Documentação do Sistema de Avaliações de Restaurantes

Abstração:

```java
package util;

public interface ClienteInterface {  1 usage  1 implementation
    int getIdcliente();  27 usages  1 implementation
    String getCpf();  2 usages  1 implementation
    String getNome();  10 usages  1 implementation
    String getEmail();  2 usages  1 implementation
    String getSenha();  2 usages  1 implementation
    void setIdcliente(int idcliente);  1 usage  1 implementation
    String toString();  1 implementation
}
```

Descrição: Interface para classe Cliente

Justificativa: Oculta detalhes internos

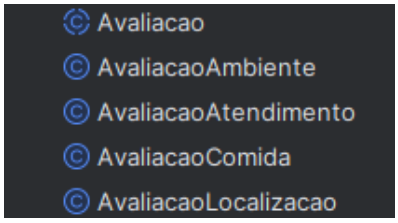
Encapsulamento:

```java
public int getIdcliente() { return idcliente; }

public String getCpf() { return cpf; }

public String getNome() { return nome; }

public String getEmail() { return email; }

public String getSenha() { return senha; }

public void setIdcliente(int idcliente) { this.idcliente = idcliente; }
```

Descrição: Controla os acessos dos dados

Justificativa: Acesso controlado dos dados acima

Herança:




```java
public abstract class Avaliacao
    private int idAvaliacao;  2
```

Descrição: A classe avaliação da origem a diversos tipos de avaliação

Justificativa: Reutilização de código ao invés de reescrever tudo

Polimorfismo:

```java
package util;

public abstract class Avaliacao {  4 usages  4 inheritors
    private int idAvaliacao;  2 usages

    public Avaliacao() {  8 usages
    }

    public int getIdAvaliacao() { return idAvaliacao; }

    public void setIdAvaliacao(int idAvaliacao) { this.idAvaliacao = idAvaliacao; }

    protected abstract void comentar();  no usages  4 implementations
}
```

Descrição:

Justificativa:

Classe abstrata:

```java
package util;

public abstract class Avaliacao {  4 usages  4 inheritors
    private int idAvaliacao;  2 usages

    public Avaliacao() {  8 usages
    }

    public int getIdAvaliacao() { return idAvaliacao; }

    public void setIdAvaliacao(int idAvaliacao) { this.idAvaliacao = idAvaliacao; }

    protected abstract void comentar();  no usages  4 implementations
}
```

Descrição: Classe que deu origem a diversas formas de avaliação

Justificativa: A classe avaliação por si só não é eficiente para diversas avaliações

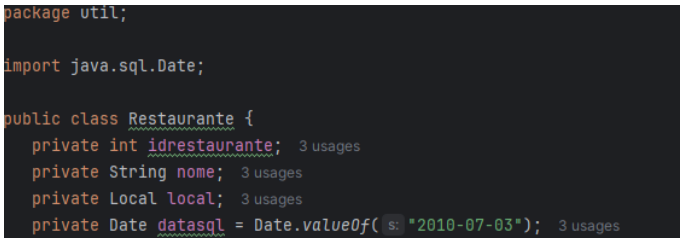
Interface:

```java
package util;

public interface ClienteInterface {  1 usage  1 implementation
    int getIdcliente();  27 usages  1 implementation
    String getCpf();  2 usages  1 implementation
    String getNome();  10 usages  1 implementation
    String getEmail();  2 usages  1 implementation
    String getSenha();  2 usages  1 implementation
    void setIdcliente(int idcliente);  1 usage  1 implementation
    String toString();  1 implementation
}
```

Descrição- A interface descreve o que um cliente pode fazer, mas não como ele faz

Justificativa- Criar um código flexível, separado e fácil de manter

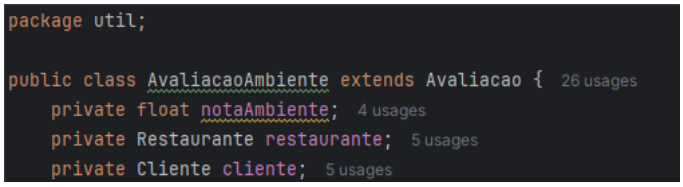
Cardinalidade 1:1 -



Descrição: Instância restaurante só pode ter um local

Justificativa: Um restaurante não pode ter mais de um local

Cardinalidade 1:N -



Descrição: Instancia da avaliação do ambiente

Justificativa: Um restaurante pode ter diversas avaliações

Cardinalidade N:N -

Descrição: Lista de classificações

Justificativa: Varios clientes podem ter varias classificações

Unidirecional:

```java
package util;

public class AvaliacaoComida extends Avaliacao {  15 usages
    private float notaComida;  4 usages
    private Restaurante restaurante;  5 usages
    private Cliente cliente;  5 usages

    public AvaliacaoComida(float notaComida) {  1 usage
        super();
        this.notaComida = notaComida;
    }
```

Descrição: Funciona como uma especialização de uma classe mais genérica, Avaliacao através do uso de extends

Justificativa: Criar um sistema mais simples, separado e performática

Agregação:

```java
package util;

public class AvaliacaoComida extends Avaliacao {  15 usages
    private float notaComida;  4 usages
    private Restaurante restaurante;  5 usages
    private Cliente cliente;  5 usages

    public AvaliacaoComida(float notaComida) {  1 usage
        super();
        this.notaComida = notaComida;
    }

    public AvaliacaoComida(int idAvaliacao, float notaComida, Restaurante restaurante, Cliente cliente) {  2 usages
        super();
        setIdAvaliacao(idAvaliacao);
        this.notaComida = notaComida;
        this.restaurante = restaurante;
        this.cliente = cliente;
    }

    public float getNotaComida() { return notaComida; }


    public Restaurante getRestaurante() { return restaurante; }

    public void setRestaurante(Restaurante restaurante) { this.restaurante = restaurante; }

    public Cliente getCliente() { return cliente; }

    public void setCliente(Cliente cliente) { this.cliente = cliente; }
```

Descrição: Esta classe perminte que avaliem o restaurante de diversas formas

Justificativa: Deixar o codigo mais organizado e de fácil manutenção

List e operação de busca:

```java
    public Object buscarPorId(int id) {
        String sql = "SELECT id_avaliacao_localizacao, nota_localizacao, fk_restaurante, fk_cliente FROM avaliacao_localizacao WHERE id_avaliacao_localizacao = ?";
        AvaliacaoLocalizacao avaliacao = null;
        try (Connection conn = ConnectionFactory.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setInt( parameterIndex: 1, id);
            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    int idRestaurante = rs.getInt( columnLabel: "fk_restaurante");
                    int idCliente = rs.getInt( columnLabel: "fk_cliente");

                    RestauranteDAO restauranteDAO = new RestauranteDAO();
                    ClienteDAO clienteDAO = new ClienteDAO();

                    Restaurante restaurante = (Restaurante) restauranteDAO.buscarPorId(idRestaurante);
                    Cliente cliente = (Cliente) clienteDAO.buscarPorId(idCliente);

                    avaliacao = new AvaliacaoLocalizacao(
                            rs.getInt( columnLabel: "id_avaliacao_localizacao"),
                            rs.getFloat( columnLabel: "nota_localizacao"),
                            restaurante,
                            cliente
                    );
                    System.out.println("Avaliação encontrada: ID " + avaliacao.getIdAvaliacao());
                }
            }
        } catch (SQLException e) {
            System.err.println("Não foi possivel buscar: " + e.getMessage());
            e.printStackTrace();
        }
        return avaliacao;
    }
```

Descrição- buscarPorId busca no banco de dados uma avaliação de localização específica, usando id com critério .

Justificativa- A necessidade de reconstruir um objeto complexo e completo a partir de dados do banco de dados.

Get, set e operação de remoção:

```
    public void excluir(int id) {
        String sql = "DELETE FROM avaliacao_localizacao WHERE id_avaliacao_localizacao = ?";
        try (Connection conn = ConnectionFactory.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setInt( parameterIndex: 1, id);
            int affectedRows = stmt.executeUpdate();
            if (affectedRows > 0) {
                System.out.println("Avaliação de Localização, ID " + id + " excluída.");
            } else {
                System.out.println("Avaliação de Localização, ID " + id + " não encontrada.");
            }
        } catch (SQLException e) {
            System.err.println("Não foi possível excluir: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Descrição: Operação do SQL que remove a avaliação especificada

Justificativa: Seguir os princípios do CRUD

Operações de adição:

```
public float calcularClassificacao() {  1 usage
    float somaNotas = 0.0f;
    int numeroDeAvaliacoesValidas = 0;

    if (avaliacaoComida != null) {
        somaNotas += avaliacaoComida.getNotaComida();
        numeroDeAvaliacoesValidas++;
    }
    if (avaliacaoAmbiente != null) {
        somaNotas += avaliacaoAmbiente.getNotaAmbiente();
        numeroDeAvaliacoesValidas++;
    }
    if (avaliacaoAtendimento != null) {
        somaNotas += avaliacaoAtendimento.getNotaAtendimento();
        numeroDeAvaliacoesValidas++;
    }
    if (avaliacaoLocalizacao != null) {
        somaNotas += avaliacaoLocalizacao.getNotaLocalizacao();
        numeroDeAvaliacoesValidas++;
    }
```

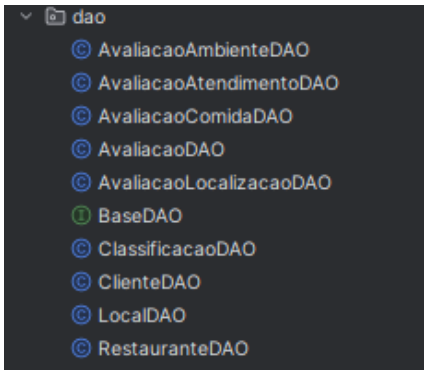
Descrição: Método que calcula a média das avaliações

Justificativa: Facilitar a visualização

Conexão com banco relacional via JDBC:

```java
package bd;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class ConnectionFactory {  55 usages

    public static Connection getConnection() throws SQLException {
        String sgbd = "mysql";
        String endereco = "localhost";
        String bd = "restaurante_avaliacao_db";
        String usuario = "root";
        String senha = "admin";

        try {
            Class.forName( className: "com.mysql.cj.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            throw new SQLException("Driver JDBC do MySQL não encontrado. Verifique suas dependências (ex: mysql-connector-java.jar).", e);
        }

        Connection connection = DriverManager.getConnection(
                url: "jdbc:" + sgbd + "://" + endereco + "/" + bd, usuario, senha);

        return connection;
    }

    public static void closeConnection(Connection conn) {  no usages
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                System.err.println("Erro ao fechar a conexão: " + e.getMessage());
            }
        }
    }
}
```

DAO:



CRUD:

```java
    public void salvar(Object obj) {
        if (obj instanceof Classificacao) {
            Classificacao classificacao = (Classificacao) obj;
            String sql = "INSERT INTO classificacao_final (fk_restaurante, fk_cliente, nota_final, data_classificacao) VALUES (?, ?, ?, ?)";

            try (Connection conn = ConnectionFactory.getConnection();
                 PreparedStatement stmt = conn.prepareStatement(sql, PreparedStatement.RETURN_GENERATED_KEYS)) {

                if (classificacao.getRestaurante() == null || classificacao.getRestaurante().getIdrestaurante() == 0) {
                    throw new SQLException("O restaurante associado à Classificacao é nulo ou não tem ID.");
                }
                if (classificacao.getCliente() == null || classificacao.getCliente().getIdcliente() == 0) {
                    throw new SQLException("O cliente associado à Classificacao é nulo ou não tem ID.");
                }

                stmt.setInt( parameterIndex: 1, classificacao.getRestaurante().getIdrestaurante());
                stmt.setInt( parameterIndex: 2, classificacao.getCliente().getIdcliente());
                stmt.setFloat( parameterIndex: 3, classificacao.getNotaFinal());
                stmt.setDate( parameterIndex: 4, classificacao.getDataClassificacao());

                int affectedRows = stmt.executeUpdate();

                if (affectedRows > 0) {
                    try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
                        if (generatedKeys.next()) {
                            classificacao.setIdClassificacao(generatedKeys.getInt( columnIndex: 1));
                            System.out.println("Classificação salva (ID: " + classificacao.getIdClassificacao() + ")");
                        } else {
                            System.err.println("Não foi possivel obter o ID.");
                        }
                    }
                } else {
                    System.err.println("Nenhuma linha modificada ao salvar a Classificação final.");
                }

            } catch (SQLException e) {
                System.err.println("Não foi possivel salvar: " + e.getMessage());
                e.printStackTrace();
            }
        } else {
            System.out.println("O objeto não é uma instância de Classificacao. Não salvo no banco de dados.");
        }
    }
```

```java
public void atualizar(Object obj) {
    if (obj instanceof Classificacao) {
        Classificacao classificacao = (Classificacao) obj;
        String sql = "UPDATE classificacao_final SET fk_restaurante = ?, fk_cliente = ?, nota_final = ?, data_classificacao = ? WHERE id_classificacao = ?";

        try (Connection conn = ConnectionFactory.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            if (classificacao.getRestaurante() == null || classificacao.getRestaurante().getIdrestaurante() == 0) {
                throw new SQLException("O restaurante associado à Classificacao é nulo ou não tem ID.");
            }
            if (classificacao.getCliente() == null || classificacao.getCliente().getIdcliente() == 0) {
                throw new SQLException("O cliente associado à Classificacao é nulo ou não tem ID.");
            }

            stmt.setInt( parameterIndex: 1, classificacao.getRestaurante().getIdrestaurante());
            stmt.setInt( parameterIndex: 2, classificacao.getCliente().getIdcliente());
            stmt.setFloat( parameterIndex: 3, classificacao.getNotaFinal());
            stmt.setDate( parameterIndex: 4, classificacao.getDataClassificacao());
            stmt.setInt( parameterIndex: 5, classificacao.getIdClassificacao());

            int affectedRows = stmt.executeUpdate();
            if (affectedRows > 0) {
                System.out.println("Classificação final atualizada (ID: " + classificacao.getIdClassificacao() + ")");
            } else {
                System.err.println("Nenhuma linha modificada ao atualizar a Classificação final.");
            }
        } catch (SQLException e) {
            System.err.println("Não foi possivel atualizar: " + e.getMessage());
            e.printStackTrace();
        }
    } else {
        System.out.println("o Objeto não é uma instância de Classificacao. Não atualizado no banco de dados.");
    }
}
```

```java
public Object buscarPorId(int id) {
    String sql = "SELECT id_classificacao, fk_restaurante, fk_cliente, nota_final, data_classificacao FROM classificacao_final WHERE id_classificacao = ?";
    Classificacao classificacao = null;
    try (Connection conn = ConnectionFactory.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt( parameterIndex: 1, id);
        try (ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
                int fkRestaurante = rs.getInt( columnLabel: "fk_restaurante");
                int fkCliente = rs.getInt( columnLabel: "fk_cliente");
                float notaFinal = rs.getFloat( columnLabel: "nota_final");
                Date dataClassificacao = rs.getDate( columnLabel: "data_classificacao");

                RestauranteDAO restauranteDAO = new RestauranteDAO();
                Restaurante restaurante = (Restaurante) restauranteDAO.buscarPorId(fkRestaurante);

                ClienteDAO clienteDAO = new ClienteDAO();
                Cliente cliente = (Cliente) clienteDAO.buscarPorId(fkCliente);

                classificacao = new Classificacao(
                        rs.getInt( columnLabel: "id_classificacao"),
                        restaurante,
                        cliente,
                        notaFinal,
                        dataClassificacao
                );
                System.out.println("Classificação encontrada: ID " + classificacao.getIdClassificacao() + ", Nota Final: " + classificacao.getNotaFinal());
            }
        }
    } catch (SQLException e) {
        System.err.println("Não foi possivel buscar: " + e.getMessage());
        e.printStackTrace();
    }
    return classificacao;
}
```

```java
public void excluir(int id) {
    String sql = "DELETE FROM classificacao_final WHERE id_classificacao = ?";
    try (Connection conn = ConnectionFactory.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {
        stmt.setInt( parameterIndex: 1, id);
        int affectedRows = stmt.executeUpdate();
        if (affectedRows > 0) {
            System.out.println("Classificação final excluida (ID: " + id + ")");
        } else {
            System.err.println("Nenhuma linha modificada ao excluir a Classificação final. Classificação não encontrada.");
        }
    } catch (SQLException e) {
        System.err.println("Não foi possivel excluir: " + e.getMessage());
        e.printStackTrace();
    }
}
}
```

Descrição: Base fundamental para criação de um software que interage com dados

Justificativa: Construir um software escalável

Diagrama de classes:

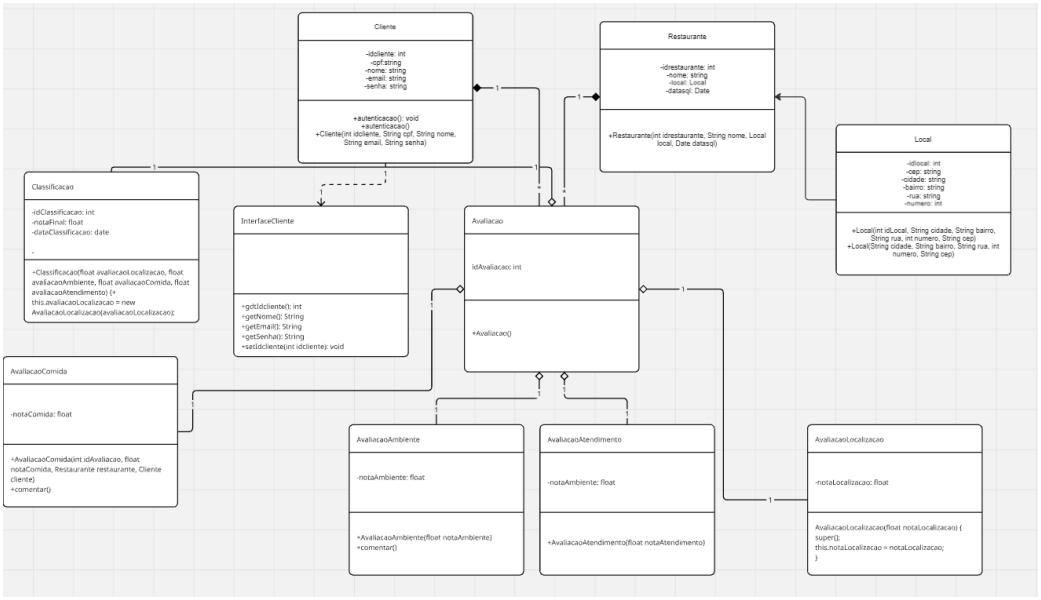
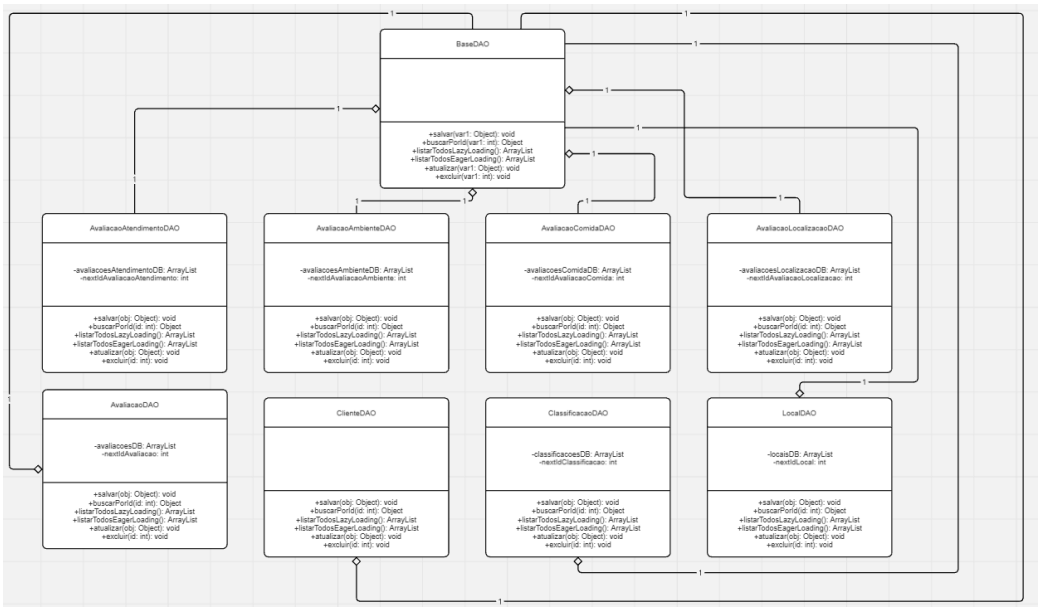


Diagrama de classes DAO:



Integrantes do grupo:

Barbara Malta Moraes – 202402898892

Breno Chaves - 202402798502

Thiago Neves - 202307539741