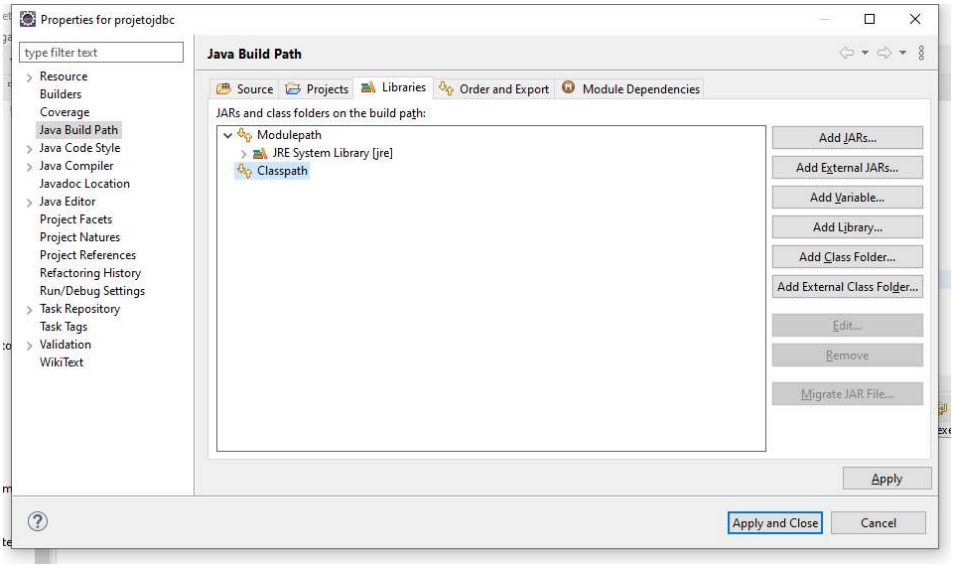
Exercícios JDBC

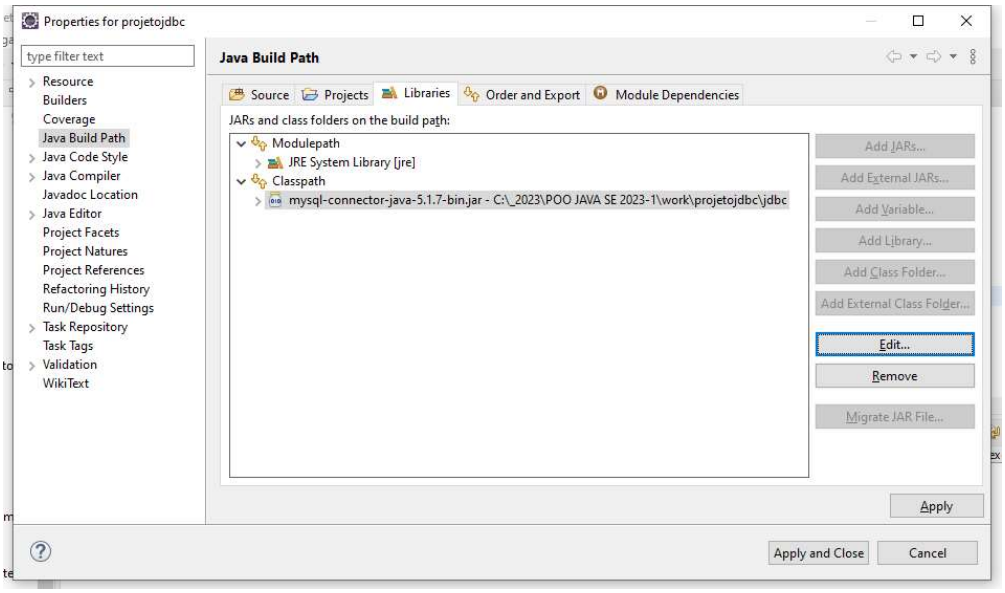
1) Utilize o projeto projetojdbc fornecido pelo professor

2) Verifique se o driver do Mysql está na pasta libs. Clique com o botão direito do mouse sobre o projeto e dê um refresh.

3) Agora precisamos vincular o Driver jdbc ao projeto. Clique sobre o projeto com o botão direito do mouse. Escolha BuidPath e Configure Build Path.



Em seguida clique em Add Jars. Entre na pasta libs do seu projeto e selecione o Driver.



Em seguida clique no botão Apply and Close. Voltando ao projeto, dê um novo refresh. Pronto! Agora seu Driver está vinculado ao projeto!

4) Abra o xampp, mande iniciar o serviço do mysql e copie todo o código de testebd.txt para a aba SQL do phpMyAdmin. Assim você estará criando a tabela pessoa na base testebd. Dê uma boa olhada na estrutura e nas linhas desta tabela.
5) Voltando ao projeto, observe bem a classe Pessoa.
6) Ainda no Projeto, vamos criar a fábrica de conexões. Clique em **File → New → Class**:

Crie-a no pacote **br.com.cefet.projetojdbc.modelo** como **ConnectionFactory.**
No código, crie o método getConnection( ) que obtem uma nova conexão. Quando perguntado, importe as classes do pacote java.sql (cuidado para não importar de com.mysql).

```
7  public class ConnectionFactory {
8      public Connection getConexao() {
9          String urlCon = "jdbc:mysql://localhost/testebd?UseUnicode=yes&characterEncoding=UTF-8";
10         try {
11             Class.forName("com.mysql.jdbc.Driver");
12             return DriverManager.getConnection(urlCon, "root", "");
13         } catch (SQLException | ClassNotFoundException e) {
14             throw new RuntimeException("Erro ao conectar com o bd."+e.getMessage());
15         }
16     }
17 }
```

7) Crie uma classe chamada TestaConexaoInsert no pacote br.com.cefet.projetojdbc.testes. Todas as nossas classes testes deverão ficar neste pacote.

```
11 public class TestaConexaoInsert {
12
13     public static void main(String[] args) {
14         Connection conexao = new ConnectionFactory().getConexao();
15
16         Pessoa p = new Pessoa("Oriel");
17         String sql = "INSERT INTO pessoa(nome) VALUES(?)";
18
19         try {
20             PreparedStatement stmt = conexao.prepareStatement(sql);
21             stmt.setString(1, p.getNome());
22             stmt.execute();
23             System.out.println("Pessoa "+p.getNome()+" inserida com sucesso!!");
24         } catch (SQLException e) {
25             System.out.println("Erro ao inserir. "+e.getMessage());
26         }
27     }
28 }
```

8) No mesmo pacote crie a classe TestaConexaoSelect

```
18 public class TestaConexaoSelect {
19     public static void main(String[] args) {
20         Connection con = new ConnectionFactory().getConexao();
21
22         String sql = "SELECT * FROM pessoa";
23         List<Pessoa> pessoas = new ArrayList<Pessoa>();
24         try {
25             PreparedStatement stmt = con.prepareStatement(sql);
26             ResultSet rs = stmt.executeQuery();
27             while(rs.next()) {
28                 Pessoa p = new Pessoa(rs.getString("nome"));
29                 p.setId(rs.getInt("id"));
30                 pessoas.add(p);
31             }
32         } catch (SQLException e) {
33             System.out.println("Erro ao listar."+e.getMessage());
34         }
35
36         //Mostra as linhas da tabela
37         for (Pessoa p : pessoas) {
38             System.out.println("Id: "+p.getId());
39             System.out.println("Nome: "+p.getNome());
40         }
41     }
42 }
```

9) Como não queremos trabalhar com várias conexões ao mesmo tempo, utilize o padrão Singleton para criar a classe BDSingleton.

```java
public class BDSingleton {
    private Connection conexao = null;
    private static BDSingleton instancia = null;
    //Construtor privado
    private BDSingleton() {
        String urlCon = "jdbc:mysql://localhost/testebd?UseUnicode=yes&characterEncoding=UTF-8";
        try {
            Class.forName("com.mysql.jdbc.Driver");
            this.conexao = DriverManager.getConnection(urlCon, "root", "");
        } catch (SQLException | ClassNotFoundException e) {
            throw new RuntimeException("Erro ao conectar com o bd."+e.getMessage());
        }
    }
    //synchronized garante acesso exclusivo ao método
    public static synchronized BDSingleton getInstancia() {
        if(BDSingleton.instancia==null)
            BDSingleton.instancia = new BDSingleton();
        return BDSingleton.instancia;
    }

    public Connection getConexao() {
        return this.conexao;
    }
}
```

10) Na classe TestaConexaoSelect, crie duas conexões com ConnectioFactory e imprima seus endereços de memória. Depois troque por BDSingleton e veja a diferença.

```java
public class TestaConexaoSelect {
    public static void main(String[] args) {
        Connection con = new ConnectionFactory().getConexao();
        Connection con2 = new ConnectionFactory().getConexao();
        System.out.println(con);
        System.out.println(con2);

    public static void main(String[] args) {
        Connection con = BDSingleton.getInstancia().getConexao();
        Connection con2 = BDSingleton.getInstancia().getConexao();
        System.out.println(con);
        System.out.println(con2);
```

Ficou clara a diferença? Converse com seu professor.

Exercícios JDBC – Parte 2

1) Caso tenha ficado alguma dúvida, peça ao seu professor para explicar novamente o padrão Repository ou DAO.
2) Crie a classe DAOException de tratamento obrigatório.

```java
public class DaoException extends Exception {
    public DaoException(String msg) {
        super(msg);
    }
}
```

3) Crie a interface Persistivel com todos os métodos obrigatórios de qualquer classe DAO. Todos os métodos estarão sujeitos a lançar DAOException. Utilize o conceito de Generics para que esta interface sirva para qualquer entidade que você venha a utilizar.

```java
public interface Persistivel<T> {
    public int insert(T obj) throws DaoException;
    public int update(T obj) throws DaoException;
    public int delete(int id) throws DaoException;
    public List<T> getAll() throws DaoException;
    public T getById(int id) throws DaoException;
}
```

4) Crie a classe abstrata PessoaDAOGeneric que implementa a interface Persistivel utilizando Pessoa para tipar nossos métodos. Esta classe deverá receber uma conexão por injeção de dependência.

```java
public abstract class PessoaDaoGeneric implements Persistivel<Pessoa>{
    protected Connection conexao = null;
    protected String instrucaoSql = null;
    protected PreparedStatement stmt = null;
    protected ResultSet rs = null;
    protected List<Pessoa> pessoas = null;

    public PessoaDaoGeneric(Connection conexao) {
        this.conexao = conexao;
    }

    @Override
    public int insert(Pessoa p) throws DaoException{
        int idGerado;
        this.instrucaoSql = "INSERT INTO pessoa(nome,data_nascimento) VALUES(?,?)";
        try {
            this.stmt = this.conexao.prepareStatement(this.instrucaoSql, Statement.RETURN_GENERATED_KEYS);
            this.stmt.setString(1, p.getNome());
            this.stmt.setDate(2, DataUtil.calendarToSqlDate(p.getDataDeNascimento()));
            this.stmt.execute();
            this.rs = this.stmt.getGeneratedKeys();
            this.rs.next();
            idGerado = this.rs.getInt(1);
            System.out.println("Pessoa " + p.getNome() + " inserida com sucesso!!");
        } catch (SQLException e) {
            throw new DaoException("Erro ao inserir." + e.getMessage());
        }
        return idGerado;
    }
```

```java
    @Override
    public int update(Pessoa p) throws DaoException {
        this.instrucaoSql = "UPDATE pessoa SET nome=?, data_nascimento=? WHERE id=?";
        int rowsAffected;
        try {
            this.stmt = this.conexao.prepareStatement(this.instrucaoSql);
            stmt.setString(1, p.getNome());
            stmt.setDate(2, DataUtil.calendarToSqlDate(p.getDataDeNascimento()));
            stmt.setInt(3, p.getId());
            rowsAffected = this.stmt.executeUpdate();
            System.out.println("Atualizção efetuada!! "+rowsAffected+" linhas afetadas.");
        } catch (SQLException e) {
            throw new DaoException("Erro ao atualizar." + e.getMessage());
        }
        return rowsAffected;
    }
```

```java
    @Override
    public int delete(int id) throws DaoException {
        this.instrucaoSql = "DELETE FROM pessoa WHERE id=?";
        int rowsAffected;
        try {
            this.stmt = this.conexao.prepareStatement(this.instrucaoSql);
            this.stmt.setInt(1, id);
            rowsAffected = this.stmt.executeUpdate();
            if(rowsAffected>0)
                System.out.println(rowsAffected+" pessoa(s) removida(s) com sucesso!!");
            else
                System.out.println("Não foi possível remover");
        } catch (SQLException e) {
            throw new DaoException("Erro ao remover." + e.getMessage());
        }
        return rowsAffected;
    }
```

```java
    @Override
    public List<Pessoa> getAll() throws DaoException {
        this.pessoas = new ArrayList();
        this.instrucaoSql = "SELECT * FROM pessoa";
        try {
            this.stmt = this.conexao.prepareStatement(this.instrucaoSql);
            this.rs = this.stmt.executeQuery();
            while (this.rs.next()) {
                Pessoa p = new Pessoa(this.rs.getString("nome"));
                p.setId(this.rs.getInt("id"));
                p.setDataDeNascimento(DataUtil.sqlDateToCalendar(rs.getDate("data_nascimento")));
                // Adiciono cada pessoa à lista de pessoas
                this.pessoas.add(p);
            }
        } catch (SQLException e) {
            throw new DaoException("Erro ao consultar." + e.getMessage());
        }
        return pessoas;
    }

    @Override
    public Pessoa getById(int id) throws DaoException {
        this.instrucaoSql = "SELECT * FROM pessoa WHERE id =?";
        Pessoa p=null;
        try {
            this.stmt = this.conexao.prepareStatement(this.instrucaoSql);
            this.stmt.setInt(1, id);
            this.rs = this.stmt.executeQuery();
            while (this.rs.next()) {
                p = new Pessoa(this.rs.getString("nome"));
                p.setId(this.rs.getInt("id"));
                p.setDataDeNascimento(DataUtil.sqlDateToCalendar(rs.getDate("data_nascimento")));
            }
        } catch (SQLException e) {
            throw new DaoException("Erro ao obter pessoa pelo id." + e.getMessage());
        }
        return p;
    }

}
```

5) Crie a classe PessoaDAO que é uma extensão de PessoaDAOGeneric e que contém ao menos um método mais específico.

```java
public class PessoaDao extends PessoaDaoGeneric{

    public PessoaDao(Connection conexao) {
        super(conexao);
    }

    public List<Pessoa> getByName(String name) throws DaoException {
        this.pessoas = new ArrayList();
        this.instrucaoSql = "SELECT * FROM pessoa WHERE nome LIKE ?";
        try {
            this.stmt = this.conexao.prepareStatement(this.instrucaoSql);
            this.stmt.setString(1, "%"+name+"%");
            this.rs = this.stmt.executeQuery();
            while (this.rs.next()) {
                Pessoa p = new Pessoa(this.rs.getString("nome"));
                p.setId(this.rs.getInt("id"));
                p.setDataDeNascimento(DataUtil.sqlDateToCalendar(rs.getDate("data_nascimento")));
                // Adiciono cada pessoa à lista de pessoas
                this.pessoas.add(p);
            }
        } catch (SQLException e) {
            throw new DaoException("Erro ao consultar." + e.getMessage());
        }
        return pessoas;
    }
}
```

6) Crie uma classe para testar as operações do DAO utilizando controle de transações (cujo funcionamento depende da versão da JVM).

```java
public class TestaDao {
    public static void main(String[] args) throws SQLException {
        Connection conexao = BDSingleton.getInstancia().getConexao();
        try {
            int linhasAfetadas;
            conexao.setAutoCommit(false);//Iniciando uma transação
            Pessoa p = new Pessoa("Alberto");
            p.setDataDeNascimento(DataUtil.stringToCalendar("08/09/2002"));
            PessoaDao dao = new PessoaDao(conexao);
            int ultimoId = dao.insert(p);
            System.out.println("ÚltimoId: "+ultimoId);
            Pessoa pessoaParaAlterar = dao.getById(2);
            pessoaParaAlterar.setNome("Renata Guimarães");
            linhasAfetadas = dao.update(pessoaParaAlterar);
            System.out.println("Linhas afetadas no update: "+linhasAfetadas);
            linhasAfetadas = dao.delete(6);
            System.out.println("Linhas afetadas no delete: "+linhasAfetadas);
            p = new Pessoa("Fulano 3");
            p.setDataDeNascimento(DataUtil.stringToCalendar("28/01/2000"));
            dao.insert(p);
            List<Pessoa> pessoas = dao.getAll();
            System.out.println();
            //Obter pessoa(s) por busca aproximada
            pessoas = dao.getByName("na");

              System.out.println("Lista retornada:");
              for (Pessoa pessoa : pessoas) {
                  System.out.println("Id: " + pessoa.getId());
                  System.out.println("Nome: " + pessoa.getNome());
                  System.out.println("Data de Nascimento: "+DataUtil.calendarToString(pessoa.getDataDeNascimento()));
                  System.out.println();
              }
              conexao.commit();//Confirmando a transação
              System.out.println("Coisa linda!! Transação efetuada!!");
          } catch (DaoException | SQLException e) {
              conexao.rollback();//Cancelando e voltando ao estado original
              System.out.println(e.getMessage());
          }finally {
                  conexao.setAutoCommit(true);//Tirando do modo de transação
                  conexao.close();
          }
      }
}
```

Na primeira execução tudo vai funcionar normalmente.

7) Vamos gerar um problema no meio da transação para forçar um rollback.

Abra o PhpMyadmin e execute o seguinte código:

```sql
USE testebd;
ALTER TABLE pessoa ADD UNIQUE INDEX idx_nome(nome);
```

A partir de agora, não aceitamos mais nomes repetidos. No código de TestaDao, faça as seguintes alterações:

Linha 20 → Troque o nome da pessoa para “Mariana”

Linha 26 → Troque o nome da pessoa para “Renata Irmã do Rafael”

Linha 29 → Troque o id para 1

Execute e perceba que os comandos não serão executados porque “Fulano 3” já existe. Consequentemente, Mariana não será inserida de fato, Renata não será alterada e a linha de id 1 não será apagada.