Curso de Java 8 para Web

Professor
Antonio Benedito Coimbra Sampaio Jr

abc | Treinamentos

www.abctreinamentos.com.br

Terceira

Disciplina

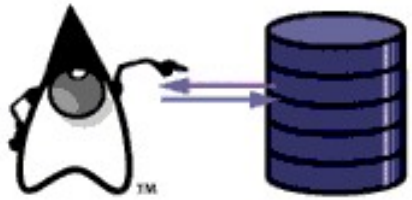
JEE - Persistência de Dados com JDBC e Hibernate

- **UNIDADE 1:** Arquitetura JEE
- **UNIDADE 2:** Introdução a Banco de Dados com Oracle
- **UNIDADE 3: Persistência de Dados com JDBC**
- **UNIDADE 4:** Framework Hibernate
- **UNIDADE 5:** Introdução ao JPA

UNIDADE 3

PERSISTENCIA DE DADOS COM JDBC

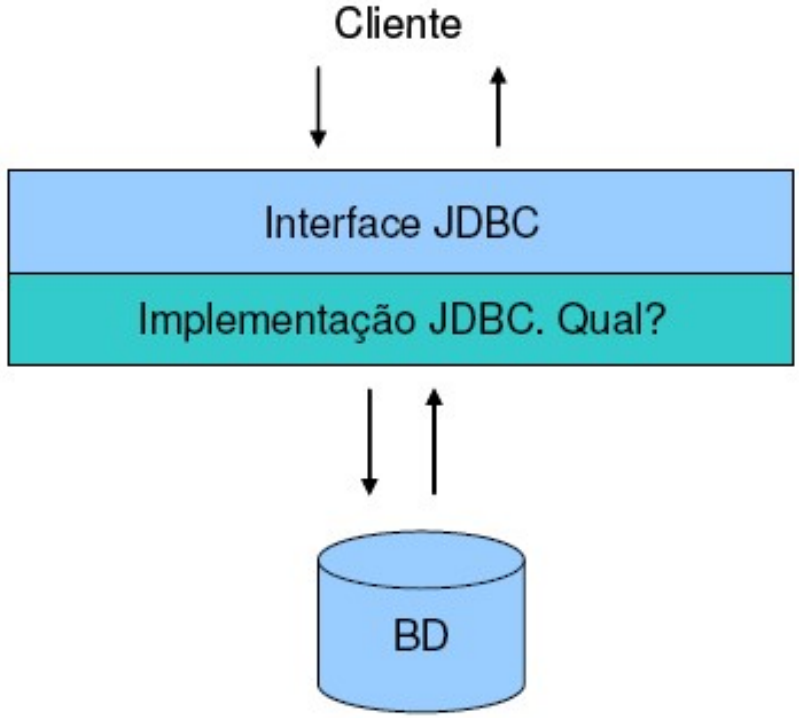
API JDBC



API JDBC

- A API JDBC (*Java DataBase Conectivity*) fornece um conjunto de classes e interfaces para manipular as Bases de Dados.

- A API JDBC é uma camada de abstração que permite a uma aplicação Java utilizar uma interface padrão para acessar um banco de dados relacional através da linguagem SQL.



API JDBC

- Abaixo, o exemplo de uma aplicação corporativa JEE que faz uso da API JDBC para ter acesso a um SGDB.

API JDBC

- **A Versão atual do JDBC é a 4.2.**

- É composta pelos pacotes **java.sql** e **javax.sql** já incluídos no Java 8.

- O pacote **javax.sql** contém outras classes e pacotes que permitem o uso de conexões JDBC de forma mais eficiente e portável.

Classe javax.sql.DataSource

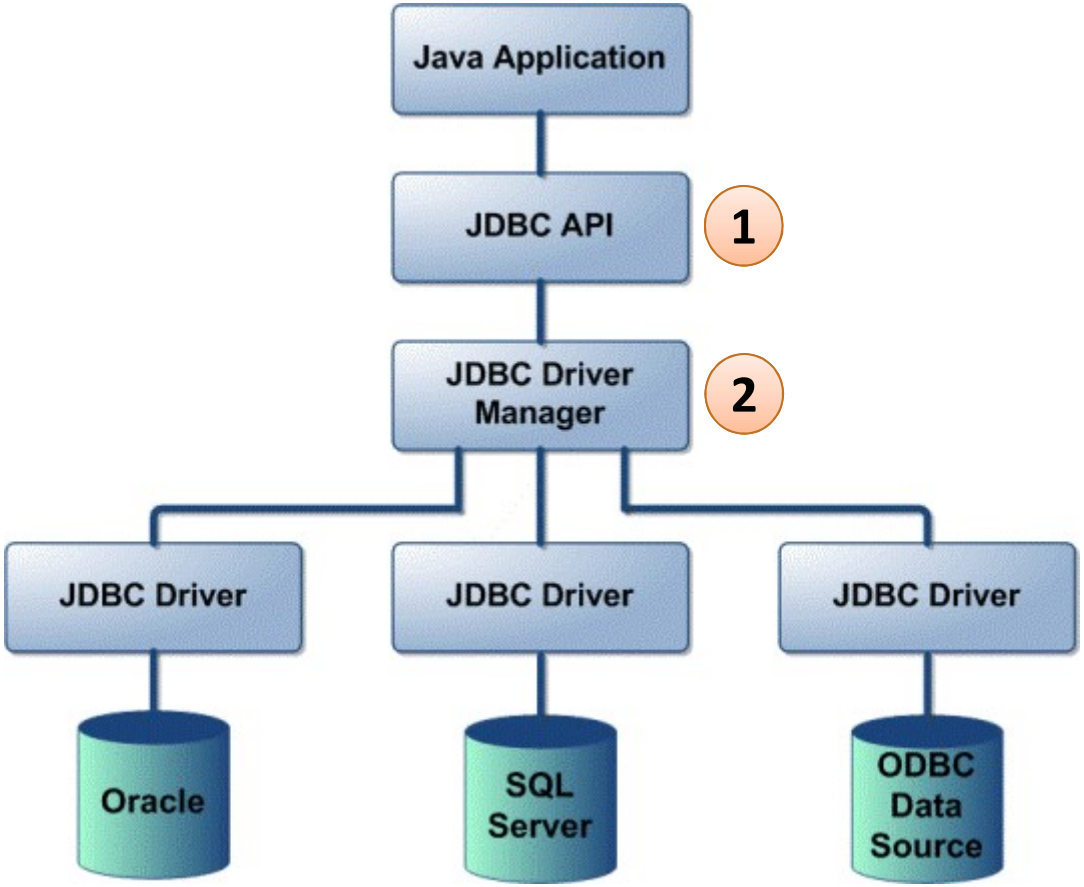
- Obtém uma conexão a partir de um sistema de nomes JNDI;
- *DataSource* é uma alternativa mais eficiente que *DriverManager*. Possui pool de conexões embutido.

Classe javax.sql.RowSet

- Extensão de *ResultSet*;
- Permite manipulação customizada de *ResultSet.*

Arquitetura JDBC

- **JDBC consiste em duas partes:**
  - (1) API JDBC, puramente escrita em Java;
  - (2) Gerenciador de Driver JDBC, o qual se comunica com os drivers dos fabricantes de Bancos de Dados.



Driver JDBC

- É o componente de software utilizado para que uma aplicação escrita em Java tenha acesso a um Banco de Dados.

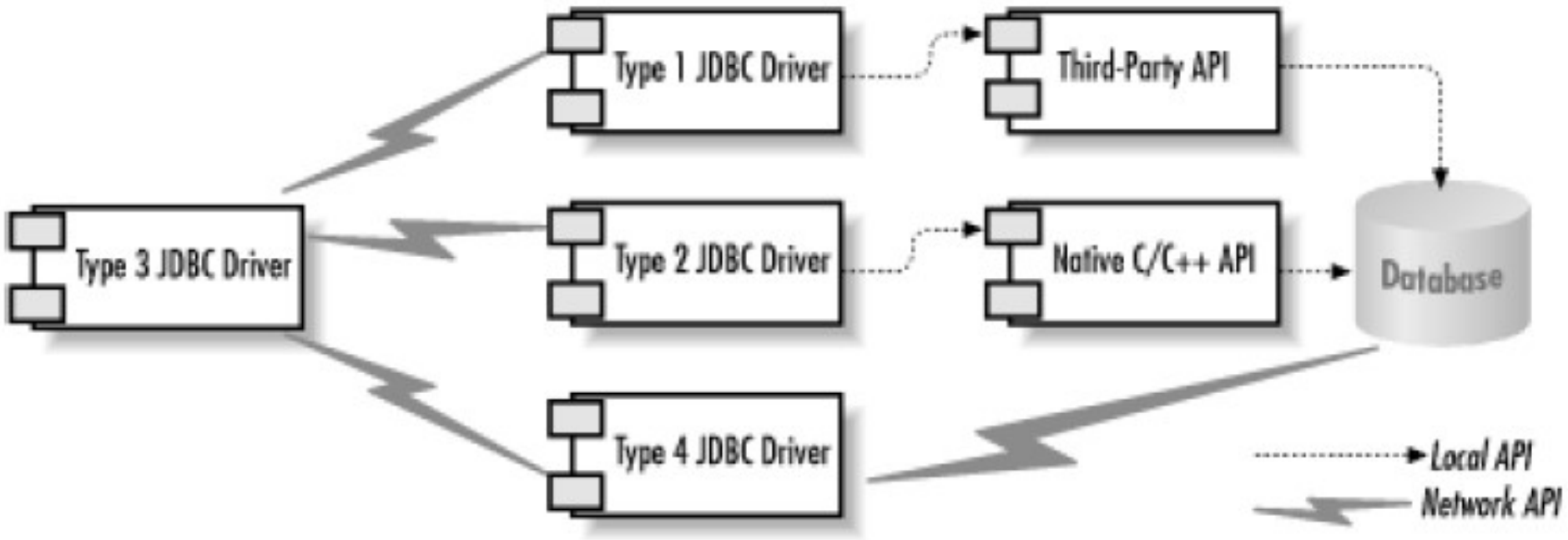
- Todos os principais bancos de dados do mercado possuem os seus drivers JDBC.

- O driver JDBC atua como tradutor entre uma aplicação Java e o Bando de Dados, implementando um protocolo de comunicação para a transferência de consultas e resultados.

Tipos de Drivers:

- **TIPO 1 – *JDBC-ODBC***
- **TIPO 2 – *Native-API partly Java Technology-enabled driver***
- **TIPO 3 – *Pure Java Driver for Database Middleware***
- **TIPO 4 – *Direct-to-Database Pure Java Driver***

Tipos de Drivers JDBC



Tipos de Drivers JDBC

- **TIPO 1 - JDBC-ODBC**

  - Utilizado para fazer a ligação (“ponte”) entre o JDBC e um driver ODBC.

- **TIPO 2 – *Native-API partly Java Technology-enabled driver***

  - Converte chamadas JDBC em chamadas internas da API do cliente do banco de dados.

- **TIPO 3 – *Pure Java Driver for Database Middleware***

  - A API do JDBC acessa uma aplicação intermediária (middleware) encarregada de traduzir chamadas JDBC e enviá-las ao banco de dados.

- **TIPO 4 – *Direct-to-Database Pure Java Driver***

  - Este driver converte as chamadas JDBC diretamente para o protocolo nativo do SGBD.

Vantagens JDBC

- Uma aplicação Java utiliza uma API JDBC única que independe do banco de dados ou driver que estiver sendo utilizado.

- Os drivers para conexão e acesso aos principais bancos de dados são fornecidos pelos seus próprios fabricantes.

- O desenvolvedor precisa apenas conhecer a API JDBC e utilizar o driver adequado.

Exercícios

- 1) [CESPE - 2008 - SERPRO] No BrOffice.org a configuração da conexão com o MySQL precisa de um conector para poder mover dados entre o OpenOffice.org e o MySQL. Os drivers para esse efeito podem ser o ODBC (Connector/ODBC) e o JDBC (Connector/J). O JDBC pode ser usado no Linux, Windows mas não no MaC OS.x.

  a) Certo b) Errado

- 2) [FCC - 2008 - TRT] A utilização de JDBC, em um programa Java, inicia com a indicação do pacote que contém a JDBC API pela declaração:

  a) import java.awt.*;

  b) import java.util.*;

  c) import java.sql.*;

  d) import java.swing.*;

  e) import java.jdbc.*;

Exercícios

- 1) [CESPE - 2008 - SERPRO] No BrOffice.org a configuração da conexão com o MySQL precisa de um conector para poder mover dados entre o OpenOffice.org e o MySQL. Os drivers para esse efeito podem ser o ODBC (Connector/ODBC) e o JDBC (Connector/J). O JDBC pode ser usado no Linux, Windows mas não no MaC OS.x.

  a) Certo **b) Errado**

- 2) [FCC - 2008 - TRT] A utilização de JDBC, em um programa Java, inicia com a indicação do pacote que contém a JDBC API pela declaração:

  a) import java.awt.*;

  b) import java.util.*;

  c) import java.sql.*;

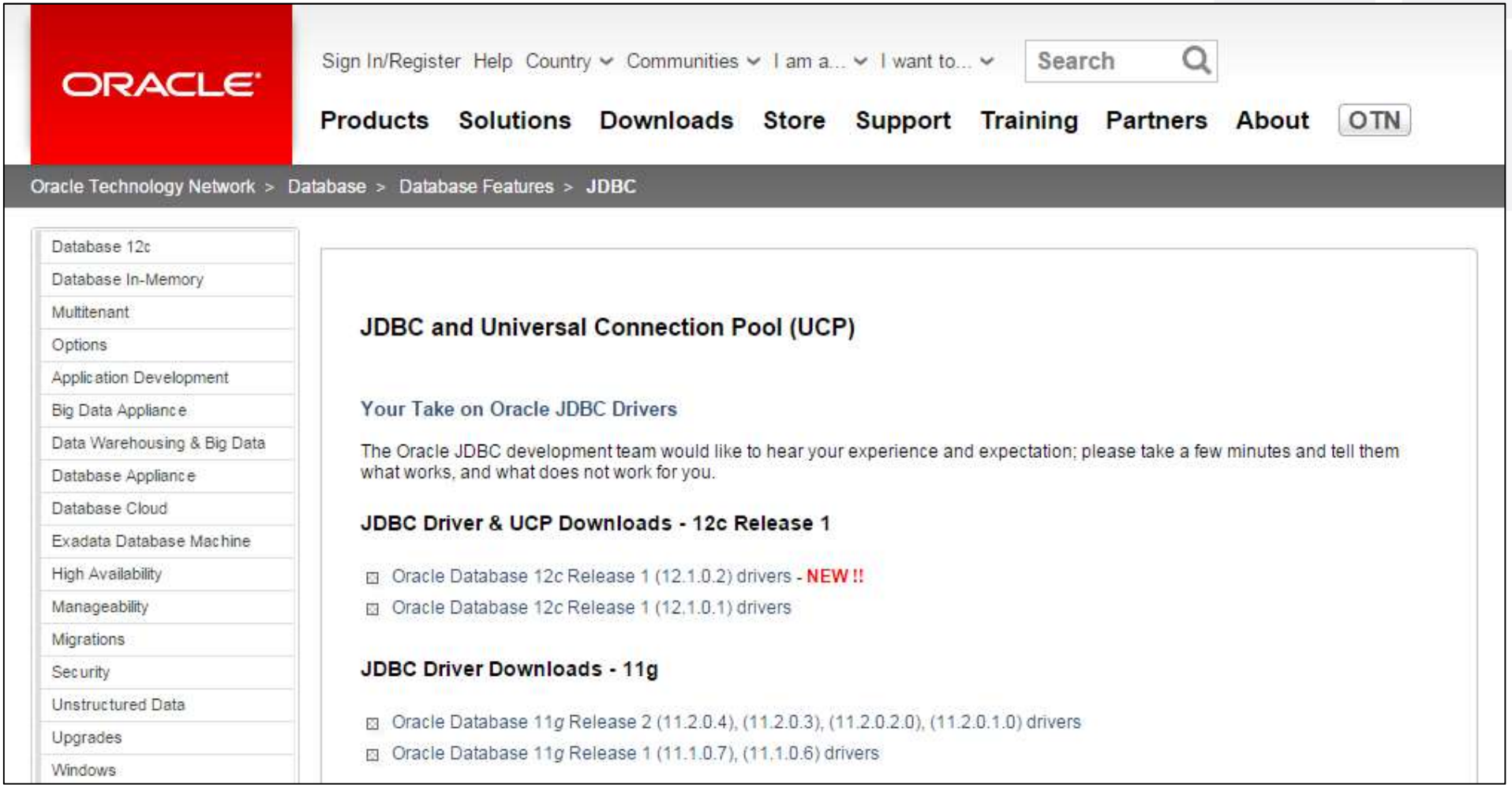
  d) import java.swing.*;

  e) import java.jdbc.*;

Driver SGBD ORACLE

Obtenção do Driver JDBC

- A ORACLE disponibiliza um site com todos os seus drivers JDBC (Tipo 4) para download gratuito.

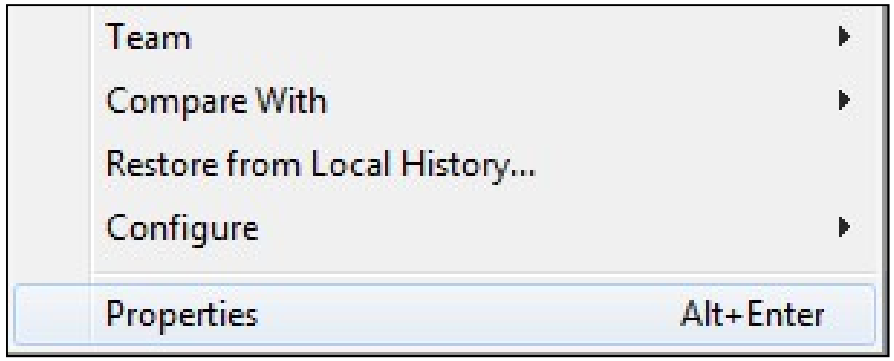


http://www.oracle.com/technetwork/database/features/jdbc/index-091264.html

Configuração do Driver JDBC no Eclipse

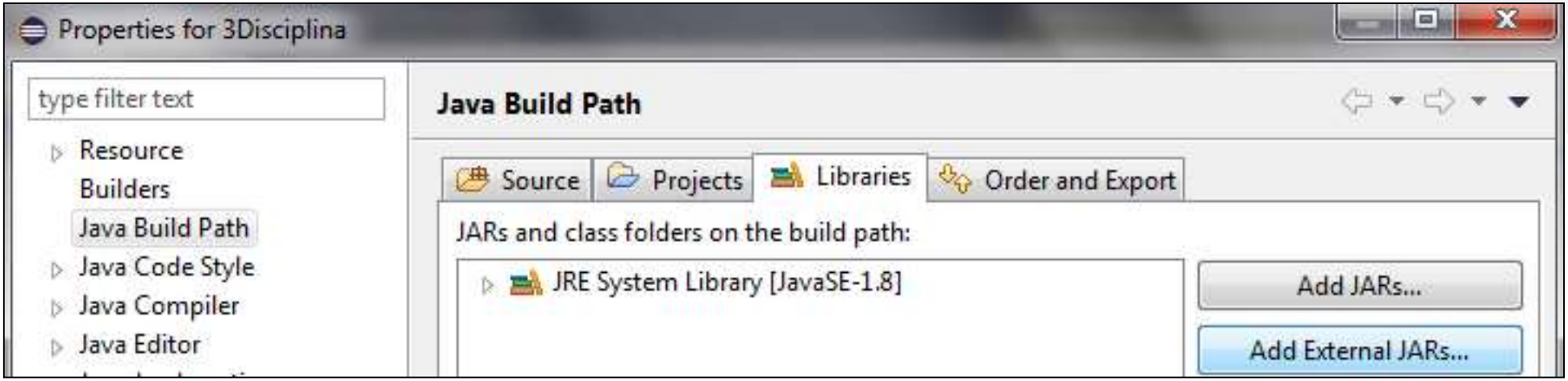
- Será utilizado o driver (**ojdbc6.jar**) disponibilizado para a versão do SGBD Oracle 11g Release 2.

- No Eclipse, criar um novo Projeto Java com o nome de **"3Disciplina".** Posteriormente, clicar com o botão direito no projeto e escolher a opção "**Properties**".

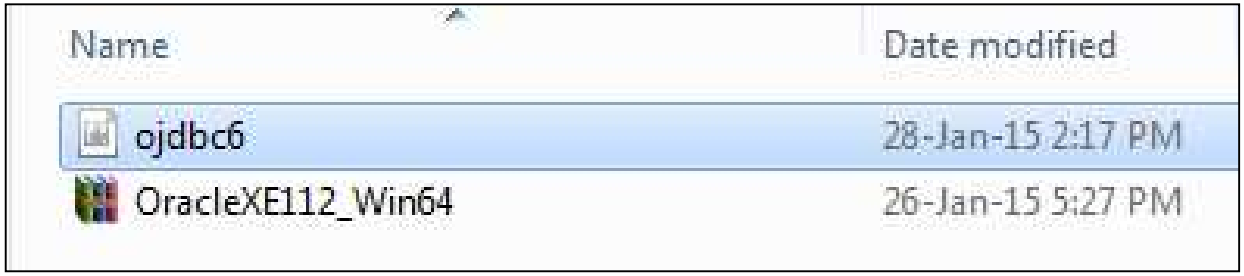


Configuração do Driver JDBC no Eclipse

- Selecionar a aba "**Libraries**" e clicar no botão "**Add External Jars**".

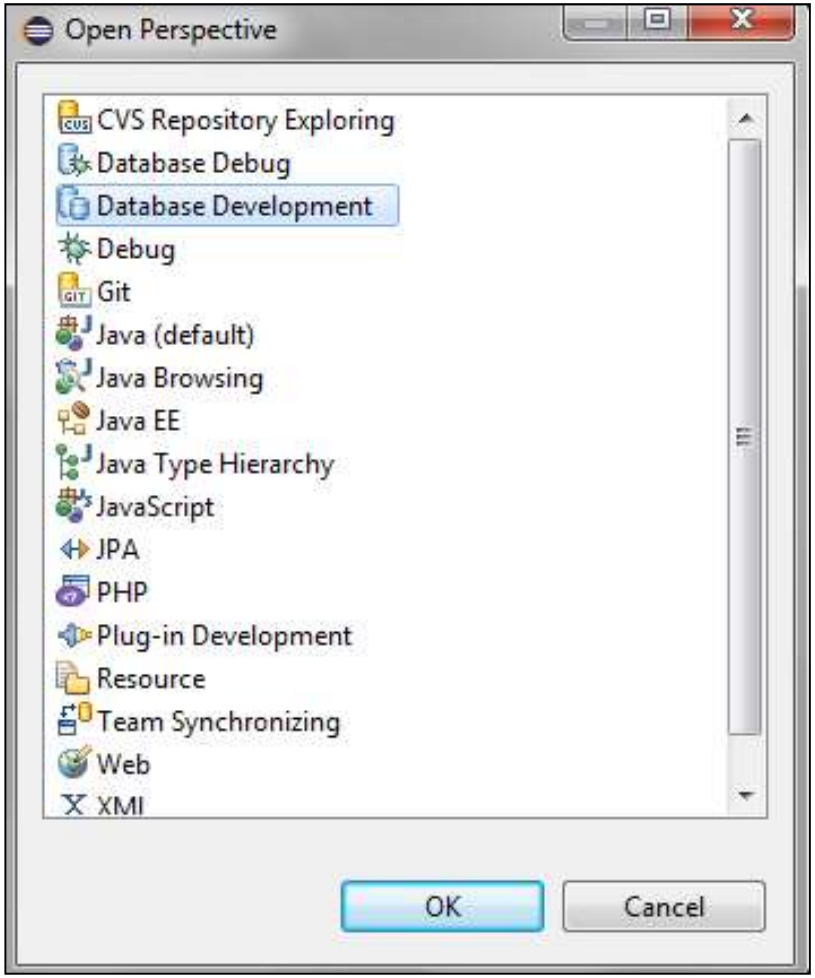


- Selecionar o arquivo "**ojdbc6.jar**" e clicar no botão "**OK**".



Configuração do SGBD Oracle no Eclipse

- Selecionar a perspectiva "**DataBase Development**".

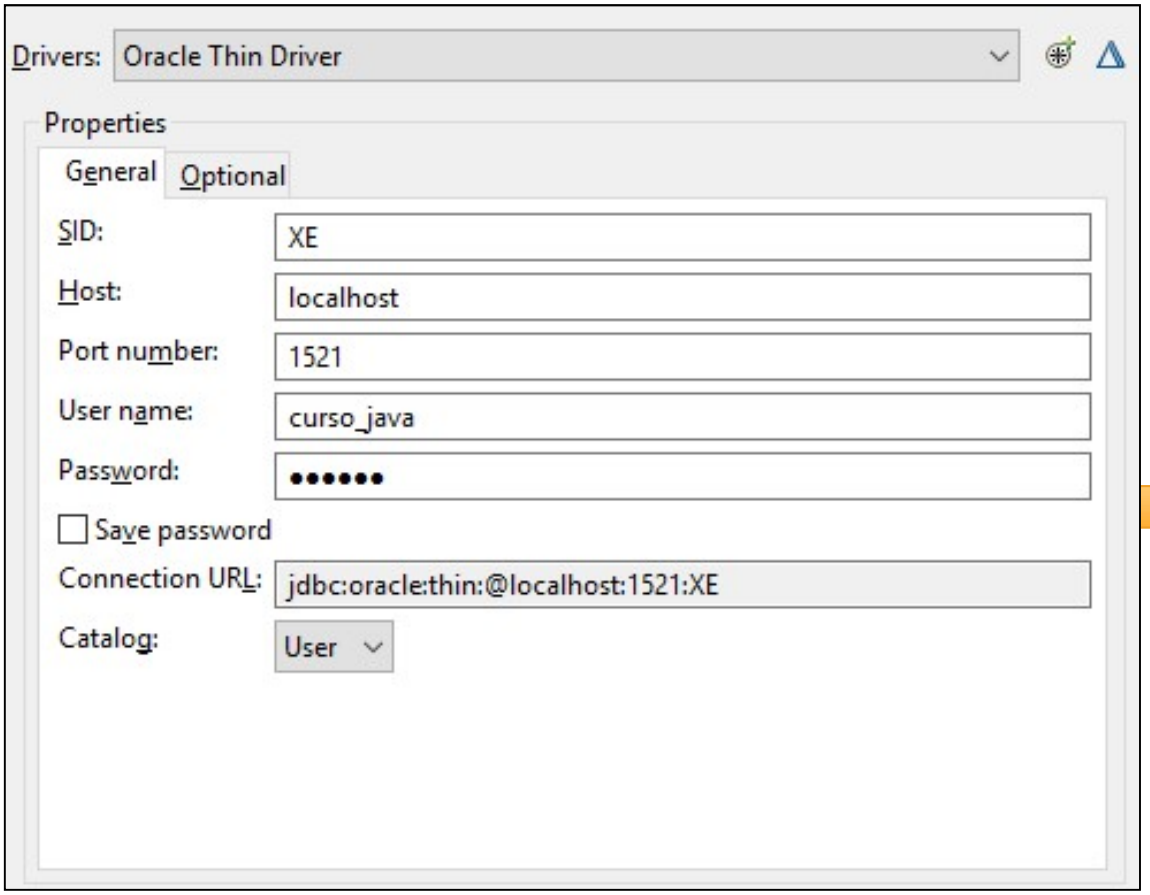


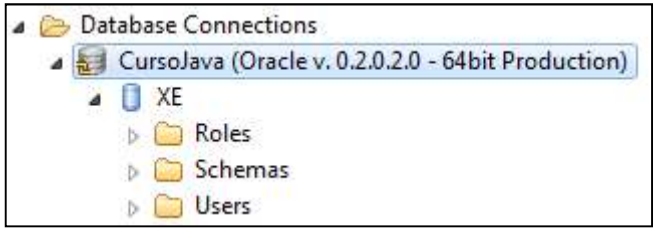
Configuração do SGBD Oracle no Eclipse

- Selecionar a opção "**DataBase Connections ⇒ New ⇒ Oracle**". Adicionar o driver (**ojdbc6.jar**) e escolher a opção "Oracle Thin Driver".

- Informar:
  - SID: **XE**
  - Host: **localhost**
  - User Name: **curso_java**
  - Passaword: **schema**

Configuração do SGBD Oracle no Eclipse





Exercícios

1) Acessar as Tabelas **Cliente** e **Curso** no esquema '**curso_java**' via Eclipse.

2) Realizar a operação de consulta nessas duas tabelas via Eclipse.

Principais Classes e Interfaces JDBC

Classes e Interfaces JDBC

- As principais Classes e Interfaces JDBC estão listadas abaixo:

  - **public class DriverManager**
  - **public interface Connection**
  - **public interface Statement**
  - **public interface PreparedStatement**
  - **public interface CallableStatement**
  - **public interface ResultSet**
  - **public class SQLException**

DRIVERMANAGER

- É a classe utilizada para estabelecer uma conexão entre o driver apropriado e o Banco de Dados.

- O serviço de encontrar o driver certo é delegado para um controlador de drivers (**DriverManager**).

- **Método *getConnection(...)***

```
public static Connection getConnection (String url)
       throws SQLException
//Inicia uma conexão ao banco de dados e retorna um
//objeto Connection.
```

- O padrão da **url** para o driver Oracle é:

```
jdbc:oracle:thin:@ip:1521:schema
```

DRIVERMANAGER

- **Exemplo**

```
DriverManager.getConnection
            ("jdbc:oracle:thin:@localhost:1521:XE");
```

- O ***DriverManager*** procura por algum de seus Drivers que aceite essa URL como parâmetro.

- O sistema ainda não consegue descobrir qual implementação do JDBC deve ser usado para a URL mencionada.

- O primeiro passo é adicionar o driver ORACLE JDBC (**ojdbc6.jar**) no projeto Java relacionado.

CONNECTION

- É a interface utilizada para representar uma conexão com um Banco de Dados específico. Oferece as seguintes funcionalidades:
  - Executar os comandos SQL (***select, insert, update, delete, etc***.) via métodos (***prepareStatement(...)***, etc.) e retornar os seus resultados;
  - Tratar das transações com o BD (***commit, rollback***)
  - Obter informações de metadados do BD (***getMetaData())***

Informações do Banco de Dados	*Informações Físicas do Banco de Dados*	*Esquemas do Banco de Dados*

CONNECTION

- **Principais Métodos**

```
public Statement createStatement() throws SQLException
//Cria um objeto Statement para enviar comandos
//SQL ao Banco de Dados

public void commit() throws SQLException
//Confirma a transação.

public void rollback() throws SQLException
//Cancela a transação.

public void close() throws SQLException
//Fecha a conexão atual.
```

STATEMENT

- É a interface utilizada para representar uma instrução SQL e retornar os resultados produzidos.

- **Principais Métodos**

```
public ResultSet executeQuery (String sql)
                                        throws SQLException
//Executa a consulta SQL passada em SQL e
//retorna o ResultSet com o resultado da consulta.

public int executeUpdate(String sql) throws SQLException
//Executa as instruções SQL INSERT, UPDATE ou DELETE.

public void cancel() throws SQLException
//Cancela a execução de uma instrução SQL.
```

- **Exemplo**

```
Statement stm = connection.createStatement();
```

PREPARED STATEMENT

- É a interface utilizada para representar uma instrução SQL pré-compilada e retornar os resultados produzidos.

- É ideal para ser executada várias vezes consecutivas, bem como receber a passagem de parâmetros.

- **Principais Métodos**

```
public ResultSet executeQuery (String sql)
                                        throws SQLException
//Executa a consulta SQL passada em SQL e
//retorna o ResultSet com o resultado da consulta.

public int executeUpdate(String sql) throws SQLException
//Executa as instruções SQL INSERT, UPDATE ou DELETE.

public setInt(int indice, int valor)
//Define o valor de um parâmetro do tipo inteiro
```

PREPARED STATEMENT

- O código abaixo exemplifica o uso do método **prepareStatement(...)** para o recebimento de instrução SQL, cujos parâmetros são passados como argumentos posteriormente.

- **Exemplo**

```
PreparedStatement stmt = con.prepareStatement(
"insert into cliente(cpf,nome,email)
values (?,?,?)"); // instrução SQL

// passagem dos argumentos
stmt.setInt(1, 573618);
stmt.setString(2, "Antonio Sampaio Jr");
stmt.setString(3, "assoftbel@gmail.com");

//uma chamada ao método executeUpdate() para
//executar a instrução SQL.
stmt.executeUpdate();
```

CALLABLE STATEMENT

- É a interface utilizada para executar *stored procedures* SQL.

- **Métodos**

```
public CallableStatement prepareCall
            (String storedProcedure) throws SQLException
//Executa a stored procedured armazenada no BD.

public setString(String paramName, String x)
//Define o valor de um parâmetro do tipo String
```

CALLABLE STATEMENT

- **Exemplo**

```
CallableStatment cstmt =
            connection.prepareCall("{call sp_xxx(?,?)}");
cstmt.registerOutParameter(2, Types.FLOAT);
cstmt.setInt(1, accountID);
cstmt.setFloat(2, 2343.23);
cstmt.execute();
out.println("Novo valor:" + cstmt.getFloat(2));
```

- Esta procedure possui dois parâmetros. O primeiro é só de entrada, e para este é passado o conteúdo da variável **accountID**. O segundo é do tipo IN OUT. Para este, é um passado um número (2343.23) como entrada e o seu retorno, após execução, é impresso na tela. Note-se que o parâmetro de retorno foi registrado juntamente com seu tipo de dados.

RESULT SET

- É a interface que representa o resultado de uma consulta SQL em um Banco de Dados.

- Fornece acesso aos dados dos registros retornados após a consulta.

- **Métodos**

```
public boolean next()
//Move para o próximo registro do objeto ResultSet

getInt(int col) ou (String col)
//Retorna o valor inteiro da coluna informada

getString(int col) ou (String col)
//Retorna o valor String da coluna informada
```

RESULT SET

- **Exemplo**

```
consulta = "select * from cliente";

Resultset rs = sql.executeQuery(consulta);
while(rs.next()){
  System.out.println(rs.getString("nome"));
  System.out.println(rs.getString("email"));
}
```

SQL EXCEPTION

- É a classe utilizada para tratar as possíveis exceções que podem ser geradas quando se acessa um Banco de Dados.

- **Método**

```
public int getErrorCode()
//Obtém o código de erro específico do fabricante do BD.
```

- **Exemplo**

```
try {
    //código Java
}
catch(SQLException e){...}
```

Passos Necessários para Acessar o BD com JDBC

1. **Informar as propriedades do BD**

```
String url = "jdbc:oracle:thin:@localhost:1521:XE";
String usuario = "curso_java";
String senha = "schema";
```

2. **Conectar com o BD**

```
Connection conexao = DriverManager.getConnection(url,usuario,senha);
```

3. **Criar um Objeto Statement**

```
Statement statement = conexao.createStatement();
```

Passos Necessários para Acessar o BD com JDBC

4. Definir a Instrução SQL a ser executada

```
String consulta = "SELECT * FROM Cliente";
```

5. Criar um Objeto ResultSet

```
ResultSet rs = statement.executeQuery(consulta);
```

6. Listar o resultado da Instrução SQL

```
while(rs.next()) {
      JOptionPane.showMessageDialog(null, "cpf:"+rs.getInt(1)+
      " nome:"+ rs.getString(2)+ " email"+ rs.getString(3));
}
```

Passos Necessários para Acessar o BD com JDBC

7. **Fechar a conexão com o BD**

```
conexao.close();
```

Exercícios

1) Criar a classe AcessoBD que faz uso das classes e interfaces JDBC listadas abaixo para realizar uma consulta na Tabela **Cliente**:
   - **public class DriverManager**
   - **public interface Connection**
   - **public interface Statement**
   - **public interface ResultSet**
   - **public class SQLException**

```
static String url = "jdbc:oracle:thin:@localhost:1521:XE";
static String usuario = "curso_java";
static String senha = "schema";
```

2) Utilizar a interface **DatabaseMetaData** para obter o nome do fabricante do BD e a sua versão utilizada.

3) Gerar uma exceção **ClassNotFoundException** removendo o driver Oracle do projeto '3Disciplina'.

Stored Procedures e Transações

Stored Procedures

DEFINIÇÃO

- ***Stored Procedure (SP)*** é um grupo de instruções SQL que formam uma unidade lógica para a realização de determinada tarefa específica.

- A ***SP*** agrupa um conjunto de operações ou consultas para serem executadas em um servidor de Banco de Dados.

- Uma ***SP*** encapsula tarefas repetitivas, aceita parâmetros de entrada e retorna um valor de status (para indicar aceitação ou falha na execução).

- O uso de ***SP*** pode reduzir o tráfego na rede, melhorar a performance da aplicação e aumentar os controles de segurança.

Stored Procedures

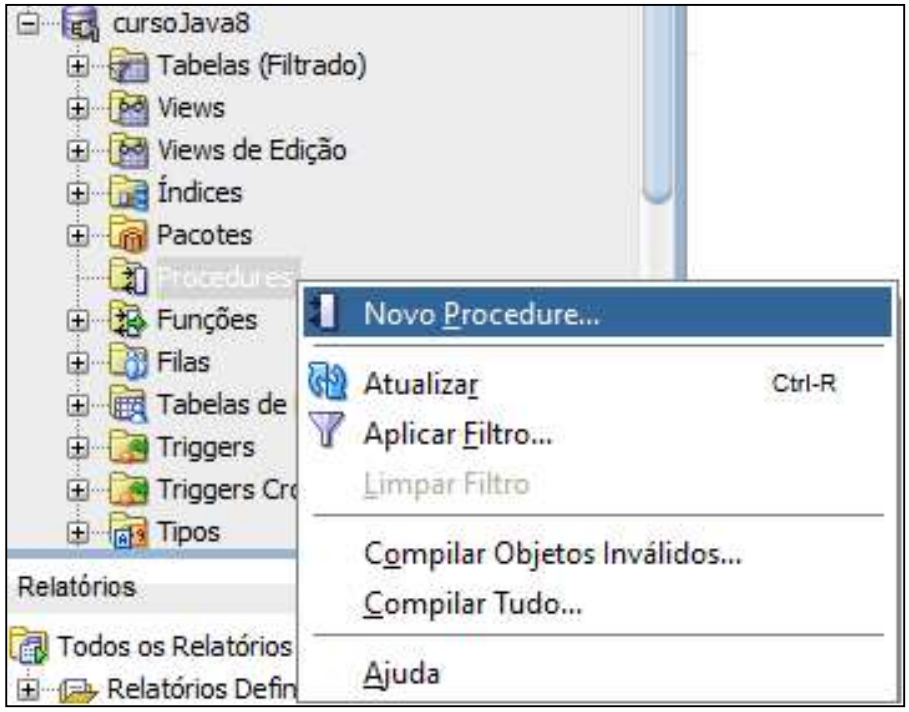
PL/SQL

- PL/SQL (*Procedural Language/Structured Query Language*) é uma extensão da linguagem padrão SQL para a criação de ***SP*** bem específicas para o SGBD Oracle.

```
DECLARE
   number1 NUMBER(2);
   number2 NUMBER(2)     := 17;
   text1   VARCHAR2(12)  := 'Hello world';
   text2   DATE          := SYSDATE;
BEGIN
   SELECT street_number
     INTO number1
     FROM address
    WHERE name = 'Billa';
END;
```

Criação de Stored Procedure no SQL Developer

- Selecionar Procedures > Novo Procedure...




```
CREATE OR REPLACE PROCEDURE
SP_INSERIRCLIENTE
(CPF IN INTEGER,
NOME IN VARCHAR2,
EMAIL IN VARCHAR2) AS
BEGIN
  INSERT INTO CLIENTE
          VALUES(CPF,NOME,EMAIL);
END SP_INSERIRCLIENTE;
```

SP_INSERIRREGISTROCLIENTE.SQL

- Executar a SP no Java

```
try {
    cstat = conexao.prepareCall
            ("{call SP_INSERIRREGISTROCLIENTE('"+cpf+"','"+nome+"','"+email+"')}");
    cstat.execute();
```

Transações

- Uma transação representa um conjunto de ações que devem ser realizadas de forma atômica, isto é, ou todas são realizadas com sucesso ou todas são canceladas.

- O conceito de transação em Banco de Dados visa preservar as ações realizadas pelas aplicações em um banco de dados.

- Quando múltiplas instruções SQL são agrupadas em uma única transação, todas as operações podem ser confirmadas (***committed***) ou canceladas (***rolled back***).

- Na interface Connection são definidos os métodos ***commit()*** e ***rollback()***.

- Por padrão, toda conexão JDBC considera uma instrução SQL como sendo uma transação. Pode-se utilizar o método ***setAutoCommit(false)*** para desabilitar esta opção.

Transações

- **Exemplo**

```
try {
  Statement stmt = con.createStatement();
  con.setAutoCommit(false);
  stmt.executeUpdate("UPDATE ...");
  stmt.executeUpdate("DELETE ...");
  ...
  con.commit();
}
catch (Exception e)
{
  try {
    con.rollback();
  }
  catch (SQLException e){}
}
```

Exercícios

1) Criar a classe ClienteApp para realizar as operações de CRUD na Tabela Cliente.

2) Criar na classe ClienteApp mais dois métodos **inserir(...)**:
   - o primeiro para fazer uso da interface PreparedStatement;
   - e o segundo uso da SP SP_INSERIRREGISTROCLIENTE.

3) Experimente retirar o controle de transações nas operações de CRUD. O que acontece?

4) ATIVIDADE EXTRA: realizar os exercícios (1) e (2) para a Tabela Curso.

Padrões de Projeto

Padrões de Projeto

DEFINIÇÃO

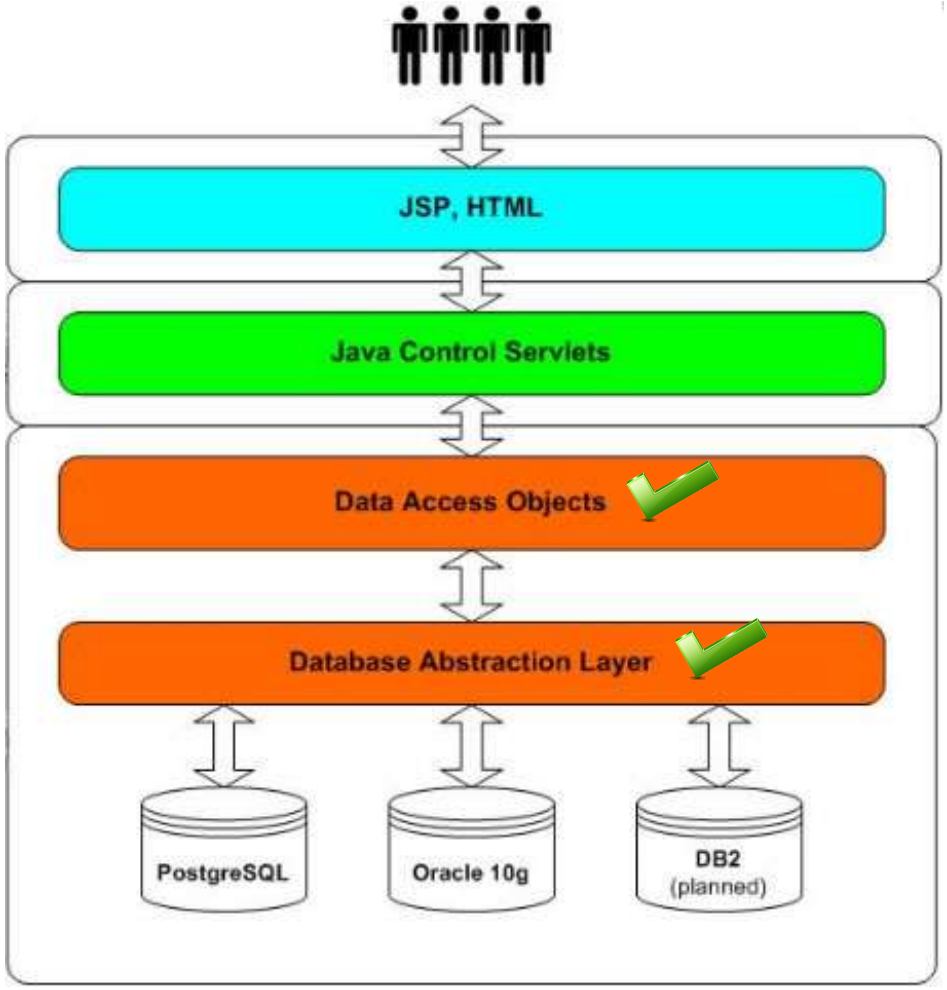
- **Padrões de Projeto** ou *Design Patterns* são utilizados para se alcançar objetivos na engenharia de software usando classes e métodos em linguagens orientadas a objeto.

- Padrões são um repertório de soluções e princípios que ajudam os desenvolvedores a criar software e que são codificados em um formato estruturado consistindo de: Nome, Problema que soluciona e Solução do problema.

- O objetivo dos padrões é codificar conhecimento (*knowing*) existente de uma forma que possa ser reaplicado em contextos diferentes.

- Há vários catálogos de padrões em software. Muitos são específicos a uma determinada área (padrões JEE, padrões de implementação em Java, em C#, padrões para concorrência, sistemas distribuídos, etc.).

Padrões de Projeto no JEE



- Os dois padrões apresentados nesta unidade são específicos para a melhor manipulação de dados em um SGBD. São eles: ***Transfer Object ou Value Object (VO)*** e ***Data Access Object (DAO)***.

Padrão de Projeto VO

OBJETIVO

- Reduzir a quantidade de requisições necessárias para recuperar um objeto. **VO** permite encapsular em um objeto um subconjunto de dados utilizável pelo cliente e utilizar apenas uma requisição para transferi-lo.

- Uma única chamada remota é necessária para transferir o **VO**.

- **VO** é a solução indicada para dados read-only ou informações que não são alteradas com freqüência, ou ainda, quando as alterações não são críticas (não afetam o processo).

VANTAGENS e DESVANTAGENS

- Transfere mais dados em menos chamadas, reduzindo o tráfego de rede;
- Reduz a duplicação de código;
- Pode introduzir objetos obsoletos e aumentar a complexidade do sistema.

Padrão de Projeto DAO

OBJETIVO

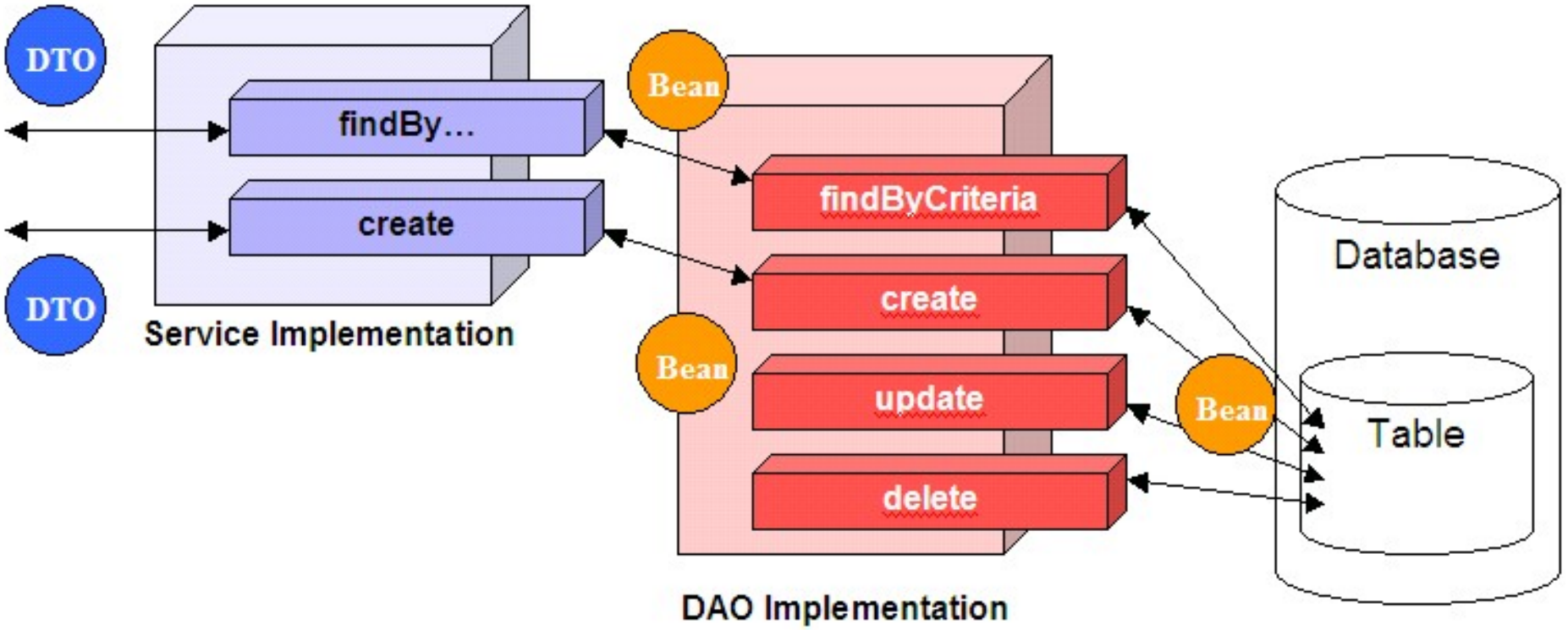
- Abstrair e encapsular todo o acesso a uma fonte de dados. O **DAO** gerencia a conexão com a fonte de dados para obter e armazenar os dados.

- O **DAO** oferece uma interface comum de acesso a dados e esconde as características de uma implementação específica, com métodos genéricos para ler e gravar dados em um SGBD.

VANTAGENS e DESVANTAGENS

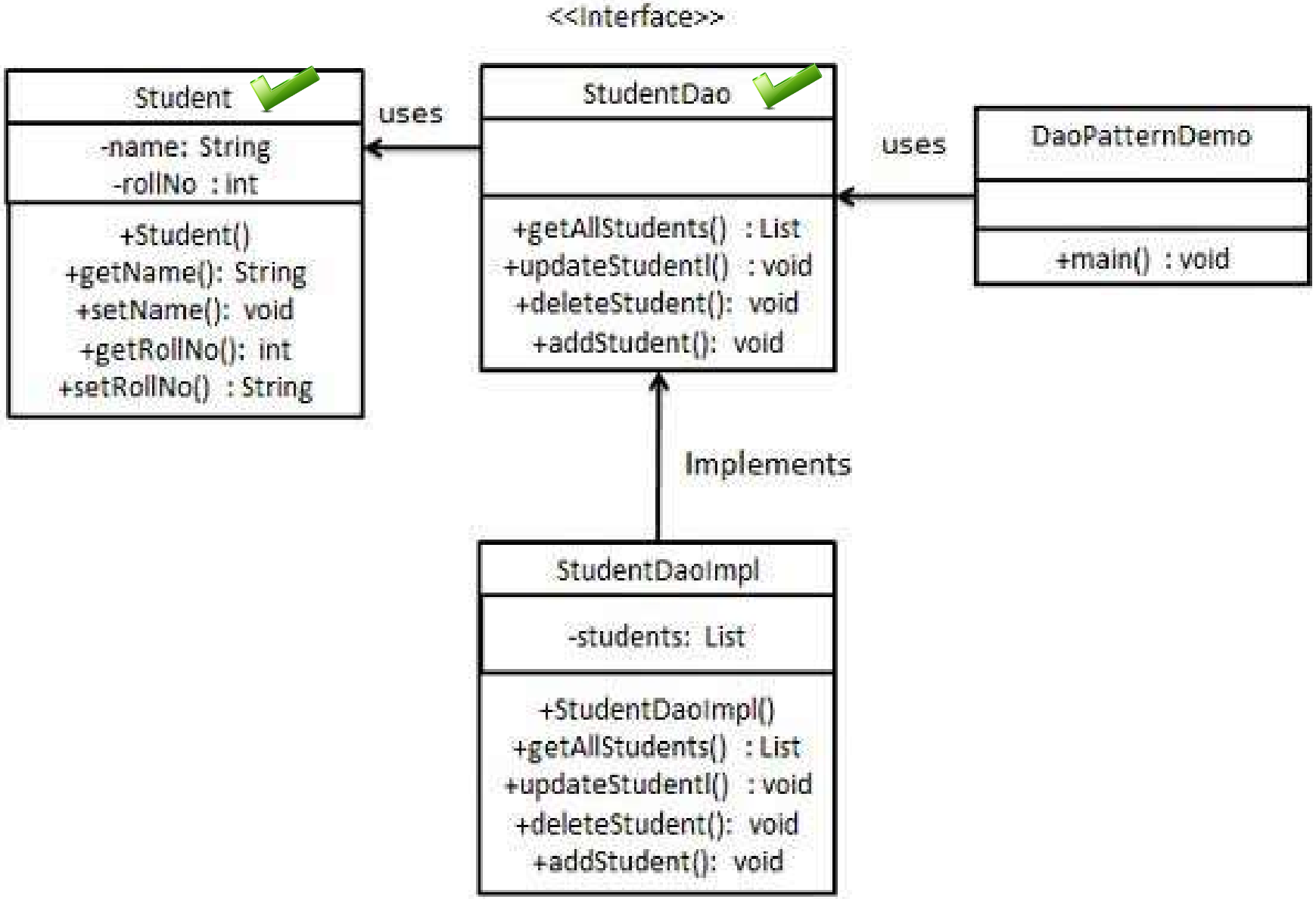
- Transparência quanto à fonte de dados, facilitando a migração para outras implementações;
- Reduz complexidade do código nos objetos de negócio;
- Centraliza todo acesso aos dados em camada separada;
- Camada adicional que pode gerar impacto na performance.

VO (DTO) e DAO



http://slideplayer.com.br

Exemplo de VO e DAO em Java



http://www.tutorialspoint.com/design_pattern/data_access_object_pattern.htm

Exemplo de VO e DAO em Java

Student.java

```
public class Student {
   private String name;
   private int rollNo;

   Student(String name, int rollNo){
      this.name = name;
      this.rollNo = rollNo;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public int getRollNo() {
      return rollNo;
   }

   public void setRollNo(int rollNo) {
      this.rollNo = rollNo;
   }
}
```

StudentDao.java

```
import java.util.List;

public interface StudentDao {
   public List<Student> getAllStudents();
   public Student getStudent(int rollNo);
   public void updateStudent(Student student);
   public void deleteStudent(Student student);
}
```

Exemplo de VO (DTO) em Java

StudentDaoImpl.java

```
import java.util.ArrayList;
import java.util.List;

public class StudentDaoImpl implements StudentDao {

   //list is working as a database
   List<Student> students;

   public StudentDaoImpl(){
      students = new ArrayList<Student>();
      Student student1 = new Student("Robert",0);
      Student student2 = new Student("John",1);
      students.add(student1);
      students.add(student2);
   }
   @Override
   public void deleteStudent(Student student) {
      students.remove(student.getRollNo());
      System.out.println("Student: Roll No " + student.getRollNo() + ", 
   }

   //retrive list of students from the database
   @Override
   public List<Student> getAllStudents() {
      return students;
   }
```

Exemplo de VO (DTO) em Java

DaoPatternDemo.java

```
public class DaoPatternDemo {
   public static void main(String[] args) {
      StudentDao studentDao = new StudentDaoImpl();

      //print all students
      for (Student student : studentDao.getAllStudents()) {
         System.out.println("Student: [RollNo : " + student.getRollNo() + ", Name : "
      }


      //update student
      Student student =studentDao.getAllStudents().get(0);
      student.setName("Michael");
      studentDao.updateStudent(student);

      //get the student
      studentDao.getStudent(0);
      System.out.println("Student: [RollNo : " + student.getRollNo() + ", Name : " +
   }
}
```

Exercícios

1) Criar o pacote **com.abctreinamentos** e reescrever a classe ClienteApp para fazer uso dos padrões de projeto DAO e VO.

2) ATIVIDADE EXTRA: realizar o exercício (1) para a Tabela Curso.

RESUMO

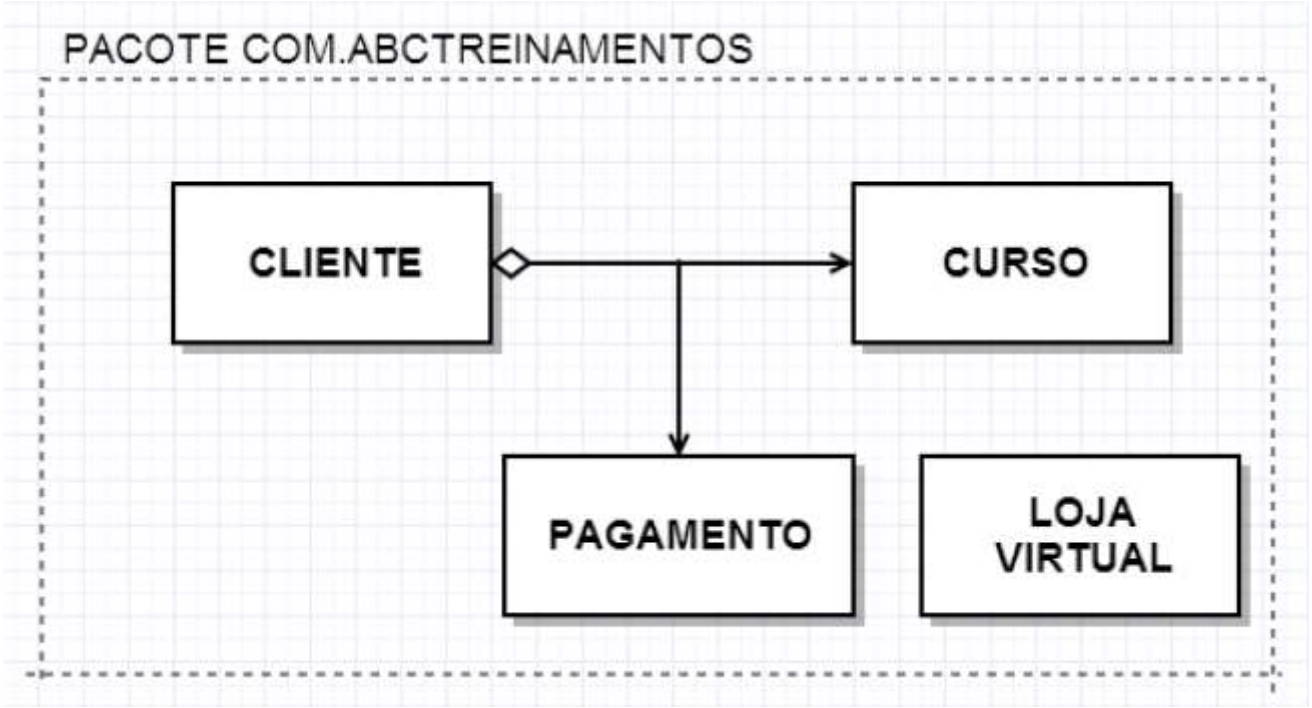
TÓPICOS APRESENTADOS

- Nesta aula nós estudamos:

  - **API JDBC**
  - **Driver SGBD ORACLE**
  - **Principais Classes e Interfaces JDBC**
  - **Stored Procedures e Transações**
  - **Padrões de Projeto**

ATIVIDADES PARA SE APROFUNDAR

1) **Dado o DIAGRAMA DE CLASSES ABAIXO, fazer o que se pede:**



a) **Criar a Tabela Pagamento (#cpf,#cdcurso,datainscricao).**

b) **Criar no pacote com.abctreinamentos a classe Pagamento.**

c) **Criar as operações de CRUD na classe LojaVirtual.**

ATIVIDADES PARA SE APROFUNDAR

- **2) Refazer os exercícios de BD propostos nesta unidade em outro Banco de Dados (MySQL, por exemplo).**
  - **Procure e instale o driver JDBC correspondente**
  - **Altere a URL utilizada**
  - **Altere as estruturas de dados definidas**

- **3) Fazer um estudo dos padrões de projeto JEE, baseado nos *blueprints JEE*.**