Campus de Mogi Guaçu – SP

Aluno: João Lucas Menicuci

Matrícula: 2023 0421 4425

Curso: Desenvolvimento Full-Stack

Turma: 2023.1

Disciplina: RPG0016 – BackEnd sem banco não tem

Missão Prática Nível 3 – Mundo 3

Mogi Guaçu

2024

I. Primeiro Procedimento

1. Título da Prática: “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 de Desenvolvimento: Na seção “III. Anexo” deste relatório. Link para o repositório do GitHub: https://github.com/joaomenicuci/mp-mundo3-nivel3.

4. Resultados:

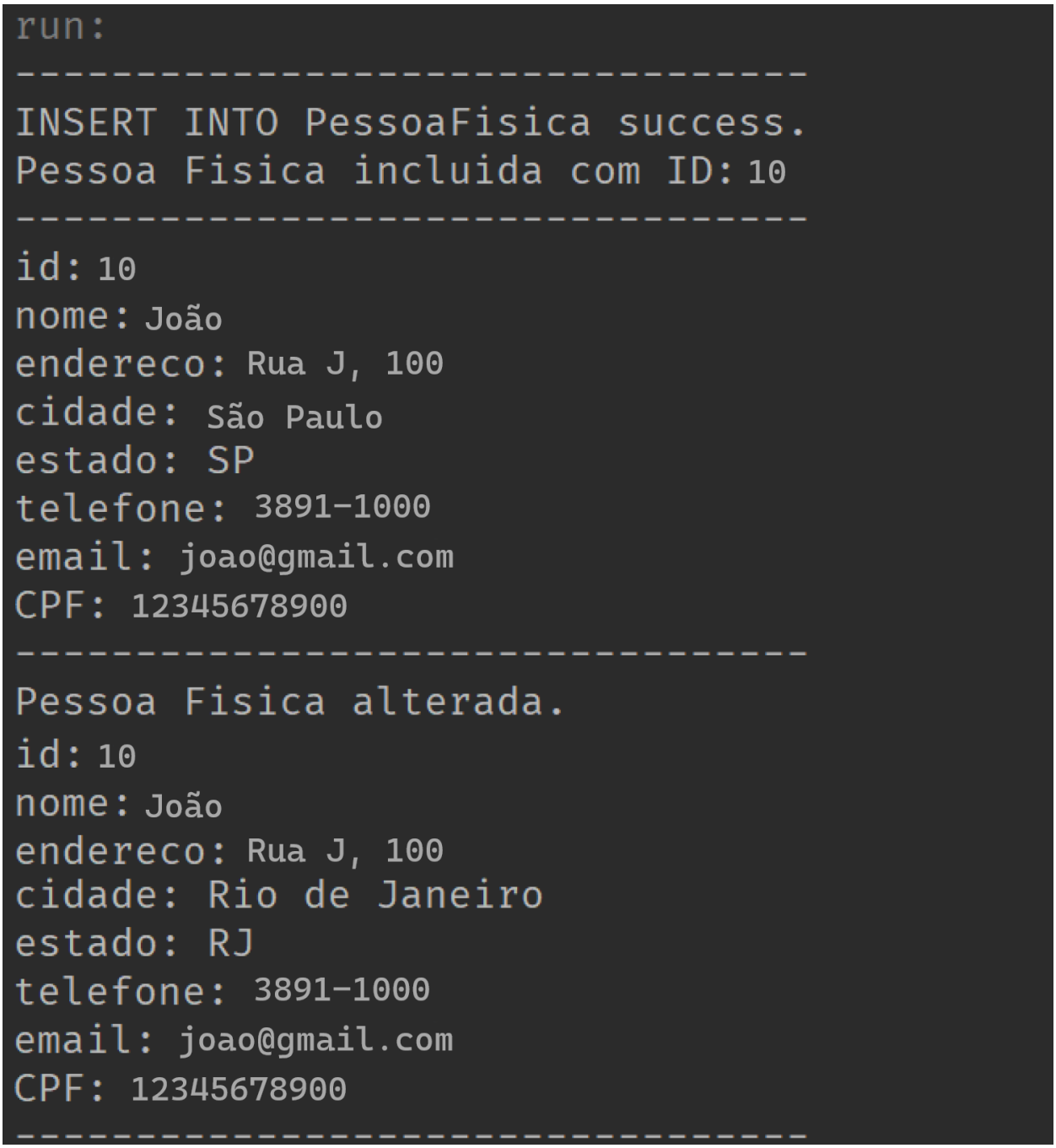

Figura 1: Execução do CadastroBD. Inserção e alteração de Pessoa Física.

Figura 2: Execução do CadastroBD. Inserção e Exclusão de Pessoa Jurídica.

5. Análise e Conclusão:

    a. Qual a importância dos componentes de middleware, como JDBC?

    O JDBC é uma API que fornece acesso a fonte de dados em Java, e com isso é possível interligar as aplicações Java a banco de dados relacionais SQL, e até mesmo com planilhas de Excel por exemplo. O JDBC é de extrema importância pois é ele que faz o intermédio dos dados de maneira padronizada, entre aplicação Java e o banco de dados.

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

    Ambos as duas classes são utilizadas para executar comandos SQL em Java, porém a classe *PreparedStatement* é mais recomendada para prevenir ataques de injeção SQ, pois os valores dos parâmetros são considerados dados e não como consulta SQL em si, e com isso reduz o risco de injeção de códigos maliciosos.

    A classe *PreparedStatement* também é considerada mais eficiente que a *Statement* quando é realizada uma mesma consulta com diferentes valores. Também ocorre o fato de alguns bancos de dados armazenarem cache, e com isso a utilização de *PreparedStatement* é ainda mais eficiente.

    c. Como o padrão DAO melhora a manutenibilidade do software?

    O padrão DAO melhora a manutenibilidade, pois separa ou isola o código responsável pelo acesso e manipulação de dados do restante do resto do código, e isso facilita a manutenção do código, pois as alterações em partes do código não interferem no restante do sistema. A utilização do padrão DAO também beneficia a legibilidade do código.

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

    Quando utilizamos um modelo relacional em banco de dados, a herança é refletida com o uso da técnica chamada de tabela de junção, que é quando uma consulta é feita para recuperar dados de uma herança. Para isto é necessário fazer uma junção entre uma tabela base e a tabela de subclasse usando chaves primárias e estrangeiras correspondentes.

II. Segundo Procedimento

1. Título da Prática: “Alimentando a Base”.

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 de Desenvolvimento: Na seção “III. Anexo” deste relatório. Link para o repositório do GitHub: https://github.com/joaomenicuci/mp-mundo3-nivel3.

4. Resultados:

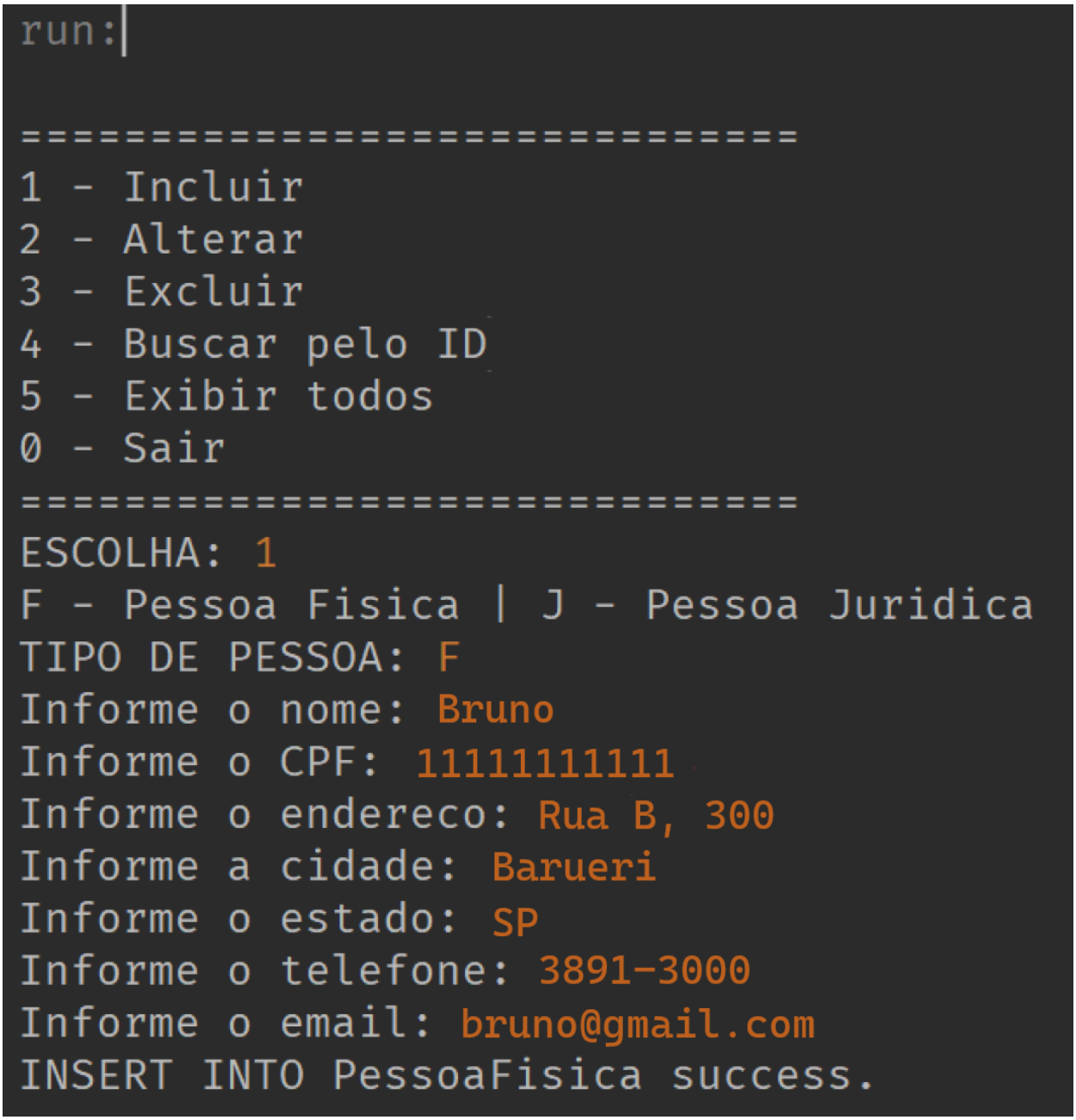

Figura 3: Execução do CadastroBD. Opção de Incluir.

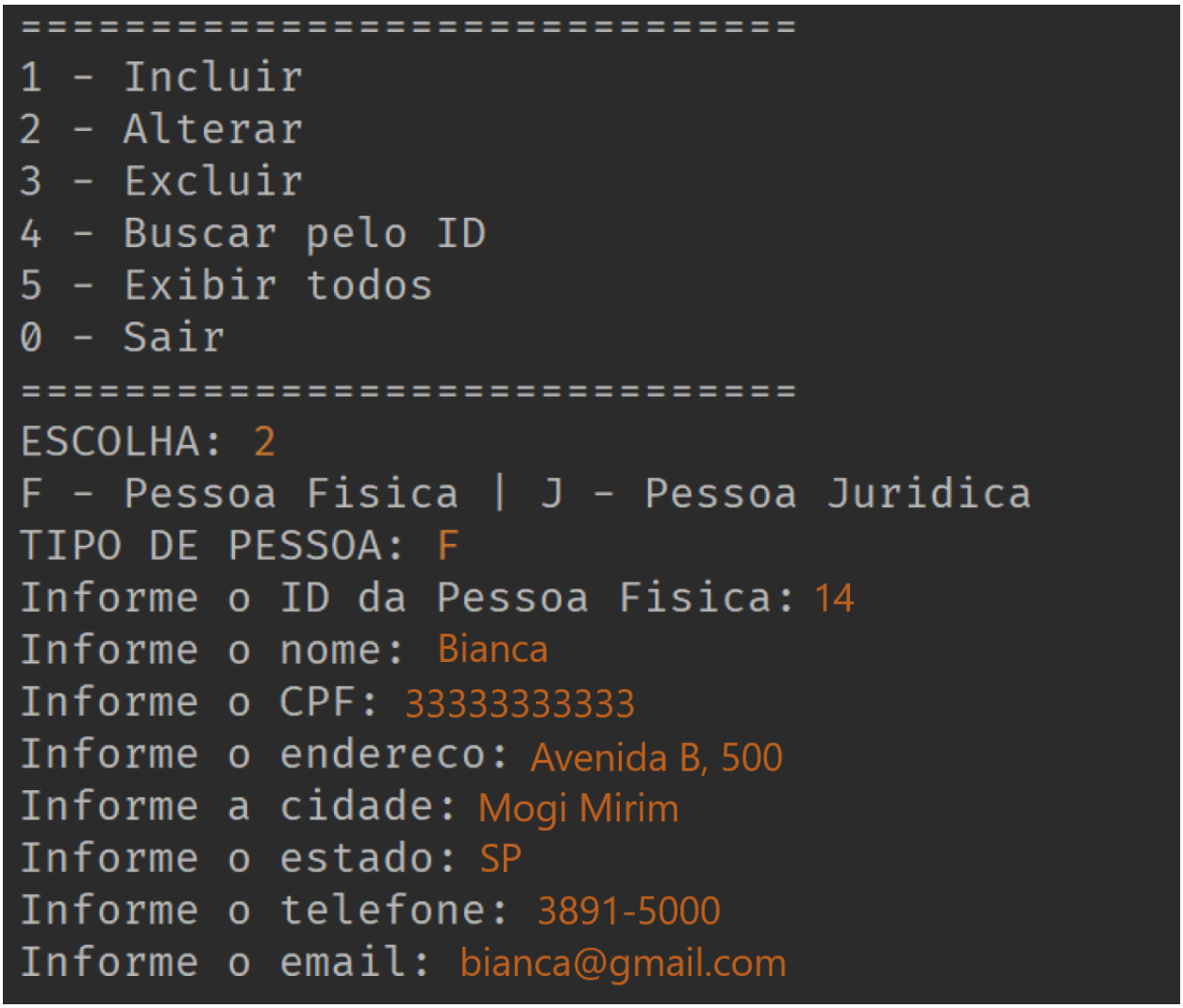

Figura 4: Execução do CadastroBD. Opção de Alterar.

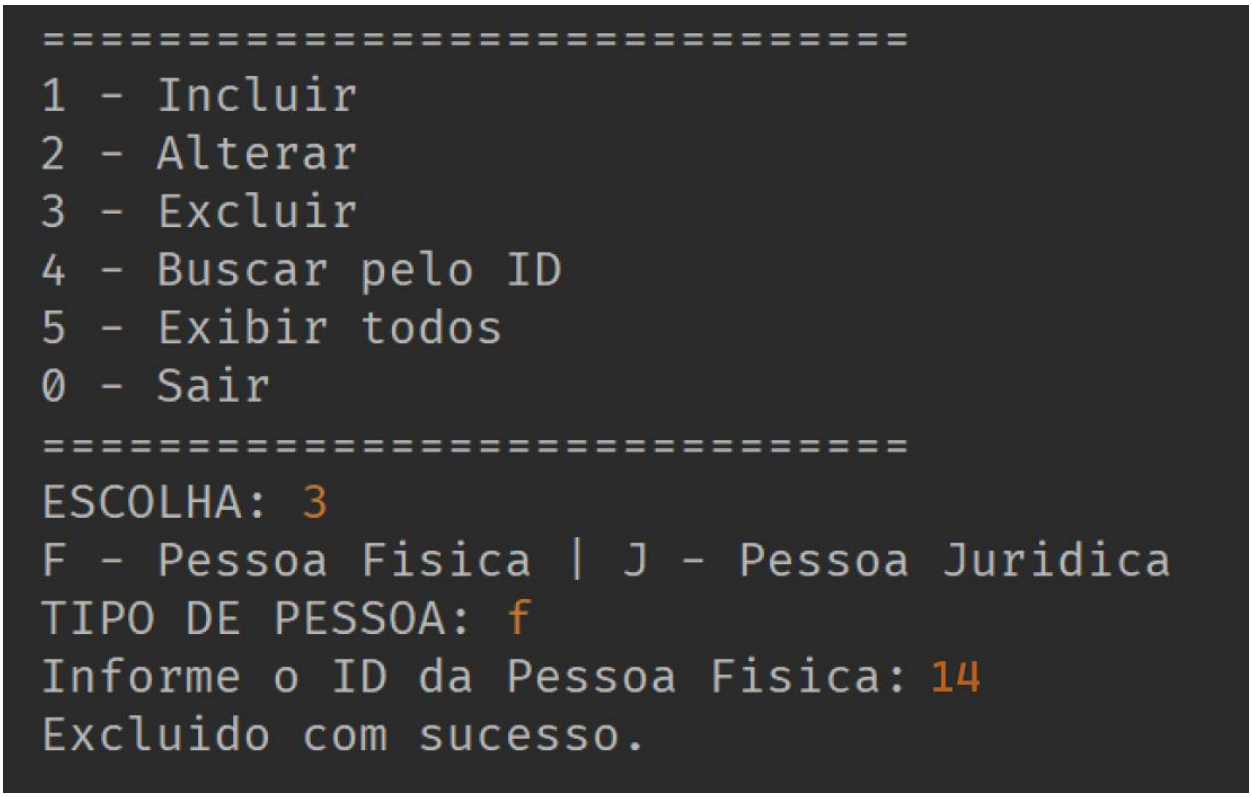

Figura 5: Execução do CadastroBD. Opção de Excluir.

5. Análise e Conclusão:

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

   A persistência em arquivos é quando no armazenamento dos dados ocorre no formato de arquivos em formato binário, porém estes arquivos não é a forma mais recomendada para consultas complexas e manipulação dos dados, mesmo sendo uma forma simples de armazenar dados.

   Já a persistência em banco de dados utiliza sistemas de gerenciamento de banco de dados para armazenar, recupera e gerenciar dados de maneira estruturada, e isto facilita muito a manipulação e consulta dos dados.

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

   O operador lambda simplificou a impressão de valores contidos em entidades, pois permite que os programadores escrevam o código mais conciso e legível. Com o operador lambda foi possível substituir várias linhas de código pois não é mais necessário utilizar loops no código.

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

Em Java, o método *main* é o ponto de entrada para um programa. Ele é marcado como *static* porque é chamado pela JVM sem a necessidade de criar um objeto da classe que o contém. O *static* indica que o método pertence à classe em si, permitindo que seja invocado diretamente pela JVM para iniciar a execução do programa.

III. Anexo

- CadastroBD.java

```
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 Joao_
 */
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();
    }
}
```

- Pessoa.java

```
package cadastrobd.model;

/**
 *
 * @author Joao_
 */
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;
    }
}
```

- PessoaFísica.java

```
package cadastrobd.model;

/**
 *
 * @author Joao_
 */
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;
    }
}
```

- PessoaFísicaDAO.java

```
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 Joao_
 */
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();
  }

}
```

- PessoaJurídica.java

```
package cadastrobd.model;
```

```
/**
 *
 * @author Joao_
 */
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;
    }
}
```

- PessoaJurídicaDAO.java

```
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 Joao_
 */
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();
    }
}
```

- ConectorBD.java

```
package cadastrobd.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 Joao_
 */
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();
        }
    }
}
```

- CredentialsLoader.java

```
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 Joao_
 */
```

```
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() {
      DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
      try {
         dbf.setFeature(XMLConstants.FEATURE_SECURE_PROCESSING, true);
         DocumentBuilder db = dbf.newDocumentBuilder();
         Document doc = db.parse(new File(FILENAME));
         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();
                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;
    }
}
```

- SequenceManager.java

```
package cadastrobd.model.util;

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

```
/**
 *
 * @author Joao_
 */
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));
        }
    }
}
```

- JdbcSQLServerConnection.java

```
package cadastrobd.test;


/**
 *
 * @author Joao_
 */
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();
```

```
    }

}
```