

### **Engenharia da Computação – 3ª série**

## **Conexão com BD usando JDBC** **(L1/1, L2/1 e L3/1)**

**2025**

# ECM251 – Linguagens de Programação I

## Aula 10 – L1/1, L2/1 e L3/1

### Horário

Terça-feira: 2 x 2 aulas/semana

- L1/1 (07h40min-09h20min): *Prof. Calvetti*;
- L1/2 (09h30min-11h10min): *Prof. Calvetti*;
- L2/1 (07h40min-09h20min): *Prof. Menezes*;
- L2/2 (11h20min-13h00min): *Prof. Calvetti*;
- L3/1 (09h30min-11h10min): *Prof. Evandro*;
- L3/2 (11h20min-13h00min): *Prof. Evandro*.

# ECM251 – Linguagens de Programação I

## Conexão com BD usando JDBC

### Tópico

- *JDBC*

### Definição



- *Java Database Connectivity* – **JDBC**, isto é, Conectividade de Banco de Dados em Java, é a tecnologia usada para que programas Java clientes acessem servidores de bancos de dados;
- Consiste em um conjunto de classes e interfaces (API) implementadas pelos fabricantes de bancos de dados pertencentes ao pacote ***java.sql***;
- Esta implementação é reunida em um arquivo ***.jar***, pelo fabricante do Banco de Dados – BD, denominado driver de JDBC.

## Conexão com BD usando JDBC

### Tópico

- Java com Banco de Dados

### Definição



- Para ser realizar uma conexão de um aplicativo em Java com um Banco de Dados, são necessários os seguintes itens:
  - ✓ *Driver do JDBC (.jar);*
  - ✓ *String de Conexão;*
  - ✓ *Objeto Connection;*
  - ✓ *Objeto PreparedStatement;*
  - ✓ *Objeto ResultSet.*

### Definição



- Deve-se, então:
  - ✓ Colocar esses itens nas classes de uma aplicação Java;
  - ✓ Dividir em métodos “CRUD”: inserir, consultar, atualizar e excluir;
  - ✓ Criar uma classe para obter a conexão com o BD;
  - ✓ Tratar as transações, quando forem necessárias;
  - ✓ Preparar o aplicativo para lidar com exceções frequentemente.

### Tópico

- Exceções em Java



## Exceções em Java

### Definição



- Problemas em programas acontecem;
- O Java tem mecanismos para lidar com eles, evitando que simplesmente interrompam seus processamentos;
- Faz isso através dos blocos ***try-catch-finally*** e das ***exceptions***.

## Exceções em Java

### Exemplo



- Existem situações sobre as quais o programador tem total controle, como fazer um laço corretamente ou evitar divisão por zero e, neste caso, é opcional para ele usar um ***try-catch***;
- Existem outras situações, entretanto, sobre as quais o programador não tem controle, por exemplo, quando o usuário escolhe salvar um arquivo em um diretório no qual não se tem permissão de escrita, ou quando o usuário tenta inserir um cliente no BD que já existe, ou quando a rede cai e o programa Java não consegue a conexão com a Internet, sendo nestes casos, obrigatório para ele usar um ***try-catch***.

### Tópico

- O bloco ***try-catch-finally***

## O bloco *try-catch-finally*

### Definição



- Problemas em programas podem acontecer;
- O Java tem mecanismos para lidar com eles, evitando que simplesmente interrompam o processamento dos programas;
- O Java faz isso através dos seus blocos ***try-catch-finally*** e das suas ***exceptions***.

## O bloco try-catch-finally

### Definição



- Sua estrutura é:

```
try
{
    // Código que eventualmente pode encontrar problemas
}
catch(Exception e)
{
    // Código do que será executado quando houver problema no código do try
}
finally
{
    // Código que será executado sempre, quando houver problema ou não no código do try
}
```

## O bloco *try-catch-finally*

### Definição



- A partir do Java 1.7 passou a existir um tipo de ***try-catch*** que fecha automaticamente os recursos que foram abertos no ***try*** e que precisam ser fechados, como, por exemplo, os arquivos ou as conexões com o BD;
- A diferença das versões anteriores é que abre-se e fecha-se parênteses depois do ***try*** e, dentro destes parênteses, faz-se a abertura dos recursos que serão automaticamente fechados.

## O bloco try-catch-finally

### Definição



- Sua estrutura é:

```
try( /* Código que abre os recursos que precisarão ser fechados, e.g. conexões */ )
{
    // Código que eventualmente pode encontrar problemas
}
catch(Exception e)
{
    // Código do que será executado quando houver problema no código do try
}
finally
{
    // Código que será executado sempre, quando houver problema ou não no código do try
}
```

### Tópico

- Exceções em Java



### Definição



- A classe ***Exception*** é a superclasse de todas as ***exceptions***, que são tratadas por ela caso qualquer uma delas aconteça;
- Porém, existem exceções mais específicas, usadas para dar tratamentos específicos para um problema;
- Neste caso, pode-se simplesmente repetir o bloco ***catch*** várias vezes, por ordem de especificidade, com as mais específicas antes e as mais genéricas depois, terminando com a ***Exception***.

## Exceções em Java

### Definição



- Pode-se, também, utilizar um ***try-multicatch***, para quando o tratamento dado a diversas exceções precisar ser o mesmo;
- O uso do ***finally*** não é obrigatório para tratar exceções, apenas um ***try*** e um ***catch*** (ou vários) são necessários;
- No caso de se optar por não tratar exceção, pode-se lança-la por meio do comando ***throws***.

### Definição



- Sua estrutura é:

```
try
{
    // Código que eventualmente pode encontrar problemas
    // que possam gerar exceções do SQL ou IO, por exemplo
}
catch(SQLException e)
{
    // Código do que será executado quando houver uma exceção no try do tipo SQL
}
catch(IOException e)
{
    // Código do que será executado quando houver uma exceção no try do tipo IOL
}
catch(SQLException e)
{
    // Código do que será executado quando houver uma exceção genérica no tryL
}
finally
{
    // Código que será executado sempre, quando houver problema ou não no código do try
}
```

### Definição



- Nunca irá acontecer mais de uma **exception** ao mesmo tempo, porque quando uma exceção acontece, a execução do bloco **try** é imediatamente interrompida e passa-se a execução para o bloco **catch** correspondente;
- Caso o **catch** da **Exception** genérico seja colocado antes dos **catchs** específicos **SQLException** e **IOException**, o código não irá compilar, pois as duas **exceptions** específicas nunca serão alcançadas devido ao **Exception** pegar todas as exceções antes.

## Exceções em Java

### Exemplo



- Um ***try-multicatch*** utilizando o operador **|** (ou) para unir logicamente as exceções:

```
import javax.swing.JOptionPane;
public class Excecoes{
    public static void main(String[] args){
        String s = JOptionPane.showInputDialog("Valor");

        try{
            int x = Integer.parseInt(s.substring(2,3));
        } catch (NullPointerException | NumberFormatException | StringIndexOutOfBoundsException e){
            throw new RuntimeException(e);
        }
    }
}
```

## Conexão com BD usando JDBC

### Tópico

- Carregando um JDBC no Java

## Carregando um JDBC no Java

### Definição



- Utiliza-se um bloco estático para isso;
- Carrega-se o *driver* utilizando o método ***Class.forName***;
- A ***String*** para pegar o *driver* depende de cada fabricante;
- No caso do **MySQL**, a ***String*** deve ser ***com.mysql.jdbc.Driver***;
- Deve-se tratar a exceção ***ClassNotFoundException***, que acontece quando a classe não é encontrada;
- O caminho do *driver* deve ser colocado no ***CLASSPATH***.

## Carregando um JDBC no Java

### Exemplo



- A classe de conexão com o BD:

```
public class ConexaoBD {  
  
    static {  
        try {  
            Class.forName("com.mysql.jdbc.Driver");  
        }  
        catch (ClassNotFoundException e) {  
            throw new RuntimeException(e);  
        }  
    }  
}
```



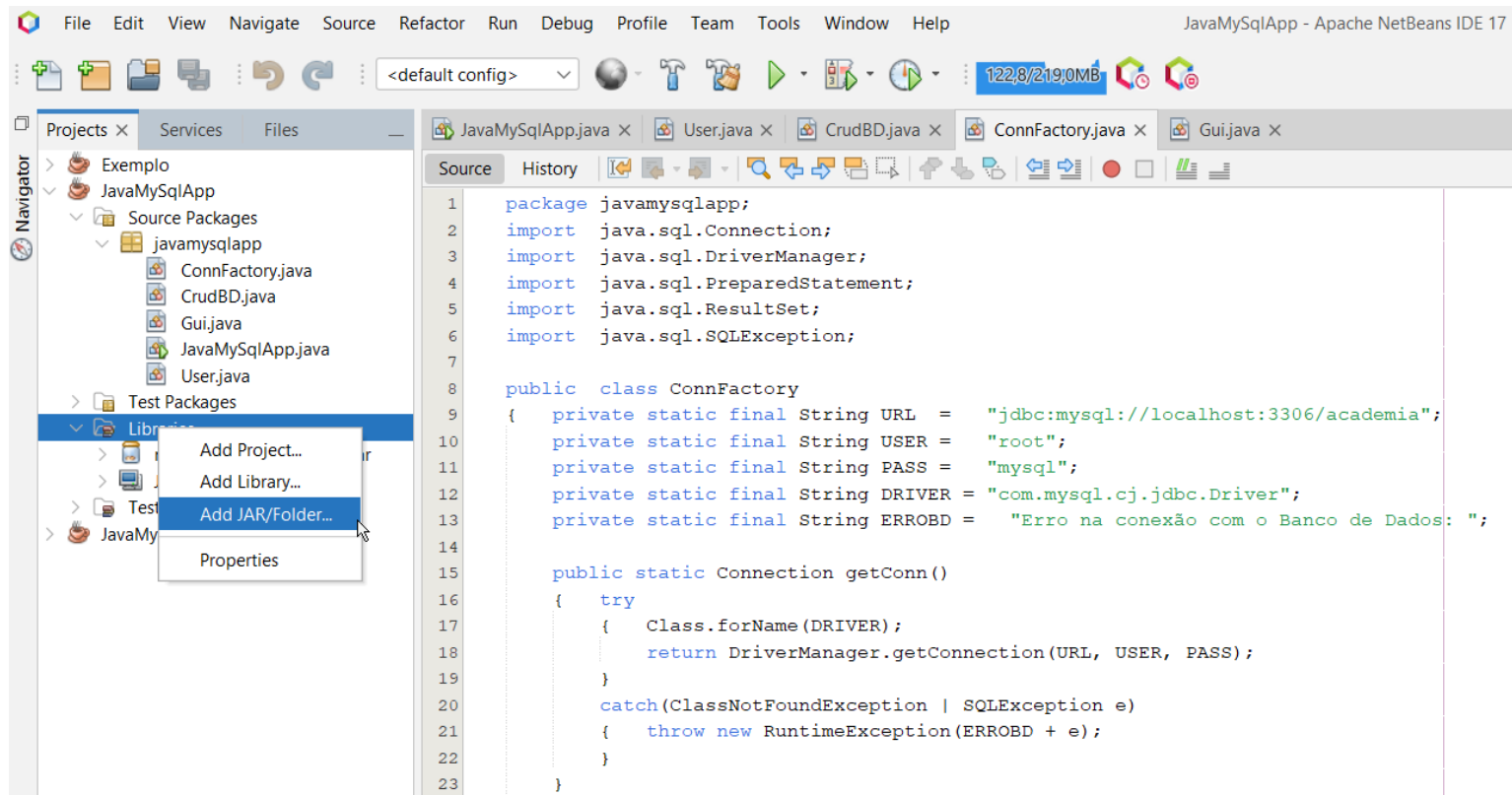
# ECM251 – Linguagens de Programação I

## Carregando um JDBC no Java

### Exemplo



- A configuração do CLASSPATH no NetBeans:



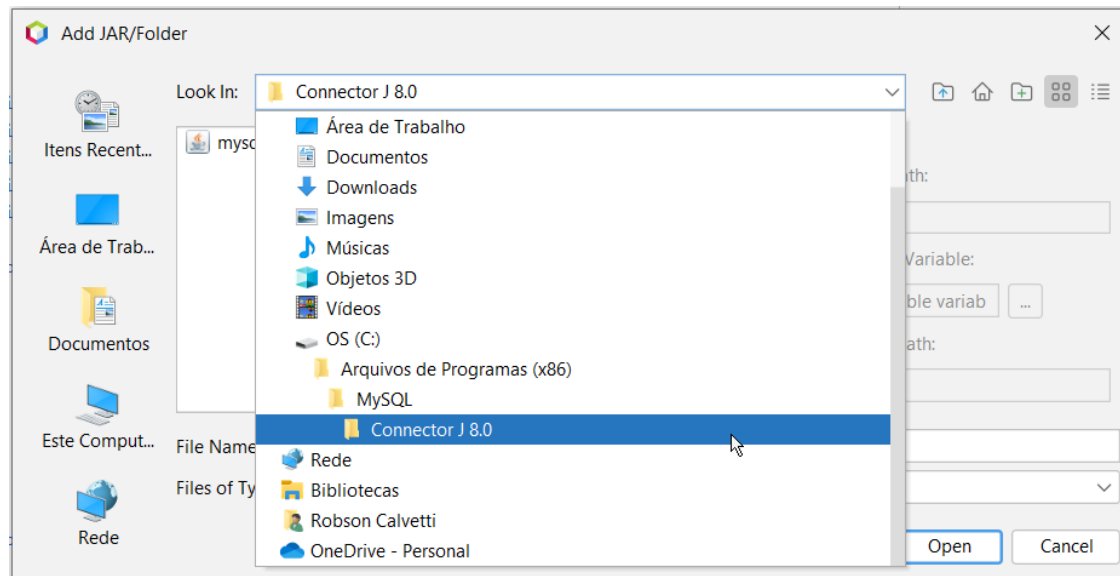
# ECM251 – Linguagens de Programação I

## Carregando um JDBC no Java

### Exemplo



- A configuração do CLASSPATH no NetBeans:



# ECM251 – Linguagens de Programação I

## Carregando um JDBC no Java

### Exemplo



- A configuração do CLASSPATH no NetBeans:

### Tópico

- Pegando uma Conexão no Java

## Pegando uma Conexão no Java

### Definição



- Usa-se o ***DriverManager.getConnection*** em Java para pegar uma conexão com o BD;
- Neste método, deve-se passar uma *String* como parâmetro de conexão, parecida com a ***URL*** (endereço) de um portal, usando-se o ***jdbc*** no lugar do ***http***;
- Nessa *String* passa-se o fabricante do banco, o servidor, a porta, o *database*, o usuário e a senha;
- Para desconectar, basta dar um ***close*** na conexão aberta.

### Exemplo



- Abaixo, o método está lançando uma ***SQLException***, pois vários problemas podem acontecer nesta operação: rede fora, banco fora, usuário e senha inválidos, acesso negado etc.:

```
public static Connection conectar() throws SQLException {
    String servidor = "localhost";
    String porta = "3306";
    String database = "tutorial";
    String usuario = "aluno";
    String senha = "alunos";
    return DriverManager
        .getConnection("jdbc:mysql://" + servidor + ":" + porta +
            "/" + database + "?user=" + usuario + "&password=" + senha);
}

public static void desconectar(Connection conn) throws SQLException {
    conn.close();
}
```

## Conexão com BD usando JDBC

### Tópico

- Incluindo, Excluindo e Atualizando dados no BD com Java

### Definição



- Nestes três casos, os métodos são bem parecidos:
  1. Escreva, em uma **String**, o **comando SQL** com pontos de interrogação (?) nos campos parametrizáveis;
  2. Pegue uma conexão;
  3. Peça um **PreparedStatement** para a conexão, passando a **String** como parâmetro;
  4. Atribua valores, via **set**, para os pontos de interrogação, pelo seu número de ordem e pelo tipo de dado, sendo o primeiro ponto de interrogação o 1, o segundo o 2 e assim por diante, configurando um **inteiro** como **setInt** e um **varchar**, como **setString**.



### Definição



- Nestes três casos, os métodos são bem parecidos:
  5. Invoque o método ***execute()***, do ***PreparedStatement***;
  6. Dê ***commit***, caso **não** esteja em ***auto commit***;
  7. Feche o ***preparedStatement***, se não tiver usado ***try with resources***;
  8. Feche a conexão, somente depois que encerrar a transação;
  9. Trate as exceções e dando ***rollback*** se der errado.

# ECM251 – Linguagens de Programação I

*Incluindo, Excluindo e Atualizando dados no BD com Java*

## Definição



- Para os exemplos a seguir considere:
  - ✓ Uma classe **Cliente** com os atributos ***idCliente(int), nome(String), telefone(String)***;
  - ✓ Uma banco **tutorial** com uma tabela **Cliente**, com os campos ***id(smallint, pk), nome(varchar 60), fone(char 10)***.

# ECM251 – Linguagens de Programação I

## Incluindo, Excluindo e Atualizando dados no BD com Java

### Exemplo



```
public void incluir(Connection conn) {
    String sqlInsert =
        "INSERT INTO cliente(id, nome, fone) VALUES (?, ?, ?)";

    try (PreparedStatement stm = conn.prepareStatement(sqlInsert);) {
        stm.setInt(1, getIdCliente());
        stm.setString(2, getNome());
        stm.setString(3, getFone());
        stm.execute();
    }
    catch (Exception e) {
        e.printStackTrace();
        try {
            conn.rollback();
        }
        catch (SQLException e1) {
            System.out.print(e1.getStackTrace());
        }
    }
}
```

# ECM251 – Linguagens de Programação I

## Incluindo, Excluindo e Atualizando dados no BD com Java

### Exemplo



```
public void excluir(Connection conn) {  
    String sqlDelete = "DELETE FROM cliente WHERE id = ?";  
    try (PreparedStatement stm = conn.prepareStatement(sqlDelete);) {  
        stm.setInt(1, getIdCliente());  
  
        stm.execute();  
    }  
    catch (Exception e) {  
        e.printStackTrace();  
        try {  
            conn.rollback();  
        }  
        catch (SQLException e1) {  
            System.out.print(e1.getStackTrace());  
        }  
    }  
}
```

# ECM251 – Linguagens de Programação I

## Incluindo, Excluindo e Atualizando dados no BD com Java

### Exemplo



```
public void atualizarTelefone(Connection conn, String novoFone) {  
    String sqlUpdate = "UPDATE CLIENTE SET FONE = ? WHERE ID = ?";  
  
    try (PreparedStatement stm = conn.prepareStatement(sqlUpdate);) {  
        stm.setString(1, novoFone);  
        stm.setInt(2, getIdCliente());  
  
        stm.execute();  
    }  
    catch (Exception e) {  
        e.printStackTrace();  
        try {  
            conn.rollback();  
        }  
        catch (SQLException e1) {  
            System.out.print(e1.getStackTrace());  
        }  
    }  
}
```

### Tópico

- Consultando dados no BD com Java

### Definição



- Parecido com os anteriores, mas com a diferença de que se está buscando dados no BD e não os enviando:
  1. Escreva em uma **String** o comando **SELECT** com pontos de interrogação (?) nos campos parametrizáveis;
  2. Pegue uma conexão;
  3. Peça um **PreparedStatement** para a conexão. passando a **String** como parâmetro.
  4. Atribua valores, via **set**, para as interrogações, por seu número de ordem e tipo de dado, sendo a primeira interrogação o 1, a segunda o 2 e assim por diante e para configurar um **inteiro** use **setInt** e para **varchar** use **setString**.



### Definição



- Parecido com os anteriores, mas com a diferença de que se está buscando dados no BD e não os enviando:
  5. Chame o método ***executeQuery()***, do ***PreparedStatement***;
  6. Pegue o ***ResultSet***;
  7. Navegue no ***ResultSet***, usando o método ***next()***, pegando as informações via ***get*** e passando o número de ordem da coluna dentro do ***select***, pegando um inteiro na primeira coluna com ***getInt(1)***;



### Definição



- Parecido com os anteriores, mas com a diferença de que se está buscando dados no BD e não os enviando:
  8. Feche o **resultSet** e o **preparedStatement**, nesta ordem, se não houver usado **try with resources**;
  9. Feche a conexão, somente depois que encerrar alguma transação pendente;
  10. Trate as exceções.

# ECM251 – Linguagens de Programação I

## Consultando dados no BD com Java

### Exemplo



```
public void carregar(Connection conn) {
    String sqlSelect =
        "SELECT * FROM cliente WHERE cliente.id = ?";

    try (PreparedStatement stm = conn.prepareStatement(sqlSelect);){
        stm.setInt(1, getIdCliente());
        try (ResultSet rs = stm.executeQuery());{
            /*este outro try e' necessario pois nao da' para abrir o resultset
            *no anterior uma vez que antes era preciso configurar o parametro
            *via setInt; se nao fosse, poderia se fazer tudo no mesmo try
            */
            if (rs.next()) {
                this.setNome(rs.getString(2));
                this.setFone(rs.getString(3));
            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
    catch (SQLException e1) {
        System.out.print(e1.getStackTrace());
    }
}
```

1 Cliente - busca pela PK

# ECM251 – Linguagens de Programação I

## Consultando dados no BD com Java

### Exemplo



```
public ArrayList<Cliente> buscarClientes(Connection conn){
    String sqlSelect = "SELECT id, nome, fone FROM CLIENTE";
    ArrayList<Cliente> lista = new ArrayList<>();

    try(PreparedStatement stm = conn.prepareStatement(sqlSelect);
        ResultSet rs = stm.executeQuery());{
        //veja que desta vez foi possivel usar o mesmo try
        while(rs.next()){
            Cliente cliente = new Cliente();
            cliente.setIdCliente(rs.getInt("id"));
            cliente.setNome(rs.getString("nome"));
            cliente.setFone(rs.getString("fone"));
            lista.add(cliente);
        }
    } catch (Exception e){
        e.printStackTrace();
    }
    return lista;
}
```

Vários clientes.  
Usar uma coleção para retorná-los,

### Tópico

- Controle de Transação

### Definição



- Uma **transação** é um conjunto de alterações de dados no banco que não pode ser parcial, ou seja, ou acontece totalmente, ou não acontece;
- O **MySQL** trabalha no modo **auto commit**, isto é, cada **insert** ou **delete** é efetivado no banco e não dá para desfazê-los após isso;
- Para controlar a transação, usa-se o método **setAutoCommit(false)** da conexão;
- Para efetivar a transação, usa-se **commit()** da conexão e para desfazê-la, usa-se **rollback()** da conexão;
- Importante lembrar que não se deve abrir ou fechar a conexão no meio, pois perderá a transação se o fizer.

# ECM251 – Linguagens de Programação I

## Conexão com BD usando JDBC

### Exemplo 1



```
import java.sql.SQLException;
import java.sql.Connection;

public class Teste {

    public static void main(String[] args) {
        Connection conn = null;
        Cliente cl;
        Vendedor vd;

        try {
            // obtém conexão com o banco
            ConexaoBD bd = new ConexaoBD();
            conn = bd.conectar();
            vd = new Vendedor();

            // *** IMPORTANTE ***: Força o uso de transação.
            conn.setAutoCommit(false);
            // *** Inclusão do Primeiro Cliente ***
            cl = new Cliente(1001, "Zé das Couves", "1127991999");
            cl.incluir(conn);

            // *** Inclusão do Segundo Cliente ***
            cl = new Cliente();
            cl.setIdCliente(1002);
            cl.setNome("João das Couves");
            cl.setFone("1160606161");
            cl.incluir(conn);

            // *** Inclusão do Terceiro Cliente ***
            cl = new Cliente(1003, "Maria das Couves", "1121212121");
            cl.incluir(conn);

            // *** IMPORTANTE ***: Efetiva inclusões
            conn.commit();

            // *** Lista todos os clientes
            System.out.println("\nLista todos os clientes");
            vd.listarClientes(conn);
        }
    }
}
```



### Exemplo 1 - continuação



```
// *** Carregar o cliente 1003 a partir do bd ***
cl = new Cliente(1003);
System.out.println("\nLista o 1003 antes de carregar os dados");
System.out.println(cl);
cl.carregar(conn);
System.out.println("\nLista o 1003 depois de carregar os dados");
System.out.println(cl);
// *** Excluir o cliente 1003 (carregado em cl) do bd
cl.excluir(conn);

// *** IMPORTANTE ***: Efetiva exclusão
conn.commit();

// *** Lista novamente todos os clientes
System.out.println("\nLista todos os clientes depois de apagar o 1003");
vd.listarClientes(conn);
}
catch (Exception e) {
    e.printStackTrace();
    if (conn != null) {
        try {
            conn.rollback();
        }
        catch (SQLException e1) {
            System.out.print(e1.getStackTrace());
        }
    }
}
finally {
    if (conn != null) {
        try {
            conn.close();
        }
        catch (SQLException e1) {
            System.out.print(e1.getStackTrace());
        }
    }
}
}
```

## Conexão com BD usando JDBC

### Exemplo 2



- Um aplicativo exemplo, em Java, capaz de manter uma lista de usuários e senha, conectando-se ao BD **MySQL**, é apresentado no arquivo anexo denominado ***JavaMySqlApp.rar***.



### Exercício 1

- Desenvolver um aplicativo em Java, com conexão ao MySQL, capaz de manter as notas dos alunos de uma sala de aula qualquer, tendo os seguintes campos obrigatórios para cada aluno cadastrado: ID do aluno, RA do aluno, Nome do aluno, Código da matéria, Nome da matéria, nota P1 da matéria, nota P2 da matéria, média Ms da matéria:  $Ms = (P1 + P2)/2$



### Exercício 2

- Desenvolver um aplicativo em Java, com conexão ao MySQL, capaz de manter um cadastro de animais e suas principais características (uma ou muitas para cada animal), para ser utilizado em um jogo de aprendizado de máquina;
- Não é necessário implementar o jogo, apenas a parte de manutenção do referido cadastro.



# ECM251 – Linguagens de Programação I

## Conexão com BD usando JDBC

### Exercícios Extras



- Propostos pelo professor em aula, utilizando os conceitos abordados neste material...

# ECM251 – Linguagens de Programação I

## Aula 10 – L1/1, L2/1 e L3/1

### Bibliografia Básica



- MILETTO, Evandro M.; BERTAGNOLLI, Silvia de Castro. Desenvolvimento de software II: introdução ao desenvolvimento web com HTML, CSS, javascript e PHP (Tekne). Porto Alegre: Bookman, 2014. E-book. Referência Minha Biblioteca:  
<https://integrada.minhabiblioteca.com.br/#/books/9788582601969>
- WINDER, Russel; GRAHAM, Roberts. Desenvolvendo Software em Java, 3ª edição. Rio de Janeiro: LTC, 2009. E-book. Referência Minha Biblioteca:  
<https://integrada.minhabiblioteca.com.br/#/books/978-85-216-1994-9>
- DEITEL, Paul; DEITEL, Harvey. Java: how to program early objects. Hoboken, N. J: Pearson, c2018. 1234 p. ISBN 9780134743356.

*Continua...*

### Bibliografia Básica (continuação)



- HORSTMANN, Cay S; CORNELL, Gary. Core Java. SCHAFRANSKI, Carlos (Trad.), FURMANKIEWICZ, Edson (Trad.). 8. ed. São Paulo: Pearson, 2010. v. 1. 383 p. ISBN 9788576053576.
- LIANG, Y. Daniel. Introduction to Java: programming and data structures comprehensive version. 11. ed. New York: Pearson, c2015. 1210 p. ISBN 9780134670942.
- TURINI, Rodrigo. Desbravando Java e orientação a objetos: um guia para o iniciante da linguagem. São Paulo: Casa do Código, [2017]. 222 p. (Caelum).

### Bibliografia Complementar



- HORSTMANN, Cay. Conceitos de Computação com Java. Porto Alegre: Bookman, 2009. E-book. Referência Minha Biblioteca:  
<https://integrada.minhabiblioteca.com.br/#/books/9788577804078>
- MACHADO, Rodrigo P.; FRANCO, Márcia H. I.; BERTAGNOLLI, Silvia de Castro. Desenvolvimento de software III: programação de sistemas web orientada a objetos em java (Tekne). Porto Alegre: Bookman, 2016. E-book. Referência Minha Biblioteca:  
<https://integrada.minhabiblioteca.com.br/#/books/9788582603710>
- BARRY, Paul. Use a cabeça! Python. Rio de Janeiro: Alta Books, 2012. 458 p.  
ISBN 9788576087434.

*Continua...*

# ECM251 – Linguagens de Programação I

## Aula 10 – L1/1, L2/1 e L3/1

### Bibliografia Complementar (continuação)



- LECHETA, Ricardo R. Web Services RESTful: aprenda a criar Web Services RESTful em Java na nuvem do Google. São Paulo: Novatec, c2015. 431 p.  
ISBN 9788575224540.
- SILVA, Maurício Samy. JQuery: a biblioteca do programador. 3. ed. rev. e ampl. São Paulo: Novatec, 2014. 544 p.  
ISBN 9788575223871.
- SUMMERFIELD, Mark. Programação em Python 3: uma introdução completa à linguagem Python. Rio de Janeiro: Alta Books, 2012. 506 p.  
ISBN 9788576083849.

*Continua...*



### Bibliografia Complementar (continuação)



- YING, Bai. Practical database programming with Java. New Jersey: John Wiley & Sons, c2011. 918 p.
- ZAKAS, Nicholas C. The principles of object-oriented JavaScript. San Francisco, CA: No Starch Press, c2014. 97 p. ISBN 9781593275402.



# ECM251 – Linguagens de Programação I

## Aula 10 – L1/1, L2/1 e L3/1

FIM

*Conexão com BD usando JDBC*

***Engenharia da Computação – 3ª série***

***Conexão com BD usando JDBC*  
*(L1/2, L2/2 e L3/2)***

**2025**

# ECM251 – Linguagens de Programação I

## Aula 10 – L1/2, L2/2 e L3/2

### Horário

Terça-feira: 2 x 2 aulas/semana

- L1/1 (07h40min-09h20min): *Prof. Calvetti*;
- L1/2 (09h30min-11h10min): *Prof. Calvetti*;
- L2/1 (07h40min-09h20min): *Prof. Menezes*;
- L2/2 (11h20min-13h00min): *Prof. Calvetti*;
- L3/1 (09h30min-11h10min): *Prof. Evandro*;
- L3/2 (11h20min-13h00min): *Prof. Evandro*.

### Exemplo



- Crie a classe **Livro** com três construtores (padrão, que recebe só **idLivro** e que recebe todos os parâmetros), métodos de acesso e modificadores e os atributos privados **titulo**, do tipo **String**, **edicao**, do tipo **int** e **idLivro**, do tipo **int**. Crie o método **toString()** que retorna o valor dos atributos.
- Crie os métodos de persistência da classe **Livro**:
  - *public void inserir(Connection conn);*
  - *public void alterar(Connection conn);*
  - *public void excluir(Connection conn);*
  - *public void carregar(Connection conn);*

# ECM251 – Linguagens de Programação I

## Conexão com BD usando JDBC

### Exemplo



```
1 import java.sql.Connection;
2 import java.sql.PreparedStatement;
3 import java.sql.ResultSet;
4 import java.sql.SQLException;
5
6 public class Livro
7 { private int idLivro;
8   private String titulo;
9   private int edicao;
10
11   public Livro()
12   {
13   }
14   public Livro(int idLivro)
15   { this.idLivro = idLivro;
16   }
17   public Livro(int idLivro, String titulo, int edicao)
18   { this.idLivro = idLivro;
19     this.titulo = titulo;
20     this.edicao = edicao;
21   }
22
23   public int getIdLivro()
24   { return idLivro;
25   }
26
```

I

# ECM251 – Linguagens de Programação I

## Conexão com BD usando JDBC

### Exemplo



```
27 public String getTitulo()
28 { return titulo;
29 }
30
31 public int getEdicao()
32 { return edicao;
33 }
34
35 public void setIdLivro(int idLivro)
36 { this.idLivro = idLivro;
37 }
38
39 public void setTitulo(String titulo)
40 { this.titulo = titulo;
41 }
42
43 public void setEdicao(int edicao)
44 { this.edicao = edicao;
45 }
46
47 public String toString()
48 { return "Livro [idLivro=" + idLivro + ", titulo=" + titulo + ", edicao=" + edicao + "];"
49 }
50
```

# ECM251 – Linguagens de Programação I

## Conexão com BD usando JDBC

### Exemplo



```
51 public void incluir(Connection conn)
52 { String sqlInsert = "INSERT INTO livro(idLivro, Titulo, Edicao) VALUES (?, ?, ?)";
53
54     PreparedStatement stm = null;
55     try
56     { stm = conn.prepareStatement(sqlInsert);
57       stm.setInt(1, getIdLivro());
58       stm.setString(2, getTitulo());
59       stm.setInt(3, getEdicao());
60       stm.execute();
61     }
62     catch (Exception e)
63     { e.printStackTrace();
64       try
65       { conn.rollback();
66       }
67       catch (SQLException e1)
68       { System.out.print(e1.getStackTrace());
69       }
70     }
```

# ECM251 – Linguagens de Programação I

## Conexão com BD usando JDBC

### Exemplo



```
71     finally
72     {   if (stm != null)
73         {   try
74             {   stm.close();
75                 }
76             catch(SQLException e1)
77             {   System.out.print(e1.getStackTrace());
78                 }
79             }
80     }
81 }
82
```



# ECM251 – Linguagens de Programação I

## Conexão com BD usando JDBC

### Exemplo



```
83 public void excluir(Connection conn)
84 { String sqlDelete = "DELETE FROM LIVRO WHERE idLivro = ?";
85   PreparedStatement stm = null;
86   try
87   { stm = conn.prepareStatement(sqlDelete);
88     stm.setInt(1, getIdLivro());
89     stm.execute();
90   }
91   catch(Exception e)
92   { e.printStackTrace();
93     try
94     { conn.rollback();
95     }
96     catch(SQLException e1)
97     { System.out.print(e1.getStackTrace());
98     }
99   }
```

# ECM251 – Linguagens de Programação I

## Conexão com BD usando JDBC

### Exemplo



```
100     finally
101     { if (stm != null)
102       { try
103         { stm.close();
104         }
105         catch(SQLException e1)
106         { System.out.print(e1.getStackTrace());
107         }
108       }
109     }
110 }
111
```

# ECM251 – Linguagens de Programação I

## Conexão com BD usando JDBC

### Exemplo



```
112 public void atualizar(Connection conn)
113 { String sqlUpdate = "UPDATE LIVRO SET Titulo = ?, Edicao = ? WHERE IdLivro = ?";
114   PreparedStatement stm = null;
115   try
116   { stm = conn.prepareStatement(sqlUpdate);
117     stm.setString(1, getTitulo());
118     stm.setInt(2, getEdicao());
119     stm.setInt(3, getIdLivro());
120     stm.execute();
121   }
122   catch(Exception e)
123   { e.printStackTrace();
124     try
125     { conn.rollback();
126     }
127     catch(SQLException e1)
128     { System.out.print(e1.getStackTrace());
129     }
130   }
```

# ECM251 – Linguagens de Programação I

## Conexão com BD usando JDBC

### Exemplo



```
131     finally
132     { if(stm != null)
133       { try
134         { stm.close();
135         }
136         catch(SQLException e1)
137         { System.out.print(e1.getStackTrace());
138         }
139       }
140     }
141 }
142
```



# ECM251 – Linguagens de Programação I

## Conexão com BD usando JDBC

### Exemplo



```
143 public void carregar(Connection conn)
144 { String sqlSelect = "SELECT Titulo, Edicao FROM LIVRO WHERE idLivro = ?";
145   PreparedStatement stm = null;
146   ResultSet rs = null;
147   try
148   { stm = conn.prepareStatement(sqlSelect);
149     stm.setInt(1, getIdLivro());
150     rs = stm.executeQuery();
151     if (rs.next())
152     { this.setTitulo(rs.getString(1));
153       this.setEdicao(rs.getInt(2));
154     }
155   }
156   catch(Exception e)
157   { e.printStackTrace();
158     try
159     { conn.rollback();
160     }
161     catch(SQLException e1)
162     { System.out.print(e1.getStackTrace());
163     }
164   }
```

# ECM251 – Linguagens de Programação I

## Conexão com BD usando JDBC

### Exemplo



```
165     finally
166     { if(rs != null)
167       { try
168         { rs.close();
169         }
170         catch(SQLException e1)
171         { System.out.print(e1.getStackTrace());
172         }
173       }
174       if(stm != null)
175       { try
176         { stm.close();
177         }
178         catch(SQLException e1)
179         { System.out.print(e1.getStackTrace());
180         }
181       }
182     }
183   }
184 }
185
```

# ECM251 – Linguagens de Programação I

## Conexão com BD usando JDBC

### Exemplo



```
1 import java.sql.Connection;
2 import java.sql.DriverManager;
3 import java.sql.SQLException;
4
5 public class ConexaoBD
6 { static
7   { try
8     { Class.forName("com.mysql.jdbc.Driver");
9   }
10    catch(ClassNotFoundException e)
11    { throw new RuntimeException(e);
12    }
13  }
14
15  public Connection conectar() throws SQLException
16  { String servidor = "localhost";
17    String porta = "3306";
18    String database = "editora";
19    String usuario = "alunos";
20    String senha = "alunos";
21    return DriverManager.getConnection("jdbc:mysql://" + servidor + ":" + porta + "/" + database + "?user=" + usuario + "&password=" + senha);
22  }
23 }
24
```

# ECM251 – Linguagens de Programação I

## Conexão com BD usando JDBC

### Exemplo



```
1 import java.sql.SQLException;
2 import java.sql.Connection;
3
4 public class Teste
5 { public static void main(String[] args)
6   { Connection conn = null;
7     Livro livro;
8     try
9     { ConexaoBD bd = new ConexaoBD();
10       conn = bd.conectar();
11       // *** Inclusao do Primeiro Livro ***
12       livro = new Livro(1, "O Senhor dos Aneis", 32);
13       livro.incluir(conn);
14       System.out.println(livro);
15       // *** Inclusao do Segundo Livro ***
16       livro = new Livro();
17       livro.setIdLivro(2);
18       livro.setTitulo("1984");
19       livro.setEdicao(22);
20       livro.incluir(conn);
21       System.out.println(livro);
22       // *** Carregar o segundo livro a partir do bd ***
23       livro = new Livro(2);
24       System.out.println(livro);
25       livro.carregar(conn);
26       System.out.println(livro);
```



# ECM251 – Linguagens de Programação I

## Conexão com BD usando JDBC

### Exemplo



```
23     livro = new Livro(2);
24     System.out.println(livro);
25     livro.carregar(conn);
26     System.out.println(livro);
27     // *** Alterar o livro 2 (carregado em livro) do bd
28     livro.setTitulo("Admiravel Mundo Novo");
29     livro.setEdicao(3);
30     livro.atualizar(conn);
31     livro = new Livro(2);
32     livro.carregar(conn);
33     System.out.println(livro);
34 }
35 catch(Exception e)
36 { e.printStackTrace();
37   if(conn != null)
38   { try
39     { conn.rollback();
40     }
41     catch(SQLException e1)
42     { System.out.print(e1.getStackTrace());
43     }
44   }
45 }
```

# ECM251 – Linguagens de Programação I

## Conexão com BD usando JDBC

### Exemplo



```
46     finally
47     { if(conn != null)
48       { try
49         { conn.close();
50         }
51         catch(SQLException e1)
52         { System.out.print(e1.getStackTrace());
53         }
54       }
55     }
56   }
57 }
58
```

### Exercícios



1. Crie a classe **Professor** com três construtores (um padrão, um que receba só **matrícula** e outro que receba todos os parâmetros), métodos de acesso e modificadores e os atributos privados **nome**, do tipo ***String***, **idade**, do tipo ***int*** e **matricula**, do tipo ***int***. Crie o método ***toString()*** que retorna o valor dos atributos.

Crie os métodos de persistência da classe **Professor**:

- *public void inserir(Connection conn);*
- *public void alterar(Connection conn);*
- *public void excluir(Connection conn);*
- *public void carregar(Connection conn);*

### Exercícios



2. Crie a classe **Disciplina** com três construtores (um padrão, um que receba só **codigo** e outro que receba todos os parâmetros), métodos de acesso e modificadores e os atributos privados **nome**, do tipo **String**, **professores**, do tipo **ArrayList<Professor>**, **codigo**, do tipo **String**. Crie o método **toString()** que retorna o valor dos atributos.

Crie os métodos de persistência da classe **Disciplina**:

- *public void inserir(Connection conn);*
- *public void alterar(Connection conn);*
- *public void excluir(Connection conn);*
- *public void carregar(Connection conn);*

### Exercícios



3. Crie as classes **Cliente**, **ContaCorrente** e **Agencia** conforme abaixo:

a) A classe **Cliente** possui os atributos **nome** e **cpf**, ambos do tipo ***String***, e um atributo **conta** do tipo ***ContaCorrente***. Crie um construtor que recebe os atributos como parâmetros e os métodos de acesso e os modificadores;

### Exercícios



3. Crie as classes **Cliente**, **ContaCorrente** e **Agencia** conforme abaixo:

b) A classe **ContaCorrente** tem os atributos **numero** e **digito**, ambos **int**, o atributo **agencia** do tipo **Agencia** e o atributo **saldo** do tipo **double**. Crie um construtor que recebe os atributos como parâmetros e os métodos de acesso e os modificadores. Crie também um método **depositar()** que receba um parâmetro **double** com o valor do depósito e aumente o saldo da conta. Crie também um método **sacar()** que receba um parâmetro **double** com o valor do saque e diminua o saldo da conta.

### Exercícios



3. Crie as classes **Cliente**, **ContaCorrente** e **Agencia** conforme abaixo:

b) A conta não pode ficar negativa. Neste caso, deve ser dada uma mensagem que o saque não foi efetuado e o retorno deve ser zero. Caso contrário o retorno deve ser o valor sacado. Crie também um método ***consultarSaldo()*** que não recebe parâmetros e retorne o saldo. Crie, finalmente, um método ***imprimirSaldo()*** que imprima o número da conta corrente com dígito, o número da agência com dígito e o saldo da conta corrente.



### Exercícios



3. Crie as classes **Cliente**, **ContaCorrente** e **Agencia** conforme abaixo:

c) Ainda na classe **ContaCorrente**, o número da conta deve ter no máximo 4 dígitos e ser positivo. O dígito da conta deve ser validado a partir do seguinte algoritmo de módulo 11:

*“Multiplique o primeiro dígito da conta por 4, o segundo por 6, o terceiro por 8 e o quarto por 2; some tudo e calcule o resto da divisão (módulo) da soma por 11. Este é o valor do dígito”;*

- Obs: se o resultado for 10 o dígito é 0.



### Exercícios



3. Crie as classes **Cliente**, **ContaCorrente** e **Agencia** conforme abaixo:

d) A classe **Agencia** tem os atributos **nome** do tipo ***String***, **numero** e **digito** do tipo ***int***. Crie um construtor que recebe os atributos como parâmetros e os métodos de acesso e os modificadores. O número e o dígito da agência devem seguir os mesmos padrões do número e do dígito da conta corrente;

### Exercícios



3. Crie as classes **Cliente**, **ContaCorrente** e **Agencia** conforme abaixo:

e) Em cada uma das três classes crie os métodos CRUD de persistência. O método ***atualizar()*** sempre deve atualizar todos os campos da tabela, exceto a chave primária (**PK**). Não persista todos os campos. Faça conforme abaixo:

Cliente: **cpf (*pk*)**, **nome**

Conta Corrente: **numero (*pk*)**, **digito**

Agencia: **numero (*pk*)**, **digito**

# ECM251 – Linguagens de Programação I

## Conexão com BD usando JDBC

### Exercícios



Outro: Utilizar o tempo de aula restante para adiantar a codificação do Projeto I, semestral, solicitado na matéria, pelo professor.

### Bibliografia (apoio)



- LOPES, ANITA. GARCIA, GUTO. Introdução à Programação: 500 algoritmos resolvidos. Rio de Janeiro: Elsevier, 2002.
- DEITEL, P. DEITEL, H. Java: como programar. 8 Ed. São Paulo: Prentice-Hall (Pearson), 2010;
- BARNES, David J.; KÖLLING, Michael. Programação orientada a objetos com Java: uma introdução prática usando o BlueJ . 4. ed. São Paulo: Pearson Prentice Hall, 2009.

# ECM251 – Linguagens de Programação I

## Aula 10 – L1/2, L2/2 e L3/2

FIM