MONITORAMENTO DE SISTEMA COM PRINCÍPIOS SOLID

SPTECH
2023

Bruno Henrique de Almeida Lima

OBJETIVO

Desenvolver um aplicativo Java para monitoramento de recursos do sistema, aderindo aos princípios SOLID e utilizando a biblioteca Looca (API que coleta dados de maquina desenvolvida para facilitar o aprendizado em contextos acadêmicos, visando ajudar os alunos da Faculdade SPTech em projetos na disciplina de Pesquisa e Inovação.

REQUISITOS

1. BIBLIOTECA LOOCA

Looca é uma API que coleta dados de maquina e foi desenvolvida para facilitar o aprendizado em contextos acadêmicos. Esse projeto visa ajudar os alunos da Faculdade SPTech em projetos na disciplina de Pesquisa e Inovação.

Dentro do projeto, ela foi usada para capturar a porcentagem de uso dos componentes processador, memória e disco da máquina.

```java
@Override
public Double porcentagemUso() {
    Looca looca = new Looca();
    com.github.britooo.looca.api.group.processador.Processador processador = looca.getProcessador();
    return processador.getUso();
}
```

Método Abstrato "porcentagemUso()" da Classe "Processador"

```java
@Override
public Double porcentagemUso() {
    Looca looca = new Looca();
    com.github.britooo.looca.api.group.memoria.Memoria memoria = looca.getMemoria();
    DecimalFormat decimalFormat = new DecimalFormat( pattern: "#,##0.00");
    decimalFormat.setRoundingMode(RoundingMode.DOWN);
    long porcentagem = (memoria.getEmUso() * 100) / memoria.getTotal();
    return Double.parseDouble(decimalFormat.format(porcentagem));
}
```

Método Abstrato "porcentagemUso()" da Classe "Memoria"

```java
@Override
public Double porcentagemUso() {
    Looca looca = new Looca();

    DiscoGrupo grupoDeDiscos = new DiscoGrupo();
    List<Volume> volumes = grupoDeDiscos.getVolumes();

    Double usoDisco = 0.0;
    Double volumeTotalDiscos = 0.0;
    Double volumeDisponivelDiscos = 0.0;

    for (Volume volume : volumes) {
        volumeTotalDiscos += volume.getTotal();
        volumeDisponivelDiscos += volume.getDisponivel();
    }

    Double usoDiscos = (Double) (((volumeTotalDiscos - volumeDisponivelDiscos) * 100) / volumeTotalDiscos);

    return usoDiscos;
}
```

Método Abstrato "porcentagemUso()" da Classe "Disco"

2. ABSTRAÇÕES E INTERFACES

Como apresentado anteriormente, pode ser visto que todos possuem o mesmo nome no método de capturar o uso do componente em porcentagem, e a explicação para isto é bem simples, as classes dos componentes processador, memoria e disco herdam este método de uma classe abstrata chamada "Componente".

```java
public abstract class Componente {
    4 usages
    private Integer idComponente;
    4 usages
    private String nome;
    4 usages
    private String unidadeMedida;

    3 usages   brunolimabh *
    public Componente(Integer idComponente, String nome, String unidadeMedida) {
        this.idComponente = idComponente;
        this.nome = nome;
        this.unidadeMedida = unidadeMedida;
    }

    6 usages   3 implementations   brunolimabh
    public abstract Double porcentagemUso();
```

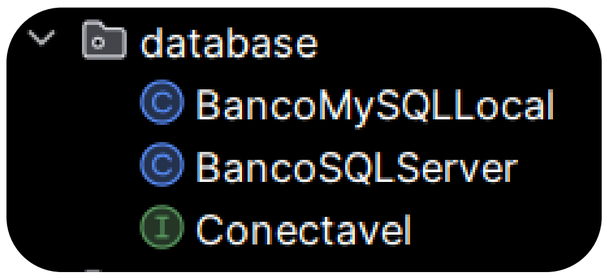
Classe "Componente"

Falando de Interface, foi utilizado para uma classe chamada “Conectavel”, na qual faz a ponte de conexão com os bancos de dados propostos no projeto sendo eles MySQL e SQL SERVER.

```java
public interface Conectavel {
    6 usages  3 implementations  brunolimabh
    JdbcTemplate Conexao();

    6 usages  3 implementations  brunolimabh
    void insertRegistro(Double valor, LocalDateTime dataHora, Integer fkComponente, Integer fkTotem);
}
```

Interface Conectavel



Package “databse”

Falando de Interface, foi utilizado para uma classe chamada “Conectavel”, na qual faz a ponte de conexão com os bancos de dados propostos no projeto sendo eles MySQL e SQL SERVER.

```java
@Override
public JdbcTemplate Conexao() {
    BasicDataSource dataSource = new BasicDataSource();
    dataSource.setUrl("jdbc:mysql://localhost:3306/ConWay");
    dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
    dataSource.setUsername("root");
    dataSource.setPassword("xpto");
    this.conexao = new JdbcTemplate(dataSource);
    return conexao;
}
```

Conexão MySQL local (backup)

```java
@Override
public JdbcTemplate Conexao() {
    BasicDataSource dataSource = new BasicDataSource();
    dataSource.setUrl("jdbc:sqlserver://18.232.10.255:1433;databaseName=ConWay;encrypt=false;");
    dataSource.setDriverClassName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
    dataSource.setUsername("sa");
    dataSource.setPassword("xpto");
    this.conexao = new JdbcTemplate(dataSource);
    return conexao;
}
```

Conexão SQL SERVER na AWS

3. DAO (DATA ACCESS OBJECT)

O padrão DAO é um padrão de projeto de software que visa separar a lógica de acesso a dados do restante da aplicação. Ele fornece uma interface para acessar dados armazenados em um banco de dados ou outro mecanismo de persistência. O objetivo é isolar o código que trata do armazenamento e recuperação de dados, facilitando a manutenção e modificação do código.
Conforme apresentado anteriormente e no decorrer deste documento, ele foi empregado em toda a estrutura do projeto, visando a possibilidade de alterações futuras.

4. CAPTURA DE DADOS COM TIMERTASK

Através de uma classe chamada “InserirRegistroTask”, é feito a captura do uso dos componentes que o cliente deseja monitorar em porcentagem. Esta escolha é feita no cadastro do totem dentro da WebApp ConWay (projeto de Pesquisa e Inovação). Após a captura, os dados são inseridos tanto no banco de dados MySQL (máquina do cliente) quanto no banco SQL SERVER (servidor do projeto).

```
Timer agendador = new Timer();
InserirRegistroTask task = new InserirRegistroTask( delay: 1000, periodo: 1000,totemAtual.getIdTotem(),totemComponentes);
agendador.schedule(task, task.getDelay(), task.getPeriodo());

inputString.nextLine();
agendador.cancel();

System.out.println("Monitoramento encerrado!");
```

Chamada do TimerTask “InserirRegistroTask” dentro da classe “Main”

Dentro da classe “Main”, é criado uma classe “Timer” para controlarmos de uma forma prática a Task criada, na qual é passado argumentos como delay (tempo para iniciar), período (intervalo de repetição), o identificador único daquele totem e a lista de componentes que será monitorado para aquele totem.

5. TABELA DE PARAMETRIZAÇÃO

Como dito acima, é coletado através do banco de dados as informações sobre qual componente monitorar a partir de um identificar único de um totem. Com isso, a consulta ao banco é feita através do comando “SELECT” em uma tabela chamada “TotemComponente”, na qual armazena o nome, o identificador único e o valor (desconsiderar) do totem.

fkComponente	fkTotem	valor
1	1	NULL
2	1	NULL
3	1	NULL

*“SELECT * FROM TotemComponente WHERE fkTotem = 1;”*

No exemplo, o “fkComponente” 1 seria o processador, 2 a memória e 3 o disco.

6. USO DE MAPPER

A utilização da interface RowMapper desempenha um papel crucial na eficiência e organização das consultas ao banco de dados. O RowMapper atua como um mapeador entre as linhas retornadas de uma consulta SQL e os objetos Java correspondentes, facilitando a conversão de dados entre o banco de dados relacional e a aplicação.

Para cada tipo de entidade que interage com o banco de dados, implementamos a interface RowMapper. No contexto atual, criamos três RowMappers específicos:

TotemRowMapper (responsável por mapear os resultados das consultas relacionadas aos totens, “FuncionarioRowMapper” (utilizado para mapear os dados associados aos funcionários) e “ComponentesRowMapper” (essencial para identificar e mapear os componentes que um totem específico está configurado para monitorar).

```java
public Totem mapRow(ResultSet resultSet, int i) throws SQLException {
    Totem totem = new Totem();
    totem.setIdTotem(resultSet.getInt( columnLabel: "idTotem"));
    totem.setNome(resultSet.getString( columnLabel: "nome"));
    totem.setFkAeroporto(resultSet.getInt( columnLabel: "fkAeroporto"));
    totem.setFkEmpresa(resultSet.getInt( columnLabel: "fkEmpresa"));
    return totem;
}
```

TotemRowMapper

```java
public Funcionario mapRow(ResultSet resultSet, int i) throws SQLException {
    Funcionario funcionario = new Funcionario();
    funcionario.setIdFuncionario(resultSet.getInt( columnLabel: "idFuncionario"));
    funcionario.setEmail(resultSet.getString( columnLabel: "email"));
    funcionario.setSenha(resultSet.getString( columnLabel: "senha"));
    funcionario.setNome(resultSet.getString( columnLabel: "nome"));
    funcionario.setFkEmpresa(resultSet.getInt( columnLabel: "fkEmpresa"));
    return funcionario;
}
```

FuncionarioRowMapper

```java
public TotemComponente mapRow(ResultSet resultSet, int i) throws SQLException {
    TotemComponente totemComponente = new TotemComponente();
    totemComponente.setFkComponente(resultSet.getInt( columnLabel: "fkComponente"));
    totemComponente.setFkTotem(resultSet.getInt( columnLabel: "fkTotem"));
    totemComponente.setValor(resultSet.getInt( columnLabel: "valor"));
    return totemComponente;
}
```

TotemComponenteRowMapper

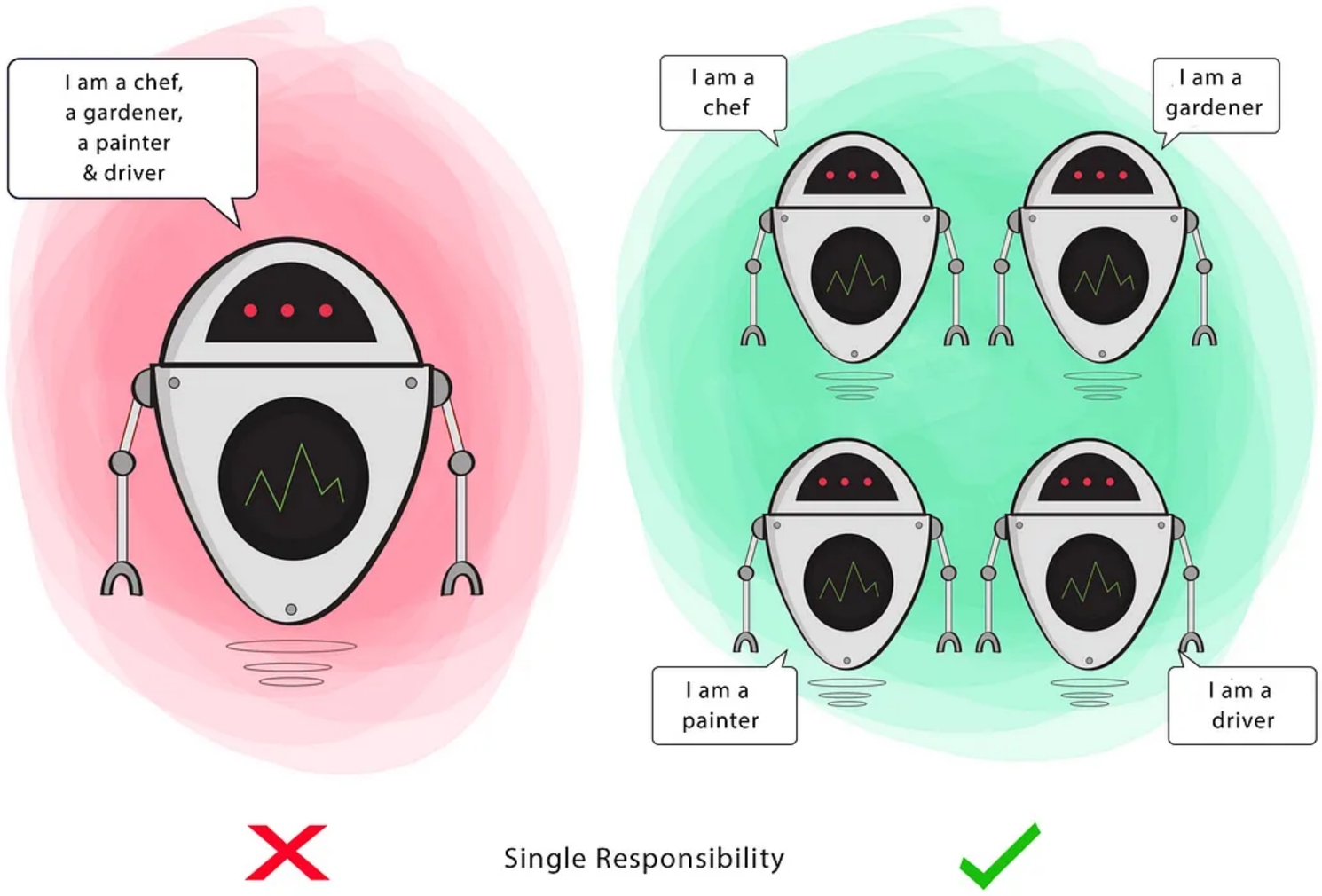
7. NOMENCLATURA

A escolha de nomes apropriados para variáveis, parâmetros, métodos e classes é crucial para a clareza, legibilidade e manutenibilidade do código Java. A seguir, será apresentado algumas boas práticas em padrões de nomenclatura Java usados no projeto:

- camelCase:
  - Variáveis e Métodos: Utilize o CamelCase para nomes de variáveis e métodos. Comece com uma letra minúscula e capitalize as primeiras letras de cada palavra subsequente. Exemplo: idComponente , porcentagemUso().
- PascalCase:
  - Parecido com o CamelCase, mas comece com uma letra maiúscula. Usado para nome de classes. Exemplo: TotemComponente, Processador.
- pacotes:
  - É recomendado utilizar apenas uma palavra como nome do pacote. Exemplo: dao e database. O pacote não pode ter o nome de uma palavra-chave do java e não pode começar com número ou caracteres especiais.
- MACRO_CASEs:
  - Todas as letras em caixa alta e cada palavra é separada por um underline, aplicado em constantes e enum. Exemplo: MEMORIA e VALOR_MÁXIMO.
- Evitar Abreviações Obscuras:
  - Evite abreviações obscuras que possam tornar o código difícil de entender. Prefira nomes descritivos mesmo que sejam mais longos. Exemplo: idComp.

8. PRINCÍPIO 'S' DO SOLID

Conhecido como o Responsabilidade Única (Single Responsibility), enfatiza a importância de que uma classe tenha uma única razão para mudar, ou seja, uma única responsabilidade. Ao aplicar este princípio, cada classe deve ser projetada para realizar apenas uma tarefa específica e não deve ser sobrecarregada com responsabilidades diversas. Segue uma ilustração

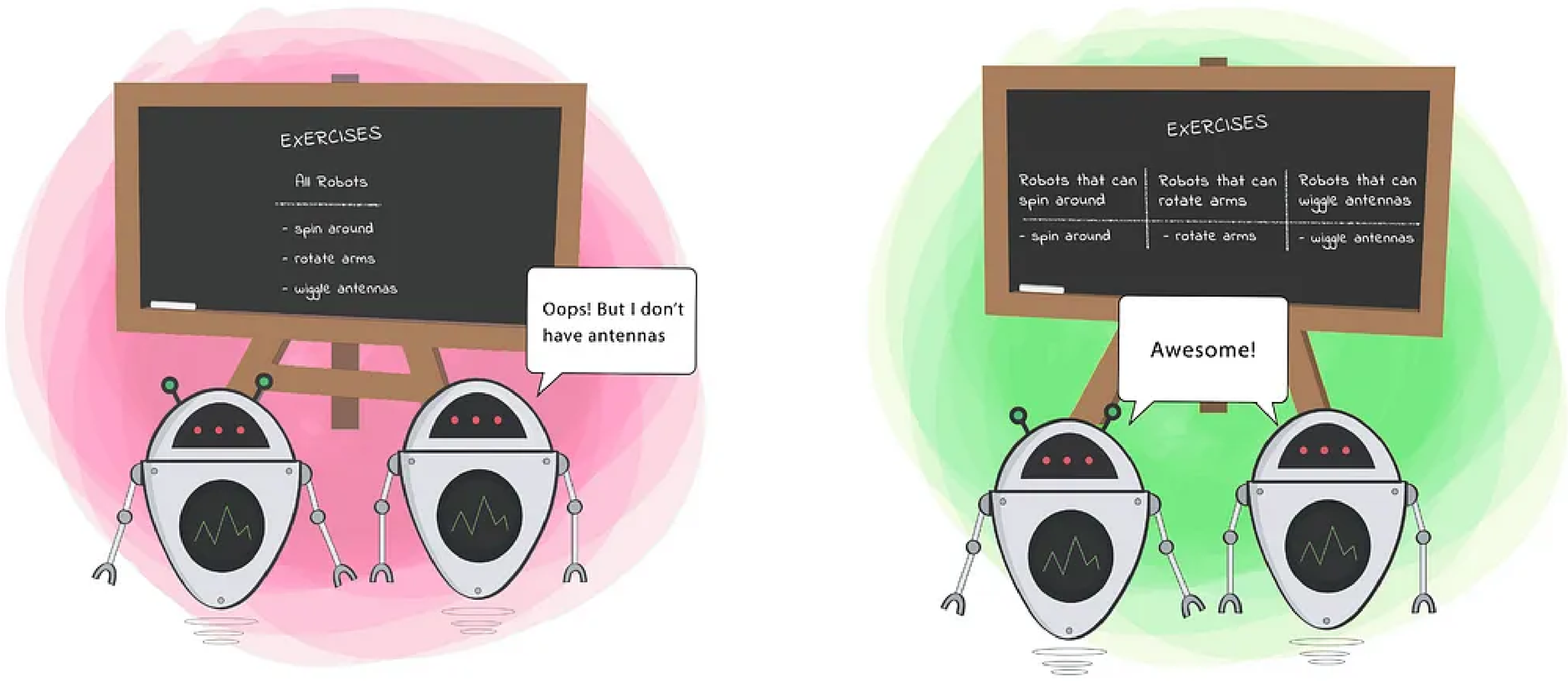


https://medium.com/backticks-tildes/the-s-o-l-i-d-principles-in-pictures-b34ce2f1e898

Este princípio visa separar comportamentos para que, se surgirem bugs como resultado de sua mudança, isso não afete outros comportamentos não relacionados. No âmbito do meu projeto Java e como apresentado durante este documento, o princípio da Single Responsibility foi cuidadosamente implementado para garantir a coesão e a manutenibilidade do código.

9. PRINCÍPIO 'I' DO SOLID

Conhecido como o Segregação de Interfaces (Interface Segregation), destaca a importância de não forçar classes a dependerem de interfaces que elas não utilizam. Em outras palavras, uma classe não deve ser obrigada a implementar interfaces que contêm métodos que ela não precisa. Segue uma ilustração:



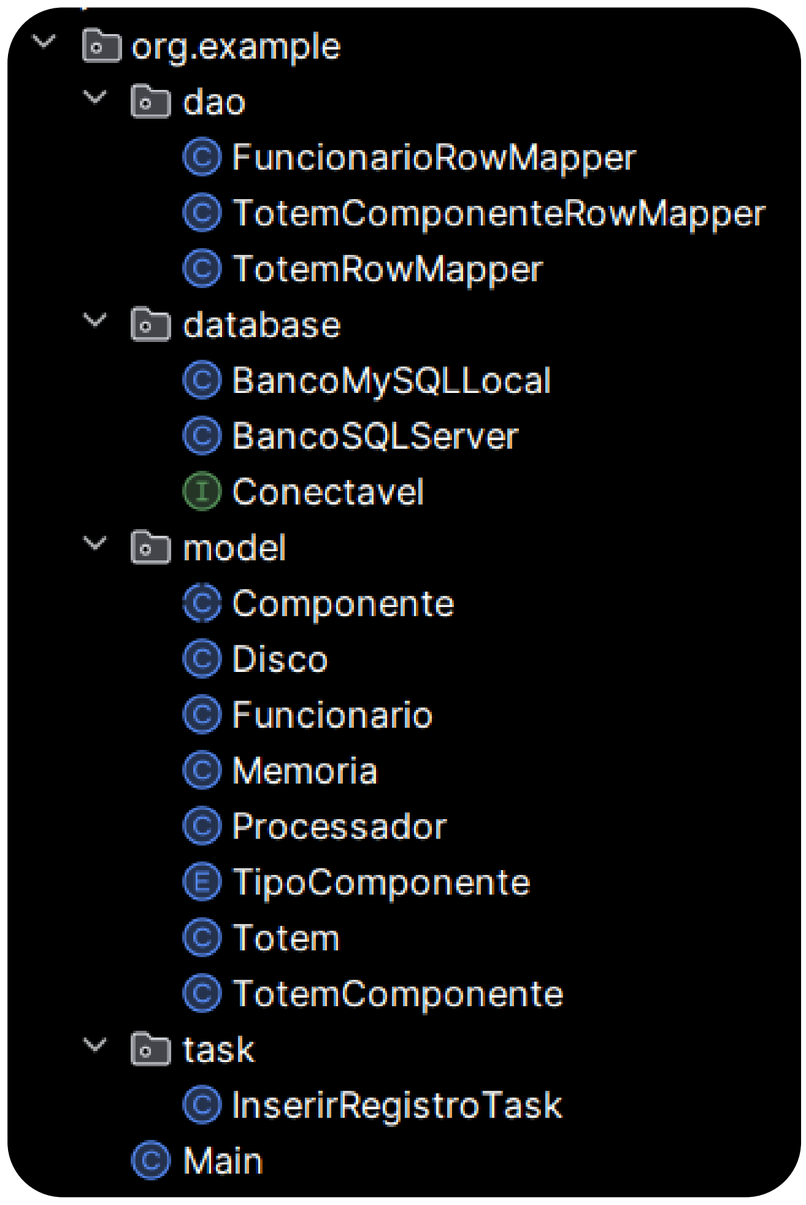
https://medium.com/backticks-tildes/the-s-o-l-i-d-principles-in-pictures-b34ce2f1e898

O princípio foi aplicado de maneira consistente no projeto, resultando em um sistema com interfaces bem definidas e dependências mínimas entre classes. Essa abordagem promove a flexibilidade, a manutenibilidade e a extensibilidade do código ao evitar acoplamentos desnecessários.

10. ORGANIZAÇÃO DE PACOTES

O projeto foi estruturado seguindo boas práticas de organização de código, visando a clareza, manutenibilidade e escalabilidade do sistema. A estrutura de pacotes foi pensada de forma a refletir a arquitetura do projeto e agrupar classes relacionadas de maneira lógica. A seguir, detalhamos a intenção de cada pacote:

- dao: Agrupa as classes relacionadas ao padrão DAO (Data Access Object). Aqui, encontram-se os mapeadores (RowMapper) que facilitam a conversão entre os resultados das consultas e objetos Java.
- database: Contém as classes responsáveis pela interação com o banco de dados.
- model: Abriga as entidades principais do domínio do sistema.
- task: Engloba as classes relacionadas a tarefas temporizadas ou agendadas.



packeges gerais

11. UTILIZAÇÃO DE ENUMERADORES

Os enumeradores, ou Enums, são uma forma de representar um conjunto fixo de valores constantes. Eles oferecem uma maneira elegante de definir e utilizar valores padrões em um projeto. No contexto deste projeto, um enumerador foi criado para representar os diferentes tipos de componentes do sistema, onde cada valor constante do enumerador (PROCESSADOR, MEMORIA, DISCO) é declarado com um identificador e uma descrição.

```
public enum TipoComponente {
    1 usage
    PROCESSADOR( fkComponente: 1, descricao: "cpu"),
    1 usage
    MEMORIA( fkComponente: 2, descricao: "memoria"),
    1 usage
    DISCO( fkComponente: 3, descricao: "disco");
    3 usages
    private int fkComponente;
    3 usages
    private String descricao;
```

Atributos do enum “TipoComponente”

Este enumerador tem a finalidade de possibilitar a identificação do tipo de componente que a task “InserirRegistroTask” deve monitorar, associando-o ao seu identificador único no banco de dados. Essa abordagem facilita o monitoramento e a manipulação de diferentes tipos de componentes dentro da tarefa, tornando o código mais semântico e compreensível.

FUNCIONAMENTO

1 - Rode o script mysql em sua máquina local.
2 - Rode o arquivo .jar com as suas dependências.
3 - Insira um login válido.
4 - Escolha um totem já cadastrado pelo site.
5 - Informe o tipo de monitoramento.
6 - Aperte qualquer tecla para sair.

```
0 - Sair
1 - Acessar Sistema
1
Insira seu email de acesso:
pedro.henrique@latam.com
Insira sua senha:
12345
```

```
Bem-vindo ao sistema de monitoramento da AIRWAY!
Informe o nome destá máquina (cadastrado no nosso site)
0 - Voltar
2 - TLT-2
3 - TLT-3
4 - TLT-4
```

```
21
Deseja começar o monitoramento?
0 - Não, voltar!
1 - Não, redefinir métricas! (Em Breve)
2 - Sim, entre 5-20 segundos!
```

```
------------------------
Data: 07/12/2023 16:46:21
CPU: 3,50%
Memoria: 88,00%
Disco: 68,65%
------------------------
```