

# Programação I

**Persistência de Dados  
Spring / Hibernate**

Samuel da Silva Feitosa

Aula 16

# Persistência de Dados

- Persistência de dados é fazer com que dados sejam **armazenados / gravados no computador**, possibilitando sua **recuperação** em outra execução do programa.
  - O armazenamento desses dados pode acontecer de diferentes maneiras.
- Exemplos:
  - Bancos de dados.
  - Arquivos no disco do computador.
  - Armazenamento em nuvem.
  - Etc.

# Arquitetura Utilizada

- Nesta aula vamos utilizar uma arquitetura moderna de desenvolvimento, muito utilizada no mercado atualmente.
  - Vamos desenvolver WEB Services REST, que vão ser responsáveis por disponibilizar um CRUD (Create, Read, Update e Delete) em um banco de dados em memória.
  - Este formato é o padrão atual de novos projetos WEB (backend).
- Para a interface com o usuário utilizaremos a biblioteca Swing com o projeto que iniciamos na aula anterior.
  - Geralmente, são utilizadas bibliotecas de JavaScript (React, Angular, Vue, etc.) para desenvolvimento do frontend em aplicações WEB.
  - Em nosso projeto, vamos ‘consumir’ os WEB Services REST via Java.

# Projeto Backend

- Para o backend, vamos criar a estrutura e os WEB Services para permitir a criação, leitura, atualização e exclusão de clientes e cartões.
- No frontend, vamos associar as telas já desenvolvidas com as operações disponíveis através do backend.

# Preparação

- Primeiramente, para facilitar o processo, vamos baixar a **extensão** do VSCode para trabalhar com o Spring.
  - **Spring** é um framework criado com o objetivo de facilitar o desenvolvimento de aplicações, explorando para isso, os conceitos de Inversão de Controle e Injeção de Dependências.
  - Não precisa de um servidor para rodar, utiliza apenas aquilo que é necessário para o projeto.
  - **Spring Boot** é um framework (ou extensão do Spring) que facilita a criação de aplicações Spring, possibilitando a execução imediata.
  - Permite gastar o mínimo de tempo possível configurando o projeto.
- Extensão: Spring Boot Extension Pack

# Criação do Projeto (1)

- Neste projeto, vamos utilizar o gerenciador de dependências Maven.
  - Sendo assim, ao criar o novo projeto, vamos escolher as opções:
  - Create Java Project -> Spring Boot -> Maven Project -> *Outros detalhes*
  - Selecionar dependências:
    - Spring Boot DevTools
    - Lombok
    - Spring Web
    - Spring Data JPA
    - H2 Database
  - O projeto será iniciado para a criação de WEB Services automaticamente.

# Desenvolvimento do Backend

- Após a criação, o projeto está pronto para a codificação das regras de negócio do sistema.
  - Para baixar as dependências na primeira execução é necessário clicar em Maven -> backend -> Lifecycle -> Install
- Vamos criar mais três pacotes para organizar as nossas classes:
  - Controllers: onde ficarão os códigos com a lógica do sistema.
  - Entities: onde serão descritas as entidades do sistema.
  - Repositories: usados para fornecer as principais funcionalidades de acesso, inserção, remoção e listagem de informações do banco de dados.

# Entity - Cliente

- Em nossa aplicação, teremos duas entidades (Cliente e Cartão).

```
@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Cliente {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    Long id;
    String cpf;
    String nome;
    String dataNasc;
}
```

- Notem o uso de diversas anotações (iniciadas com @).
- Estas anotações, em sua maior parte, servem para instruir o mecanismo a realizar a geração de código automaticamente.

- Notem também a não criação de construtores, getters e setters, etc.



# Entity - Cartão

```
@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Cartao {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    Long id;
    @ManyToOne
    private Cliente cliente;
    String numero;
    String dataVal;
}
```

# Repositories - Cliente e Cartão

- Da mesma forma, teremos os *repositories* para a Cliente e Cartão.
  - Este repositório é implementado como uma Interface, estendendo a interface JpaRepository.
  - Notem que, como estamos querendo usar apenas operações padrão, não precisamos inserir nenhum código na interface, e ela herdará todos os métodos diretamente.

```
public interface ClienteRepository extends JpaRepository<Cliente, Long> {  
  
}  
  
public interface CartaoRepository extends JpaRepository<Cartao, Long> {  
  
}
```

# ClienteController

- Nesta classe, vamos criar um WEB Service REST para Cliente.
  - Cada ação será mapeada para uma rota de execução.
  - Estas rotas são controladas automaticamente pelo Spring.
  - Chamamos o *repository* para executar as ações no banco de dados.

```
@RestController
@AllArgsConstructor
public class ClienteController {
    ClienteRepository repos;

    @GetMapping("/clientes")
    public List<Cliente> getAllClientes() {
        return repos.findAll();
    }

    @GetMapping("/cliente/{id}")
    public Cliente getClienteById(@PathVariable Long id) {
        return repos.findById(id).get();
    }

    @PostMapping("/cliente")
    public Cliente saveCliente(@RequestBody Cliente cliente) {
        return repos.save(cliente);
    }

    @DeleteMapping("/cliente/{id}")
    public void deleteCliente(@PathVariable Long id) {
        repos.deleteById(id);
    }
}
```

# CartaoController

- Nesta classe, vamos criar outro WEB Service REST para Cartão.
  - Cada ação será mapeada para uma rota de execução.
  - Estas rotas são controladas automaticamente pelo Spring.
  - Chamamos o *repository* para executar as ações no banco de dados.

```
@RestController
@AllArgsConstructor
public class CartaoController {
    CartaoRepository repos;

    @GetMapping("/cartoes")
    public List<Cartao> getAllCartoes() {
        return repos.findAll();
    }

    @GetMapping("/cartao/{id}")
    public Cartao getCartaoById(@PathVariable Long id) {
        return repos.findById(id).get();
    }

    @PostMapping("/cartao")
    public Cartao saveCartao(@RequestBody Cartao cliente) {
        return repos.save(cliente);
    }

    @DeleteMapping("/cartao/{id}")
    public void deleteCartao(@PathVariable Long id) {
        repos.deleteById(id);
    }
}
```

# Testando nossos WEB Services

- Nossos WEB Services já estão prontos, lendo as informações e salvando-as no banco de dados.
- Podemos testar se tudo está OK através do **Postman**, que é uma ferramenta que permite enviar requisições HTTP remotas e locais.
- Vamos testar os 4 *endpoints* que criamos para cada rota:
  - Get All
  - Save
  - Get
  - Delete

# Desenvolvimento do Frontend

- Com o *backend* desenvolvido, vamos integrar as telas (*frontend*) desenvolvidas na aula anterior.
  - <https://github.com/sfeitosa/prog1-front-base.git>

# Implementando as Entidades Cliente e Cartão

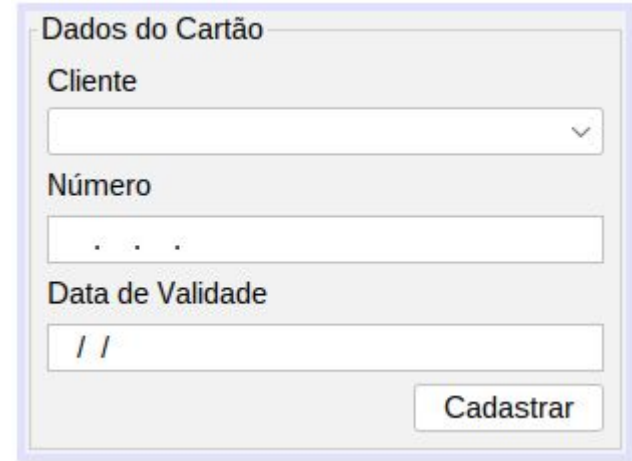
- Vamos implementar as classes Cliente e Cartao.
  - Adicionar os 4 atributos.
  - Gerar automaticamente:
    - Construtores (com e sem argumentos)
    - Getters e Setters
    - Método ToString

```
public class Cliente {  
    private Long id;  
    private String cpf;  
    private String nome;  
    private String dataNasc;  
}
```

```
public class Cartao {  
    private Long id;  
    private Cliente cliente;  
    private String numero;  
    private String dataVal;  
}
```

# Carregando os Clientes

- A tela desenvolvida espera que o usuário informe o Cliente, o Número do Cartão e a Data de Validade.
  - Clientes devem estar pré-cadastrados.
  - É preciso consultar o web service para obter as informações dos clientes.
  - Chamar *loadClientes* a partir do construtor.



Dados do Cartão

Cliente

Número

Data de Validade

Cadastrar

```
public void loadClientes() {  
    RestTemplate req = new RestTemplate();  
  
    ResponseEntity<Cliente[]> response = req.getForEntity(  
        "http://localhost:8080/clientes", Cliente[].class);  
  
    Cliente[] clientes = response.getBody();  
  
    for (var c : clientes) {  
        cbCliente.addItem(c);  
    }  
}
```



# Evento do botão ‘Cadastrar’

- Criar o objeto.
- Ler os campos.
- Enviar os dados lidos através de uma requisição POST / REST.

```
private void btnCadastrarActionPerformed(java.awt.event.ActionEvent evt) {  
    Cartao cartao = new Cartao();  
  
    cartao.setCliente((Cliente) cbCliente.getSelectedItem());  
    cartao.setNumero(edtNumero.getText());  
    cartao.setDataVal(edtDataVal.getText());  
  
    JOptionPane.showMessageDialog(this, cartao.toString());  
  
    try {  
        RestTemplate req = new RestTemplate();  
        req.postForObject("http://localhost:8080/cartao",  
            cartao, Cartao.class);  
        JOptionPane.showMessageDialog(this, "Cartao salvo com sucesso!");  
    }  
    catch (RestClientException e) {  
        JOptionPane.showMessageDialog(this, e.getMessage());  
    }  
  
    FrmCartaoLista frm = new FrmCartaoLista();  
  
    frm.setVisible(true);  
    this.dispose();  
}
```

# Preenchimento da tabela de Cartões

- Obter todas as informações dos cartões via GET.
- Apresentar as informações na tabela.
- Chamar essa função via construtor da tela de listagem.

```
public void loadTableData() {  
    RestTemplate req = new RestTemplate();  
  
    ResponseEntity<Cartao[]> response = req.getForEntity(  
        "http://localhost:8080/cartoes", Cartao[].class);  
  
    Cartao[] cartoes = response.getBody();  
  
    DefaultTableModel tbl = (DefaultTableModel) tblCartoes.getModel();  
  
    tbl.setNumRows(0);  
  
    for (Cartao c : cartoes) {  
        Object[] row = { c.getId(), c.getCliente().getNome() ,  
            c.getNumero(), c.getDataVal() };  
        tbl.addRow(row);  
    }  
}
```

# Considerações Finais

- Nesta aula estudamos a forma ‘moderna’ de realizar a integração entre sistemas (frontend e backend).
- Utilizamos REST como o padrão de comunicação.
- Estudamos como fazer persistência de dados usando Spring / JPA e um banco de dados em memória.
  - O backend faz a persistência em banco de dados.
  - O frontend consome uma API, que pode estar rodando localmente ou em nuvem.
- Existem várias outras formas de fazer persistência de dados.
  - Salvar informações em arquivos, utilizar JDBC para salvar um banco de dados local, etc.

# Exercícios

1. Implementar as ações da tela de cadastro de Clientes.
  - a. Listagem dos clientes cadastrados através de requisição GET/REST.
  - b. Cadastro de um novo cliente através de requisição POST/REST.
2. Implementar a ação de exclusão de um Cliente ou Cartão.
  - a. Utilizar a requisição DELETE/REST.