

Persistência em BD

usando Java

Vinicius Takeo Friedrich Kuwaki

Universidade do Estado de Santa Catarina

Seções

Exemplo

Resolução

Instalando Driver JDBC

Aplicação

Exercício

Exemplo

- A partir do banco de dados postgresql disponível nesse [link](#), crie uma interface via console com o usuário que permita realizar as operações no banco de dados de:
 - inserção de novas pessoas;
 - remoção de pessoas;
 - atualização atributos das pessoas;
 - busca de todas as pessoas existentes no banco;

Exemplo - Script

- Caso não deseje baixar o banco de dados, utilize o script de criação a seguir:

```
create sequence id_pessoa;  
create sequence id_endereco;  
  
create table pessoa(  
    id int,  
    nome varchar(50),  
    cpf int,  
    telefone int,  
    primary key (id)  
);  
  
create table endereco(  
    id int,  
    rua varchar(50),  
    numero integer,  
    cidade varchar(50),  
    id_pessoa integer,  
    primary key (id),  
    foreign key (id_pessoa) references pessoa  
);
```

Exemplo - Diagrama Geral

- Crie a aplicação em um sistema de camadas de acordo com os pacotes a seguir:

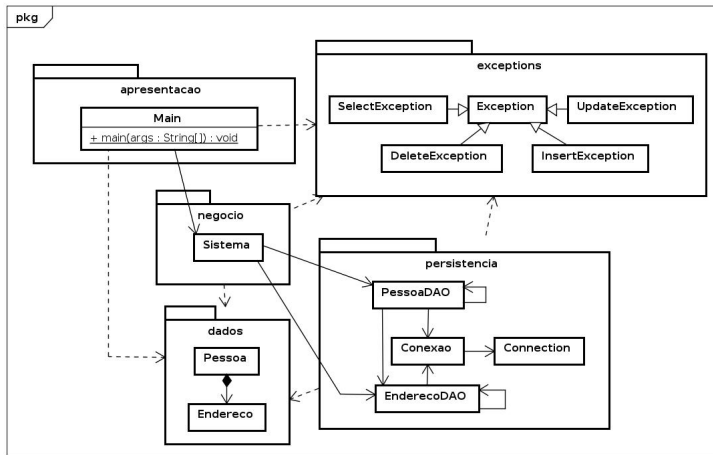


Figura 1: Diagrama UML contendo os pacotes. Veja a descrição dos pacotes a seguir:

Exemplo - Camada de Dados

- O pacote de dados contém os objetos com as informações manipuladas pela aplicação;
- Cada pessoa possui um relacionamento de composição com um endereço;
- Quando uma pessoa for removida do banco de dados, o endereço deve ser removido junto;

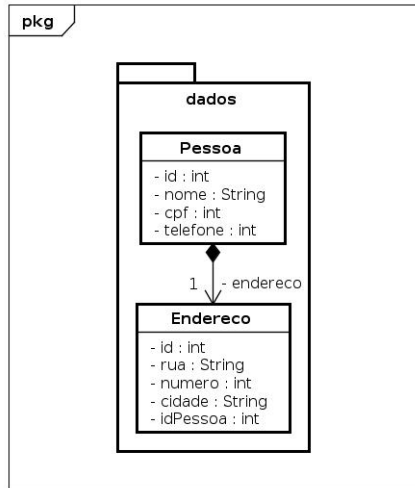


Figura 2: Pacote de dados

Exemplo - Camada de Exceções

- Crie as quatro exceções descritas no diagrama para serem lançadas pela camada de persistência;
- As exceções `ClassNotFoundException` e `SQLException` também devem ser utilizadas;

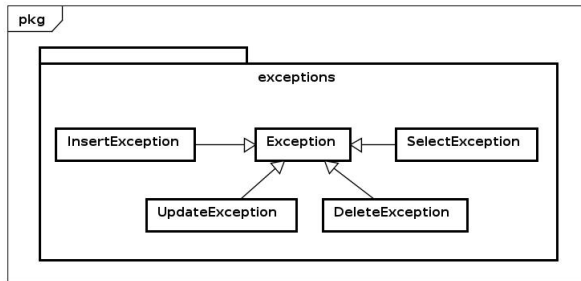
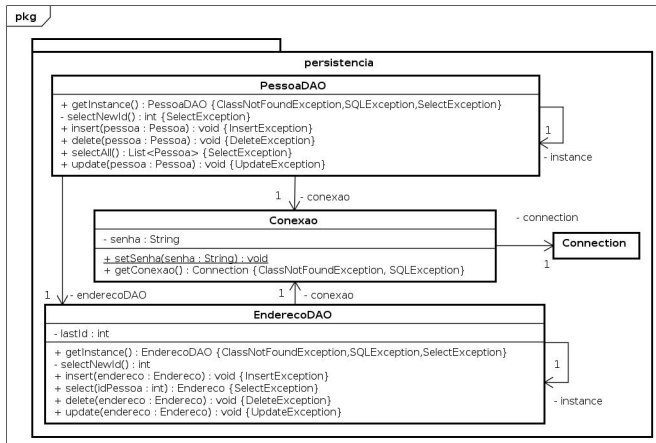


Figura 3: Exceções que estendem a classe `Exception`

Exemplo - Camada de Persistência

- A camada de persistência terá as seguintes classes:



Exemplo - Camada de Persistência

- A classe Conexão deve possuir um atributo estático para manter a senha do banco de dados;
- Esse atributo deve possuir um método setter;
- A classe também deve possuir um Singleton para o atributo conexão;
- No método **getConexao()**, caso o atributo seja null o método vai realizar a conexão com o banco de dados (veja a documentação do postgres);

Exemplo - Camada de Persistência

- Os métodos das classes DAO realizam a persistência dos objetos nas tabelas;
- O método **selectNewId()** deve buscar um novo id para um objeto a ser persistido;
 - Utilize o sql `select nextval('nome_da_sequencia')` para obter um novo valor de uma sequencia;
 - Esse método deve lançar uma `SelectException` caso algum erro ocorra. Informe a tabela onde ocorreu o erro na exceção;
- O método **insert()** recebe um novo objeto e persiste ele no banco de dados;
 - Utilize o sql `insert into nomeTabela (valor1, valor2, ..., valorN)`;
 - Esse método deve buscar uma nova chave e persistir o objeto na sua respectiva tabela;
 - Caso algum erro ocorra o método deve lançar uma `InsertException` informando a tabela em que o erro ocorreu;

Exemplo - Camada de Persistência

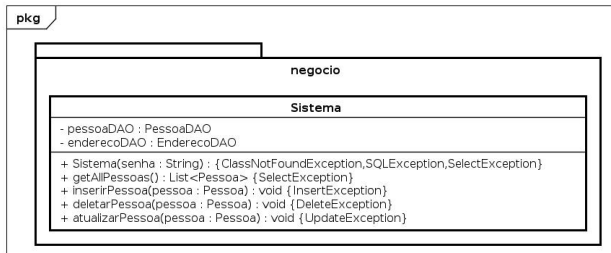
- O método **delete()** deve receber um objeto e removê-lo da sua respectiva tabela
 - Caso algum erro ocorra o método deve lançar uma `DeleteException` informando a tabela em que o erro ocorreu;
 - Utilize o sql `delete from nomeTabela where atributoX = valor;`
- O método **update()** também recebe um objeto e deve atualizar todos os atributos desse objeto em sua respectiva tabela
 - Utilize o sql `update nomeTabela set atributo1 = valor1, atributo2 = valor2, ..., atributoN = valorN where atributoX = valorX;`
 - Caso algum erro ocorra o método deve lançar uma `UpdateException` informando a tabela em que o erro ocorreu;

Exemplo - Camada de Persistência

- O método **selectAll()** deve retornar todos os objetos armazenados em uma tabela;
- Já o método **select()** deve retornar um objeto específico de acordo com o id passado como parâmetro
 - Utilize o sql **select * from nomeTabela** para obter todos os objetos de uma tabela;
 - Ou **select * from nomeTabela where atributoX = valorX** para obter o(os) objeto(s) de acordo com uma condição (no caso do exemplo: atributoX = valorX);
- Ambos os métodos de select devem lançar uma exceção do tipo **SelectException** caso algum erro ocorra. Informe a tabela onde ocorreu a exceção;

Exemplo - Camada de Negócio e Apresentação

- A camada de negócio deve atuar como Façade entre a camada de persistência e a de apresentação;
- Ela deve possuir os seguintes métodos:



- O método **main()** na camada de apresentação deve implementar a interface com o usuário para cada um dos métodos descritos na classe Sistema;

Exemplo

Resolução

Instalando Driver JDBC

Aplicação

Exercício

Instalando Driver

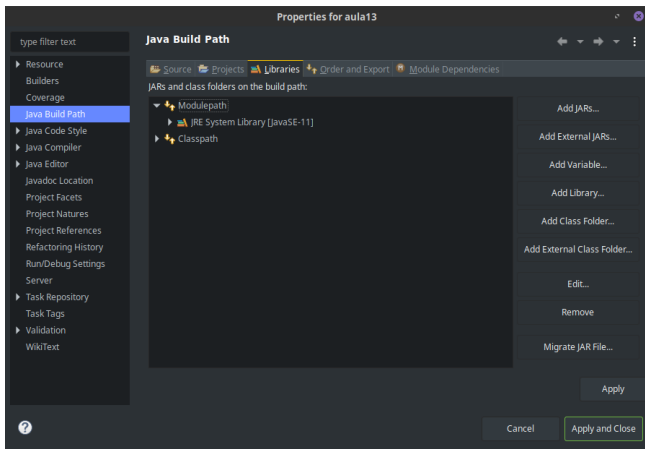
- Para começar a criar a aplicação, primeiro precisamos realizar o download do Driver JDBC;
- Acesse o link <https://jdbc.postgresql.org/download.html>;
- Baixe a versão mais recente do driver;
- Nesse material a versão utilizada foi: **PostgreSQL JDBC 4.2 Driver 42.2.12**;
- O Driver é um arquivo com extensão .jar.

Instalando Driver

- Agora precisamos importar o Driver no Build Path do projeto;
- No seu projeto Eclipse crie uma pasta chamada **libs**;
- Vamos manter as bibliotecas externas nessa pasta;
- Após criada a pasta, mova o Driver JDBC (arquivo .jar) para dentro dela;
- Acesse as propriedades do projeto pressionando ALT + ENTER;

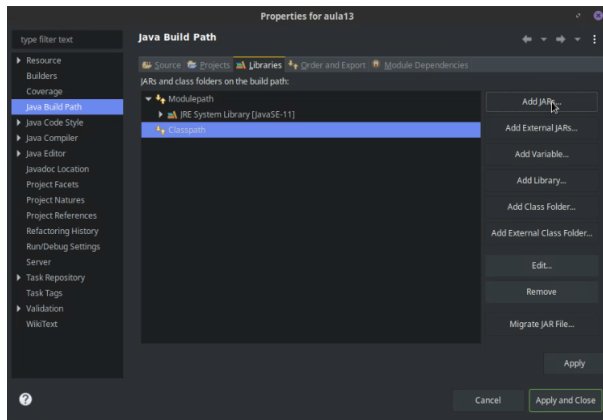
Instalando Driver

- Feito isso, selecione a opção **Java Build Path** no menu lateral;
- Selecione a aba **Libraries**:



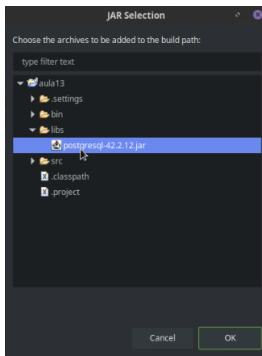
Instalando Driver

- Agora selecione o **Classpath** clicando com o botão esquerdo do mouse nele;
- Após isso selecione a opção **Add JARs**:



Instalando Driver

- Feito isso selecione o Driver e clique em Ok;
- No nosso caso, o arquivo .jar contido dentro da pasta **libs**.

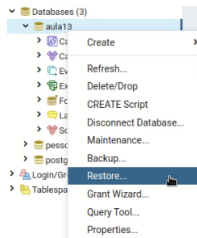


Instalando Driver

- Com o driver instalado podemos criar a aplicação;
- Utilizaremos o mesmo banco de dados apresentado nos slides Aula 13: Postresql;
- Caso você não tenha o banco de dados em seu computador, utilize o arquivo.backup disponível nesse [link](#).

Instalando Driver

- Crie um banco de dados no pgadmin4 e selecione a opção **Restore** para restaurá-lo:



- Faça upload do arquivo que contém o backup do banco de dados.

- Feita a criação do banco de dados, vamos começar a construção da aplicação;
- Primeiro, vamos fazer a **camada de dados**;
- Os métodos getters e setters devem ser implementados, mas não serão apresentados aqui.

Pacote de Dados - Classe Endereco

```
public class Endereco {  
  
    private int id;  
    private String rua;  
    private int numero;  
    private String cidade;  
    private int idPessoa;  
  
    ...  
  
}
```

- Note que o Endereço possui dois ids, o primeiro referente a si mesmo e o segundo referente ao objeto Pessoa que o possui;
- A única camada que de fato irá lidar com esses ids é a de persistência;

Pacote de Dados - Classe Endereco

- O mesmo vale para a classe Pessoa:

```
public class Pessoa {  
  
    private int id;  
    private String nome;  
    private int cpf;  
    private int telefone;  
    private Endereco endereco;
```

- O objeto Pessoa tem uma referência a um objeto Endereco, que representa seu endereço.

Pacote de Exceções

- Vamos criar as exceções que serão lançadas pela camada de persistência;
- Todas elas irão estender a classe `Exception` e terão apenas o construtor que recebe uma mensagem:

```
public class SelectException extends Exception {  
    public SelectException(String mensagem) {  
        super(mensagem);  
    }  
}
```

Pacote de Exceções

```
public class InsertException extends Exception {  
    public InsertException(String mensagem) {  
        super(mensagem);  
    }  
}  
  
public class UpdateException extends Exception {  
    public UpdateException(String mensagem) {  
        super(mensagem);  
    }  
}  
  
public class DeleteException extends Exception {  
    public DeleteException(String mensagem) {  
        super(mensagem);  
    }  
}
```

Pacote de Persistência - Classe Conexao

- No pacote de persistência, vamos começar criando a classe que realiza a conexão com o banco de dados;
- Em nosso caso, um banco de dados local;
- A classe será um Singleton, que tem como atributos uma instância de um objeto Connection (da biblioteca java.sql) e uma String com a senha do banco de dados;
- Ambos os atributos são estáticos;
- Para o atributo senha, teremos um método **setSenha()**:

```
public class Conexao {  
  
    private static Connection conexao = null;  
    private static String senha;  
  
    private Conexao() {}  
  
    public static void setSenha(String password) {  
        senha = password;  
    }  
}
```

Pacote de Persistência - Classe Conexao

- O construtor é privado;
- O método **getConexao()** é estático e irá retornar o atributo conexao;
- Esse método pode lançar duas exceções: *ClassNotFoundException* e *SQLException*;
- Para garantir que só haja uma instância da conexão, no método **getConexao()** verifica se a conexão existe, caso não exista ele a cria, e depois retorna ela:

```
public static Connection getConexao() throws ClassNotFoundException, SQLException {  
    if (conexao == null) {  
        // Proximos slides ...  
    }  
  
    return conexao;  
}
```

Pacote de Persistência - Classe Conexao

- Caso seja a primeira vez que o método é chamado a conexão ainda é **null**. Nesse caso, vamos criar uma conexão utilizando o método **getConnection()** da classe **DriverManager**;
- Para esse método iremos passar a *url*, *usuário* e *senha* do banco de dados;
- Entretanto, antes precisamos buscar a classe do Driver do banco de dados usando o método estático **forName()** da classe **Class**:

```
public static Connection getConexao() throws ClassNotFoundException, SQLException {  
    if (conexao == null) {  
        String url = "jdbc:postgresql://localhost:5432/pessoas";  
        String usuario = "postgres";  
        Class.forName("org.postgresql.Driver");  
        conexao = DriverManager.getConnection(url, usuario, senha);  
    }  
    return conexao;  
}
```

Pacote de Persistência - Classe Conexao

- Para descobrir a porta em que seu server está, entre nas propriedades do servidor e na aba **Connection** estará o *Host name* e a porta;
- Como estamos usando um servidor local, o *Host name* será *localhost*;
- Por padrão, a porta é 5432;
- Fique atento que se o servidor for local ele só funcionará no computador em que o servidor estiver!
- A [Amazon Web Services](#) possui servidores de PostgreSQL para testes de forma gratuita;

Pacote de Persistência - Classe EnderecoDAO

- Terminada a classe Conexão, vamos para o DAO do Endereço;
- Essa classe também será um Singleton;
- Para cada consulta no banco, teremos um atributo do tipo **PreparedStatement** que deixa a SQL pronta para ser usada:

```
public class EnderecoDAO {  
  
    private static EnderecoDAO instance = null;  
  
    private PreparedStatement selectNewId;  
    private PreparedStatement select;  
    private PreparedStatement insert;  
    private PreparedStatement delete;  
    private PreparedStatement update;  
  
    public static EnderecoDAO getInstance() throws ClassNotFoundException ,  
        SQLException, SelectException {  
        if (instance == null) {  
            instance = new EnderecoDAO();  
        }  
        return instance;  
    }  
}
```

Pacote de Persistência - Classe EnderecoDAO

- Vamos iniciar cada PreparedStatement dentro do construtor privado com sua respectiva SQL;
- Mas, para prepararmos o SQL no PreparedStatement precisamos utilizar um objeto Connection;
- Para isso, vamos chamar o método **getConexao()** da classe **Conexao** que criamos anteriormente:

```
private EnderecoDAO() throws ClassNotFoundException, SQLException, SelectException {  
    Connection conexao = Conexao.getConexao();  
}
```


Pacote de Persistência - Classe EnderecoDAO

- Primeiro vamos preparar o SQL para buscar novos ids;
- Utilizaremos a sequência **id_endereco** criada no banco de dados;
- Utilizaremos a função **nextval** para isso;
- E para preparar o SQL, vamos utilizar o método **prepareStatement()** que a classe Connection possui:

```
private EnderecoDAO() throws ClassNotFoundException, SQLException, SelectException {  
    Connection conexao = Conexao.getConexao();  
    selectNewId = conexao.prepareStatement("select nextval('id_endereco')");
```

Pacote de Persistência - Classe EnderecoDAO

- Por enquanto deixaremos o construtor assim, vamos primeiro construir o método que utilizara esse PreparedStatement e depois voltamos para o construtor;
- O método **selectNewId()** vai utilizar o método **executeQuery()** do PreparedStatement;
- Esse método lança uma exceção do tipo **SQLException** e retorna um **ResultSet** contendo a resposta da busca no banco de dados;
- Vamos capturar a exceção e lançá-la com sendo uma instância de **SelectException**;
- Caso o ResultSet contenha algum valor, isto é, o resultado da busca, vamos retornar esse valor utilizando o método **getInt()** passando a posição 1;

Pacote de Persistência - Classe EnderecoDAO

```
private int selectNewId() throws SelectException {  
    try {  
        ResultSet rs = selectNewId.executeQuery();  
        if (rs.next()) {  
            return rs.getInt(1);  
        }  
    } catch (SQLException e) {  
        throw new SelectException("Erro ao buscar novo id da tabela endere o");  
    }  
    return 0;  
}
```

- É necessário você saber a ordem em que os dados serão retornados em um ResultSet;
- Como esse SQL apenas retorna o id, é fácil saber que será o 1;
- **Atenção, o ResultSet é indexado a partir de 1 e não de 0!**
- Para saber a ordem dos campos no resultado contido do ResultSet, execute o seu SQL no pgAdmin e analise o resultado.

Pacote de Persistência - Classe EnderecoDAO

- Continuando a implementação, vamos voltar ao construtor e definir o SQL para inserir endereços no banco de dados;
- Vamos definir o SQL de insert;
- Deixaremos interrogações “?” nos campos que iremos substituir mais tarde:

```
private EnderecoDAO() throws ClassNotFoundException, SQLException, SelectException {  
    Connection conexao = Conexao.getConexao();  
    selectNewId = conexao.prepareStatement("select nextval('id_endereco')");  
    insert = conexao.prepareStatement("insert into endereco values (?, ?, ?, ?, ?)");  
}
```

- Lembre-se que o objeto insert é um PreparedStatement!

Pacote de Persistência - Classe EnderecoDAO

- O método que fará a inserção é o **insert()**;
- Esse método lança exceções do tipo `SelectException` já que ele irá utilizar o método **selectNewId()**;
- Também lança exceções do tipo `InsertException`, caso seja lançada alguma `SQLException`;
- Para substituímos as interrogações que definimos no `PreparedStatement` anteriormente, vamos utilizar o método **set()** para cada tipo de dado (`setInt`, `setString`, `setBoolean`, etc...);
- O método que utilizaremos é o **executeUpdate()** porque esse SQL altera o banco de dados;
- Na hora de passarmos o id a ser inserido, faremos uma chamada do método **selectNewId()**, definido anteriormente, para obter um novo valor de id válido;

Pacote de Persistência - Classe EnderecoDAO

```
public void insert(Endereco endereco) throws InsertException, SelectException {  
    try {  
        insert.setInt(1, selectNewId());  
        insert.setString(2, endereco.getRua());  
        insert.setInt(3, endereco.getNumero());  
        insert.setString(4, endereco.getCidade());  
        insert.setInt(5, endereco.getIdPessoa());  
        insert.executeUpdate();  
    } catch (SQLException e) {  
        throw new InsertException("Erro ao inserir endereço");  
    }  
}
```

- No momento de chamar os setters do PreparedStatement, fique atento também a ordem com a qual os campos estão dispostos na tabela no banco de dados;
- Para saber a ordem, execute um `select * from endereco` no pgAdmin;

Pacote de Persistência - Classe EnderecoDAO

- Vamos repetir o mesmo processo para o método **select()**;
- Primeiro vamos voltar ao construtor e definir o SQL do PreparedStatement;

```
private EnderecoDAO() throws ClassNotFoundException, SQLException, SelectException {  
    Connection conexao = Conexao.getConexao();  
    selectNewId = conexao.prepareStatement("select nextval('id_endereco')");  
    insert = conexao.prepareStatement("insert into endereco values (?, ?, ?, ?, ?)");  
    select = conexao.prepareStatement("select * from endereco where id_pessoa = ?");  
}
```


Pacote de Persistência - Classe EnderecoDAO

- O método **select()** precisa do id da pessoa que o endereço pertence;
- Esse valor é passado como parâmetro e depois inserido no PreparedStatement através do método **setInt()**;
- Feito isso, realizamos o select no banco de dados utilizando o método **executeQuery()**, porque esse SQL não altera o banco de dados;
- Se o ResultSet retornar algum registro, instanciamos um novo objeto do tipo Endereco e setamos seus atributos;
- Novamente, é necessário saber a ordem em que os campos estão contidos no ResultSet;
- Após a instanciação, retornamos o objeto;
- Caso uma exceção SQLException for capturada, lançamos uma SelectException no seu lugar.

Pacote de Persistência - Classe EnderecoDAO

```
public Endereco select(int pessoa) throws SelectException {  
    try {  
        select.setInt(1, pessoa);  
        ResultSet rs = select.executeQuery();  
        if (rs.next()) {  
            int id = rs.getInt(1);  
            String rua = rs.getString(2);  
            int numero = rs.getInt(3);  
            String cidade = rs.getString(4);  
            return new Endereco(id, rua, numero, cidade);  
        }  
    } catch (SQLException e) {  
        throw new SelectException("Erro ao buscar endere o da pessoa");  
    }  
    return null;  
}
```

Pacote de Persistência - Classe EnderecoDAO

- Note que utilizamos um id para realizar a busca;
- Mas, se tivéssemos utilizado um atributo qualquer?
- A busca teria retornado mais endereços;
- Teríamos uma lista ao invés de um único registro;
- Caso isso aconteça, substitua o `if(rs.next())` por um `while(rs.next());`;
- Assim, enquanto houver um próximo registro retornado pelo ResultSet você irá percorre-lo;

Pacote de Persistência - Classe EnderecoDAO

- O próximo método é **update()**;
- No construtor, vamos criar o PreparedStatement para ele;

```
private EnderecoDAO() throws ClassNotFoundException, SQLException, SelectException {  
    Connection conexao = Conexao.getConexao();  
    selectNewId = conexao.prepareStatement("select nextval('id_endereco')");  
    insert = conexao.prepareStatement("insert into endereco values (?, ?, ?, ?, ?)");  
    select = conexao.prepareStatement("select * from endereco where id_pessoa = ?");  
    update = conexao.prepareStatement("update endereco set rua = ?, numero = ?,  
    cidade = ? where id_pessoa = ?");  
}
```

Pacote de Persistência - Classe EnderecoDAO

- O método em si segue a mesma lógica do insert, com as seguintes diferenças:
- A exceção lançada é uma `UpdateException`;
- O id vem do objeto `Endereço` ao invés da chamada do método **`getNewId()`**;

```
public void update(Endereco endereco) throws UpdateException {  
    try {  
        update.setString(1, endereco.getRua());  
        update.setInt(2, endereco.getNumero());  
        update.setString(3, endereco.getCidade());  
        update.setInt(4, endereco.getIdPessoa());  
        update.executeUpdate();  
    } catch (SQLException e) {  
        throw new UpdateException("Erro ao atualizar rua");  
    }  
}
```

Pacote de Persistência - Classe EnderecoDAO

- O método **delete()** também segue a mesma lógica;
- No construtor:

```
private EnderecoDAO() throws ClassNotFoundException, SQLException, SelectException {  
    Connection conexao = Conexao.getConexao();  
    selectNewId = conexao.prepareStatement("select nextval('id_endereco')");  
    insert = conexao.prepareStatement("insert into endereco values (?, ?, ?, ?, ?)");  
    select = conexao.prepareStatement("select * from endereco where id_pessoa = ?");  
    update = conexao.prepareStatement("update endereco set rua = ?, numero = ?,  
    cidade = ? where id_pessoa = ?");  
    delete = conexao.prepareStatement("delete from endereco where id_pessoa = ?");  
}
```

Pacote de Persistência - Classe EnderecoDAO

- Ele lança a exceção DeleteException;
- O id da Pessoa é obtido do objeto Endereco passado como parâmetro.

```
public void delete(Endereco endereco) throws DeleteException {  
    try {  
        delete.setInt(1, endereco.getIdPessoa());  
        delete.executeUpdate();  
    } catch (SQLException e) {  
        throw new DeleteException("Erro ao deletar endereco");  
    }  
}
```

Pacote de Persistência - Classe PessoaDAO

- Agora que terminamos de criar os métodos referentes ao DAO de objetos do tipo Endereço, vamos fazer o mesmo para objetos do tipo Pessoa;
- Vamos começar a classe PessoaDAO já definindo todos os PreparedStatements, preparando o Singleton com o atributo instance e deixando uma instância do endereçoDAO pronta como atributo;

```
public class PessoaDAO {  
  
    private static PessoaDAO instance = null;  
    private static EnderecoDAO enderecoDAO = null;  
  
    private PreparedStatement selectNewId;  
    private PreparedStatement insert;  
    private PreparedStatement delete;  
    private PreparedStatement selectAll;  
    private PreparedStatement update;  
}
```


Pacote de Persistência - Classe PessoaDAO

- Como é um Singleton, vamos tornar o construtor privado e preparar todos os SQL's nos PreparedStatements;
- O processo é exatamente o mesmo que na classe EnderecoDAO;

```
private PessoaDAO() throws ClassNotFoundException, SQLException, SelectException {  
    Connection conexao = Conexao.getConexao();  
  
    selectNewId = conexao.prepareStatement("select nextval('id_pessoa')");  
    insert = conexao.prepareStatement("insert into pessoa values (?, ?, ?, ?)");  
    delete = conexao.prepareStatement("delete from pessoa where id = ?");  
    selectAll = conexao.prepareStatement("select * from pessoa");  
    update = conexao.prepareStatement("update pessoa set nome = ?, cpf = ?, telefone =  
    ? where id = ?");  
  
    enderecoDAO = EnderecoDAO.getInstance();  
}
```

- Faremos o método **getInstance()** exatamente igual a todo Singleton:

```
public static PessoaDAO getInstance() throws ClassNotFoundException, SQLException,
SelectException {
    if (instance == null) {
        instance = new PessoaDAO();
    }
    return instance;
}
```

Pacote de Persistência - Classe PessoaDAO

- O método **selectNewId()** exatamente igual ao da classe EnderecoDAO:

```
public int selectNewId() throws SelectException {  
    try {  
        ResultSet rs = selectNewId.executeQuery();  
        if (rs.next()) {  
            return rs.getInt(1);  
        }  
    } catch (SQLException e) {  
        throw new SelectException("Erro ao buscar novo id da tabela pessoa");  
    }  
    return 0;  
}
```

Pacote de Persistência - Classe PessoaDAO

- O método **insert()** também com a mesma lógica do insert() da classe EnderecoDAO, apenas adaptado aos atributos da classe Pessoa;
- Apenas note que é necessário fazer um **setIdPessoa()** no endereço antes de usar o método **insert()** do EnderecoDAO.

```
public void insert(Pessoa pessoa) throws InsertException, SelectException {  
    try {  
        pessoa.setId(selectNewId());  
        insert.setInt(1, pessoa.getId());  
        insert.setString(2, pessoa.getNome());  
        insert.setInt(3, pessoa.getCpf());  
        insert.setInt(4, pessoa.getTelefone());  
        insert.executeUpdate();  
        pessoa.getEndereco().setIdPessoa(pessoa.getId());  
        enderecoDAO.insert(pessoa.getEndereco());  
    } catch (SQLException e) {  
        throw new InsertException("Erro ao inserir pessoa");  
    }  
}
```

Pacote de Persistência - Classe PessoaDAO

- O delete, mesmo processo;
- Note que o endereço é deletado também, usando o método **delete()** da classe EnderecoDAO:

```
public void delete(Pessoa p) throws DeleteException {  
    enderecoDAO.delete(p.getEndereco());  
    try {  
        delete.setInt(1, p.getId());  
        delete.executeUpdate();  
    } catch (SQLException e) {  
        throw new DeleteException("Erro ao deletar pessoa");  
    }  
}
```

- E o update também:

```
public void update(Pessoa pessoa) throws UpdateException {  
    try {  
        enderecoDAO.update(pessoa.getEndereco());  
        update.setString(1, pessoa.getNome());  
        update.setInt(2, pessoa.getCpf());  
        update.setInt(3, pessoa.getTelefone());  
        update.setInt(4, pessoa.getId());  
        update.executeUpdate();  
    } catch (SQLException e) {  
        throw new UpdateException("Erro ao atualizar pessoa");  
    }  
}
```

Pacote de Persistência - Classe PessoaDAO

- O único método que difere dos outros é o **selectAll()**;
- Nesse método vamos retornar uma lista de objetos, e não mais um único objeto.
- Do ponto de vista da orientação a objetos, seria prático criar um método **select()** que recebe um id e retorna um objeto;
- Assim o método **selectAll()** usaria esse método para retornar todos os ids do banco;
- Mas como não estamos interessados em buscar objetos soltos, apenas a lista inteira, não tem necessidade de se criar esse método no momento.
- Para retornarmos uma lista, primeiro precisamos criar uma lista de objetos e executar a sql:

Pacote de Persistência - Classe PessoaDAO

```
public List<Pessoa> selectAll() throws SelectException {  
    List<Pessoa> pessoas = new LinkedList<Pessoa>();  
    try {  
        ResultSet rs = selectAll.executeQuery();  
        ...  
    }  
}
```

- Agora o que precisamos fazer é: enquanto tiver mais um objeto no ResultSet, instanciamos um objeto Pessoa,
- Buscamos seu endereço correspondente no banco;
- Adicionamos o objeto Pessoa a lista;
- E retornamos a lista:

Pacote de Persistência - Classe PessoaDAO

```
public List<Pessoa> selectAll() throws SelectException {
    List<Pessoa> pessoas = new LinkedList<Pessoa>();
    try {
        ResultSet rs = selectAll.executeQuery();
        while (rs.next()) {
            int id = rs.getInt(1);
            String nome = rs.getString(2);
            int cpf = rs.getInt(3);
            int telefone = rs.getInt(4);

            Endereco endereco = enderecoDAO.select(rs.getInt(1));

            pessoas.add(new Pessoa(id, nome, cpf, telefone, endereco));
        }
    } catch (SQLException e) {
        throw new SelectException("Erro ao buscar pessoa");
    }
    return pessoas;
}
```

- Caso ocorra uma exceção, lançaremos uma SelectException, assim como foi feito na classe EnderecoDAO.

Pacote de Negócio - Classe Sistema

- Agora que terminamos a camada de persistência, vamos fazer a de negócio;
- Na classe Sistema, precisamos que o construtor receba a senha do banco de dados;
- Com essa senha, iremos repassa-la para a classe Conexão;
- Como ela é um Singleton, por toda a vida daquela classe, a senha estará salva e o acesso ao banco, acessível a todas as classes da aplicação.

```
public class Sistema {  
  
    private PessoaDAO pessoaDAO;  
  
    public Sistema(String senha) throws ClassNotFoundException, SQLException, SelectException {  
  
        Conexao.setSenha(senha);  
        pessoaDAO = PessoaDAO.getInstance();  
  
    }  
}
```

Pacote de Negócio - Classe Sistema

- Os demais métodos apenas chamam os métodos da classe PessoaDAO:

```
public void inserirPessoa(Pessoa p) throws InsertException, SelectException {  
    pessoaDAO.insert(p);  
}  
  
public List<Pessoa> selectAll() throws SelectException {  
    return pessoaDAO.selectAll();  
}  
  
public void deletePessoa(Pessoa p) throws DeleteException {  
    pessoaDAO.delete(p);  
}  
  
public void atualizarPessoa(Pessoa p) throws UpdateException {  
    pessoaDAO.update(p);  
}
```

Pacote de Apresentação

- O pacote de apresentação não será detalhado aqui pois não é foco da aula criar uma interface console;
- O código do projeto se encontra nesse [link](#)

Seções

Exemplo

Resolução

Instalando Driver JDBC


Aplicação

Exercício

Exercício

- A partir do projeto da aula anterior: Aula Prática 9 - Tratamento de Exceções, transforme a camada de persistência em arquivos da lista de contatos para persistência em banco de dados;
- Utilize o script a seguir para criar o banco de dados:

```
create sequence id_contato;  
  
create table contato(  
    id int,  
    nome varchar(50),  
    telefone int,  
    primary key (id)  
);
```

 KUWAKI, V. T. F. Modelo de slides udesc lattex. In: . [S.l.]: Disponível em: <<https://github.com/takeofriedrich/slidesUdescLattex>>. Acesso em: 24 jan. 2020.

Duvidas:
Vinicius Takeo Friedrich Kuwaki
vinicius.kuwaki@edu.udesc.br
github.com/takeofriedrich



UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA