

Mais sobre persistência...

DAO

Mapeamento objeto/relacional



Problema

- Colocar o código SQL dentro das classes muitas vezes não reflete a modelagem do sistema:
 - Perda da representatividade
 - Dificuldade de manutenção
 - Entendimento da modelagem de dados
- Idéia: remover o código de acesso ao banco das classes do sistema e colocá-lo em uma classe responsável pelo acesso aos dados



Objetivo

- Passagem de objetos ao invés da especificação das suas propriedades:

```
// adiciona os dados no banco  
Misterio bd = new Misterio();  
bd.adiciona("meu nome", "meu email", "meu endereço", meuCalendar);
```

```
// adiciona um contato no banco  
Misterio bd = new Misterio();
```

```
// método muito mais elegante  
bd.adiciona(contato);
```



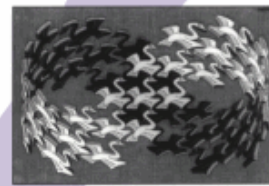
Design Pattern

- Padrões de Projeto:
 - Factory
 - Data Access Object: DAO
Isola a camada do modelo (sistema) da camada de persistência

Design Patterns

Elements of Reusable
Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



Cover art © 1994 MIT. Fisher - Gordon Art - Boston - Holland. All rights reserved.

Foreword by Grady Booch

ADDITION-WESLEY PROFESSIONAL COMPUTING SERIES



DAO

Classes do
Modelo

M1

M2

M_n

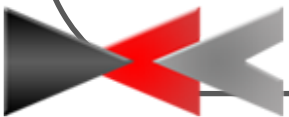
DAO

Persistência

BD

xml

LDAP

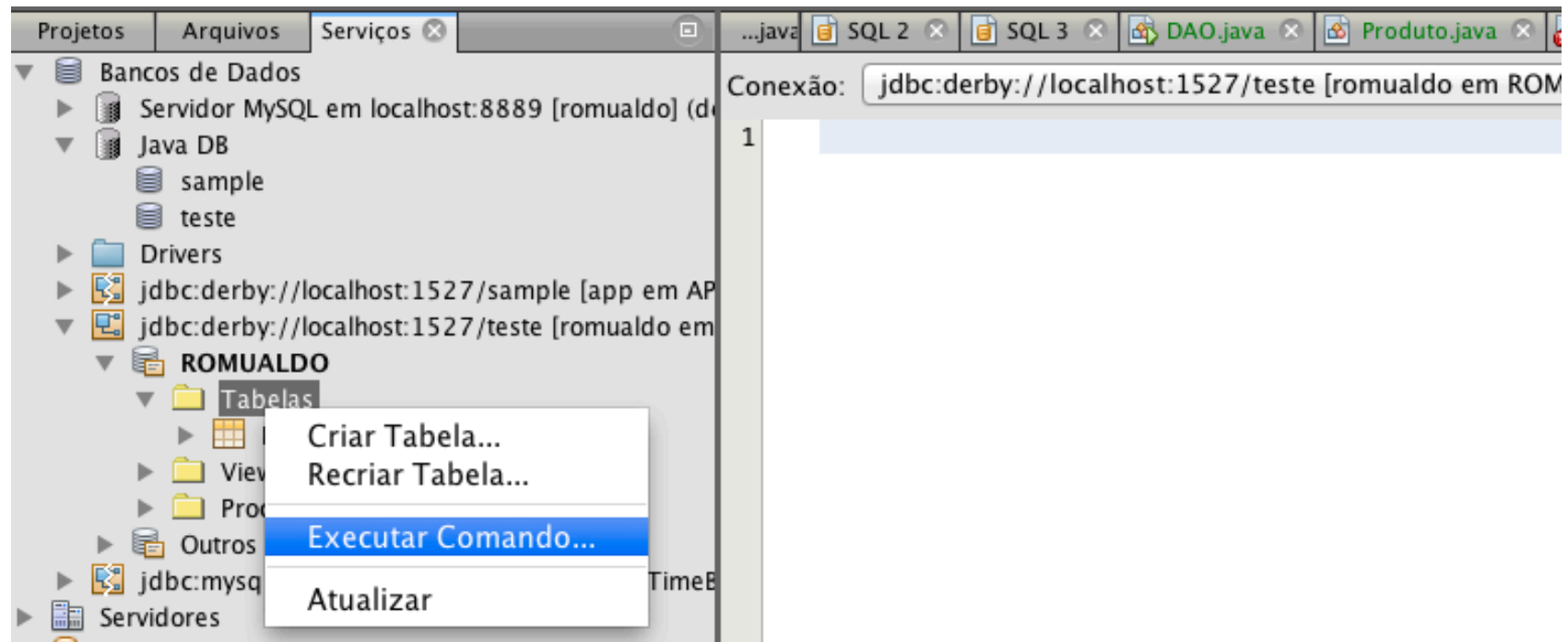


Vantagens

- Independência entre o modelo de persistência e o modelo do sistema:
 - Pode-se mudar a persistência sem modificar o sistema
 - Clientes diferentes podem utilizar SGBDs diferentes
 - Mecanismos de persistência distintos podem ser utilizados (Relacional, BigData, Arquivos etc.)
 - Encapsulamento das conexões quando necessárias



Persistência



Construir a Tabela

Conexão: jdbc:derby://localhost:1527/teste [romualdo em ROMUA...]

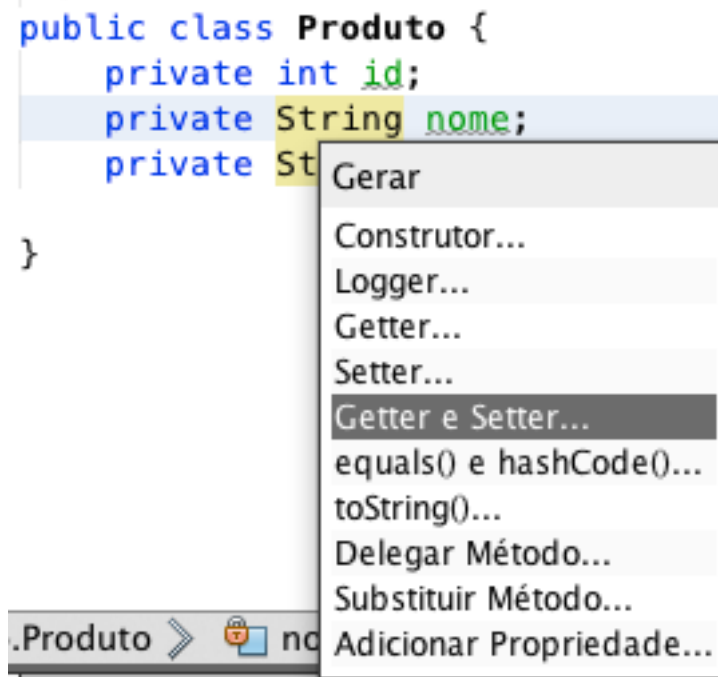
```
1 CREATE TABLE produto (  
2     id int NOT NULL GENERATED ALWAYS AS IDENTITY  
3         (START WITH 1, INCREMENT BY 1),  
4     nome char(60) NOT NULL,  
5     descricao char(80) NOT NULL,  
6     PRIMARY KEY (id)  
7 )  
8
```



Classe Produto

- Criar um novo projeto
- Especificar a classe produto

```
public class Produto {  
    private int id;  
    private String nome;  
    private St  
}
```



```
public class Produto {  
    private int id;  
    private String nome;  
    private String descricao;  
  
    public int getId() {  
        return id;  
    }  
  
    public void setId(int id) {  
        this.id = id;  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public String getDescricao() {  
        return descricao;  
    }  
  
    public void setDescricao(String descricao) {  
        this.descricao = descricao;  
    }  
}
```



CRUD

- Create, Read, Update e Delete
 - Quatro operações básicas utilizadas em bases de dados relacionais
- Especificar os métodos para realizar as operações de CRUD
 - Criar a classe CRUD
 - Métodos para as operações



CRUD

```
public class CRUD {  
  
    private Connection con;  
  
    public CRUD(Connection con){  
        this.con = con;  
    }  
}
```



Create

```
public void create(Produto produto) throws Exception {  
    PreparedStatement p =  
        con.prepareStatement("insert into produto (nome, descricao) values (?,?)");  
    p.setString(1, produto.getNome());  
    p.setString(2, produto.getDescricao());  
    p.executeUpdate();  
    p.close();  
}
```



Read

```
public Produto read(Produto produto) throws Exception {  
    PreparedStatement p = con.prepareStatement("Select * from produto where id = ?");  
    p.setInt(1, produto.getId());  
    ResultSet rs = p.executeQuery();  
    Produto produtoRead;  
    if (rs.next()){  
        produtoRead = new Produto();  
        produtoRead.setId(rs.getInt("id"));  
        produtoRead.setNome(rs.getString("nome"));  
        produtoRead.setDescricao(rs.getString("descricao"));  
    } else  
        produtoRead=null;  
    p.close();  
    return produtoRead;  
}
```



Update

```
public void update(Produto produto) throws Exception {  
    PreparedStatement p =  
        con.prepareStatement("update produto set nome = ?, descricao = ? where id = ?");  
    p.setString(1, produto.getNome());  
    p.setString(2, produto.getDescricao());  
    p.setInt(3, produto.getId());  
    p.executeUpdate();  
    p.close();  
}
```



Delete

```
public void delete(Produto produto) throws Exception {  
    PreparedStatement p =  
        con.prepareStatement("delete from produto where id = ?");  
    p.setInt(1, produto.getId());  
    p.executeUpdate();  
    p.close();  
}
```



Outros

```
public List<Produto> readAll() throws Exception{
    List<Produto> produtos = new ArrayList<Produto>();
    PreparedStatement p = con.prepareStatement("select * from produto");
    ResultSet rs = p.executeQuery();
    while(rs.next()){
        Produto produto = new Produto();
        produto.setId(rs.getInt("id"));
        produto.setNome(rs.getString("nome"));
        produto.setDescricao(rs.getString("descricao"));
        produtos.add(produto);
    }
    rs.close();
    p.close();
    return produtos;
}
```



Factory

- Considerando a possibilidade de conexões a diferentes bancos de dados, podemos implementar uma “fábrica” de conexões
 - Transparência para a aquisição dos programadores

```
public class ConnectionFactory {  
    public Connection getConnection() {  
        try {  
            return DriverManager.getConnection(  
                "jdbc:mysql://localhost/fj21", "root", "");  
        } catch (SQLException e) {  
            throw new RuntimeException(e);  
        }  
    }  
}
```



Classe ConnectionFactory

```
import java.sql.Connection;
import java.sql.DriverManager;

/**
 *
 * @author romualdocosta
 */
public class ConnectionFactory {

    static final String DB_URL = "jdbc:derby://localhost:1527/teste";
    static final String USER = "romualdo";
    static final String PASS = "0";

    public Connection getConnection() throws Exception {
        return DriverManager.getConnection(DB_URL, USER, PASS);
    }
}
```



Main

```
public static void main(String[] args) throws Exception {  
    // TODO code application logic here  
    Connection con = ConnectionFactory.getConnection();  
    CRUD crud = new CRUD(con);  
  
    Produto p = new Produto();  
    p.setDescricao("produto teste");  
    p.setNome("teste");  
    crud.create(p);  
  
    List list = crud.readAll();  
    Iterator i = list.iterator();  
    while (i.hasNext()) {  
        p = (Produto) i.next();  
        System.out.printf(p.getDescricao());  
        System.out.printf(p.getNome());  
    }  
}
```



Adicionar Dependências

