

Entrega 1

Grupo: Adriano de Souza, Carolina Arêas, Daniel Neves, Natália Bruno Rabelo, Vitor Hugo Prado

1. Descrição do escopo do sistema

1.1 Introdução

O Sistema de Gestão Clínica WeB foi projetado para fornecer uma solução abrangente e integrada para o gerenciamento eficaz das operações dentro de uma clínica médica.

1.2 Funções Principais

O sistema inclui uma série de módulos funcionais estrategicamente concebidos para abordar as diversas necessidades operacionais de uma clínica médica, tais como:

Agendamento de Consultas e Exames: Esta funcionalidade lida com a marcação eficiente de consultas e exames, permitindo uma gestão de tempo otimizada. O usuário também tem a possibilidade de editar e reagendar consultas.

Gestão de Usuários: Esta funcionalidade inclui o cadastro, a manutenção e o controle de login para vários tipos de usuários - médicos, administradores e pacientes.

Gestão de Especialidades Médicas: Esta funcionalidade fornece ferramentas para gerenciar e rastrear as várias especialidades médicas oferecidas na clínica.

Gestão de Planos de Saúde: Esta funcionalidade permite o controle eficiente dos diferentes planos de saúde disponíveis, assegurando um atendimento adequado aos pacientes.

1.3 Requisitos Não Funcionais

RNF1: Sistema deverá ser responsivo sendo obrigatório o uso do Bootstrap 5.0 (ou superior);

RNF2: Todas as bibliotecas, scripts, imagens etc., necessários para o funcionamento devem estar disponíveis localmente;

4.2 Requisitos Funcionais

4.2.1 Requisitos – Paciente

RF1: O paciente faz o registro na aplicação, informando nome, CPF, senha e tipo do plano de saúde.

RF2: O paciente autorizado acessa a área privada da aplicação (faz o login) e tem acesso ao seu painel de controle que possui: a lista das consultas (realizadas e não realizadas) e a ação de marcação de consulta.

RF3: O paciente autorizado marca a consulta escolhendo o médico (especialidade médica) e a data.

4.2.2 Requisitos - Médico

RF4: O médico autorizado acessa a área privada da aplicação (faz o login) e tem acesso ao seu painel de controle que possui: a lista das consultas (realizadas e não realizadas) e a ação de realização da consulta.

RF5: O médico realiza a consulta descrevendo o que ocorreu e solicitando exames quando necessário.

RF6: O médico visualiza e edita os dados de uma consulta já realizada por ele.

4.2.3 Requisitos - Administradores

RF7: O administrador acessa a área privada da aplicação (faz login) e tem acesso ao seu painel de controle que possui as seguintes ações: cadastrar médicos, cadastrar pacientes, cadastrar administradores, cadastrar os tipos de plano de saúde e cadastrar as especialidades médicas.

RF8: O administrador cadastra os médicos.

RF9: O administrador cadastra as especialidades.

RF10: O administrador cadastra os tipos de planos de saúde.

RF11: O administrador retira a autorização do paciente para acessar a aplicação.
RF12: O administrador retira a autorização do médico para acessar a aplicação.
RF13: O administrador visualiza as consultas de um médico.

2. Plano de testes

Os testes serão realizados em várias fases do desenvolvimento para garantir que o sistema atenda aos padrões de qualidade e às expectativas do usuário.

- **Testes unitários:** Esses testes verificarão a funcionalidade de cada componente individual do sistema.
- **Testes de integração:** Esses testes verificarão a interação adequada entre os diferentes componentes do sistema.
- **Testes de sistema:** Esses testes verificarão a funcionalidade do sistema como um todo, para garantir que todas as partes estejam funcionando corretamente juntas.
- **Testes de aceitação:** Esses testes serão realizados com a ajuda dos usuários finais para garantir que o sistema atenda às suas necessidades e expectativas.
- **Testes de qualidade de código:** Uma fase no processo de testes que se concentra em melhorar a qualidade do código, aumentando a manutenibilidade, reduzindo a complexidade e corrigindo pequenos erros que podem se tornar grandes problemas no futuro.
- **Testes de performance:** Esses testes serão realizados para garantir que o sistema é capaz de funcionar eficientemente sob carga pesada ou grande volume de dados.

2.1 Artefatos gerados

- Documentação de requisitos do sistema
- Código-fonte Java
- Casos de teste
- Relatório de inspeção de código fonte

2.2 Ferramentas utilizadas

- Java JDK para desenvolvimento
- Netbeans para desenvolvimento integrado de ambiente (IDE)
- Testlink para elaboração de casos de teste e planos de teste
- JUnit e Mockito para testes unitários
- Postman para testes de integração
- JMeter e Locust para testes automatizados de sistema
- JBehave para testes de aceitação baseados em BDD
- SonarLint e Checkstyle para qualidade de código
- Mantis para criação de relatório de incidente de teste

3. Código-fonte original

O código-fonte original do projeto encontra-se neste repositório abaixo:

<https://github.com/adriano-uff/ClinicaWeB/tree/main/ProjetoWeb>

4. Indicação das medidas de cada atributo de qualidade da ISO 25010 seguindo uma escala. Justificar as decisões para indicação das medidas.

A ISO 25010 estabelece vários atributos de qualidade para software, que foram levados em consideração na concepção deste sistema. A seguir estão as medidas de cada atributo de qualidade seguindo uma escala de 0 a 5, onde 5 atinge o nível mais alto, com justificativas para as decisões tomadas:

- **Adequação funcional:**

- Nota: 5
- Justificativa: O sistema atende aos requisitos necessários para a gestão de uma clínica médica, a decisão de incluir determinadas funções foi tomada com base nas necessidades dos usuários finais. Aprofundando essa questão, o sistema implementa de forma correta todas as funcionalidades planejadas e comportamentos esperados.

- **Eficiência do desempenho:**

- Nota: 5
- Justificativa: O sistema é otimizado para processamento rápido e eficiente, a eficaz utilização dos recursos também torna o sistema capaz de suportar a carga de trabalho em uma clínica médica em tempo real e suas demais necessidades.

- **Compatibilidade:**

- Nota: 3
- Justificativa: O sistema foi projetado para ser compatível com sistemas operacionais e bancos de dados comumente utilizados. A escolha do Java como linguagem de programação foi feita devido à sua portabilidade e compatibilidade, no entanto, a versão das ferramentas torna possível ou não o funcionamento do sistema (como exemplo: a versão do Mysql). Além disso, uma nota máxima nesse atributo exigiria garantir que não haja conflitos ou impactos negativos em suas operações individuais, apesar de baixa probabilidade, ainda é possível esse ocorrido.

- **Usabilidade:**

- Nota: 3
- Justificativa: O sistema tem uma interface de usuário intuitiva, tornando-o fácil de usar mesmo para pessoas sem experiência técnica. Porém, a cobertura de proteção contra erros do usuários é baixa, ilustrando esse fato, apesar de haver validações (exemplo: validação do CPF), são realizadas em poucas funcionalidades, também possui poucas formas de avisos e questionamentos ao usuário, embora na maioria dos casos é avisado quando uma ação teve sucesso, não há questionamentos de confirmação de ações. Além disso possui baixa acessibilidade, como exemplo, não possui funcionalidades para auxiliar durante o uso feito por usuários com deficiência visual,

- **Confiabilidade:**

- Nota: 4
- Justificativa: O sistema possui um backup completo e bem documentado, assim a recuperabilidade do sistema se torna fácil. Porém, apesar de ser um sistema completo, ele foi desenvolvido em um curto período de tempo e depois não sofreu mais análises ou mudanças, e assim não passou por um ciclo de aprendizado e melhoria contínua, tendo como consequência uma falta de maturidade,

- **Segurança:**

- Nota: 3
- Justificativa: O sistema foi projetado para ser robusto e confiável, com redundância de dados e recuperação de falhas para garantir a integridade dos dados. Além disso, não realiza nenhuma forma de divulgação ou uso não autorizado. Porém, apesar de possuir senhas para garantir sua

autenticidade, tais senhas apresentam falhas e assim o sistema não consegue assegurar uma das principais formas de segurança.

- **Manutenibilidade:**

- Nota: 5
- Justificativa: O código é estruturado de forma clara, tornando-o fácil de manter, atualizar e testar. Além disso sua transparência, estrutura objetiva e boa documentação torna simples sua análise é possível a reutilização de código, funcionalidades ou recursos.

- **Portabilidade:**

- Nota: 5
- Justificativa: O sistema foi projetado para ser facilmente portado para diferentes sistemas operacionais e bancos de dados, é capaz de realizar a instalação sem erros ou problemas. Também no caso de um componente, recurso etc ser substituído, o sistema tem a capacidade de suportar essa mudança sem causar impactos significativos.

As decisões tomadas ao determinar as medidas para cada atributo de qualidade foram baseadas na necessidade de fornecer um sistema de alta qualidade que atenda às necessidades dos usuários e cumpra os padrões de segurança e ética no setor médico.

5. Relatório de inspeção do código-fonte e ou documentação do software.

5.1. Objetivos:

Realizar uma inspeção de código com software adequado para o projeto para analisar a qualidade do código e consequentemente detectar bugs, vulnerabilidades e code smells.

5.2. Metodologia:

Foi utilizado o plugin SonarLint para o NetBeans, executando a função Analyzer em cada pasta da aplicação para descobrirmos todos os problemas do código.

5.3 Resultados:

Apresentação dos problemas encontrados durante a inspeção de código, incluindo:

5.4.1 Problema 1:

- **Descrição:** O erro “AvoidCommentOutCodeCheck” é caracterizado pela presença de comentários em trechos de código que não deveriam estar comentados.

- **Localização:** O erro se localiza no arquivo AreaAdministrador.jsp.

- **Prioridade:** Major (Code Smell).

- **Recomendação:** Remoção dos comentários desnecessários.

5.4.2 Problema 2:

- **Descrição:** O erro “PageWithoutTitleCheck” descreve que todas as páginas Web deveriam ter um título.
- **Localização:** Há ocorrência em 3 arquivos diferentes (CadastrarEspecialidade.jsp, CadastrarExame.jsp e CadastrarPlano.jsp). Abaixo o exemplo é demonstrado na página CadastrarExame.jsp.

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="aplicacao.Exame"%>
<%
Web:PageWithoutTitleCheck: "<title>" should be present in all pages
Click to show details
    {
        exame = (Exame)session.getAttribute("exame");
    }
    catch(NullPointerException e){
    }
}%>

<!DOCTYPE html>
<html lang="pt-BR">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<%
    if(exame != null){
        out.println("<title>Editor Exame</title>");
    }else{
        out.println("<title>Cadastrar Exame</title>");
    }
}%>
<link href="./bootstrap/bootstrap.min.css" rel="stylesheet">

```

- **Prioridade:** Major (Bug).
- **Recomendação:** O SonarLint não detectou a condição para mudar o texto exibido como título e acusou como erro.

5.4.3 Problema 3:

- **Descrição:** O erro “S5254” destaca a falta da atribuição de uma idioma no elemento <html>.
- **Localização:** Esse erro teve 15 ocorrências diferentes no código fonte da aplicação. O arquivo AreaAdministrador.jsp é um exemplo.

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="aplicacao.Administrador"%>
Web:S5254: "<html>" element should have a language attribute
Click to show details
    {
        adm = (Administrador)session.getAttribute("adm");
    }
}%>

<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Área do Administrador</title>
<link rel="stylesheet" type="text/css" href="./styles/BotaoAreaInterna.css"/>
<link href="./bootstrap/bootstrap.min.css" rel="stylesheet">
<style>
.quadro {
    padding: 50px;
    margin: 20px;
    text-align: center;

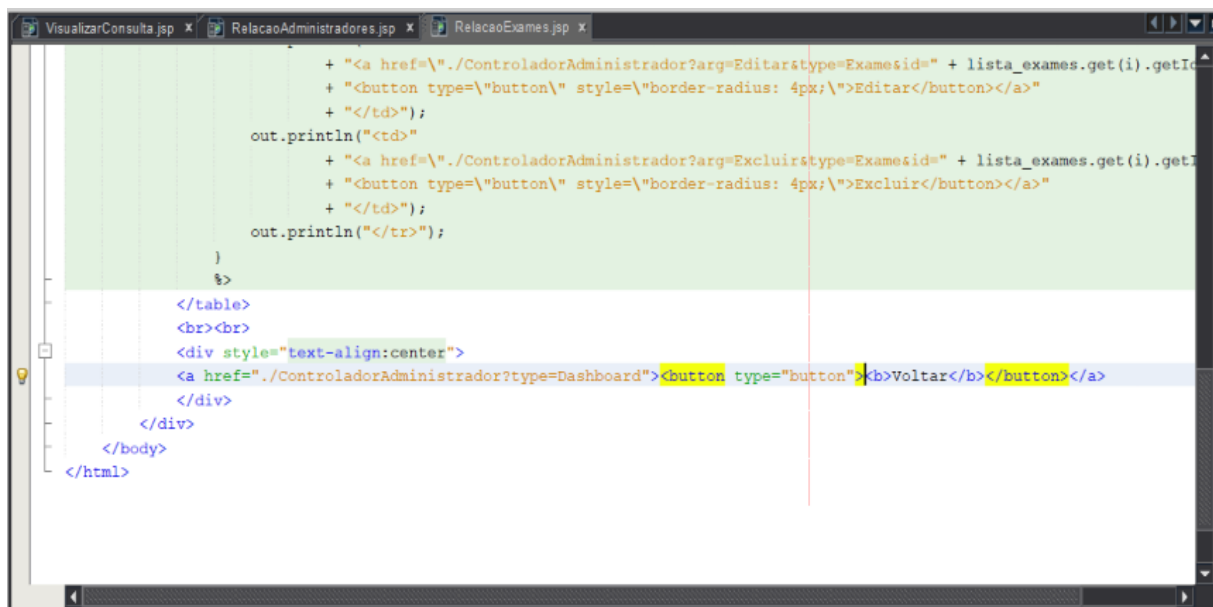
    background-color: white;
    border: 15px solid blue;
    padding: 50px;
    margin-top: 50px;
    width: 800px;
}

```

- **Prioridade:** Major (Bug).
- **Recomendação:** Adicionar o parâmetro lang="idioma escolhido" dentro da tag <html>.

5.4.4 Problema 4:

- **Descrição:** O erro “BoldAndItalicTagsCheck” indica casos onde foram usadas as tags ou <i>.
- **Localização:** Existem 27 ocorrências desse erro. Abaixo é mostrado um exemplo no trecho de código no arquivo JSP RelacaoExame.

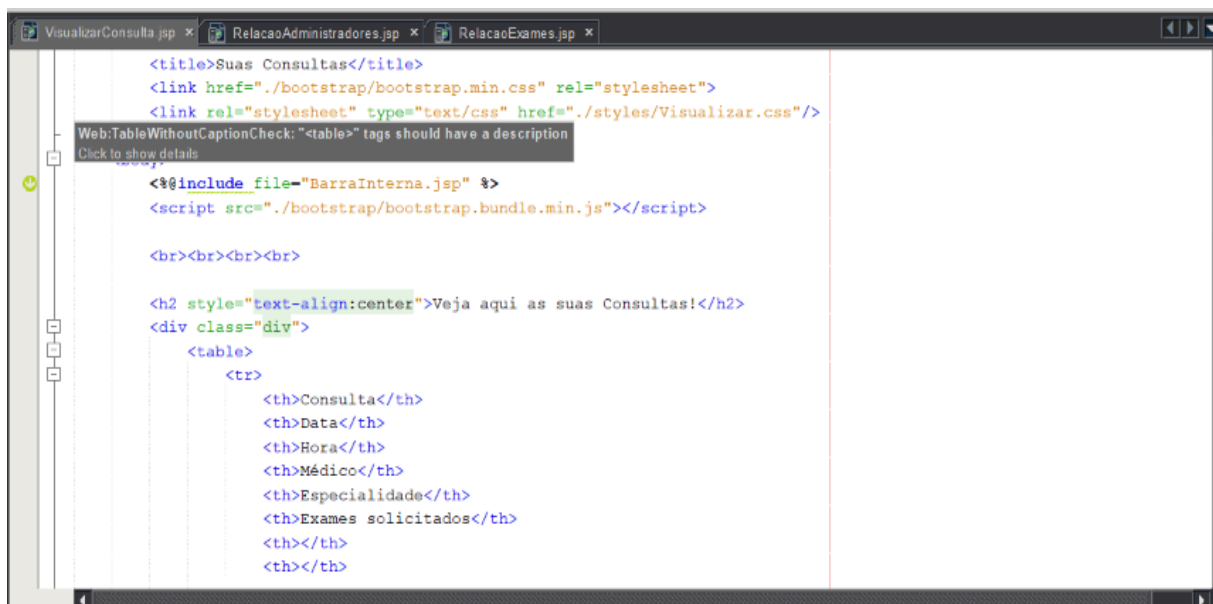


```
+ "<a href=\"../ControladorAdministrador?arg=Editar&type=Exames&id=\" + lista_exames.get(i).getId() + \"><button type=\"button\" style=\"border-radius: 4px;\">Editar</button></a>"
+ "</td>";
out.println("<td>");
+ "<a href=\"../ControladorAdministrador?arg=Excluir&type=Exames&id=\" + lista_exames.get(i).getId() + \"><button type=\"button\" style=\"border-radius: 4px;\">Excluir</button></a>"
+ "</td>";
out.println("</tr>");
}
%>
</table>
<br><br>
<div style="text-align:center">
<a href=" ../ControladorAdministrador?type=Dashboard"><button type="button"><b>Voltar</b></button></a>
</div>
</div>
</body>
</html>
```

- **Prioridade:** Menor (Bug).
- **Recomendação:** Substituir as tags e <i> por e no código por recomendação.

5.4.5 Problema 5:

- **Descrição:** O erro “TableWithoutCaptionCheck” acusa a falta de legendas para descrever o conteúdo da tabela para possíveis usuários com deficiência visual.
- **Localização:** O erro apresentou 9 ocorrências diferentes e uma delas acontece no arquivo VisualizacaoConsulta.jsp.



```
<title>Suas Consultas</title>
<link href=" ../bootstrap/bootstrap.min.css" rel="stylesheet">
<link rel="stylesheet" type="text/css" href=" ../styles/Visualizar.css"/>
<script src=" ../bootstrap/bootstrap.bundle.min.js"></script>

<br><br><br><br>

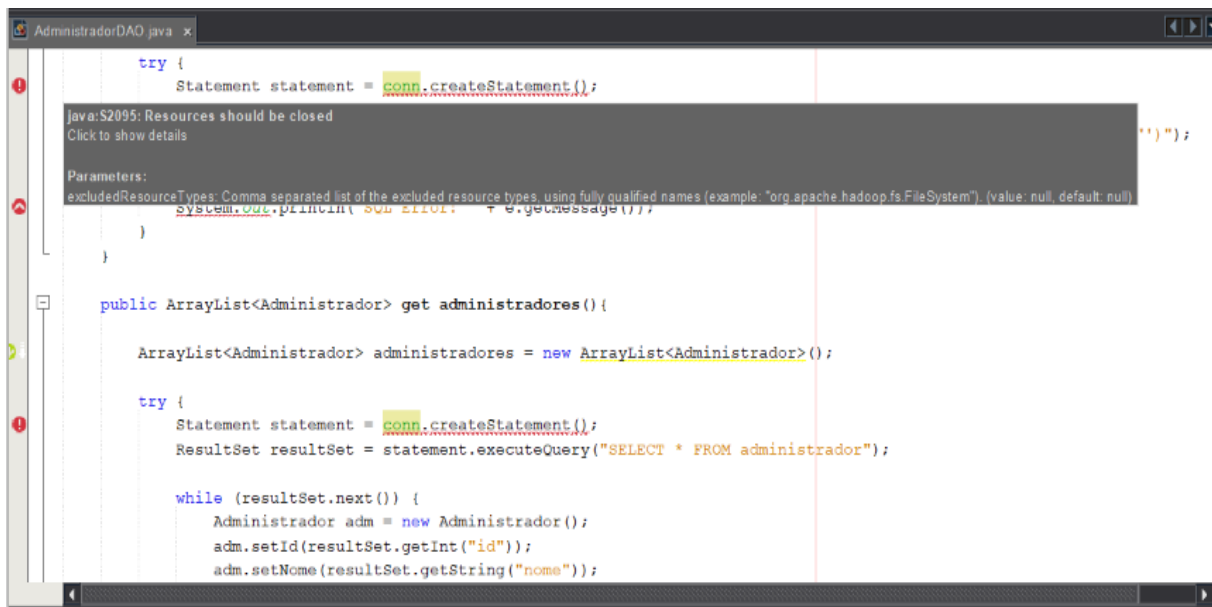
<h2 style="text-align:center">Veja aqui as suas Consultas!</h2>
<div class="div">
<table>
<tr>
<th>Consulta</th>
<th>Data</th>
<th>Hora</th>
<th>Médico</th>
<th>Especialidade</th>
<th>Exames solicitados</th>
<th></th>
<th></th>

```

- **Prioridade:** Menor (Bug).
- **Recomendação:** Adicionar a tag <caption> dentro da tag <table> para que haja descrição do conteúdo presente.

5.4.6 Problema 6:

- **Descrição:** O erro “S2095” indica que recursos foram abertos e não foram fechados posteriormente. Isso pode resultar no vazamento de recursos.
- **Localização:** Há 53 eventos detectados pelo SonarLint. Uma das incidências acontece na classe AdministradorDAO.java.



```
try {
    Statement statement = conn.createStatement();

    java:S2095: Resources should be closed
    Click to show details

    Parameters:
    excludedResourceTypes: Comma separated list of the excluded resource types, using fully qualified names (example: "org.apache.hadoop.fs.FileSystem", (value: null, default: null))
    System.out.println("SQL Error: " + e.getMessage());
}

}

public ArrayList<Administrador> get administradores(){

    ArrayList<Administrador> administradores = new ArrayList<Administrador>();

    try {
        Statement statement = conn.createStatement();
        ResultSet resultSet = statement.executeQuery("SELECT * FROM administrador");

        while (resultSet.next()) {
            Administrador adm = new Administrador();
            adm.setId(resultSet.getInt("id"));
            adm.setNome(resultSet.getString("nome"));
        }
    }
}
```

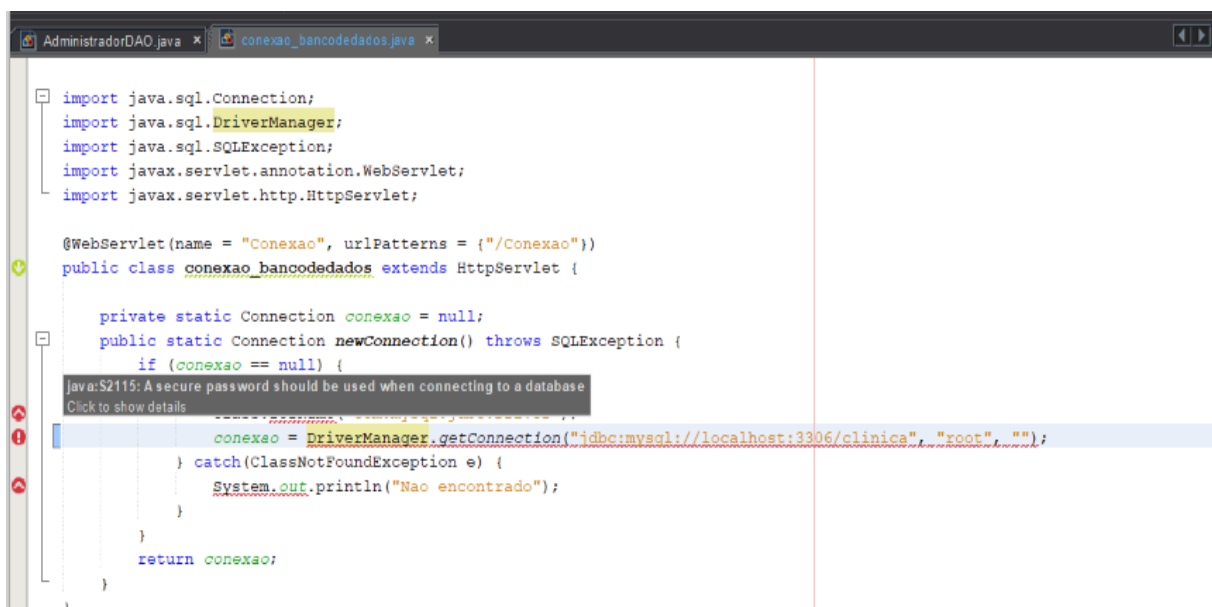
- **Prioridade:** Blocker (Bug).

- **Recomendação:** Garantir que seja feito o fechamento de todos os recursos que implementam a classe *Closeable* ou a sua super interface *AutoCloseable*.

5.4.7 Problema 7:

- **Descrição:** O erro “S2115” aponta que a senha usada para se conectar ao banco de dados é insegura.

- **Localização:** O erro está localizado na classe conexao_bancodedados.java que realiza a conexão com o banco de dados.



```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;

@WebServlet(name = "Conexao", urlPatterns = {"/Conexao"})
public class conexao_bancodedados extends HttpServlet {

    private static Connection conexao = null;

    public static Connection newConnection() throws SQLException {
        if (conexao == null) {
            java:S2115: A secure password should be used when connecting to a database
            Click to show details

            conexao = DriverManager.getConnection("jdbc:mysql://localhost:3306/clinica", "root", "");
        } catch (ClassNotFoundException e) {
            System.out.println("Nao encontrado");
        }
    }

    return conexao;
}
```

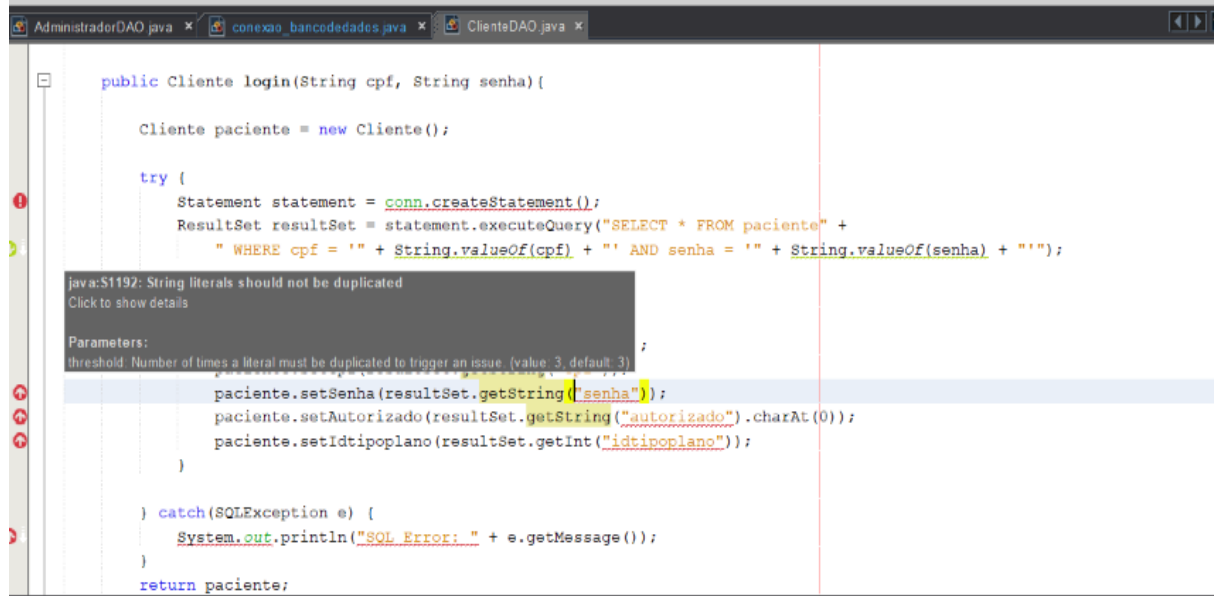
- **Prioridade:** Blocker (Vulnerability).

- **Recomendação:** Utilizar uma senha mais forte, que seja longa e contenha caracteres maiúsculos, minúsculos e caracteres especiais.

5.4.8 Problema 8:

- **Descrição:** O erro “S1192” demonstra a presença literais String duplicados, algo que pode deixar o processo de refatoração propício a erro.

- **Localização:** Existem 33 ocorrências no código-fonte. Um exemplo ocorre na classe ClienteDAO.java.



```
public Cliente login(String cpf, String senha) {  
    Cliente paciente = new Cliente();  
  
    try {  
        Statement statement = conn.createStatement();  
        ResultSet resultSet = statement.executeQuery("SELECT * FROM paciente" +  
            " WHERE cpf = '" + String.valueOf(cpf) + "' AND senha = '" + String.valueOf(senha) + "'");  
  
        paciente.setSenha(resultSet.getString("senha"));  
        paciente.setAutorizado(resultSet.getString("autorizado").charAt(0));  
        paciente.setIdtipoPlano(resultSet.getInt("idtipoPlano"));  
    } catch (SQLException e) {  
        System.out.println("SQL Error: " + e.getMessage());  
    }  
  
    return paciente;  
}
```

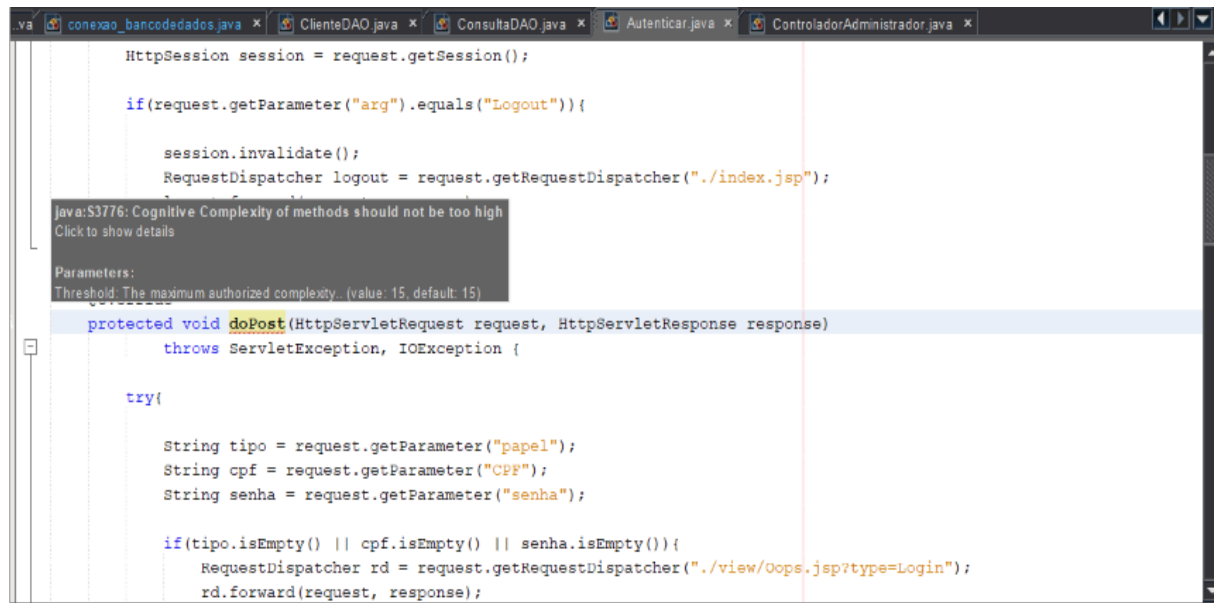
- **Prioridade:** Critical (Code Smell).

- **Recomendação:** Transformar as variáveis em constantes para evitar a probabilidade de erro e permitir que as mesmas sejam referenciadas de qualquer lugar.

5.4.9 Problema 9:

- **Descrição:** O erro “S3776” elucida a existência de métodos de alta complexidade cognitiva e esses métodos são difíceis de manter.

- **Localização:** Esse erro aparece nas classes Autenticar.java e ControladorAdministrador.java.



```
HttpSession session = request.getSession();  
  
if(request.getParameter("arg").equals("Logout")){  
    session.invalidate();  
    RequestDispatcher logout = request.getRequestDispatcher("./index.jsp");  
  
protected void doPost(HttpServletRequest request, HttpServletResponse response)  
    throws ServletException, IOException {  
  
    try{  
  
        String tipo = request.getParameter("papel");  
        String cpf = request.getParameter("CPF");  
        String senha = request.getParameter("senha");  
  
        if(tipo.isEmpty() || cpf.isEmpty() || senha.isEmpty()){  
            RequestDispatcher rd = request.getRequestDispatcher("./view/Oops.jsp?type=Login");  
            rd.forward(request, response);  
        }  
    }  
}
```

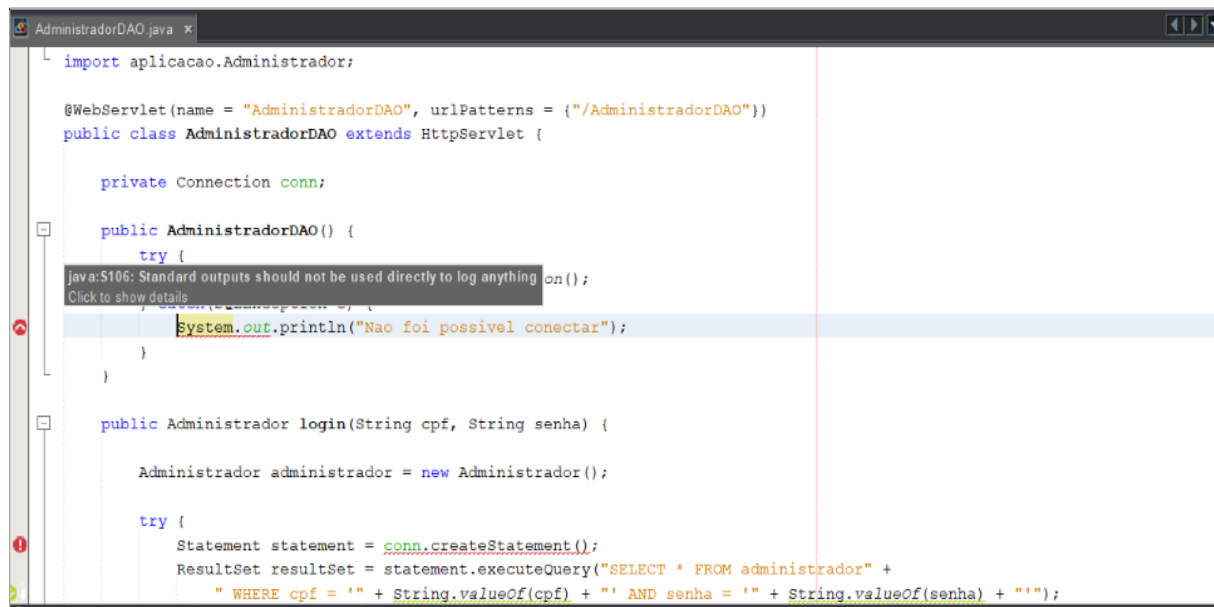
- **Prioridade:** Critical (Code Smell).

- **Recomendação:** Diminuir a complexidade dos métodos criados. Embora a grande quantidade de argumentos do método doPost seja o padrão oferecido pela própria linguagem.

5.4.10 Problema 10:

- **Descrição:** O erro “S106” descreve o problema de exibir mensagens de log em outputs comuns (system.out.println).

- **Localização:** Existem 63 ocorrências desse erro na totalidade do código. Abaixo é mostrado um exemplo na classe AdministradorDAO.java.



```
import aplicacao.Administrador;

@WebServlet(name = "AdministradorDAO", urlPatterns = {"/AdministradorDAO"})
public class AdministradorDAO extends HttpServlet {

    private Connection conn;

    public AdministradorDAO() {
        try {
            java:S106: Standard outputs should not be used directly to log anything on();
            System.out.println("Nao foi possivel conectar");
        }
    }

    public Administrador login(String cpf, String senha) {

        Administrador administrador = new Administrador();

        try {
            Statement statement = conn.createStatement();
            ResultSet resultSet = statement.executeQuery("SELECT * FROM administrador" +
                " WHERE cpf = '" + String.valueOf(cpf) + "' AND senha = '" + String.valueOf(senha) + "'");
        }
    }
}
```

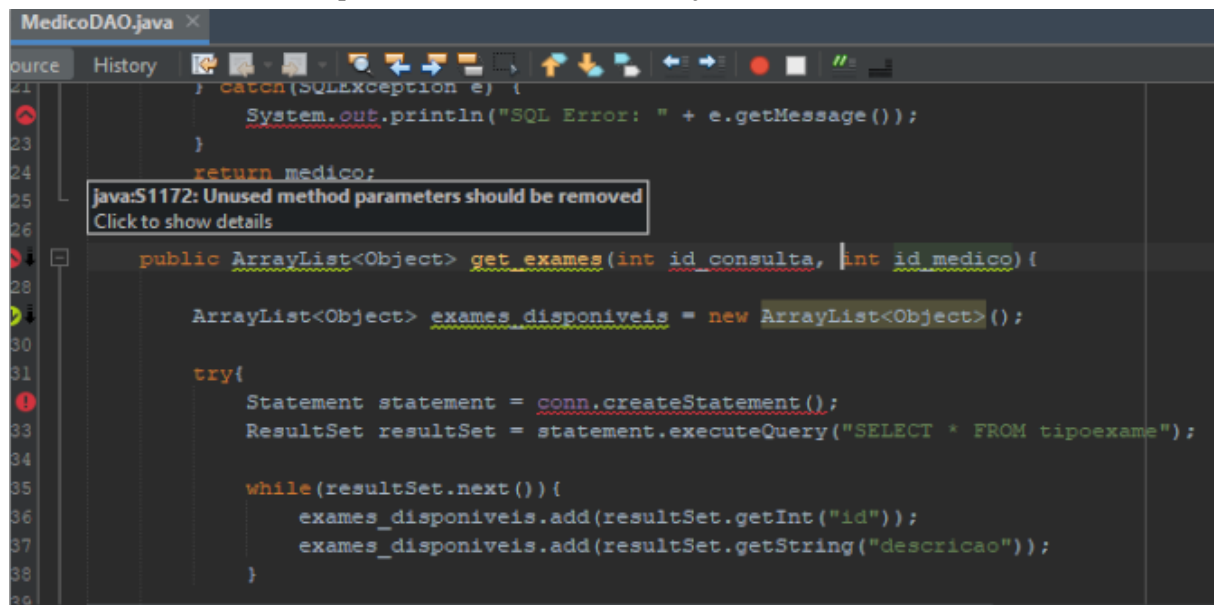
- **Prioridade:** Major (Code Smell).

- **Recomendação:** A solução é salvar os logs para o usuário poder consultá-los posteriormente e usar um formato uniforme para padronização.

5.4.11 Problema 11:

- **Descrição:** O erro “S1172” mostra a ocorrência de inserção de parâmetros que não são usados.

- **Localização:** Esse erro aparece na classe MedicoDAO.java.



```
catch(SQLException e) {
    System.out.println("SQL Error: " + e.getMessage());
}

return medico;

public ArrayList<Object> get_examenes(int id_consulta, int id_medico) {
    ArrayList<Object> exames_disponiveis = new ArrayList<Object>();

    try{
        Statement statement = conn.createStatement();
        ResultSet resultSet = statement.executeQuery("SELECT * FROM tipoexame");

        while(resultSet.next()) {
            exames_disponiveis.add(resultSet.getInt("id"));
            exames_disponiveis.add(resultSet.getString("descricao"));
        }
    }
}
```

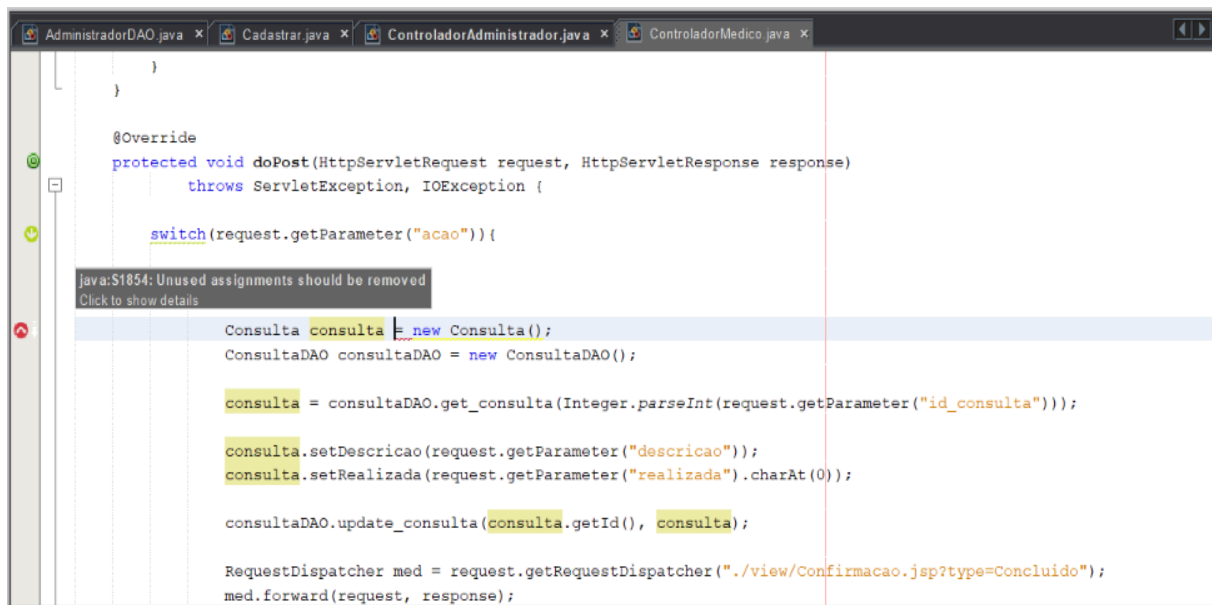
- **Prioridade:** Major (Code Smell).

- **Recomendação:** Remover todos os parâmetros que não estão sendo usados para melhorar a legibilidade do código.

5.4.12 Problema 12:

- **Descrição:** O erro “S1854” indica a ocorrência de atribuições feitas que não são usadas.

- **Localização:** 33 aparições desse erro foram capturadas pelo SonarLint. A imagem mostra uma das ocorrências na classe ControladorMedico.java.



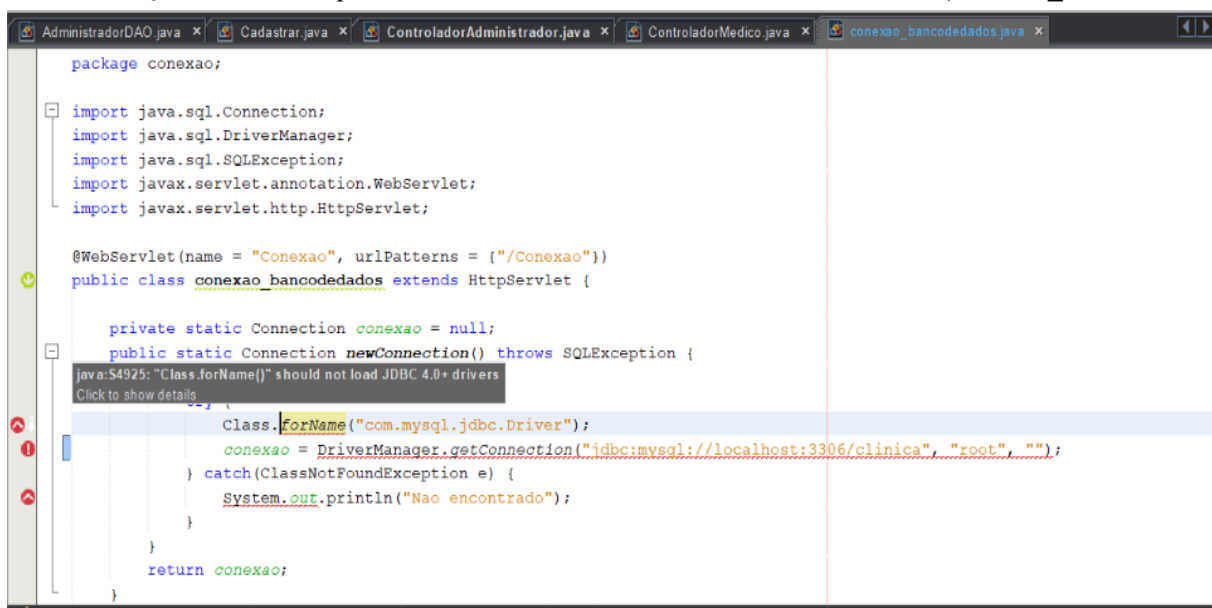
- **Prioridade:** Major (Code Smell).

- **Recomendação:** Mesmo que não seja um erro propriamente dito, pode ser considerado um desperdício de recursos. Portanto, todos os valores calculados devem ser usados ou removidos se não forem ter uso factível.

5.4.13 Problema 13:

- **Descrição:** O erro “S4925” aponta que drivers JDBC 4.0+ não precisam ser carregados pela “Class.forName()”.

- **Localização:** Esse erro aparece na classe de conexão com o banco de dados (conexao_bancodedados.java).



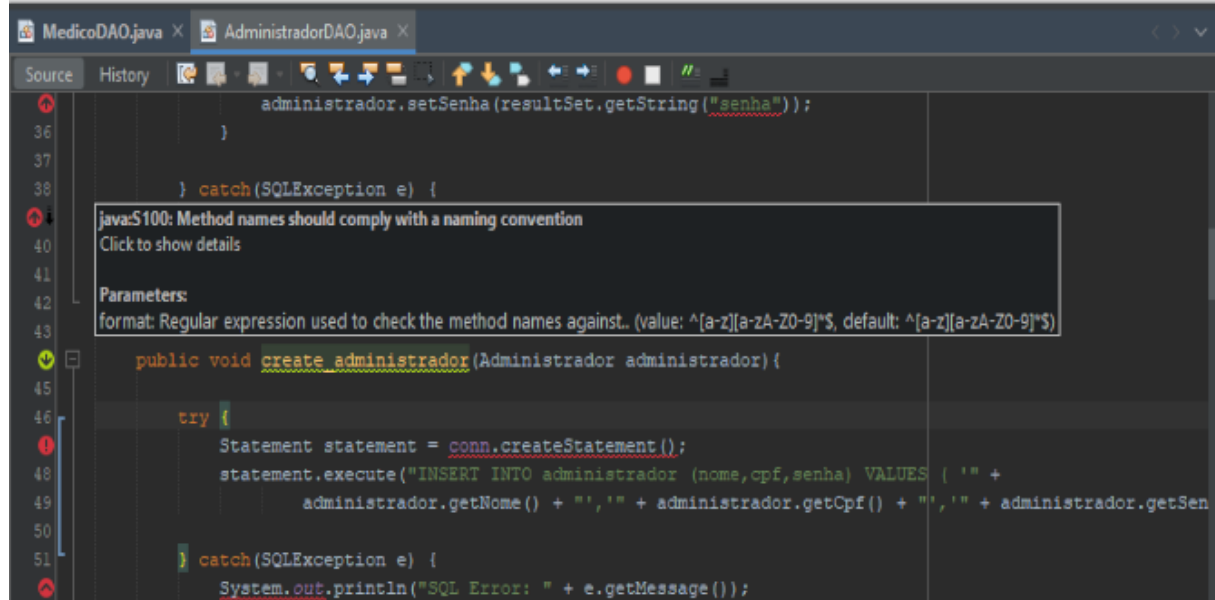
- **Prioridade:** Major (Code Smell).

- **Recomendação:** Não há necessidade de carregar o driver, pois nas versões 4.0 ou acima do driver JDBC isso é feito automaticamente.

5.4.14 Problema 14:

- **Descrição:** O erro “S100” exhibe o uso de nomes de métodos que não seguem a convenção de nomenclatura.

- **Localização:** O erro possui 49 ocorrências e o exemplo abaixo ocorre na classe AdministradorDAO.java.



```
administrador.setSenha(resultSet.getString("senha"));
}
} catch (SQLException e) {
    // ...
}

public void create_administrador(Administrador administrador) {
    try {
        Statement statement = conn.createStatement();
        statement.execute("INSERT INTO administrador (nome,cpf,senha) VALUES ( '" +
            administrador.getNome() + "','" + administrador.getCpf() + "','" + administrador.getSen
    } catch (SQLException e) {
        System.out.println("SQL Error: " + e.getMessage());
    }
}
```

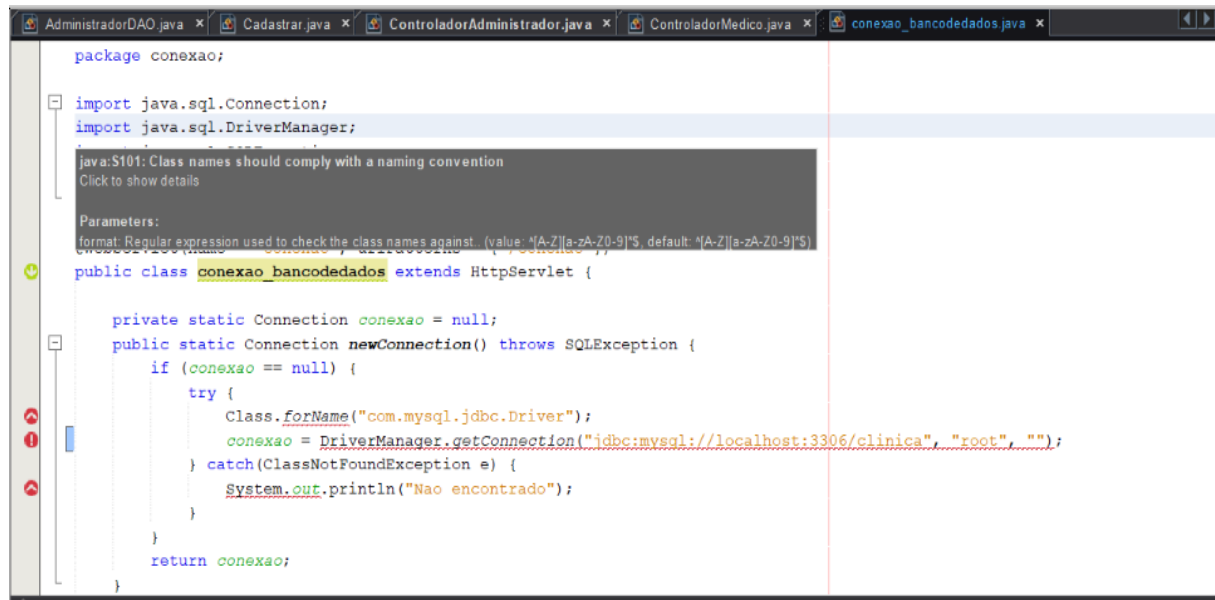
- **Prioridade:** Major (Code Smell).

- **Recomendação:** Adotar um nome que seja complacente com o padrão, como createAdministrador().

5.4.15 Problema 15:

- **Descrição:** O erro “S101” acusa o uso de um nome para a classe que não segue a convenção de nomenclatura.

- **Localização:** Assim como o erro anterior, a classe conexão_bancodedados.java que apresenta o problema.



```
package conexao;

import java.sql.Connection;
import java.sql.DriverManager;

public class conexao_bancodedados extends HttpServlet {

    private static Connection conexao = null;

    public static Connection newConnection() throws SQLException {
        if (conexao == null) {
            try {
                Class.forName("com.mysql.jdbc.Driver");
                conexao = DriverManager.getConnection("jdbc:mysql://localhost:3306/clinica", "root", "");
            } catch (ClassNotFoundException e) {
                System.out.println("Nao encontrado");
            }
        }
        return conexao;
    }
}
```

- **Prioridade:** Minor (Code Smell).

- **Recomendação:** Adotar um nome que seja complacente com o padrão, como ConexaoBancoDeDados.java.

5.4.16 Problema 16:

- **Descrição:** O erro “S1125” ilustra o uso de uma comparação booleana redundante.

- **Localização:** A questão ocorre na classe ClienteDAO.java.



```
public boolean jaCadastrado(String cpf_paciente) {  
    boolean resp = false;  
  
    try {  
        Statement statement = conn.createStatement();  
        ResultSet resultSet = statement.executeQuery("SELECT * FROM paciente "  
        if (resultSet.next() == true) {  
            resp = true;  
        }  
    } catch (SQLException e) {  
        System.out.println("SQL Error: " + e.getMessage());  
    }  
    return resp;  
}  
  
public void create_paciente(Cliente novo_paciente) {  
    try {
```

- **Prioridade:** Minor (Code Smell).

- **Recomendação:** Alterar a condição para apenas if(resultSet.next()) para acabar com a redundância.

5.4.17 Problema 17:

- **Descrição:** O erro “S1153” relata o problema em concatenar múltiplas Strings. Por se tratar de um objeto imutável, é necessário a criação de um objeto intermediário para que seja feito a junção (append) e depois a conversão novamente para String. A performance diminui à medida que o tamanho da String aumenta.

- **Localização:** Existem 6 acontecimentos desse erro. O exemplo abaixo exhibe o mesmo na classe MedicoDAO.java.



```
System.out.println("Nao foi possivel conectar");  
}  
  
public Medico login(String cpf, String senha) {  
    Medico medico = new Medico();  
  
    try {  
        Statement statement = conn.createStatement();  
        ResultSet resultSet = statement.executeQuery("SELECT * FROM medico "  
        if (resultSet.next()) {  
            medico.setId(resultSet.getInt("id"));  
            medico.setNome(resultSet.getString("nome"));  
            medico.setCrm(resultSet.getInt("crm"));  
            medico.setEstadocrm(resultSet.getString("estadocrm"));  
            medico.setCpf(resultSet.getString("cpf"));  
            medico.setSenha(resultSet.getString("senha"));  
            medico.setAutorizado(resultSet.getString("autorizado").charAt(0));  
            medico.setIdespecialidade(resultSet.getInt("idespecialidade"));  
        }  
    } catch (SQLException e) {  
        System.out.println("SQL Error: " + e.getMessage());  
    }  
    return medico;  
}
```

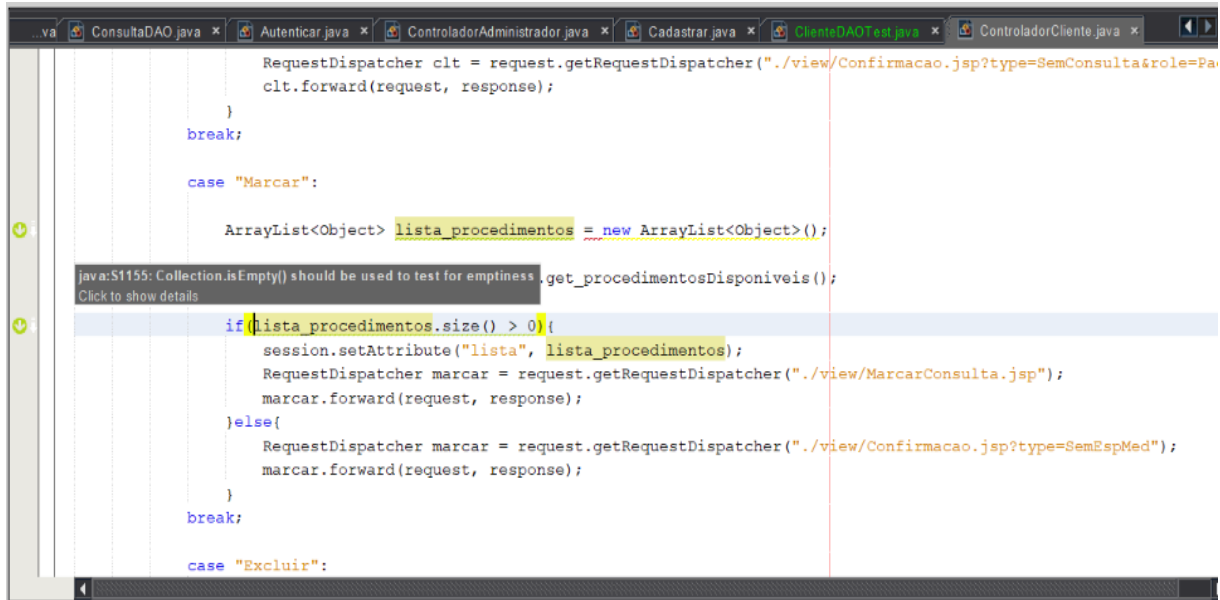
- **Prioridade:** Minor (Code Smell).

- **Recomendação:** Usar a classe StringBuilder para fazer a manipulação e concatenação dessas Strings, já que a classe faz o uso de Strings dinâmicas.

5.4.18 Problema 18:

- **Descrição:** O erro “S1155” indica que a verificação se o array está vazio foi feita de forma não otimizada.

- **Localização:** A localização do erro é a classe ControladorCliente.java.



```
RequestDispatcher clt = request.getRequestDispatcher("../view/Confirmacao.jsp?type=SemConsulta&role=Pa");
clt.forward(request, response);
}
break;

case "Marcar":

    ArrayList<Object> lista_procedimentos = new ArrayList<Object>();

    java:S1155: Collection.isEmpty() should be used to test for emptiness. get_procedimentosDisponiveis();
    Click to show details

    if (lista_procedimentos.size() > 0) {
        session.setAttribute("lista", lista_procedimentos);
        RequestDispatcher marcar = request.getRequestDispatcher("../view/MarcarConsulta.jsp");
        marcar.forward(request, response);
    } else {
        RequestDispatcher marcar = request.getRequestDispatcher("../view/Confirmacao.jsp?type=SemEspMed");
        marcar.forward(request, response);
    }
    break;

case "Excluir":
```

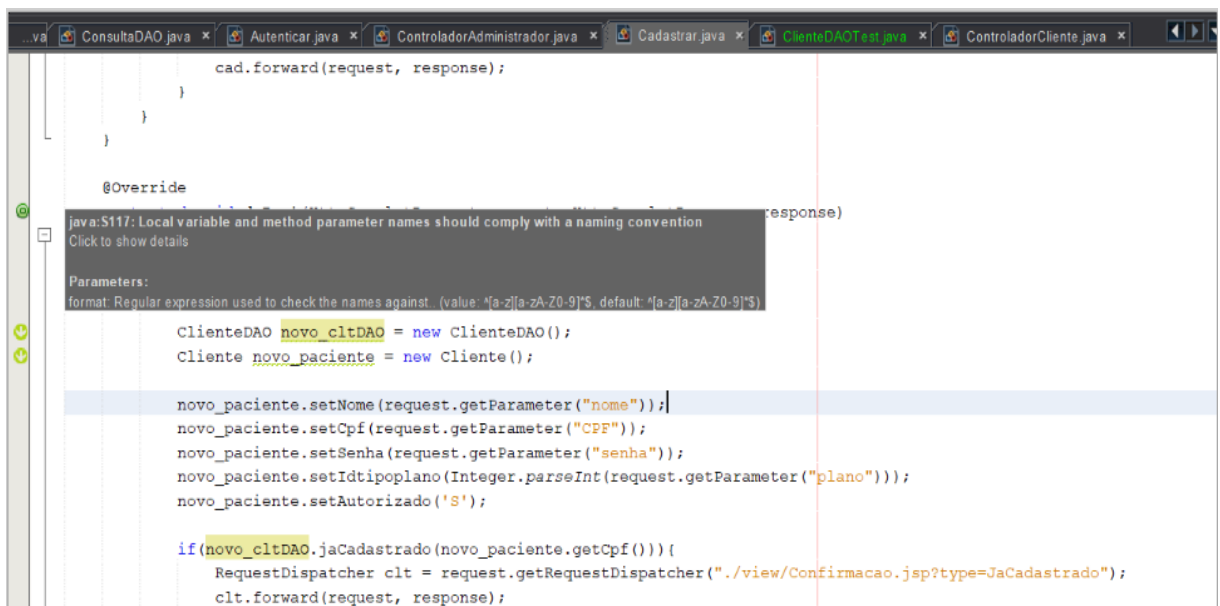
- **Prioridade:** Minor (Code Smell).

- **Recomendação:** Substituir a condição que testa o tamanho do array usando o método isEmpty(). Exemplo: lista_procedimentos.isEmpty().

5.4.19 Problema 19:

- **Descrição:** O erro “S117” acusa o uso de nomes para as variáveis e parâmetros de métodos que não seguem a convenção de nomenclatura.

- **Localização:** Esse problema apresenta 133 aparições no código-fonte. Abaixo é mostrado um caso na classe Cadastrar.java.



```
cad.forward(request, response);
}
}

@Override
java:S117: Local variable and method parameter names should comply with a naming convention
Click to show details
Parameters:
format: Regular expression used to check the names against (value: '[a-zA-Z0-9]*', default: '[a-z][a-zA-Z0-9]*')

    ClienteDAO novo_clienteDAO = new ClienteDAO();
    Cliente novo_paciente = new Cliente();

    novo_paciente.setNome(request.getParameter("nome"));
    novo_paciente.setCpf(request.getParameter("CPF"));
    novo_paciente.setSenha(request.getParameter("senha"));
    novo_paciente.setIdtipoPlano(Integer.parseInt(request.getParameter("plano")));
    novo_paciente.setAutorizado('S');

    if (novo_clienteDAO.jaCadastrado(novo_paciente.getCpf())) {
        RequestDispatcher clt = request.getRequestDispatcher("../view/Confirmacao.jsp?type=JaCadastrado");
        clt.forward(request, response);
    }
```

- **Prioridade:** Minor (Code Smell).

- **Recomendação:** Adotar um nome que seja complacente com o padrão, como novoCltDAO.

5.4.20 Problema 20:

- **Descrição:** O erro “S1197” demonstra que o indicador da declaração de um array “[]” não está no local apropriado.

- **Localização:** O erro acontece na classe ControladorCliente.java.



```
MedicoDAO medicoDAO = new MedicoDAO();
Consulta consulta = new Consulta();
ConsultaDAO consultaDAO = new ConsultaDAO();

consulta.setData(request.getParameter("data") + " " + request.getParameter("hora"));
consulta.setDescricao("Sem descricao");
consulta.setRealizada('N');
consulta.setIdmedico(Integer.valueOf(request.getParameter("id_med")));

String datahora[] = consulta.getData().split(" ");

ArrayList<Integer> colisoas = new ArrayList<Integer>();
colisoas = medicoDAO.medico_disponivel(consulta.getIdmedico(), datahora[0]);

switch(request.getParameter("acao")){
    case "Enviar":

        if(colisoas.size() < 2){
            consultaDAO.create_consulta(consulta);
            RequestDispatcher voltar = request.getRequestDispatcher("../view/Confirmacao.jsp?type=Marcado");
            voltar.forward(request, response);
        }
    }
}
```

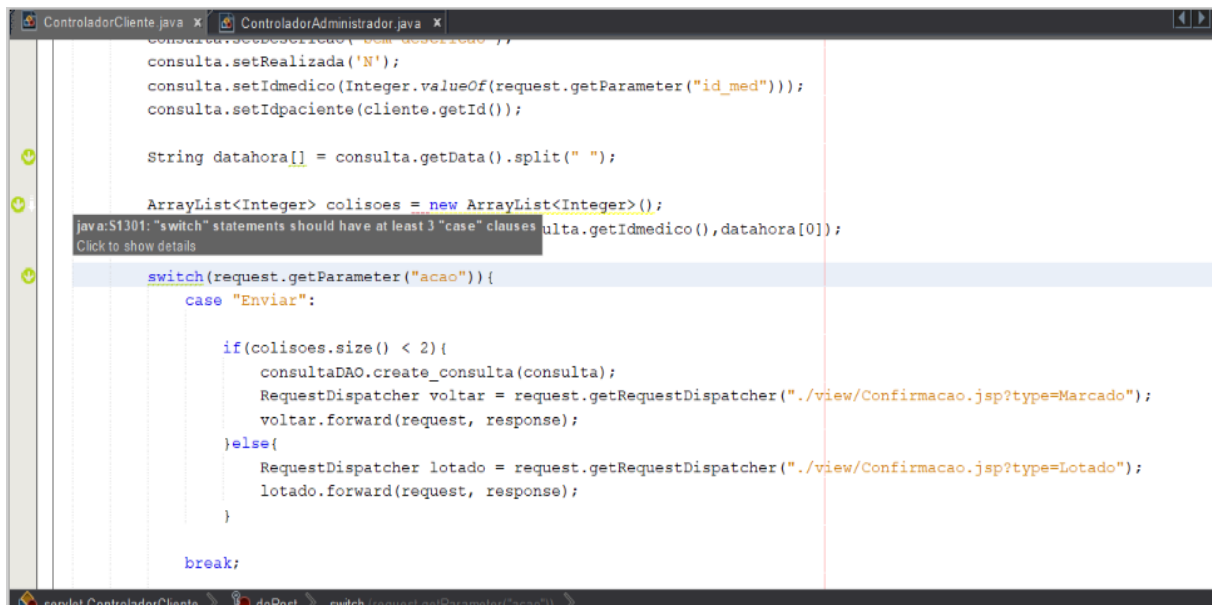
- **Prioridade:** Minor (Code Smell).

- **Recomendação:** Declarar o designador “[]” de array junto com o tipo e não com a variável para melhorar a legibilidade.

5.4.21 Problema 21:

- **Descrição:** O erro “S1301” indica o uso não apropriado da estrutura “switch”.

- **Localização:** 8 ocorrências aparecem e estão distribuídas entre as classes ControladorCiente.java e ControladorAdministrador.java.



```
Consulta.setDescricao("Sem descricao");
consulta.setRealizada('N');
consulta.setIdmedico(Integer.valueOf(request.getParameter("id_med")));
consulta.setIdpaciente(cliente.getId());

String datahora[] = consulta.getData().split(" ");

ArrayList<Integer> colisoas = new ArrayList<Integer>();
colisoas = medicoDAO.medico_disponivel(consulta.getIdmedico(), datahora[0]);

switch(request.getParameter("acao")){
    case "Enviar":

        if(colisoas.size() < 2){
            consultaDAO.create_consulta(consulta);
            RequestDispatcher voltar = request.getRequestDispatcher("../view/Confirmacao.jsp?type=Marcado");
            voltar.forward(request, response);
        }
    }
    break;
}
```

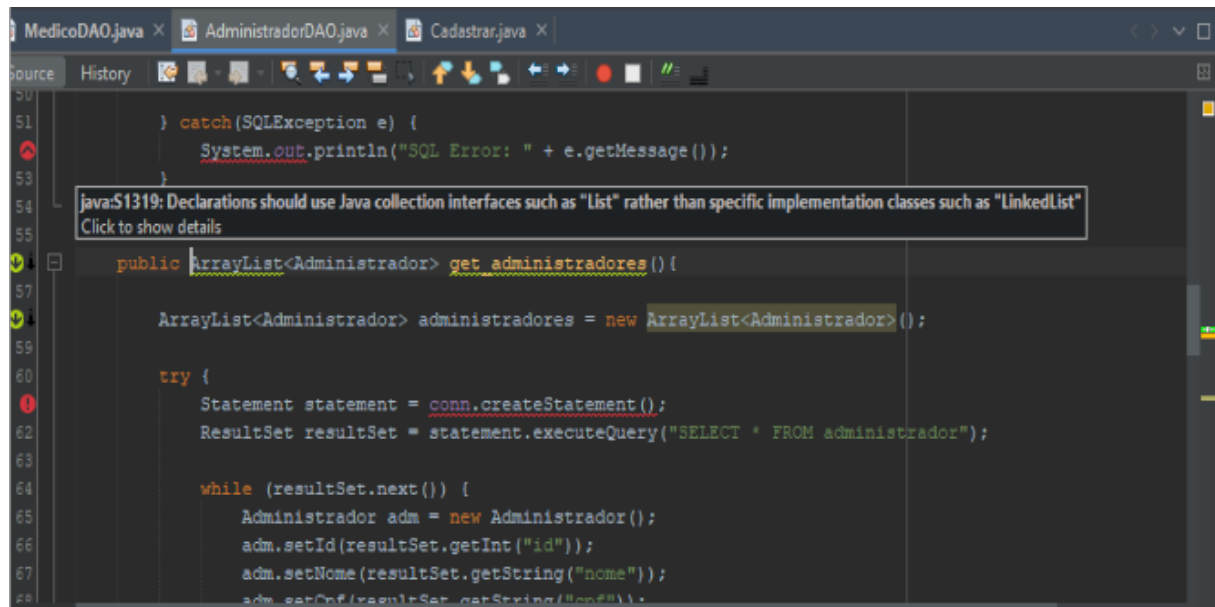
- **Prioridade:** Minor (Code Smell).

- **Recomendação:** Para facilitar a leitura do código, é indicado o uso da estrutura “if” se não existem mais de 3 casos condicionais.

5.4.22 Problema 22:

- **Descrição:** O erro “S1319” acusa a declaração de implementações específicas de lista, como o ArrayList<Administrador>.

- **Localização:** Existem 20 ocorrências desse erro. Abaixo o mesmo é ilustrado na classe AdministradorDAO.java.



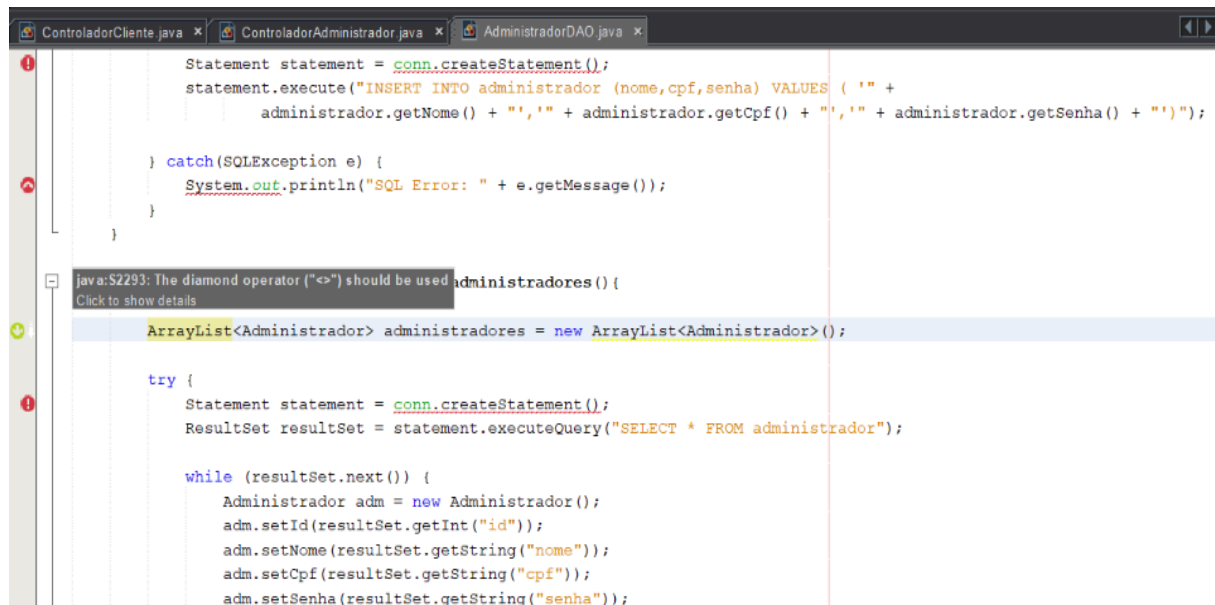
- **Prioridade:** Minor (Code Smell).

- **Recomendação:** Usar uma declaração mais genérica de lista, como List<Administrador>.

5.4.23 Problema 23:

- **Descrição:** O erro “S2293” apresenta o uso desnecessário da declaração do tipo do array no construtor.

- **Localização:** Existem 60 ocorrências no código-fonte. Abaixo o exemplo é na classe AdministradorDAO.java.



- **Prioridade:** Minor (Code Smell).

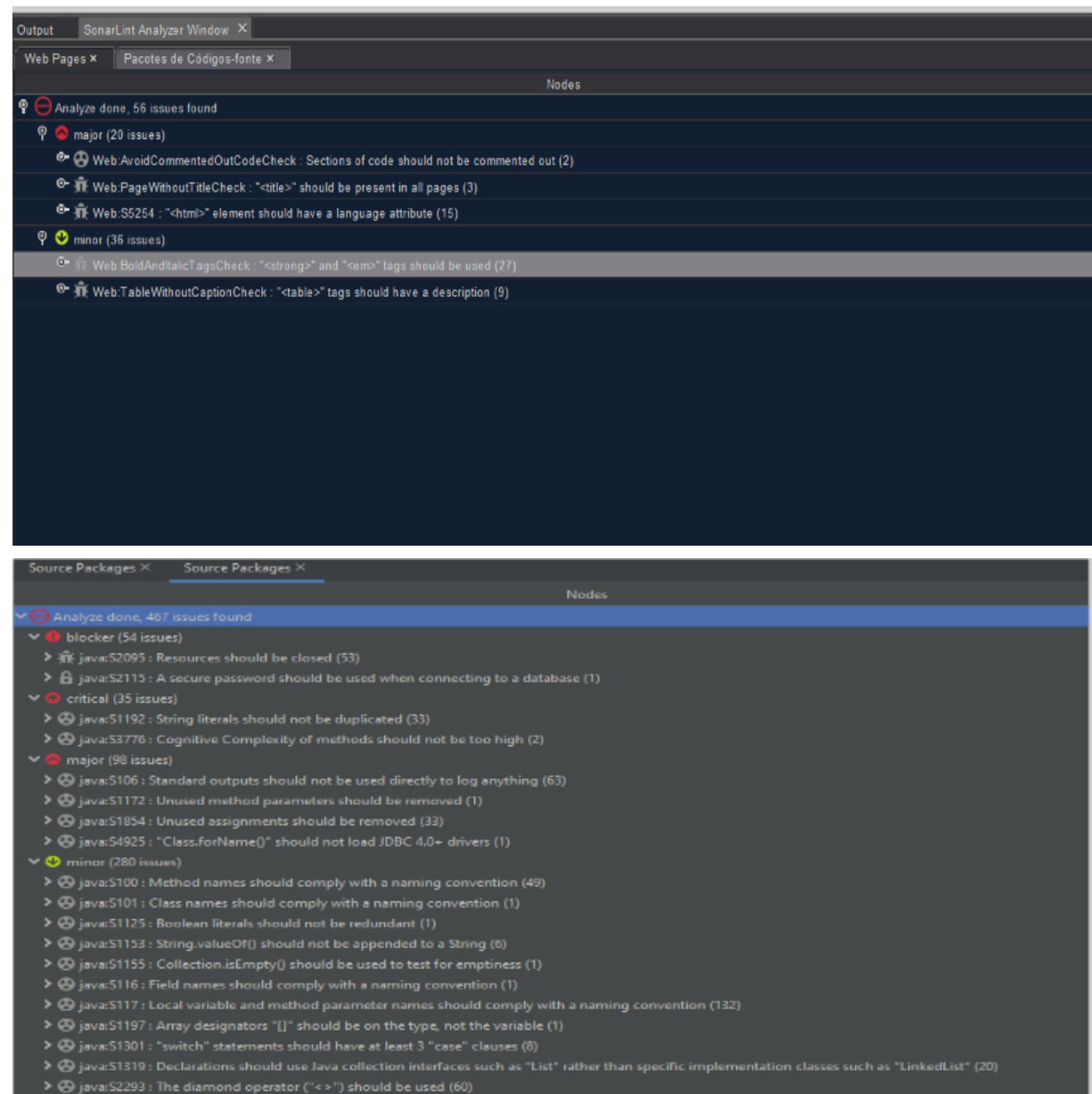
- **Recomendação:** Deixar a declaração de tipo do array apenas no objeto que vai instanciar a classe. A partir do Java 7, o compilador já infere o tipo do construtor sem que seja necessário declará-lo.

5.5. Conclusões:

Durante o processo de inspeção de código, com a ajuda do Sonar Lint, pudemos constatar a presença de diversos tipos de erro e diferentes níveis de severidade, mas que a maioria consistia em “code smells”. A partir disso, foi possível tomar conhecimento a respeito dos erros, sobre os diferentes tipos e bem como ao grau de seriedade. Dessa forma, os integrantes do grupo se uniram de forma a estudar e solucionar os erros encontrados no projeto.

5.6. Anexos:

Lista geral com os erros detalhes no item 5.4:



6. Projeto de casos de testes unitários.

Estão localizados em formato de testes unitários no repositório