Projeto da Disciplina de Desenvolvimento Seguro

PUCPR

Equipe:

Carlos Leonardo Pires

Michello Viana de Almeida

Ronald Martins Fagundes

Tiago Saddi Domingues

**Brasília**

**2024**

**Sumário**

[1 Introdução 3](#_Toc181537472)

[2 Documento de projeto 3](#_Toc181537473)

[3 Regras do projeto 3](#_Toc181537474)

[3.1 Critérios de avaliação: 4](#_Toc181537475)

[4 Descrição do Sistema de Gerenciamento de Pedidos 5](#_Toc181537476)

[4.1 Objetivo do Projeto: 5](#_Toc181537477)

[4.2 Descrição do Sistema: 5](#_Toc181537478)

[4.3 Requisitos Funcionais: 5](#_Toc181537479)

[4.4 Design Patterns Utilizados: 6](#_Toc181537480)

[5 Repositório da Solução 6](#_Toc181537481)

[6 Banco de Dados 6](#_Toc181537482)

[7 Justificativa dos padrões de projetos escolhidos 7](#_Toc181537483)

[7.1 Singleton 7](#_Toc181537484)

[7.2 Strategy 9](#_Toc181537485)

[7.3 Factory Method 12](#_Toc181537486)

[7.4 Observer 14](#_Toc181537487)

[7.5 Decorator 19](#_Toc181537488)

[7.6 Data Access Object 22](#_Toc181537489)

[8 Aplicação das Regras do Projeto do CEI CERT 26](#_Toc181537490)

[8.1 Validação e Sanitização de Entrada 26](#_Toc181537491)

[8.1.1 IDS00-J. Prevent SQL injection (e XSS, caso web) 26](#_Toc181537492)

[8.1.2 IDS01-J. Normalize strings before validating them 27](#_Toc181537493)

[8.1.3 IDS03-J. Do not log unsanitized user input 28](#_Toc181537494)

[8.2 Declarações 29](#_Toc181537495)

[8.2.1 DCL01-J. Do not reuse public identifiers from the Java Standard Library 29](#_Toc181537496)

[8.2.2 DCL02-J. Do not modify the collection's elements during na enhanced for statement 29](#_Toc181537497)

[8.3 Comportamento Excepcional 29](#_Toc181537498)

[8.3.1 ERR00-J. Do not suppress or ignore checked exceptions 29](#_Toc181537499)

[8.3.2 ERR07-J. Do not throw RuntimeException, Exception, or Throwable 30](#_Toc181537500)

[8.4 Diversos: 30](#_Toc181537501)

[8.4.1 MSC01-J. Do not use an empty infinite loop 30](#_Toc181537502)

[8.4.2 MSC04-J. Do not leak memory 32](#_Toc181537503)

[8.4.3 MSC02-J. Generate strong random number 32](#_Toc181537504)

[8.4.4 MSC03-J. Never hard code sensitive information 33](#_Toc181537505)

[8.4.5 MSC07-J. Prevent multiple instantiations of singleton objects 33](#_Toc181537506)

# Introdução

A codificação segura é a prática de desenvolver software de computador de forma a evitar a introdução acidental de vulnerabilidades de segurança. Defeitos, bugs, falhas lógicas e falta de padronização são consistentemente a principal causa das vulnerabilidades de software comumente exploradas [CWE, 2021]

O objetivo desse projeto e a aplicação dessa prática de desenvolvimento de software. Devesse observar que existem diversas comunidades que oferecem suporte à codificação segura com padrões e regras para diversas linguagens de programação, como SEI CERT, OWASP, CWE, entre outros.

Para o projeto, optou-se pela aplicação dos [padrões e regra do SEI CERT](https://wiki.sei.cmu.edu/confluence/display/java/SEI+CERT+Oracle+Coding+Standard+for+Java).

O SEI CERT Coding Standards é um conjunto de práticas de programação que visam aumentar a segurança e a robustez de códigos em várias linguagens de programação, incluindo Java. Essas diretrizes são publicadas pelo Software Engineering Institute (SEI) da Carnegie Mellon University.

# Documento de projeto

Este documento descreve o Sistema de Gerenciamento de Pedidos desenvolvido em Java que utiliza vários Design Patterns para resolver problemas comuns de design de software e aplica, para essa disciplina, boas práticas para desenvolvimento seguro de software.

Este projeto integra padrões de projeto criacionais, estruturais e comportamentais para criar uma aplicação robusta, modular e fácil de manter, além dos requisitos obrigatórios.

# Regras do projeto

O projeto aplica as seguintes regras:

* [Validação e Sanitização de Entrada](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487865) :
  + **IDS00-J. Prevent SQL injection (e XSS, caso web)**
  + **IDS01-J. Normalize strings before validating them**
  + **IDS03-J. Do not log unsanitized user input**
* [Declarações](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487858):
  + **DCL01-J. Do not reuse public identifiers from the Java Standard Library**
  + **DCL02-J. Do not modify the collection's elements during an enhanced for statement**
* [Comportamento Excepcional](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487704):
  + **ERR00-J. Do not suppress or ignore checked exceptions**
  + **ERR07-J. Do not throw RuntimeException, Exception, or Throwable**
* [Diversos](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487686):
  + **MSC01-J. Do not use an empty infinite loop**
  + **MSC04-J. Do not leak memory**
  + **MSC02-J. Generate strong random number**
  + **MSC03-J. Never hard code sensitive information**
  + **MSC07-J. Prevent multiple instantiations of singleton objects**
* **Deverá desenvolver um projeto em Java (web ou console)**
* Deverá cadastrar as tarefas no Trello.
* Deverá possuir pelo menos 6 funcionalidades diferentes neste projeto, com persistência em banco de dados relacional.
* **Deverá desenvolver o projeto com MVC, DAO e Singleton para a classe de conexão ao banco de dados relacional**.
* Deverá considerar o Cognito para cadastro e autenticação dos usuários do projeto em um ou mais grupos.

## Critérios de avaliação:

* Qualidade da Solução: Os projetos serão avaliados com base na clareza do plano, na relevância e aplicabilidade das boas práticas escolhidas, e na qualidade da justificativa para cada escolha.
* Criatividade: Inovação na escolha do tema, na abordagem do problema e na solução proposta.
* O projeto deve ter pelo menos 6 funcionalidades, com aplicação de x boas práticas.

# Descrição do Sistema de Gerenciamento de Pedidos

## Objetivo do Projeto:

Desenvolver um sistema de gerenciamento de pedidos em Java com cinco funcionalidade aplicando-se boas práticas de sistema seguro. Este projeto integrará padrões criacionais, estruturais e comportamentais para criar uma aplicação robusta, modular e fácil de manter.

## Descrição do Sistema:

O sistema permitirá o cadastro de clientes, a realização de pedidos, o processamento desses pedidos e o gerenciamento do envio.

A aplicação deverá também oferecer diferentes métodos de cálculo para frete e diferentes formas de notificação para atualizações de pedido.

## Requisitos Funcionais:

O sistema tem os seguintes requisitos funcionais:

* **Cadastro de Clientes**: Adicionar, remover e editar informações de clientes.
* **Realização de Pedidos**: Permitir que clientes realizem pedidos, escolhendo produtos e quantidades.
* **Processamento de Pedidos**: Incluir etapas de verificação de estoque, cálculo de frete e finalização do pedido.
* **Gerenciamento de Envio**: Programar e acompanhar o envio de pedidos.
* **Notificações**: Enviar atualizações sobre o status do pedido ao cliente via email ou SMS.

## Design Patterns Utilizados:

A seguir uma breve explicação dos padrões de projetos aplicados:

* + **Singleton (padrão criacional) (ok):** Utilizado para gerenciar a configuração global do sistema, como informações de conexão com o banco de dados.
  + **Strategy** **(padrão comportamental) (ok)**: permite diferentes estratégias de cálculo de frete que podem variar de acordo com o destino e o método de entrega.
  + **Factory Method (Padrão Criacional)(ok):** Para criar diferentes possibilidade de login dependendo da parametrização da solução.
  + **Observer (publish-subscribe)** notifica os clientes que desejam saber de promoções do sítio, utilizando a interface do sistema; **(https://www.youtube.com/watch?v=ioYkXh8NhKc)**
  + **Decorator** adiciona responsabilidades adicionais a um objeto de maneira dinâmica. Neste projeto esse padrão será utilizado para adicionar funcionalidades adicionais aos pedidos, como embalagens para presente ou garantias estendidas.
  + **Data Transfer Object (DTO) (padrão estrutural)** é um padrão utilizado para transferir dados complexos entre diferentes partes de um sistema de forma eficiente.

# Repositório da Solução

O projeto está no seguinte repositório do Github:

<https://github.com/tiagosaddidomingues/desingpatterns/>

# Banco de Dados

Para o projeto utilizamos o MySQL, sistema de gerenciamento de banco de dados relacional (SGBDR) amplamente utilizado, desenvolvido pela Oracle Corporation.

Pontos importantes para escolha do MySQL:

* Código aberto: O MySQL é um software de código aberto;
* Popularidade: É um dos SGBDs mais utilizados no mundo;
* Portabilidade: Suporta diversas plataformas;
* Desempenho: Oferece excelente desempenho e estabilidade;
* Facilidade de uso: Possui uma sintaxe simples e várias ferramentas gráficas para a administração.

# Justificativa dos padrões de projetos escolhidos

A seguir os padrões escolhidos com uma breve explicação de como eles serão aplicados e por que foram selecionados.

## Singleton

O Singleton é um padrão de projeto criacional. Padrões Criacionais ajudam a gerenciar a criação de objetos e aplicações de software, promovendo flexibilidade e escalabilidade, bem como facilitam a mudança de comportamento em tempo de execução e a comunicação entre objetos.

O Singleton é utilizado neste projeto para gerenciar a configuração global do sistema.

A principal motivação para utilizar o Singleton é garantir que apenas uma única instância de um determinado objeto exista em todo o sistema. No contexto do gerenciamento de pedidos, seria ineficiente e potencialmente problemático permitir que múltiplas instâncias de configuração global fossem criadas, pois isso poderia levar a inconsistências e dificuldades de manutenção.

A solução proposta pelo padrão Singleton é garantir que o objeto seja instanciado a partir da Classe Configuration uma única vez em todo o programa. Para isso, o construtor dessa classe é tornado privado e um método estático é fornecido para acessar a instância única do objeto. Se o objeto não existir, ele será criado; se já existir, será retornada a referência do objeto existente.

A figura 1 apresenta o diagrama de classes da utilização do padrão.

Uma imagem contendo Diagrama

Descrição gerada automaticamente

Figura 1 Diagrama de Classes da utilização do padrão Singleton no projeto

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

## Strategy

O padrão Strategy é um padrão de design comportamental que permite definir uma família de algoritmos, encapsulá-los e torná-los intercambiáveis. O Strategy permite que o algoritmo varie independentemente dos clientes que o utilizam. (GoF pág. 315).

Este padrão é útil quando há múltiplas formas de realizar uma operação específica, e você deseja selecionar a implementação apropriada em tempo de execução.

Dessa forma, a utilização do Padrão Strategy neste projeto proporciona:

* Encapsulamento em objetos de família de Algoritmos (estratégias) para cálculo de frete tornando o código mais organizado e de fácil manutenção.
* Intercambialidade facilitando a troca de algoritmos de frete, permitindo que o sistema escolha a estratégia de frete mais adequada com base em condições específicas e em tempo de execução.
* Aderência ao Princípio Aberto/Fechado: Novas estratégias podem ser adicionadas sem modificar as classes existentes, promovendo a extensibilidade do código.
* Simplificação de Código Condicional: Evita a complexidade de grandes blocos de código condicional, delegando a responsabilidade para classes específicas.

Para implementar o padrão, criamos uma interface *ShippingStrategy* (que define o comportamento), uma classe *Order* (que implementa o uso do comportamento), e “classes de algoritmos concretas” (que implementam comportamentos específicos *NormalShipping, ExpressShipping e Sedex10Shipping*).

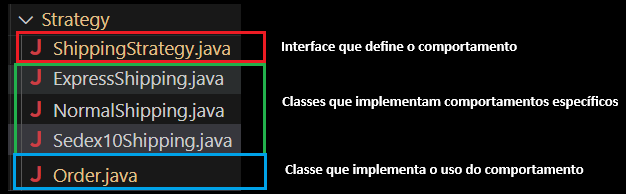
Assim, em tempo de execução, selecionamos uma instância das classes concretas (*NormalShipping, ExpressShipping e Sedex10Shipping*) dentro da classe de contexto (Order).

A figura 1 apresenta o diagrama de classes da utilização do padrão.

Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente

Figura 2 Diagrama de Classes da utilização do padrão Strategy no projeto





Interface:

Texto

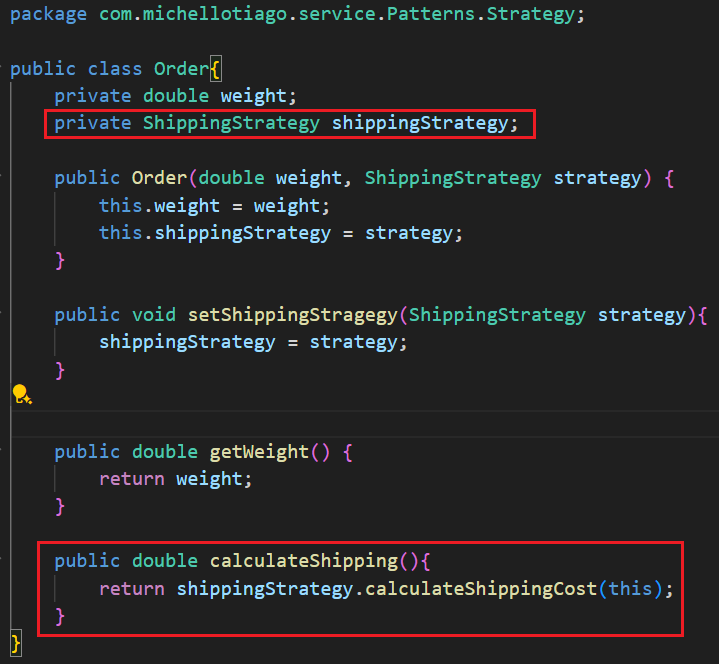
Descrição gerada automaticamente

Classes que implementam a interface:

Texto

Descrição gerada automaticamente

Classe que implementa o uso do comportamento:



Classe App:

Texto

Descrição gerada automaticamente

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

## Factory Method

O padrão de projeto Factory Method, assim como o Singleton (apresentado anteriormente), pertence à categoria dos padrões criacionais os quais simplificam a criação de objetos para tornar sistemas de software mais independentes das formas como seus objetos são criados e compostos, promovendo flexibilidade e escalabilidade.

O Factory Method é um padrão criacional que permite a criação de objetos sem especificar a classe exata a ser criada, utilizando uma função especializada.

Neste projeto, útil para flexibilizar e expandir os componentes de um sistema sem a necessidade de modificar o código existente.

Diagrama

Descrição gerada automaticamente

Figura 3 Diagrama de Classes da utilização do padrão Factory Method no projeto

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Texto

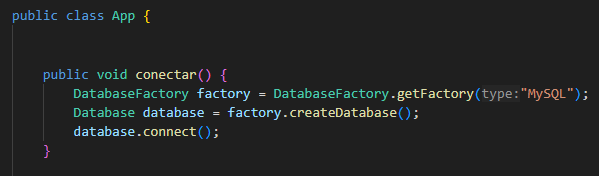
Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente



## Observer

O Observer é um padrão de projeto da categoria comportamental a qual indica como organizar o projeto e a comunicação entre os objetos.

O padrão de projeto Observer permite a um objeto notificar outros objetos sobre mudanças em seu estado. Ele define uma dependência um-para-muitos entre objetos de modo que quando um objeto muda de estado, todos os seus dependentes são automaticamente notificados e atualizados.

O Obsever será utilizado neste projeto para notificar os clientes sobre mudanças no status dos produtos que ele manifestou interesse. Isso é feito sem manter os objetos rigidamente acoplados, criando uma interface “Isubject” que mantém uma lista de seus dependentes e os notifica qualquer mudança de estado. As instâncias das classes concretas que implementam a interface “Iobservador” são aquelas que desejam ser informados quando ocorrem mudanças no Subject. Em suma, os Observadores se registram com o Subject e são notificados automaticamente de quaisquer mudança.

Esse padrão traz vários benefícios ao projeto, entre eles o desacoplamento, uma vez que o Subject não precisa saber nada sobre os Observadores, além do que eles implementam a interface do observador. Isso também permite maior flexibilidade uma vez que novos Observadores podem ser adicionados a qualquer momento sem notificar o Subject. Tudo isso proporciona maior coesão, uma vez que os objetos se mantêm focados em suas responsabilidades.

A seguir reproduzimos na Figura 4 o Diagrama de Classes e na sequência um excerto do código com a implementação desse padrão de projeto Observer.

Interface gráfica do usuário

Descrição gerada automaticamente com confiança média

Figura 4 Diagrama de Classes da utilização do padrão Observer no projeto

O método update é chamado quando o estado do Subject mudar.

Texto

Descrição gerada automaticamente

Os observadores concretos são ObservadorEmail e ObservadorSistema.

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Abaixo o o objeto observado “Subject” que implementa a interface “Isubject”. Essa é uma implementação geral. O objeto “produto” é o que se observa a partir da atualização da variável “state” pelo método “setState”.

Texto

Descrição gerada automaticamente



A seguir a criação de um produto novo, os registros dos observadores concretos e a mudança de estado que notifica os dois observadores sobre um novo produto de interesse do cliente.

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

## Decorator

Adiciona responsabilidades adicionais a um objeto dinamicamente, oferecendo uma alternativa flexível ao uso de subclasses para extensão de funcionalidades.

Interface gráfica do usuário, Site, Teams

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Adição de responsabilidades adicionais ao objeto endereçado de forma dinâmica, oferecendo alternativa flexível ao uso de subclasses para extensão de funcionalidades (endereçado simples, caixa alta e com bordas).

Texto

Descrição gerada automaticamente

## Data Access Object

É um padrão estrutural utilizado para transferir dados entre diferentes partes de um sistema. Define um objeto simples para encapsular um conjunto de dados relacionados, sem funcionalidades complexas, ajudando a reduzir o tráfego de rede e aprimorando a eficiência da comunicação em sistemas distribuídos. Também permite uma representação unificada de dados, independente das fontes originais. É especialmente útil em aplicações que lidam com grandes volumes de dados ou integração de sistemas heterogêneos.

Linha do tempo

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

# Aplicação das Regras do Projeto do CEI CERT

## Validação e Sanitização de Entrada

### IDS00-J. Prevent SQL injection (e XSS, caso web)

As vulnerabilidades de injeção SQL surgem em aplicativos onde os elementos de uma consulta SQL se originam de uma fonte não confiável. Sem precauções, os dados não confiáveis ​​podem alterar a consulta de forma maliciosa, resultando em vazamento de informações ou modificação de dados. Os principais meios de evitar a injeção de SQL são a sanitização e a validação, que normalmente são implementadas como consultas parametrizadas e procedimentos armazenados.

A solução é usa uma consulta paramétrica com caractere coringa, “?”, representando ele um espaço reservado para o argumento.

Texto

Descrição gerada automaticamente

### IDS01-J. Normalize strings before validating them

Muitos aplicativos que aceitam strings de entrada não confiáveis ​​empregam mecanismos de filtragem e validação de entrada com base nos dados de caracteres das strings. Por exemplo, a estratégia de um aplicativo para evitar vulnerabilidades de cross-site scripting (XSS) pode incluir a proibição de tags <script> nas entradas. Esses mecanismos de lista negra são uma parte útil de uma estratégia de segurança, embora sejam insuficientes para a validação e higienização completas das entradas.

As strings antes de serem usadas são validadas. Representações alternativas da string são normalizadas para os colchetes canônicos. Consequentemente, a validação de entrada detecta corretamente a entrada maliciosa e lança uma IllegalStateException.

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

### IDS03-J. Do not log unsanitized user input

Uma vulnerabilidade de injeção de log surge quando uma entrada de log contém entrada de usuário não higienizada. Um usuário mal-intencionado pode inserir dados de log falsos e, consequentemente, enganar os administradores do sistema quanto ao comportamento do sistema [OWASP 2008]. Por exemplo, um invasor pode dividir uma entrada de log legítima em duas entradas de log, inserindo uma sequência de retorno de carro e alimentação de linha (CRLF) para enganar um auditor. Os ataques de injeção de log podem ser evitados higienizando e validando qualquer entrada não confiável enviada para um log.

Esta solução compatível limpa o nome de usuário antes de registrá-lo, evitando ataques de injeção.

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

## Declarações

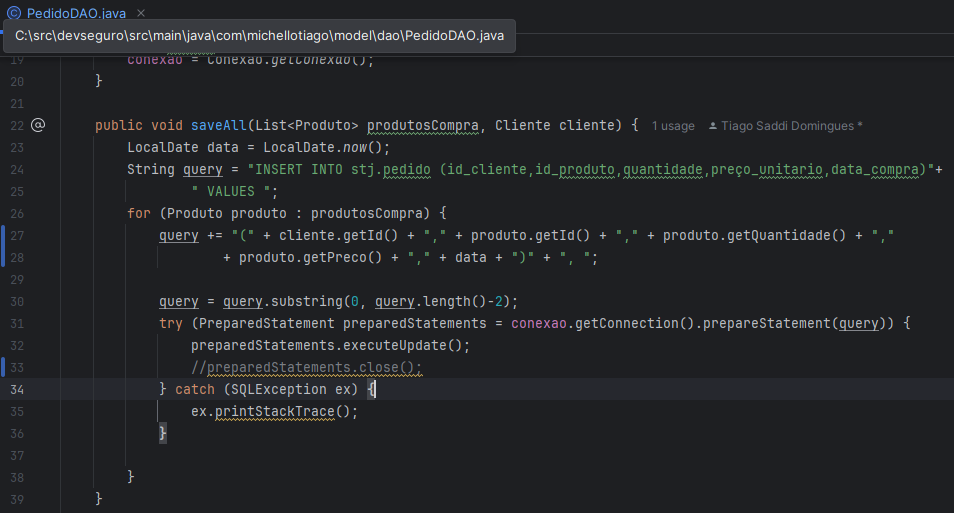
### DCL01-J. Do not reuse public identifiers from the Java Standard Library

Quando um desenvolvedor usa um identificador que tem o mesmo nome de uma classe pública, como Vector, um mantenedor subsequente pode não saber que esse identificador na verdade não se refere a java.util.Vector e pode usar involuntariamente o Vector personalizado em vez do original Classe java.util.Vector. O tipo customizado Vector pode ocultar um nome de classe de java.util.Vector, conforme especificado pela The Java Language Specification (JLS), §6.3.2, "Obscured Statements" [JLS 2005], e pode ocorrer comportamento inesperado do programa.

**Essa prática foi aplicada em todo o código-fonte.**

### DCL02-J. Do not modify the collection's elements during na enhanced for statement

No Código, não há casos de modificação de coleções na iteração, por se usar os padrões da linguagem aplicada ao Java 8 e superior (“for each”, por exemplo).



## Comportamento Excepcional

### ERR00-J. Do not suppress or ignore checked exceptions

A regra EXP00-J do SEI CERT Oracle Coding Standard for Java enfatiza a importância de não ignorar os valores retornados pelos métodos. Ignorar esses valores pode levar a riscos de segurança ou erros lógicos no seu programa.

Aqui está um exemplo da aplicação desta regra no código, mais especificamente linhas 80 a 83):

Texto

Descrição gerada automaticamente

### ERR07-J. Do not throw RuntimeException, Exception, or Throwable

A regra ERR07-J do SEI CERT Oracle Coding Standard for Java desaconselha lançar RuntimeException, Exception ou Throwable diretamente. Essa prática pode obscurecer a causa do erro e dificultar o tratamento adequado da exceção pelo chamador.

Obs.: michello incluir exemplo do banco de dados

**O Código do projeto não lança RuntimeException, Exception ou Throwable em nenhum ponto.**

## Diversos:

### MSC01-J. Do not use an empty infinite loop

Um loop infinito com corpo vazio consome ciclos de CPU, mas não faz nada. A otimização de compiladores e sistemas just-in-time (JITs) tem permissão para (talvez inesperadamente) remover esse loop. Consequentemente, os programas não devem incluir loops infinitos com corpos vazios.

No programa não há loop com corpo vazio:

Texto

Descrição gerada automaticamente

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

### MSC04-J. Do not leak memory

Erros de programação podem impedir a coleta de lixo de objetos que não são mais relevantes para a operação do programa. O coletor de lixo coleta apenas objetos inacessíveis; consequentemente, a presença de objetos atingíveis que permanecem sem uso indica má administração da memória. O consumo de todo o espaço de heap disponível pode causar um OutOfMemoryError, que geralmente resulta no encerramento do programa.

Vazamentos excessivos de memória podem levar ao esgotamento da memória e negação de serviço (DoS) e devem ser evitados (consulte MSC05-J. Não esgote o espaço de heap para obter mais informações). Para retirar a possibilidade de leak, a lista foi criada no método, linha 18:

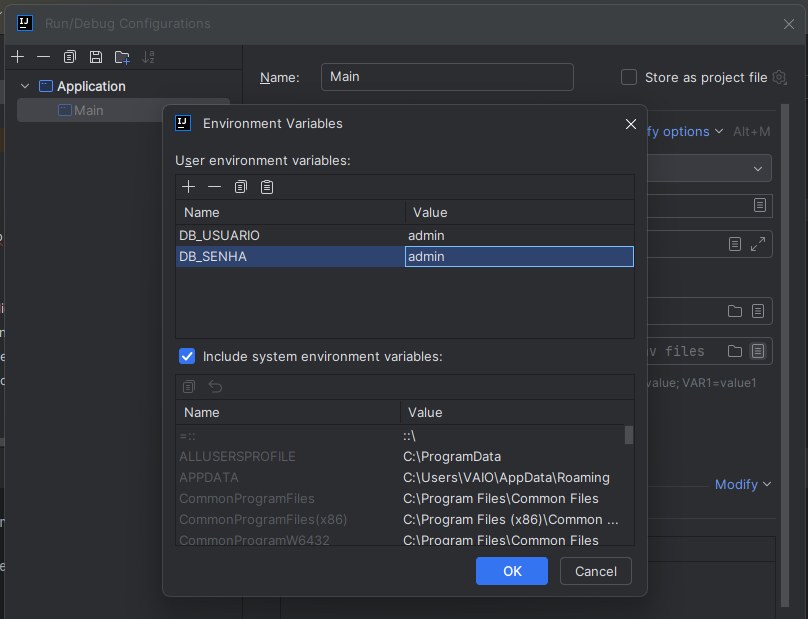
Texto

Descrição gerada automaticamente

### MSC02-J. Generate strong random number

**Não se aplica ao projeto, por não fazermos uso.**

### MSC03-J. Never hard code sensitive information



Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

### MSC07-J. Prevent multiple instantiations of singleton objects

Como há apenas uma instância singleton, "quaisquer campos de instância de um singleton ocorrerão apenas uma vez por classe, assim como campos estáticos. Singletons geralmente controlam o acesso a recursos como conexões de banco de dados ou soquetes" [Fox 2001].

Outras aplicações de singletons envolvem manter estatísticas de desempenho, monitorar e registrar a atividade do sistema, implementar spoolers de impressora e até mesmo tarefas como garantir que apenas um arquivo de áudio seja reproduzido por vez. Classes que contêm apenas métodos estáticos são boas candidatas para o padrão singleton.

O padrão singleton normalmente usa uma única instância de uma classe que envolve um campo de classe estático privado. A instância pode ser criada usando inicialização lenta, o que significa que a instância não é criada quando a classe é carregada, mas quando é usada pela primeira vez.

Uma classe que implementa o padrão de design singleton deve evitar múltiplas instanciações. Técnicas relevantes incluem o seguinte:

* Tornando seu construtor privado;
* Empregando mecanismos de bloqueio para evitar que uma rotina de inicialização seja executada simultaneamente por vários threads;
* Garantindo que a classe não seja serializável;
* Garantindo que a classe não possa ser clonada;
* Evitando que a classe seja coletada como lixo se ela foi carregada por um carregador de classe personalizado.

No código temos a classe configuration, usando o padrão singleton:

Texto

Descrição gerada automaticamente