1. Banco de Dados e JDBC

1.1 Conceitos

Muitos sistemas precisam manter as informações com as quais eles trabalham, seja para permitir consultas futuras, geração de relatórios ou possíveis alterações nas informações. Para que esses dados sejam mantidos por um tempo indeterminado, esses sistemas geralmente guardam essas informações em um banco de dados, que as mantém de forma organizada e prontas para consultas. A maioria dos bancos de dados comerciais são os chamados relacionais, que é uma forma de trabalhar e pensar diferente ao paradigma orientado a objetos.

No contexto de programação orientada a objetos com Java e banco de dados, devemos levar em conta que cada BD possui seu protocolo de comunicação, sendo assim impossível manter um padrão de comunicação "Java puro". A abstração dos BDs que contém diferentes protocolos de comunicação só é possível graças a API puramente desenvolvida em Java para acesso SQL, juntamente com um gerenciador de *drivers* que se conecta ao *driver* proprietário do BD, fornecido diretamente por seu desenvolvedor. Esse conjunto de interfaces para conexão Java↔BD encontra-se dentro do pacote *java.sql* e nos referimos a ela como JDBC[1].

A maior vantagem do JDBC sobre outros ambientes de programação é que os programas desenvolvidos em Java com JDBC são independentes de plataforma ou de banco de dados. Isso significa que um programa Java que acessa uma base de dados pode funcionar em qualquer Sistema Operacional e também acessar qualquer banco de dados

[1] *Java Database Connectivity* (Conectividade em Banco de Dados em Java)

1.2 Trabalhando com JDBC

1.2.1 Registrando o Driver para Acesso ao Banco de Dados

O primeiro passo para implementarmos um programa com conexão a banco de dados é definir que servidor de banco de dados utilizar. Para cada servidor, um determinado tipo de driver será utilizado, bem como a forma de invocar este driver.

Após decidir qual servidor será utilizado, é necessário registrar seu driver no código Java. Registrar o driver é o processo pelo qual as classes do driver do banco de dados serão carregadas na memória para poderem ser utilizadas como uma implementação das interfaces JDBC.

É necessário fazer este registro apenas uma vez no programa. Este registro é feito da seguinte maneira:

```
Class.forName(string_do_driver);
```

Exemplo: `Class.forName("oracle.jdbc.OracleDriver");`

É digno de nota que, ao trabalharmos com banco de dados, o tratamento de exceções torna-se padrão; logo, o código completo de registro de driver encontra-se abaixo:

```
try {
   Class.forName("oracle.jdbc.OracleDriver");
}
catch(ClassNotFoundException ex) {
   System.out.println("Impossível carregar o driver!");
}
```

Uma segunda abordagem para registrar um driver é utilizar o método estático *DriverManager.registerDriver().* Este método é muito útil quando utilizamos uma JVM não compatível com o JDK, como a *Microsoft Java Virtual Machine*. O exemplo a seguir mostra a utilização do *registerDriver()* para registrar o driver Oracle:

```
try {
      Driver myDriver = new oracle.jdbc.OracleDriver();
      DriverManager.registerDriver( myDriver );
   }
catch(ClassNotFoundException ex) {
      System.out.println("Impossível carregar o driver!");
   }
```

1.2.2 Conectando-se ao Banco de Dados

Em JDBC, a conexão ao banco de dados dá-se através de uma URL. A Figura 1 exemplifica algumas URLs para conexão de alguns dos mais utilizados bancos de dados existentes. Lembrando que só será possível se conectar ao BD se o .jar do driver estiver incluído na biblioteca do projeto.

SGBD	Formato da URL de conexão
ORACLE	jdbc:oracle:thin:@**host**:**porta**:**nomeBD** (Porta padrão 1521)
PostgreSQL	jdbc:postgresql://**host**:**porta**/**nomeBD** (Porta padrão 5432)
IBM DB2	jdbc:db2://**host**:**porta**/**nomeBD** (Porta padrão 446, 6789, ou 50000)
Microsoft SQL Server	jdbc:sqlserver://**host**:**porta**;databaseName=**nomeBD** (Porta padrão 1433)
MySQL	jdbc:mysql://**host**:**porta**/**nomeBD** (Porta padrão 3306)
	jdbc:firebirdsql:**host**/**porta**:**caminhoArquivo**.**fdb** (Porta padrão 3050)
Data Sources (ODBC) ODBC	jdbc:odbc:**nomeBD**

Figura 1. Banco de Dados de URL para conexão via JDBC

Para estabelecer uma conexão com o banco de dados, utilizamos o método *DriverManager.getConnection().* A classe *DriverManager* possui uma sobrecarga do método *getConnection* (), sendo:

```
getConnection (String url)
getConnection (String url, Properties prop)
getConnection (String url, user String, password String)
```

Como podemos perceber cada método necessita receber a URL do banco de dados. Os métodos listados retornarão um objeto do tipo *Connection*, que será utilizado posteriormente para manipularmos a base de dados.

Na URL é necessário definir **nome do host**, **porta** e **nome da base de dados**. O nome do host é o nome da máquina a qual o banco de dados se encontra (geralmente, *localhost*). A porta é definida no momento da instalação do servidor de banco de dados (para ORACLE, a porta padrão é 1521).

A seguir, exemplos de como se conectar com o banco de dados utilizando cada um dos três métodos citados:

```
String URL = "jdbc:oracle:thin:USUARIO/SENHA@HOST:1521:NOME_BD";
Connection conn = DriverManager.getConnection(URL);
```

```
import java.util.*;

String URL = "jdbc:oracle:thin:@HOST:1521:NOME_BD";
Properties info = new Properties( );
info.put( "user", "USUARIO" );
info.put( "password", "SENHA" );
Connection conn = DriverManager.getConnection(URL, info);
```

```
String URL = "jdbc:oracle:thin:@HOST:1521:NOME_BD";
String USER = "USUARIO";
String PASS = "SENHA";
Connection conn = DriverManager.getConnection(URL, USER, PASS);
```

1.2.3 Interagindo com o Banco de Dados

Para interagir com o banco de dados é necessário criar um objeto de ligação. Objetos JDBC do tipo *Statement, CallableStatement*, e *PreparedStatement* possuem métodos e propriedades que possibilitam o envio e recebimento de cláusulas SQL ou PL/SQL da database.

A tabela a seguir apresenta um resumo do propósito de cada interface, para escolha conforme o tipo de projeto.

Tabela 1. Interfaces de interação com banco de dados

Statement	Útil para acessos de propósito gerais no banco de dados. Ex.: instruções SQL estáticas em tempo de execução. A interface *Statement* não aceita parâmetros.
PreparedStatement	Útil quando se planeja usar instruções SQL muitas vezes. A interface *PreparedStatement* aceita parâmetros de entrada em tempo de execução.
CallableStatement	Útil quando se deseja acessar *PROCEDURES* armazenadas. A interface *CallableStatement* também aceita parâmetros de entrada em tempo de execução.

Em nosso curso usaremos apenas as interfaces *Statement* e *PreparedStatement.*

1.2.3.1 Utilizando Statement

Antes de poder usar um objeto *Statement* para executar uma instrução SQL, é necessário criá-lo usando o método *createStatement ()* do objeto *Connection*, como demonstrado a seguir:

```
Statement stmt = null;
try {
   stmt = conn.createStatement( );
  // . . .
}catch (SQLException e) {
  // . . .
}finally{
      stmt.close();
      conn.close();
}
```

Após isso, é necessário definir qual dos seguintes métodos utilizar (conforme comando SQL a ser executado):

Tabela 2. Métodos *Statement*

`boolean execute(String SQL)`	Retorna um valor booleano *true* se um objeto *ResultSet* puder ser recuperado, ou *false*, caso contrário. Utilize este método para executar instruções SQL DDL ou quando é necessário usar SQL dinâmico.
`int executeUpdate(String SQL)`	Retorna a quantidade de linhas afetadas pela execução de uma instrução SQL. Este método é utilizado para executar comandos INSERT, UPDATE ou DELETE.
`ResultSet executeQuery(String SQL)`	Utilizado quando se deseja executar o comando SELECT

Outro fator a ser levado em conta é o tipo de dado. Para manipular dados SQL via JDBC, deve-se fazer uso de uma tabela de conversão Java-JDBC (Tabela 3).

Tabela 3 Mapeamento de tipos JDBC para 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

A Figura 2 exemplifica um código de inserção de dados numa tabela:

```
//String para armazenar o comando SQL
String sql_ins = "insert into alunos (id, nome) values(1, 'Ana')";

//Cria o objeto a partir da conexão
Statement stmt = con.createStatement();

//Executa o comando de atualização pré-definido
int linhas_afetadas = stmt.executeUpdate(sql_ins);

//Retorna o número de linhas afetadas (insert, delete, update)
```

Figura 2. Exemplo de *Statement* para inserção de dados (comando INSERT)

1.2.3.2 Utilizando PreparedStatement

Provê flexibilidade quanto à inserção de vários elementos ao mesmo tempo. Suponhamos uma tabela a qual se deseja inserir vários nomes de alunos e suas respectivas notas. Uma possível solução com a utilização de *PreparedStatement* encontra-se abaixo:

Tabela 4. Tabela alunos no banco de dados TURMA1

Nome	Media
Ana	9.5
Glauber	10.0

```
String sql = "INSERT INTO alunos (Nome,Media) VALUES(?,?)";
PreparedStatement pstmt = conn.prepareStatement(sql);

for (int i = 0; i < turma.length; i++) {
        // Setar os valores
       pstmt.setString(1,turma.nome[i]);  //1 = primeira '?' da String sql
       pstmt.setDouble(2,turma.media[i]); //2 =  segunda '?' da String sql
       pstmt.addBatch(); //insere uma atualização
}
int count[] = pstmt.executeBatch(); //executa todas as atualizações
for(int i = 1;i < = count.length; i++){
        System.out.println("Query "+i+" modificada "+count[i]+" vezes");
}
conn.commit(); //aplica as atualizações
System.out.println("Dados inseridos na tabela!");
pstmt.close(); //encerra objeto
conn.close(); //encerra a conexão
```

1.2.3.3 Realizando Consultas

Para realizar consultas em determinada tabela, é necessário instanciar um objeto da classe *ResultSet* (importando o pacote java.sql.ResultSet) e utilizar o método *executeQuery* de *Statement*. Um exemplo de como efetuar uma consulta em uma tabela é ilustrado na Figura 3.

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

Figura 3. Exemplo de consulta utilizando JDBC

A classe *ResultSet* possui métodos adequados para recuperar dados de uma tabela conforme seu tipo. A Figura 4 ilustra a tabela de métodos e tipos de dados, sendo aqueles indicados com 'X' em negrito o mais recomendado para o tipo de 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

Figura 4. Tabela x Tipo de Dado

Um exemplo de consulta de dados em uma tabela encontra-se a seguir.

```
//...
SQL = "SELECT * from Alunos";
ResultSet rs = stmt.executeQuery(SQL);

    while (rs.next()) { //enquanto houverem linhas na tabela
            System.out.println(rs.getString("nome")); //coluna nome
            System.out.println(rs.getString("curso")); //coluna curso
      }
```

2. Exercícios de Fixação

1. Analise o código “Principal.java” fornecido pela professora. O código presente encontra-se estruturado e utiliza o servidor MySQL. Modifique o mesmo de forma a se tornar orientado a objetos e utilizar o servidor banco de dados de sua preferência. Crie atributos e métodos a vontade.

2. Insira no código as seguintes características:

    a. O usuário possa entrar com informações pelo teclado e as inserir na tabela;

    b. O usuário possa digitar informações pelo teclado e buscá-las na tabela;

    c. O usuário possa deletar um registro a partir do cpf (digitado pelo usuário);

    d. Modificar a fase de um aluno, e imprimir em tela os dados anteriores e posteriores a modificação.

3. ***DESAFIO:*** Crie um menu de forma que o usuário possa escolher uma das seguintes opções:

    a. Digite 1 para ‘Conexão ao Banco de Dados’;

    b. Digite 2 para ‘Criação de uma nova DATABASE’;

    c. Digite 3 para ‘Conexão à uma DATABASE’;

    d. Digite 4 para ‘Criação de Tabelas em uma DATABASE’;

    e. Digite 5 para ‘Inserção de informações numa tabela’;

    f. Digite 6 para ‘Busca de Informações numa tabela'.

Cada opção deve efetuar no banco de dados a ação que oferece.

4. ***DESAFIO:*** Modifique o código de forma que dados como driver do BD e URL estejam gravados num arquivo txt e sejam recuperados pelo programa quando o mesmo necessitar de tais dados

Os exercícios podem ser feitos em dupla e entregues à professora via moodle. Estes complementarão a nota de A1.