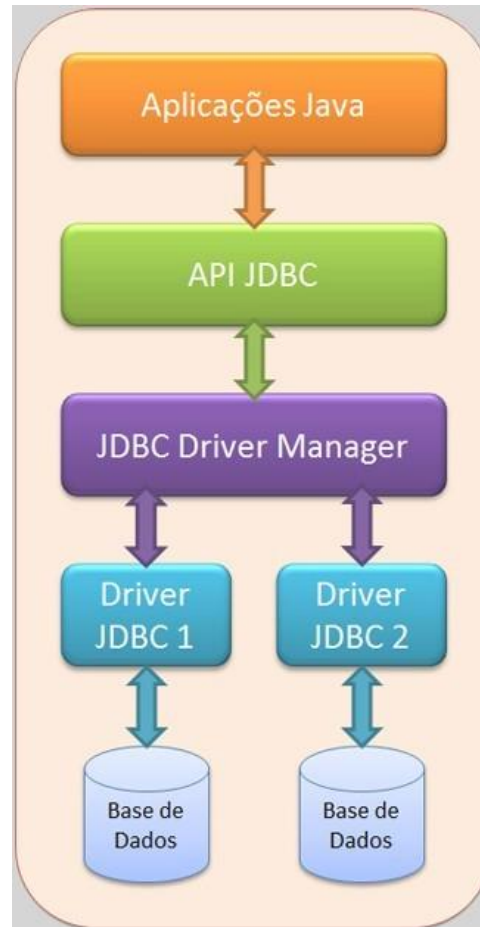


Disciplina: Aplicação de Linguagem de Programação Orientada a Objetos (ALPOO)

Java Database Connectivity (JDBC)



Arquitetura - JDBC



SQLException

Uma exceção pode ocorrer tanto no driver quanto no banco de dados. Quando uma exceção ocorre, um objeto do tipo SQLException é lançada e deve ser tratada no bloco catch.

O objeto da exceção do tipo SQLException possui alguns métodos para obter mais informações sobre a exceção lançada.

SQLException

Método	Descrição
getErrorCode()	Retorna o número associado à exceção
getMessage()	Retorna a mensagem de erro do driver JDBC (se o erro ocorreu no driver) ou do banco de dados
getSQLState()	Retorna o um código de erro definido na especificação XOPEN SQLState
getNextException()	Retorna a próxima exceção na cadeia de exceções
printStackTrace()	Imprime a exceção e sua pilha de execução

Tratamento de exceções

Ao utilizar a informação disponível no objeto da exceção, o programador pode adicionar um tratamento para aquele erro e continuar com a execução do programa.

Um bloco de try-catch-finally é feito da seguinte forma:

```
try {  
    // Aqui vai o código que pode lançar exceções  
} catch(Exception ex) {  
    // Se ocorrer uma exceção do tipo indicado entre parenteses  
    // Esse trecho de código é executado. Ao se utilizar a  
    // exceção mais genérica Exception, qualquer exceção é tratada  
} finally {  
    // Esse trecho de código é executado sempre.  
    // Aqui deve ser adicionado o código para fazer a limpeza antes  
    // de finalizar a execução, como por exemplo fechar uma conexão  
    // com o banco de dados.  
}
```

Tipos de dados JDBC

SQL	JDBC/Java	setXXX	updateXXX
VARCHAR	java.lang.String	setString	updateString
CHAR	java.lang.String	setString	updateString
LONGVARCHAR	java.lang.String	setString	updateString
BIT	boolean	setBoolean	updateBoolean
NUMERIC	java.math.BigDecimal	setBigDecimal	updateBigDecimal
TINYINT	byte	setByte	updateByte
SMALLINT	short	setShort	updateShort
INTEGER	int	setInt	updateInt
BIGINT	long	setLong	updateLong
REAL	float	setFloat	updateFloat
FLOAT	Float	setFloat	updateFloat

Tipos de dados JDBC

SQL	JDBC/Java	setXXX	updateXXX
DOUBLE	double	setDouble	updateDouble
VARBINARY	byte[]	setBytes	updateBytes
BINARY	byte[]	setBytes	updateBytes
DATE	java.sql.Date	setDate	updateDate
TIME	java.sql.Time	setTime	updateTime
TIMESTAMP	java.sql.Timestamp	setTimestamp	updateTimeStamp
CLOB	java.sql.Clob	setClob	updateClob
BLOB	java.sql.Blob	setBlob	updateBlob
ARRAY	java.sql.Array	setArray	updateArray
REF	java.sql.Ref	setRef	updateRef
STRUCT	java.sql.Struct	setStruct	updateStruct

JDBC – Processamento em batch

Processamento em batch permite que se agrupe comandos SQLs relacionados e submetê-los com apenas uma chamada ao banco de dados.

Quando se manda vários comandos para o banco de dados de uma só vez, reduz o overhead de comunicação, melhorando a performance.

Nem todos os drivers JDBC suportam essa operação. O método `DatabaseMetaData.supportsBatchUpdates()` deve ser usado para determinar se o banco de dados suporta operações em batch.

O método `addBatch(sql)` é usado para adicionar comandos SQL ao batch

O método `executeBatch()` é usado para executar todos os comandos adicionados previamente ao batch.

O método `clearBatch()` é usado para remover os comandos que foram adicionados ao batch.

JDBC – PreparedStatement

Normalmente utilizado para queries com parâmetro, mas também pode ser usada sem parâmetros.

A query é passada para o banco de dados para um pré-compilação.

Devido a esse pré-compilação, a query fica armazenada no banco e pode ser reutilizada para executar outras vezes de forma mais rápida que uma query normal.

Para cada execução, é possível passar valores diferentes para os parâmetros, que é muito melhor do que ter que criar uma query diferente cada vez que os parâmetros são alterados.

JDBC – PreparedStatement

Exemplo de um PreparedStatement:

```
// Especifica o driver JDBC usado para a conexao
// Nesse exemplo eh MySQL
Class.forName("com.mysql.jdbc.Driver");

// URL de conexao com o banco
String url = "jdbc:mysql://localhost:3306/world";

// Usuario de acesso ao banco
String login = "root";

// Senha de acesso ao banco
String senha = "password";

conn = DriverManager.getConnection(url, login, senha);

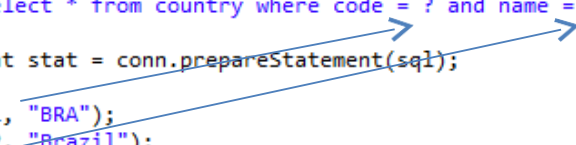
String sql = "select * from country where code = ? and name = ?";
PreparedStatement stat = conn.prepareStatement(sql);

stat.setString(1, "BRA");
stat.setString(2, "Brazil");

ResultSet rs = stat.executeQuery();

while(rs.next()) {
    String code = rs.getString("code");
    String nome = rs.getString("name");

    System.out.println("Cod: " + code + " Nome: " + nome);
}
```

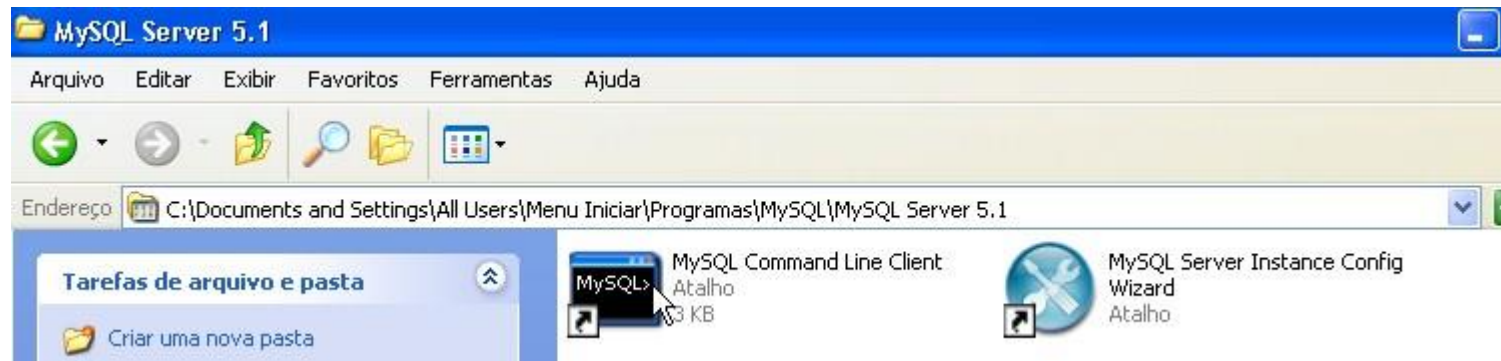


Os parâmetros (?) devem ser substituídos antes de se executar a Query.

Os métodos setString, setInt, setLong, etc são usados para substituir as variáveis da query. O primeiro parâmetro desse set é a posição da variável que será substituída.

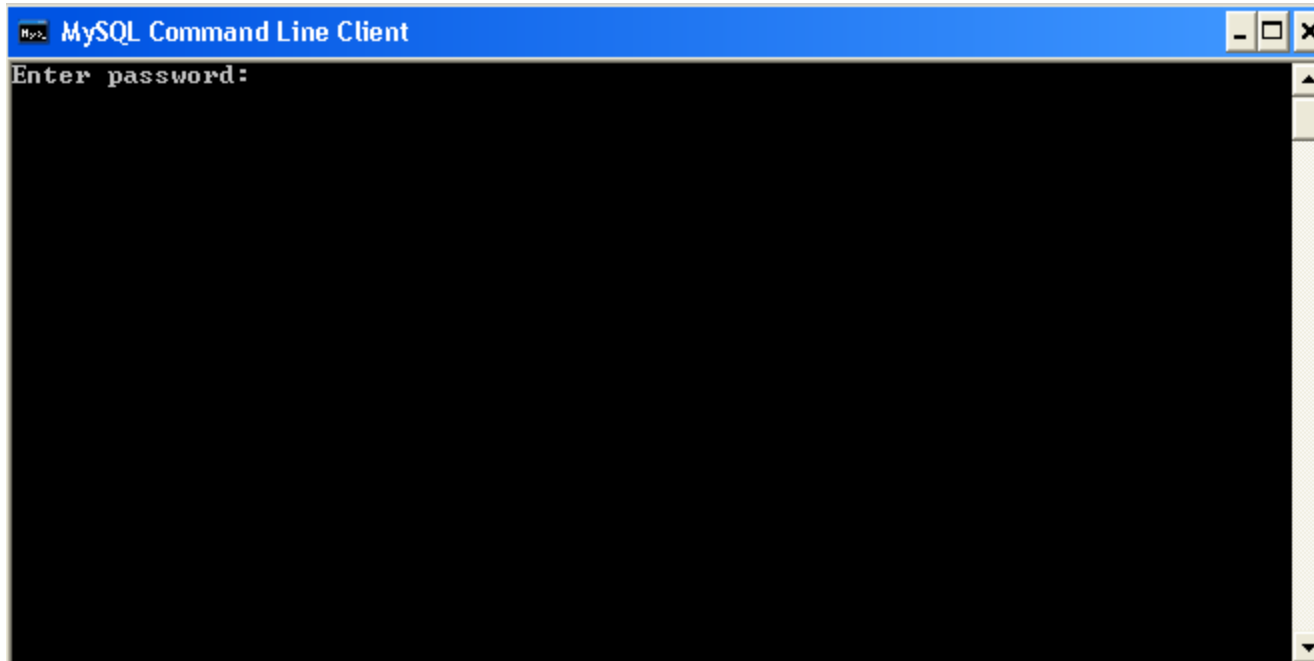
Preparando o Banco de Dados - MySQL

Iniciar o MySQL Command Line Client



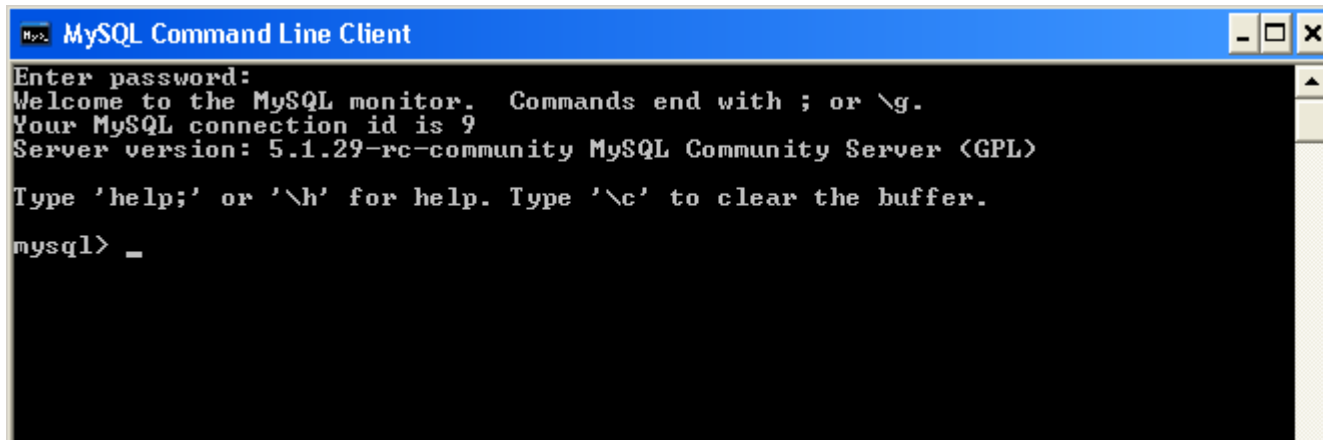
Preparando o Banco de Dados - MySQL

Digitar a senha de administrador do Banco de Dados



Preparando o Banco de Dados - MySQL

Verificar que a conexão foi feita com sucesso e a linha de comando do MySQL foi iniciada



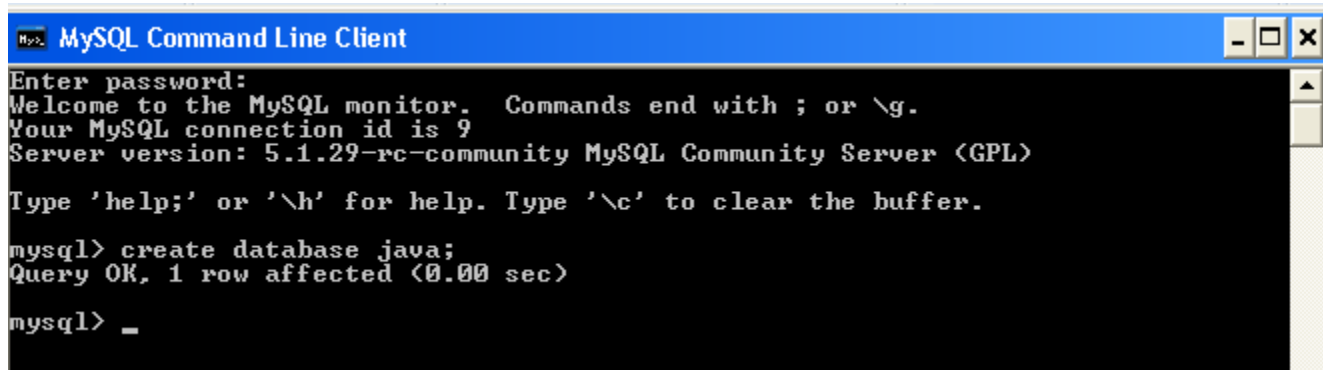
```
MySQL Command Line Client
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 5.1.29-rc-community MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> _
```

Preparando o Banco de Dados - MySQL

Criar uma base de dados nova para ser utilizada nas aulas



```
MySQL Command Line Client
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 5.1.29-rc-community MySQL Community Server (GPL)
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> create database java;
Query OK, 1 row affected (0.00 sec)

mysql> _
```

Preparando o Banco de Dados - MySQL

Alterar para a base de dados recém criada

```
mysql> use java;  
Database changed  
mysql>
```


Preparando o Banco de Dados - MySQL

Criar uma tabela para armazenar os dados da nossa janela (Cadastro de pessoa).

Será criada uma única tabela com todos os dados apenas para facilitar o entendimento.

```
mysql> create table cadastro(nome varchar(50), time varchar(50), sexo varchar(15), pratica_esporte boolean, pratica_arte boolean, endereco varchar(200));  
Query OK, 0 rows affected (0.09 sec)  
  
mysql> _
```

Preparando o Banco de Dados - MySQL

Executar o comando para verificar que a tabela foi criada com sucesso

```
mysql> desc cadastro;
```

Field	Type	Null	Key	Default	Extra
nome	varchar(50)	YES		NULL	
time	varchar(50)	YES		NULL	
sexo	varchar(15)	YES		NULL	
pratica_esporte	tinyint(1)	YES		NULL	
pratica_arte	tinyint(1)	YES		NULL	
endereco	varchar(200)	YES		NULL	

```
6 rows in set (0.08 sec)
```

```
mysql>
```

Preparando o Banco de Dados - MySQL

Também é possível executar queries SQL na tabela para verificar quando ela for populada

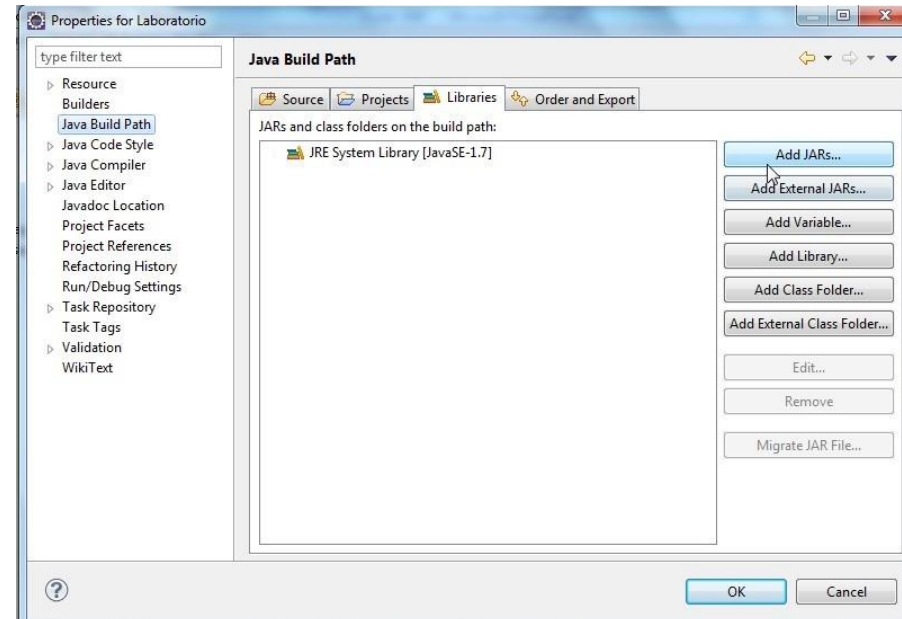
```
mysql> select * from cadastro;  
Empty set (0.00 sec)
```

Alterando a classe Janela para gravar os dados no Banco de Dados

O primeiro passo necessário é adicionar o jar do driver JDBC do banco de dados MySQL no projeto.

Entrar nas propriedades do Projeto (clcando com o botao direito no projeto e escolhendo a opção “Properties”).

Escolher a sub-opção Java Build Path



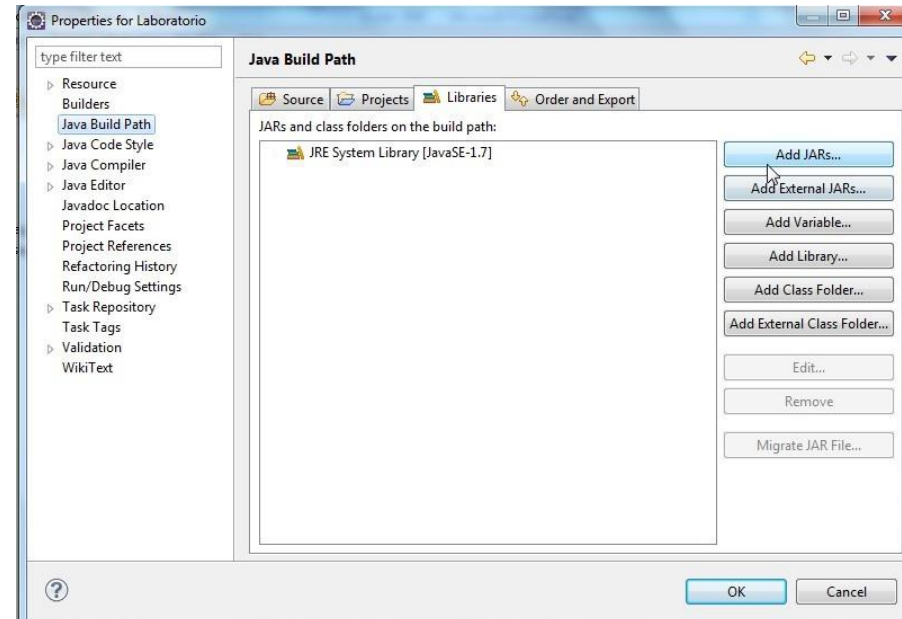
Alterando a classe Janela para gravar os dados no Banco de Dados

O primeiro passo necessário é adicionar o jar do driver JDBC do banco de dados MySQL no projeto.

Entrar nas propriedades do Projeto (clcando com o botao direito no projeto e escolhendo a opção “Properties”).

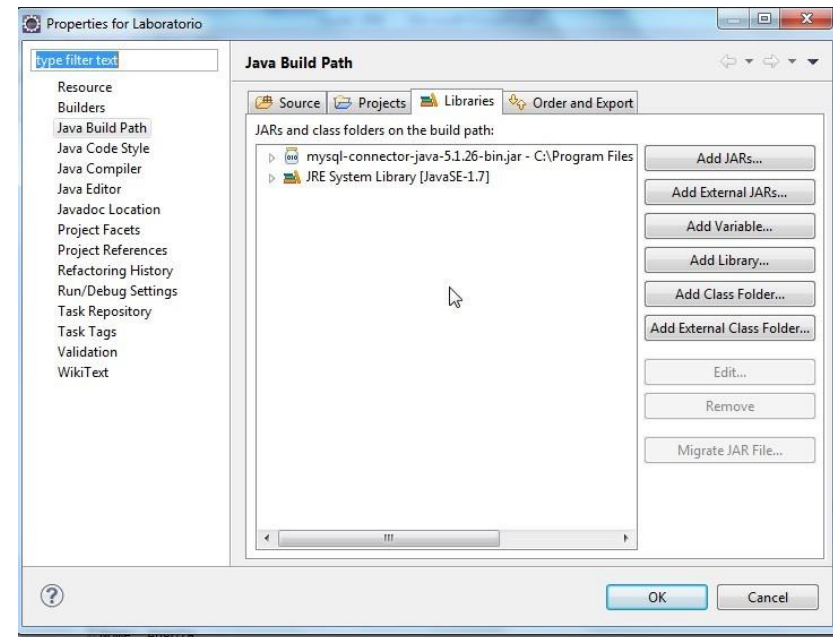
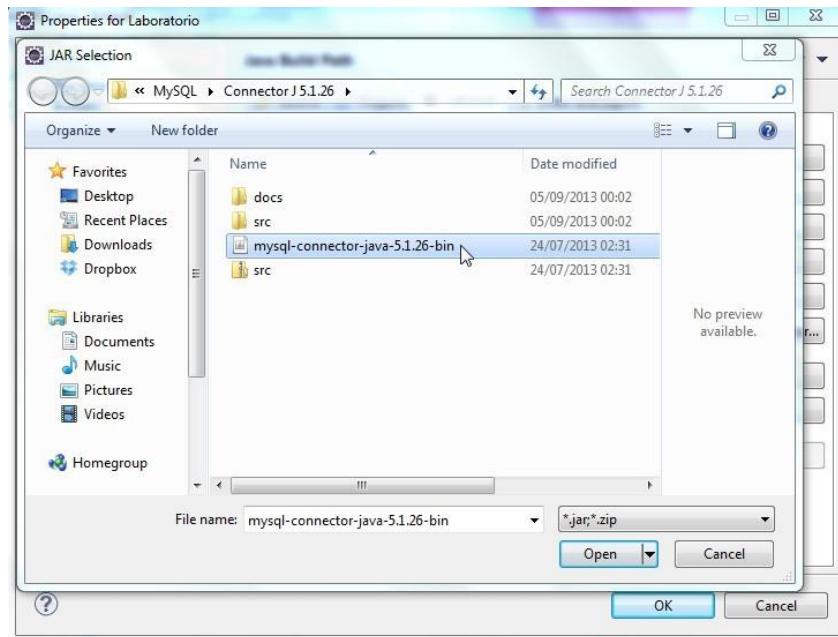
Escolher a sub-opção Java Build Path

Escolher Add External Jar



Alterando a classe Janela para gravar os dados no Banco de Dados

Apontar para a pasta onde está localizado o mysql connector



Criando variáveis de instância com as propriedades de conexão com o banco

```
public class Janela extends Frame {  
  
    private Label labelNome;  
    private TextField nome;  
    private Label labelTime;  
    private Choice time;  
    private Label labelSexo;  
    private Checkbox chkboxMasc;  
    private Checkbox chkboxFem;  
    private CheckboxGroup chkboxGroup;  
    private Label labelPratica;  
    private Checkbox esporte;  
    private Checkbox arte;  
    private Label labelEndereco;  
    private TextArea endereco;  
    private Button botaoGravar;  
    private Button botaoLimpar;  
    private Button botaoSair;  
    private Button botaoListar;  
  
    // URL de conexao com o banco  
    String url = "jdbc:mysql://localhost:3306/java";  
    // Usuario de acesso ao banco  
    String login = "root";  
    // Senha de acesso ao banco  
    String senha = "password";  
}
```

Criando método para gravar os dados no banco de dados na classe Janela

```
public void gravar() {  
    Connection conn = null;  
    try {  
        // Especifica o driver JDBC usado para a conexao  
        // Nesse exemplo eh MySQL  
        Class.forName("com.mysql.jdbc.Driver");  
  
        conn = DriverManager.getConnection(url, login, senha);  
  
        String sql = "INSERT INTO cadastro (nome, time, sexo, pratica_esporte, pratica_arte, endereco) ";  
        sql += "VALUES (?, ?, ?, ?, ?, ?) ";  
  
        PreparedStatement s = conn.prepareStatement(sql);  
  
        s.setString(1, nome.getText());  
        s.setString(2, time.getItem(time.getSelectedIndex()));  
        s.setString(3, chkboxGroup.getSelectedCheckbox().getLabel());  
        s.setBoolean(4, esporte.getState());  
        s.setBoolean(5, arte.getState());  
        s.setString(6, endereco.getText());  
  
        s.execute();  
  
        conn.close();  
  
    } catch (ClassNotFoundException e) {  
        System.out.println("Driver jdbc nao encontrado");  
        System.out.println(e.getMessage());  
    } catch (SQLException e) {  
        System.out.println("Problema na execucao da instrucao no banco de dados");  
        System.out.println(e.getMessage());  
    }  
}
```


Alterar na classe de tratamento de mouse o método a ser chamado pelo botão Gravar

```
public class TratamentoEventosMouse implements MouseListener {  
  
    private Janela janela;  
  
    public TratamentoEventosMouse(Janela janela) {  
        this.janela = janela;  
    }  
  
    @Override  
    public void mouseClicked(MouseEvent e) {  
        Button botao = (Button) e.getSource();  
  
        if(botao.getName().equals("botaoGravar")) {  
            // Aqui entra o código de tratamento do botão gravar  
            janela.gravar();  
        } else if(botao.getName().equals("botaoSair")) {  
            // Aqui entra o código de tratamento do botão sair  
            System.exit(1);  
        } else if(botao.getName().equals("botaoLimpar")) {  
            // Aqui entra o código de tratamento do botão limpar  
            janela.limpar();  
        }  
    }  
}
```

Executando a aplicação e tentando inserir algum dado no banco

The screenshot shows a Java Swing window titled "Título da Janela". The window contains a form with the following fields and controls:

- Nome:** A text field containing the value "Fernando".
- Time:** A text field.
- Local:** A dropdown menu showing "São Paulo".
- Sexo:** Two radio buttons: "Feminino" (unselected) and "Masculino" (selected).
- Pratica:** Two checkboxes: "Esporte" (checked) and "Arte" (unchecked).
- Endereco:** A text field containing the value "Rua da Paz 1234".

At the bottom of the window, there are three buttons: "Gravar", "Limpar", and "Sair".

Executando uma query no Banco de Dados

```
mysql> select * from cadastro;
+-----+-----+-----+-----+-----+-----+
| nome   | time   | sexo   | pratica_esporte | pratica_arte | endereco |
+-----+-----+-----+-----+-----+-----+
| Fernando | São Paulo | Masculino | 1 | 0 | Rua da Pa |
z 1234 |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>
```

Alterando o método imprimeRespostas para buscar os dados do banco de dados

```
public void imprimeRespostas() {  
    Connection conn = null;  
    try {  
        // Especifica o driver JDBC usado para a conexao  
        // Nesse exemplo eh MySQL  
        Class.forName("com.mysql.jdbc.Driver");  
  
        conn = DriverManager.getConnection(url, login, senha);  
  
        String sql = "SELECT * FROM cadastro";  
  
        PreparedStatement s = conn.prepareStatement(sql);  
  
        ResultSet rs = s.executeQuery();  
  
        JTable table = new JTable(montaModeloTabela(rs));  
  
        conn.close();  
  
        JOptionPane.showMessageDialog(null, new JScrollPane(table), "Listagem", JOptionPane.NO_OPTION);  
  
    } catch (ClassNotFoundException e) {  
        System.out.println("Driver jdbc nao encontrado");  
        System.out.println(e.getMessage());  
    } catch (SQLException e) {  
        System.out.println("Problema na execucao da instrucao no banco de dados");  
        System.out.println(e.getMessage());  
    }  
}
```

Criando o método montaModeloTabela na classe Janela

```
private TableModel montaModeloTabela(ResultSet rs) throws SQLException {
    ResultSetMetaData metaData = rs.getMetaData();
    // Cria um vetor com o nome das colunas retornadas pelo Select
    // Os nomes são obtidos a partir do metadados do retorno da consulta
    // e serão mostrados na tabela com o mesmo nome das colunas no banco.
    // Para alterar o valor é possível adicionar apelidos para as colunas
    // no select
    Vector<String> columnNames = new Vector<String>();
    int columnCount = metaData.getColumnCount();
    for (int column = 1; column <= columnCount; column++) {
        columnNames.add(metaData.getColumnName(column));
    }
    // Cria um vetor de vetores. Para cada linha retornada do banco de dados,
    // é criado um vetor com todos os valores das colunas daquela linha. Cada vetor
    // de linha é adicionado ao vetor principal de dados.
    Vector<Vector<Object>> data = new Vector<Vector<Object>>();
    while (rs.next()) {
        Vector<Object> vector = new Vector<Object>();
        for (int columnIndex = 1; columnIndex <= columnCount; columnIndex++) {
            vector.add(rs.getObject(columnIndex));
        }
        data.add(vector);
    }
    // Cria um TableModel, que é necessário para popular o componente JTable.
    // O DefaultTableModel recebe como parametro um vetor de vetores com os valores
    // das linhas retornadas pelo banco de dados, e um vetor com os nomes das colunas
    DefaultTableModel model = new DefaultTableModel(data, columnNames) {
        // Além de instanciar uma instancia de DefaultTableModel, para a nossa instancia
        // estamos sobrescrevendo o método isCellEditable de forma que nenhuma célula da tabela
        // seja editável. Por padrão todas as células da JTable são editáveis.
        @Override
        public boolean isCellEditable ( int row, int column )
        {
            return false;
        }
    };
    return model;
}
```

Esse método é usado para converter o ResultSet com o resultado da consulta ao banco de dados em um Vetor com o nome das colunas e com os dados das linhas retornadas pelo select.

Esses vetores são passados para a classe DefaultTableModel, que recebe um vetor de vetores (com os dados) e os nomes das colunas para montar a tabela

Adicionando um novo botão para Listar

```
public class Janela extends Frame {  
    private Label labelNome;  
    private TextField nome;  
    private Label labelTime;  
    private Choice time;  
    private Label labelSexo;  
    private Checkbox chkboxMasc;  
    private Checkbox chkboxFem;  
    private CheckboxGroup chkboxGroup;  
    private Label labelPratica;  
    private Checkbox esporte;  
    private Checkbox arte;  
    private Label labelEndereco;  
    private TextArea endereco;  
    private Button botaoGravar;  
    private Button botaoLimpar;  
    private Button botaoSair;  
    private Button botaoListar;
```

```
// Instancia os quatro botoes: Gravar, Limpar, Sair e Listar
```

```
botaoGravar = new Button("Gravar");  
botaoGravar.setName("botaoGravar");  
botaoLimpar = new Button("Limpar");  
botaoLimpar.setName("botaoLimpar");  
botaoSair = new Button("Sair");  
botaoSair.setName("botaoSair");  
botaoListar = new Button("Listar");  
botaoListar.setName("botaoListar");
```

```
// Instancia um objeto da classe TratamentoEventosMouse e adiciona esse objeto  
// como listener dos botoes
```

```
TratamentoEventosMouse eventosMouse = new TratamentoEventosMouse(this);
```

```
botaoGravar.addMouseListener(eventosMouse);  
botaoLimpar.addMouseListener(eventosMouse);  
botaoSair.addMouseListener(eventosMouse);  
botaoListar.addMouseListener(eventosMouse);
```

Alterar a classe tratadora de eventos de mouse

```
public class TratamentoEventosMouse implements MouseListener {  
  
    private Janela janela;  
  
    public TratamentoEventosMouse(Janela janela) {  
        this.janela = janela;  
    }  
  
    @Override  
    public void mouseClicked(MouseEvent e) {  
        Button botao = (Button) e.getSource();  
  
        if(botao.getName().equals("botaoGravar")) {  
            // Aqui entra o codigo de tratamento do botao gravar  
            janela.gravar();  
        } else if(botao.getName().equals("botaoSair")) {  
            // Aqui entra o codigo de tratamento do botao sair  
            System.exit(1);  
        } else if(botao.getName().equals("botaoLimpar")) {  
            // Aqui entra o codigo de tratamento do botao limpar  
            janela.limpar();  
        } else if(botao.getName().equals("botaoListar")) {  
            // Aqui entra o codigo de tratamento do botao limpar  
            janela.imprimeRespostas();  
        }  
    }  
}
```

Alterando o grid do painel de botões para ter 2 linhas e 2 colunas e adicionar o novo botão

```
// Cria um grid layout de 2 linha e 2 colunas para adicionar os botoes
GridLayout gridBotoes = new GridLayout(2, 2);
// Cria um painel para adicionar os botoes. Esse painel usara o grid criado acima
Panel painelBotoes = new Panel();
painelBotoes.setLayout(gridBotoes);

// Adiciona os botoes no painel criado
painelBotoes.add(botaoGravar);
painelBotoes.add(botaoLimpar);
painelBotoes.add(botaoSair);
painelBotoes.add(botaoListar);

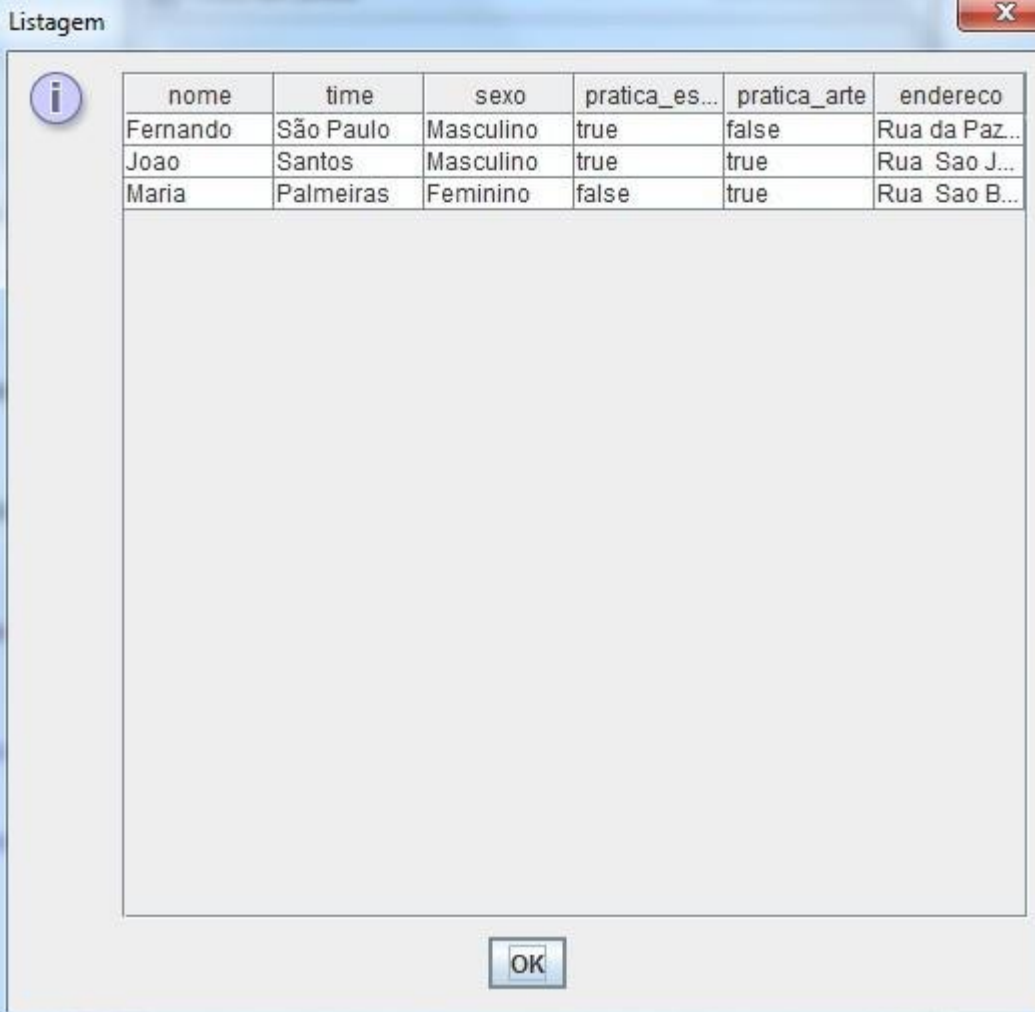
// Finalmente adiciona o painel na janela
this.add(painelBotoes);
```


Executando a aplicação

A screenshot of a Java Swing application window titled "Título da Janela". The window contains the following elements:

- A text field labeled "Nome:".
- A text field labeled "Time:".
- A dropdown menu labeled "Escolha o time" with a downward arrow.
- A label "Sexo:" followed by two radio buttons: "Feminino" and "Masculino".
- A label "Pratica:" followed by two checkboxes: "Esporte" and "Arte".
- A text field labeled "Endereco:" with the placeholder text "Digite seu endereço".
- A button grid at the bottom with four buttons: "Gravar", "Limpar", "Sair", and "Listar".

Executando a aplicação



Window Title: Listagem

nome	time	sexo	pratica_es...	pratica_arte	endereco
Fernando	São Paulo	Masculino	true	false	Rua da Paz...
Joao	Santos	Masculino	true	true	Rua Sao J...
Maria	Palmeiras	Feminino	false	true	Rua Sao B...

Buttons: OK