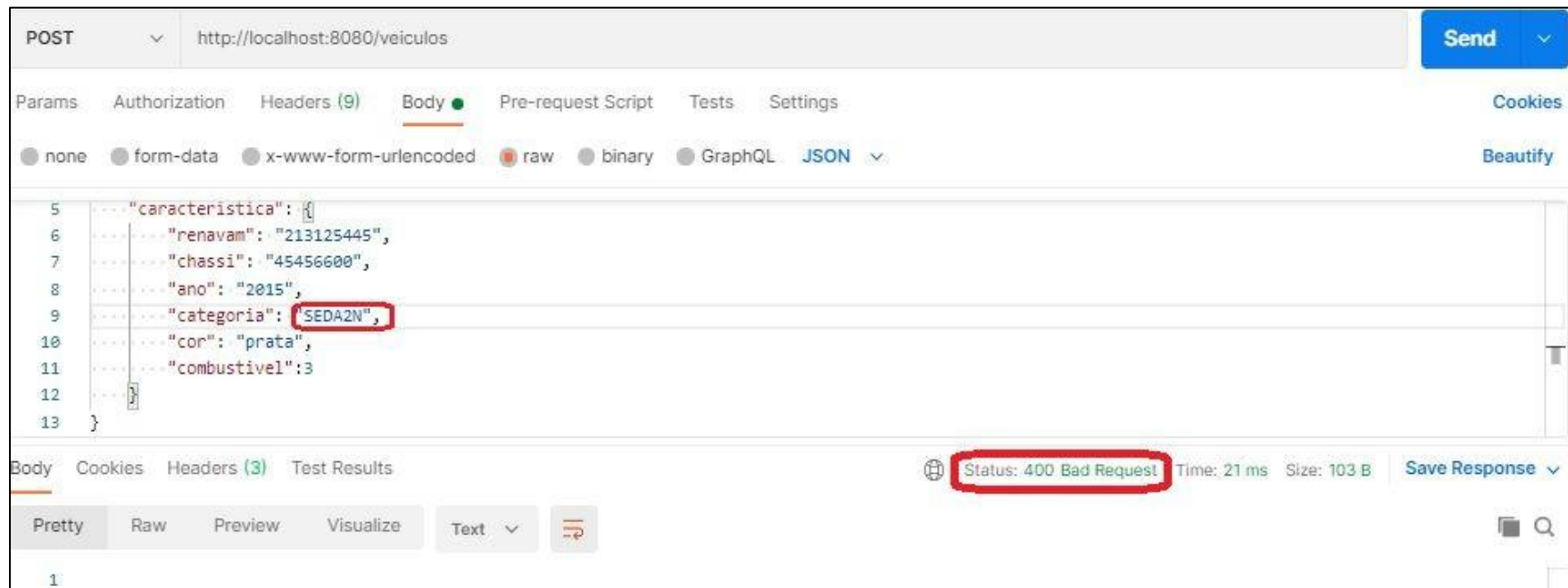
```
public class ErroResposta {
    private Integer status;
    private String titulo;
    private LocalDateTime dataHora;
    private List<String> erros;

    public ErroResposta(Integer status, String titulo, LocalDateTime dataHora, List<String> erros) {
        this.status = status;
        this.titulo = titulo;
        this.dataHora = dataHora;
        this.erros = erros;
    }
}
```


Exception - Enum



No exercício anterior caso os Enums sejam digitados de forma errada ou uma categoria inexistente não temos nenhuma resposta.



Exception - Enum



Vamos inserir um método para verificação da categoria e do tipo de combustível com a anotação `@JsonCreator` no enum `Categoria` e `Combustivel`.

Criar a primeiro a classe para tratamento de exceção com o nome `EnumValidationException`

```
public class EnumValidationException extends Exception{
    public EnumValidationException(String message){
        super(message);
    }
}
```

```
@JsonCreator
public static Categoria verifica(String value) throws EnumValidationException{
    for(Categoria c : values()) {
        if (value.equals(c.name())) {
            return c;
        }
    }
    throw new EnumValidationException("Categoria preenchida incorretamente");
}
```

```
@JsonCreator
public static Combustivel verifica(Integer value) throws EnumValidationException{
    for(Combustivel c : values()) {
        if (value.equals(c.getCodigo())) {
            return c;
        }
    }
    throw new EnumValidationException("Categoria preenchida incorretamente");
}
```

Na iteração do **for** comparamos o valor passado como argumento (**value**) da requisição com o elementos contidos no enum (**values**) para verificar a existência do elemento.

Exception - Enum



Vamos inserir a exception do tipo `handleHttpMessageNotReadable` na classe `ControllerExceptionHandler` que é o tipo de exceção retornado no console, depois faremos o teste no Postman.

```
2022-09-03 21:03:16.544 INFO 1397463 --- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2022-09-03 21:03:16.545 INFO 1397463 --- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2022-09-03 21:03:16.656 WARN 1397463 --- [nio-8080-exec-2] .m.m.a.ExceptionHandlerExceptionResolver : Resolved [org.springframework.http.converter.HttpMessageNotReadableException: JSON parse error:
```

```
@ControllerAdvice
public class ControllerExceptionHandler extends ResponseEntityExceptionHandler{
    @Override
    protected ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException ex,
        HttpHeaders headers, HttpStatus status, WebRequest request) {

        List<String> erros = new ArrayList<>();
        for(FieldError error: ex.getBindingResult().getFieldErrors()){
            erros.add(error.getField() + ": " + error.getDefaultMessage());
        }
        ErroResposta erroResposta = new ErroResposta(status.value(), "Existem Campos Inválidos",
            LocalDateTime.now(), erros);
        return super.handleExceptionInternal(ex, erroResposta, headers, status, request);
    }

    @Override
    protected ResponseEntity<Object> handleHttpMessageNotReadable(HttpMessageNotReadableException ex,
        HttpHeaders headers, HttpStatus status, WebRequest request) {
        return ResponseEntity.badRequest().body(ex.getMessage());
    }
}
```

POST http://localhost:8080/veiculos

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
4  .... "modelo": "Fox",
5  .... "caracteristica": {
6  ..... "renavam": "213125445",
7  ..... "chassi": "4546600",
8  ..... "ano": "2015",
9  ..... "categoria": "SEDA2N",
10 ..... "cor": "prata",
```

Body Cookies Headers (4) Test Results 400 Bad Request 264 ms 743 B Save Response

Pretty Raw Preview Visualize Text

```
1  JSON parse error: Cannot construct instance of `org.serratec.backend.h2banco.domain.Categoria`,
   problem: Categoria preenchida incorretamente; nested exception is com.fasterxml.jackson.databind.
   exc.ValueInstantiationException: Cannot construct instance of `org.serratec.backend.h2banco.domain.
   Categoria`, problem: Categoria preenchida incorretamente
2  at [Source: (org.springframework.util.StreamUtils$NonClosingInputStream); line: 9, column: 21]
   (through reference chain: org.serratec.backend.h2banco.domain.Veiculo["caracteristica"]->org.
   serratec.backend.h2banco.domain.Caracteristica["categoria"])
```

Relacionamento - Exercício



No projeto da **terceira aula** usando o banco criado no **Postgres** criar a classe **Endereco** que deverá ser anotada com **@Embeddable** com os seguintes atributos abaixo todos do tipo **String**:

- logradouro
- numero
- bairro
- cidade
- Estado

Adicionar o atributo **endereco** do tipo **Endereco** na classe **Cliente**. Inserir a anotação **@Embedded**

Inserir os campos referente ao endereço na tabela **cliente** do banco de dados que foi utilizado no **Postgres** com o nome **curso**

```
alter table cliente
add column logradouro varchar(50),
add column numero varchar(6), add
column bairro varchar(40),
add column cidade varchar(40),
add column estado varchar(30);
```

Relacionamento - Exercício



No exemplo abaixo estamos dizendo que a tabela cliente irá conter todos os campos de endereço no banco de dados

```
@Embeddable
public class Endereco {
    private String logradouro;
    private String numero;
    private String bairro;
    private String cidade;
    private String estado;
    public String getLogradouro() {
        return logradouro;
    }
    public void setLogradouro(String logradouro) {
        this.logradouro = logradouro;
    }
    public String getNumero() {
        return numero;
    }
}
```

Inserir a classe Endereco com seus Getters e Setters

```
@Entity
@Table(name="cliente")
public class Cliente {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id_cliente")
    private Long id;

    @NotBlank(message="Preencha o nome")
    @Size(max=60)
    @Column
    private String nome;

    @CPF(message="CPF Inválido")
    @Column
    private String cpf;

    @Email(message="Email inválido")
    @Column
    private String email;

    @Embedded
    private Endereco endereco;
```

Inserir o atributo endereco e os seus Getters e Setters

Relacionamento - Exercício



Testar no Postman

The screenshot shows the Postman interface with a POST request to `http://localhost:8080/clientes`. The request body is a JSON object representing a client. The response status is `201 Created`, which is highlighted with a red box. The response body is also shown in the bottom panel.

Request:

```
POST http://localhost:8080/clientes
```

Body (JSON):

```
{  "nome": "Marcos Antônio",  "cpf": "36553859868",  "email": "marcosa@yahoo.com.br",  "endereco": {    "logradouro": "Rua Fonseca Ramos",    "numero": "185",    "bairro": "Centro",    "cidade": "Petrópolis",    "estado": "Rio de Janeiro"  }}
```

Response:

```
{  "id": 7,  "nome": "Marcos Antônio",  "cpf": "36553859868",  "email": "marcosa@yahoo.com.br",  "endereco": {    "logradouro": "Rua Fonseca Ramos",    "numero": "185",    "bairro": "Centro",
```

Herança



@MappedSuperclass - Essa anotação serve para indicar que as tabelas filhas sejam criadas com todos os dados da classe super. Herdamos o mapeamento de colunas de uma classe concreta ou abstrata que não tenha a anotação **@Entity**.

@Inheritance(strategy=InheritanceType.SINGLE_TABLE)

@DiscriminatorColumn(name="DTYPE")

Essa anotação serve para criar uma tabela única com todos os dados das tabelas filhas. A desvantagem neste método é que teremos campos nulos para determinadas situações. A segunda anotação serve para criar uma coluna com o nome da entidade filha.

@Inheritance(strategy=InheritanceType.JOINED)

Serve para gerar uma tabela por classe.

@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)

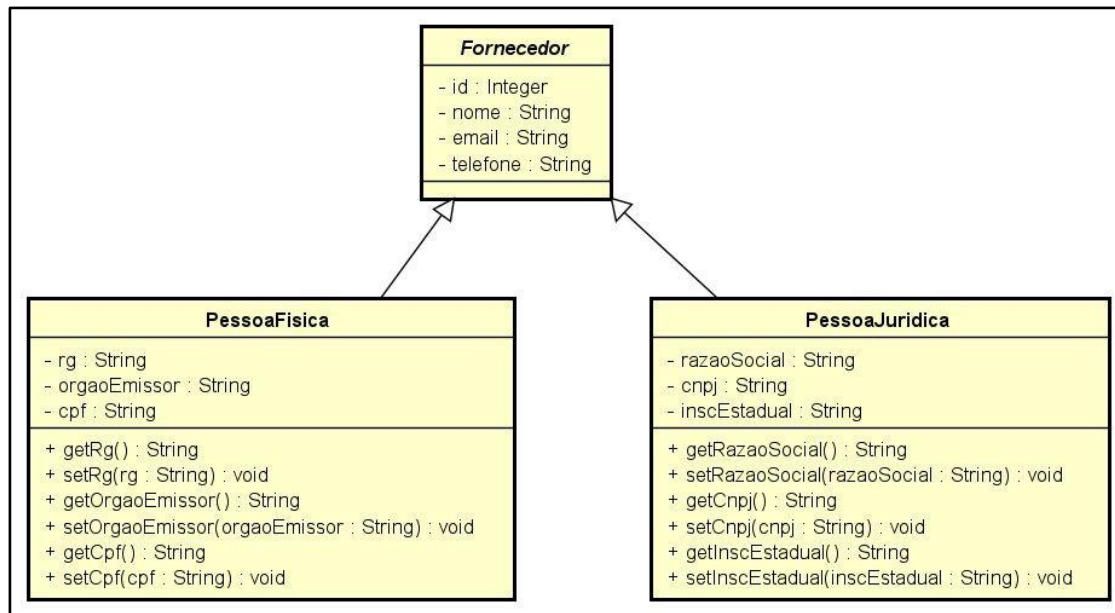
Define que apenas as classes concretas serão criadas.

Herança



No exemplo abaixo onde temos o diagrama de classe vamos utilizar os mapeamentos do hibernate.

Vamos inserir a classe **Fornecedor** com a anotação **@MappedSuperclass** inserindo os **getters** e **setters**



```
@MappedSuperclass
public class Fornecedor {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(length=50)
    @NotBlank(message="Nome da Pessoa deve ser preenchido")
    @Size(min=2, max=50, message="Nome da pessoa deve ter entre {min} e {max} letras")
    private String nome;

    @Column
    private String email;

    @Column
    private String telefone;

    public Long getId() {
        return id;
    }

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


Herança



Vamos inserir a classe **PessoaFisica** e **PessoaJuridica** herdando de **Fornecedor**

```
@Entity
public class PessoaFisica extends Fornecedor {
    @Column
    private String rg;
    @Column
    private String orgaoEmissor;
    @Column String cpf;
    public String getRg() {
        return rg;
    }
    public void setRg(String rg) {
        this.rg = rg;
    }
}
```

```
@Entity
public class PessoaJuridica extends Fornecedor{
    @Column
    private String razaoSocial;
    @Column
    private String cnpj;
    @Column
    private String insEstadual;

    public String getRazaoSocial() {
        return razaoSocial;
    }
    public void setRazaoSocial(String razaoSocial) {
        this.razaoSocial = razaoSocial;
    }
}
```

Acessar <http://localhost:8080/h2-console>
Verificar no **H2** as classes criadas



Herança

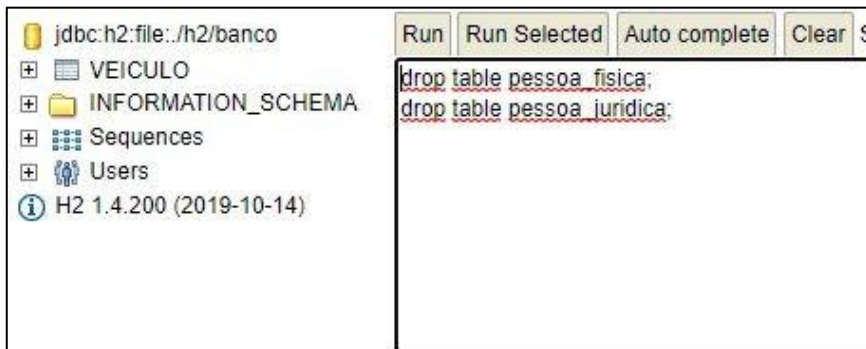


Vamos apagar as tabelas criadas no h2 **pessoa_física** e **pessoa_jurídica** para fazer o teste.

Neste exemplo, vamos utilizar a estratégia

@Inheritance(strategy=InheritanceType.SINGLE_TABLE)

Realizando alteração destacada na classe **Fornecedor**, **PessoaFisica** e **PessoaJuridica**



```
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)  
@DiscriminatorColumn(name="DTYPE")  
@Entity  
public class Fornecedor {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @Column(length=50)  
    @NotBlank(message="Nome da Pessoa deve ser preenchido")  
    @Size(min=2, max=50, message="Nome da pessoa deve ter entre {min} e {max}  
letras")  
    private String nome;  
  
    @Column  
    private String email;  
  
    @Column  
    private String telefone;
```

Herança



A tabela fornecedor foi criada com todos os dados das filhas

The screenshot shows the H2 database console interface. On the left, a tree view displays the database structure, including the 'FORNECEDOR' table with columns: DTYPE, ID, EMAIL, NOME, TELEFONE, CPF, ORGAO_EMISSOR, RG, CNPJ, INSC_ESTADUAL, RAZAO_SOCIAL, and Indexes. Below the tree, it shows 'VEICULO', 'INFORMATION_SCHEMA', 'Sequences', 'Users', and the version 'H2 1.4.200 (2019-10-14)'. The main area on the right has a toolbar with buttons: Run, Run Selected, Auto complete, and Clear. Below the toolbar is a large text area for the SQL statement. At the bottom right, there is a section titled 'Important Commands' with a table of shortcuts.

Important Commands		
?		Displays this Help Page
		Shows the Command History
Ctrl+Enter		Executes the current SQL statement
Shift+Enter		Executes the SQL statement defined by the text selection
Ctrl+Space		Auto complete
		Disconnects from the database

Exercicios



Criar uma classe com o nome **Funcionario** com os seguintes atributos **protected**:

String(nome,cpf)

double(salario)

String(turno)

Herança:

Criar uma classe com o nome Gerente com os seguintes atributos privados:

String(setor)

Utilizar o banco de arquivo **h2** para criação das tabelas.

Fazer o mapeamento objeto relacional. Escolher a melhor estratégia para mapeamento da herança.

Criar a classe **Controller** fazendo um CRUD com as principais operações.

Fazer a validação de dados para **nome**, **cpf** e **salario**.

Criar a classe de exceção para tratar os erros de validação.