DESIGN PATTERNS II

CONNECTION FACTORY

Connection Factory é um padrão de criação do tipo FACTORY, utilizado para estabelecer conexão com bancos de dados, através de String de conexão (tipo de banco de dados, endereço do banco, schema, usuário e senha) implementada no método estático getConnection da classe DriverManager.

```
return DriverManager.getConnection
("jdbc:mysql://localhost:3306/contatos", "root", "root");
```

"O Factory Method define uma interface para criar um objeto, mais deixa que as subclasses decidam qual classe instânciar. O Factory Method permite a uma classe adiar a instânciação às subclasses."

O método getConnection é uma fábrica de conexões, isto é, ele cria novas conexões para nós. Basta invocar o método e recebemos uma conexão pronta para uso, não importando de onde elas vieram e eventuais detalhes de criação.

DAO (DATA ACCESS OBJECT)

Data Access Object é um padrão de projeto, utilizado para persistência em banco de dados. O DAO busca dados no banco e transforma em objetos e vice versa, mantendo uniformidade no código e evitando que o acesso e manipulação aos dados fique espalhado por toda aplicação. A classe RegistroDAO implementa em seu construtor um objeto do tipo Connection Factory que será utilizado junto do método PreparedStatement para execução de Querys no banco de dados utilizado.

JAVABEANS

JavaBeans são classes que possuem o construtor sem argumentos e métodos de acesso do tipo get e set. A classe Registro é um exemplo de JavaBens, com variáveis e métodos get e set para cada das variáveis.

Podem ser definidos como sendo componentes de software que permitem a geração de partes reutilizáveis. Seu objetivo e deixar o programador trabalhar mais nas regras de negócios.

ITERATOR

O Iterator é um padrão de Comportamento que tem por objetivo prover uma forma de acessar sequencialmente os elementos de uma coleção sem expor sua representação interna. Existem formas simples de iterar com os dados

que podem ser utilizadas através do FOR normal e do FOREACH, como os exemplos abaixo:

PS. O Iterator é um padrão poderoso que encapsula estruturas de dados de diversas formas, sendo que uma árvore pode ser varrida em ordem, em pré-ordem e em pós-ordem. Podemos ter um iterador como filtro que só retorna certos elementos, por exemplo.

```
import java.util.ArrayList;
import java.util.Iterator;a em ordem
import java.util.List;

public class Iterator_Test {

	public static void main(String[] args) {
		Iterator_Test test = new Iterator_Test();
		test.listarComFor();
		test.listarComForeach();
		test.listarComIterator();
	}

	List<String> lista;

	public Iterator_Test() {
		lista = new ArrayList<String>();
		lista.add("Pedro");
		lista.add("João");
		lista.add("Maria");
		lista.add("Marcos");
		lista.add("Anna");
		lista.add(null);
		lista.add(null);
	}

//interessante para pesquisar em arrays primitivos, por exemplo
	public void listarComFor() {

		for (int i = 0; i < lista.size(); i++) {
			System.out.println(lista);
		}
	}

// interessante para realizar pesquisas, por exemplo.
```

```
    public void listarComForeach() {
        for (String string : lista) {
            System.out.println(string);
        }
    }

//interessante quando é necessário realizar exclusões durante a iteração, por exemplo.
    public void listarComIterator() {
        Iterator<String> it = lista.iterator();
        while (it.hasNext()) {
            String nome = it.next();
            if (nome.equals(null)) {
                it.remove();
            }
            System.out.println(nome);
        }
    }
}
```

Referências

Freeman, Eric; Freeman, Elisabeth; Sierra, Kathy; Bates, Bert. Use a cabeça! Padrões de Projetos. Ed. Alta Books, Rio de Janeiro: 2007.

Caelum ensino e Inovação: http://www.caelum.com.br/