	**Campus**: Polo Cohama **Curso**: Desenvolvimento FullStack **Turma**: 9001 **Disciplina**: Back-end Sem Banco Não Tem **Nome**: João Victor Sá de Araújo

Relatório de Prática: Mapeamento Objeto-Relacional e DAO e Alimentando a Base

1. Objetivo da Prática

A prática desenvolvida no repositório tem como objetivo principal demonstrar o funcionamento de um backend sem a utilização de um banco de dados tradicional, focando na criação de uma estrutura funcional para gerenciar dados localmente e explorar conceitos fundamentais de desenvolvimento backend. Isso inclui a aplicação de padrões de projeto como DAO (Data Access Object), o uso de middleware para a abstração de operações, e a simulação de interações típicas entre a aplicação e um banco de dados.

2. Códigos

```java
public class PessoaJuridica extends Pessoa{
	private String cnpj;

	public PessoaJuridica(int id, String nome, String logradouro, String cidade, String estado, String telefone,
			String email, String cnpj) {
		super(id, nome, logradouro, cidade, estado, telefone, email);
		this.cnpj = cnpj;
	}

	public String getCnpj() {
		return cnpj;
	}

	public void setCnpj(String cnpj) {
		this.cnpj = cnpj;
	}
```

```java
public class CadastroBDTeste {

	public static void main(String[] args) throws ClassNotFoundException, SQLException {

		PessoaFisicaDAO pessoaFisicaDao = new PessoaFisicaDAO();
		PessoaJuridicaDAO pessoaJuridicaDao = new PessoaJuridicaDAO();

		try (Scanner leia = new Scanner(System.in)) {
			while (true) {

				try {
					System.out.println("Cadastro De Pessoas");
					System.out.println();
					System.out.println("1 - Incluir Pessoa\n" + "2 - Alterar Pessoa\n" + "3 - Excluir Pessoa\n"
							+ "4 - Buscar pelo Id\n" + "5 - Exibir Todos Cadastrados\n" + "0 - Finalizar Programa ");
					System.out.println();
					System.out.println("Escolha uma das opcoes a cima: ");

					Integer opc = leia.nextInt();
					leia.nextLine();

					if (opc >= 1 && opc <= 5) {

						System.out.println("Digite:\n F - Pessoa Fisica | J - Pessoa Juridica");
						String cadast = leia.nextLine().toUpperCase();
						String cadastoL = cadast.substring(0, 1);

						try {
							switch (opc) {
							case 1:
								if (cadastoL.equals("F")) {

									pessoaFisicaDao.inserindoPessoa();

								} else if (cadastoL.equals("J")) {
```

```java
public class PessoaJuridicaDAO {

	PessoaJuridica pessoaJuridica = new PessoaJuridica(0, null, null, null, null, null, null, null);

	Scanner leia = new Scanner(System.in);

	Connection conn = null;
	PreparedStatement preparedStatemet = null;

	public void incluir(PessoaJuridica pessoaJuridica) {
		String sql = "INSERT INTO pessoajuridica (nome, logradouro, cidade, estado, telefone, email,cnpj) VALUES (?,?,?,?,?,?,?)";

		try {
			// Criar uma conexão com o banco de dados
			conn = ConectorBD.createConnectionToMySQL();

			// Criar uma preparedStatemet, para executar uma query
			preparedStatemet = conn.prepareStatement(sql);

			preparedStatemet.setString(1, pessoaJuridica.getNome());
			preparedStatemet.setString(2, pessoaJuridica.getLogradouro());
			preparedStatemet.setString(3, pessoaJuridica.getCidade());
			preparedStatemet.setString(4, pessoaJuridica.getEstado());
			preparedStatemet.setString(5, pessoaJuridica.getTelefone());
			preparedStatemet.setString(6, pessoaJuridica.getEmail());
			preparedStatemet.setString(7, pessoaJuridica.getCnpj());

			// Executar a query
			preparedStatemet.execute();

		} catch (Exception e) {
			e.printStackTrace();
		} finally {
			// Fechar as conexões
			try {
				if (conn != null) {
					conn.close();
				}
			} catch (Exception e) {
				e.printStackTrace();
			}
		}
	}

	public List<PessoaJuridica> getPessoas() throws ClassNotFoundException, SQLException {
```

3. Análise e Conclusão

1. **Qual a importância dos componentes de middleware, como o JDBC?**
   Componentes de middleware, como o JDBC (Java Database Connectivity), são fundamentais para estabelecer a comunicação entre a aplicação e o banco de dados. Eles fornecem uma interface padronizada que permite às aplicações interagir com diversos sistemas de gerenciamento de banco de dados (SGBDs) de forma transparente. O JDBC, especificamente, facilita a execução de operações como consultas e atualizações, além de gerenciar conexões e transações, abstraindo detalhes específicos de cada banco de dados e permitindo que os desenvolvedores foquem na lógica da aplicação.
2. **Qual a diferença no uso de Statement ou PreparedStatement para a manipulação de dados?**
   A principal diferença entre `Statement` e `PreparedStatement` reside na forma como as instruções SQL são enviadas e processadas pelo banco de dados:
   - **Statement:** Utilizado para executar instruções SQL estáticas. Cada vez que uma instrução é executada, o banco de dados a compila e executa, o que pode resultar em menor desempenho quando a mesma instrução é executada repetidamente com parâmetros diferentes.
   - **PreparedStatement:** Projetado para instruções SQL que serão executadas múltiplas vezes com parâmetros diferentes. A instrução é pré-compilada pelo banco de dados na primeira execução, permitindo reutilização e melhor desempenho nas execuções subsequentes. Além disso, o

`PreparedStatement` ajuda a prevenir ataques de SQL Injection, pois trata os parâmetros de forma segura.

3. **Como o padrão DAO melhora a manutenibilidade do software?**
   O padrão DAO (Data Access Object) melhora a manutenibilidade do software ao separar a lógica de acesso a dados da lógica de negócio. Ao encapsular todas as operações de acesso ao banco de dados em classes específicas (DAOs), facilita-se a manutenção e evolução do código, permitindo alterações no mecanismo de persistência sem impactar outras partes da aplicação. Essa separação também promove a reutilização de código e facilita a realização de testes unitários, resultando em um sistema mais modular e de fácil manutenção.
4. **Como a herança é refletida no banco de dados, quando lidamos com um modelo estritamente relacional?**
   Em um modelo relacional, a representação de hierarquias de herança presentes no modelo orientado a objetos pode ser realizada de diferentes maneiras, cada uma com suas vantagens e desvantagens:
   - **Tabela única (Single Table Inheritance):** Todas as classes da hierarquia são representadas por uma única tabela, com colunas para todos os atributos e um campo discriminador para identificar o tipo. Embora simples, pode resultar em muitos campos nulos.
   - **Tabela por classe concreta (Concrete Table Inheritance):** Cada classe concreta possui sua própria tabela, contendo todas as colunas necessárias. Não há compartilhamento de colunas entre tabelas, mas pode haver redundância de dados.
   - **Tabela por hierarquia (Class Table Inheritance):** Cada classe na hierarquia possui sua própria tabela, e as tabelas são relacionadas por chaves estrangeiras. Essa abordagem mantém a normalização, mas pode exigir junções complexas para recuperar os dados.
5. A escolha da estratégia adequada depende dos requisitos específicos da aplicação, considerando fatores como desempenho, complexidade e integridade dos dados.

Em resumo, a utilização de componentes de middleware como o JDBC, o emprego do padrão DAO e a compreensão das estratégias para mapear herança em bancos de dados relacionais são aspectos cruciais para o desenvolvimento de sistemas robustos, eficientes e de fácil manutenção.