DAO – Data Access Object e Conversão de datas para gravar no banco de dados

A idéia do DAO é remover todo o código de acesso a Banco de Dados de suas classes de lógica e colocá-lo em uma única classe responsável pelo acesso a dados. Separando o código de acesso ao banco de suas classes de lógica, a manutenção fica facilitada.

Agora a idéia é poder ter um código que adicione um Contato ao banco. Veja o código a seguir:

```
//Adiciona os dados no banco
ClasseDao dao = new ClasseDao();
dao.adiciona("meu nome","meu email","meu endereco",meuCalendar);
```

Repare que todos os parâmetros que estamos passando são as informações do Contato. Se o Contato tivesse 30 atributos, passaríamos 30 parâmetros? Java é orientado a Strings? Agora vamos tentar um código mais simples e funcional:

```
//Adiciona um contato no banco
ClasseDao dao = new ClasseDao();
//Método muito mais elegante
dao.adiciona(contato);
```

Vamos tentar chegar ao código anterior. O mais elegante seria poder chamar um único método para inclusão, certo?

```
public class TestaInsere {
	public static void main(String[] args) {
		Contato contato = new Contato();
		contato.setNome("Fulano de tal");
		contato.setEmail("fulano@gmail.com");
		contato.setEndereco("Rua Monsenhor Miranda, 86, Centro");
		contato.setDataNascimento(Calendar.getInstance());

		// Grave nesta conexão
		ClasseDAO dao = new ClasseDAO();

		// Método elegante
		dao.adiciona(contato);

		System.out.println("Contato Gravado");
	}
}
```

Repare que estamos isolando todo o acesso a banco de dados em classes bem simples, cuja instância é um objeto responsável por acessar os dados. Desta responsabilidade surgiu o termo Data Access Object ou simplesmente DAO, um dos mais famosos padrões de projeto (design pattern).

O que falta agora é criarmos a classe ClasseDAO, ou melhor, chamaremos de ContatoDAO, que possui um método adiciona. Vamos criar uma que se conecta ao banco ao criarmos uma instância dela:

```
public class ContatoDAO {
```

```
        private Connection conexao;

        public ContatoDAO() {
                this.conexao = new ConnectionFactory().getConnection();
        }
}
```

Agora que todo ContatoDAO possui uma conexão com o banco, podemos focar no método adiciona, que recebe um Contato como argumento e éresponsável por adicioná-lo através de código SQL:

```
public void adiciona(Contato c) {
    //Testando se preciso fabricar uma conexao
        try {
            if(conexao.isClosed())
                conexao=new ConnectionFactory().getConexao();
        } catch (SQLException e1) {
            System.out.println("Erro ao obter conexao."+e1.getMessage());
        }
    //declarar minha instrução sql
    this.comandoSQL="INSERT INTO contatos(nome,email,endereco,dataNascimento) VALUES(?,?,?,?)";
    //Preparando minha instrução através do método prepareStatement da minha conexão
    try {
        this.stmt=this.conexao.prepareStatement(comandoSQL);
        //passando os argumentos
        stmt.setString(1, c.getNome());
        stmt.setString(2, c.getEmail());
        stmt.setString(3, c.getEndereco());
        //Convertendo a data para gravar
        java.sql.Date dataParaGravar = new java.sql.Date(c.getDataNascimento().getTimeInMillis());
        stmt.setDate(4, dataParaGravar);
        //executa a instrução preparada
        stmt.execute();
        System.out.println("Contato "+c.getNome()+" adicionado com sucesso!");//ferindo o SRP
    } catch (SQLException e) {
        System.out.println("Erro ao adicionar"+e.getMessage());
    } finally{
        //liberando os recusros
        try {
            stmt.close();
            conexao.close();
            System.out.println(conexao);
        } catch (SQLException e) {
            System.out.println("Não foi possivel liberar os recursos"+e.getMessage());
        }
    }
}
```

Perceba que tratamos a SQLException respeitando o SRP (princípio da responsabilidade única).

Exercícios com Javabeans e DAO

1) Crie a classe Contato:
    a) No pacote br.com.cefet.modelo, crie uma classe chamada Contato.

```
import java.util.Calendar;

public class Contato {
    private long Id;
    private String nome, email, endereco;
    private Calendar dataNascimento;

    //métodos get e set....
```

    b) Aperte Ctrl + 3 e digite ggas (abreviação de Generate Getters and Setters) e selecione todos os getters e setters.
2) Crie a classe ContatoDAO no pacote br.com.cefet.modelo.dao (utilize CTRL + SHIFT + O para os imports)

```java
package br.com.cefet.modelo.dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

import br.com.cefet.modelo.ConnectionFactory;
import br.com.cefet.modelo.Contato;

public class ContatoDAO {

    //Atributo privado para guardar a conexão (interface Connection)
    private Connection conexao;
    //Instrução preparada (interface PreparedStatement)
    private PreparedStatement stmt;
    //Comando SQL a ser executado pelo PreparedStatement
    private String sql;

    /*Construtor que obtém uma conexão com o bd através da
    ConnectionFactory */
    public ContatoDAO() {
        conexao = new ConnectionFactory().getConexao();
    }

public void adiciona(Contato c) {
    //Testando se preciso fabricar uma conexao
        try {
            if(conexao.isClosed())
                conexao=new ConnectionFactory().getConexao();
        } catch (SQLException e1) {
            System.out.println("Erro ao obter conexao."+e1.getMessage());
        }
    //declarar minha instrução sql
    this.comandoSQL="INSERT INTO contatos(nome,email,endereco,dataNascimento) VALUES(?,?,?,?)";
    //Preparando minha instrução através do método prepareStatement da minha conexão
    try {
        this.stmt=this.conexao.prepareStatement(comandoSQL);
        //passando os argumentos
        stmt.setString(1, c.getNome());
        stmt.setString(2, c.getEmail());
        stmt.setString(3, c.getEndereco());
        //Convertendo a data para gravar
        java.sql.Date dataParaGravar = new java.sql.Date(c.getDataNascimento().getTimeInMillis());
        stmt.setDate(4, dataParaGravar);
        //executa a instrução preparada
        stmt.execute();
        System.out.println("Contato "+c.getNome()+" adicionado com sucesso!");//ferindo o SRP
    } catch (SQLException e) {
        System.out.println("Erro ao adicionar"+e.getMessage());
    } finally{
        //liberando os recusros
        try {
            stmt.close();
            conexao.close();
            System.out.println(conexao);
        } catch (SQLException e) {
            System.out.println("Não foi possivel liberar os recursos"+e.getMessage());
        }
    }
}
```

Apesar de termos tratado todos os erros de SQL no DAO, o código acima ainda fere o SRP (Princípio da Responsabilidade Única). Por que???

Lembre-se de importar as classes SQL do pacote Java.sql, **inclusive a classe Date!**

3) Crie, no pacote br.com.cefet.teste, uma classe chamada TestaInsereDAO com um método main:

```
public class TestaInsereDAO {
    public static void main(String[] args) {
        //Instanciando um contato e preenchendo seus atributos
        Contato contato= new Contato();
        contato.setNome("Fulano");
        contato.setEmail("fulano@ffsd.br");
        contato.setEndereco("Rua Piauí, 25");
        //opcional
        Calendar dataNascimento = Calendar.getInstance();
        try {
            java.util.Date data = new SimpleDateFormat("dd/MM/yyyy").parse("05/10/1992");
            dataNascimento.setTime(data);
            contato.setDataNascimento(dataNascimento);
        } catch (ParseException e) {
            System.out.println("Erro ao converter data");
        }
        //Instanciando um dao
        ContatoDAO dao = new ContatoDAO();
        //Adicionar o contato ao banco de dados
        dao.adiciona(contato);
    }
}
```

4) Teste o programa
5) Verifique, no banco, se o seu Contato foi adicionado.

❖ 6 – Pesquisando no Banco de Dados

Para pesquisar também utilizamos a interface PreparedStatement. A diferença é que usaremos método executeQuery que retorna todos os contatos de uma determinada Query.

O objeto retornado é uma implementação da interface ResultSet que permite navegar pelos registros através do método next. Esse método irá retornar false quando chegar ao fim da pesquisa, portanto ele é normalmente utilizado para fazer um loop nos registros como no exemplo a seguir:

```
// pega a conexão
Connection conexao = new ConnectionFactory().getConexao();
// pega o Statement
PreparedStatement stmt = conexao.prepareStatement("select * from contatos");

// executa um select
ResultSet rs = stmt.executeQuery();

// itera no ResultSet
while (rs.next()) {
    ;
}
//Fecha o ResultSet
rs.close();
//Fecha o Statment
stmt.close();
```

Chamando um dos métodos get do ResultSet (o mais comum é o getString), podemos obter o valor de uma coluna no banco de dados.

```
// pega a conexão
Connection conexao = new ConnectionFactory().getConexao();
// pega o Statement
PreparedStatement stmt = conexao.prepareStatement("select * from contatos");

// executa um select
ResultSet rs = stmt.executeQuery();

// itera no ResultSet
while (rs.next()) {
    System.out.println(rs.getString("nome")+" :: "+rs.getString("email"));
}
//Fecha o ResultSet
rs.close();
//Fecha o Statment
stmt.close();
```

Assim como o cursor do banco de dados, só é possível mover para frente. Para permitir um processo de leitura para trás é necessário especificar na abertura do ResultSet que tal cursor deve ser utilizado.

Mais uma vez podemos aplicar as idéias de DAO e criar um método getLista() no nosso ContatoDAO. Mas o que esse método retornaria? Um ResultSet? E teríamos o código de manipulação de ResultSet espalhado por todo o código? Vamos fazer nosso getLista() devolver algo mais interessante, uma lista de Contato:

```
List<Contato> contatos = new ArrayList<Contato>();
String sql = "select * from contatos";
PreparedStatement s;
s = conexao.prepareStatement(sql);
ResultSet r = s.executeQuery();
while (r.next()) {
    Contato c = new Contato();
    c.setNome(r.getString("nome"));
    c.setEmail(r.getString("email"));
    c.setEndereco(r.getString("endereco"));
    Calendar data = Calendar.getInstance();
    data.setTime(r.getDate("dataNascimento"));
    c.setDataNascimento(data);

    contatos.add(c);
}
r.close();
s.close();
return contatos;
```

Exercícios: Listagem

1) Crie o método getLista na classe ContatoDAO. Import o List do Java.util:

```
public List<Contato> getLista() {
    try {
        if (conexao.isClosed())
            conexao = new ConnectionFactory().getConexao();
    } catch (SQLException e1) {
        System.out.println("Erro ao obter conexao." + e1.getMessage());
    }
    this.comandoSQL = "SELECT * FROM contatos";
    List<Contato> contatos = new ArrayList<Contato>();
    try {
        this.stmt = conexao.prepareStatement(comandoSQL);
        // Obtenho os dados e guardo em um ResultSet
        ResultSet rs = stmt.executeQuery();
        while (rs.next()) {
            Contato c = new Contato();
            c.setNome(rs.getString("nome"));
            c.setEmail(rs.getString("email"));
            c.setEndereco(rs.getString("endereco"));
            c.setId(rs.getInt("id"));
            Calendar dataEmCalendar = Calendar.getInstance();
            dataEmCalendar.setTime(rs.getDate("dataNascimento"));
            c.setDataNascimento(dataEmCalendar);
            contatos.add(c);
        }
        return contatos;
    } catch (SQLException e) {
        System.out.println("Não foi possivel listar" + e.getMessage());
        return null;
    } finally {
        try {
            stmt.close();
            conexao.close();
        } catch (SQLException e) {
            System.out.println("Não foi possível liberar os recursos. "
                    + e.getMessage());
        }
    }
}
```

A saber:
List → Interface para coleções do pacote java.util
ArrayList e Vector → Classes (também do pacote Java.util) que implementam a interface List.
<Contato> → Generics (veremos mais adiante) define um tipo de objeto específico para nossa List para que não tenhamos que fazer casting mais adiante.

2) Agora vamos usar o método getLista para listar todos os contatos do nosso banco de dados.
3) Crie uma classe chamada TestaLista com um método main:
    a) Crie um ContatoDAO:

        ContatoDAO dao = new ContatoDAO();

    b) Liste os contatos com o DAO:

        List<Contato> contatos = dao.getLista();

    c) Itere nessa lista e imprima as informações dos contatos:

```
List<Contato> contatos = dao.getLista();
for (Contato c : contatos) {
    System.out.println();
    System.out.println("Id: "+c.getId());
    System.out.println("Nome: "+c.getNome());
    System.out.println("E-mail: "+c.getEmail());
    System.out.println("Endereço: "+c.getEndereco());
    //formatando a data (coisa chata!!!!)
    SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
    String dataEmTexto = sdf.format(c.getDataNascimento().getTime());
    System.out.println("Nascimento: "+dataEmTexto);
}
```

4) Rode o programa que acabamos de escrever. Clique em Run → Run as → Java Application.

A saber:

- O Hibernate é a ferramenta ORM que domina o mercado atualmente;
- Se um projeto não usa nenhuma das tecnologias ORM disponíveis no mercado, o mínimo a se fazer é usar o DAO.

Mais exercícios:

1) Pesquise sobre a classe SimpleDateFormat.
2) Crie uma classe DAOException que estenda de RuntimeException e utilize-a no seu ContatoDAO.
3) Utilize a cláusula WHERE para refinar suas pesquisas no banco de dados.
4) Crie o método pesquisarId, que receberá um id (inteiro) e retornará um objeto do tipo Contato.
5) Crie um método para fazer uma busca aproximada por nome, chamado pesquisarPorNome.

❖ 7 – Adicionando outros métodos ao DAO:

Já sabemos que o PreparedStatement pode executar qualquer tipo de código SQL e que ResultSet recebe dados retornados de sua consulta. Escrever código de diferentes métodos de uma classe típica de DAO fica fácil, porém maçante.

Veja o método altera:

```
public void altera(Contato c) {
    try {
        if (conexao.isClosed())
            conexao = new ConnectionFactory().getConexao();
    } catch (SQLException e1) {
        System.out.println("Erro ao obter conexao." + e1.getMessage());
    }
    this.comandoSQL = "UPDATE contatos SET nome=?,email=?,endereco=?,dataNascimento=? WHERE id=?";
    try {
        this.stmt = this.conexao.prepareStatement(comandoSQL);
        stmt.setString(1, c.getNome());
        stmt.setString(2, c.getEmail());
        stmt.setString(3, c.getEndereco());
        // Convertendo a data para gravar
        java.sql.Date dataParaGravar = new java.sql.Date(c
                .getDataNascimento().getTimeInMillis());
        stmt.setDate(4, dataParaGravar);
        stmt.setInt(5, c.getId());
        stmt.execute();
        System.out.println("Contato " + c.getId()
                + " alterado com sucesso!");// ferindo o SRP
    } catch (SQLException e) {
        System.out.println("Erro ao alterar" + e.getMessage());
    } finally {
        try {
            stmt.close();
            conexao.close();
            System.out.println(conexao);
        } catch (SQLException e) {
            System.out.println("Não foi possivel liberar os recursos"
                    + e.getMessage());
        }
    }
}
```

O código para remoção baseia-se em um Contato, mas usa apenas o id dele para executar o comando delete:

```
public void remove(Contato c) {
    try {
        if (conexao.isClosed())
            conexao = new ConnectionFactory().getConexao();
    } catch (SQLException e1) {
        System.out.println("Erro ao obter conexao." + e1.getMessage());
    }
    this.comandoSQL = "DELETE FROM contatos WHERE id=?";
    try {
        this.stmt = this.conexao.prepareStatement(comandoSQL);
        stmt.setInt(1, c.getId());
        stmt.execute();
        System.out.println("Contato " + c.getId()
                + " removido com sucesso!");// ferindo o SRP
    } catch (SQLException e) {
        System.out.println("Erro ao remover" + e.getMessage());
    } finally {
        try {
            stmt.close();
            conexao.close();
            System.out.println(conexao);
        } catch (SQLException e) {
            System.out.println("Não foi possivel liberar os recursos"
                    + e.getMessage());
        }
    }
}
```

Exercícios com DAO:

1) Implemente os métodos altera e remove escritos acima.

2) Use os métodos acima para fazer testes. Altere e remova um Contato. Verifique as modificações no banco de dados.
3) Crie um novo projeto (Java Project) no Eclipse, com o nome de projeto-empresa.
4) Crie, no pacote br.com.cefet.modelo a classe Departamento com os campos id (Long) e descrição (String).
5) Crie a base de dados empresa e a tabela departamentos no banco de dados.
6) Crie, no pacote br.com.cefet.modelo.dao, a classe DepartamentoDAO.
7) Crie a classe TestaDepartamentoDAO (contendo um método main) no pacote br.com.cefet.testes.
8) Na classe criada acima use a classe DepartamentoDAO, com os 5 métodos vistos anteriormente e/ou outros que julgar necessário, para instanciar novos departamentos e colocá-los no seu banco de dados.
9) Crie, no pacote br.com.cefet.modelo, a classe Funcionário com os campos id (Long), nome, login e senha (String) e departamento (Departamento).
10) Crie a tabela na base de dados empresa.
11) Crie uma classe DAO para Funcionário.
12) No pacote br.com.cefet.testes, crie a classe TestaFuncionarioDAO e use a classe FuncionarioDAO, com os 5 métodos vistos anteriormente e/ou outros que julgar necessário, para instanciar novos funcionários e colocá-los no seu banco de dados.