Extensão da ferramenta Delphi2Java-II para suportar componentes de banco de dados

Janira Silveira

Joyce Martins - Orientadora

Roteiro da apresentação:

- **Introdução**
- **Objetivos**
- **Fundamentação teórica**
- **Desenvolvimento**
- **Conclusão**
- **Extensões**

Introdução:

- Necessidades atuais

  -mudanças tecnológicas

  -migração de aplicações para plataformas: Java e.Net

- Solução proposta

Objetivos: **Ampliar a ferramenta Delphi2Java-II incorporando as funcionalidades de acesso a banco de dados**

- migrar a ferramenta Delphi2Java-II de Delphi para Java
- converter os componentes de visualização de dados (*TDBCheckBox, TDBComboBox, TDBEdit, TDBGrid, TDBMemo, TDBRadioGroup e TDBText*)
- implementar a funcionalidade dos componentes de acesso a dados (*TDataBase, TQuery e TTable*) e agregando também a funcionalidade do componente *TDataSource*

Fundamentação teórica:

- **Migração de código**
- **Geradores de código**
- **Desenvolvimento de geradores de código**
- **Formulário Delphi**
- **Análise do formulário Delphi**
- **Padrão MVC**

Fundamentação teórica:

- **Acesso a banco de dados Java (JDBC)**
- **Classes bean**
- ***Template* Velocity**
- **Trabalhos correlatos**

- **Migração de código**:
  - diminuição dos custos
  - facilidade na manutenção
- **Geradores de código**:
  - vantagens e desvantagens
  - aplicações

❖ Desenvolvimento de geradores de código

- definir qual será a entrada e como será analisada

- identificar o que se deseja obter de saída

- interpretar e recuperar as informações da entrada, definir a formatação e a geração da saída

- gerar a saída a partir das informações extraídas do arquivo de entrada

Formulário Delphi:

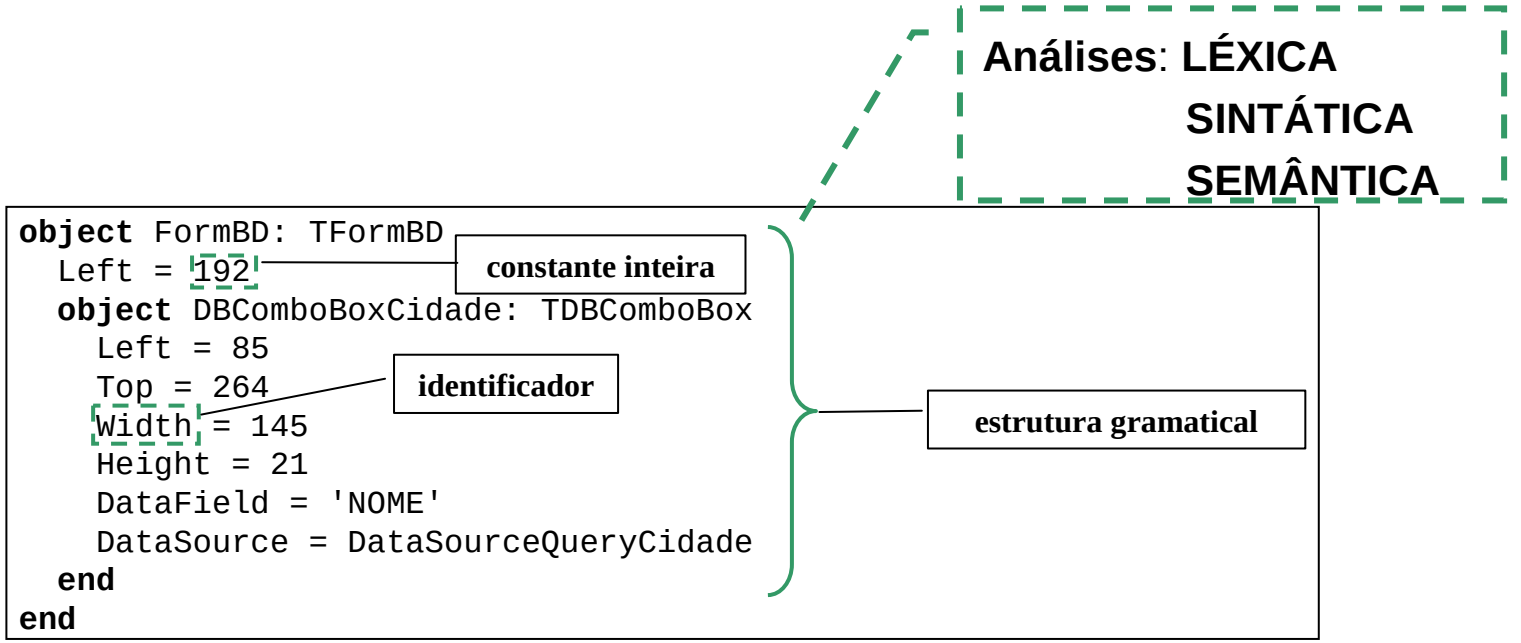
```
object FormBD: TFormBD
  Left = 192
  object DBComboBoxCidade: TDBComboBox
    Width = 145
    DataField = 'NOME'
    DataSource = DataSourceQueryCidade
  end
  object TableTBAluno: TTable
    DatabaseName = 'BDoracle'
    TableName = 'TBALUNO'
    object TableTBAlunoMATRICULA: TStringField
      FieldName = 'MATRICULA'
    end
    object TableTBAlunoNOME: TStringField
      FieldName = 'NOME'
    end
    object TableTBAlunoDESCRICAO: TMemoField
      FieldName = 'DESCRICAO'
    end
  end
end
```

identificação do componente

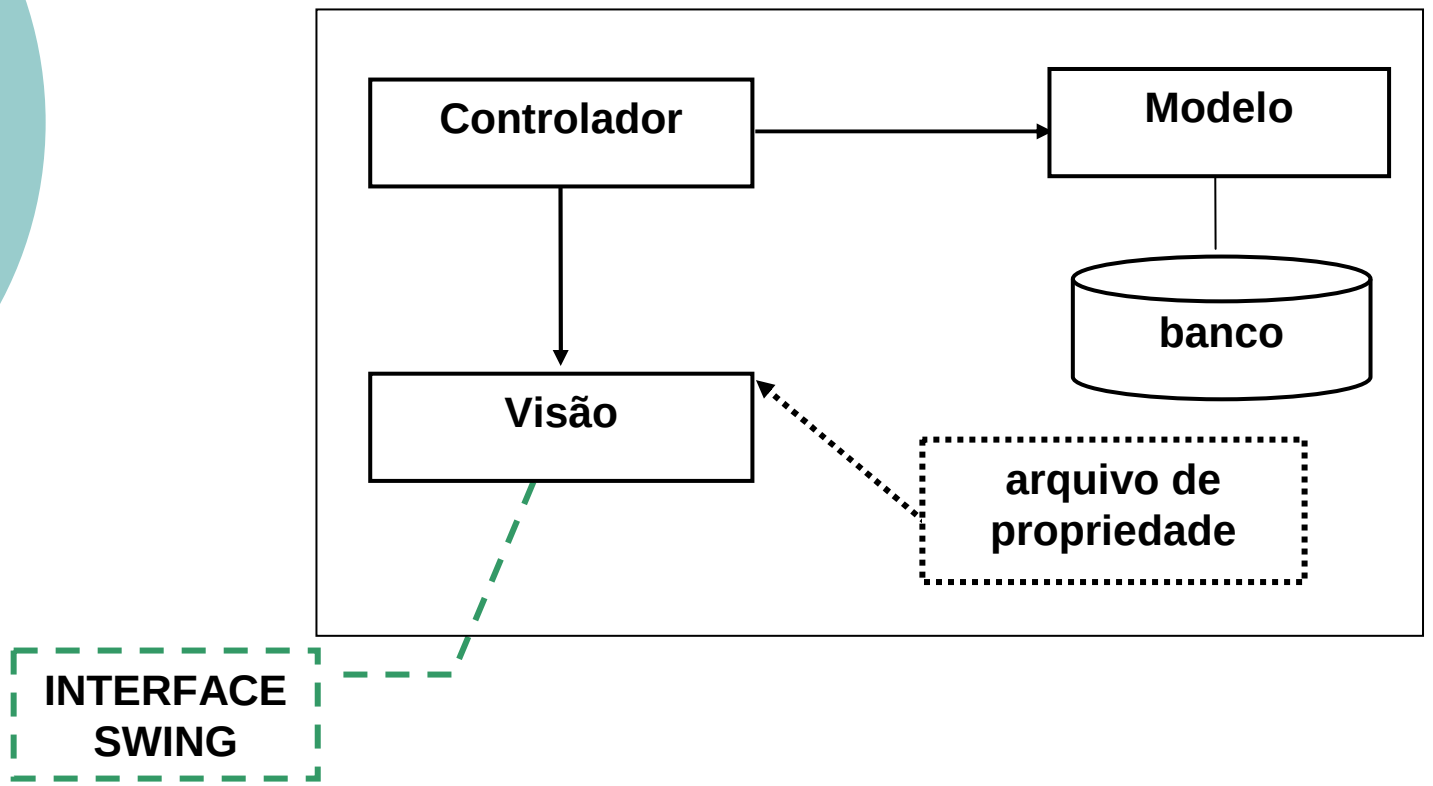
lista de propriedades

objeto componente

Análise do Formulário Delphi:



Padrão MVC:


Controlador
Modelo
banco
Visão
arquivo de propriedade
INTERFACE SWING

- **Acesso a banco de dados em Java (*JDBC*)**:

Tipos de *driver* JDBC:

- *driver* **JDBC /ODBC**
- *driver* de **acesso nativo**
- *driver* de **acesso por middleware**
- *driver* de **acesso direto ao servidor**

- **Acesso a banco de dados em Java(*JDBC*)**:

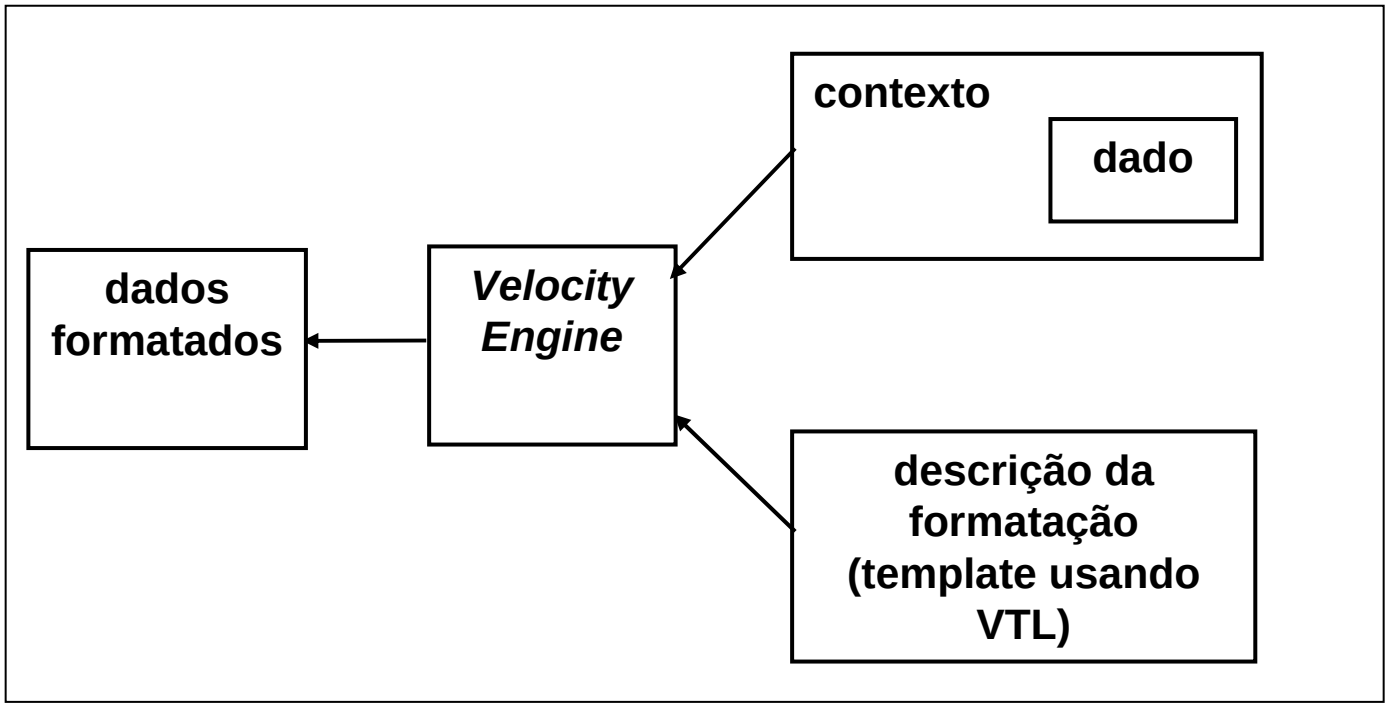
Classes de manipulação:

- *java.sql.Connection*
- *java.sql.DriverManager*
- *java.sql.PreparedStatement*
- *java.sql.ResultSet*

❖ Classes Bean:

```
public class BeanTBALUNO extends ConnectionManager{

    //-- Campos da tabela TBALUNO
    private String  NOME;
    //...
    public String getNOME(){
        return NOME;
    }

    public void setNOME(String NOME) {
        this.NOME = NOME;
    }
    //...
}
```

Template Velocity:

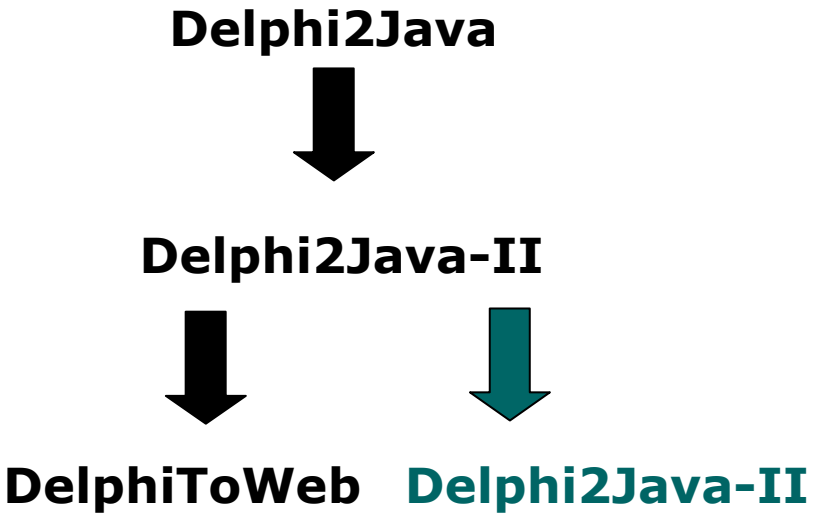


```
object DBComboBoxCidade: TDBComboBox
  Left = 85
  Top = 264
  Width = 145
  Height = 21
  DataField = 'NOME'
  DataSource = DataSourceQueryCidade
end
```

❖ *Template* Velocity:

```
#*Aqui encontra-se a estrutura dos componentes DB.*#

#foreach ($TDBComboBox in $arrayDBComboBox)        #*----Componente DBComboBox------*#
  $!{TDBComboBox.getClassePai()}.add(classeEvento.criaComboBox$!{TDBComboBox.getName()}(),
    javax.swing.JLayeredPane.DEFAULT_LAYER);
  $!{TDBComboBox.getName()}.setBackground(new java.awt.Color($!{TDBComboBox.setarBackground()}));
  $!{TDBComboBox.getName()}.setBounds($!{TDBComboBox.getLeft()},
                                      $!{TDBComboBox.getTop()},
                                       $!{TDBComboBox.getWidth()},
                                        $!{TDBComboBox.getHeight()});
  $!{TDBComboBox.getName()}.setVisible($!{TDBComboBox.getVisible()});
  #if ($!TDBComboBox.getFontJava())
    $!{TDBComboBox.getName()}.setFont(new java.awt.Font($!{TDBComboBox.getFontJava()}));
    $!{TDBComboBox.getName()}.setForeground(new java.awt.Color($!{TDBComboBox.getForeground()}));
  #end
  #if ($!TDBComboBox.getText())
    $!{TDBComboBox.getName()}.setText("$!{TDBComboBox.getText()}");
  #end
  #if ($!TDBComboBox.getEnabled())
    $!{TDBComboBox.getName()}.setEnabled($!{TDBComboBox.getEnabled()});
  #end
#end
```

Trabalhos Correlatos:



Desenvolvimento:

- **Requisitos**
- **Identificação dos componentes**
- **Especificação da saída**
- **Análise da entrada**
- **Especificação**
- **Implementação**

Requisitos Funcionais:

-converter componentes de visualização de dados de Delphi para Java

-converter conexões com banco de dados existentes nos componentes de acesso a dados de Delphi para Java

-efetuar a interligação dos componentes de acesso a dados com os componentes de visualização de dados

Requisitos Não Funcionais:

-manter o *layout* original e a funcionalidade dos componentes

-utilizar a biblioteca Swing e a JVM 1.4 para a implementação do *layout* da ferramenta

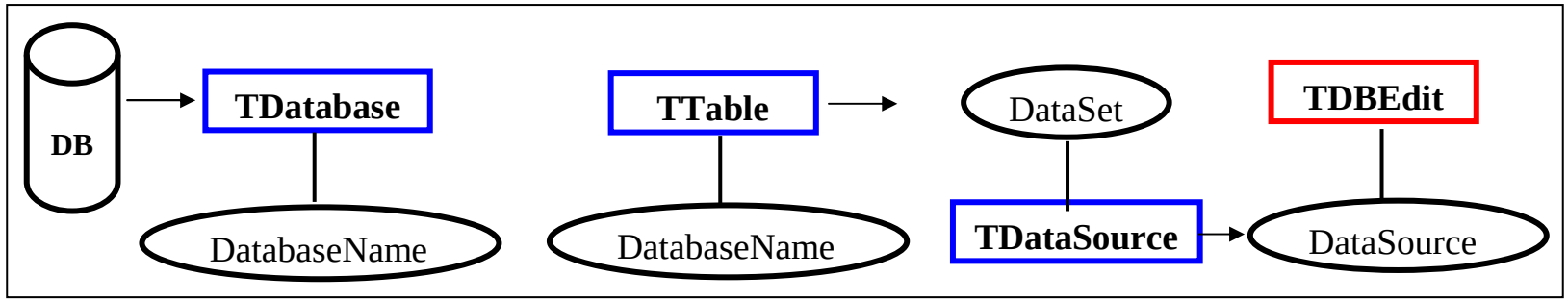
-utilizar o *driver* JDBC para integração do banco de dados

-utilizar o banco de dados Oracle8i para realizar os testes

-gerar código com a extensão .java

Identificação dos componentes

- Componentes de **visualização**
- Componentes de **visualização de dados**
- Componentes de **acesso a dados**



❖ Especificação da Saída:

- **view**: classe que desenha os componentes de visualização e visualização de dados

```
FormDB.add(classeEvento.criaComboBoxDBComboBoxCidade(),
                  javax.swing.JLayeredPane.DEFAULT_LAYER);
DBComboBoxCidade.setBackground(new java.awt.Color(229,229,229));
DBComboBoxCidade.setBounds(85, 264, 145, 21);
DBComboBoxCidade.setVisible(true);
DBComboBoxCidade.setFont(new java.awt.Font("MS Sans Serif", 0,12));
DBComboBoxCidade.setForeground(new java.awt.Color(0, 0, 0));
```

❖ Especificação da Saída:

- **model**:classes de acesso à camada de banco de dados

```
public class ConnectionManager {

    private static java.sql.Connection conn;

    /**Recupera o Objeto Connection*/
    public java.sql.Connection getConnection(String driver,
                                             String url,
                                             String usuario,
                                             String senha) throws Exception{
        try{
           Class.forName(driver);
           if(ConnectionManager.conn == null){
              ConnectionManager.conn = DriverManager.getConnection(url,usuario,senha);
              ConnectionManager.conn.setAutoCommit(false);
           }
        } catch (Exception e) {
           JOptionPane.showMessageDialog(null, "Erro na conexão: "+e.getMessage());
        }
        return ConnectionManager.conn;
    }
```

❖ Especificação da Saída:

- **bean**: possui classes que representam os campos das tabelas utilizadas

```
public void insertIntoTBALUNO() throws Exception {

    conn = new ConnectionManager().retornaConexao();

    String sql = "insert into TBALUNO (";
    sql += "MATRICULA, ";
    sql += "NOME,   ";

    String sqlValues =" values (";
    sqlValues += "?,";
    sqlValues += "?";
    sqlValues+= ")";

    PreparedStatement pst = conn.prepareStatement(sql + sqlValues);
    try{
        pst.setString(1,this.getMATRICULA());
        pst.setString(2,this.getNOME());
        pst.execute();
```

❖ Especificação da Saída:

- **cursor**: classes que representam as consultas utilizadas pelos componentes de visualização de dados

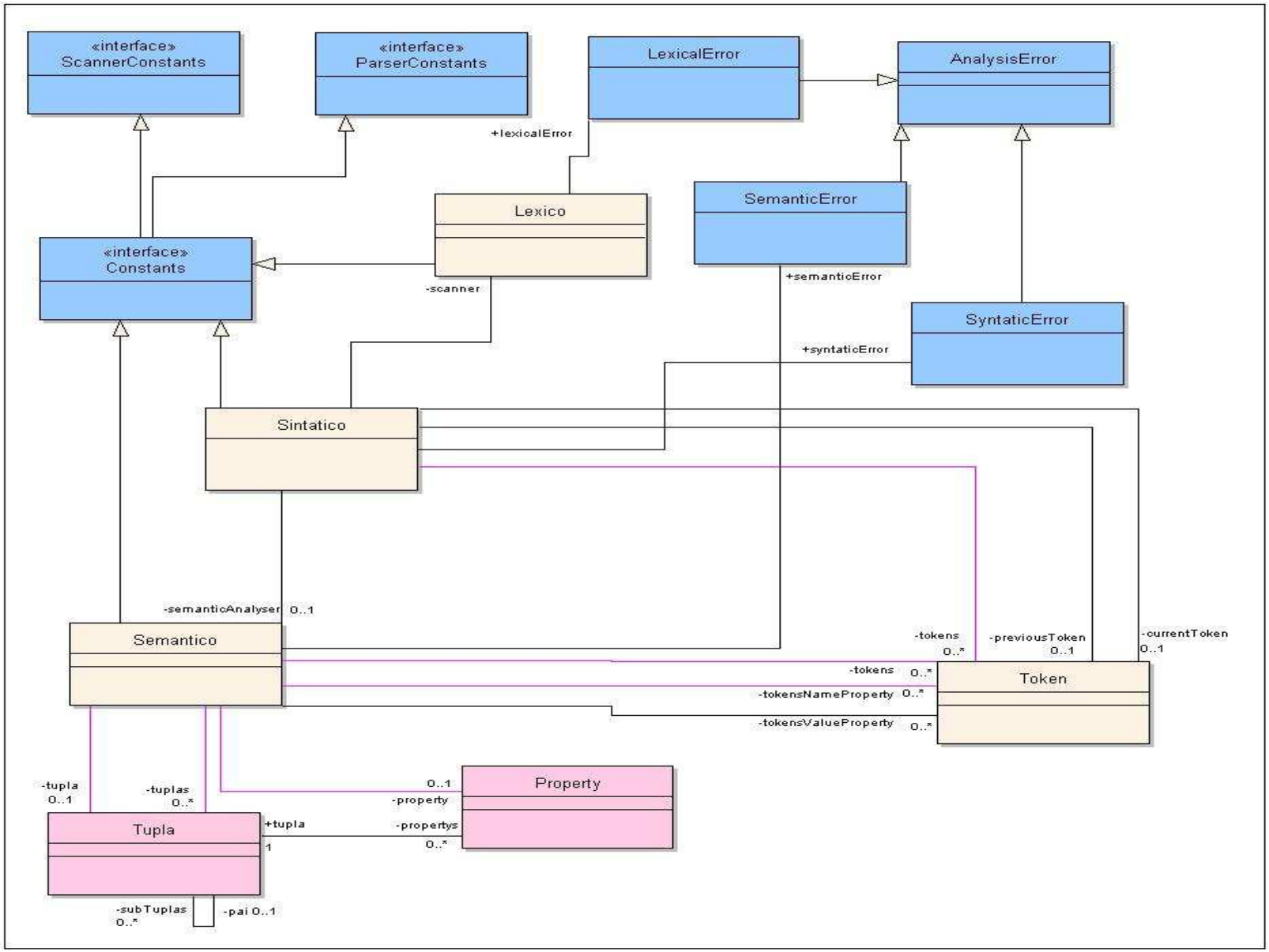
```
public class CursorTBALUNO extends ConnectionManager{

    public ArrayList getDadosTBALUNO() throws SQLException, Exception{

        Connection conn = this.retornaConexao();
        String sql = "Select * from TBALUNO";
        PreparedStatement preparedStatement = conn.prepareStatement(sql);

        ResultSet       resultado    = preparedStatement.executeQuery();
        HashMap    hmResultTBALUNO   = new HashMap();
        ArrayList arrayResultTBALUNO = new ArrayList();
        int cont = 0;

        while (resultado.next()){
            hmResultTBALUNO = new HashMap();
            hmResultTBALUNO.put("MATRICULA", resultado.getString("MATRICULA"));
            hmResultTBALUNO.put("NOME", resultado.getString("NOME"));
            arrayResultTBALUNO.add(cont++, hmResultTBALUNO);
        }
        preparedStatement.close();
        return arrayResultTBALUNO;
    }}
```

❖ Especificação da Saída:

- **controller**: classes de eventos da interface

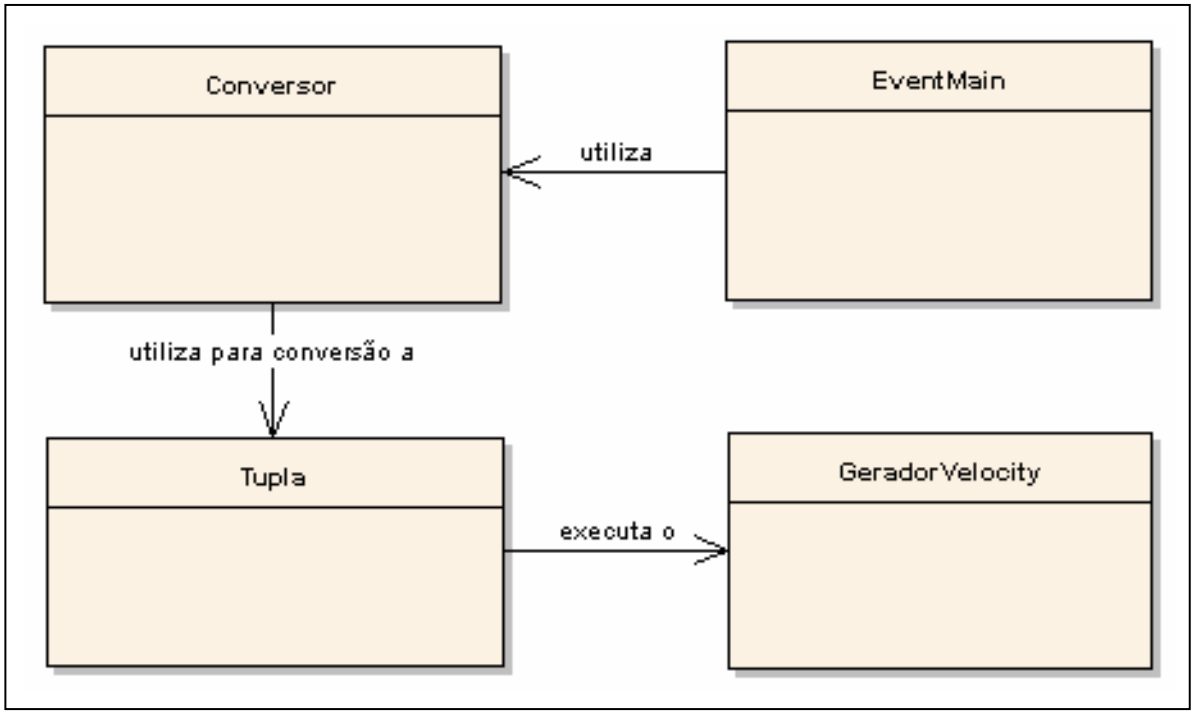
```
/***
 * Atribui valor da base aos campos que estão ligados somente ao Componente de Acesso TBALUNO
 */
  public void carregaValoresCamposTBALUNO() throws SQLException, Exception {

    CursorTBALUNO  TBALUNOCursor = new CursorTBALUNO();
    ArrayList   arrDadosTBALUNO = new ArrayList();
    arrDadosTBALUNO  = TBALUNOCursor.getDadosTBALUNO();

    if (arrDadosTBALUNO.size() > 0){

      HashMap mapdados = (HashMap) arrDadosTBALUNO.get(0);
      screen.getDBTextMatricula().setText(mapdados.get("MATRICULA") == null ? "" : mapdados.g
      screen.getDBEditNome().setText(mapdados.get("NOME") == null ? "" : mapdados.get("NOME")
      screen.getDBMemoDescricao().setText(mapdados.get("DESCRICAO") == null ? "" : mapdados.g
      if (mapdados.get("FLGREMETENTE").toString().trim().equalsIgnoreCase("S")){
        screen.getDBCheckBoxRemetente().setSelected(true);
      }else{
        screen.getDBCheckBoxRemetente().setSelected(false);
      }
      int value = new Integer(String.valueOf(mapdados.get("FLGESPECIAL"))).intValue();
      ((JRadioButton)screen.getDBRadioGroupStatus().getComponent(value)).setSelected(true);
    }
  }
```

❖ Análise de Entrada:



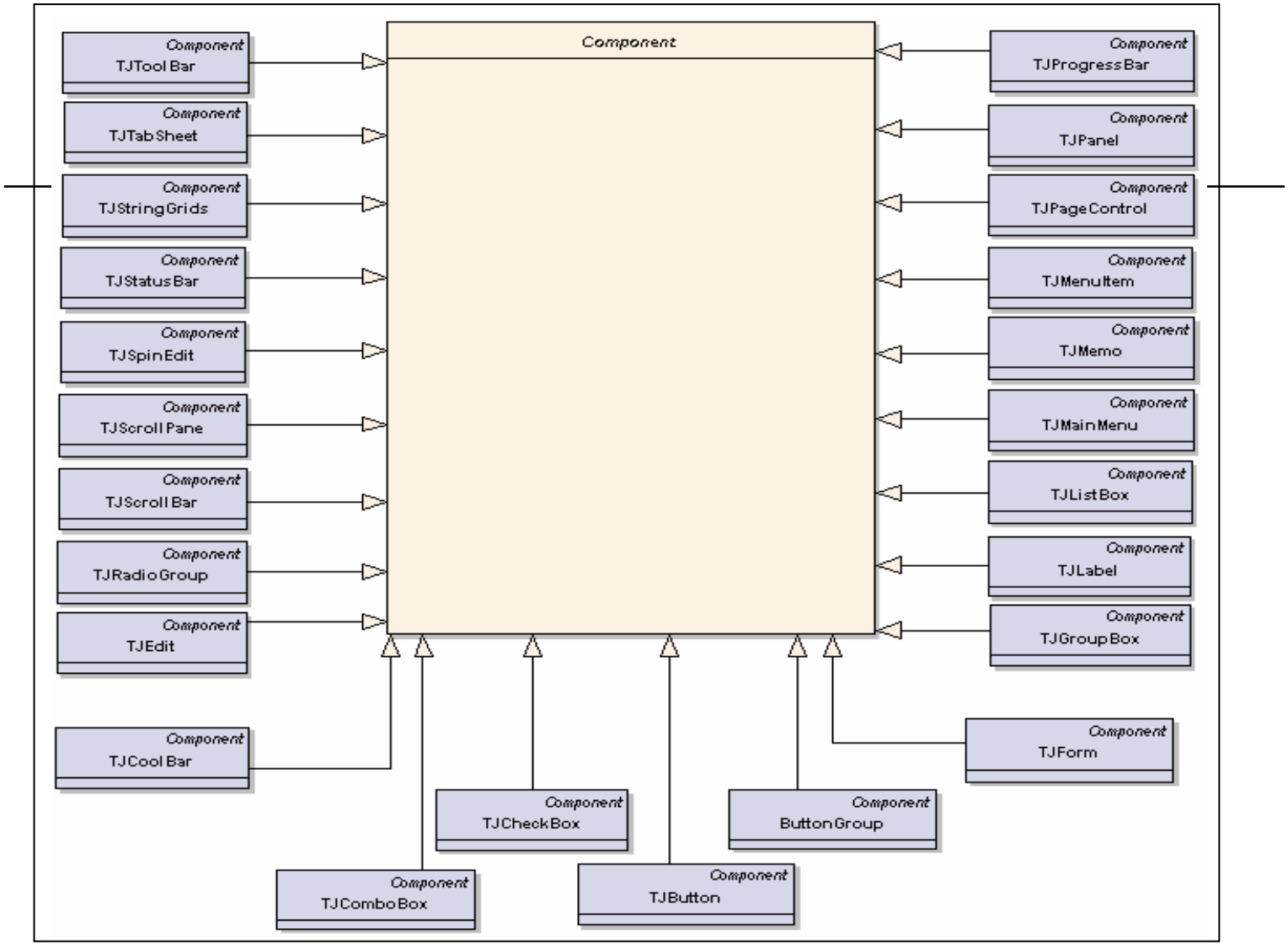
❖ Especificação – Diagrama de classes:

➢Para geração do código



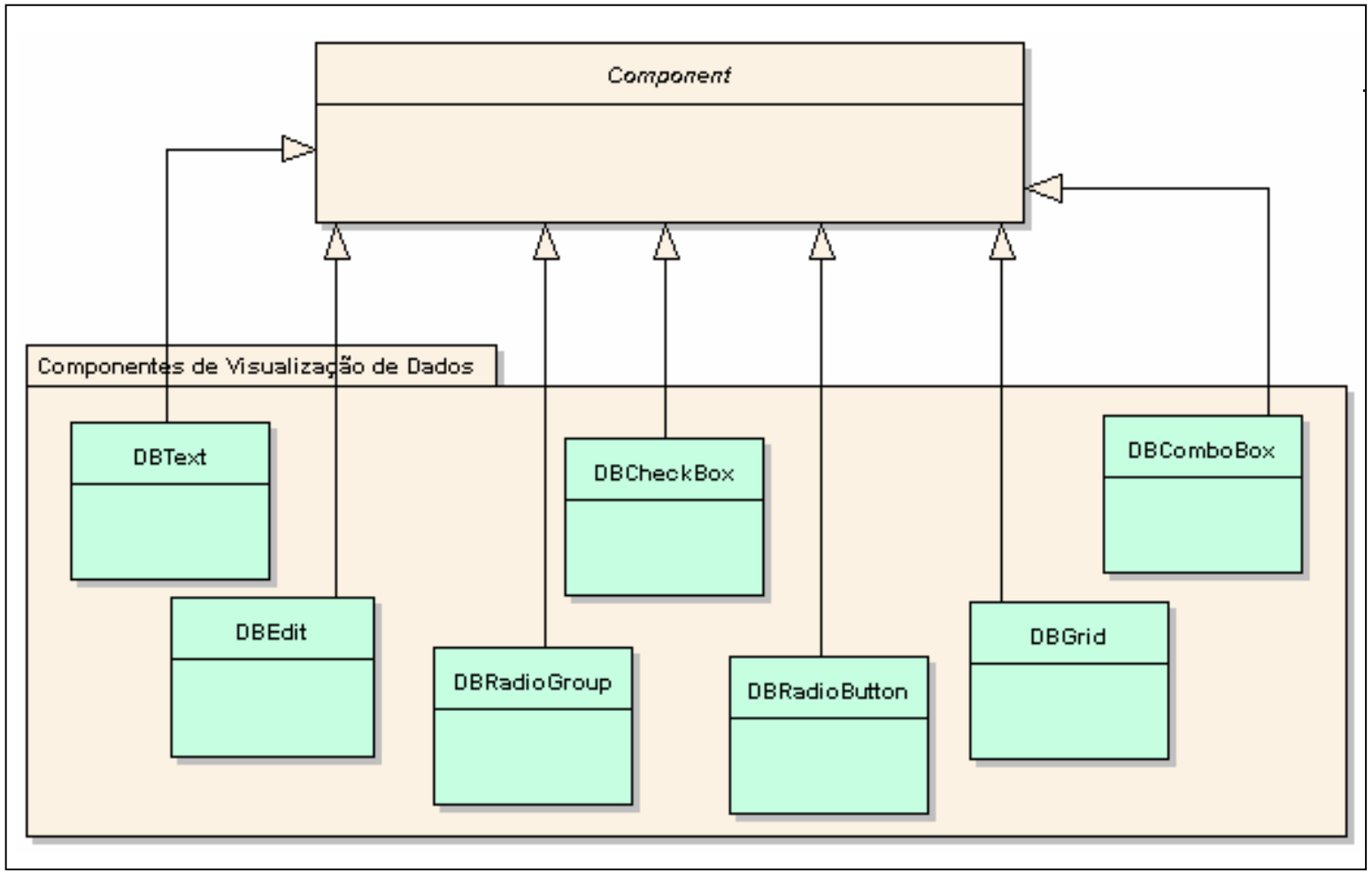
❖ Especificação – Diagrama de classes:

➢Para componentes de visualização



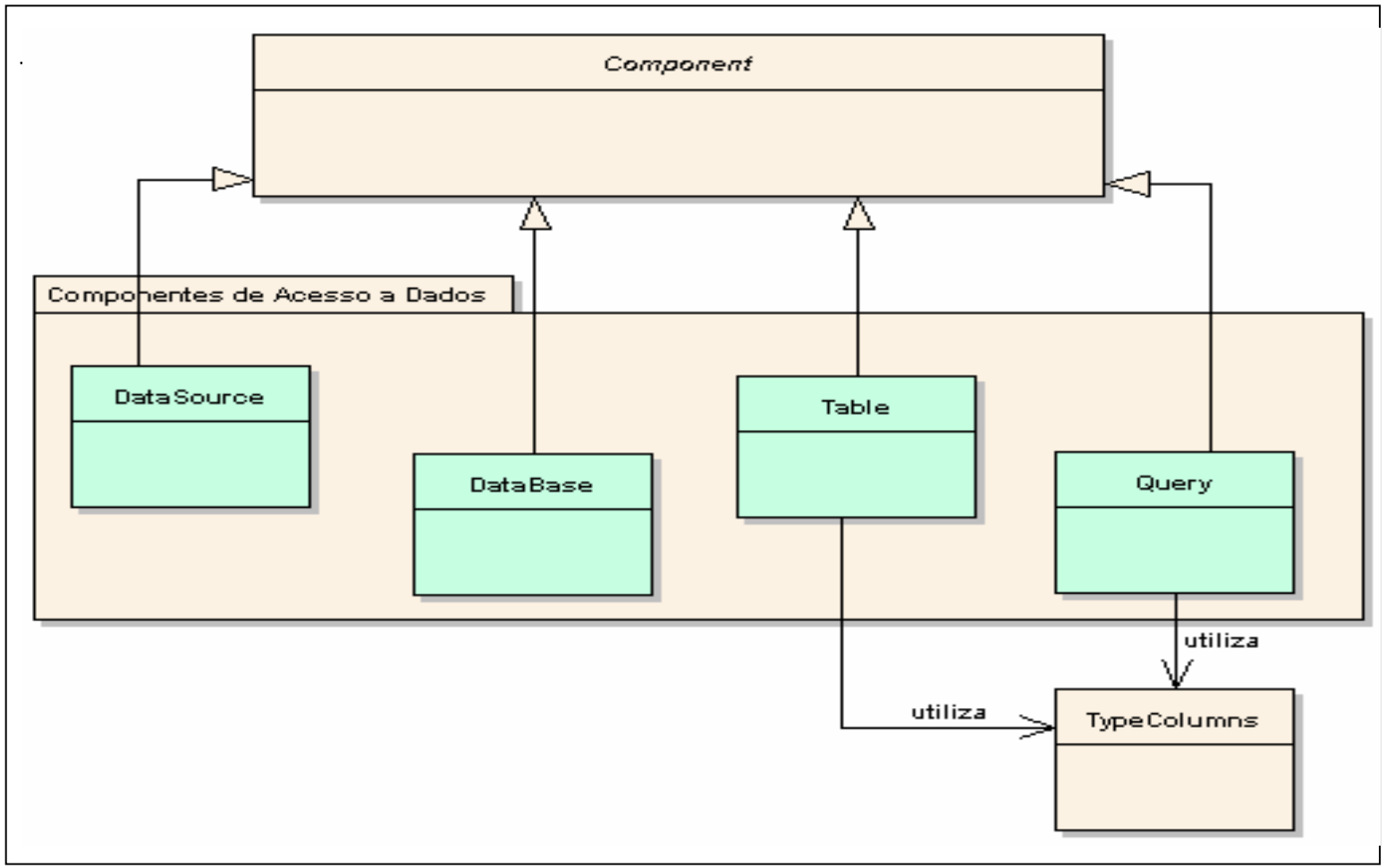
❖ Especificação – Diagrama de classes:

➢Para componentes de visualização de dados

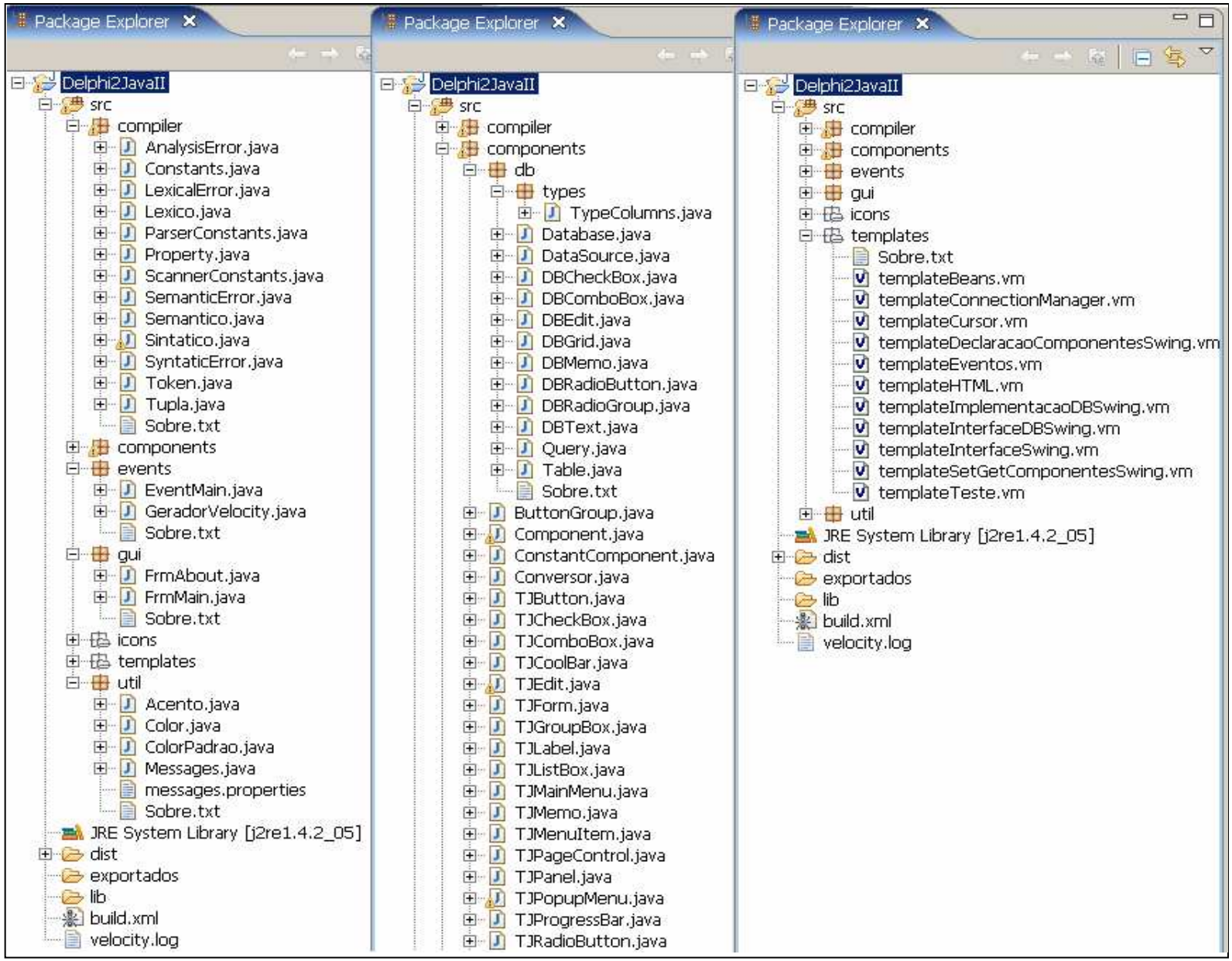


❖ Especificação – Diagrama de classes:

➢ **Para componentes de acesso a banco de dados**



❖ Implementação - Eclipse:



❖ Implementação:

```
public void getSubObjects(ArrayList subObjetos) {

    Conversor      conv  = Conversor.getInstance();
    //...

    if (subObjetos != null) {
        for (int i = 0; i < subObjetos.size(); i++) {

            Tupla tu = (Tupla) subObjetos.get(i);
            //-------------------------------------Componente Label
            if (tu.getType().equals("TLabel")) {
                addLabel(new TJLabel(tu));
                acionaEvento(tu);
            //-------------------------------------Componentes Banco de dados
            }else if (tu.getType().equals("TDatabase")){
                new Database(tu);
                setPossuiBD(true);
            //------------------------------------ Componentes DBComboBox
            }else if (tu.getType().equals("TDBComboBox")){
                addDBComboBox(new DBComboBox(tu));
                acionaEvento(tu);
                addAllComponentDB(new DBComboBox(tu));
                setPossuiComponentInterfaceDB(true);
            }
            //...
        }
    }
}
```

❖ Implementação:

```
public GeradorVelocity(TJForm form, FrmMain screen, String nomeUnitForm, String nomeUnitEve

    this.screen             = screen;
    this.nomeClasseLayout   = nomeUnitForm;
    this.nomeClasseEventos  = nomeUnitEventos;
    this.form               = form;

    //--Caminho de acesso aos arquivos de template;
    caminhoDiretorioTemplates.setProperty("file.resource.loader.path", "./src/templates/");

    //----------Saídas de Código a serem geradas por Template.
    geraSaidaInterface();  //--Saida da classe de layout Java.
    geraSaidaEventos();    //--Saída da classe de Eventos Java.

    //---Caso possua acesso a banco de dados , cria a classe de Conexao ao banco.
    if (Component.isPossuiBD()== true){
        geraClasseConexaoBD();
    }

    //--Caso possua objeto TABLE , cria o Bean e Cursor específico.
    if (Component.isPossuiTable()== true){
        for (int i=0 ;i < Component.getArrayTable().size();i++){
            geraClasseBean((Table)Component.getArrayTable().get(i));
            geraClasseCursor((Table)Component.getArrayTable().get(i));
        }
    }

    //--Caso possua objeto query cria o Cursor especifico
    if (Component.isPossuiQuery() == true){
        for (int j = 0; j < Component.getArrayQuery().size(); j++){
            geraClasseCursor((Query) Component.getArrayQuery().get(j));
        }
    }

    //--Saida HTML de Teste
    geraSaidaHTML();

}
```

❖ Implementação:

```
public void geraSaidaInterface() throws Exception{

  try{

    //--inicializando o velocity
    VelocityEngine vInterf = new VelocityEngine();
    vInterf.init(caminhoDiretorioTemplates);

    //--criando o contexto que liga o Java ao Template
    //----------o contexto é usado para interagir com o arquivo de templates,
    //como por exemplo setando as variáveis e valores usadas no mesmo.
    VelocityContext context = new VelocityContext();

    //--definindo o template
    Template tInterf = vInterf.getTemplate("templateInterfaceSwing.vm");

    //--variaveis que serão acessada pelo template
    //--Adicionando as variaveis para o contexto.
    //  diz ao Velocity que quando ele encontrar, no arquivo de template,
    //  ex:uma variável chamada "$arrayButtons" é para usar os métodos e
    //  propriedades do objeto "arrayButtons" (o qual representa nossa ArrayList, internamente)

    context.put("nomeClasseLayout", this.nomeClasseLayout);   //Nome da classe de Interface
    context.put("nomeClasseEventos", this.nomeClasseEventos); //Nome da classe de Eventos

    context.put("TForm",  this.form);                         //Carrega o Objeto   :JDesktopPane
    context.put("arrayButton",Component.getArrayButtons());  //Carrega os Objetos :JButton
    context.put("arrayLabel",Component.getArrayLabels());    //Carrega os Objetos :JLabel
    context.put("arrayEdit",Component.getArrayEdits());      //Carrega os Objetos :JFormattedTextField o
    context.put("arrayCombo",Component.getArrayComboBox());  //Carrega os Objetos :JComboBox
    context.put("arrayDBText",Component.getArrayDBText());   //Carrega os Objetos :JDbText = >JLabel
    context.put("arrayDBEdit",Component.getArrayDBEdit());   //Carrega os Objetos :JDbEdit = >JTextField
    context.put("arrayDBMemo",Component.getArrayDBMemo());   //Carrega os Objetos :JDbMemo = >JTextArea
```

❖ Implementação:

```
#*Aqui fica encontra-se a estrutura dos componentes DB.*#
          #foreach ($TDBText in $arrayDBText)                              #*----Componente DBText-----
          $!{TDBText.getClassePai()}.add($!{TDBText.getName()}, javax.swing.JLayeredPane.DEFAULT_I
          $!{TDBText.getName()}.setBackground(new java.awt.Color($!{TDBText.setarBackground()}));
          $!{TDBText.getName()}.setBounds($!{TDBText.getLeft()}, $!{TDBText.getTop()}, $!{TDBText.
          $!{TDBText.getName()}.setFont(new java.awt.Font($!{TDBText.getFontJava()}));
          $!{TDBText.getName()}.setForeground(new java.awt.Color($!{TDBText.getForeground()}));
          $!{TDBText.getName()}.setOpaque($!{TDBText.getOpaque()});
          $!{TDBText.getName()}.setVisible($!{TDBText.getVisible()});
          #if ($!TDBText.getEnabled())
          $!{TDBText.getName()}.setEnabled($!{TDBText.getEnabled()});
          #end
          #end
#foreach ($TDBEdit in $arrayDBEdit)                                  #*----Componentes DBEdit------*#
          $!{TDBEdit.getClassePai()}.add($!{TDBEdit.getName()}, javax.swing.JLayeredPane.DEFAULT_I
          $!{TDBEdit.getName()}.setBackground(new java.awt.Color($!{TDBEdit.setarBackground()}));
          $!{TDBEdit.getName()}.setBounds($!{TDBEdit.getLeft()}, $!{TDBEdit.getTop()}, $!{TDBEdit.
          $!{TDBEdit.getName()}.setVisible($!{TDBEdit.getVisible()});
          #if ($!{TDBEdit.getFontJava()})
$!{TDBEdit.getName()}.setFont(new java.awt.Font($!{TDBEdit.getFontJava()}));
          $!{TDBEdit.getName()}.setForeground(new java.awt.Color($!{TDBEdit.getForeground()}));
          #end
          #if ($!TDBEdit.getEnabled())
 $!{TDBEdit.getName()}.setEnabled($!{TDBEdit.getEnabled()});
          #end
#end
#foreach ($TDBMemo in $arrayDBMemo)                          #*----Componente DBMemo ------*#
          $!{TDBMemo.getName()}.setBackground(new java.awt.Color($!{TDBMemo.setarBackground()}));
          $!{TDBMemo.getName()}.setBounds($!{TDBMemo.getLeft()}, $!{TDBMemo.getTop()}, $!{TDBMemo.
          $!{TDBMemo.getName()}.setVisible($!{TDBMemo.getVisible()});
          $!{TDBMemo.getName()}.setBorder(new javax.swing.border.EtchedBorder());
          #if ($!TDBMemo.getFontJava())
$!{TDBMemo.getName()}.setFont(new java.awt.Font($!{TDBMemo.getFontJava()}));
          $!{TDBMemo.getName()}.setForeground(new java.awt.Color($!{TDBMemo.getForeground()}));
          #end
          #if ($!TDBMemo.getEnabled())
$!{TDBMemo.getName()}.setEnabled($!{TDBMemo.getEnabled()});
          #end
```

Resultados e Discussão:

CARACTERÍSTICAS	Delphi2Java-II (FONSECA,2005)	DelphiToWeb	Delphi2Java-II
conversão de componentes de visualização	27	22	22
componentes de visualização de dados	-	-	7
componentes de acesso a banco de dados	-	-	4
uso de analisadores (léxico, sintático e semântico)	-	sim	sim
uso de templates para geração de código	-	-	sim
geração de código de saída utilizando o padrão MVC	-	-	sim

Conclusão:

- objetivos atingidos
- qualidade na utilização de analisadores para leitura dos arquivos .DFM
- flexibilidade na criação de componentes com a utilização de templates
- maior facilidade com a utilização de um padrão de projeto para as classes geradas, melhorando o processo de desenvolvimento

Extensões:

- efetuar a conversão de novos componentes de banco de dados, entre eles o TDBNavigator
- efetuar a análise e conversão da Unit correspondente ao arquivo .DFM
- adicionar a possibilidade de seleção do tipo de saída desejada em função dos *templates* definidos, por exemplo HTML, XML.

Apresentação prática: