	<p>Universidade Estácio Campus Belém Curso de Desenvolvimento Full Stack Relatório da Missão Prática 3 - Mundo 3</p>
Disciplina:	RPG0016 - Backend sem banco não tem
Nome:	Cleyton Isamu Muto
Turma:	2023.1

Criação de aplicativo Java, com acesso ao BD SQL Server através do middleware JDBC

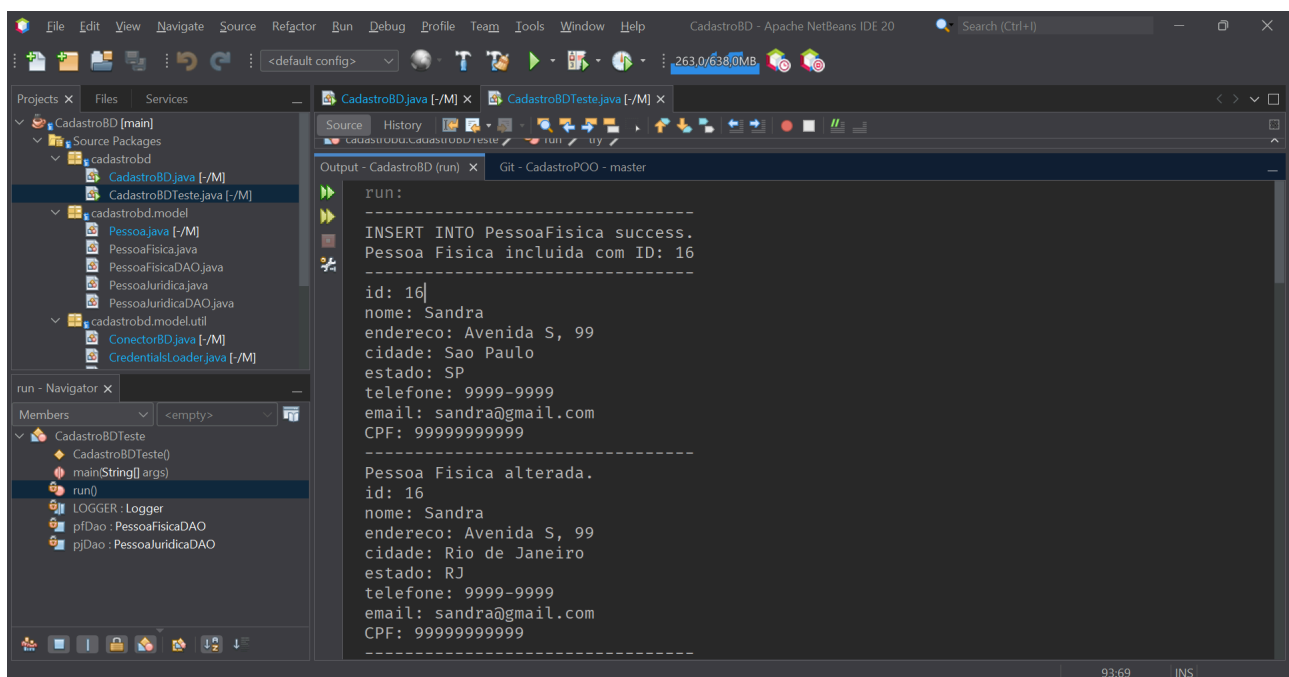
1. Título da Prática: “1º Procedimento | Mapeamento Objeto-Relacional e DAO”

2. 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.
- No final do exercício, o aluno terá criado um aplicativo cadastral com uso do SQL Server na persistência de dados.

3. Códigos solicitados: estão no repositório <https://github.com/cleytonmuto/CadastroBD> e também no Anexo I, no final do relatório.

4. Resultados da execução dos códigos



```

run:
-----
INSERT INTO PessoaFisica success.
Pessoa Fisica incluída com ID: 16
-----
id: 16|
nome: Sandra
endereço: Avenida S, 99
cidade: Sao Paulo
estado: SP
telefone: 9999-9999
email: sandra@gmail.com
CPF: 99999999999
-----
Pessoa Fisica alterada.
id: 16
nome: Sandra
endereço: Avenida S, 99
cidade: Rio de Janeiro
estado: RJ
telefone: 9999-9999
email: sandra@gmail.com
CPF: 99999999999
-----

```

Figura 1. Execução da classe CadastroBDTeste: inserção e alteração de Pessoa Física.

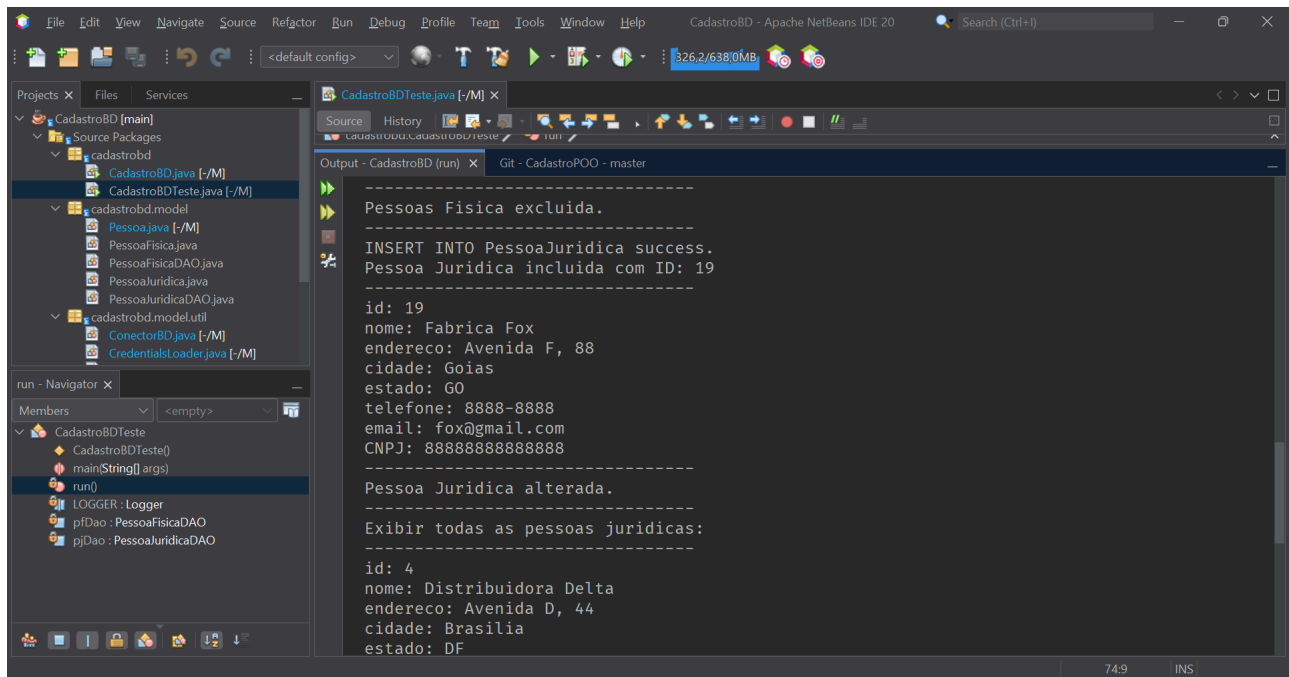


Figura 2. Execução da classe CadastroBDTeste: exclusão de Pessoa Física, inserção e alteração de Pessoa Jurídica.

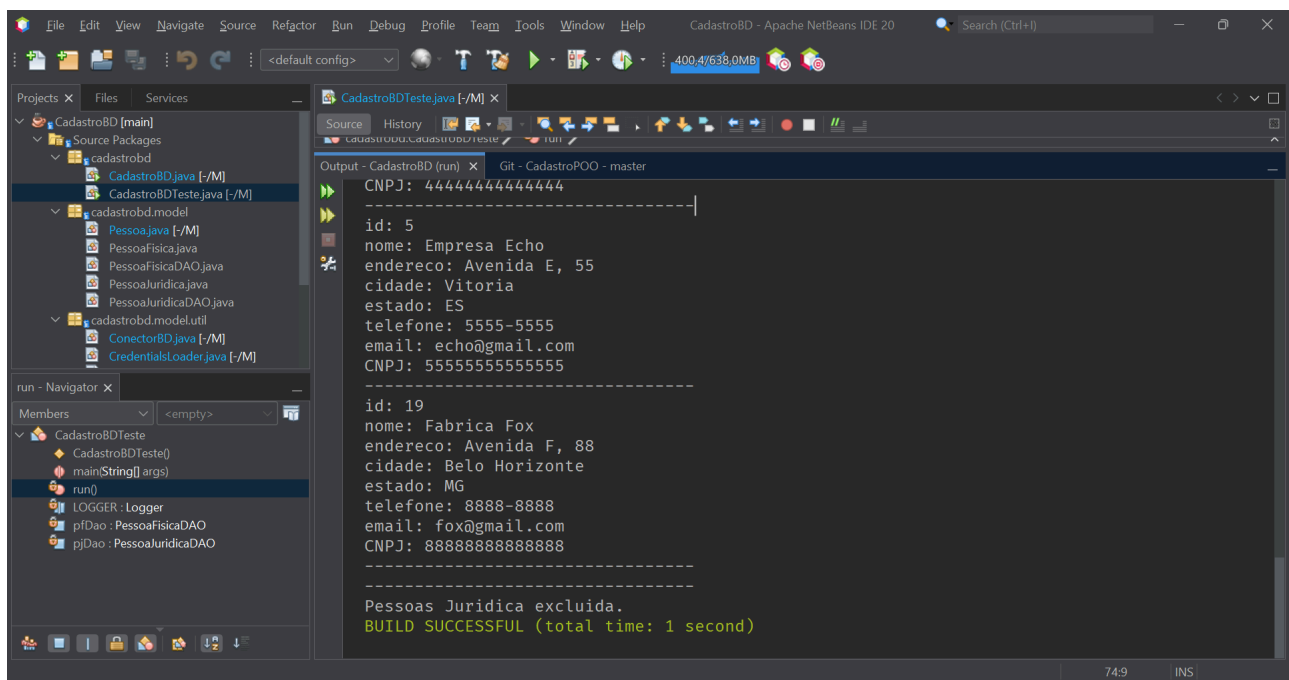


Figura 3. Execução da classe CadastroBDTeste: exclusão de Pessoa Jurídica.

5. Análise e Conclusão

(a) Qual a importância dos componentes de middleware, como o JDBC?

JDBC (*Java Database Connectivity*) é uma interface de programação de aplicações (*API - Application Program Interface*) que fornece acesso universal a fonte de dados em Java. Com JDBC é possível interligar aplicações Java a bancos de dados relacionais SQL, planilhas Excel e até arquivos textos [1]. Sua importância consiste em agir como intermediário entre a aplicação e a fonte de dados, de modo a permitir a comunicação e o intercâmbio de dados, de maneira padronizada, independente do banco de dados; basta que haja um driver adequado do banco de dados, configurado na aplicação e que este BD seja relacional, padrão SQL. Como consequência do uso de JDBC, há a maior portabilidade de aplicações Java entre diferentes plataformas, a simplificação do desenvolvimento com comandos padronizados e a interoperabilidade entre sistemas heterogêneos.

(b) Qual a diferença no uso de *Statement* ou *PreparedStatement* para a manipulação de dados?

Ambas as classes *Statement* e *PreparedStatement* são utilizadas para executar comandos SQL em Java; entretanto, diferem em características e desempenho [2]:

- *Statement* é utilizada para executar comandos SQL baseados em String.
- *PreparedStatement* é utilizada para executar comandos SQL parametrizados.

Em relação à segurança, a classe *PreparedStatement* é preferencial quando se trata de prevenir ataques de injeção SQL, pois permite definir parâmetros de consulta que são tratados de forma segura pelo banco de dados. Isso significa que os valores dos parâmetros são tratados como dados e não como parte da consulta SQL em si, reduzindo significativamente o risco de injeção de código malicioso.

Em termos de desempenho, *PreparedStatement* é geralmente mais eficiente do que *Statement*, especialmente quando a mesma consulta é executada várias vezes com valores diferentes. Isso ocorre porque o banco de dados pode compilar a consulta apenas uma vez e reutilizá-la com diferentes parâmetros.

Em relação à legibilidade e manutenção do código, *PreparedStatement* geralmente torna o código mais legível e mais fácil de manter, especialmente quando se lida com consultas complexas, com muitos parâmetros. Isso ocorre porque os parâmetros podem ser definidos de forma clara e separada da consulta SQL, o que torna mais fácil entender o que a consulta faz.

Em relação aos tipos de dados, com *PreparedStatement*, o JDBC pode inferir automaticamente o tipo de dados dos parâmetros, o que simplifica o código. Com *Statement*, é necessário lidar manualmente com a conversão dos tipos de dados.

Ainda, alguns bancos de dados podem armazenar cache de consultas, o que pode melhorar ainda mais o desempenho em consultas subsequentes com *PreparedStatement*.

(c) Como o padrão DAO melhora a manutenibilidade do software?

O padrão DAO (*Data Access Object*) melhora a manutenibilidade do software, ao separar ou isolar a código responsável pelo acesso e manipulação dos dados, do restante da lógica de negócios da aplicação. Isso facilita a manutenção do código, uma vez que as alterações nos

requisitos de acesso aos dados podem ser realizadas de maneira separada, sem afetar diretamente outras partes do sistema.

O padrão DAO também promove a reutilização de código ao encapsular a lógica de acesso a dados em classes específicas. Assim, se houver a necessidade de modificar a forma como os dados são acessados (por exemplo, mudança de banco de dados relacional para um banco de dados não-relacional), apenas as classes DAO precisam ser modificadas, de modo a manter o restante do código inalterado.

Outro benefício é que o DAO torna o código mais legível e compreensível, pois fornece uma abstração clara e consistente para as operações de acesso a dados em todo o sistema. Isso facilita a colaboração entre os membros da equipe e a identificação e correção de problemas relacionados ao acesso a dados.

(d) Como a herança é refletida no banco de dados, quando lidamos com um modelo estritamente relacional?

Quando se lida com um modelo estritamente relacional em banco de dados, a herança geralmente é refletida com uso de uma técnica conhecida como "tabelas de junção" (ou "tabelas de associação"). Esta abordagem é chamada de modelagem de herança de tabela única, ou modelagem de herança de tabela por classe. Eis como funciona:

- Tabela Base (ou Superclasse): Uma tabela é criada para representar a classe base ou superclasse. Esta tabela contém os atributos comuns a todas as subclasses.
- Tabelas de Subclasse: Para cada subclasse, é criada uma tabela separada contendo apenas os atributos específicos daquela subclasse. Essas tabelas também terão uma chave estrangeira que referencia a tabela base.
- Chave Estrangeira: A chave primária da tabela base é usada como chave estrangeira nas tabelas de subclasse para estabelecer a relação entre elas.
- Junção de Tabelas: Quando uma consulta é feita para recuperar dados de uma hierarquia de herança, é necessário fazer uma junção (JOIN) entre a tabela base e as tabelas de subclasse usando as chaves primárias e estrangeiras correspondentes.

Essas abordagens permitem que a hierarquia de herança seja representada de forma eficiente em um modelo relacional, de modo a manter a integridade referencial entre as tabelas e permitir consultas que recuperam dados de todas as classes relacionadas na hierarquia de herança.

1. Título da Prática: “2º Procedimento | Alimentando a Base”

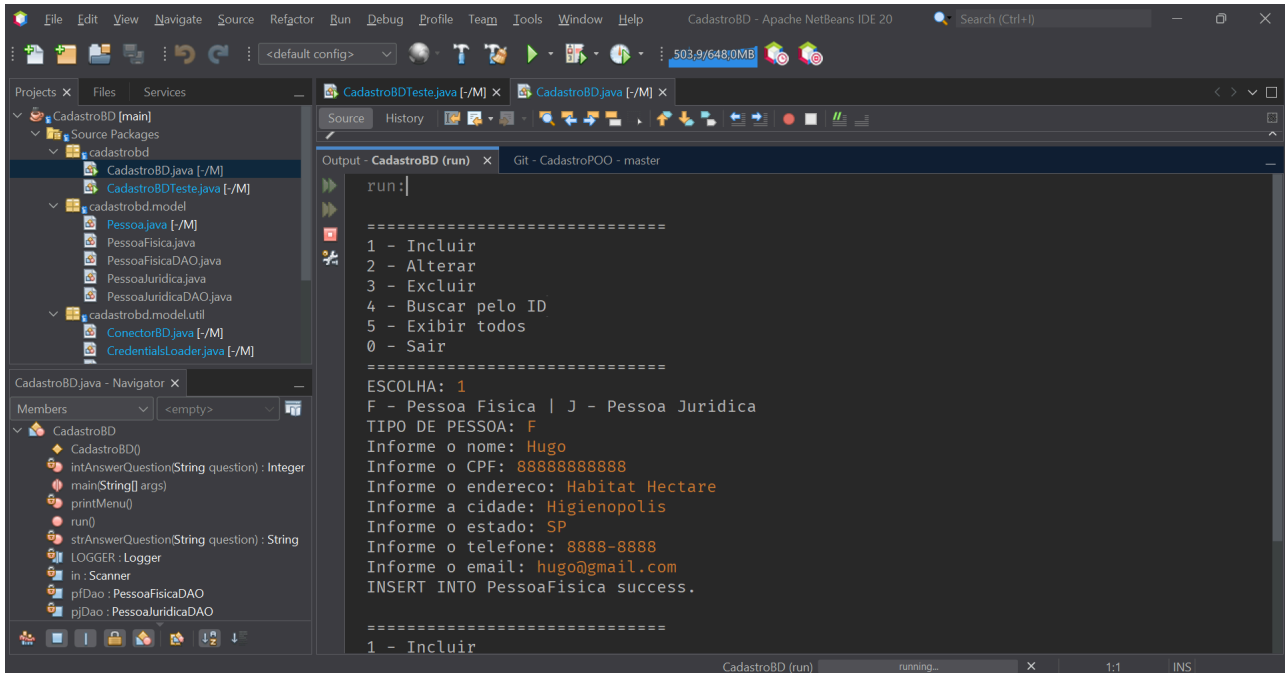
2. Objetivo da Prática

- Identificar os requisitos de um sistema e transformá-los no modelo adequado.
- Utilizar ferramentas de modelagem para bases de dados relacionais.
- Explorar a sintaxe SQL na criação das estruturas do banco (DDL).
- Explorar a sintaxe SQL na consulta e manipulação de dados (DML)

- No final do exercício, o aluno terá vivenciado a experiência de modelar a base de dados para um sistema simples, além de implementá-la, através da sintaxe SQL, na plataforma do SQL Server.

3. Códigos solicitados: estão no repositório <https://github.com/cleytonmuto/CadastroBD> e também no Anexo I, no final do relatório.

4. Resultados da execução dos códigos



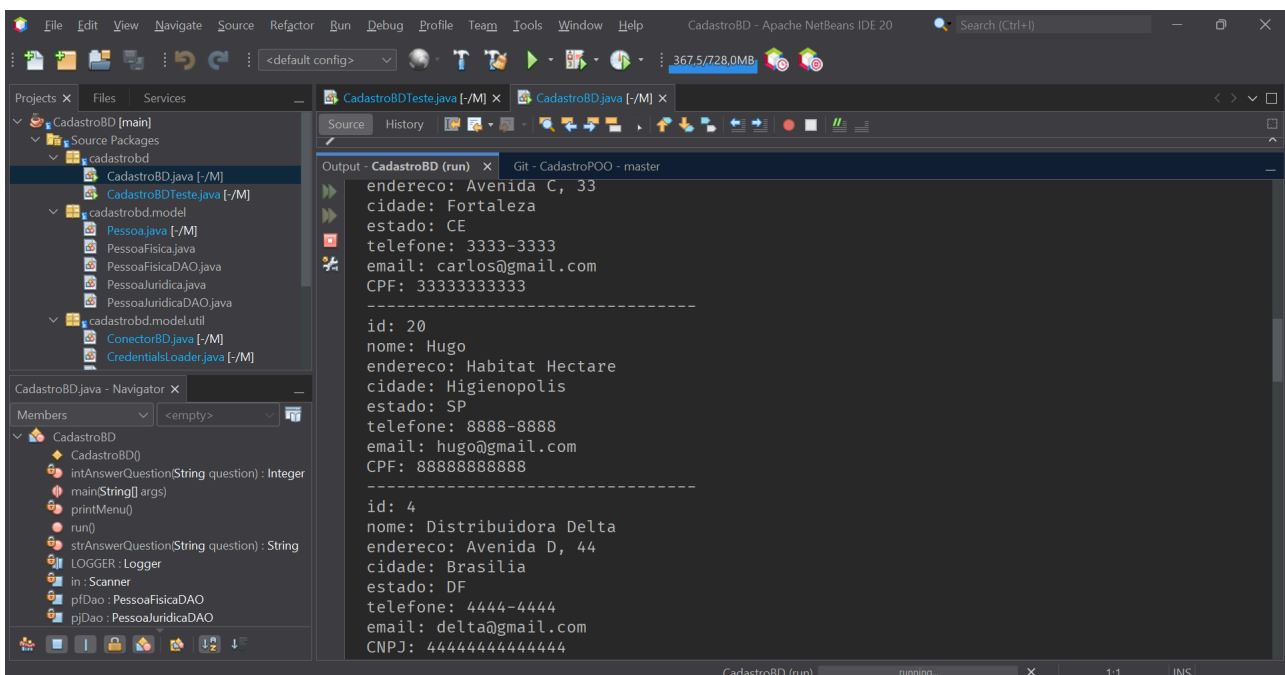
```

run:

=====
1 - Incluir
2 - Alterar
3 - Excluir
4 - Buscar pelo ID
5 - Exibir todos
0 - Sair
=====
ESCOLHA: 1
F - Pessoa Fisica | J - Pessoa Juridica
TIPO DE PESSOA: F
Informe o nome: Hugo
Informe o CPF: 88888888888
Informe o endereco: Habitat Hectare
Informe a cidade: Higienopolis
Informe o estado: SP
Informe o telefone: 8888-8888
Informe o email: hugo@gmail.com
INSERT INTO PessoaFisica success.
=====
1 - Incluir

```

Figura 4. Execução da classe CadastroBD: opção 1 - Incluir.



```

endereco: Avenida C, 33
cidade: Fortaleza
estado: CE
telefone: 3333-3333
email: carlos@gmail.com
CPF: 33333333333
-----
id: 20
nome: Hugo
endereco: Habitat Hectare
cidade: Higienopolis
estado: SP
telefone: 8888-8888
email: hugo@gmail.com
CPF: 88888888888
-----
id: 4
nome: Distribuidora Delta
endereco: Avenida D, 44
cidade: Brasilia
estado: DF
telefone: 4444-4444
email: delta@gmail.com
CNPJ: 4444444444444444

```

Figura 5. Execução da classe CadastroBD. Pessoa incluída com sucesso.

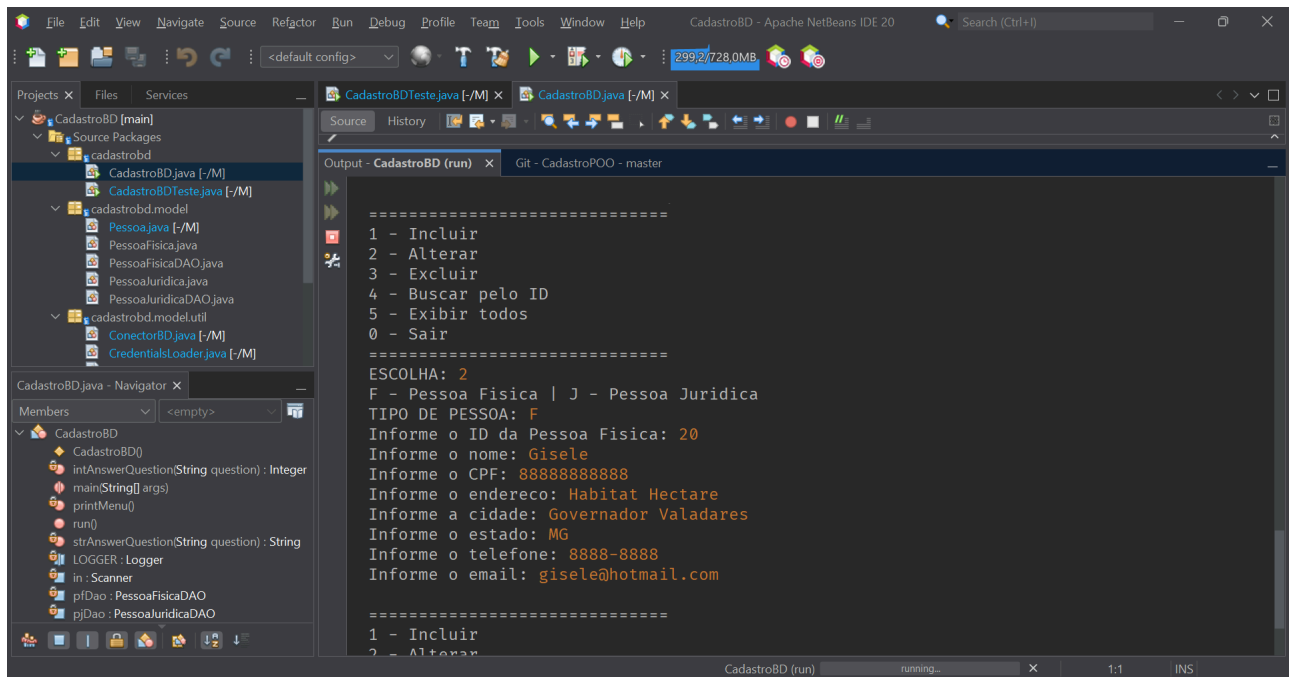


Figura 6. Execução da classe CadastroBD: opção 2 - Alterar.

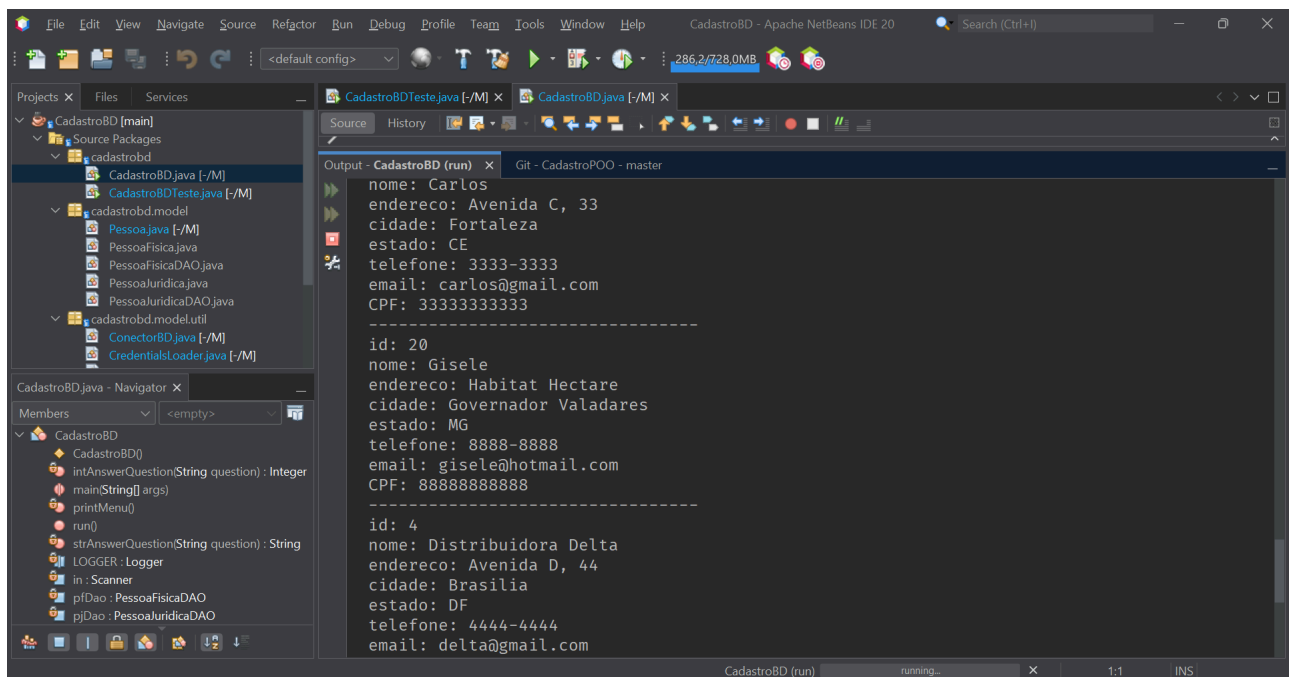


Figura 7. Execução da classe CadastroBD. Pessoa alterada com sucesso.

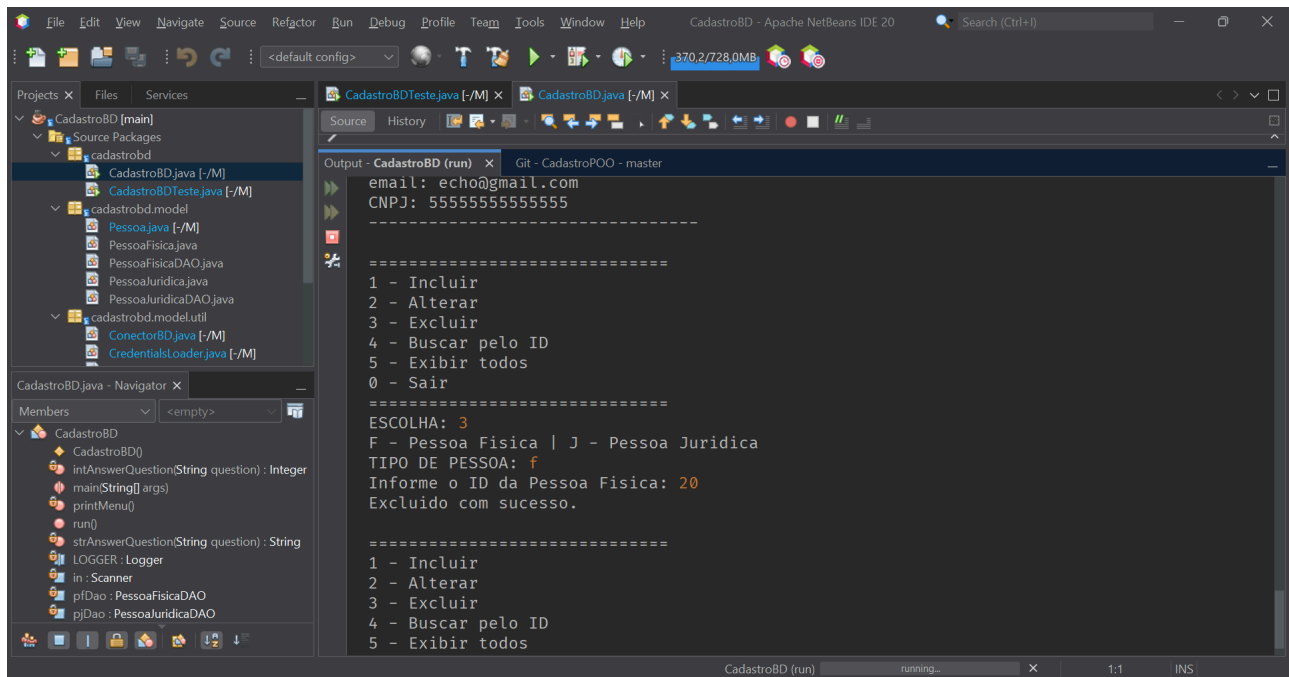


Figura 8. Execução da classe CadastroBD. Opção 3 - Excluir.

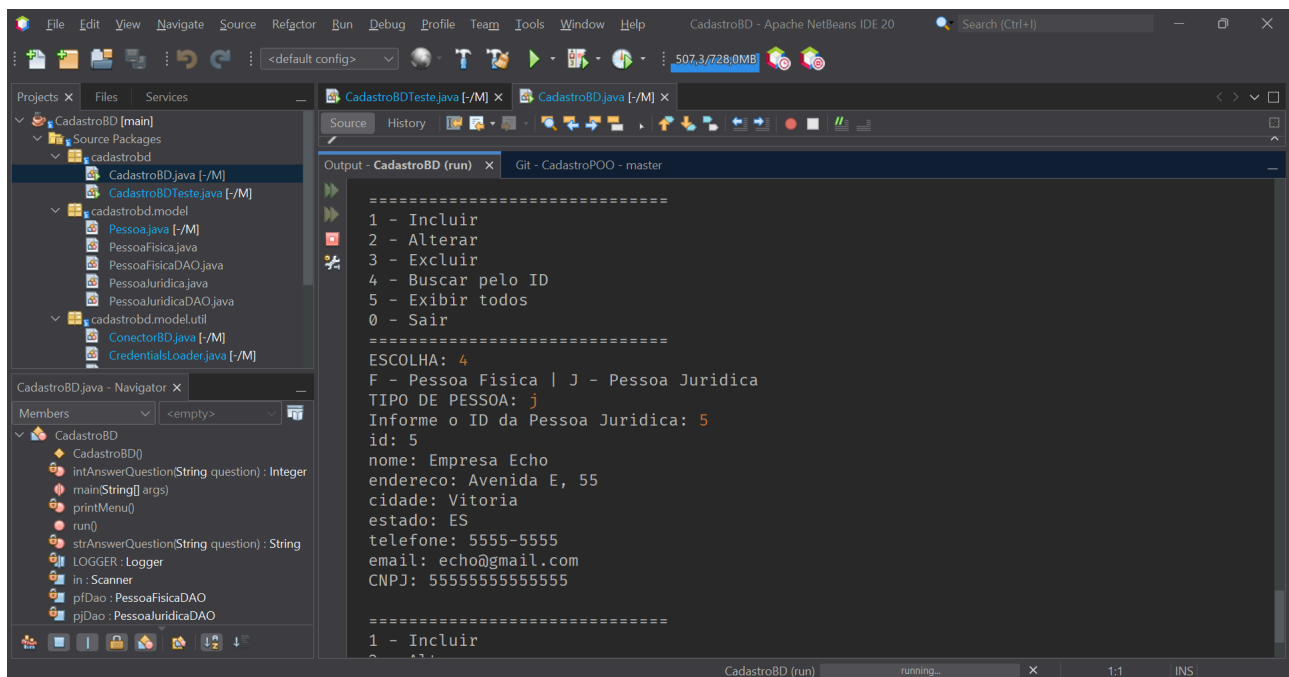


Figura 9. Execução da classe CadastroBD. Opção 4 - Buscar pelo ID.

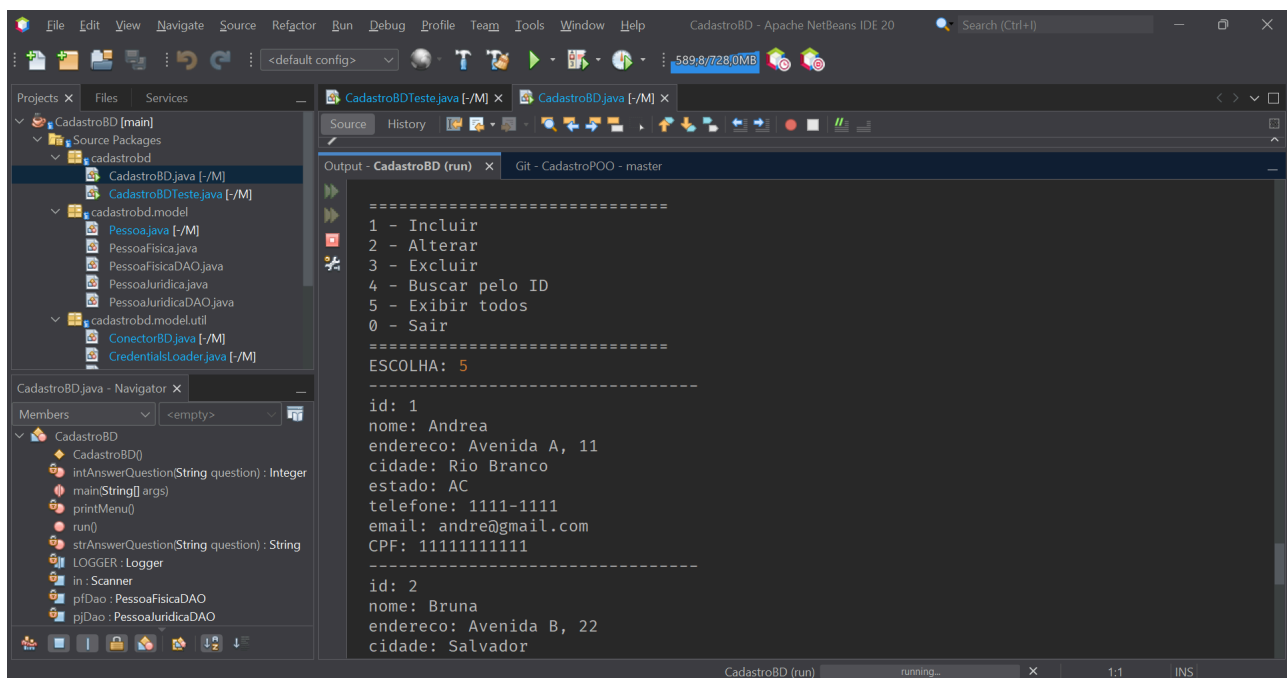


Figura 10. Execução da classe CadastroBD. Opção 5 - Exibir todos.

5. Análise e Conclusão

(a) Quais as diferenças entre a persistência em arquivo e a persistência em banco de dados?

A persistência em arquivos consiste no armazenamento de dados em um sistema de arquivos em formatos binário, por exemplo, arquivos com extensão .dat, .bin, .xls, .xlsx; ou arquivos texto, por exemplo, com extensões .csv, .json, .xml, são reconhecidamente arquivos de dados. É uma forma simples e direta de salvar dados, mas não é facilmente escalável, inicialmente segura ou eficiente, para consultas complexas e manipulação de dados quando comparada à persistência em banco de dados.

Por outro lado, a persistência em bancos de dados utiliza sistemas de gerenciamento de banco de dados (SGBD) para armazenar, recuperar e gerenciar dados de maneira estruturada, com uso de tabelas, índices, suporta operações complexas e transações. Também oferece recursos avançados como atomicidade, consistência, isolamento, durabilidade (ACID), além de segurança, backup e otimizações para acessos simultâneos.

(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, a partir do Java 8, simplificou a impressão de valores contidos em entidades, ao permitir que desenvolvedores escrevam código mais legível e conciso, sem a necessidade de criar classes anônimas para operações simples. Com operadores lambda, é possível passar comportamentos de forma direta e declarativa.

Por exemplo, para imprimir os valores de uma lista, antes do Java 8, seria necessário criar um loop explícito. Com lambdas, isso pode ser feito de maneira simplificada usando métodos de stream e operações como **forEach**

```
lista.forEach(elemento -> System.out.println(elemento));
```


Esse código substitui várias linhas de um loop for ou loop aprimorado for-each, tradicionais, ao mesmo tempo que torna o código mais expressivo, focado no que realmente se deseja realizar (a ação de imprimir), e não em como realizar o loop ou controle de fluxo.

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

O método main é o primeiro método a ser executado quando uma classe Java é compilada e executada, caso, obviamente, a classe contenha um método main. A principal razão pela qual o método main é definido como estático, é justamente para conceder a possibilidade de ser diretamente executado sem necessidade de instanciar um objeto do tipo da classe, para depois invocar o método main, ou seja, uma comodidade para evitar código desnecessário.

Por definição, atributos ou métodos estáticos pertencem à classe onde são definidos, e não ao objeto instanciado a partir dessa classe. Assim, se o atributo ou método forem públicos, podem ser referenciados externamente por outras classes, através somente do nome da classe e não por objeto. Contudo, métodos estáticos, em suas implementações, ao tentar invocar outros métodos sem uso de objetos de referência, somente podem invocar diretamente outros métodos estáticos. Logo, ambos métodos precisam ser estáticos, tanto o método que invoca o outro método, como o método invocado.

Referências

[1] **“Java JDBC API”**

Disponível em <https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc>

Acesso em 1º de maio de 2024.

[2] **“*Difference between Statement and PreparedStatement*”**

Disponível em <https://www.baeldung.com/java-statement-preparedstatement>

Acesso em 1º de maio de 2024.

Anexo I: códigos do projeto

```
package cadastrobd;

import java.util.ArrayList;
import java.util.Scanner;

import cadastrobd.model.PessoaFisica;
import cadastrobd.model.PessoaFisicaDAO;
import cadastrobd.model.PessoaJuridica;
import cadastrobd.model.PessoaJuridicaDAO;

import java.sql.SQLException;
import java.util.logging.Logger;
import java.util.logging.Level;

/**
 *
 * @author Cleyton
 */
public class CadastroBD {

    private static final Logger LOGGER = Logger.getLogger(CadastroBD.class.getName());
    private Scanner in;
    private PessoaFisicaDAO pfDao;
    private PessoaJuridicaDAO pjDao;

    public CadastroBD() {
        in = new Scanner(System.in);
        pfDao = new PessoaFisicaDAO();
        pjDao = new PessoaJuridicaDAO();
    }

    private String strAnswerQuestion(String question) {
        System.out.print(question);
        return in.nextLine();
    }

    private Integer intAnswerQuestion(String question) {
        System.out.print(question);
        String strValue = in.nextLine();
        Integer intValue = 0;
        try {
            intValue = Integer.valueOf(strValue);
        }
        catch (NumberFormatException e) {
            LOGGER.log(Level.SEVERE, e.toString(), e);
        }
        return intValue;
    }

    private void printMenu() {
        System.out.println("\n=====");
        System.out.println("1 - Incluir");
        System.out.println("2 - Alterar");
        System.out.println("3 - Excluir");
        System.out.println("4 - Buscar pelo ID");
        System.out.println("5 - Exibir todos");
        System.out.println("0 - Sair");
        System.out.println("=====");
    }

    public void run() {
        int opcao = -1;
        while (opcao != 0) {
            printMenu();
            opcao = intAnswerQuestion("ESCOLHA: ");
            switch (opcao) {
                case 1: {
                    System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
                    String escolhaIncluir = strAnswerQuestion("TIPO DE PESSOA: ").toUpperCase();
                    if (escolhaIncluir.equals("F")) {
                        String nome = strAnswerQuestion("Informe o nome: ");

```

```

        String cpf = strAnswerQuestion("Informe o CPF: ");
        String endereco = strAnswerQuestion("Informe o endereco: ");
        String cidade = strAnswerQuestion("Informe a cidade: ");
        String estado = strAnswerQuestion("Informe o estado: ");
        String telefone = strAnswerQuestion("Informe o telefone: ");
        String email = strAnswerQuestion("Informe o email: ");
        try {
            pfDao.incluir(new PessoaFisica(null, nome, endereco, cidade, estado,
telefone, email, cpf));
        }
        catch (SQLException e) {
            LOGGER.log(Level.SEVERE, e.toString(), e);
        }
    }
    else if (escolhaIncluir.equals("J")) {
        String nome = strAnswerQuestion("Informe o nome: ");
        String cnpj = strAnswerQuestion("Informe o CNPJ: ");
        String endereco = strAnswerQuestion("Informe o endereco: ");
        String cidade = strAnswerQuestion("Informe a cidade: ");
        String estado = strAnswerQuestion("Informe o estado: ");
        String telefone = strAnswerQuestion("Informe o telefone: ");
        String email = strAnswerQuestion("Informe o email: ");
        try {
            pjDao.incluir(new PessoaJuridica(null, nome, endereco, cidade,
estado, telefone, email, cnpj));
        }
        catch (SQLException e) {
            LOGGER.log(Level.SEVERE, e.toString(), e);
        }
    }
    else {
        System.out.println("Erro: Escolha Invalida!");
    }
}; break;
case 2: {
    System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
    String escolhaAlterar = strAnswerQuestion("TIPO DE PESSOA: ").toUpperCase();
    if (escolhaAlterar.equals("F")) {
        try {
            Integer id = intAnswerQuestion("Informe o ID da Pessoa Fisica: ");
            PessoaFisica pf = pfDao.getPessoa(id);
            if (pf != null) {
                pf.setNome(strAnswerQuestion("Informe o nome: "));
                pf.setCpf(strAnswerQuestion("Informe o CPF: "));
                pf.setEndereco(strAnswerQuestion("Informe o endereco: "));
                pf.setCidade(strAnswerQuestion("Informe a cidade: "));
                pf.setEstado(strAnswerQuestion("Informe o estado: "));
                pf.setTelefone(strAnswerQuestion("Informe o telefone: "));
                pf.setEmail(strAnswerQuestion("Informe o email: "));
                pfDao.alterar(pf);
            }
            else {
                System.out.println("ID nao encontrado!");
            }
        }
        catch (NullPointerException | SQLException e) {
            LOGGER.log(Level.SEVERE, e.toString(), e);
        }
    }
    else if (escolhaAlterar.equals("J")) {
        try {
            Integer id = intAnswerQuestion("Informe o ID da Pessoa Juridica: ");
            PessoaJuridica pj = pjDao.getPessoa(id);
            if (pj != null) {
                pj.setNome(strAnswerQuestion("Informe o nome: "));
                pj.setCnpj(strAnswerQuestion("Informe o CNPJ: "));
                pj.setEndereco(strAnswerQuestion("Informe o endereco: "));
                pj.setCidade(strAnswerQuestion("Informe a cidade: "));
                pj.setEstado(strAnswerQuestion("Informe o estado: "));
                pj.setTelefone(strAnswerQuestion("Informe o telefone: "));
                pj.setEmail(strAnswerQuestion("Informe o email: "));
                pjDao.alterar(pj);
            }
        }
    }
}

```

```

        else {
            System.out.println("ID nao encontrado!");
        }
    }
    catch (NullPointerException | SQLException e) {
        LOGGER.log(Level.SEVERE, e.toString(), e);
    }
}
else {
    System.out.println("Erro: Escolha Invalida!");
}
}; break;
case 3: {
    System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
    String escolhaExcluir = strAnswerQuestion("TIPO DE PESSOA: ").toUpperCase();
    if (escolhaExcluir.equals("F")) {
        try {
            Integer id = intAnswerQuestion("Informe o ID da Pessoa Fisica: ");
            PessoaFisica pf = pfDao.getPessoa(id);
            if (pf != null) {
                pfDao.excluir(pf);
                System.out.println("Excluido com sucesso.");
            }
            else {
                System.out.println("ID nao encontrado.");
            }
        }
        catch (NullPointerException | SQLException e) {
            LOGGER.log(Level.SEVERE, e.toString(), e);
        }
    }
    else if (escolhaExcluir.equals("J")) {
        try {
            Integer id = intAnswerQuestion("Informe o ID da Pessoa Juridica: ");
            PessoaJuridica pj = pjDao.getPessoa(id);
            if (pj != null) {
                pjDao.excluir(pj);
                System.out.println("Excluido com sucesso.");
            }
            else {
                System.out.println("ID nao encontrado.");
            }
        }
        catch (NullPointerException | SQLException e) {
            LOGGER.log(Level.SEVERE, e.toString(), e);
        }
    }
    else {
        System.out.println("Erro: Escolha Invalida!");
    }
}; break;
case 4: {
    System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
    String escolhaExibir = strAnswerQuestion("TIPO DE PESSOA: ").toUpperCase();
    if (escolhaExibir.equals("F")) {
        try {
            PessoaFisica pf = pfDao.getPessoa(intAnswerQuestion("Informe o ID da
Pessoa Fisica: "));
            if (pf != null) {
                pf.exibir();
            }
        }
        catch (SQLException e){
            System.err.println("Pessoa nao encontrada!");
            LOGGER.log(Level.SEVERE, e.toString(), e);
        }
    }
    else if (escolhaExibir.equals("J")) {
        try {
            PessoaJuridica pj = pjDao.getPessoa(intAnswerQuestion("Informe o ID
da Pessoa Juridica: "));
            if (pj != null) {
                pj.exibir();
            }
        }
    }
}
}

```

```

        }
        catch (SQLException e){
            System.err.println("Pessoa nao encontrada!");
            LOGGER.log(Level.SEVERE, e.toString(), e);
        }
    }
    else {
        System.out.println("Erro: Escolha Invalida!");
    }
}; break;
case 5: {
    try {
        ArrayList<PessoaFisica> listaPf = pfDao.getPessoas();
        for (PessoaFisica pessoa : listaPf) {
            System.out.println("-----");
            pessoa.exibir();
        }
        System.out.println("-----");
        ArrayList<PessoaJuridica> listaPj = pjDao.getPessoas();
        for (PessoaJuridica pessoa : listaPj) {
            pessoa.exibir();
            System.out.println("-----");
        }
    }
    catch (SQLException e) {
        LOGGER.log(Level.SEVERE, e.toString(), e);
    }
}; break;
default: System.out.println("Escolha invalida!");
}
}
}

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    new CadastroBD().run();
}
}

```

```

package cadastrobd;

import java.sql.SQLException;
import java.util.ArrayList;

import cadastrobd.model.PessoaFisica;
import cadastrobd.model.PessoaFisicaDAO;
import cadastrobd.model.PessoaJuridica;
import cadastrobd.model.PessoaJuridicaDAO;

import java.util.logging.Logger;
import java.util.logging.Level;

/**
 *
 * @author Cleyton
 */
public class CadastroBDTeste {

    private static final Logger LOGGER = Logger.getLogger(CadastroBDTeste.class.getName());

    private final PessoaFisicaDAO pfDao;
    private final PessoaJuridicaDAO pjDao;

    public CadastroBDTeste() {
        pfDao = new PessoaFisicaDAO();
        pjDao = new PessoaJuridicaDAO();
    }
}

```

```

private void run() {
    PessoaFisica pf = new PessoaFisica(null, "Sandra", "Avenida S, 99", "Sao Paulo",
        "SP", "9999-9999", "sandra@gmail.com", "999999999999");

    if (pf.getNome() == null || pf.getNome().trim().isEmpty()) {
        System.out.println("'nome' cannot be empty or null.");
        return;
    }

    try {
        System.out.println("-----");
        System.out.println("Pessoa Fisica incluida com ID: " + pfDao.incluir(pf));
        System.out.println("-----");
        pf.exibir();
        pf.setCidade("Rio de Janeiro");
        pf.setEstado("RJ");
        pfDao.alterar(pf);
        System.out.println("-----");
        System.out.println("Pessoa Fisica alterada.");
        pf.exibir();
        ArrayList<PessoaFisica> listaPf = pfDao.getPessoas();
        System.out.println("-----");
        System.out.println("Exibir todas as pessoas fisicas:");
        for (PessoaFisica pessoa : listaPf) {
            System.out.println("-----");
            pessoa.exibir();
        }
        System.out.println("-----");
        pfDao.excluir(pf);
        System.out.println("-----");
        System.out.println("Pessoa Fisica excluida.");
        pfDao.close();
    }
    catch (SQLException e) {
        LOGGER.log(Level.SEVERE, e.toString(), e);
    }

    PessoaJuridica pj = new PessoaJuridica(null, "Fabrica Fox", "Avenida F, 88", "Goiias",
        "GO", "8888-8888", "fox@gmail.com", "8888888888888888");

    if (pj.getNome() == null || pj.getNome().trim().isEmpty()) {
        System.out.println("'nome' cannot be empty or null.");
        return;
    }

    try {
        System.out.println("-----");
        System.out.println("Pessoa Juridica incluida com ID: " + pjDao.incluir(pj));
        System.out.println("-----");
        pj.exibir();
        pj.setCidade("Belo Horizonte");
        pj.setEstado("MG");
        pjDao.alterar(pj);
        System.out.println("-----");
        System.out.println("Pessoa Juridica alterada.");
        ArrayList<PessoaJuridica> listaPj = pjDao.getPessoas();
        System.out.println("-----");
        System.out.println("Exibir todas as pessoas juridicas:");
        for (PessoaJuridica pessoa : listaPj) {
            System.out.println("-----");
            pessoa.exibir();
        }
        System.out.println("-----");
        pjDao.excluir(pj);
        System.out.println("-----");
        System.out.println("Pessoa Juridica excluida.");
        pjDao.close();
    }
    catch (SQLException e) {
        LOGGER.log(Level.SEVERE, e.toString(), e);
    }
}

```

```
    public static void main(String[] args) {  
        new CadastroBDTeste().run();  
    }  
}
```

```
package cadastrdbd.model;  
  
/**  
 *  
 * @author Cleyton  
 */  
public class Pessoa {  
  
    private Integer id;  
    private String nome;  
    private String endereco;  
    private String cidade;  
    private String estado;  
    private String telefone;  
    private String email;  
  
    public Pessoa() {  
  
    }  
  
    public Pessoa(Integer id, String nome, String endereco, String cidade,  
        String estado, String telefone, String email) {  
        this.id = id;  
        this.nome = nome;  
        this.endereco = endereco;  
        this.cidade = cidade;  
        this.estado = estado;  
        this.telefone = telefone;  
        this.email = email;  
    }  
  
    public void exibir() {  
        System.out.println(this);  
    }  
  
    public Integer getId() {  
        return id;  
    }  
  
    public void setId(Integer id) {  
        this.id = id;  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public String getEndereco() {  
        return endereco;  
    }  
  
    public void setEndereco(String endereco) {  
        this.endereco = endereco;  
    }  
  
    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;
    }

    @Override
    public String toString() {
        String output = "id: ".concat(id.toString());
        output = output.concat("\nnome: ".concat(nome));
        output = output.concat("\nendereco: ".concat(endereco));
        output = output.concat("\ncidade: ".concat(cidade));
        output = output.concat("\nestado: ".concat(estado));
        output = output.concat("\ntelefone: ".concat(telefone));
        output = output.concat("\nemail: ".concat(email));
        return output;
    }
}

```

```

package cadastrabd.model;

/**
 *
 * @author Cleyton
 */
public class PessoaFisica extends Pessoa {

    private String cpf;

    public PessoaFisica() {

    }

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

    @Override
    public void exibir() {
        System.out.println(this);
    }

    public String getCpf() {
        return cpf;
    }

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

    @Override

```

```

    public String toString() {
        String output = super.toString();
        output = output.concat("\nCPF: ".concat(cpf));
        return output;
    }
}

```

```

package cadastrbd.model;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import cadastrbd.model.util.ConectorBD;

/**
 *
 * @author Cleyton
 */
public class PessoaFisicaDAO {

    private final ConectorBD connector;

    public PessoaFisicaDAO() {
        connector = new ConectorBD();
    }

    public PessoaFisica getPessoa(Integer id) throws SQLException {
        String sql = "SELECT pf.FK_Pessoa_idPessoa, pf.cpf, p.nome, p.endereco, p.cidade, "
            + "p.estado, p.telefone, p.email "
            + "FROM PessoaFisica pf "
            + "INNER JOIN Pessoa p ON pf.FK_Pessoa_idPessoa = p.idPessoa "
            + "WHERE pf.FK_Pessoa_idPessoa = ?";
        try (Connection con = connector.getConnection(); PreparedStatement stmt =
            con.prepareStatement(sql)) {
            stmt.setInt(1, id);
            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    return new PessoaFisica(
                        rs.getInt("FK_Pessoa_idPessoa"),
                        rs.getString("nome"),
                        rs.getString("endereco"),
                        rs.getString("cidade"),
                        rs.getString("estado"),
                        rs.getString("telefone"),
                        rs.getString("email"),
                        rs.getString("cpf")
                    );
                }
            }
        }
        return null;
    }

    public ArrayList<PessoaFisica> getPessoas() throws SQLException {
        ArrayList<PessoaFisica> list = new ArrayList<>();
        String sql = "SELECT pf.FK_Pessoa_idPessoa, pf.cpf, p.nome, p.endereco, p.cidade, "
            + "p.estado, p.telefone, p.email "
            + "FROM PessoaFisica pf "
            + "INNER JOIN Pessoa p ON pf.FK_Pessoa_idPessoa = p.idPessoa";
        try (Connection con = connector.getConnection(); PreparedStatement stmt =
            con.prepareStatement(sql); ResultSet rs = stmt.executeQuery()) {
            while (rs.next()) {
                list.add(new PessoaFisica(
                    rs.getInt("FK_Pessoa_idPessoa"),
                    rs.getString("nome"),
                    rs.getString("endereco"),
                    rs.getString("cidade"),
                    rs.getString("estado"),

```

```

        rs.getString("telefone"),
        rs.getString("email"),
        rs.getString("cpf"));
    }
    return list;
}

public int incluir(PessoaFisica pf) throws SQLException {
    if (pf.getNome() == null || pf.getNome().trim().isEmpty()) {
        throw new IllegalArgumentException("'nome' cannot be empty or null.");
    }
    String sqlInsertPessoa = "INSERT INTO Pessoa(nome, endereco, cidade, estado, telefone, email) VALUES(?, ?, ?, ?, ?, ?)";
    String sqlInsertPessoaFisica = "INSERT INTO PessoaFisica(FK_Pessoa_idPessoa, cpf) VALUES(?, ?)";
    try (Connection con = connector.getConnection(); PreparedStatement stmtPessoa = con.prepareStatement(sqlInsertPessoa, Statement.RETURN_GENERATED_KEYS)) {
        String[] pfArray = {"", pf.getNome(), pf.getEndereco(), pf.getCidade(), pf.getEstado(), pf.getTelefone(), pf.getEmail()};
        for(int i = 1; i < 7; i++) {
            stmtPessoa.setString(i, pfArray[i]);
        }
        if (stmtPessoa.executeUpdate() != 0) {
            System.out.println("INSERT INTO PessoaFisica success.");
        } else {
            throw new SQLException("Creating user failed, no rows affected.");
        }
    }
    try (ResultSet generatedKeys = stmtPessoa.getGeneratedKeys()) {
        if (generatedKeys.next()) {
            int idNovaPessoa = generatedKeys.getInt(1);
            pf.setId(idNovaPessoa);
            try (PreparedStatement stmtPessoaFisica = con.prepareStatement(sqlInsertPessoaFisica, Statement.RETURN_GENERATED_KEYS)) {
                stmtPessoaFisica.setInt(1, idNovaPessoa);
                stmtPessoaFisica.setString(2, pf.getCpf());
                stmtPessoaFisica.executeUpdate();
            }
            return idNovaPessoa;
        } else {
            throw new SQLException("Creating user failed. No ID obtained.");
        }
    }
}

public void alterar(PessoaFisica pf) throws SQLException {
    String sqlUpdatePessoa = "UPDATE Pessoa SET nome = ?, endereco = ?, cidade = ?, estado = ?, telefone = ?, email = ? WHERE idPessoa = ?";
    String sqlUpdatePessoaFisica = "UPDATE PessoaFisica SET cpf = ? WHERE FK_Pessoa_idPessoa = ?";
    try (Connection con = connector.getConnection();
        PreparedStatement stmtPessoa = con.prepareStatement(sqlUpdatePessoa);
        PreparedStatement stmtPessoaFisica = con.prepareStatement(sqlUpdatePessoaFisica)) {
        String[] pfArray = {"", pf.getNome(), pf.getEndereco(), pf.getCidade(), pf.getEstado(), pf.getTelefone(), pf.getEmail()};
        for(int i = 1; i < 7; i++) {
            stmtPessoa.setString(i, pfArray[i]);
        }
        stmtPessoa.setInt(7, pf.getId());
        stmtPessoa.executeUpdate();
        stmtPessoaFisica.setString(1, pf.getCpf());
        stmtPessoaFisica.setInt(2, pf.getId());
        stmtPessoaFisica.executeUpdate();
    }
}

public void excluir(PessoaFisica pf) throws SQLException {
    String sqlDeletePessoaFisica = "DELETE FROM PessoaFisica WHERE FK_Pessoa_idPessoa = ?";
    String sqlDeletePessoa = "DELETE FROM Pessoa WHERE idPessoa = ?";
    try (Connection con = connector.getConnection(); PreparedStatement stmtPessoaFisica = con.prepareStatement(sqlDeletePessoaFisica);
        PreparedStatement stmtPessoa = con.prepareStatement(sqlDeletePessoa)) {
        stmtPessoaFisica.setInt(1, pf.getId());
    }
}

```

```

        stmtPessoaFisica.executeUpdate();
        stmtPessoa.setInt(1, pf.getId());
        stmtPessoa.executeUpdate();
    }
}

public void close() throws SQLException {
    connector.close();
}
}

```

```

package cadastrobd.model;

/**
 *
 * @author Cleyton
 */
public class PessoaJuridica extends Pessoa {

    private String cnpj;

    public PessoaJuridica() {

    }

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

    @Override
    public void exibir() {
        System.out.println(this);
    }

    public String getCnpj() {
        return cnpj;
    }

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

    @Override
    public String toString() {
        String output = super.toString();
        output = output.concat("\nCNPJ: ".concat(cnpj));
        return output;
    }
}

```

```

package cadastrobd.model;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import cadastrobd.model.util.ConectorBD;

/**
 *
 * @author Cleyton
 */
public class PessoaJuridicaDAO {

```

```

private ConectorBD connector;

public PessoaJuridicaDAO() {
    connector = new ConectorBD();
}

public PessoaJuridica getPessoa(Integer id) throws SQLException {
    String sql = "SELECT pj.FK_Pessoa_idPessoa, pj.cnpj, p.nome, p.endereco, p.cidade,
p.estado, p.telefone, p.email "
        + "FROM PessoaJuridica pj "
        + "INNER JOIN Pessoa p ON pj.FK_Pessoa_idPessoa = p.idPessoa "
        + "WHERE pj.FK_Pessoa_idPessoa = ?";
    try (Connection con = connector.getConnection();
        PreparedStatement stmt = con.prepareStatement(sql)) {
        stmt.setInt(1, id);
        try (ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
                return new PessoaJuridica(
                    rs.getInt("FK_Pessoa_idPessoa"),
                    rs.getString("nome"),
                    rs.getString("endereco"),
                    rs.getString("cidade"),
                    rs.getString("estado"),
                    rs.getString("telefone"),
                    rs.getString("email"),
                    rs.getString("cnpj")
                );
            }
        }
    }
    return null;
}

public ArrayList<PessoaJuridica> getPessoas() throws SQLException {
    ArrayList<PessoaJuridica> list = new ArrayList<>();
    String sql = "SELECT pj.FK_Pessoa_idPessoa, pj.cnpj, p.nome, p.endereco, p.cidade,
p.estado, p.telefone, p.email "
        + "FROM PessoaJuridica pj "
        + "INNER JOIN Pessoa p ON pj.FK_Pessoa_idPessoa = p.idPessoa";
    try (Connection con = connector.getConnection();
        PreparedStatement stmt = con.prepareStatement(sql);
        ResultSet rs = stmt.executeQuery()) {
        while (rs.next()) {
            list.add(new PessoaJuridica(
                rs.getInt("FK_Pessoa_idPessoa"),
                rs.getString("nome"),
                rs.getString("endereco"),
                rs.getString("cidade"),
                rs.getString("estado"),
                rs.getString("telefone"),
                rs.getString("email"),
                rs.getString("cnpj")
            ));
        }
    }
    return list;
}

public int incluir(PessoaJuridica pj) throws SQLException {
    if (pj.getNome() == null || pj.getNome().trim().isEmpty()) {
        throw new IllegalArgumentException("'nome' cannot be empty or null.");
    }
    String sqlInsertPessoa = "INSERT INTO Pessoa(nome, endereco, cidade, estado, telefone,
email) VALUES(?, ?, ?, ?, ?, ?)";
    String sqlInsertPessoaJuridica = "INSERT INTO PessoaJuridica(FK_Pessoa_idPessoa, cnpj)
VALUES(?, ?)";

    try (Connection con = connector.getConnection();
        PreparedStatement stmtPessoa =
con.prepareStatement(sqlInsertPessoa, Statement.RETURN_GENERATED_KEYS)) {
        String[] pfArray = {"", pj.getNome(), pj.getEndereco(), pj.getCidade(),
pj.getEstado(), pj.getTelefone(), pj.getEmail()};
        for(int i = 1; i < 7; i++) {
            stmtPessoa.setString(i, pfArray[i]);
        }
    }
}

```

```

        if (stmtPessoa.executeUpdate() != 0) {
            System.out.println("INSERT INTO PessoaJuridica success.");
        }
        else {
            throw new SQLException("Creating user failed, no rows affected.");
        }
        try (ResultSet generatedKeys = stmtPessoa.getGeneratedKeys()) {
            if (generatedKeys.next()) {
                int idNovaPessoa = generatedKeys.getInt(1);
                pj.setId(idNovaPessoa);
                try (PreparedStatement stmtPessoaFisica =
con.prepareStatement(sqlInsertPessoaJuridica,Statement.RETURN_GENERATED_KEYS)) {
                    stmtPessoaFisica.setInt(1, idNovaPessoa);
                    stmtPessoaFisica.setString(2, pj.getCnpj());
                    stmtPessoaFisica.executeUpdate();
                }
                return idNovaPessoa;
            } else {
                throw new SQLException("Creating user failed. No ID obtained.");
            }
        }
    }

    public void alterar(PessoaJuridica pj) throws SQLException {
        String sqlUpdatePessoa = "UPDATE Pessoa SET nome = ?, endereco = ?, cidade = ?, estado =
?, telefone = ?, email = ? WHERE idPessoa = ?;";
        String sqlUpdatePessoaJuridica = "UPDATE PessoaJuridica SET cnpj = ? WHERE
FK_Pessoa_idPessoa = ?;";
        try (Connection con = connector.getConnection();
            PreparedStatement stmtPessoa =
con.prepareStatement(sqlUpdatePessoa,Statement.RETURN_GENERATED_KEYS);
            PreparedStatement stmtPessoaJuridica =
con.prepareStatement(sqlUpdatePessoaJuridica,Statement.RETURN_GENERATED_KEYS)) {
            String[] pfArray = {"", pj.getNome(), pj.getEndereco(), pj.getCidade(),
pj.getEstado(), pj.getTelefone(), pj.getEmail()};
            for(int i = 1; i < 7; i++) {
                stmtPessoa.setString(i, pfArray[i]);
            }
            stmtPessoa.setInt(7, pj.getId());
            stmtPessoa.executeUpdate();
            stmtPessoaJuridica.setString(1, pj.getCnpj());
            stmtPessoaJuridica.setInt(2, pj.getId());
            stmtPessoaJuridica.executeUpdate();
        }
    }

    public void excluir(PessoaJuridica pj) throws SQLException {
        String sqlDeletePessoaJuridica = "DELETE FROM PessoaJuridica WHERE FK_Pessoa_idPessoa =
?;";
        String sqlDeletePessoa = "DELETE FROM Pessoa WHERE idPessoa = ?;";
        try (Connection con = connector.getConnection();
            PreparedStatement stmtPessoaJuridica =
con.prepareStatement(sqlDeletePessoaJuridica,Statement.RETURN_GENERATED_KEYS);
            PreparedStatement stmtPessoa =
con.prepareStatement(sqlDeletePessoa,Statement.RETURN_GENERATED_KEYS)) {
            stmtPessoaJuridica.setInt(1, pj.getId());
            stmtPessoaJuridica.executeUpdate();
            stmtPessoa.setInt(1, pj.getId());
            stmtPessoa.executeUpdate();
        }
    }

    public void close() throws SQLException {
        connector.close();
    }
}

```

```

package cadastrbd.model.util;

import java.sql.Connection;

```

```

import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.DriverManager;
import java.sql.Statement;
import java.sql.SQLException;

/**
 *
 * @author Cleyton
 */
public class ConectorBD {

    private Connection con;
    private PreparedStatement stmt;
    private ResultSet rs;
    private CredentialsLoader loader;

    private final String HOSTNAME;
    private final String DBNAME;
    private final String LOGIN;
    private final String PASSWORD;

    public ConectorBD() {
        loader = new CredentialsLoader();
        HOSTNAME = loader.getHostname();
        DBNAME = loader.getDbname();
        LOGIN = loader.getLogin();
        PASSWORD = loader.getPassword();
    }

    public Connection getConnection() throws SQLException {
        String URL = String.format("jdbc:sqlserver://%s:1433;databaseName=%s;",
            HOSTNAME, DBNAME).concat("encrypt=true;trustServerCertificate=true");
        con = DriverManager.getConnection(URL, LOGIN, PASSWORD);
        return con;
    }

    public PreparedStatement getPrepared(String sql) throws SQLException {
        stmt = getConnection().prepareStatement(sql);
        return stmt;
    }

    public ResultSet getSelect(String sql) throws SQLException {
        stmt = getPrepared(sql);
        rs = stmt.executeQuery();
        return rs;
    }

    public int insert(String sql) throws SQLException {
        stmt = getConnection().prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
        stmt.executeUpdate();
        rs = stmt.getGeneratedKeys();
        if (rs.next()) {
            return rs.getInt(1);
        } else {
            throw new SQLException("Data insert failed.");
        }
    }

    public boolean update(String sql) throws SQLException {
        return getPrepared(sql).executeUpdate() > 0;
    }

    public void close() throws SQLException {
        if (stmt != null && !stmt.isClosed()) {
            stmt.close();
        }
        if (rs != null && !rs.isClosed()) {
            rs.close();
        }
        if (con != null && !con.isClosed()) {
            con.close();
        }
    }
}

```

```
}  
  
}
```

```
package cadastrobd.model.util;  
  
import javax.xml.parsers.DocumentBuilderFactory;  
  
import org.w3c.dom.Document;  
import org.w3c.dom.Element;  
import org.w3c.dom.Node;  
import org.w3c.dom.NodeList;  
import org.xml.sax.SAXException;  
  
import javax.xml.parsers.DocumentBuilder;  
import javax.xml.XMLConstants;  
import javax.xml.parsers.ParserConfigurationException;  
  
import java.io.File;  
import java.io.IOException;  
  
import java.util.logging.Logger;  
import java.util.logging.Level;  
import java.util.Iterator;  
import java.util.NoSuchElementException;  
  
/**  
 *  
 * @author Cleyton  
 * @source https://mkyong.com/java/how-to-read-xml-file-in-java-dom-parser/  
 */  
public class CredentialsLoader {  
  
    private static final Logger LOGGER = Logger.getLogger(CredentialsLoader.class.getName());  
    private final String FILENAME = "resources/credentials.xml";  
  
    private String hostname;  
    private String dbname;  
    private String login;  
    private String password;  
  
    public CredentialsLoader() {  
        run();  
    }  
  
    private void run() {  
        // Instantiate the Factory  
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();  
  
        try {  
            // optional, but recommended: process XML securely, avoid attacks like XML External  
            Entities (XXE)  
            dbf.setFeature(XMLConstants.FEATURE_SECURE_PROCESSING, true);  
  
            // parse XML file  
            DocumentBuilder db = dbf.newDocumentBuilder();  
            Document doc = db.parse(new File(FILENAME));  
  
            // optional, but recommended, according to:  
            // http://stackoverflow.com/questions/13786607/normalization-in-dom-parsing-with-java-how-does-it-work  
            doc.getDocumentElement().normalize();  
  
            NodeList list = doc.getElementsByTagName("user");  
  
            for (Node node : iterable(list)) {  
                if (node.getNodeType() == Node.ELEMENT_NODE) {  
                    Element element = (Element) node;  
                    hostname = element.getElementsByTagName("hostname").item(0).getTextContent();  
                    dbname = element.getElementsByTagName("dbname").item(0).getTextContent();  
                    login = element.getElementsByTagName("login").item(0).getTextContent();  
                }  
            }  
        } catch (ParserConfigurationException | SAXException | IOException e) {  
            LOGGER.log(Level.SEVERE, "Failed to load credentials", e);  
        }  
    }  
}
```



```

        password = element.getElementsByTagName("password").item(0).getTextContent();
    }
}
} catch (ParserConfigurationException | SAXException | IOException e) {
    LOGGER.log(Level.SEVERE, e.toString(), e);
}
}

private Iterable<Node> iterable(final NodeList nodeList) {
    return () -> new Iterator<Node>() {
        private int index = 0;
        @Override
        public boolean hasNext() {
            return index < nodeList.getLength();
        }
        @Override
        public Node next() {
            if (!hasNext()) {
                throw new NoSuchElementException();
            }
            return nodeList.item(index++);
        }
    };
}

public String getHostname() {
    return hostname;
}

public void setHostname(String hostname) {
    this.hostname = hostname;
}

public String getDbname() {
    return dbname;
}

public void setDbname(String dbname) {
    this.dbname = dbname;
}

public String getLogin() {
    return login;
}

public void setLogin(String login) {
    this.login = login;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}
}
}

```

```

package cadastrorbd.model.util;

import java.sql.ResultSet;
import java.sql.SQLException;

/**
 *
 * @author Cleyton
 */
public class SequenceManager {

    public int getValue(String sequence) throws SQLException {
        ResultSet rs = new ConectorBD().getSelect("SELECT NEXT VALUE FOR ".concat(sequence));
    }
}

```

```

        if (rs.next()) {
            return rs.getInt(1);
        } else {
            throw new SQLException("Next value not achievable: ".concat(sequence));
        }
    }
}

```

```

package cadastrobd.test;

/**
 *
 * @author Cleyton
 */
import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.logging.Logger;
import java.util.logging.Level;

import cadastrobd.model.util.CredentialsLoader;

/**
 * This program demonstrates how to establish database connection to Microsoft SQL Server.
 * @author www.codejava.net
 * @source https://www.codejava.net/java-se/jdbc/connect-to-microsoft-sql-server-via-jdbc
 */
public class JdbcSQLServerConnection {

    private static final Logger LOGGER =
        Logger.getLogger(JdbcSQLServerConnection.class.getName());

    private final String HOSTNAME;
    private final String DBNAME;
    private final String LOGIN;
    private final String PASSWORD;

    public JdbcSQLServerConnection() {
        CredentialsLoader loader = new CredentialsLoader();
        HOSTNAME = loader.getHostname();
        DBNAME = loader.getDbname();
        LOGIN = loader.getLogin();
        PASSWORD = loader.getPassword();
    }

    private void run() {
        Connection conn = null;
        try {
            String dbURL = "jdbc:sqlserver://" + HOSTNAME + ":1433;databaseName="
                + DBNAME + ";encrypt=true;trustServerCertificate=true;";
            conn = DriverManager.getConnection(dbURL, LOGIN, PASSWORD);
            if (conn != null) {
                DatabaseMetaData dm = (DatabaseMetaData) conn.getMetaData();
                System.out.println("Driver name: " + dm.getDriverName());
                System.out.println("Driver version: " + dm.getDriverVersion());
                System.out.println("Product name: " + dm.getDatabaseProductName());
                System.out.println("Product version: " + dm.getDatabaseProductVersion());
            }
        } catch (SQLException e) {
            LOGGER.log(Level.SEVERE, e.toString(), e);
        } finally {
            try {
                if (conn != null && !conn.isClosed()) {
                    conn.close();
                }
            } catch (SQLException e) {
                LOGGER.log(Level.SEVERE, e.toString(), e);
            }
        }
    }
}

```

```
    }  
}  
  
public static void main(String[] args) {  
    new JdbcSQLServerConnection().run();  
}  
  
}
```