Disciplina: Nível 3: Back-end Sem Banco Não Tem
Semestre: 1º - 2024
Aluno: DAVIDSON PEREIRA DE OLIVEIRA
Matrícula: 2022.11.54585-5
Repositório no GIT: [dev-davidson/RPG0016---BackEnd-sem-banco-n-o-tem (github.com)]

Relatório discente de acompanhamento

Título da Prática: RPG0016 - BackEnd sem banco não tem

Objetivo da Prática: Implementar persistência com base no middleware JDBC. Utilizar o padrão DAO (Data Access Object) no manuseio de dados. Implementar o mapeamento objeto-relacional em sistemas Java. Criar sistemas cadastrais com persistência em banco relacional.

👉 **1º Procedimento | Mapeamento Objeto-Relacional e DAO**

- **Cadastro Banco de dados**

```
public class CadastroDB {
    private static boolean mainloop = true;
    private static boolean navigate = false;

    public static void main(String[] args) {
    }

    // Método para pausar e retornar
    public static void pause() {
        System.out.println("Insira R para retornar!");
        Scanner input = new Scanner(System.in);
        while (navigate) {
            if ("r".equals(input.nextLine().toLowerCase())) {
                navigate = false;
            }
        }
    }

    // Verifica o CPF se já está cadastrado no banco
    public static boolean cpfExists(String cpf) {
        // Retorna true se o CPF já existe, caso contrário, retorna false
        return false;
    }

    // Verifica o CNPJ já está cadastrado no banco
    public static boolean cnpjExists(String cnpj) {
        // Retorna true se o CNPJ já existe, caso contrário, retorna false
        return false;
    }
```

```
    // verificar duplicatas durante a inclusão
    public static boolean checkForDuplicates(String tipo, String identificador) {
        if (tipo.equalsIgnoreCase("f")) {
            return cpfExists(identificador);
        } else if (tipo.equalsIgnoreCase("j")) {
            return cnpjExists(identificador);
        } else {
            return false;
        }
    }

    // verificar duplicatas durante a alteração
    public static boolean checkForDuplicatesDuringUpdate(String tipo, String identificador, String novoValor) {
        // Você pode adaptar a lógica de acordo com sua implementação específica
        return false;
    }
}
```

- **Cadastro BD Teste**

```
package cadastrodb;

import cadastrodb.model.PessoaFisica;
import cadastrodb.model.PessoaFisicaDAO;
import cadastrodb.model.PessoaJuridica;
import cadastrodb.model.PessoaJuridicaDAO;
import java.util.ArrayList;

public class CadastroDBTeste {

    public static void main(String[] args) {
        testePessoaFisica();
        testePessoaJuridica();
    }

    public static void testePessoaFisica() {
        // Incluir Pessoa Física para teste
        PessoaFisica tester = new PessoaFisica("",0,"","","","","","");
        PessoaFisicaDAO.incluir(tester);

        // Exibir Pessoas Físicas
        ArrayList<PessoaFisica> pessoas = PessoaFisicaDAO.getPessoas();
        pessoas.forEach((e) -> e.exibir());

        // Excluir Pessoa Física de teste
        PessoaFisicaDAO.excluir(tester.getId());
    }
```

```
    public static void testePessoaJuridica() {
        // Incluir Pessoa Jurídica para teste
        PessoaJuridica teste = new PessoaJuridica("12345678910",0,"Bruno Silva","Av.Brasil 500","Ipatinga","MG","10987654321","bruno@ipatinga.com.br");
        PessoaJuridicaDAO.incluir(teste);

        // Exibir Pessoas Jurídicas
        ArrayList<PessoaJuridica> pessoas = PessoaJuridicaDAO.getPessoas();
        pessoas.forEach((e) -> e.exibir());

        // Excluir Pessoa Jurídica de teste
        PessoaJuridicaDAO.excluir(teste.getId());
    }
}
```

- **Pessoa**

```
package cadastrodb.model;

public abstract class Pessoa {
    private int id;
    private String nome, logradouro, cidade, estado, telefone, email;

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

    public void exibir() {
        System.out.println("id: " + this.id);
        System.out.println("nome: " + this.nome);
        System.out.println("logradouro: " + this.logradouro);
        System.out.println("cidade: " + this.cidade);
        System.out.println("estado: " + this.estado);
        System.out.println("telefone: " + this.telefone);
        System.out.println("email: " + this.email);
    }

    // Getters e Setters
    public int getId() {
        return id;
    }
```

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

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getLogradouro() {
        return logradouro;
    }

    public void setLogradouro(String logradouro) {
        this.logradouro = logradouro;
    }

    public String getCidade() {
        return cidade;
    }

    public void setCidade(String cidade) {
        this.cidade = cidade;
    }

    public String getEstado() {
        return estado;
    }
    public void setEstado(String estado) {
        this.estado = estado;
    }

    public String getTelefone() {
        return telefone;
    }

    public void setTelefone(String telefone) {
        this.telefone = telefone;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

- **Cadastro Pessoa Física**

```
package cadastrodb.model;

public class PessoaFisica extends Pessoa {
    private String cpf;

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

    // Getter e Setter para CPF
    public String getCpf() {
        return cpf;
    }

    public void setCpf(String cpf) {
        this.cpf = cpf;
    }

    // Sobrescrita do método exibir() para incluir informações específicas de PessoaFisica
    @Override
    public void exibir() {
        super.exibir();
        System.out.println("CPF: " + this.cpf);
    }
}
```

- **Cadastro Pessoa Jurídica**

```
package cadastrodb.model;

public class PessoaJuridica extends Pessoa {
    private String cnpj;

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

    // Getter e Setter para CNPJ
    public String getCnpj() {
        return cnpj;
    }

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

    // Sobrescrita do método exibir() para incluir informações específicas de PessoaJuridica
    @Override
    public void exibir() {
        super.exibir();
        System.out.println("CNPJ: " + this.cnpj);
    }
}
```

Análise e Conclusão:
A - Qual a importância dos componentes de middleware, como o JDBC?

Os componentes de middleware, como o JDBC (Java Database Connectivity), desempenham um papel fundamental na integração de aplicativos Java com bancos de dados. O JDBC fornece uma interface de programação que permite que os desenvolvedores se comuniquem e interajam com diferentes bancos de dados usando a linguagem Java. A importância do JDBC reside na sua capacidade de abstrair os detalhes de conexão e interação com o banco de dados, fornecendo uma camada de abstração que simplifica o desenvolvimento de aplicativos que necessitam de acesso a dados persistentes.

B - Qual a diferença no uso de Statement ou PreparedStatement para a manipulação de dados?

A diferença fundamental entre **Statement** e **PreparedStatement** está na forma como os parâmetros são tratados. **Statement** é usado para consultas SQL estáticas, onde a consulta é definida uma vez e executada múltiplas vezes com diferentes valores. **PreparedStatement**, por outro lado, permite a criação de consultas parametrizadas, o que significa que você pode pré-compilar consultas SQL e vincular parâmetros a elas posteriormente. Isso oferece benefícios de desempenho e segurança, além de prevenir contra injeção de SQL.

C - 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 negócios. Com o DAO, as operações de acesso a dados são encapsuladas em classes específicas, o que torna mais fácil modificar ou substituir a camada de acesso a dados sem afetar outras partes do sistema. Isso promove a coesão e a separação de preocupações, tornando o código mais modular e fácil de entender, testar e manter.

D - Como a herança é refletida no banco de dados, quando lidamos com um modelo estritamente relacional?

No contexto de um modelo estritamente relacional, a herança pode ser refletida de várias maneiras no banco de dados. Uma abordagem comum é a utilização de tabelas separadas para cada classe na hierarquia de herança (conhecida como modelagem de tabela por classe). Isso implica em uma tabela para a classe base (superclasse) e uma tabela para cada classe derivada (subclasse), com um relacionamento entre elas para representar a herança. Outra abordagem é a modelagem de tabela única, onde todos os atributos das classes são armazenados em uma única tabela, com colunas adicionais para identificar o tipo de objeto. Essas abordagens têm suas vantagens e desvantagens, e a escolha depende dos requisitos específicos do sistema e das preferências de design.

- **CadastroBase**

```
import java.util.Scanner;

public class CadastroDBTeste {

    private static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        boolean sair = false;

        while (!sair) {
            System.out.println("Selecione uma opção:");
            System.out.println("1. Incluir");
            System.out.println("2. Alterar");
            System.out.println("3. Excluir");
            System.out.println("4. Exibir pelo ID");
            System.out.println("5. Exibir todos");
            System.out.println("0. Sair");

            int opcao = scanner.nextInt();
            scanner.nextLine(); // Limpar o buffer do scanner

            switch (opcao) {
                case 1:
                    incluir();
                    break;
                case 2:
                    alterar();
                    break;
                case 3:
                    excluir();
                    break;
```

```
                case 4:
                    exibirPorId();
                    break;
                case 5:
                    exibirTodos();
                    break;
                case 0:
                    sair = true;
                    break;
                default:
                    System.out.println("Opção inválida!");
            }
        }
    }

    private static void incluir() {
        System.out.println("Selecione o tipo (1 para Pessoa Física, 2 para Pessoa Jurídica):");
        int tipo = scanner.nextInt();
        scanner.nextLine(); // Limpar o buffer do scanner

        if (tipo == 1) {
            // Incluir pessoa física
            // Implementar lógica para receber os dados e adicionar no banco de dados
        } else if (tipo == 2) {
            // Incluir pessoa jurídica
            // Implementar lógica para receber os dados e adicionar no banco de dados
        } else {
            System.out.println("Tipo inválido!");
        }
    }

    private static void alterar() {
        // Implementar lógica para alterar dados no banco de dados
    }

    private static void excluir() {
        // Implementar lógica para excluir dados do banco de dados
    }

    private static void exibirPorId() {
        // Implementar lógica para exibir dados por ID do banco de dados
    }

    private static void exibirTodos() {
        // Implementar lógica para exibir todos os dados do banco de dados
    }
}
```

Análise e Conclusão:

A - Quais as diferenças entre a persistência em arquivo e a persistência em banco de dados?

Formato de Armazenamento: Em persistência em arquivo, os dados são armazenados em arquivos no sistema de arquivos do computador, enquanto na persistência em banco de dados, os dados são armazenados em tabelas em um sistema de gerenciamento de banco de dados (SGBD).
Recursos e Funcionalidades: Os bancos de dados oferecem recursos avançados de consulta, indexação, transações ACID (Atomicidade, Consistência, Isolamento, Durabilidade) e escalabilidade, o que os torna mais robustos para aplicações complexas. Os arquivos são mais simples e geralmente oferecem menos recursos de gerenciamento.

Concorrência e Acesso: Os bancos de dados oferecem mecanismos para controlar a concorrência e o acesso simultâneo aos dados por vários usuários, garantindo integridade e consistência. Em arquivos, o controle de acesso pode ser mais manual e propenso a erros.

Escalabilidade e Desempenho: Os bancos de dados podem escalar horizontal e verticalmente para lidar com grandes volumes de dados e demanda de usuários. Arquivos podem enfrentar limitações de tamanho e desempenho com o aumento da carga de trabalho.

B - Como o uso de operador lambda simplificou a impressão dos valores contidos nas entidades, nas versões mais recentes do Java?

O uso de operadores lambda simplifica a escrita de código ao permitir a definição de expressões inline para a implementação de interfaces funcionais. Nas versões mais recentes do Java, especialmente a partir do Java 8, os operadores lambda são amplamente utilizados para simplificar operações em coleções, como a iteração e processamento de elementos.

C - Por que métodos acionados diretamente pelo método main, sem o uso de um objeto, precisam ser marcados como static?

Os métodos acionados diretamente pelo método main precisam ser marcados como static porque o método main é estático. Isso significa que ele pertence à classe em si e não a instâncias específicas da classe.

Quando o programa é iniciado, não há instância da classe principal disponível, pois o método main é o ponto de entrada do programa e é chamado pela JVM (Java Virtual Machine) sem criar uma instância da classe. Portanto, para que outros métodos possam ser chamados diretamente do método main, eles também devem ser estáticos, garantindo que possam ser invocados sem a necessidade de uma instância da classe.