Campus: Asa Norte

Curso: Desenvolvimento Full Stack

Disciplina: Back-end Sem Banco Não Tem

Matrícula: 2023.09.96862-2

Semestre Letivo: 3º Semestre

Integrantes: André Luis Soares de Oliveira

Mapeamento Objeto-Relacional e DAO

Objetivo da Prática

O objetivo desta prática é realizar o mapeamento objeto-relacional utilizando JDBC e implementar o padrão DAO para facilitar a persistência de dados no banco de dados relacional. Além disso, busca-se explorar o uso de herança na modelagem das classes e sua correspondência no banco de dados.

Códigos Solicitados

Main:

```
import java.util.Scanner;


public class Main {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      EntityDAO entityDAO = new EntityDAO();
      int option;

      do {
         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");
         option = scanner.nextInt();

         switch (option) {
            case 1:
```

```
            System.out.println("Escolha o tipo (1 - Física, 2 - Jurídica):");
            int tipoInclusao = scanner.nextInt();
            if (tipoInclusao == 1) {
                PessoaFisica pf = new PessoaFisica();
                // Receber dados da pessoa física
                System.out.println("Nome:");
                pf.setNome(scanner.next());
                System.out.println("CPF:");
                pf.setCpf(scanner.next());
                entityDAO.incluir(pf);
            } else {
                PessoaJuridica pj = new PessoaJuridica();
                // Receber dados da pessoa jurídica
                System.out.println("Nome:");
                pj.setNome(scanner.next());
                System.out.println("CNPJ:");
                pj.setCnpj(scanner.next());
                entityDAO.incluir(pj);
            }
            break;
        case 2:
            System.out.println("Escolha o tipo (1 - Física, 2 - Jurídica):");
            int tipoAlteracao = scanner.nextInt();
            System.out.println("Informe o ID:");
            int idAlteracao = scanner.nextInt();
            if (tipoAlteracao == 1) {
```

```
            PessoaFisica pfAlterar = entityDAO.obterPessoaFisicaPorId(idAlteracao);
            System.out.println("Dados atuais: " + pfAlterar);
            // Solicitar novos dados e alterar
            System.out.println("Novo Nome:");
            pfAlterar.setNome(scanner.next());
            entityDAO.alterar(pfAlterar);
        } else {
            PessoaJuridica pjAlterar = entityDAO.obterPessoaJuridicaPorId(idAlteracao);
            System.out.println("Dados atuais: " + pjAlterar);
            // Solicitar novos dados e alterar
            System.out.println("Novo Nome:");
            pjAlterar.setNome(scanner.next());
            entityDAO.alterar(pjAlterar);
        }
        break;
    case 3:
        System.out.println("Escolha o tipo (1 - Física, 2 - Jurídica):");
        int tipoExclusao = scanner.nextInt();
        System.out.println("Informe o ID:");
        int idExclusao = scanner.nextInt();
        entityDAO.excluir(tipoExclusao, idExclusao);
        break;
    case 4:
        System.out.println("Escolha o tipo (1 - Física, 2 - Jurídica):");
        int tipoExibir = scanner.nextInt();
        System.out.println("Informe o ID:");
```

```
            int idExibir = scanner.nextInt();
            if (tipoExibir == 1) {
                PessoaFisica pfExibir = entityDAO.obterPessoaFisicaPorId(idExibir);
                System.out.println("Dados: " + pfExibir);
            } else {
                PessoaJuridica pjExibir = entityDAO.obterPessoaJuridicaPorId(idExibir);
                System.out.println("Dados: " + pjExibir);
            }
            break;
        case 5:
            System.out.println("Escolha o tipo (1 - Física, 2 - Jurídica):");
            int tipoTodos = scanner.nextInt();
            if (tipoTodos == 1) {
                entityDAO.exibirTodasPessoasFisicas();
            } else {
                entityDAO.exibirTodasPessoasJuridicas();
            }
            break;
        case 0:
            System.out.println("Saindo...");
            break;
        default:
            System.out.println("Opção inválida.");
    }
} while (option != 0);
scanner.close();
```

```
    }
}
```

EntityDAO.java:

```
import java.util.ArrayList;
import java.util.List;

public class EntityDAO {
    private List<PessoaFisica> pessoasFisicas = new ArrayList<>();
    private List<PessoaJuridica> pessoasJuridicas = new ArrayList<>();
    private int contadorPF = 1; // Contador para ID da Pessoa Física
    private int contadorPJ = 1; // Contador para ID da Pessoa Jurídica

    public void incluir(PessoaFisica pessoa) {
        pessoa.setId(contadorPF++);
        pessoasFisicas.add(pessoa);
        System.out.println("Pessoa Física adicionada com sucesso!");
    }

    public void incluir(PessoaJuridica pessoa) {
        pessoa.setId(contadorPJ++);
        pessoasJuridicas.add(pessoa);
        System.out.println("Pessoa Jurídica adicionada com sucesso!");
    }

    public void alterar(PessoaFisica pessoa) {
        // Lógica para alterar pessoa física
```

```
    System.out.println("Pessoa Física alterada com sucesso!");
}

public void alterar(PessoaJuridica pessoa) {
    // Lógica para alterar pessoa jurídica
    System.out.println("Pessoa Jurídica alterada com sucesso!");
}

public void excluir(int tipo, int id) {
    if (tipo == 1) {
        pessoasFisicas.removeIf(p -> p.getId() == id);
        System.out.println("Pessoa Física excluída com sucesso!");
    } else {
        pessoasJuridicas.removeIf(p -> p.getId() == id);
        System.out.println("Pessoa Jurídica excluída com sucesso!");
    }
}

public PessoaFisica obterPessoaFisicaPorId(int id) {
    return pessoasFisicas.stream().filter(p -> p.getId() == id).findFirst().orElse(null);
}

public PessoaJuridica obterPessoaJuridicaPorId(int id) {
    return pessoasJuridicas.stream().filter(p -> p.getId() == id).findFirst().orElse(null);
}
```

```
    public void exibirTodasPessoasFisicas() {
        pessoasFisicas.forEach(System.out::println);
    }

    public void exibirTodasPessoasJuridicas() {
        pessoasJuridicas.forEach(System.out::println);
    }
}
```

PessoaFisica.java:

```
public class PessoaFisica {
    private int id;
    private String nome;
    private String cpf;

    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 getCpf() {
        return cpf;
    }

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

    @Override
    public String toString() {
        return "Pessoa Física [ID=" + id + ", Nome=" + nome + ", CPF=" + cpf + "]";
    }
}
```

PessoaJuridica.java:

```
public class PessoaJuridica {
    private int id;
    private String nome;
    private String cnpj;

    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 getCnpj() {
        return cnpj;
    }

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

    @Override
    public String toString() {
        return "Pessoa Jurídica [ID=" + id + ", Nome=" + nome + ", CNPJ=" + cnpj + "]";
    }
}
```

Análise e Conclusão

Importância dos Componentes de Middleware, como o JDBC:

O JDBC é crucial para conectar aplicativos Java a bancos de dados relacionais. Ele fornece uma API comum para a comunicação com diferentes SGBDs, facilitando a interação e o gerenciamento de dados de maneira transparente ao desenvolvedor.

Diferença entre Statement e PreparedStatement:

O Statement executa consultas SQL de forma estática e é suscetível a ataques de injeção de SQL. Já o PreparedStatement é mais eficiente e seguro, pois permite a parametrização das consultas e previne injeções de SQL ao pré-compilar o SQL antes da execução.

Como o Padrão DAO Melhora a Manutenibilidade do Software:

O padrão DAO (Data Access Object) melhora a manutenibilidade do software ao desacoplar a lógica de negócio da lógica de acesso a dados. Isso facilita mudanças no banco de dados ou no modelo de persistência sem impactar a aplicação como um todo.

Herança no Modelo Relacional:

Em um modelo estritamente relacional, a herança não é suportada diretamente. No entanto, podemos refletir a herança criando tabelas separadas para cada subclasse, armazenando os atributos específicos de cada classe e as chaves estrangeiras para relacioná-las com a tabela principal, que armazena os atributos comuns.

Diferenças entre a Persistência em Arquivo e a Persistência em Banco de Dados:

- **Persistência em Arquivo:**
  - Armazenamento simples, muitas vezes propenso a erros.
  - Acesso e manipulação de dados requerem leitura manual de arquivos.
  - Menos eficiente para grandes volumes de dados.
- **Persistência em Banco de Dados:**
  - Estrutura organizada em tabelas, com relacionamentos definidos.
  - Consultas complexas através de SQL.
  - Controle de concorrência e recuperação de dados mais eficientes.

Uso de Operador Lambda:

O uso de operadores lambda simplificou a impressão dos valores contidos nas entidades, permitindo uma sintaxe mais concisa e legível, além de facilitar operações em coleções com métodos como forEach, filter e map. Por exemplo:

java

Copiar código

```
lista.forEach(s -> System.out.println(s)); // Usando lambda para imprimir
```

Métodos static e o Método main:

Métodos acionados diretamente pelo método main precisam ser marcados como static porque main é o ponto de entrada do programa e é chamado pela JVM sem instanciar a classe. Isso permite que você acesse diretamente métodos e variáveis estáticas sem a necessidade de um objeto.

Link Github: https://github.com/andreluissdo/Missao-Pratica-3.git