

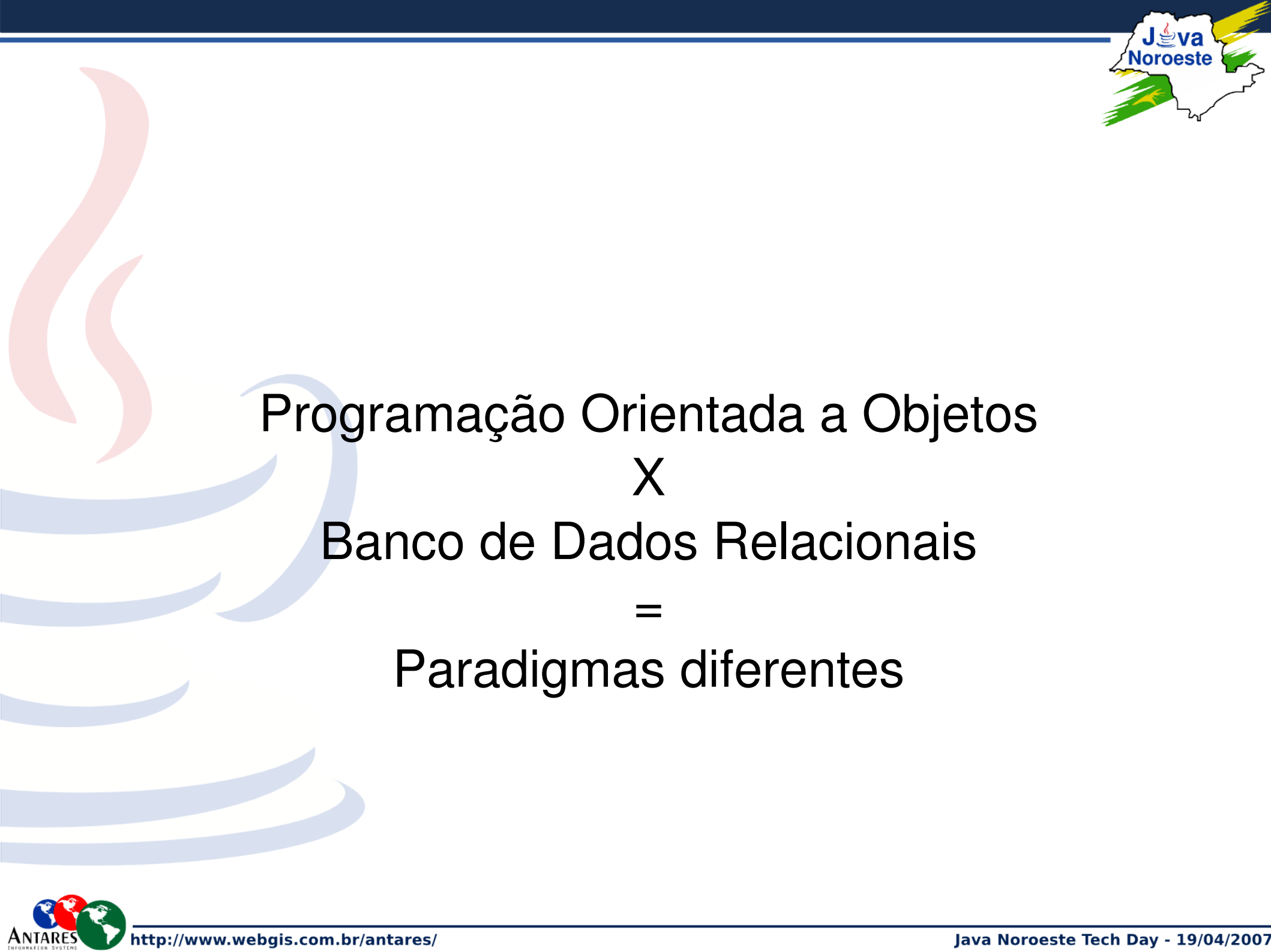


JPA: Persistência padronizada em Java

FLÁVIO HENRIQUE CURTE

Bacharel em Engenharia de Computação

flaviocurte.java@gmail.com

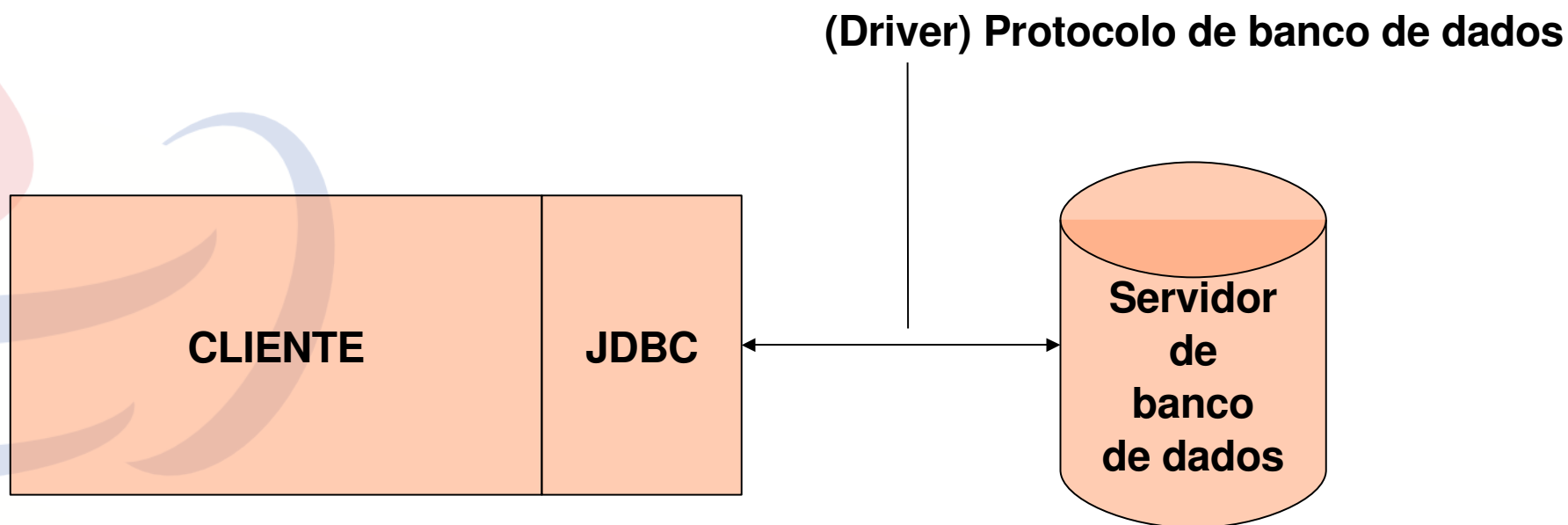
The background of the slide features abstract, flowing shapes in shades of pink and light blue on the left side.

Programação Orientada a Objetos
X
Banco de Dados Relacionais
=
Paradigmas diferentes

Java Database Connectivity (JDBC)

Trabalha no mesmo nível do banco de dados. O acesso as informações contidas no banco são realizadas através de comandos SQL.

Java Database Connectivity (JDBC)



Horstmann, Cay S., Cornell, Gary. Corejava 2 – Volume II – Recursos avançados, pág.181. São Paulo: Makron Books, 2001.

Java Database Connectivity (JDBC)

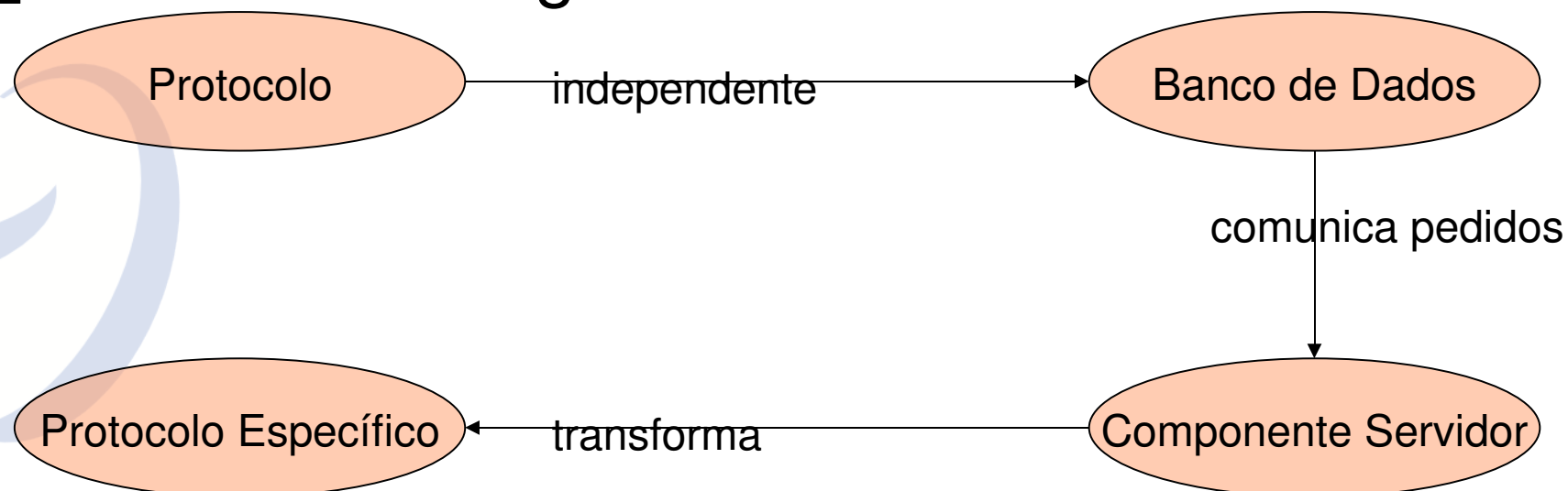
Tipos de drivers JDBC:

- Tipo 1: JDBC-ODBC bridge. Adaptação do driver ODBC para JDBC, utilizando os recursos do ODBC para se comunicar com o banco de dados;
- Tipo 2: JDBC-Native Bridge. Converte chamadas JDBC da aplicação Java para chamadas de um driver nativo já instalado na máquina. A comunicação do driver nativo é realizada através de um protocolo proprietário;

Java Database Connectivity (JDBC)

Tipos de drivers JDBC (continuação):

- Tipo 3: JDBC-Net bridge. Biblioteca cliente Java pura.



- Tipo 4: All Java JDBC Driver. 100% escrito em Java que transforma pedidos JDBC diretamente em um protocolo específico do banco de dados.

Java Database Connectivity (JDBC)

Exemplo:

```
import java.sql.*;

public class JDBC {
    private static Connection con = null;

    public JDBC() {
    }

    public static Connection open() {
        String user = "postgres";
        String pass = "postgres";
        String url =
            "jdbc:postgresql://192.168.1.11/javanoroeste";

        try {
            Class.forName("org.postgresql.Driver");
            con =
                DriverManager.getConnection(url,user,pass);
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        return con;
    }
}
```

```
public static void main(String[] args) throws
Exception {
    String sql = "SELECT * FROM jpa";
    con = open();
    try {
        Statement comando =
            con.createStatement();
        ResultSet resultado =
            comando.executeQuery(sql);
        while (resultado.next()) {

            System.out.println(resultado.getString("nome"));

            System.out.println(resultado.getString("email"));
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
    } finally {
        con.close();
    }
}
```

C

```

public static Connection open() {
    String user = "postgres";
    String pass = "postgres";
    String url = "jdbc:postgresql://192.168.1.11/javanoroeste";
    try {
        Class.forName("org.postgresql.Driver");
        con = DriverManager.getConnection(url,user,pass);
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    return con;
}

```

CREATE

R

```

Class.forName("org.postgresql.Driver");
con = DriverManager.getConnection(url,user,pass);
try {
    String url = "jdbc:postgresql://192.168.1.11/javanoroeste";
    String user = "postgres";
    String pass = "postgres";
    String url = "jdbc:postgresql://192.168.1.11/javanoroeste";
    try {
        Class.forName("org.postgresql.Driver");
        con = DriverManager.getConnection(url,user,pass);
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    return con;
}
}

```

UPDATE

U

```

public static Connection open() {
    String user = "postgres";
    String pass = "postgres";
    String url = "jdbc:postgresql://192.168.1.11/javanoroeste";
    try {
        Class.forName("org.postgresql.Driver");
        con = DriverManager.getConnection(url,user,pass);
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    return con;
}
}

```

UPDATE

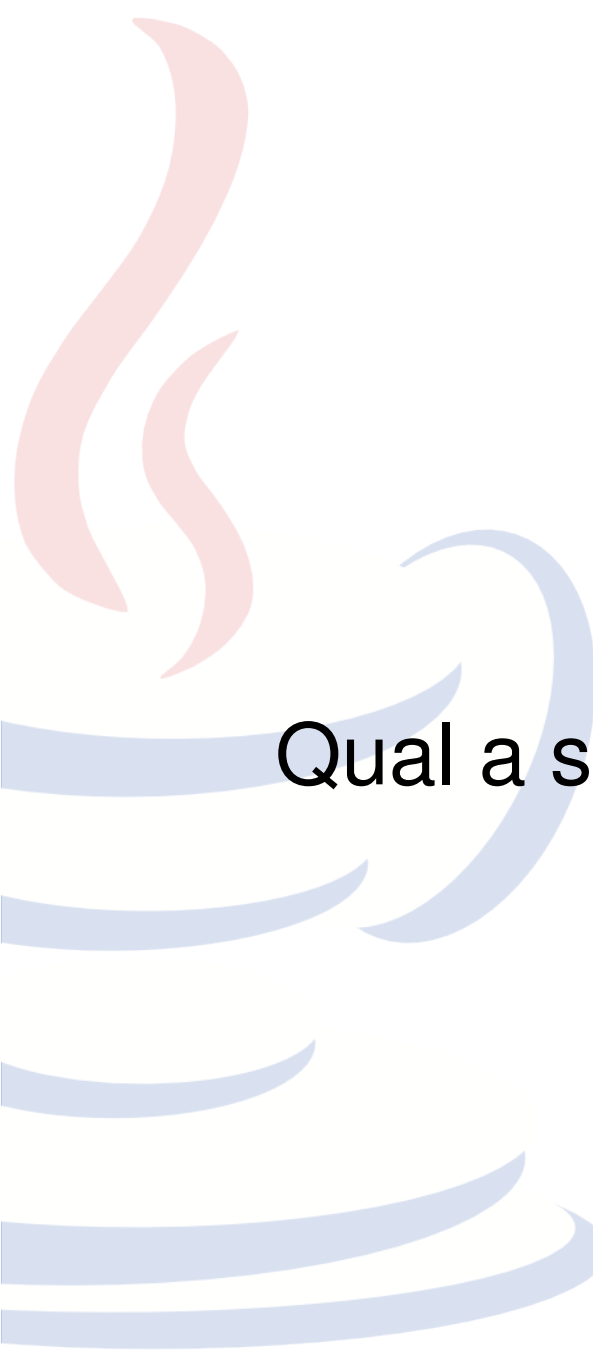
D

```

public static Connection open() {
    String user = "postgres";
    String pass = "postgres";
    String url = "jdbc:postgresql://192.168.1.11/javanoroeste";
    try {
        Class.forName("org.postgresql.Driver");
        con = DriverManager.getConnection(url,user,pass);
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    return con;
}
}

```

DELETE

On the left side of the slide, there are decorative swirls. Two are light red and several are light blue, creating a dynamic, flowing background element.

Qual a solução para armazenar objetos na tecnologia relacional?

Mapeamento Objeto/Relacional

A idéia da persistência O/R é reunir as vantagens de se utilizar um modelo orientado a objetos para a construção de uma aplicação, com a performance e a confiabilidade dos bancos de dados relacionais.

Mapeamento Objeto/Relacional

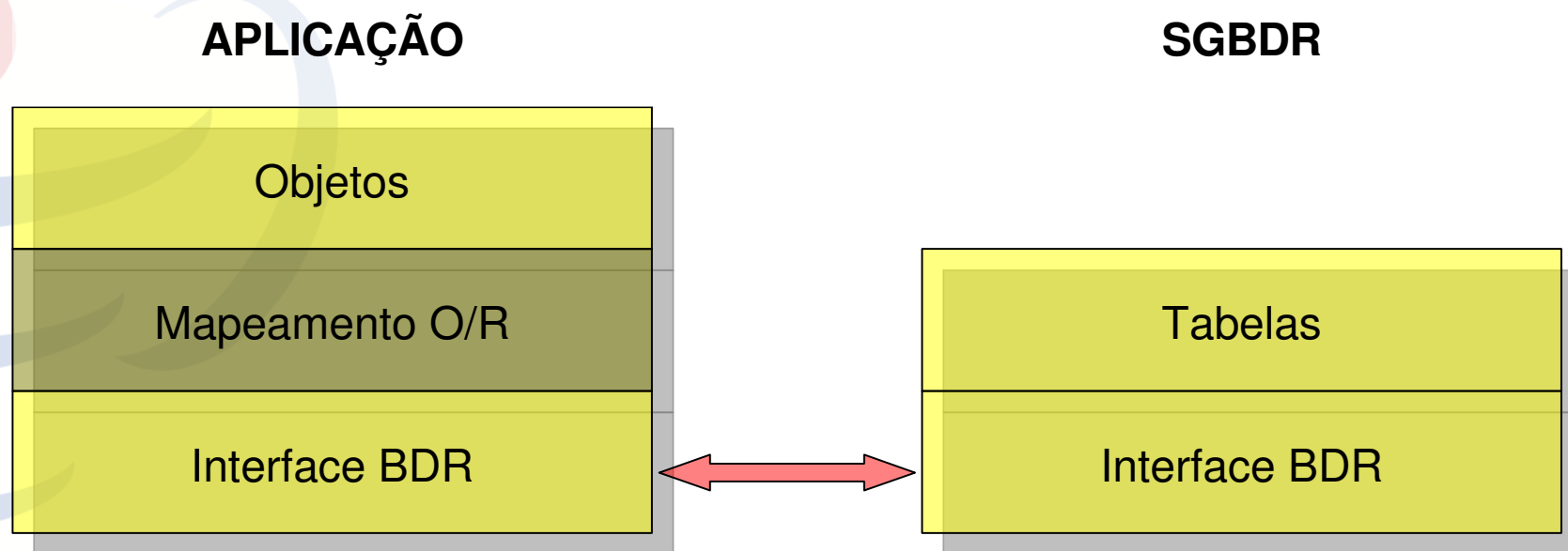


Figura: Doederlein, Osvaldo Pinali. Revista Java Magazine, ed. 42, p. 22.

Mapeamento Objeto/Relacional

Framework Hibernate:

- Programação OO (herança, polimorfismo etc);
- Sem aumento de tempo na construção da aplicação
- Gratuito e aberto
- Portável para todos os bancos compatíveis com o padrão SQL

Mapeamento Objeto/Relacional

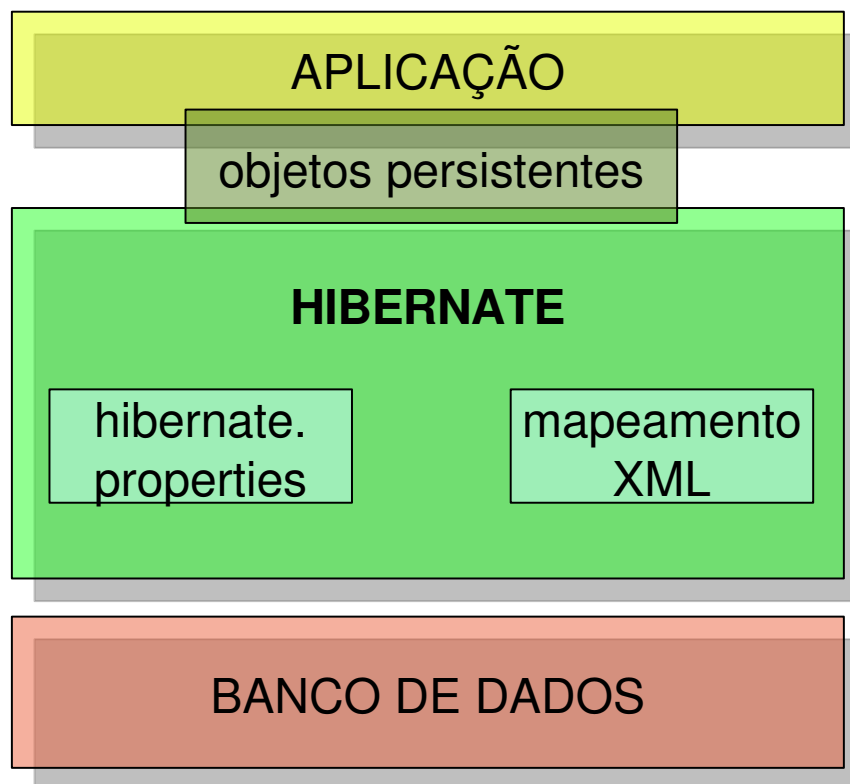


Figura: Boaglio, Fernando. Revista SQL Magazine, ed. 17, p. 40.

Mapeamento Objeto/Relacional

- Framework Hibernate e JDO: duas grandes ferramentas de persistência ORM.
- JPA: Assim como os frameworks citados acima, são baseados em POJOs (*Plain Old Java Objects*).

Java Persistence API 1.0

- Definida na JSR-220 (*Enterprise JavaBeans, Version 3.0*)
- *Forma simples de mapear objetos no banco de dados*
- *Padroniza o mapeamento O/R*

Java Persistence API 1.0

- *Solução completa para mapeamento e persistência de objetos: modo declarativo de descrever mapeamento O/R, linguagem de consulta, ferramentas para manipular entidades*
- *Aumento de produtividade: JCP=padronização*

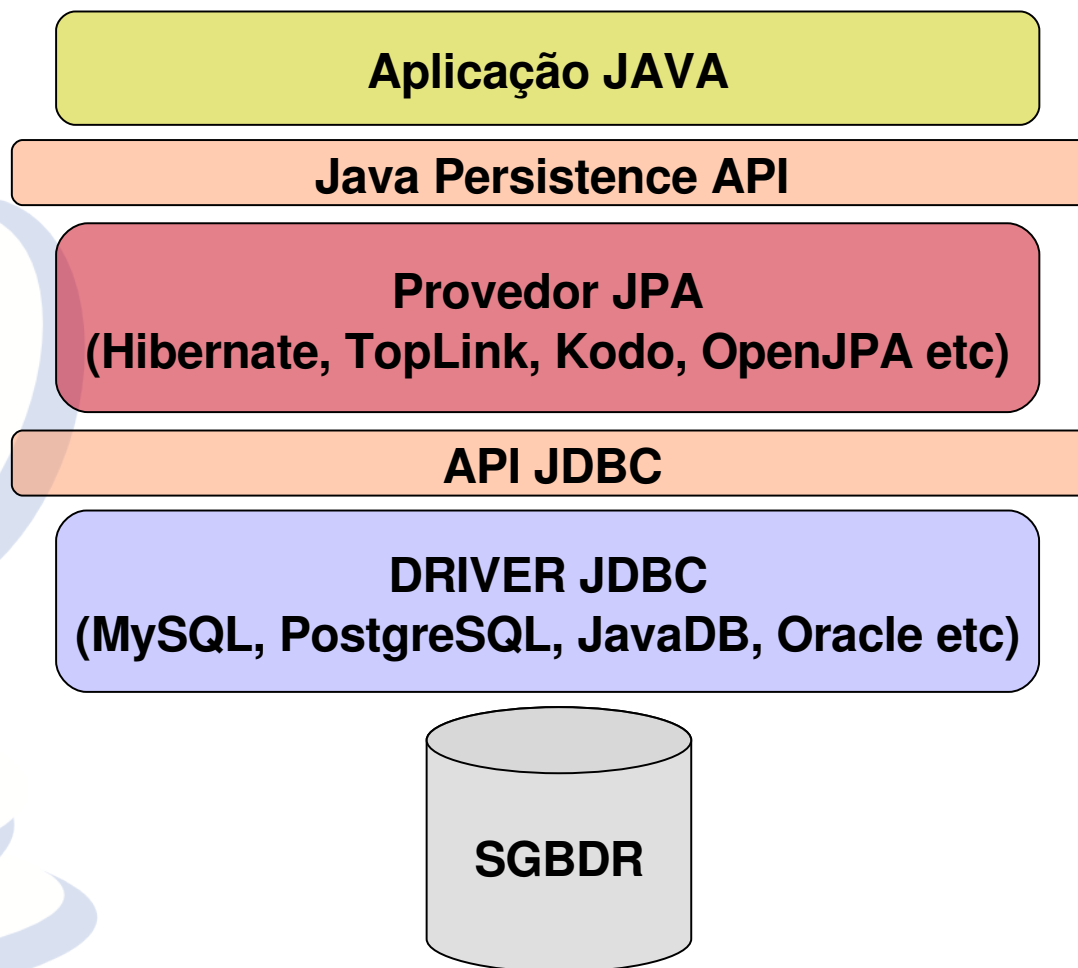
Java Persistence API 1.0

EntityManager: responsável por praticamente todas operações de persistência de objetos

PersistenceContext: área de memória que mantém os objetos que estão sendo manipulados pelo EntityManager

Provedores: especificação para frameworks de persistência

Java Persistence API 1.0



Adaptado de Bellia, Renato. Revista Java Magazine, ed. 44, p. 28.

Java Persistence API 1.0

Persistence Unit: Configuração do provedor JPA para localizar o banco de dados e estabelecer conexões JDBC.

Java Persistence API 1.0

Mapeamento:

- Classes e interfaces estão no pacote:
javax.persistence
- Uso anotações (@)
- Configuração por exceção

Java Persistence API 1.0

Anotação:

@Entity designa um POJO

@Entity

```
public class Funcionario implements Serializable {
```

```
...
```

```
}
```

Java Persistence API 1.0

Anotação:

@Table Por padrão, a JPA assume que todos os campos persistentes de uma entidade serão armazenados em um BD com o mesmo nome da entidade.

```
@Entity
```

```
@Table(name="FUN")
```

```
public class Funcionario implements Serializable {
```

```
...
```

```
}
```

Java Persistence API 1.0

Anotação:

@Column por padrão, a JPA assume que o nome de cada atributo corresponde ao mesmo nome na tabela

```
@Entity
public class Funcionario implements Serializable {
    ...
    @Column(name="FUN_ID")
    private Long Id;
    ...
}
```

Java Persistence API 1.0

Anotação:

@Id identificador da chave primária

```
@Entity
public class Funcionario implements Serializable {
    ...
    @Id
    @Column(name="FUN_ID")
    private Long Id;
    ...
}
```

Java Persistence API 1.0

Anotação:

@NamedQuery cria consulta pré-definida (estática)
associado com o **@Entity**

@Entity

**@NamedQuery(name = “ConsultaPorId”, query = “SELECT f FROM
Funcionarios f WHERE f.id = :id”)**

public class Funcionario implements Serializable {

...

}

Java Persistence API 1.0

Outras anotações mais comuns:

***@GeneratedValue** geração automática de identificadores*

***@Temporal** para informações relacionadas ao tempo
(DATE, TIME e TIMESTAMP)*

***@OneToMany** para relacionamentos “um-para-muitos” em
BD relacional*

***@ManyToOne** para relacionamento “muitos-para-um” em
BD relacional*

Demonstração





JPA: Persistência padronizada em Java

FLÁVIO HENRIQUE CURTE
Bacharel em Engenharia de Computação

flaviocurte.java@gmail.com