



# TP1 – TAP

## SISTEMA DE GESTÃO DE RESTAURANTE

ALUNA: ANA CAROLINE DA ROCHA BRAZ

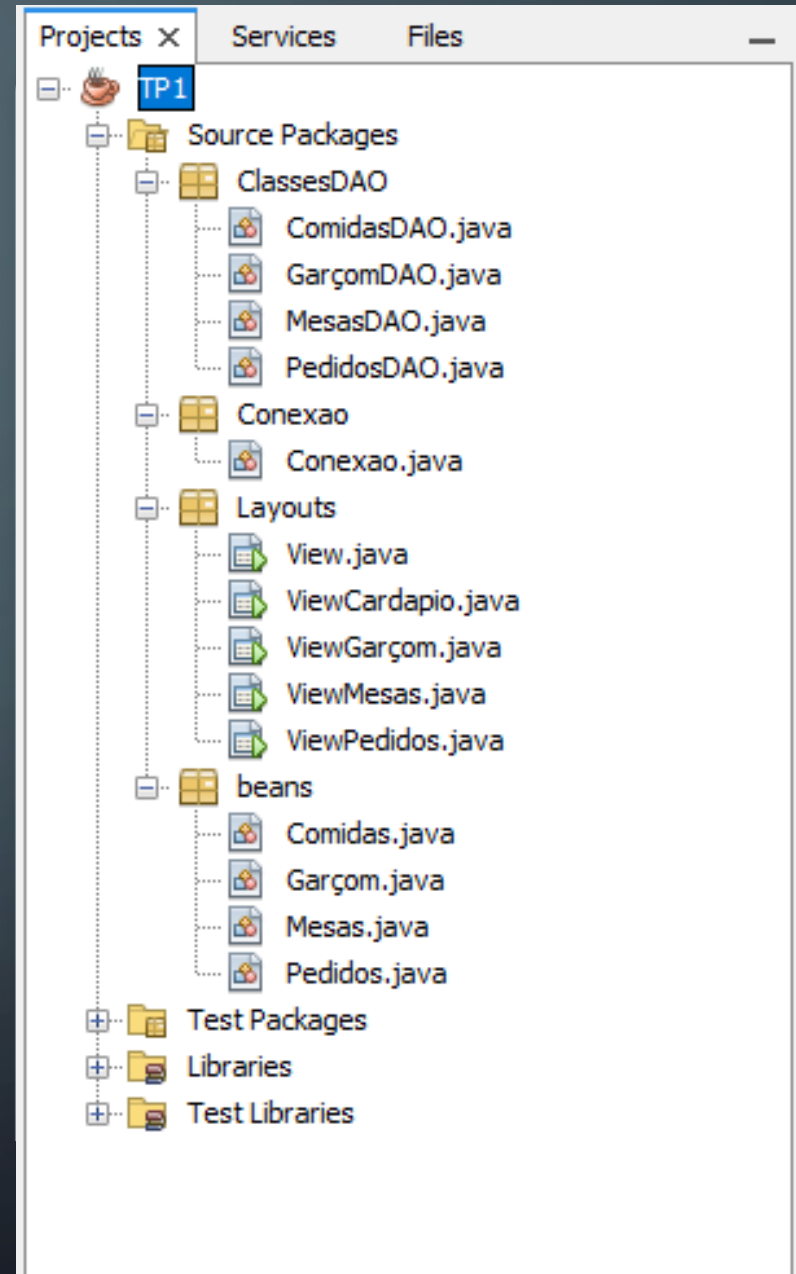
MATRÍCULA: 22050003

# IDE E MYSQL

- Para a realização do trabalho foi utilizada a IDE netbeans, pois, além da dificuldade em achar ajuda na internet, a IDE eclipse deu muitos erros sendo um deles a conexão com o banco de dados
- Para criar o banco de dados foi utilizado o programa MySQL

# CLASSES

- Foram criadas 4 pacotes para separar as classes e suas principais funções
- O pacote “ClassesDAO” possui todas as classes necessárias para a conexão com o banco de dados
- O pacote “Conexao” possui a classe para a conexão com o banco de dados
- O pacote “Layouts” possui as classes necessárias para interface gráfica
- O pacote “beans” possui todas as classes necessários com métodos get e set para serem utilizadas nas outras classes



# Pacote beans

## CLASSE COMIDAS.JAVA

- Nessa classe foram declaradas as variáveis necessárias para a criação de um cardápio:
  - Nome da comida, ingredientes e seu valor
  - O id é referente ao id dado pelo banco de dados

```
public class Comidas {  
    private int id;  
    private String ingredientes, nome;  
    private double valor;  
  
    public int getId() { ...3 lines }  
  
    public void setId(int id) { ...3 lines }  
  
    public String getIngredientes() { ...3 lines }  
  
    public void setIngredientes(String ingredientes) { ...3 lines }  
  
    public String getNome() { ...3 lines }  
  
    public void setNome(String nome) { ...3 lines }  
  
    public double getValor() { ...3 lines }  
  
    public void setValor(double valor) { ...3 lines }
```

## CLASSE GARÇOM.JAVA

- Nessa classe foram declarados as variáveis necessárias para um garçom ser cadastrado
  - Nome e salário
  - O id é referente ao id dado pelo banco de dados

```
public class Garçom {  
    private int id;  
    private String nome;  
    private double salario;  
  
    public int getId() { ...3 lines }  
  
    public void setId(int id) { ...3 lines }  
  
    public String getNome() { ...3 lines }  
  
    public void setNome(String nome) { ...3 lines }  
  
    public double getSalario() { ...3 lines }  
  
    public void setSalario(double salario) { ...3 lines }  
}
```

# Pacote beans

## CLASSE MESAS.JAVA

- Nessa classe foram declaradas as variáveis necessárias para obter informações sobre certa mesa:
  - Numero (da mesa)
  - idPedidos – se refere ao id do pedido que aquela certa mesa pediu
  - idGarçom – se refere ao id do garçom que atendeu a mesa
- O id é referente ao id dado pelo banco de dados

## CLASSE PEDIDOS.JAVA

- Nessa classe foram declaradas as variáveis necessárias para obter informações sobre certo pedido:
  - Observação – caso algum cliente tenha a fazer sobre o pedido
  - Status – se o pedido está em prepara, finalizado ou em atrado
  - idComida – se refere ao id da comida que foi pedido
  - O id é referente ao id dado pelo banco de dados

```
public class Mesas {  
    private int id;  
    private int numero;  
    private Pedidos idPedidos;  
    private Garçom idGarçom;  
  
    public int getNumero() {...3 lines }  
    public void setNumero(int numero) {...3 lines }  
    public int getId() {...3 lines }  
    public void setId(int id) {...3 lines }  
    public Pedidos getIdPedidos() {...3 lines }  
    public void setIdPedidos(Pedidos idPedidos) {...3 lines }  
    public Garçom getIdGarçom() {...3 lines }  
    public void setIdGarçom(Garçom idGarçom) {...3 lines }  
}
```

```
public class Pedidos {  
    private int id;  
    private String observação,status;  
    private Comidas idComidas;  
  
    public int getId() {...3 lines }  
    public void setId(int id) {...3 lines }  
    public String getObservação() {...3 lines }  
    public void setObservação(String observação) {...3 lines }  
    public String getStatus() {...3 lines }  
    public void setStatus(String status) {...3 lines }  
    public Comidas getIdComidas() {...3 lines }  
    public void setIdComidas(Comidas idComidas) {...3 lines }  
}
```

# PACOTE CONEXAO - CLASSE CONEXAO

Importações necessárias  
para a construção da classe

```
import java.sql.*;
import java.util.logging.Level;
import java.util.logging.Logger;
public class Conexao {
```

Declaração das viáveis necessárias  
Para realizar a conexão com o banco de dados

```
// Declaração das variáveis necessárias
private static final String DRIVER = "com.mysql.jdbc.Driver";
private static final String url = "jdbc:mysql://localhost:3306/mydb";
private static final String user = "root";
private static final String password = "112417sma";
```

```
//Nesse método é feita a conexão com o banco de dados, em caso de erro um erro é lançado
public static Connection getConnection()
{
    try
    {
        Class.forName(DRIVER);
        return DriverManager.getConnection(url, user, password);
    }
    catch(ClassNotFoundException | SQLException ex)
    {
        throw new RuntimeException("Erro na conexão: ", ex);
    }
}
```

- Nesse método é feita a conexão com o banco de dados atrás do DriverManager
- Caso não haja a conexão é lançada um erro de "Tempo ultrapassado"

# PACOTE CONEXÃO – CLASSE CONEXÃO (CONTINUAÇÃO)

```
//Nesse método é feito o fechamento da conexão
public static void closeConnection(Connection con)
{
    try {
        if(con != null) {
            con.close();
        }
    }
    catch (SQLException ex) {
        Logger.getLogger(Conexao.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

- Nesse método é realizado o fechamento da conexão
- Caso haja erro é lançado um erro de serialização de um objeto

```
//Nesse método é feito o fechamento da conexão e da instrução que é passada pro banco de dados
public static void closeConnection(Connection con, PreparedStatement stmt)
{
    closeConnection(con);

    try {
        if(stmt != null) {
            stmt.close();
        }
    }
    catch (SQLException ex) {
        Logger.getLogger(Conexao.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

- Nesse método é realizado o fechamento da conexão e da instrução que é passado para o banco de dados
- Caso haja erro é lançado um erro de serialização de um objeto



# PACOTE CONEXÃO – CLASSE CONEXÃO (CONTINUAÇÃO)

```
//Nesse método é feito o fechamento da conexão, da instrução e dos resultados guardados no banco de dados
public static void closeConnection(Connection con, PreparedStatement stmt, ResultSet rs)
{
    closeConnection(con,stmt);

    try {
        if(rs != null) {
            rs.close();
        }
    }
    catch (SQLException ex) {
        Logger.getLogger(Conexao.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

- Nesse último método da classe é feito o fechamento da conexão, da instrução passa para o banco de dados E dos resultados guardados no banco de dados
  - Caso haja erro é lançado um erro de serialização de um objeto



# PACOTE CLASSESDAO – CLASSE COMIDASDAO

- Foi realizada a importações dos pacotes necessários para criação da classe

```
package ClassesDAO;  
  
import Conexao.Conexao;  
import beans.Comidas;  
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.util.ArrayList;  
import java.util.List;  
import javax.swing.JOptionPane;
```

# PACOTE CLASSESDAO – CLASSE COMIDASDAO

```
public class ComidasDAO {

    /* Neste método é cadastrado uma nova comida no cardápio
    É feita a conexão com o banco de dados e nela é inserida as novas comidas ou bebidas, os ingredientes e o valor.
    Caso tudo de certo, um painel será mostrado "Salvo com sucesso", caso contrário é mostrado o erro*/
    public void create(Comidas c) {

        Connection con = Conexao.getConnection();
        PreparedStatement stmt = null;

        try {
            stmt = con.prepareStatement("INSERT INTO comidas (Nome, Ingredientes, Valor) VALUES (?, ?, ?)");
            stmt.setString(1, c.getNome());
            stmt.setString(2, c.getIngredientes());
            stmt.setDouble(3, c.getValor());

            stmt.executeUpdate();
            JOptionPane.showMessageDialog(null, "Salvo com sucesso!");
        }
        catch(SQLException ex) {
            JOptionPane.showMessageDialog(null, "Erro ao salvar: " + ex);
        }
        finally {
            Conexao.closeConnection(con, stmt);
        }
    }
}
```

- O primeiro método dessa classe é referente ao cadastro da comida ou bebida ao cardápio
- É feita a conexão com o banco de dados e nele é inserido os nomes das novas comidas e bebidas, os ingredientes e o seu valor
- Quando cadastrado é mostrando na janela que foi “Salvo com sucesso”, caso contrário uma mensagem de erro é mostrada na tela indicando qual erro foi gerado
- Por fim é feita o fechamento da conexão e do banco de dados

# PACOTE CLASSESDAO – CLASSE COMIDASDAO

```
public List<Comidas> read() {
    Connection con = Conexao.getConnection();
    PreparedStatement stmt = null;
    ResultSet rs = null;

    List<Comidas> comida = new ArrayList<>();

    try{
        stmt = con.prepareStatement("SELECT * FROM comidas");
        rs = stmt.executeQuery();

        while(rs.next()){
            Comidas c = new Comidas();

            c.setId(rs.getInt("idComidas"));
            c.setNome(rs.getString("Nome"));
            c.setIngredientes(rs.getString("Ingredientes"));
            c.setValor(rs.getDouble("Valor"));
            comida.add(c);
        }
    } catch (SQLException ex) {
        JOptionPane.showMessageDialog(null, "Erro ao salvar: " + ex);
    } finally{
        Conexao.closeConnection(con, stmt, rs);
    }
    return comida;
}
```

- No segundo método é feita a listagem das comidas e bebidas para aparecerem na interface gráfica
- É feita a conexão com o banco de dados
- É feito um ArrayList para salvar as comidas/bebidas
- São selecionadas as comidas e durante o laço é feita a inclusão de cada comida/bebida no ArrayList
- Caso haja algum erro é mostrado na tela o erro
- E, por fim, é feita o fechamento da conexão e do banco de dados e retornado o ArrayList criado

# PACOTE CLASSESDAO – CLASSE COMIDASDAO

```
/*Neste método acontece a alteração das informações de certa comida ou bebida, caso necessário no banco de dados*/  
public void update(Comidas c) {  
  
    Connection con = Conexao.getConnection();  
    PreparedStatement stmt = null;  
  
    try {  
        stmt = con.prepareStatement("UPDATE comidas SET Nome = ?, Ingredientes = ?, Valor = ? WHERE idComidas = ?");  
        stmt.setString(1, c.getNome());  
        stmt.setString(2, c.getIngredientes());  
        stmt.setDouble(3, c.getValor());  
        stmt.setInt(4, c.getId());  
  
        stmt.executeUpdate();  
        JOptionPane.showMessageDialog(null, "Atualizado com sucesso!");  
    }  
    catch(SQLException ex) {  
        JOptionPane.showMessageDialog(null, "Erro ao atualizar: " + ex);  
    }  
    finally {  
        Conexao.closeConnection(con, stmt);  
    }  
}
```

- Nesse método é feita a alteração de comida/bebida no banco de dados
  - É feita a conexão com o banco de dados
- Utilizando o idComidas é possível localizar qual será a comida/bebida a ser atualizada
  - Caso haja erro é lançado na tela o erro dado
  - E, por fim, é feito o fechamento da conexão e do banco de dados

# PACOTE CLASSESDAO – CLASSE COMIDASDAO

```
/*Neste método acontece a exclusão de certa comida ou bebida no banco de dados*/
public void delete(Comidas c) {

    Connection con = Conexao.getConnection();
    PreparedStatement stmt = null;

    try {
        stmt = con.prepareStatement("DELETE FROM comidas WHERE idComidas = ?");
        stmt.setInt(1, c.getId());

        stmt.executeUpdate();
        JOptionPane.showMessageDialog(null, "Excluido com sucesso!");
    }
    catch(SQLException ex) {
        JOptionPane.showMessageDialog(null, "Erro ao excluir: " + ex);
    }
    finally {
        Conexao.closeConnection(con, stmt);
    }
}
```

- Nesse último método é feita a exclusão de certa comida ou bebida do banco de dados
- É feita a conexão com o banco de dados
- Utilizando o idComidas é possível selecionar qual comida ou bebida será excluída
- Caso haja algum erro, é lançada na tela o erro
- E, por fim, é feito o fechamento da conexão e do banco de dados

# PACOTE CLASSESDAO – CLASSE GARÇOMDAO

```
package ClassesDAO;

/**
 *
 * @author anaca
 */
import beans.Garçom;
import Conexao.Conexao;
import java.sql.Connection;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import javax.swing.JOptionPane;
```

- Foi realizada a importações dos pacotes necessários para criação da classe

# PACOTE CLASSESDAO – CLASSE GARÇOMDAO

```
public class GarçomDAO{

    /*Neste método é feito o cadastro do garçom. Informações, como Nome e Salário, fazem parte desse método */
    public void create(Garçom g) {...20 lines }

    /*Neste método é listado todos os garçons para aparecerem na interface gráfica*/
    public List<Garçom> read(){...31 lines }

    /*Neste método acontece a alteração das informações do garçom, caso necessário no banco de dados*/
    public void update(Garçom g) {...21 lines }

    /*Neste método acontece a exclusão do garçom no banco de dados*/
    public void delete(Garçom g) {...19 lines }
```

- Seguindo a mesma lógica e código do ComidasDAO, a classe GarçomDAO possui os mesmos métodos apenas mudando as variáveis para o cadastro, listagem, atualização e exclusão
- Nessa classe as variáveis utilizadas foram apenas 2: Nome e Salário



# PACOTE CLASSESDAO – CLASSE MESASDAO

```
package ClassesDAO;

import Conexao.Conexao;
import beans.Mesas;
import beans.Pedidos;
import beans.Garçom;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import javax.swing.JOptionPane;
```

```
/**
```

- Foi realizada a importações dos pacotes necessários para criação da classe

# PACOTE CLASSESDAO – CLASSE MESASDAO

```
public class MesasDAO {

    /* Nesse método é cadastrada uma nova mesa, e junto com ela os id dos pedidos e o id dos garçons*/
    public void create(Mesas m) {

        Connection con = Conexao.getConnection();
        PreparedStatement stmt = null;

        try {
            stmt = con.prepareStatement("INSERT INTO mesas (numero, idPedidos, idGarçom) VALUES (?, ?, ?)");
            stmt.setInt(1, m.getNumero());
            stmt.setInt(2, m.getIdPedidos().getId()); //FK
            stmt.setInt(3, m.getIdGarçom().getId()); //FK

            stmt.executeUpdate();
            JOptionPane.showMessageDialog(null, "Salvo com sucesso!");
        }
        catch(SQLException ex) {
            JOptionPane.showMessageDialog(null, "Erro ao salvar: " + ex);
        }
        finally {
            Conexao.closeConnection(con, stmt);
        }
    }
}
```

- Utilizando a mesma lógica e código das classes anteriores, o método de cadastro possui apenas a diferença ao referenciar os ids do pedido e do garçom, pois essas são as foreign Keys utilizadas no banco de dados

# PACOTE CLASSESDAO – CLASSE MESASDAO

```
/*Neste método é listado a mesas para serem mostradas na interface gráfica*/
public List<Mesas> read(){
    Connection con = Conexao.getConnection();
    PreparedStatement stmt = null;
    ResultSet rs = null;

    List<Mesas> mesas = new ArrayList<>();

    try{
        stmt = con.prepareStatement("select m.idMesas as mid, numero as Mesa, m.idPedidos as Pedido, "
            + "m.idGarçom as Garçom, p.idComidas as Comida from mesas m inner join pedidos p on "
            + "p.idPedidos = m.idPedidos");
        rs = stmt.executeQuery();

        while(rs.next()){
            Mesas m = new Mesas();

            m.setId(rs.getInt("mid"));
            m.setNumero(rs.getInt("Mesa"));

            Pedidos pedidos = new Pedidos();
            pedidos.setId(rs.getInt("Pedido"));

            Garçom garçom = new Garçom();
            garçom.setId(rs.getInt("Garçom"));

            m.setIdPedidos(pedidos);
            m.setIdGarçom(garçom);

            mesas.add(m);
        }
    } catch(SQLException ex){
        JOptionPane.showMessageDialog(null, "Erro ao salvar: " + ex);
    } finally{
        Conexao.closeConnection(con, stmt, rs);
    }
    return mesas;
}
```

- Utilizando a mesma lógica e código das classes anteriores, o método de listagem das Mesas possui a diferença na passagem de instrução, o qual é passado apenas as linhas necessárias para serem vistas e a junção das tabelas de acordo com o id do Pedido
- Outra diferença é a forma de referenciar os ids do Pedido e do Garçom. É necessário criar um novo objeto para essa referência acontecer.
- São selecionadas as mesas durante o laço e feita a inclusão de cada uma na ArrayList
- Caso haja algum erro é mostrado na tela qual erro foi encontrado
- E, por fim, é feito o fechamento da conexão e do banco de dados e retornado as mesas

# PACOTE CLASSESDAO – CLASSE MESASDAO

```
/*Neste método acontece a alteração das informações da mesa, caso necessário no banco de dados*/  
public void update(Mesas m) {...22 lines }  
  
/*Neste método acontece a exclusão da mesa no banco de dados*/  
public void delete(Mesas m) {...19 lines }
```

- Assim como nas classes anteriores, os métodos de alteração e exclusão das mesas possuem os mesmos códigos, apenas alterando as variáveis
  - Caso haja algum erro é mostrado na tela qual erro foi encontrado
  - E, por fim, é feito o fechamento da conexão e do banco de dados

# PACOTE CLASSESDAO – CLASSE MESASDAO

- Nessa classe também deveria estar inclusa 2 métodos:
  - `public void mostrar_mesa()` – Esse método teria como função retornar todas as mesas cadastradas no banco de dados. Uma nova janela aparecia com uma listagem das mesas.
  - `public double fecha_conta()` – Esse método teria como função fechar a conta de uma certa mesa já com os 10% do garçom incluso. Uma nova janela aparecia com a mensagem “A conta da mesa X deu R\$ X”.
- Infelizmente, por conta de falta de conhecimento e dúvidas, não foi possível realizar esses dois métodos, mas elas estão feitas até certo ponto e comentadas no código enviado.

# PACOTE CLASSESDAO – CLASSE PEDIDOSDAO

```
package ClassesDAO;

import Conexao.Conexao;
import beans.Comidas;
import beans.Pedidos;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import javax.swing.JOptionPane;
```

- Foi realizada a importações dos pacotes necessários para criação da classe

# PACOTE CLASSESDAO – CLASSE PEDIDOSDAO

```
public class PedidosDAO {
    /* Nesse método é cadastrado um novo pedido, o seu status e observações caso o cliente tenha alguma*/
    public void create(Pedidos p) {

        Connection con = Conexao.getConnection();
        PreparedStatement stmt = null;

        try {
            stmt = con.prepareStatement("INSERT INTO pedidos (idComidas, Status, Observações) VALUES (?, ?, ?)");
            stmt.setInt(1, p.getIdComidas().getId()); //FK
            stmt.setString(2, p.getStatus());
            stmt.setString(3, p.getObservação());

            stmt.executeUpdate();
            JOptionPane.showMessageDialog(null, "Salvo com sucesso!");
        }
        catch(SQLException ex) {
            JOptionPane.showMessageDialog(null, "Erro ao salvar: " + ex);
        }
        finally {
            Conexao.closeConnection(con, stmt);
        }
    }
}
```

- Seguindo a mesma lógica e código das classes anteriores, o método de cadastro da pedido possui a única diferença nas suas variáveis que serão inseridas no banco de dados
- Caso haja erro, é mostrado na tela qual foi o erro encontrado
- E, por fim, é feito o fechamento da conexão e do banco de dados



# PACOTE CLASSESDAO – CLASSE PEDIDOSDAO

```
/*Neste método é listado os pedidos para serem mostradas na interface gráfica*/
public List<Pedidos> read(){
    Connection con = Conexao.getConnection();
    PreparedStatement stmt = null;
    ResultSet rs = null;

    List<Pedidos> pedidos = new ArrayList<>();

    try{
        stmt = con.prepareStatement("select p.idPedidos as pid, Observações, "
            + "Status, p.idComidas as Comida, c.idComidas as cid, nome, "
            + "Ingredientes, Valor from pedidos p inner join comidas c "
            + "on c.idComidas = p.idComidas");
        rs = stmt.executeQuery();

        while(rs.next()){
            Pedidos p = new Pedidos();

            p.setId(rs.getInt("pid"));
            p.setStatus(rs.getString("Status"));
            p.setObservação(rs.getString("Observações"));

            Comidas comidas = new Comidas();
            comidas.setId(rs.getInt("Comida"));

            p.setIdComidas(comidas);

            pedidos.add(p);
        }
    }
    catch(SQLException ex){
        JOptionPane.showMessageDialog(null, "Erro ao salvar: " + ex);
    }
    finally{
        Conexao.closeConnection(con, stmt, rs);
    }
    return pedidos;
}
```

- Utilizando a mesma lógica e código das classes anteriores, a listagem dos pedidos é feito de maneira parecida com a classe Mesa. Além da mudança das variáveis, a instrução passado para o banco de dados junta de acordo com o id da Comida
- São selecionadas os pedidos durante o laço e feita a inclusão de cada um no ArrayList
- Caso haja algum erro é mostrado na tela qual erro foi encontrado
- E, por fim, é feito o fechamento da conexão e do banco de dados e retornado os pedidos

# PACOTE CLASSESDAO – CLASSE PEDIDOSDAO

```
/*Neste método acontece a alteração das informações do pedido,  
sendo muito necessário já que o status do pedido deve ser alterado manualmente*/  
public void update(Pedidos p) {  
  
    Connection con = Conexao.getConnection();  
    PreparedStatement stmt = null;  
  
    try {  
        stmt = con.prepareStatement("UPDATE pedidos SET idComidas = ?, Status = ?, "  
            + "Observações = ? WHERE idPedidos = ?");  
        stmt.setInt(1, p.getIdComidas().getId()); //FK  
        stmt.setString(2, p.getStatus());  
        stmt.setString(3, p.getObservação());  
        stmt.setInt(4, p.getId());  
  
        stmt.executeUpdate();  
        JOptionPane.showMessageDialog(null, "Atualizado com sucesso!");  
    }  
    catch(SQLException ex) {  
        JOptionPane.showMessageDialog(null, "Erro ao atualizar: " + ex);  
    }  
    finally {  
        Conexao.closeConnection(con, stmt);  
    }  
}
```

- Nesse método, assim como nas classes anteriores, é feito a atualização ou alteração do pedido por meio do id do Pedido
- Caso haja algum erro é mostrado na tela qual erro foi encontrado
- E, por fim, é feito o fechamento da conexão e do banco de dados

# PACOTE CLASSESDAO – CLASSE PEDIDOSDAO

```
/*Neste método acontece a exclusão do pedido no banco de dados*/
public void delete(Pedidos p) {

    Connection con = Conexao.getConnection();
    PreparedStatement stmt = null;

    try {
        stmt = con.prepareStatement("DELETE FROM pedidos WHERE idPedidos = ?");
        stmt.setInt(1, p.getId());

        stmt.executeUpdate();
        JOptionPane.showMessageDialog(null, "Excluido com sucesso!");
    }
    catch(SQLException ex) {
        JOptionPane.showMessageDialog(null, "Erro ao excluir: " + ex);
    }
    finally {
        Conexao.closeConnection(con, stmt);
    }
}
```

- Nesse método, assim como nas classes anteriores, é feito a exclusão do pedido por meio do id do Pedido
- Caso haja algum erro é mostrado na tela qual erro foi encontrado
- E, por fim, é feito o fechamento da conexão e do banco de dados

# PACOTE LAYOUTS – CLASSE VIEW

## Sistema de Gestão de Restaurante

By Ana Caroline Braz

**Controle de Garçom**

**Controle de Mesa**

**Controle de Pedidos**

**Controle do Cardápio**

- Nessa classe, 4 botões foram criados:
  - Controle de Garçom
  - Controle de Mesa
  - Controle de Pedidos
  - Controle do Cardápio
- Quando clicados uma nova janela é aberta de certo controle e um layout referente àquele controle

# PACOTE LAYOUTS – CLASSE VIEWCARDAPIO

## Controle de Cardápio

Nome:

Ingredientes:

Valor:

**Cadastrar**

**Atualizar**

**Excluir**

ID	Nome	Ingredientes	Valor

- Nessa classe, temos o local para digitar:
  - Nome
  - Ingredientes
  - Valor
- Após clicar no botão “Cadastrar” a comida ou bebida é registrada no banco de dados e na tabela logo abaixo, ficando sempre visível
- Para atualizar o cardápio, basta clicar em uma linha da tabela abaixo. Fazendo isso, os dados dessa linha irão aparecer nos locais de escrita e assim pode-se mudar as informações. Após clicar no botão “Atualizar”, a informação modificada é atualizada no bando de dados e na tabela
- Para a exclusão de certa comida ou bebida, basta selecionar um linha da tabela abaixo e clicar no botão “Excluir”

# PACOTE LAYOUTS – CLASSE VIEWCARDAPIO

## Controle de Garçon

**Nome:**

**Salário:**

**Cadastrar**

**Atualizar**

**Excluir**

ID

Nome

Salário

- Nessa classe, temos o local para digitar:
  - Nome
  - Salário
- Após clicar no botão “Cadastrar” o garçon é registrado no banco de dados e na tabela logo abaixo, ficando sempre visível
- Para atualizar as informações do garçon, basta clicar em uma linha da tabela abaixo. Fazendo isso, os dados dessa linha irão aparecer nos locais de escrita e assim pode-se mudar as informações. Após clicar no botão “Atualizar”, a informação modificada é atualizada no bando de dados e na tabela
- Para a exclusão de certo garçon, basta selecionar um linha da tabela abaixo e clicar no botão “Excluir”

# PACOTE LAYOUTS – CLASSE VIEWCARDAPIO

## Controle de Mesas

Mesa:

**Cadastrar**

Pedido:

**Alterar**

Garçom:

**Excluir**

**Mostrar Mesa**

**Fechar Conta**

IdMesa	Mesa	IdPedido	IdGarçom

- Nessa classe, temos o local para digitar:
  - Número da Mesa
  - Id do pedido feito pela mesa
  - Id do garçom que atendeu a mesa
- Após clicar no botão “Cadastrar” a mesa é registrada no banco de dados e na tabela logo abaixo, ficando sempre visível
- Para atualizar a mesa, basta clicar em uma linha da tabela abaixo. Fazendo isso, os dados dessa linha irão aparecer nos locais de escrita e assim pode-se mudar as informações. Após clicar no botão “Atualizar”, a informação modificada é atualizada no bando de dados e na tabela
- Para a exclusão de certa mesa, basta selecionar um linha da tabela abaixo e clicar no botão “Excluir”



# PACOTE LAYOUTS – CLASSE VIEWCARDAPIO

## Controle de Mesas

Mesa:

Cadastrar

Pedido:

Alterar

Garçom:

Excluir

Mostrar Mesa

Fechar Conta

IdMesa	Mesa	IdPedido	IdGarçom
--------	------	----------	----------

- Os botões “Mostrar Mesa” e “Fechar conta” não funcionam por conta dos códigos não finalizados na classe MesasDAO. Elas foram adicionadas apenas para mostrar como ficariam no layout.
- O botão “Mostrar Mesa” deveria mostrar em uma janela todas as mesas cadastradas no sistema
- O botão “Fechar Conta” mostraria em uma janela o total da conta de uma mesa já com os 10% do garçom

# PACOTE LAYOUTS – CLASSE VIEWPEDIDOS

## Controle de Pedidos

**Comida:**

**Status:**

**Observações:**

Pedir

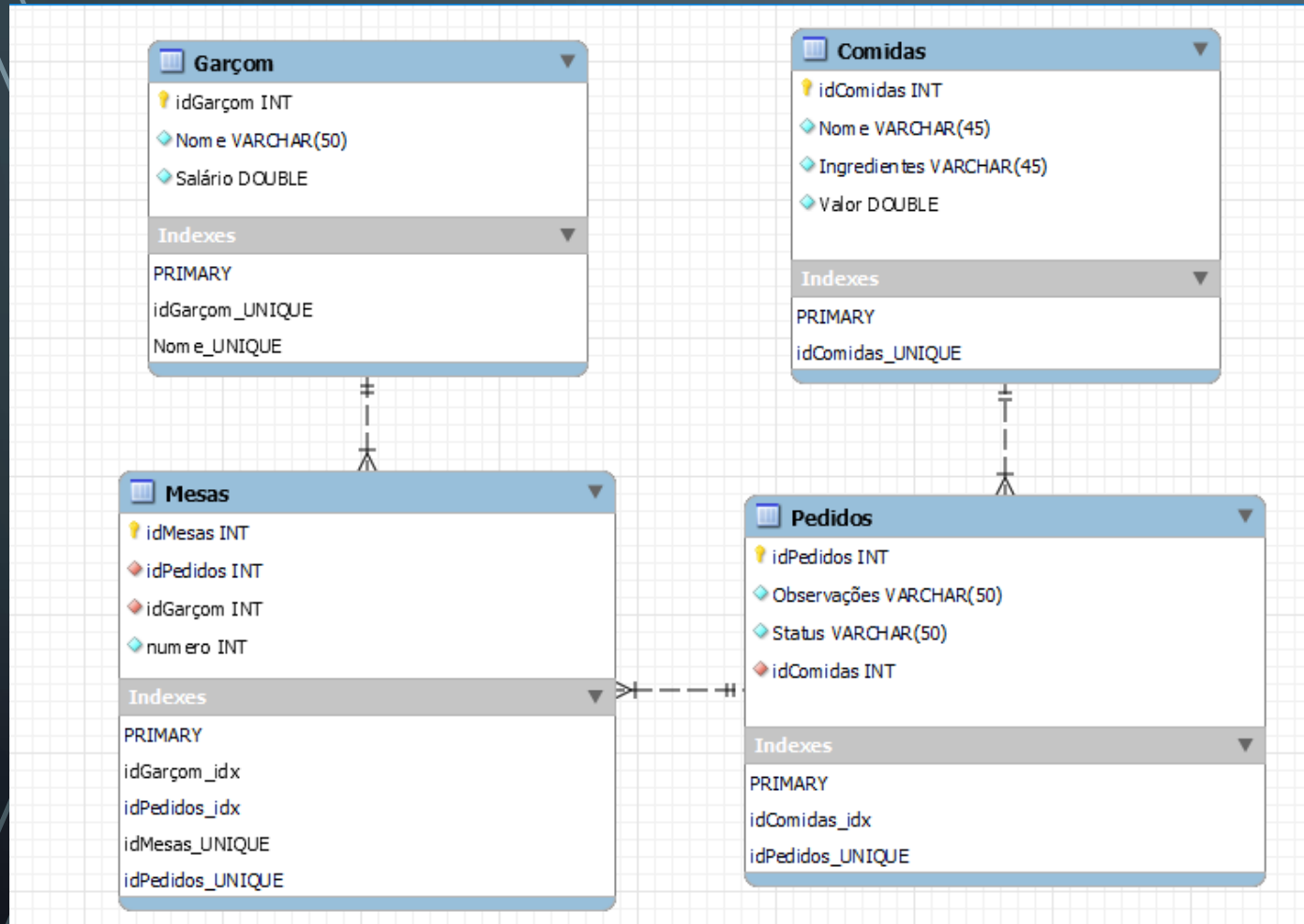
Alterar

Excluir

Pedidos	Comida	Status	Observações
---------	--------	--------	-------------

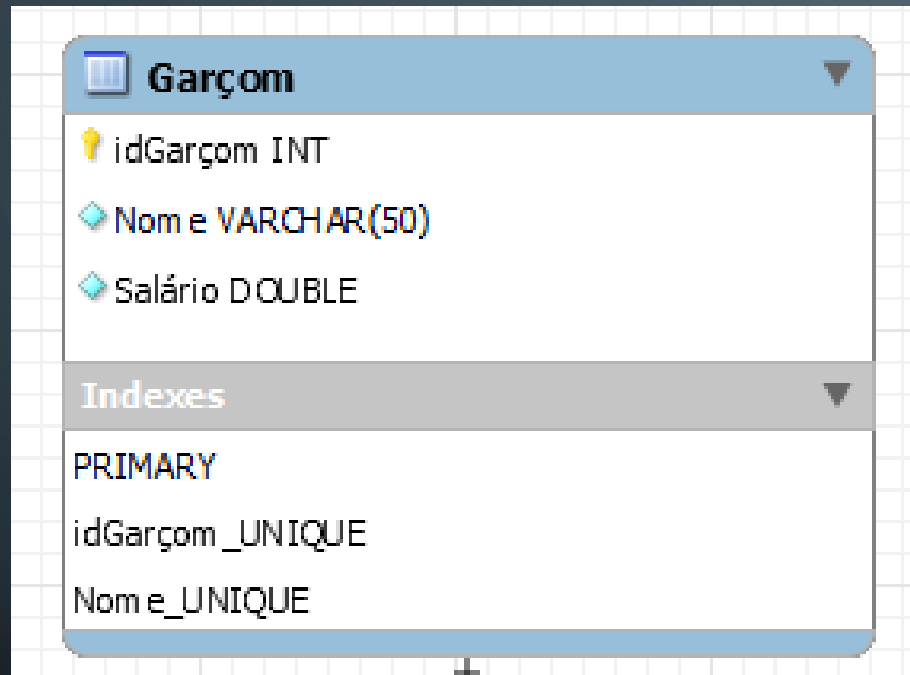
- Nessa classe, temos o local para digitar:
  - Id da Comida pedida
  - Status do pedido
  - Observações, caso o cliente tenha
- Após clicar no botão “Cadastrar” o pedido é registrado no banco de dados e na tabela logo abaixo, ficando sempre visível
- Para atualizar as informações do pedido, basta clicar em uma linha da tabela abaixo. Fazendo isso, os dados dessa linha irão aparecer nos locais de escrita e assim pode-se mudar as informações. Após clicar no botão “Atualizar”, a informação modificada é atualizada no bando de dados e na tabela.
- Para o status é necessário fazer a atualização manual, ou seja, é preciso utilizar o botão “Alterar” para atualizar a situação do pedido: em preparo, finalizado ou em atraso
- Para a exclusão de certo pedido, basta selecionar um linha da tabela abaixo e clicar no botão “Excluir”

# BANCO DE DADOS - MYSQL



- Foi feita um diagrama simples para representar o sistema de gestão de restaurante

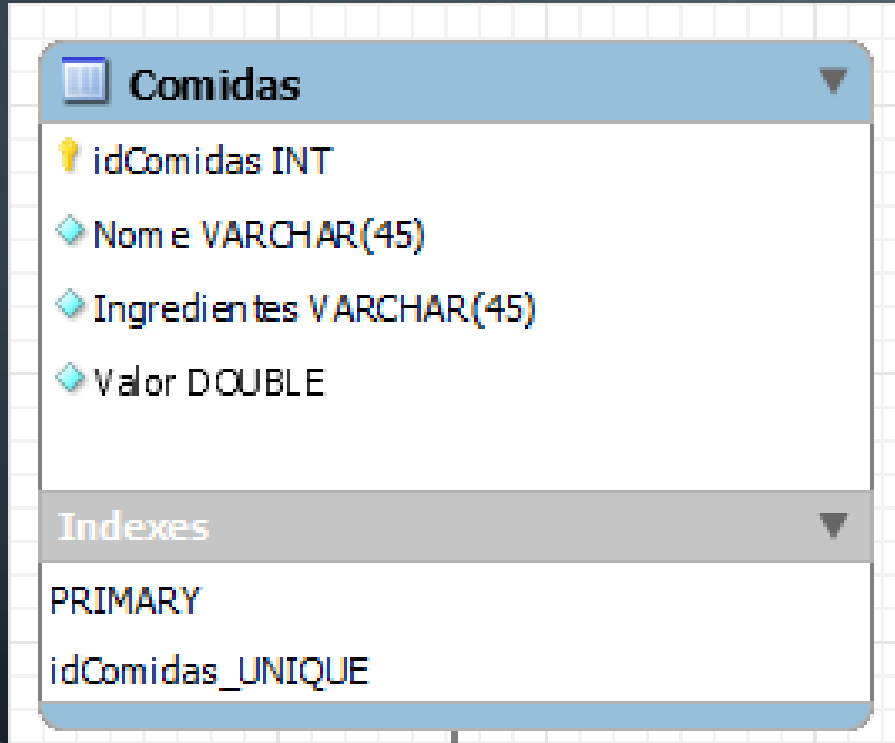
# BANCO DE DADOS - MYSQL



Garçom
idGarçom INT
Nome VARCHAR(50)
Salário DOUBLE
Indexes
PRIMARY
idGarçom_UNIQUE
Nome_UNIQUE

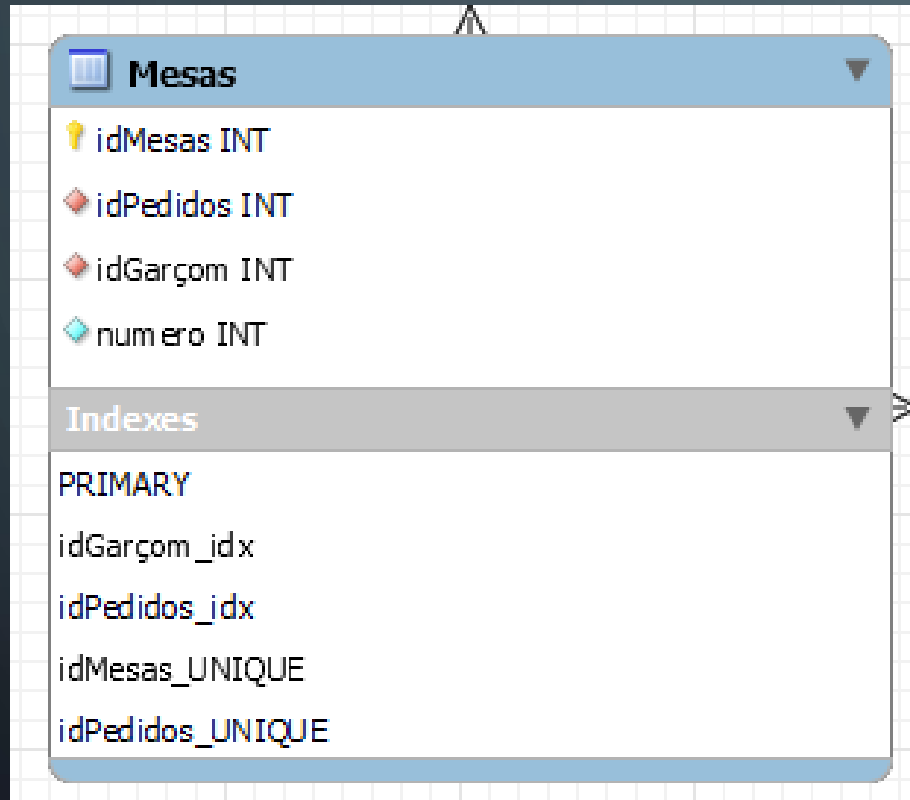
- A tabela Garçom possui 3 chaves:
  - idGarçom – Chave primária e única
  - Nome – Chave única
  - Salário
- Todas as chaves são não nulas

# BANCO DE DADOS - MYSQL



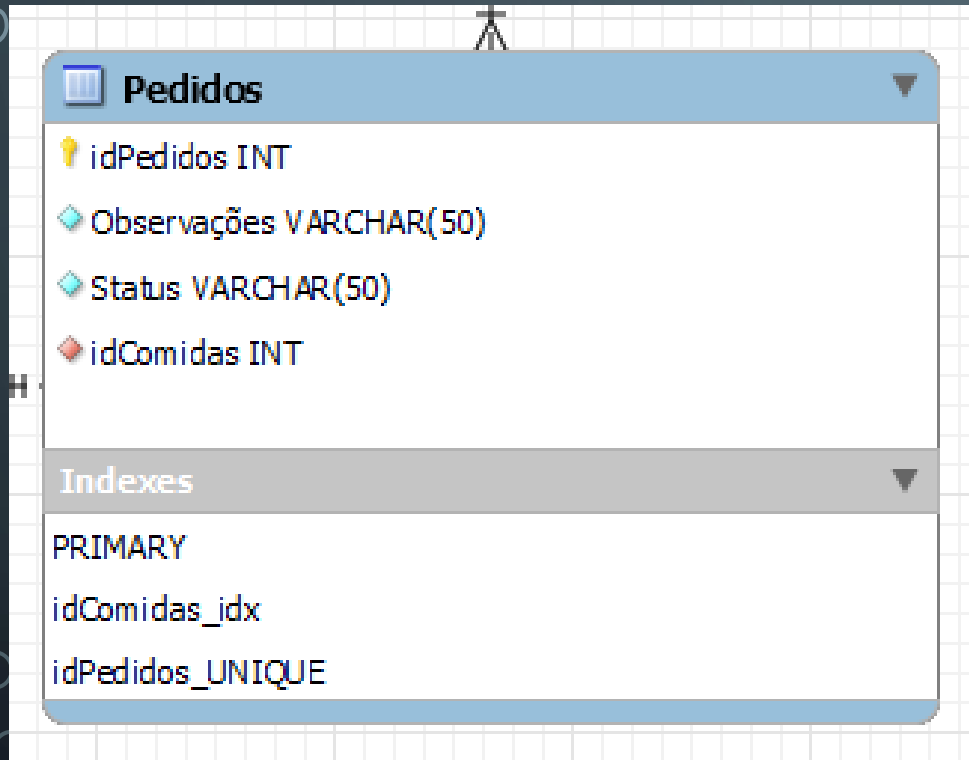
- A tabela Comidas possui 4 chaves:
  - idComidas— Chave primária e única
  - Nome
  - Salário
  - Valor
- Todas as chaves são não nulas

# BANCO DE DADOS - MYSQL



- A tabela Mesas possui 4 chaves:
  - idMesas – Chave primária e única
  - idPedidos – Chave única
  - idGarçom
  - numero – indica o número da mesa
- Todas as chaves são não nulas

# BANCO DE DADOS - MYSQL



- A tabelaPedidos possui 4 chaves:
  - idPedidos – Chave primária e única
  - Observações
  - Status – referente ao status do pedido
  - idComidas
- Todas as chaves são não nulas