Departamento de Informática, Universidade do Minho

http://www.di.uminho.pt

Desenvolvimento de Sistemas de Informação

http://sim.di.uminho.pt/ensino.php3

LESI - 4º Ano / 2º Semestre (5308O7)
LMCC - 4º Ano / 2º Semestre (7008N8 - Opção II)

2003/2004

José Creissac Campos

jose.campos@di.uminho.pt

Aulas Teórico-Práticas

JDBC

Algumas classes, interfaces e métodos relevantes:

- classe `java.lang.Class`
  - `static Class forName(String nome)`
    método utilizado para inicializar o *driver*
    Exemplo: `Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");`
- classe `java.sql.DriverManager`
  - `static Connection getConnection(String url)`
    método para estabelecer uma ligação a uma base de dados
    Exemplo: `DriverManager.getConnection("jdbc:odbc:dbase");`
- interface `java.sql.Connection` — implementa uma ligação a uma base de dados (noção de sessão)
  - `Statement createStatement()` — método para criar um `Statement`
  - `void close()` — método para fechar a ligação
  - `boolean isClosed()` — método para testar se a ligação está fechada

- interface `java.sql.Statement` — permite interagir com a base de dados
  - `int executeUpdate(String sql)`
    permite executar INSERTs, UPDATEs e DELETEs
  - `ResultSet executeQuery(String sql)`
    permite executar SELECTs (resultado é devolvido num `ResultSet`)
  - `boolean execute(String sql)`
    permite executar *statement* SQL genéricos ou que possam devolver multiplos resultados (devolve true se primeiro resultado é um `ResultSet`)
  - `ResultSet getResultSet()`
    obter `ResultSet` de um `execute` (devolve null se não há mais ou se é *UpdateCount*)
  - `int getUpdateCount()`
    obter *UpdateCount* de um `execute` (devolde -1 se não há mais ou se é ResultSet)
  - `boolean getMoreResults()`
    obter o próximo resultado de um `execute` (devolve true se o resultado é um `ResultSet`)

- interface `java.sql.ResultSet` — uma tabela representado o *result set* de uma *query* à base de dados. Inicialmente o cursor está posicionado antes da primeira linha.
  - `boolean next()`
    move o cursor para a próxima linha (devolve false quando não há mais linhas)
  - X getX(int index)
    devolve o valor da coluna index (a primeira coluna é a 1; X = String, Int, Date, etc).
  - X getX(String nome)
    devolve o valor da coluna "nome" (X = String, Int, Date, etc).
- classe `java.sql.SQLException`
  classe utilizada para erros relativos ao acesso a base de dados

Meta-dados

Algumas interface e métodos para trabalhar com meta-dados:

- interface `java.sql.ResultSet`
  - `ResultSetMetaData getMetaData()`
    devolve o número, tipos e propriedades das colunas do `ResultSet`
- interface `java.sql.ResultSetMetaData`
  - `int getColumnCount()`
    devolve o número de colunas do `ResultSet`
  - `String getColumnName(int index)`
    devolve o nome da coluna na posição index
  - `String getColumnClassName(int index)`
    devolve o nome da classe Java correspondente ao tipo dos valores da coluna

Faça você mesmo...

```
static void display(ResultSet rs) throws SQLException {
  ...
}
```


```
static void display(ResultSet rs) throws SQLException {
  ResultSetMetaData rsmd;
  int cont, i;

  rsmd = rs.getMetaData();

  cont = rsmd.getColumnCount();

  for(i=1; i<=cont; i++)
    System.out.print(rsmd.getColumnName(i) + "\t");

  System.out.println();

  while(rs.next()) {
    for (i=1; i<=cont; i++)
      System.out.print(rs.getString(i) + "\t");
    System.out.println();
  }
}
```

Pré-processamento de comandos SQL

Algumas interface e métodos para trabalhar com pré-processamento:

- interface `java.sql.Connection`
  - `PreparedStatement prepareStatement(String sql)`
- interface `java.sql.PreparedStatement` (especialização de `Statement`)
  - `int executeUpdate()` — INSERT, UPDATE, DELETE
  - `ResultSet executeQuery()` — SELECT
  - `boolean execute()` — true → SELECT
  - `ResultSet getResultSet()` — null se não há mais ou se é *UpdateCount*
  - `int getUpdateCount()` — -1 se não há mais ou se é `ResultSet`
  - `boolean getMoreResults()` — obter o próximo resultado

Faça você mesmo...

Descubra se a combinação JDBC/Access permite pré-processamento.


```
String sql = "Isto não é SQL!";

try {
  st = con.prepareStatement(sql);
}
catch (SQLException e) {
  System.out.println("Pré-processamento suportado!");
}
```

Comandos SQL parametrizados

- Usar ? para representar parâmetros:

```
SELECT marca FROM Fornecedores WHERE nome = ?
```

- interface `java.sql.PreparedStatement`
  - `void setX(int index, X valor)` — X = String, Int, etc.

```
sql = "SELECT marca FROM Fornecedores WHERE nome = ?";
prepSt = con.prepareStatement(sql);

perpSt.setString(1,"Xpto");
rs1 = prepSt.executeQuery();

perpSt.setString(1,"Flda");
rs2 = prepSt.executeQuery();
```

Invocação de Stored Procedures

- SQL:
  CREATE PROCEDURE nome (listaDeParâmetros) AS sequênciaDeComandosSQL
- interface `java.sql.Connection`
  - `CallableStatment prepareCall(String nomeSP)`
    Exemplo: `cs = con.prepareCall("{call spTeste}");`
- interface `java.sql.CallableStatment` (especialização de `PreparedStatement`)
  - `int executeUpdate()` — um único comando INSERT, UPDATE, DELETE no SP
  - `ResultSet executeQuery()` — um único comando SELECT no SP
  - `boolean execute()` — true → SELECT
  - `ResultSet getResultSet()` — null se não há mais ou se é *UpdateCount*
  - `int getUpdateCount()` — -1 se não há mais ou se é `ResultSet`
  - `boolean getMoreResults()` — obter o próximo resultado

Envio de comandos em Batch

- Tipicamente utilizado para comandos de actualização (INSERT/UPDATE);
- Suportado por `Statement`, `PreparedStatement`, `CallableStatement`
  - `void addBatch(String sql)`
    adiciona um comando SQL totalmente especificado ao *batch* (`Statement`)
  - `void addBatch()`
    adiciona um comando SQL parameterizado ao *batch* (`PreparedStatement`)
  - `int[] executeBatch()`
    executa o *batch* (resultado: número de linhas afectadas por cada comando; -2 se não se sabe; -3 se falhou)
  - `void clearBatch()`
    esvazia o *batch*

Mais sobre ResultSet

- `Connection`
  - `Statement createStatement(int tipo, int concorrência)`
  - tipo: TYPE_FORWARD_ONLY, TYPE_SCROLL_INSENSITIVE, TYPE_SCROLL_SENSITIVE
  - concorrência: CONCUR_READ_ONLY, CONCUR_UPDATABLE
- `ResultSet`
  - `boolean first()`
  - `boolean last()`
  - `boolean previous()`
  - `boolean absolute(int row)`
  - `boolean relative(int row)`
  - `void insertRow()`
  - `void deleteRow()`
  - `void refreshRow()`
  - `void updateX(int coluna, X valor)`
  - `void updateRow()`

Transacções

- `Connection`
  - `void setAutoCommit(boolean commit)`
  - `boolean getAutoCommit()`
  - `commit()`
  - `void rollback()`