Universidade do Minho
Departamento de Informática

Programação Por Camadas

Aula 2

DSS 08/09

JDBC - Comandos SQL pré-compilados

- Interface PreparedStatement
  - Permite "pré-compilar" comandos SQL
    - Optimização
    - Verificação prévia do comando
  - SubInterface de Statement
- Criação através de método definido na Interface Connection

```
PreparedStatement prepareStatement(String sql);
```

  - excepção: `SQLException`

JDBC - Comandos SQL pré-compilados

- Execução da query

```
ResultSet executeQuery();
int executeUpdate();
```

  - excepção: `SQLException`

  - Processo semelhante ao utilizado para instâncias de Statement, mas com a string SQL a ser passada como parâmetro na criação da instância.

JDBC - Comandos SQL pré-compilados

- Exemplo para comandos de selecção

```
// no início da aplicação
        ...
        sql = "SELECT nome FROM clientes WHERE numerário > 100000";
        st = con.prepareStatement(sql);
        ...


// na função que faz a consulta à base de dados
        ...
        res = st.executeQuery();
        ...
```

JDBC - Comandos SQL parametrizados

- Considere-se o comando SQL

```
INSERT INTO contas VALUES(2, "João", 15000);
```

- Este comando tem uma parte que se repete em todas as inserções, e uma parte variável

```
INSERT INTO contas VALUES( ?, ? ,?)
```

  - Substituímos as "variáveis" do comando por "?"

- Comando mais flexível:
  - pré-compilado (na criação da instância)
  - parâmetros posteriormente instanciados (na execução do comando)

JDBC - Comandos parametrizados

- Instanciação dos parâmetros
  - Família de métodos *setXXX* definidos na Interface PreparedStatement
  - `void setString(int index, String valor);`
  - `void setInt(int index, int valor);`
- Execução do comando
  - Idêntica ao caso dos comandos sem parâmetros

JDBC - Comandos parametrizados

- Exemplo

```
// na fase de arranque da aplicação  (ou quando
// se executar a primeira inserção)
     try {
          sql = "INSERT INTO clientes VALUES( ?,?,?)";
          s = con.prepareStatement(sql);
     }
     catch (SQLException e) {/* tratar excepção */}

//na execução do comando SQL
void insere(int num, String nome, int saldo) {

     try {
          s.setInt(1,num);
          s.setString(2,nome);
          s.setInt(3,saldo);
          s.executeUpdate();
     }
     catch (SQLException e) {/* tratar excepção */}
}
```

JDBC - Batch de Comandos

- Permite enviar uma sequência de comandos para a base de dados
  - restrito a UPDATE, INSERT e DELETE ou comandos DDL
  - Pode ser utilizado com Statement e PreparedStatement

// insere

```
void addBatch(String sql);          // Statement
void addBatch();                    // PreparedStatement
```

// executa

```
int[ ] executeBatch();
```

Excepções:

`SQLException` - erro no acesso à base de dados ou se o driver não suportar múltiplos comandos

BatchUpdateException - Caso algum dos comandos falhe

JDBC - Batch de Comandos

- Exemplo

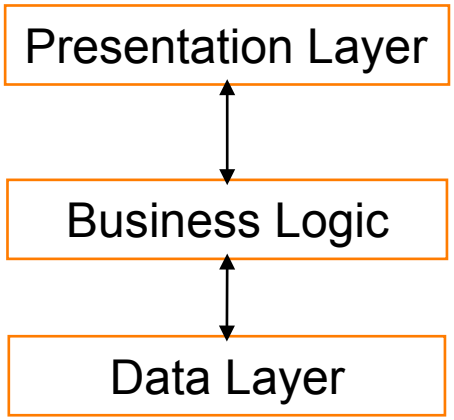
```
PreparedStatement s;
String sql = "UPDATE Prod SET preço = preço * 1.03 WHERE cod=?";
s = con.prepareStatement(sql);
...
s.setString(1,"023");
s.addBatch();

s.setString(1,"045");
s.addBatch();

int res[] = s.executeBatch();
```

Programação por Camadas

- Cada uma das camadas tem uma (ou mais) referência(s) para a camada abaixo
  - A camada de apresentação tem os métodos/funcionalidades que o utilizador pode invocar
  - A camada de negócio tem os algoritmos que permitem implementar a "lógica de negócio" - o que é particular e característico do problema
  - A camada de dados efectua o acesso aos dados e providencia métodos de acesso aos dados, preservando a independência da representação



Exercício

- Pretende-se implementar um sistema que permite duas operações:
  - criar conta (parâmetros: número conta; nome do titular; saldo)
    - O número de conta deve ser obtido automaticamente como sendo o número máximo + 1
    - A abertura de uma conta implica a cobrança de uma comissão cujo valor está armazenado na base de dados
  - consulta do total dos saldos das contas com saldo superior a X

⇒ **Que métodos é que fazem parte de cada camada?**

Exercício

- Estratégias para implementar o total de saldos (sem utilizar a função SUM do SQL) - factory pattern:
  - O data layer devolve uma lista de inteiros
  - O data layer devolve um iterador para uma lista de inteiros
  - O data layer implementa ele próprio um iterador
    - void factoryInitSaldos();
    - boolean factoryHasMoreSaldos();
    - int factoryGetNextSaldo();
    - O business layer invoca estas funções para percorrer a lista de resultados

Exercício

- Definir uma base de dados com duas tabelas:
  - contas (num,nome,saldo)
  - comissoes(descritivo,valor)
- Definir três classes para implementar as três camadas:
  - presentation
  - business
  - data
- A interface é simulada pela classe Main que se limita a invocar os métodos do presentation layer.