Especialização em Engenharia de Software

Programação Orientada a Objetos
JDBC – Java Database Connectivity

Sérgio Soares
scbs@cin.ufpe.br

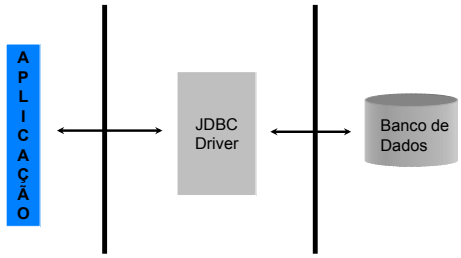
Objetivos

- Apresentar os conceitos básicos da especificação de Java Database Connectivity.
- Utilizar os conceitos na prática para integrar aplicações Java com bancos de dados.

O quê é JDBC

- Especificação de Java Database Connectivity
- API para acesso a SGBDs em Java
  - "envia comandos SQL para o SGBD e processa os resultados como tipos de dados em Java"
- Baseada em interfaces simples
  - `Driver`
  - `Connection`
  - `Statement`
  - `ResultSet`

Arquitetura JDBC



Integrando Java com Banco de Dados

- Etapas da integração
  - instalar o(s) driver(s) JDBC na máquina e configurar o ambiente de execução
  - registrar o(s) driver(s) JDBC
  - estabelecer uma conexão com o SGBD
  - submeter a execução de comandos SQL ao SGBD
  - processar os resultados
  - encerrar a conexão

Integrando Java com Banco de Dados

- Pacote **`java.sql`**
  - interfaces
    - **`Driver`**
    - **`Connection`**
    - **`Statement`**
    - **`ResultSet`**
    - **`PreparedStatement`**
    - ...
  - classes
    - **`DriverManager`**
    - **`Date`**
    - **`Time`**
    - **`Timestamp`**
    - ...

Registrando drivers JDBC

- Carregue o(s) *driver(s)* JDBC necessários à aplicação

```
Class.forName(nome_do_driver);
```

  - o nome do *driver* é fornecido pelo provedor do BD
- Exemplo:

```
Class.forName("com.mysql.jdbc.Driver");
```


Criando uma conexão

- JDBC referencia bancos de dados individualmente através do formato da URL
  - jdbc : *<subprotocol>* : *<subname>*
    - —subprotocol: nome do driver
    - —subname: nome e caminho da base de dados
  - Exemplos de url:

```
jdbc:mysql://host:port/database
jdbc:oracle:thin:@host:port:database
jdbc:db2://host:port/database
```

O formato da URL varia com o fornecedor do driver. É preciso consultar a documentação.

Criando uma conexão

- Utilize a classe `java.sql.DriverManager` para criar uma conexão com o SGBD

```
static Connection getConnection(
    String url,
    String user,
    String password) throws SQLException
```

- Exemplo

```
Connection con = DriverManager.
    getConnection(url,login, password);
```


java.sql.Connection

- Métodos básicos da API 1.0

```
Statement createStatement()
                throws SQLException
void setAutoCommit(boolean)
                throws SQLException
void commit() throws SQLException
void rollback() throws SQLException
void close() throws SQLException
```


Exemplo: criando conexão

```
Connection con = null;
Class.forName("com.mysql.jdbc.Driver");
String url = " jdbc:mysql://localhost:3306/mysql ";
String login = "scbs";
String senha = "****";

con = DriverManager.getConnection(url,login,senha);
```

- ODBC

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
String url = "jdbc:odbc:base-teste"
```


Executando comandos SQL

- Utilize a classe `java.sql.Statement` para enviar comandos SQL para o SGBD
  - crie um `Statement` a partir da conexão
- Exemplo:

```
Statement stm = con.createStatement();
```

java.sql.Statement

- `int executeUpdate(String sql) throws SQLException`
  - utilize para executar comandos *INSERT, UPDATE, DELETE, CREATE, DROP,* etc.)
- `ResultSet executeQuery(String sql) throws SQLException`
  - utilize para executar comandos *SELECT*
- `void close() SQLException`
- ...

Exemplo: criando uma tabela

```
Statement stmt = null;
String cmd = "CREATE TABLE usuarios (" +
             "email VARCHAR(60) NOT NULL PRIMARY KEY," +
             "nome VARCHAR(80) NOT NULL)";
try {
    stmt = con.createStatement();
    stmt.executeUpdate(cmd);
}
catch (SQLException ex) {...}
finally {
    if (stmt != null) {
        try {
            stmt.close();
        } catch(SQLException ex) {...}
    }
}
```


Inserindo dados na tabela

```
Statement stmt = null;
String cmd = "INSERT INTO usuarios " +
             "VALUES (\'sergio@dsc.upe.br\'," +
              "\'Sergio Soares\')";
try {
    stmt = con.createStatement();
    stmt.executeUpdate(cmd);
}
catch (SQLException ex) { ... }
finally {
    if (stmt != null) {
        try {
            stmt.close();
        } catch(SQLException ex) { ... }
    }
}
```


Executando comandos SQL

- Utilize a classe `java.sql.ResultSet` para acessar os dados resultantes de uma consulta SQL (*SELECT*)
- Exemplo:

```
ResultSet rs =
    stmt.executeQuery("SELECT * FROM usuario");
```


java.sql.ResultSet

- `boolean next() throws SQLException`
  - posiciona o cursor na próxima linha
  - inicialmente o cursor esta antes da primeira linha
- `String getString(int col) throws SQLException`
- `String getString(String nomeColuna) throws SQLException`
- `void close() throws SQLException`
- ...

ResultSet.getXXX métodos

- Tabela de métodos e tipos de dados
  - o "X" indica que o método é o mais recomendado para o tipo do dado

	TINYINT	SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC	BIT	CHAR	VARCHAR	LONGVARCHAR	BINARY	VARBINARY	LONGVARBINARY	DATE	TIME	TIMESTAMP
getByte	X	x	x	x	x	x	x	x	x	x	x	x	x						
getShort	x	X	x	x	x	x	x	x	x	x	x	x	x						
getInt	x	x	X	x	x	x	x	x	x	x	x	x	x						
getLong	x	x	x	X	x	x	x	x	x	x	x	x	x						
getFloat	x	x	x	x	X	x	x	x	x	x	x	x	x						
getDouble	x	x	x	x	x	X	X	x	x	x	x	x	x						
getBigDecimal	x	x	x	x	x	x	x	X	X	x	x	x	x						
getBoolean	x	x	x	x	x	x	x	x	x	X	x	x	x						
getString	x	x	x	x	x	x	x	x	x	x	X	X	x	x	x	x	x	x	x
getBytes														X	X	x			
getDate											x	x	x				X		x
getTime											x	x	x					X	x
getTimestamp											x	x	x				x	x	X
getAsciiStream											x	x	X	x	x	x			
getUnicodeStream											x	x	X	x	x	x			
getBinaryStream														x	x	X			
getObject	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Exemplo: consultando dados

```
Statement stmt = null;
ResultSet resultado = null;
String cmd = "SELECT * FROM usuarios ";
try {
    stmt = con.createStatement();
    resultado = stmt.executeQuery(cmd);
    while (resultado.next()) {
       String nome  = resultado.getString("nome");
       String email = resultado.getString("email");
       System.out.println("Nome: " + nome + " - Email:" +email);
    }
}
catch (SQLException ex) { ... }
finally {
    if (resultado != null) {
        try {
        resultado.close();
        } catch(SQLException ex) { ... }
    } // feche o statement (stmt) também...
}
```


java.sql.PreparedStatement

- subclasse de **java.sql.Statement**
- os comandos são pré-compilados
  - depende do suporte do driver
- os comandos SQL podem possuir um ou mais parâmetros IN
  - parâmetros sem valor especificado em tempo de compilação do comando SQL
  - representados por "?" (*parameter marker*)
  - o valor precisa ser especificado antes da execução do comando

java.sql.PreparedStatement

- reduz o tempo de execução para comandos SQL que são executados várias vezes seguidas
- os métodos **execute**, **executeQuery** e **executeUpdate** foram modificados para não receber argumentos
  - não utilize os métodos herdados de **Statement** que recebem o comando SQL como argumento

java.sql.PreparedStatement

- Criando um PreparedStatement
  - utilize o método `prepareStatement (String)` de `java.sql.Connection`
  - lembre-se de colocar as "`?`" para marcar os parâmetros que serão informados

java.sql.PreparedStatement

- Informando parâmetros
  - os parâmetros são passados por métodos do tipo `setXXX(int pos, XXX value)`
    - `XXX` corresponde ao tipo do parâmetro
    - pos é a posição do parâmetro (marcador "`?`") começando por 1
      - ATENÇÃO: a contagem não começa em 0 (zero)
    - uma vez definido, o valor do parâmetro pode ser utilizado para várias execuções do comando até ser substituído por um novo valor, ou até uma chamada ao método `clearParameters()`

java.sql.PreparedStatement Exemplo

```
// comando SQL a ser executado
String comandoSQL = "UPDATE empregados "+
                    "SET sal = (sal*1.1) " +
                    "WHERE depto_num = ?";
PreparedStatement pstmt =
    con.prepareStatement(comandoSQL);

//informe o valor do parâmetro "?"
pstmt.setInt(1, 10);

//execute o comando
pstmt.executeUpdate();
```

Mapeando tipos SQL e Java

- Java *vs* SQL
  - os tipos de dados não são iguais
  - JDBC provê mecanismos para conversão entre os tipos
    - —métodos `getXXX`, `setXXX`, `registerOutParameter` e a classe `java.sql.Types`
- Diferentes provedores de SGBDs adotam nomeclaturas de tipos diferentes

Mapeando tipos SQL e Java

- Mapeamento de tipos em JDBC para tipos em Java

JDBC	Java
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
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

Tratamento de Exceções

- A maioria dos métodos das classes **`Statement`** e **`Connection`** levantam a exceção **`SQLException`**
- Tratando as exceções

```
try {...}
catch (SQLException e) {...}
finally {...}
```

  - captura e tratamento local
  - captura e lançamento de uma nova exceção
    - —adiciona semântica da aplicação

Transações

- Preservam a consistência dos dados do SGBD
- Propriedades das transações (ACID):
  - atomicidade
  - consistência
  - isolamento
  - durabilidade

Implementando Transações

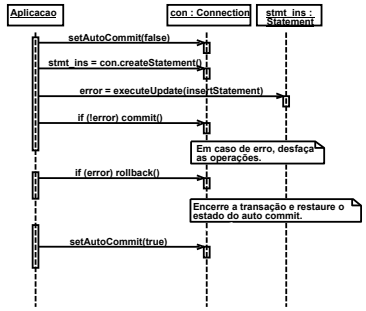
- Propriedade *auto-commit* de `java.sql.Connection`
  - `true` – realiza *commit* após cada operação executada pelo `Statement` ser completada (*default*)
  - `false` – não realiza *commit* automaticamente

Implementando Transações

- Métodos
  - `setAutoCommit(boolean autoCommit)`
    - —define o modo de *commit*
  - `commit()`
    - —confirma a transação
  - `rollback()`
    - —cancela a transação

Implementando Transações

- Exemplo de transação com JDBC



E o projeto?

- Agora podemos implementar um repositório em meio persistente!
  - Vide o FAQ avançado!

http://www.cin.ufpe.br/~scbs/ceut/

Especialização em Engenharia de Software

Programação Orientada a Objetos
JDBC – Java Database Connectivity

Sérgio Soares
scbs@cin.ufpe.br