Capítulo 1

Acesso a Banco de Dados com JDBC (Java Database Connectivity) e o Padrão de Projeto DAO (Data Access Object)

Introdução

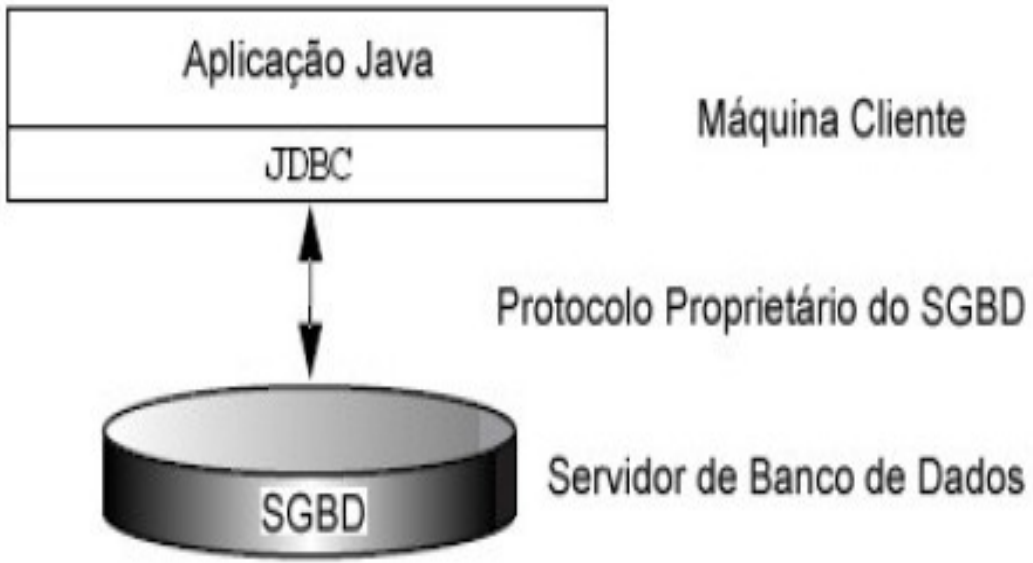
- Os programas desenvolvidos em Java comunicam-se com um banco de dados e manipulam seus dados utilizando a API JDBC.

- Um driver JDBC permite aos aplicativos java conectar-se a um banco de dados em particular e aos programadores manipular essa base de dados através da API do JDBC.

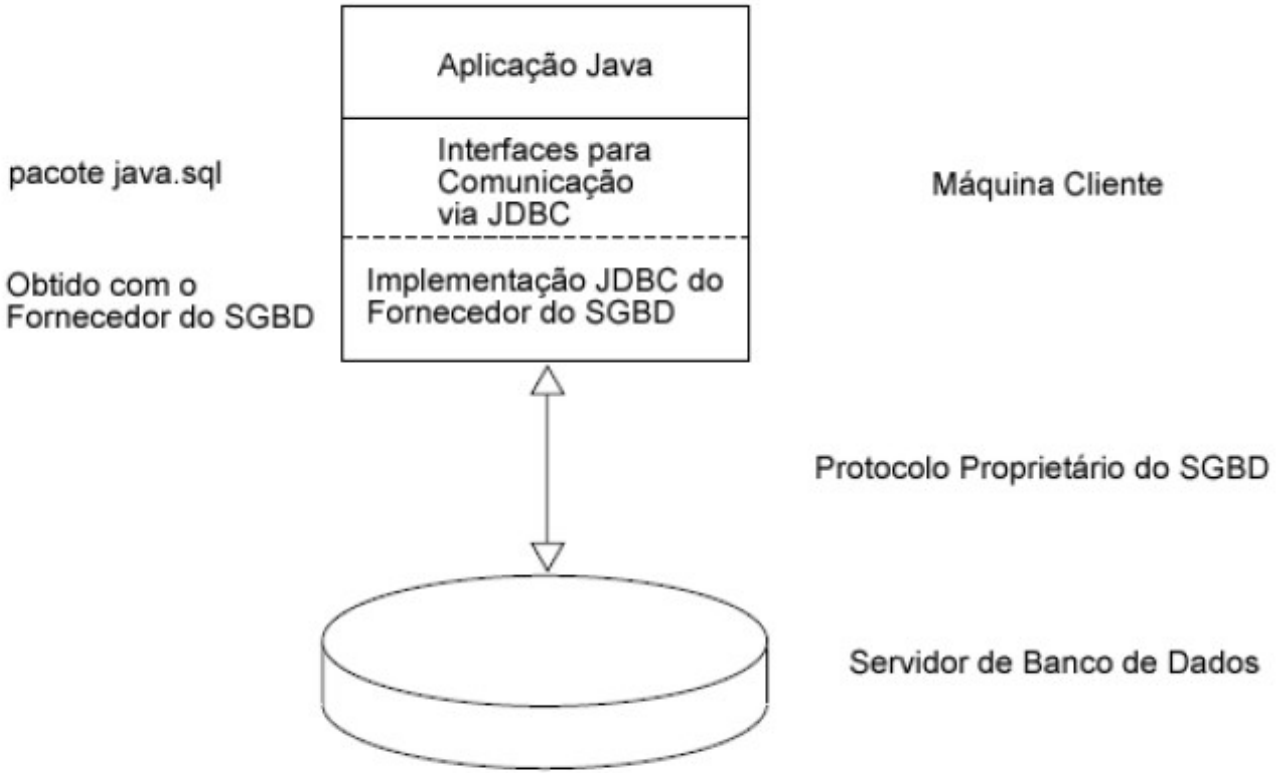
- A separação entre API do JDBC e drivers de bancos de dados particulares permite aos desenvolvedores alterar o banco de dados subjacentes sem modificar o código java que acessa o banco de dados.

- Os SGBDs mais populares fornecem seus drivers JDBC e também existem muitos drivers independentes disponíveis.

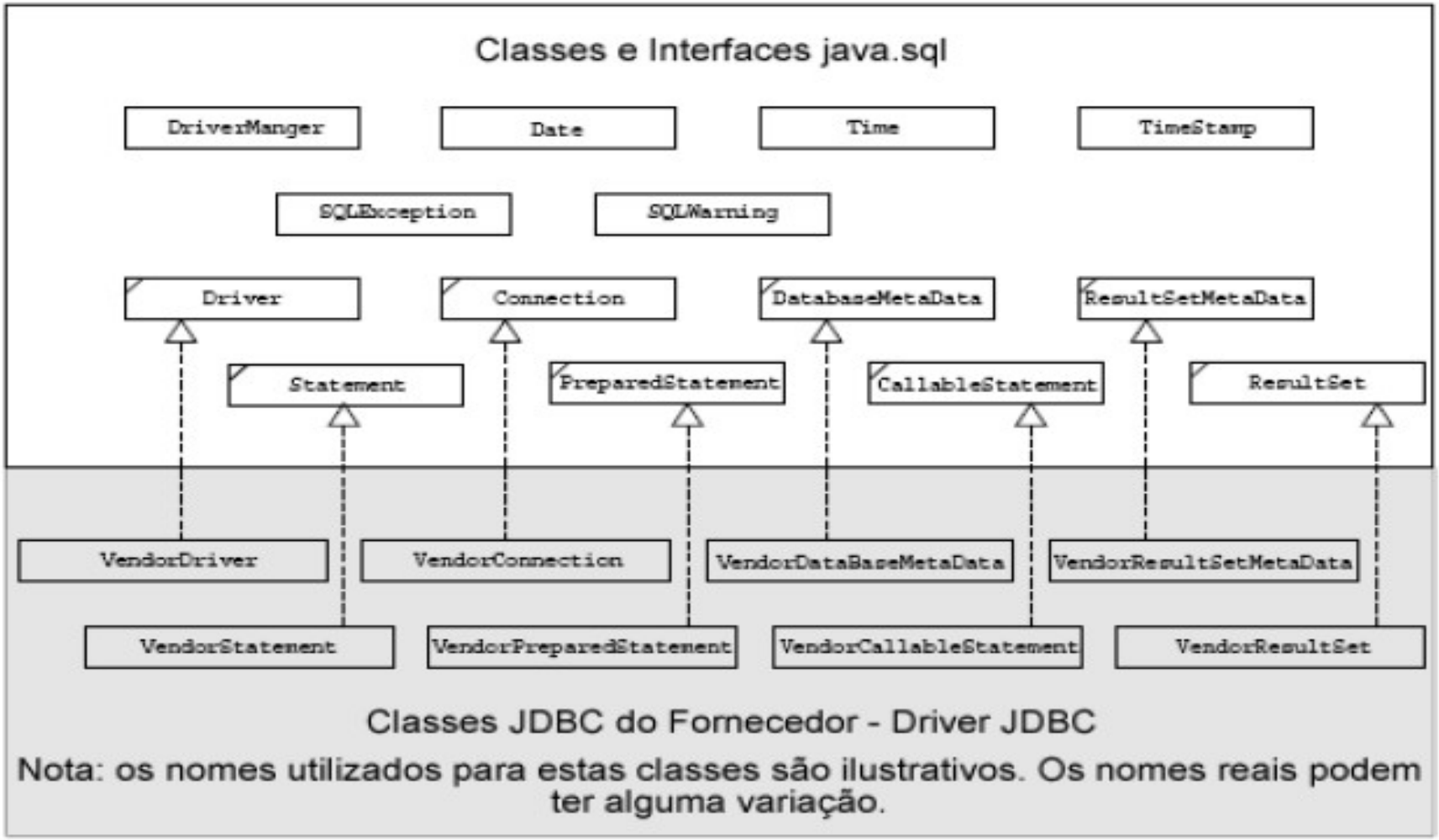
Comunicação Através de JDBC



Os Componentes JDBC



Componentes JDBC em Detalhes



Classificação Funcional da API JDBC

Categoria	*Classe ou Interface*	*Fornecido por*	*Comentario*
Conexão com uma fonte de dados	Classe DriverManager	java.sql (Sun)	Gerencia um conjunto de drivers JDBC
	Interface Driver	Fabricante do SGBD	Fornece comunicacão com o banco de dados
	Interface Connection	Fabricante do SGBD	Gerencia uma sessão com o bancode dados
Envio de instrucões SQL	Interface Statement	java.sql	Instrucão SQL estática Objeto empacotador
	Interface PreparedStatement	java.sql	Instrucão SQL pré-compilada Objeto empacotador
	Interface CallableStatement	java.sql	Procedimento armazenado Objeto empacotador
Recuperacão de resultados de uma consulta	Interface ResultSet	java.sql	ResultSet do Banco de Dados Objeto empacotador

Classificação Funcional da API JDBC (Cont.)

Categoria	*Classe ou Interface*	*Fornecido por*	*Comentario*
Mapeamento entre tipos de dados SQL e tipos de dados Java	Classe Date	java.sql	Valor SQL para data
	Classe Time	java.sql	Valor SQL para horario
	Classe TimeStamp	java.sql	Valor SQL para “TimeStamp”
Fornecimento de metadados sobre o banco de dados	Interface DatabaseMetadata	Fabricante do SGBD	Informacão sobre o banco de dados Objeto empacotador
	Interface ResultSetMetadata	Fabricante do SGBD	Colunas ResultSet Objeto empacotador de informacão
Geracao de Excecões	Classe SQLException	java.sql	Erros de acesso ao banco de dados
	Classe SQLWarning	java.sql	Alertas de acesso ao banco de dados

Conexão Através da Interface JDBC – Visão Geral

1. Registre o driver JDBC fornecido pelo fabricante do mysql com a classe DriverManager
   Class.forName(“com.mysql.jdbc.Driver”);

2. Estabeleca uma sessão com o SGBD.
   String url = "jdbc:mysql://localhost:3306/facebook”
   Connection con = DriverManager.getConnection(url, “usuario”, “senha”)

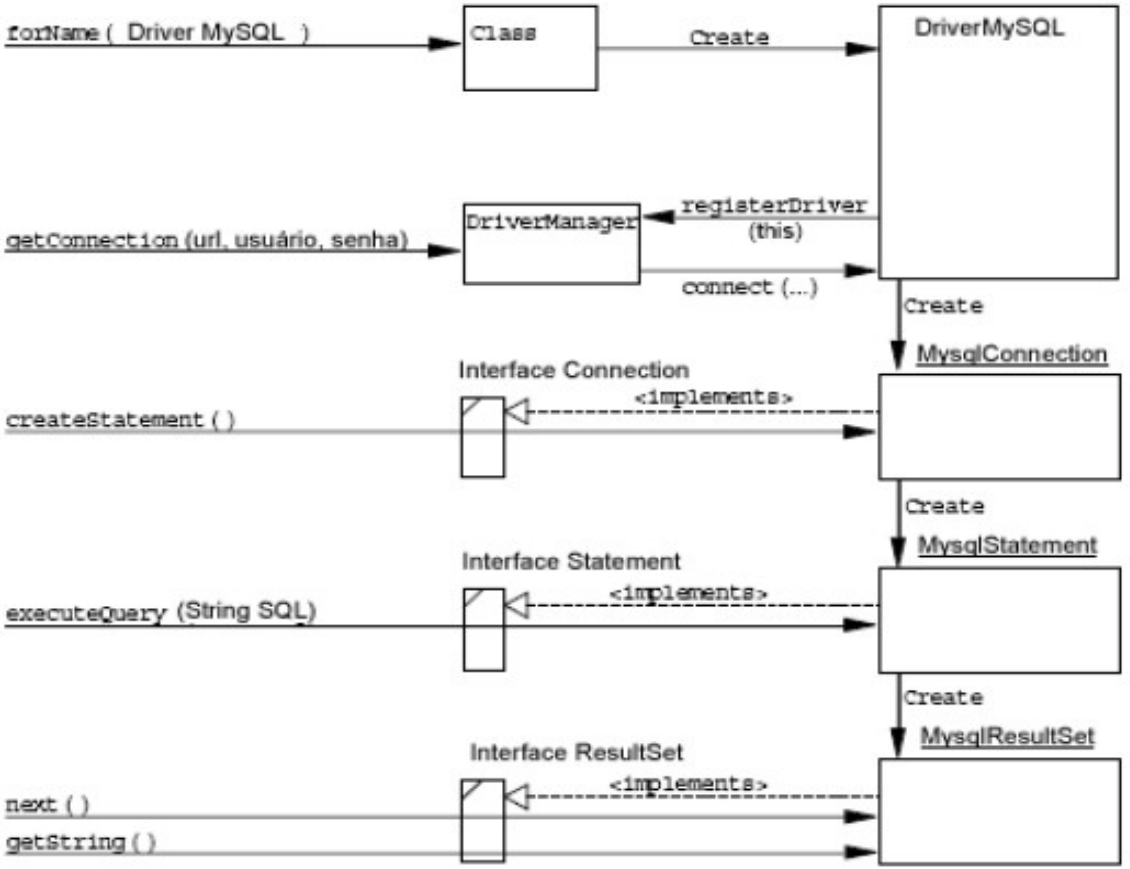
3. Crie um objeto empacotador SQL.
   Statement stmt = con.createStatement();

4. Envie uma consulta e receba um resultado
   String query = "SELECT * FROM PARTICIPANTE";
   ResultSet rs = stmt.executeQuery(query);

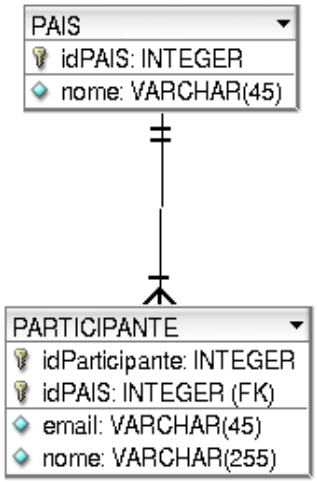
5. Extrai os dados do objeto empacotador com o resultado

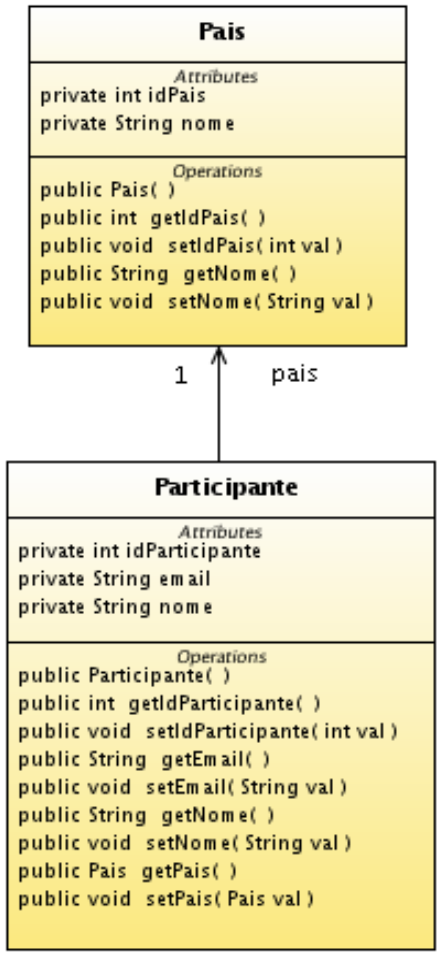
```
while (rs.next()) {
    System.out.println(" IDPARTICIPANTE  : " + rs.getString("idParticipante").trim());
    System.out.println(" IDPAIS   :"+ rs.getString("IDPAIS").trim());
    System.out.println(" NOME: " + rs.getString("NOME").trim());
    System.out.println("");
}
```

Diagrama de Seqüência JDBC



Demonstracão - ResultSet





Arquivos: ExemploResultSet.java

obs: o código deste exemplo bem como o script para criacão das tabelas está disponível no site http://www.pagliares.com.br

Mapeamento de Tipos de Dados SQL para Tipo de Dados Java

Tipo de Dado SQL	*Tipo de Dado Java*
CHAR	STRING
VARCHAR	STRING
LONGVARCHAR	STRING
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	java.math.BigDecimal
TINYINT	boolean
SMALLINT	byte
INTEGER	short

Mapeamento de Tipos de Dados SQL para Tipo de Dados Java (Cont.)

Tipo de Dado SQL	*Tipo de Dado Java*
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte []
VARBINARY	byte []
LONGVARBINARY	byte []
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Os Métodos getXXX de ResultSet

Metodo	*Tipo Java Retornado*
getASCIIStream	java.io.InputStream
getBigDecimal	java.math.BigDecimal
getBinaryStream	java.io.InputStream
getBoolean	boolean
getByte	byte
getBytes	byte []
getDate	java.sql.Date
getDouble	double
getFloat	float

Os Métodos getXXX de ResultSet(Cont.)

Metodo	*Tipo Java Retornado*
getInt	int
getLong	long
getObject	Object
getShort	short
getString	java.lang.String
getTime	java.sql.Time
getTimestamp	java.sql.Timestamp
getUnicodeStream	java.io.InputStream de caracteres Unicode

Os Métodos setXXX de ResultSet

Metodo	*Tipos SQL*
setASCIIStream	LONGVARCHAR produzido por um stream ASCII
setBigDecimal	NUMERIC
setBinaryStream	LONGVARBINARY
setBoolean	BIT
setByte	TINYINT
setBytes	VARBINARY ou LONGVARBINARY(dependendo do tamanho relativo a VARBINARY)
setDate	DATE
setDouble	DOUBLE
setFloat	FLOAT

Os Métodos setXXX de ResultSet(Cont.)

Metodo	*Tipos SQL*
setInt	INTEGER
setLong	BIGINT
setObject	O objeto que é convertido para o tipo SQL destino antes de ser enviado
setNull	NULL
setShort	SMALLINT
setString	VARCHAR ou LONGVARCHAR (dependendo do tamanho relativo ao limite do driver sobre VARCHAR)
setTime	TIME
setTimestamp	TIMESTAMP

Prepared Statements

- JDBC fornece uma maneira de executarmos comandos SQL parametrizados
  1. Interface PreparedStatement, subclasse de Statement.
- Vantagens de PreparedStatement
  1. código mais legível
  1. melhor performance pois os comandos são pré-compilados e otimizados.
  1. ganhos de seguranca minimizando ataques de injecão de SQL.
- Com PreparedStatement, a string contendo o comando SQL deve ser fornecida no momento de criacão do comando e não no momento de execucão do comando:

```
...
Connection con = null;
PreparedStatement st = con.prepareStatement("Select * FROM pessoa where nome = ?");
st.setString(1, "Rodrigo");
resultado = st.executeQuery( );
...
```

Prepared Statements(Cont.)

- Os parâmetros de um PreparedStatement são indicados por sinais de interrogacão dentro da string que contém o comando SQL.
  1. A interface PreparedStatement fornece uma série de métodos do tipo setInt, setDate, etc. que permitem passar os valores dos parâmetros antes da execucão

- Os parâmetros são identificados pela sua posicão dentro da string SQL, iniciando por 1

- Devido às vantagens de se ter o código mais simples e mais seguro, a maioria dos desenvolvedores prefere usar PreparedStatement o tempo todo, mesmo para comandos SQL que serão executados uma única vez e onde portanto não haveria ganho de performance

Demonstracão - PreparedStatement

Arquivos: ResultSetComPreparedeStatement.java

obs: o código deste exemplo está disponível no site http://www.pagliares.com.br

A Interface RowSet

- A interface RowSet fornece vários métodos set que permitem ao programador especificar as propriedades necessárias para estabelecer uma conexão e criar um Statement.

- Há dois tipos de RowSet – conectado e desconectado.

- Um objeto RowSet **conectado** conecta-se ao banco de dados uma vez e permanece conectado até que o aplicativo termine.

A Interface RowSet (cont.)

- Um objeto RowSet **desconectado** conecta-se ao banco de dados, executa uma consulta para recuperar os dados do banco e depois fecha a conexão.

- A interface RowSet possui duas subinterfaces – JdbcRowSet e CachedRowSet.

  1. A interface JdbcRowSet é um RowSet conectado.
  1. A interface CachedRowSet é um RowSet desconectado.

Demonstracão - RowSet

Arquivos: ExemploRowSet.java

obs: o código deste exemplo está disponível no site http://www.pagliares.com.br

Padrões de Projeto de Software Orientado a Objetos

- A idéia de padrões foi apresentada por Christopher Alexander em 1977 no contexto de Arquitetura (de prédios e cidades).

- Cada padrão descreve um problema que ocorre repetidamente, de novo e de novo, em nosso ambiente, e então descreve a parte central da solução para aquele problema de uma forma que você pode usar esta solução um milhão de vezes, sem nunca implementá-la duas vezes da mesma forma.

- Um padrão de projeto é uma solução geral para um problema recorrente no desenvolvimento de software orientado a objetos.

- Um padrão de projeto é uma espécie de gabarito para como resolver um problema, ou melhor dizendo, é uma solução elegante na resolução de problemas.

O Padrão Data Access Object (DAO)

- A maioria das aplicações empresariais usa normalmente sistemas de gerenciamento de bancos de dados relacional (RDBMS, relational database management system) como armazenamento persistente.

- Entretanto, os dados empresariais podem residir em outros tipos de repositórios, como mainframes ou sistemas legados, repositórios LDAP (Lightweight Directory Access Protocol), bancos de dados orientados a objetos (OODB, object-oriented databases) e arquivos simples.

- Misturar a lógica de persistência com a lógica de aplicação cria uma dependência direta entre a implementação da aplicação e do armazenamento persistente.

O Padrão Data Access Object (DAO) (Cont.)

- Tal dependência de código nos componentes torna difícil e maçante migrar a aplicação de um tipo de fonte de dados para outro.

- Quando as fontes de dados são alteradas, os componentes devem ser modificados para tratar o novo tipo de fonte de dados.

- Use um Data Access Object para abstrair e encapsular todo acesso ao armazenamento persistente.

- O Data Access Object gerencia a conexão com a fonte de dados para obter e armazenar dados.

Use o Padrão Data Access Object Quando:

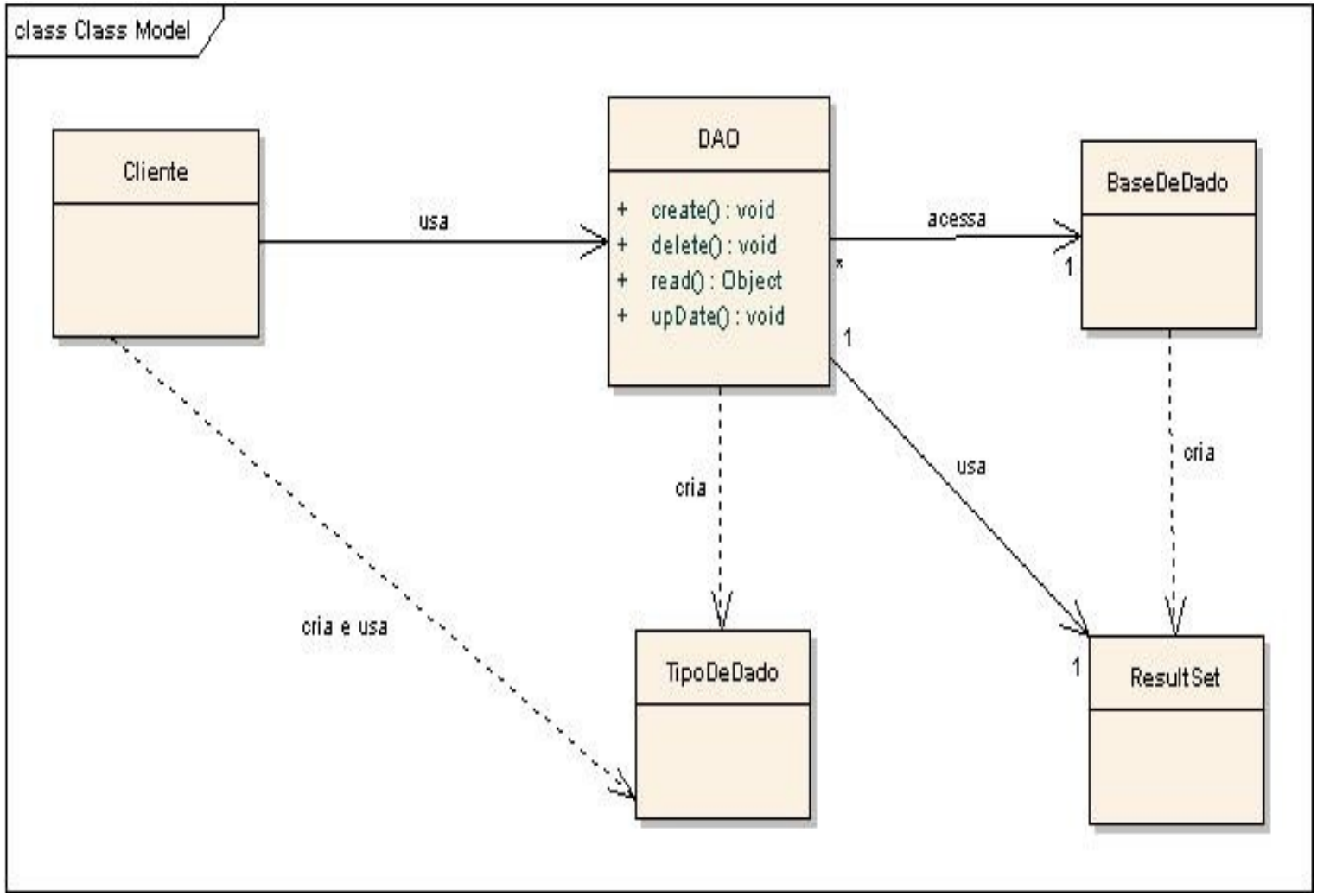
- O principal objetivo de um DAO é encapsular o acesso e a manipulação de dados em uma camada separada;
- Você deseja implementar os mecanismos de acesso a dados para acessar e manipular dados em um armazenamento persistente;
- Você deseja desacoplar a implementação do armazenamento persistente do restante da aplicação;

Use o Padrão Data Access Object Quando (Cont.)

- Você deseja fornecer uma API de acesso uniforme aos dados para um mecanismo persistente para vários tipos de fontes de dados, como repositórios RDBMS, LDAP, OODB, XML, arquivos simples e assim por diante;

- Você deseja organizar os recursos de lógica de acesso a dados e encapsular recursos proprietários para facilitar a capacidade de manutenção e a portabilidade.

A Estrutura Do Data Access Object (DAO)



Os Participantes Do Data Access Object (DAO)

- ***Cliente*** – o cliente é um objeto que requer acesso à fonte de dados para obter e armazenar dados.
- ***DAO*** – o DAO é o objeto de função principal desse padrão. Ele abstrai a implementação de acesso a dados subjacente para o cliente a fim de permitir um acesso transparente a fonte de dados.
- ***BaseDeDado*** – representa uma implementação de fonte de dados.
- ***ResultSet*** – representa os resultados de uma execução de consulta.
- ***TipoDeDado*** – representa um objeto de transferência usado como um carregador de dados.

Demonstracão - DAO

Arquivos: DAO.java, Pais.java, PaisDAO.java, BaseDeDados.java, TestaSelectPaisDAO.java, TestaInsertPaisDAO.java, TestaUpdatePaisDAO.java, TestaDeletePaisDAO.java, Participante.java, ParticipanteDAO.java, TestaSelectParticipanteDAO.java, TestaInsertParticipanteDAO, TestaUpdateParticipanteDAO.java, TestaDeleteParticipanteDAO.java, DAOException.java

obs: o código deste exemplo está disponível no site http://www.pagliares.com.br

Conclusão

- Neste capítulo fomos apresentados à arquitetura da API do JDBC.

- Também vimos exemplos de como acessar e manipular uma base de dados MySQL via API do JDBC.

- Também aprendemos sobre a nova interface RowSet introduzida no J2SE 5.0.

- Introduzimos conceitos de padrões de projeto, em especial o padrão DAO,

Estudos Complementares - Obrigatórios

- MySQL Para Desenvolvedores Java – Revista Java Magazine, ed. 40. Código fonte dos exemplos disponível em http://www.devmedia.com.br

- JDBC de Ponta a Ponta (Parte 1) – Revista Java Magazine, ed. 41. Código fonte dos exemplos disponível em http://www.devmedia.com.br

- JDBC de Ponta a Ponta (Parte 2) – Revista Java Magazine, ed. 42. Código fonte dos exemplos disponível em http://www.devmedia.com.br

observacão: Além da leitura dos textos acima, é importante testar os códigos de exemplo !!!!

Perguntas?