SIMULT - Sistemas de Multas

O SIMULT-PDF é um sistema de monitoramento/cadastramento de multas, desenvolvido inicialmente na linguagem Portugol, com o intuito de aprimorar a lógica de programação. O sistema elaborado na disciplina de Laboratório de Algoritmos e Estrutura de Dados II é simples e consiste na construção de um menu principal para auxiliar nas tomadas de decisões. Embasado nisso, o SIMULT-PDF 2.0 é resultado do aprimoramento do sistema anterior, visando mais especificamente a parte do administrador. Dessa forma, são acrescidas novas funcionalidades - além da básica geração de multas e relatórios de trânsito - ao novo programa implementado na linguagem de programação Java.

Especificações:

Entidades

class Administrador - Todo administrador é formado por 5 campos e é identificado por um id.

- *id*
  - Tipo: inteiro;
  - Encapsulamento: private;
  - Observação: O campo é gerado automaticamente quando ocorre a inserção no banco de dados (o campo no banco de dados está configurado como SERIAL).
  - Restrições:
    - Diferentes de 0;

- *nome*
  - Tipo: String;
  - Encapsulamento: private;
  - Restrições:
    - Não pode ser vazio;
    - O campo deve ter no máximo 80 caracteres;

- *email*
  - Tipo: String;
  - Encapsulamento: private;
  - Restrições:
    - Não pode ser vazio;
    - O campo deve ter no máximo 80 caracteres;

- *login*
  - Tipo: String;
  - Encapsulamento: private;
  - Restrições:

        - Não pode ser vazio;
        - O campo deve ter no máximo 80 caracteres;

- *senha*
    - Tipo: string;
    - Encapsulamento: private;
    - Observação: O campo deve ter no mínimo 8 caracteres.

---

class Veiculo - Todo veículo é identificado pelo valor de sua placa e número do renavam e é constituído por 10 campos:

- *proprietario*
    - Tipo: String
    - Encapsulamento: private;
    - Observação: O campo representa o nome do proprietário;
    - Restrições:
        - Não pode ser vazio;
        - O campo deve ter no máximo 80 caracteres;

- *placa*
    - Tipo: String
    - Encapsulamento: private;
    - Restrições:
        - O campo deve ter obrigatoriamente 7 caracteres;
        - Seguir o formato LLLNLNN (L - letras e N - números).

- *renavam*
    - Tipo: String
    - Encapsulamento: private;
    - Restrições:
        - O campo deve ter obrigatoriamente 9 ou 11 caracteres;
        - Formado por apenas números.

- *chassi*
    - Tipo: String
    - Encapsulamento: private;
    - Restrições:
        - O campo deve ter obrigatoriamente 17 caracteres alfanuméricos;

- *estadoRegistro*
    - Tipo: String
    - Encapsulamento: private;
    - Restrições:
        - Não pode ser vazio;
        - O campo deve ter no máximo 40 caracteres;

- *municipioRegistro*
    - Tipo: String
    - Encapsulamento: private;
    - Restrições:
        - Não pode ser vazio;
        - O campo deve ter no máximo 50 caracteres;

- *cor*
    - Tipo: String
    - Encapsulamento: private;
    - Restrições:
        - Não pode ser vazio;
        - O campo deve ter no máximo 30 caracteres;

- *marca*
    - Tipo: String
    - Encapsulamento: private;
    - Restrições:
        - Não pode ser vazio;
        - O campo deve ter no máximo 30 caracteres;

- *modelo*
    - Tipo: String
    - Encapsulamento: private;
    - Restrições:
        - Não pode ser vazio;
        - O campo deve ter no máximo 30 caracteres;

- *especie*
    - Tipo: String
    - Encapsulamento: private;
    - Restrições:
        - Não pode ser vazio;
        - O campo deve ter no máximo 20 caracteres;

---

class Multa - Toda multa é constituída por 5 atributos e é identificado pelo seu código.

- *codigo*
    - Tipo: String
    - Encapsulamento: private;
    - Restrições:
        - O campo deve ter obrigatoriamente 6 caracteres;
        - Seu formato deve seguir NNN-NN (N - número).

- *descricao*
    - Tipo: String

    - Encapsulamento: private;
    - Restrições:
        - Não pode ser vazio;
        - O campo deve ter no máximo 256 caracteres.

- *classificacao*
    - Tipo: String
    - Encapsulamento: private;
    - Restrições:
        - Não pode ser vazio;
        - O campo deve ter no máximo 15 caracteres;
        - O valor do campo deve ser LEVE, MÉDIA, GRAVE ou GRAVÍSSIMA.

- *pontos*
    - Tipo: int
    - Encapsulamento: private;
    - Restrições:
        - Não pode conter letras;

- *valor*
    - Tipo: double
    - Encapsulamento: private;
    - Restrições:
        - Não pode conter letras;

---

class AutuacaoTransito - Identificado por um id, toda autuação é caracterizada por 8 atributos.

- *id*
    - Tipo: inteiro;
    - Encapsulamento: private;
    - Observação: O campo é gerado automaticamente quando ocorre a inserção no banco de dados (o campo no banco de dados está configurado como SERIAL).

- *autor*
    - Tipo: Autor;
    - Encapsulamento: private;
    - Observação: O campo recebe como tipo um objeto criado especificado anteriormente.
    - Restrições:
        - O campo não deve ser nulo.

- *veiculo*
    - Tipo: Veiculo;
    - Encapsulamento: private;

    - Observação: O campo recebe como tipo um objeto criado especificado anteriormente.
    - Restrições:
        - O campo não deve ser nulo.

- *multa*
    - Tipo: Multa;
    - Encapsulamento: private;
    - Observação: O campo recebe como tipo um objeto criado especificado anteriormente.
    - Restrições:
        - O campo não deve ser nulo.

- *estado*
    - Tipo: String
    - Encapsulamento: private;
    - Restrições:
        - Não pode ser vazio;
        - O campo deve ter no máximo 50 caracteres;

- *municipio*
    - Tipo: String
    - Encapsulamento: private;
    - Restrições:
        - Não pode ser vazio;
        - O campo deve ter no máximo 80 caracteres;

- *dataHora*
    - Tipo: String
    - Encapsulamento: private;
    - Observação: O campo é gerado automaticamente no momento da criação do objeto autuação.

- *vencimento*
    - Tipo: String
    - Encapsulamento: private;
    - Observação: O campo é gerado automaticamente no momento da criação do objeto autuação com base no valor de dataHora.

---

class Relatorio - Os relatórios são formados por 4 atributos e identificador por um id.

- *id*
    - Tipo: inteiro;
    - Encapsulamento: private;

    - Observação: O campo é gerado automaticamente quando ocorre a inserção no banco de dados (o campo no banco de dados está configurado como SERIAL).

- *autor*
    - Tipo: Autor;
    - Encapsulamento: private;
    - Observação: O campo recebe como tipo um objeto criado especificado anteriormente.
    - Restrições:
        - O campo não deve ser nulo.

- *autuacoes*
    - Tipo: LinkedList<Autuacao>
    - Encapsulamento: private;
    - Observação: O campo recebe como tipo uma lista de objetos do tipo AutuacaoTransito, classe especificada anteriormente.

- *dataHora*
    - Tipo: String
    - Encapsulamento: private;
    - Observação: O campo é gerado automaticamente no momento da criação do objeto relatório.

BD

class Conexao - Criada seguindo o padrão criacional Singleton, essa classe é responsável por prover um único ponto global de acesso ao banco de dados.

- *conexcao*
    - Tipo: Connection
    - Encapsulamento: private
    - Observação: Connection exatamente um tipo, mas sim uma interface do java.sql e é utilizada para estabelecer uma conexão com um banco de dados e fornece métodos para executar consultas, atualizações e outras operações relacionadas a banco de dados.

- *instance*
    - Tipo: Conexao
    - Encapsulamento: static private;
    - Observação: Utilizada para não necessitar da criação de mais de uma instância para execução do sistema.

- *getInstance()*
    - Retorno: Conexao

- Funcionamento: O método verifica se o atributo instance é nulo, caso, sim, atribui ao atributo uma instância do objeto Conexao e retorná-lo. Caso contrário, apenas retorna o atributo instance.

- *closeConnection()*
  - Retorno: boolean
  - Funcionamento: O método verifica se a interface de conexão (atributo conexcao) é diferente de nula, se sim, encerra a conexão e retorna **true**, se não, apenas retorna **true**, já que a conexão já havia sido fechada.

DAO

class AdministradorDAO

- insereAdministrador (Administrador autor)
  - Retorno: boolean
  - Funcionamento: O método recebe um administrador como parâmetro para ser inserido no banco de dados. É feito o uso *PreparedStatement* para executar instruções SQL, se a execução for bem sucedida, retorna **true**, caso contrário, retorna **false**.
  - Observações: - - -

- buscaAdministrador (int id)
  - Retorno: Administrador
  - Funcionamento: O método recebe como parâmetro o id do administrador, a fim de realizar a busca de um administrador com essa informação. Isso é possível com o uso de *PreparedStatement*. Caso a busca encontre um administrador, o retorna, caso não, retorna null.
  - Observações: - - -

- buscaAdministrador (String login)
  - Retorno: Administrador
  - Funcionamento: O método recebe como parâmetro o login do administrador, a fim de realizar a busca de um administrador com essa informação. Isso é possível com o uso de *PreparedStatement*. Caso a busca encontre um administrador, o retorna, caso não, retorna null.
  - Observações: - - -

---

class VeiculoDAO

- buscaVeiculo(String placa, String renavam)
  - Retorno: Veiculo

- Funcionamento: O método recebe como parâmetro a placa e renavam do veículo, a fim de realizar a busca do veículo com esses identificadores. Isso é possível com o uso de *PreparedStatement*. Caso a busca encontre um veículo, o retorna, caso não, retorna null.
- Observações: - - -

---

class MultaDAO

- buscaMulta(String codigo)
  - Retorno: Multa
  - Funcionamento: O método recebe como parâmetro o código da multa, a fim de realizar a busca das multas com o mesmo código. Isso é possível com o uso de *PreparedStatement*. Caso a busca encontre uma multa, o retorna, caso não, retorna null.
  - Observações: - - -

- buscaMultaGeral()
  - Retorno: Array de multas
  - Funcionamento: O método realiza a busca de todas as multas cadastradas. Isso é possível com o uso de *PreparedStatement*. Caso a busca encontre uma multa, adiciona em um array,ao final retorna esse array;
  - Observações: - - -

- buscaMultaClassificacao(String classificacao)
  - Retorno: Array de multas
  - Funcionamento: O método recebe como parâmetro a classificação da multa, a fim de realizar a busca das multas relacionadas. Isso é possível com o uso de *PreparedStatement*. Caso a busca encontre uma multa, correspondente adiciona em um array,ao final retorna esse array;
  - Observações: - - -

---

class AutuacaoTransitoDAO

- insereAutuacao(AutuacaoTransito autuacao)
  - Retorno: boolean
  - Funcionamento: O método recebe uma autuação como parâmetro para ser inserido no banco de dados. É feito o uso *PreparedStatement* para executar instruções SQL, se a execução for bem sucedida, retorna **true**, caso contrário, retorna **false**.
  - Observações: - - -

- buscaAutuacao(int id)

    - Retorno: Autuacao
    - Funcionamento: O método recebe como parâmetro o id do administrador e a data e hora de criação da autuação, a fim de realizar a busca de seu identificador. Isso é possível com o uso de *PreparedStatement*. Caso a busca encontre uma autuação correspondente, retorna seu id, senão retorna 0;
    - Observações: - - -

- buscaIdAutuacao(int idAdministrador, String dataHora)
    - Retorno: int
    - Funcionamento: O método recebe como parâmetro o id do administrador e a data e hora de criação da autuação, a fim de realizar a busca de seu identificador. Isso é possível com o uso de *PreparedStatement*. Caso a busca encontre uma autuação correspondente, retorna seu id, senão retorna 0;
    - Observações: - - -

- removeAutuacao(int id)
    - Retorno: boolean
    - Funcionamento: O método recebe como parâmetro o id da atuação para ser removida da tabela autuacaotransito. Para isso faz uso PreparedStatement para executar instruções SQL, se a execução for bem sucedida, retorna true, caso contrário, retorna false.
    - Observações:

---

class RelatorioDAO

- *insereRelatorio(Relatorio relatorio)*
    - Retorno: boolean
    - Funcionamento: O método recebe um relatório como parâmetro para ser inserido no banco de dados. É feito o uso *PreparedStatement* para executar instruções SQL, se a execução for bem sucedida, retorna **true**, caso contrário, retorna **false**.
    - Observações:
        - Caso o relatório esteja acompanhado de autuações, é feita a chamada da função de inserir o relacionamento de relatório e autuação para todas as autuações;
        - O id do relatório é gerado apenas quando a inserção no banco de dados devido o seu tipo ser SERIAL.

- *buscaIdRelatorio(int idAdministrador, String dataHora)*
    - Retorno: int
    - Funcionamento: O método recebe como parâmetro o id do administrador e a data e hora de criação do relatório, a fim de realizar a busca de seu identificador. Isso é possível com o uso de *PreparedStatement*. Caso a busca encontre um relatório correspondente, retorna seu id, senão retorna 0;

- Observações: - - -

---

class RelatorioAutuacaoDAO

- *insereRelatorioDeAutuacao(int id_relatorio, int id_autuacao)*
  - Retorno: boolean
  - Funcionamento: O método recebe como parâmetro o id do relatório e da autuação para serem inseridos na tabela *relatorio_autuacao*. Para isso faz uso *PreparedStatement* para executar instruções SQL, se a execução for bem sucedida, retorna **true**, caso contrário, retorna **false**.
  - Observações: - - -

- *buscaAutuacoesRelatorio(int idRelatorio)*
  - Retorno: Lista de inteiros
  - Funcionamento: O método recebe como parâmetro o id de um relatório e deve retornar todas as ocorre daquele id na tabela. Para isso faz uso *PreparedStatement* para executar instruções SQL, se a busca for bem sucedida, adiciona o id da autuação numa LinkedList e após encerrar a busca a retorna. Caso contrário, retorna null.
  - Observações: - - -

<u>Controllers</u>

class VeiculoController

- *buscarVeiculo()*
  - Parâmetros: - - -
  - Entradas: String placa, String renavam
  - Retorno: Veiculo
  - Funcionamento: Quando a função é chamada é pedido ao usuário para que ele digite a placa e renavam do veículo que ele deseja buscar. Após a inserção válida dos campos (seguindo as especificações) é chamado o método *buscaVeículo()* que realiza a busca do veículo no banco de dados e o retorna caso encontre.
  - Observações: - - -

- *verDadosVeiculo()*
  - Parâmetros: - - -
  - Entradas: - - -
  - Retorno: void
  - Funcionamento: É feita a atribuição a uma variável do tipo Veiculo com a chamada da função *buscarVeiculo()* que deve retornar um veículo. Com a atribuição da variável é feito a sua impressão utilizando o método *toString()* da entidade.
  - Observações: - - -

---

class MultaController

- *buscarMulta()*
    - Parâmetros: - - -
    - Entradas: int id
    - Retorno: Multa
    - Funcionamento: É requisitado ao usuário para que ele digite os números do código da multa que deseja buscar. Após a validação do código digitado é feita a chamada do método *buscaMulta()* passando o código como parâmetro e ela deve retornar a multa caso esteja presente no banco de dados.
    - Observações:
        - Quando chamada é necessário inserir um número entre 10000 e 99999, fora desse intervalo, é requisitado novamente o dado ao usuário.
        - O código a ser digitado no campo deve conter apenas os números, sem a inclusão do hífen como padrão do sistema.
        - A formatação é feita automaticamente pelo sistema para facilitar a sua validação (com base nos requisitos).

- *verDadosMulta()*
    - Parâmetros: - - -
    - Entradas: - - -
    - Retorno: void
    - Funcionamento: Dentro do método *buscarMulta()* é feita a chamada do método que irá buscar a multa para ser apresentada. Caso encontre a multa, a apresentação é feita utilizando o método *toString()* da entidade.
    - Observações: - - -

- *listaMultas()*
    - Parâmetro: String classificação
    - Entradas: - - -
    - Retorno: void
    - Funcionamento: É feita a atribuição a uma lista de multas utilizando o método *buscaMultaClassificacao()* que recebe a classificação como parâmetro também e deve retornar uma lista de todas as multas com a respectiva classificação. Caso a lista não seja vazia, é feita a impressão utilizando um *foreach* e o método *toString()* da entidade.
    - Observações: - - -

---

class AutuacaoTransitoController

- *cadastrarAutuacao()*
    - Parâmetro: Administrador autor não nulo

- Entradas: String placa, String renavam, int opcao, int id, String estado, String municipio
- Retorno: boolean:
- Funcionamento: O método se resume a um preenchimento de formulário para realizar o cadastro de uma autuação, onde é requisitado a placa e o renavam para identificar o veículo, a opção definirá a classificação da multa para poder listá-la e assim o usuário digitar o código da multa, além do estado e município onde a autuação ocorreu. Após a inserção de todos os dados é criado um objeto de autuação com as informações e passada como parâmetro da *insereAutuacao()*, que é responsável por inseri-la no banco de dados e retorna true.
- Observações:
  - Todos os campos tem a sua validação:
    - Placa - segue as especificações do atributo;
    - Renavam - segue as especificações do atributo;
    - Opcao - recebe um número inteiro de 1 a 4 para identificar a classificação da multa;
    - Id - um campo que recebe apenas os número do código da multa para ser buscada, o sistema trata da formatação;
    - Estado - é um campo que não pode ser vazio e possuir no máximo 50 caracteres;
    - Municipio - é um campo que não pode ser vazio e possuir no máximo 80 caracteres;
  - Após a inserção no banco de dados, a autuação é armazenada também em uma variável do autor, que representará as autuações cadastradas por ele naquela seção.

- *removerAutuacao()*
  - Parâmetros: Administrador autor não nulo
  - Entradas: int id, int opcao
  - Retorno: boolean
  - Funcionamento: O método recebe do usuário um id da autuação que deseja remover, realizar a busca a partir do método *buscaAutuacao()*, caso exista, exibe a autuação e pede a confirmação do usuário para realizar a remoção. Caso confirme, é realizada a remoção do buffer de ações do autor assim como do banco de dados, caso não, pede novamente o id da autuação. Se a remoção não for bem sucedida, retorna false.
  - Observações:
    - O campo id de autuação segue as especificações do atributo;
    - Opcao - recebe um número inteiro de 1 a 2, 1 para confirmar ou 0 para negar;

class RelatorioController

- *criarRelatorio()*
    - Parâmetros: Administrador autor não nulo
    - Entradas: - - -
    - Retorno: boolean
    - Funcionamento: Logo de início é criado um objeto do tipo relatório a partir do parâmetro passado, após a criação esse relatório é utilizado como parâmetro n método *insereRelatorio()*, que será responsável por cadastrar no banco de dados esse relatório. Se a inserção for bem sucedida, o relatório é impresso para o usuário e retorna true.
    - Observações: - - -

---

class AdministradorController

- *logar()*
    - Parâmetro: - - -
    - Entradas: String login, String senha
    - Retorno: Administrador
    - Funcionamento: O sistema pede ao usuário para que ele preencha os campos de login e senha contendo as informações do usuário a ser logado. É realizado a busca se o login está mesmo cadastrado no sistema, caso sim, criptografa a senha passada e compara com a senha criptografa presente no banco de dados, se as informações baterem, significa a comprovação do usuário e retorna chama o método *buscaAdministrador()* que irá retornar o administrador.
    - Observações:
        - O preenchimento dos campos login e senha não segue exatamente a especificação da entidade, pois no momento de logar o tamanho do campo é indefinido, mas o seu tipo se mantém.
        - O usuário só tem o direito a 3 tentativas por sessão, caso no momento de logar o usuário erre suas credenciais 3 vezes, o sistema encerra automaticamente.

- *cadastrarAdministrador()*
    - Parâmetro: Administrador administrador não nulo
    - Entradas: String nome, String email, String login, String senha
    - Retorno: boolean
    - Funcionamento: Primeiro verifica se o administrador passado como parâmetro tem permissão para executar o cadastramento, caso não, encerra a função e apresenta o motivo, caso sim, o usuário é requisitado para inserir os dados válidos para o nome, email, login e senha. Após isso, a senha é criptografada para ser armazenada no banco de dados e chama a função *insereAdministrador()* que irá inserir no banco de dados o administrador passado como parâmetro (fruto dos dados requisitados), retorna true se inserção ocorrer bem, ou false caso não.
    - Observações:

        - Todos os campos tem sua padrão forma de validação que segue o que foi passado na especificação daquele atributo na entidade administrador.
        - Por questões de segurança, a senha de qualquer usuário deve ser criptografada antes de ser inserida no banco de dados.

- *vincularMulta()*
    - Parâmetro: Administrador administrador não nulo
    - Entrada: - - -
    - Retorno: boolean
    - Funcionamento: O método se resume em apenas retornar o status da execução *removerAutuacao()* passando como parâmetro o administrador. Se a remoção da autuação ocorrer conforme esperado, retorna true, senão, false.
    - Observações: - - -

- *desvincularMulta()*
    - Parâmetro: Administrador administrador não nulo
    - Entrada: - - -
    - Retorno: boolean
    - Funcionamento: O método se resume em apenas retornar o status da execução *removerAutuacao()* passando como parâmetro o administrador. Se a remoção da autuação ocorrer conforme esperado, retorna true, senão, false.
    - Observações: - - -

- *consultarVeiculo()*
    - Parâmetro: - - -
    - Entrada: - - -
    - Retorno: void
    - Funcionamento: O método se resume em apenas executar o método *verDadosVeiculo()*. Já que o método se responsabiliza em apresentar o resultado da consulta ao usuário.
    - Observações: - - -

- *consultarMulta()*
    - Parâmetro: - - -
    - Entrada: - - -
    - Retorno: void
    - Funcionamento: O método se resume em apenas executar o método *verDadosMulta()*. Já que o método se responsabiliza em apresentar o resultado da consulta ao usuário.
    - Observações: - - -

- *gerarRelatorio()*
    - Parâmetro: Administrador administrador não nulo
    - Entrada: - - -
    - Retorno: boolean

- Funcionamento: O método se resume em apenas retornar o status da execução *criarRelatorio()* passando como parâmetro o administrador. Se a geração do relatório ocorrer conforme esperado, retorna true, senão, false.
- Observações: - - -

Testes de Funcionais

Testes de Unidade:

Em um contexto orientado a objetos, os testes de unidade geralmente se concentram na avaliação dos métodos individuais ou até mesmo das classes em si, tomando como base a sua definição. Neste sistema específico, as unidades de teste serão os métodos das classes, devido à sua granularidade.

A execução dos testes de unidade ocorre de duas maneiras: manualmente, para métodos que requerem inserção de dados pelo usuário, e automaticamente, com o auxílio de ferramentas.

Roteiro dos testes de unidades:

Como estratégia para execução dos testes e para que os mesmos ocorram de maneira confiável, os testes que devem ser executados primeiro são aqueles que não necessitam de outros métodos para funcionarem, ou seja, que não tenham como pré-requisitos outros métodos.

Devido estarmos tratando de testes de unidades e por enquanto não de integração, alguns métodos devem ser modificados (sem alterar muito seu comportamento) para não necessitar de outros métodos, a fim de testar se o mesmo está fazendo o que é esperado e se possível livre de erros.

Se tratando de testes funcionais, a execução dos testes e elaboração dos casos de testes seguirá algumas abordagens, por exemplo, classe por equivalência e análise do valor limite, a escolha será aquela que melhor se encaixa para aquele campo.

A automatização de testes se dará nos métodos que não necessitam da inserção manual de dados do usuário e com o auxílio da ferramenta JUnit, que em nosso sistema são aquelas relacionadas a comunicação e conexão com banco de dados, conhecidos como DAO, Data Access Object. Já os manuais, será nos métodos que necessitam da inserção de dados dos usuários, que no caso será o restante dos métodos presente no sistema.

Com os resultados obtidos dos testes, concluímos que o sistema está sendo testado e a partir deles poderemos ter a satisfação de cobertura e possibilidade de corrigir as falhas encontradas nos testes que não foram encontradas no momento do desenvolvimento.

Realização dos testes automatizados:

- **ConnectionTest** - Irá testar os aspectos relacionados a conexão com o banco de dados que corresponde a classe ***Conexao***, vendo se está funcionando da maneira correta, especificamente sua inicialização e encerramento.

➔ public void testGetInstance() - Está sendo feita uma requisição de conexão com o banco de dados, essa requisição retorna uma Connection ser bem sucedida, sendo Null para algo inesperado. Utilizamos assertNotNull() para verificar se foi bem sucedida.

- ◆ Os testes ocorreram perfeitamente e o retorno foi o esperado, a conexão.

➔ public void testCloseConnection() - É chamado o método closeConnection() da classe em questão e o mesmo fecha a conexão com o banco de dados. Se o encerramento for bem-sucedido, o método retorna true e false para um comportamento inesperado, usamos assertTrue() para verificar.

- ◆ Os testes ocorreram perfeitamente e o retorno foi o esperado. (true)

- **MultaDAOTest**

➔ public void testSearchMulta() - O método a ser testado recebe um código como parâmetro, código que servirá como comparativo para realizar a busca dentro do banco de dados. Caso encontre a multa correspondente, retorna ela, senão null.

- ◆ Os testes ocorreram da maneira esperada, tanto no caso onde deveria retornar uma multa quanto null.

➔ public void testSearchClassifyMulta() - É passado um parâmetro que representa a classificação da multa que deseja buscar. Caso encontre multas com essa classificação, retorna todas, senão null.

- ◆ Foi realizado com sucesso o teste na busca de multas com todas as classificações possíveis (leve, média, grave e gravíssima) e as possíveis falhas. Foi usado assertNull e assertNotNull para validar.
- ◆ Com o teste foi possível identificar o parâmetro é Case Sensitive, o que pode acontecer de digitar a classificação em maiúsculo ou minúsculo pode retornar certo, ou errado dependendo de como os dados estão salvos no banco de dados. Então agora, há um padrão, tudo em maiúsculo para a classificação.

➔ public void testSearchAllMulta() - É chamado o método quando há a necessidade de retornar todas as multas cadastradas, então se há multas cadastradas o método deve retorná-las, caso não será algo inesperado.

- Como há multas cadastradas no banco de dados, o retorno da chamada foi um conjunto de multas, já que o retorno foi diferente de null. Usamos o assertNotNull para validar.

- **VeiculoDAOTest**

  ➔ public void testSearchVeiculo() - A teste leva em consideração os veículos cadastrados no banco de dados, então caso a placa e renavam passado no método corresponder a uma instância de veículo, o método deve retorná-lo, caso não, null.

  - Os testes ocorreram da maneira correta, funcionando da maneira esperado com base nos casos de testes. Usamos o assertNotNull e assertNull para validar.

- **AdministradorDAOTest**

  ➔ public void testInsertAdministrador() - Responsável pela etapa de cadastramento de administrador, o método recebe um administrador como parâmetro e o cadastra no banco de dados. Caso bem sucedida, o método retorna true, caso contrário, retorna false.

  - Era esperado que no caso de teste onde os dados/campos ultrapassem o limite (no que diz respeito ao tamanho da string) definido na criação das tabelas retorna-se false, pois é uma situação que se espera, mas não aconteceu.
  - Fora essa situação, todos os outros casos de teste funcionaram da maneira esperada. Foi utilizado assertTrue e assertFalse para validar.

  ➔ public void testSearchAdministradorLogin() - O método realiza a busca no banco de dados em busca de administradores com o login igual ao passado como parâmetro.

  - Todas as possibilidades de teste foram executadas com sucesso e retornaram o resultado esperado, true caso tenha no banco de dados e false para caso não. Usamos assertNotNull e assertNull para validar.

  ➔ public void testSearchAdministradorId() - O método realiza a busca no banco de dados em busca de administradores com o id igual ao passado como parâmetro.

  - Os testes possíveis foram executados e retornaram o resultado esperado, true caso tenha no banco de dados e false para caso não. Usamos assertNotNull e assertNull para validar.

- **AutuacaoDAOTest**

  ➔ public void testInsertAutuacao() - É feita a inserção de uma instância de autuação no banco de dados. Para isso, o método recebe os parâmetros necessários e o insere no banco de dados, retornando true caso dê certo, ou false caso não.

    ◆ Foram executados os testes e o retorno foi como esperado, true para a inserção. Usamos assertTrue para validar.

  ➔ public void testSearchAutuacao() - No momento em que é necessário buscar uma autuação, é necessário informar seu identificador, e caso autuação esteja presente mesmo no banco de dados, a retorna, caso contrário retorna null.

    ◆ Os testes ocorreram conforme o esperado, retornando a autuação ou null. Usamos assertNull e assertNotNull para validar.

  ➔ public void testSearchIdAutuacao() - Há situações em que é necessário descobrir o identificador da autuação, para isso é feita a sua busca. O método responsável recebe dois parâmetros e retorna o id da autuação correspondente a aquela autuação.

    ◆ A execução do teste não ocorreu da maneira esperada, o motivo dessa anormalidade se dá pela passagem do parâmetro da data pelo usuário e não gerado pelo sistema conforme feito no uso padrão do sistema. Foi utilizado o assertEquals para validar o resultado.
    ◆ O teste realizado de maneira manual funcionou conforme esperado.

  ➔ public void testRemoveAutuacao() - O método recebe como parâmetro um id de uma autuação, e tem a função de removê-la do banco de dados. Retorna true se a remoção ocorre conforme o esperado.

    ◆ Os testes ocorreram conforme o esperado, retornando true se a remoção funcionar. Usamos assertTrue para validar.

- **RelatorioDAOTest**

  ➔ public void testInsertRelatorio() - Com esse método, é inserido/salvo relatórios no banco de dados, caso a inserção ocorra da maneira correta, o método retorna true, caso contrário retorna false.

    ◆ A execução do teste ocorreu da maneira esperada, com todos os casos de teste. Foi feito o uso do assertTrue para validar.

➔ public void testSearchIdRelatorio() - O método de busca de id de relatórios recebe como parâmetro o id do administrador e a data e hora do relatório. Caso esteja presente no banco de dados, retorna true senão retorna false.

- ◆ A execução do teste não ocorreu da maneira esperada, o motivo dessa anormalidade se dá pela passagem do parâmetro da data pelo usuário e não gerado pelo sistema conforme feito no uso padrão do sistema. Foi utilizado o assertEquals para validar o resultado.
- ◆ O teste realizado de maneira manual funcionou conforme esperado.

- **RelatorioAutuaocaoDAOTest**

➔ public void testInsertRelatorioAutuacao() - Quando um relatório acompanha autuações, é feito a sua inserção no banco de dados. Esse método retorna true caso a inserção seja concluída com sucesso, caso contrário, retorna false.

- ◆ Os testes executados funcionaram da maneira esperada. Foi utilizado assertTrue e assertFalse para validar.

➔ public void testSearchRelatorioAutuacoes() - Responsável por retorna todas as autuações que estão relacionadas a um relatório, então o método recebe como parâmetro o id do relatório que desejar reunir as autuações. Caso o teste funcione corretamente, o retorno será a lista de autuações, senão, null.

- ◆ Os testes executados seguiram o que era de se esperar. Para validar o retorno foi usado assertEquals.

Realização dos testes manuais:

- **cadastrarAdministrador() -**

Caso de teste 01 - Executado com sucesso
- Estrutura de dados:
  - nome (min) → “L”
  - email (min) → “E”
  - login (min) → “O”
  - senha → “12345678”

Caso de teste 02 - Executado com sucesso
- Estrutura de dados:

- nome (min) → “OL”
- email (min) → “MH”
- login (min) → “CD”
- senha → “123456789”

Caso de teste 03 - Executado com sucesso

- Estrutura de dados:
  - nome (normal) → “Leonardo Inácios Dantas Morais Maia Lima”
  - email (normal) → “LeoDantasMoraisLima@hotmal.bol.email.com”
  - login (normal) → “LeoMoraisLima99”
  - senha → “12345675689”

Caso de teste 04 - Executado com sucesso

- Estrutura de dados:
  - nome (max-) → “Teresa Cristina Maria Josefa Gaspar Baltasar Melchior Januária Rosalía Lúcia Sa”
  - email (max-) → “TeresaLuciaGasparCristinaMariaJosefaBaltasrMaria@gmail.hotmail.msn.email.com.br”
  - login (max-) → “TeresasLuciasMariaBaltasr1500”
  - senha → “1234567899”

Caso de teste 05 - Executado com sucesso

- Estrutura de dados:
  - nome (max) → “Pedro de Alcântara Francisco Antônio João Carlos Xavier Paula Miguel Rafael Joaq”
  - email (max) → “PedroAlcantaraJoaoCarlos1500FranciscoAntonio@hotmail.gmail.bol.emailprovider.com”
  - login (max) → “PebloAlcantaraJoaoCarloFAN1500”
  - senha → “123456789109”

A execução de casos de testes onde os valores não correspondem a especificação foi realizada, mas o sistema não dá sequência quando o valor do não é válido devido a validação do sistema. Dentre eles estão:

nome (min-) → “” (Tamanho inváildo)
nome (max+) → “Salvador Bibiano Francisco Xavier de Paula Leocádio Miguel Gabriel Rafael Gonzaga” (Tamanho inváildo)

email (min-) → “” (Tamanho inváildo)
email (max+) → “SalvadorBGonzagaFranciscoXaviarFilhoDoHomem1500OfficialEmail@emailprovider.com.br” (Tamanho inváildo)

login (min-) → “” (Tamanho inváildo)

login (max+) → “SalvadorDaliBGolsalves_Official” (Tamanho inváildo)

senha → “1234567”

- **buscarVeiculo() -**

  Caso de teste 01 - Executado com sucesso
  - Estrutura de dados:
    - placa → “INU0D54”
    - renavam → “346212854”

  Caso de teste 02 - Executado com sucesso
  - Estrutura de dados:
    - placa → “INU0D54”
    - renavam → “34657293411”

A execução de casos de testes onde os valores não correspondem a especificação foi realizada, mas o sistema não dá sequência quando o valor do não é válido devido a validação do sistema. Dentre eles estão:

placa → “INU0D5” - (Tamanho inválido (min -))
placa → “I5U0D54” - (Padrão inválido)
placa → “INU0DG4” - (Padrão inválido)
placa → “INU0954” - (Padrão inválido)
placa → “INU0D542” - (Tamanho inválido (max +))

renavam → “2346464877” - (Tamanho inválido)
renavam → “46484648” - (Tamanho inválido)
renavam → “124968894873” - (Tamanho inválido)
renavam → “56445FGD341” - (Formato inválido)

Durante os testes para validar os campos, foi possível observar que a validação que deveria está funcionando para não permitir que o número do renavam possuisse letras não estava funcionando. A partir da execução dos testes e depuração foi possível corrigir essa falha.

- **buscarMulta() -**

  Caso de teste 01 - Executado com sucesso
  - Estrutura de dados:
    - codigo → 36124

A execução de casos de testes onde os valores não correspondem a especificação foi realizada, mas o sistema não dá sequência quando o valor do não é válido devido a validação do sistema. Dentre eles estão:

codigo → 9999 (Fora do intervalo);
codigo → 364592 (Fora do intervalo);

- **listaMultas() -**

  Caso de teste 01 - Executado com sucesso
  - Estrutura de dados:
    - classificacao → “LEVE”

  Caso de teste 02 - Executado com sucesso
  - Estrutura de dados:
    - classificacao → “MÉDIA”

  Caso de teste 03 - Executado com sucesso
  - Estrutura de dados:
    - classificacao → “GRAVE”

  Caso de teste 04 - Executado com sucesso
  - Estrutura de dados:
    - classificacao → “GRAVÍSSMA”

  Caso de teste 05 - Executado com sucesso
  - Estrutura de dados:
    - classificacao → “gRaVe”

A execução de casos de testes onde os valores não correspondem a especificação foi realizada, mas o sistema não dá sequência quando o valor do não é válido devido a validação do sistema. Dentre eles estão:

classificacao → “QUALQUERCOISA” (Valor inválido);

- **cadastrarAutuacao() -**

  Caso de teste 01 - Executado com sucesso
  - Estrutura de dados:
    - placa → “INU0D54”
    - renavam → “34657293411”
    - codigo → 67964
    - classificacao → “LEVE”
    - estado (min) → “E”
    - municipio (min) → “M”

  Caso de teste 02 - Executado com sucesso
  - Estrutura de dados:
    - placa → “INU0D54”
    - renavam → “456482665”
    - codigo → 42346
    - classificacao → “MÉDIA”
    - estado (min+) → “LO”

- municipio (min+) → “AS”

Caso de teste 03 - Executado com sucesso

- Estrutura de dados:
  - placa → “INU0D54”
  - renavam → “456482665”
  - codigo → 97986
  - classificacao → “GRAVE”
  - estado (normal) → “Estado da Federação Brasileira Maravilha”
  - municipio (normal) → “Nova Esperança do Sul das Flores Eternas”

Caso de teste 04 - Executado com sucesso

- Estrutura de dados:
  - placa → “INU0D54”
  - renavam → “34657293411”
  - codigo → 42341
  - classificacao → “GRAVÍSSMA”
  - estado (normal) → “Estado da União das Riquezas Naturais e da Cultura Diversificadas e Fascinantes”
  - municipio (normal) → “Santa Cruz do Sul da Esperança e da Prosperidade Infinita Criada Pela Naturezas”

Caso de teste 05 - Executado com sucesso

- Estrutura de dados:
  - placa → “INU0D54”
  - renavam → “456482665”
  - codigo → 31234
  - classificacao → “LeVe”
  - estado (normal) → “Estado da Federação Brasileira das Mil Cores e da Cultura Vibrante e Inspiradora”
  - municipio (normal) → “Cidade da Paz e do Progresso Sustentável Harmoniosa e Acolhedora Proximos Pontos”

A execução de casos de testes onde os valores não correspondem a especificação foi realizada, mas o sistema não dá sequência quando o valor do não é válido devido a validação do sistema. Dentre eles estão:

placa → “I5U0D54” - (Padrão inválido)
placa → “INU0DG4” - (Padrão inválido)
placa → “INU0954” - (Padrão inválido)
placa → “INU0D542” - (Tamanho inválido (max +))

renavam → “2346464877” (Tamanho inválido)
renavam → “46484648” (Tamanho inválido (min -))
renavam → “124968894873” (Tamanho inválido (max +))
renavam → “56445FGD31” (Formato inválido)

classificacao → “QUALQUERCOISA” (Valor inválido);

codigo → 6956 (Fora do intervalo);
codigo → 364592 (Fora do intervalo);

estado (min -) → “” (Tamanho inváildo)
estado (min +) → “Estado da União Federativa do Planalto Central da Eterna Prosperidade Sustentável” (Tamanho inváildo)

municipio (min -) → “” (Tamanho inváildo)
municipio (min +) → “Vale das Montanhas da Eterna Serenidade e Beleza Natural Incomparável Que Fica Al”) (Tamanho inváildo)

Durante os testes para validar os campos, foi possível observar que a validação que deveria estar funcionando para não permitir que o número do renavam possuísse letras não estava funcionando. A partir da execução dos testes e depuração foi possível corrigir essa falha.

- **removerAutuacao() -**

  Caso de teste 01 - Executado com sucesso
  - Estrutura de dados:
    - id → 4

A execução de casos de testes onde os valores não correspondem a especificação foi realizada, mas o sistema não dá sequência quando o valor é inválido devido à validação do sistema. Dentre eles estão:

id → “f” (Tipo inválido)
id → “+” (Tipo inválido)

- **main() -**

  Caso de teste 01 - Executado com sucesso
  - Estrutura de dados:
    - opcao → 6

  Caso de teste 01 - Executado com sucesso
  - Estrutura de dados:
    - opcao → 5

  Caso de teste 01 - Executado com sucesso
  - Estrutura de dados:
    - opcao → 1

A execução de casos de testes onde os valores não correspondem a especificação foi realizada, mas o sistema não dá sequência quando o valor do é inválido devido à validação do sistema. Dentre eles estão:

opcao → “d” (Tipo inválido)
opcao → “$” (Tipo inválido)

Teste de integração

Ao contrário dos testes de unidade que se concentram em métodos e classes individuais, os testes de integração avaliam a interação entre diferentes componentes ou módulos do sistema. No contexto de desenvolvimento orientado a objetos, os testes de integração garantem que as classes e métodos colaborem corretamente para alcançar os resultados desejados. Esses testes examinam as interfaces e as interações entre as unidades de código para identificar possíveis falhas de comunicação ou integração.

A execução dos testes de integração é crucial para validar o fluxo de dados e as transições entre diferentes partes do sistema. Ao contrário dos testes de unidade, que podem ser automatizados para métodos individuais, os testes de integração frequentemente envolvem cenários mais complexos que exigem a simulação de situações do mundo real.

Roteiro dos testes de integração:

Tomando como base as características do nosso sistema, a abordagem escolhida para realizar os testes de integração foi o Bottom-Up, de Baixo para Cima. Bottom-Up é uma abordagem que foca na integração progressiva dos componentes do sistema, começando pelas unidades individuais e subindo até os módulos mais complexos.

Como os testes unitários dos já foram realizados, a partir desse momento iremos tratar das próprias integrações, detalhando e executando casos de testes com intuito de verificar possíveis falhas e se possível percorrer os caminhos do sistema. A elaboração dos casos de testes se deu conforme as especificações do sistema.

Realização dos testes:

- **VeiculoController**

  - ***buscarVeiculo()*** faz o uso do método ***buscaVeiculo()*** para realizar a busca do veículo no banco de dados e enfim concluir a sua função que é de retornar o veículo. Foi realizado dois casos de testes:

    Caso de teste 01 - Executado com sucesso
    - Estrutura de dados:
      - placa → “INU0D54”
      - renavam → “90587358493”

- Valor esperado:
    - Veículo
- Valor resultante:
    - Veículo

Caso de teste 02 - Executado com sucesso

- Estrutura de dados:
    - placa → “NHG8L23”
    - renavam → “346593411”
- Valor esperado:
    - Null
- Valor resultante:
    - Null

- ***verDadosVeiculo()*** faz o uso do método ***buscarVeiculo()*** para realizar a busca do veículo que deseja imprimir. A impressão somente ocorre quando o resultado da busca é diferente de null.

Caso de teste 01 - Executado com sucesso

- Estrutura de dados:
    - veiculo → Veiculo
- Apresentação esperada:
    - Veículo
- Apresentação resultante:
    - Veículo

Caso de teste 02 - Executado com sucesso

- Estrutura de dados:
    - veiculo → null
- Apresentação esperada:
    - “Nenhum veículo cadastrado com essas informações”
- Apresentação resultante:
    - “Nenhum veículo cadastrado com essas informações”

- **MultaController**

- ***buscarMulta()*** faz o uso da ***buscaMulta()*** para realizar a busca da multa no banco de dados e enfim concluir a sua função, que é de retornar a multa. Foi realizado dois casos de testes:

Caso de teste 01 - Executado com sucesso

- Estrutura de dados:
    - id → 67900
- Valor esperado:
    - Multa
- Valor resultante:
    - Multa

Caso de teste 02 - Executado com sucesso

- Estrutura de dados:
  - id → 11111
- Valor esperado:
  - Null
- Valor resultante:
  - Null

- ***verDadosMulta()*** faz o uso da ***buscarMulta()*** para realizar a busca da multa que deseja imprimir. A impressão somente ocorre quando o resultado da busca é diferente de null.

Caso de teste 01 - Executado com sucesso

- Estrutura de dados:
  - id → 67900
- Valor esperado:
  - Multa
- Valor resultante:
  - Multa

Caso de teste 02 - Executado com sucesso

- Estrutura de dados:
  - id → 11111
- Valor esperado:
  - “Nenhuma multa cadastrada com essas informações!”
- Valor resultante:
  - “Nenhuma multa cadastrada com essas informações!”

- ***listaMultas()()*** faz o uso da ***buscaMultaClassificacao() para realizar a busca de todas multas caracterizadas pela classificação passada como parâmetro presente no banco de dados para assim poder imprimi-las.*** para realizar a busca das multas com essa classificação que deseja imprimir. A impressão somente ocorre quando o resultado da busca é diferente de null

Caso de teste 01 - Executado com sucesso

- Estrutura de dados:
  - id → "LEVE"
- Impressão esperada:
  - Lista de multas
- Impressão resultante:
  - Lista de multas

Caso de teste 02 - Executado com sucesso

- Estrutura de dados:
  - id → 11111
- Valor esperado:
  -
- Valor resultante:
  -

Caso de teste 01 - Executado com sucesso

- Estrutura de dados:
  - classificacao → “MÉDIA”
- Apresentação esperada:
  - Lista de Multas
- Apresentação resultante:
  - Lista de Multas

Caso de teste 02 - Executado com sucesso

- Estrutura de dados:
  - classificacao → “QUALQUerCOisa”
- Apresentação esperado:
  - “Nenhuma multa cadastrada com essa classificação!”
- Apresentação resultante:

“Nenhuma multa cadastrada com essa classificação!”

- **RelatorioDAO**

  - ***insereRelatorio()*** faz uso do método ***buscaIdRelatorio()*** para realizar a busca do id do relatório que foi cadastrado anteriormente para ser possível realizar a inserção na tabela RelatorioAutuacao.

  Caso de teste 01 - Executado com sucesso

  - Estrutura de dados:
    - idAutor → 4
    - dataHora → 2023-10-11 11:19:48
  - Valor esperada:
    - 8
  - Valor resultante:
    - 8

  Como a inserção do relatório ocorreu bem para que o código tenha chegado aqui, é impossível realizar um caminho de falha com casos de teste, num cenário onde os atributos passados na função são os mesmos inseridos no banco de dados.

  - ***insereRelatorio()*** faz o uso do método ***insereRelatorioDeAutuacao()*** para realizar a inserção de uma ocorrência da relação de relatório e autuação.

  Caso de teste 01 - Executado com sucesso

  - Estrutura de dados:
    - idRelatorio → 4
    - idAutuacao→ 17
  - Comportamento esperado:
    - Inserção
  - Comportamento resultante:
    - Inserção

- **AutuacaoTransitoDAO**

  - ***buscaAutuacao()*** faz o uso do método ***buscaAdministrador()*** para realizar a busca do autor no banco de dados, tomando como base no seu id presente autuação buscada, para assim criar o objeto e retorná-lo.

    Caso de teste 01 - Executado com sucesso
    - Estrutura de dados:
      - autorId → “2”
    - Comportamento esperado:
      - autor == Administrador
    - Comportamento resultante:
      - autor == Administrador

  - ***buscaAutuacao()*** faz o uso do método ***buscaVeiculo()*** para realizar a busca do veículo no banco de dados, tomando como base a placa e o renavam presente autuação buscada, para assim criar o objeto e retorná-lo.

    Caso de teste 01 - Executado com sucesso
    - Estrutura de dados:
      - placa → “INU0D54”
      - renavam → “90587358493”
    - Comportamento esperado:
      - veiculo == Veiculo
    - Comportamento resultante:
      - veiculo == Veiculo

  - ***buscaAutuacao()*** faz o uso do método ***buscaMulta()*** para realizar a busca da multa no banco de dados, tomando como base o código presente autuação buscada, para assim criar o objeto e retorná-lo.

    Caso de teste 01 - Executado com sucesso
    - Estrutura de dados:
      - codigo → “689-00”
    - Comportamento esperado:
      - multa == Multa
    - Comportamento resultante:
      - multa == Multa

- **AutuacaoTransitoController**

  - ***cadastrarAutuacao()*** faz o uso do método ***buscaVeiculo()*** para validar se o veículo informado (placa e renavam) corresponde a um veículo cadastrado no banco de dados, já que o método faz uma busca com base em seus identificadores, retornando o veículo caso ele exista.

    Caso de teste 01 - Executado com sucesso

- Estrutura de dados:
    - placa → “INU0D54”
    - renavam → “90587358493”
- Comportamento esperado:
    - veiculo == Veiculo
- Comportamento resultante:
    - veiculo == Veiculo

Caso de teste 02 - Executado com sucesso

- Estrutura de dados:
    - placa → “IDD1X12”
    - renavam → “69898612345”
- Comportamento esperado:
    - veiculo == null
- Comportamento resultante:
    - veiculo == null

- ***cadastrarAutuacao()*** faz o uso do método ***listaMultas()*** para auxiliar o administrador na visualização de todas as multas que ele pode vincular naquele veículo com base na classificação que o mesmo escolheu.

Caso de teste 01 - Executado com sucesso

- Estrutura de dados:
    - classificacao → “LEVE”
- Comportamento esperado:
    - “Lista de multas: ”
- Comportamento resultante:
    - “Lista de multas: ”

- ***cadastrarAutuacao()*** faz o uso do método ***buscaMulta()*** para verificar e validar se o código informado pelo administrador corresponde a alguma multa cadastrada no banco de dados. O método faz o retorno da multa caso exista.

Caso de teste 01 - Executado com sucesso

- Estrutura de dados:
    - codigo → “689-00”
- Comportamento esperado:
    - multa == Multa
- Comportamento resultante:
    - multa == Multa

Caso de teste 02 - Executado com sucesso

- Estrutura de dados:
    - codigo → “584-99”
- Comportamento esperado:
    - multa == null
- Comportamento resultante:

        - multa == null

- ***cadastrarAutuacao()*** faz o uso do método ***insereAutuacao()*** que o intuito de realizar a inserção/armazenamento da autuação no banco de dados, já que até o momento a autuação avia sido apenas criada. O método retorno retorna true caso a inserção ocorra como esperado.

Caso de teste 01 - Executado com sucesso

- Estrutura de dados:
    - autuacao → Objeto do tipo AutuacaoTransito
- Comportamento esperado:
    - Inserção bem sucedida
- Comportamento resultante:
    - Inserção bem sucedida

- ***cadastrarAutuacao()*** faz o uso do método ***buscaIdAutuacao()*** para obter o id da autuação recém criada e armazenada, pois o seu id gerado somente após a inserção e ele é necessário para o sistema prosseguir.

Caso de teste 01 - Executado com sucesso

- Estrutura de dados:
    - idAutor → 2
    - dataHora → 2023-10-10 19:55:36
- Comportamento esperado:
    - id = 7
- Comportamento resultante:
    - id = 7

- ***removerAutuacao()*** faz o uso do método ***buscaAutuacao()*** para obter a autuação que será removida do banco de dados para assim removê-la também do buffer que armazene as ações do administrador.

Caso de teste 01 - Executado com sucesso

- Estrutura de dados:
    - id → 7
- Comportamento esperado:
    - autuacao → Objeto AutuacaoTransito
- Comportamento resultante:
    - autuacao → Objeto AutuacaoTransito

- ***removerAutuacao()*** faz o uso do método ***removeAutuacao()*** concluir a remoção da autuação do sistema, onde nesse momento é feita a sua remoção do banco de dados.

Caso de teste 01 - Executado com sucesso

- Estrutura de dados:
    - id → 7
- Comportamento esperado:

        - Remoção concluída com sucesso
    - Comportamento resultante:
        - Remoção concluída com sucesso

- **RelatorioController**

  - ***criarRelatorio()*** faz o uso do método ***insereRelatorio()*** para efetuar de fato a inserção do relatório criado no banco de dados, pois até o momento, o relatório tinha apenas sido criado no sistema. Essa função chamada retorna um valor booleano.

    Caso de teste 01 - Executado com sucesso
    - Estrutura de dados:
        - relatorio → Objeto do tipo Relatorio
    - Comportamento esperado:
        - Apresentação do relatório
    - Comportamento resultante:
        - Apresentação do relatório

- **AdministradorController**

  - ***logar()*** faz o uso do método ***buscaAdministrador()*** para realizar a busca do administrador no banco de dados para verificar se aquele login está realmente cadastrado e assim validar o seu login. O método retorna um Administrador.

    Caso de teste 01 - Executado com sucesso
    - Estrutura de dados:
        - login → “admin”
        - senha → “Jogador123”
    - Comportamento esperado:
        - Objeto Administrador
    - Comportamento resultante:
        - Objeto Administrador

    Caso de teste 02 - Executado com sucesso
    - Estrutura de dados:
        - login → “admin”
        - senha → “Jogador”
    - Comportamento esperado:
        - “Credenciais incorretas!”
    - Comportamento resultante:
        - “Credenciais incorretas!”

    Caso de teste 03 - Executado com sucesso
    - Estrutura de dados:
        - login → “adminmin”
        - senha → “Jogador123”

- Comportamento esperado:
    - “Credenciais incorretas!”
- Comportamento resultante:
    - “Falha no sistema”

Com a execução do 3º caso de testes, foi possível observar uma falha no sistema, pois no momento em que o administrador insere o login errado o sistema encerrar devido a uma falha.

- ***cadastrarAdministrador()*** faz o uso do método ***insereAdministrador()*** para cadastrar o administrador criado a partir da entrada de dados do administrador no banco de dados. Caso a inserção funcione, retorna true.

Caso de teste 01 - Executado com sucesso

- Estrutura de dados:
    - adm → Objeto do tipo Administrador
- Comportamento esperado:
    - Inserção bem sucedida
- Comportamento resultante:
    - Inserção bem sucedida

- ***vincularMulta()*** faz a chamada do método ***cadastrarAutuacao()*** para vincular uma multa a um veículo, cadastrar uma autuação a um veículo. O método retorna true se a vinculação ocorrer bem.

Como o método *vincularMulta()* trata de apenas uma chamada de uma função, o único caso de teste possível é realizar sua chamada, e o mesmo está funcionando perfeitamente.

- ***desvincularMulta()*** faz a chamada do método ***removerAutuacao()*** para desvincular uma multa de um veículo, em outras palavras, excluir uma autuação. O método returna true se a remoção ocorrer bem.

Como o método *desvincularMulta()* trata de apenas uma chamada de uma função, o único caso de teste possível é realizar sua chamada, e o mesmo está funcionando perfeitamente.

- ***consultarVeiculo()*** faz a chamada do método ***verDadosVeiculo()*** para apresentar o veículo buscado pelo administrador.

Como o método *consultarVeiculo()* trata de apenas uma chamada de uma função, o único caso de teste possível é realizar sua chamada, e o mesmo está funcionando perfeitamente.

- ***consultarMulta()*** faz a chamada do método ***verDadosMulta()*** para apresentar a multa buscado pelo administrador.

Como o método *consultarMulta()* trata de apenas uma chamada de uma função, o único caso de teste possível é realizar sua chamada, e o mesmo está funcionando como deveria.

- ***gerarRelatorio()*** faz a chamada do método ***criarRelatorio()*** para iniciar o processo de criação e inserção do relatório no banco de dados.

Como o método *gerarRelatorio()* trata de apenas uma chamada de uma função, o único caso de teste possível é realizar sua chamada, e o mesmo está funcionando como esperado.

TESTE ESTRUTURAIS

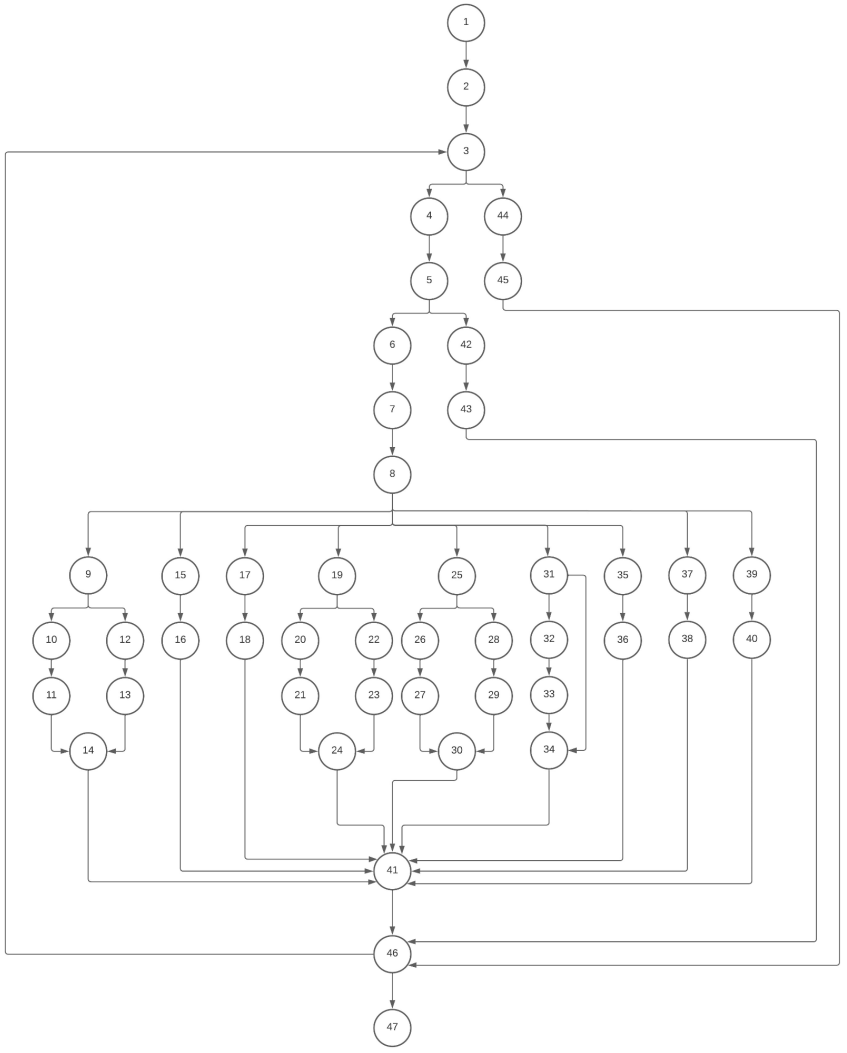
Testes estruturais, também conhecidos como testes de estrutura, são considerados uma parte fundamental no desenvolvimento de qualquer software, ele é projetado para avaliar a estrutura interna de um programa ou sistema, garantindo que cada parte individual, conhecida como unidade, funcione conforme o esperado. Esta prática é vital para identificar defeitos e problemas antes que o software seja implementado em um ambiente de produção.

Decorrente da análise do sistema, tomando como base a sua estrutura e granularidade e o tempo que temos para a realização dos testes, priorizamos realizar analisar dos caminhos lógicos do sistema, fluxo de controle, assim como o fluxo de dados das nossas variáveis durante a execução de cada método relevante para esse projeto no que diz respeito a apresentação de um resultado.

Método **main() {}**

```
/*  1  */ public static void main( String[] args ) {
/*  2  */        Scanner input = new Scanner(System.in);
/*  2  */        Administrador adm = null;
/*  2  */
/*  2  */        int opcao = 0;
/*  2  */
/*  2  */        System.out.println("\n\t\t\t **** SIMULT-PDF  V.2 **** \n");
/*  2  */
/*  2  */        adm = logar();
/*  2  */
/*  3  */        do {
/*  4  */                if (adm != null) {
/*  5  */                        System.out.println("\n\t- Que operação deseja realizar:");
/*  5  */                        System.out.print("\t\t1. Cadastrar novo administrador\n\t\t 2.
                                 Buscar veículo\n\t\t 3. Buscar multa\n\t\t 4. Vincular multa a
                                 um veículo\n\t\t 5. Desvincular multa de um veículo\n\t\t 6.
                                 Gerar relatório de autuações\n\t\t 7. Sair\n\t\t 8. Fechar
                                 sistema\n\t Opcao: ");
/*  6  */                        try {
/*  7  */                                opcao = Integer.parseInt(input.nextLine());

/*  8  */                                switch (opcao) {
/*  9  */                                        case 1:
/* 10  */                                                if (cadastrarAdministrador(adm)) {
/* 11  */                                                        System.out.println("\n\t\t
                                                                 Cadastramento concluído com
                                                                 sucesso!");
/* 12  */                                                } else if (!cadastrarAdministrador(adm)) {
/* 13  */                                                        System.out.println("\n\t\t Falha no
                                                                 cadastramento!");
/* 13  */                                                }
/* 14  */                                                break;
/* 15  */                                         case 2:
/* 16  */                                                consultarVeiculo();
/* 16  */                                                break;
/* 17  */                                        case 3:
/* 18  */                                                consultarMulta();
/* 18  */                                                break;
/* 19  */                                        case 4:
/* 20  */                                                if (vincularMulta(adm)){
/* 21  */                                                        System.out.println("\n\t\t Autuação
                                                                 vinculada com sucesso!");
/* 22  */                                                } else {
/* 23  */                                                        System.out.println("\n\t\t Falha no
                                                                 vinculação da autuação!");
/* 23  */                                                }
/* 24  */                                                break;
/* 25  */                                        case 5:
/* 26 */                                                if (desvincularMulta(adm)){
/* 27  */                                                        System.out.println("\n\t\t Autuação
                                                                 desvinculada com sucesso!");
/* 28  */                                                } else {
/* 29  */                                                        System.out.println("\n\t\t Falha no
                                                                 desvinculação da autuação!");
/* 29  */                                                }
/* 30  */                                                break;
/* 31  */                                        case 6:
/* 32  */                                                if (!gerarRelatorio(adm)){
/* 33  */                                                        System.out.println("\n\t\t Falha na
                                                                 geração do relatório!");
/* 33  */                                                }
/* 34  */                                                break;
/* 35  */                                        case 7:
```

```
/* 36  */                                    adm = null;
/* 36  */                                    break;
/* 37  */                                case 8:
/* 38  */                                    System.out.println("\n\t\t Encerrando
                                             sistema!!!\n");
/* 38  */                                    break;
/* 39  */                                default:
/* 40  */                                    System.out.println("\n\t\t Opção
                                             inválida!!!");
/* 40  */                                    break;
/* 41  */                            }
/* 42  */                        } catch (NumberFormatException e) {
/* 43  */                            System.out.println("\n\t\t O campo opção aceita
                                     somente números!\n");
/* 43  */                        }
/* 44  *                   } else {
/* 45  */                        System.out.println("\n\t\t Para acessar o sistema é
                                 necessário está logado!\n");
/* 45  */                        adm = logar();
/* 45  */                    }
/* 46  */            } while (opcao != 9);
/* 47  */    }
```



Analise do fluxo de controle do método *main()*

Considerando o grafo de fluxo controle apresentado anterior a respeito do método main() é possível realizar o cálculo da complexidade ciclomática.

- O grafo de fluxo contém **16 regiões;**
- V(G) = 61 - 47 + 2 = **16;**
- V(G) = 15 nós predicados + 1 = **16;**

A complexidade ciclomática do método *main()* é de 16 e com isso o seu valor nos fornece um limite superior para o número de caminhos independentes que podem formar o conjunto base, ou seja, 16 é número de testes que devem ser projetados e executados para garantir uma cobertura total do método. Caminhos esses seriam:

- 1 2 3 4 5 6 7 8 9 10 11 14 41 46 47
- 1 2 3 4 5 6 7 8 9 12 13 14 41 46 47
- 1 2 3 4 5 6 7 8 15 16 41 46 47
- 1 2 3 4 5 6 7 8 17 18 41 46 47
- 1 2 3 4 5 6 7 8 19 20 21 24 41 46 47
- 1 2 3 4 5 6 7 8 19 22 23 24 41 46 47
- 1 2 3 4 5 6 7 8 25 26 27 30 41 46 47
- 1 2 3 4 5 6 7 8 25 28 29 30 41 46 47
- 1 2 3 4 5 6 7 8 31 32 33 34 41 46 47
- 1 2 3 4 5 6 7 8 31 34 41 46 47
- 1 2 3 4 5 6 7 8 35 36 41 46 47
- 1 2 3 4 5 6 7 8 37 38 41 46 47
- 1 2 3 4 5 6 7 8 39 40 41 46 47
- 1 2 3 4 5 42 43 46 47
- 1 2 3 44 45 46 47
- 1 2 3 4 5 6 7 8 17 18 41 46 3 4 5 6 7 8 17 18 41 46 47

Análise do fluxo de dados de método *buscarVeiculo()*

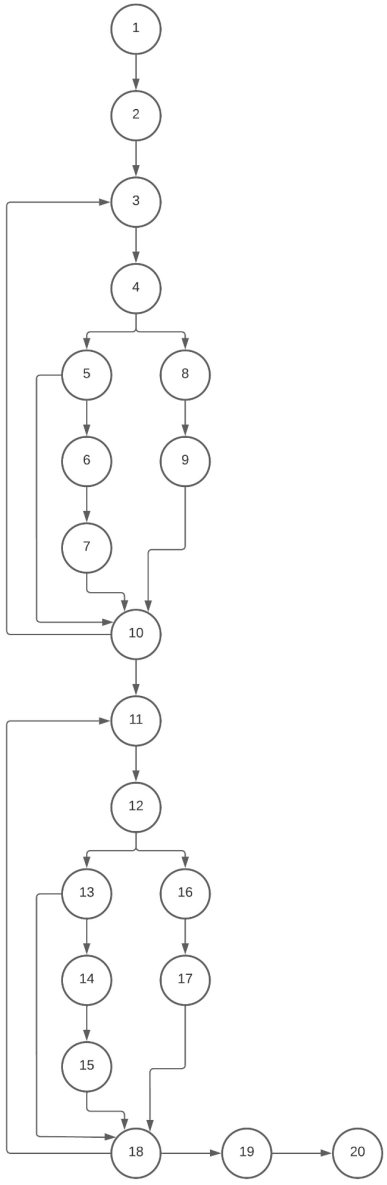
Tomando como base o código correspondente ao método em questões, a análise do fluxo de dados resultou em:

- **adm** : ~d → dd → du → uu → ud → dk
- **opcao** : ~d → dd → du → uu → uk

Apesar que dk ser um pouco estranho, seguindo o caminho de execução do código, faz sentido.

Método **buscarVeiculo() {}**

```
/*  1  */ public static Veiculo buscarVeiculo(){
/*  2  */        String placa = "";
/*  2  */        String renavam = "";
/*  2  */        System.out.println("\nInsira as informações do veículo para realizar a busca:\n");
/*  3  */        do {
/*  4  */                System.out.print("\t- Escreva a PLACA do veículo: ");
/*  4  */                placa = input.nextLine().toUpperCase();
/*  4  */                Pattern letras = Pattern.compile("^[A-Z]+$");
/*  4  */                Pattern nums = Pattern.compile("^[0-9]+$");
/*  5  */                if(placa.length() == 7){
/*  6  */                        if(!letras.matcher(placa.substring(0, 3)).find() && !nums.matcher(("" 
                                 + placa.charAt(3))).find() && !letras.matcher(("" +
                                 placa.charAt(4))).find() && !nums.matcher(placa.substring(5,
                                 7)).find()){
/*  7  */                                System.out.println("\n\t\t Formato da placa do veículo
                                         incorreta!\n");
/*  7  */                                 placa = "";
/*  7  */                        }
/*  8  */                } else {
/*  9  */                        System.out.println("\n\t\t A placa do veículo deve ser formada por 7
                                 dígitos!\n");
/*  9  */                }
/* 10  */        } while (placa.length() != 7);
/* 11  */        do {
/* 12  */                System.out.print("\t- Informe o RENAVAM do veículo: ");
/* 12  */                renavam = input.nextLine();
/* 12  */                Pattern letras = Pattern.compile("[a-zA-Z]+");
/* 13  */                if(renavam.length() == 9 || renavam.length() == 11) {
/* 14  */                        if (letras.matcher(renavam).find()){
/* 15  */                                System.out.println("\n\t\t O RENAVAM deve conter apenas
                                         números!\n");
/* 15  */                                renavam = "";
/* 15  */                        }
/* 16  */                } else {
/* 17  */                        System.out.println("\n\t\t É necessário informar o RENAVAM válido
                                 (com 9 ou 11 algarismos)!\n");
/* 17  */                }
/* 18  */        } while (renavam.length() != 9 && renavam.length() != 11);
/* 19  */        return buscaVeiculo(placa, renavam);
/* 20  */ }
```



Analise do fluxo de controle do método *buscarVeiculo()*

Considerando o grafo de fluxo controle apresentado anteriormente a respeito do método *buscarVeiculo()* é possível realizar o cálculo da complexidade cíclomática.

- O grafo de fluxo contém **7 regiões;**
- V(G) = 25 - 20 + 2 = **7;**
- V(G) = 6 nós predicados + 1 = **7;**

A complexidade ciclomática do método *buscarVeiculo()* é de 7 e com isso o seu valor nos fornece um limite superior para o número de caminhos independentes que podem formar o conjunto base, ou seja, 7 é número de testes que devem ser projetados e executados para garantir uma cobertura 100% do método. Caminhos esses seriam:

- 1 2 3 4 5 6 7 10 11 12 13 14 15 18 19 20
- 1 2 3 4 5 6 7 10 11 12 13 18 19 20
- 1 2 3 4 5 6 7 10 11 12 16 17 18 19 20
- 1 2 3 4 5 10 11 12 13 14 15 18 19 20
- 1 2 3 4 8 9 10 11 12 16 17 18 19 20
- 1 2 3 4 5 6 7 10 3 4 5 6 7 10 11 12 13 18 19 20
- 1 2 3 4 5 6 7 10 11 12 13 18 11 12 13 18 19 20

Análise do fluxo de dados de método *buscarVeiculo()*

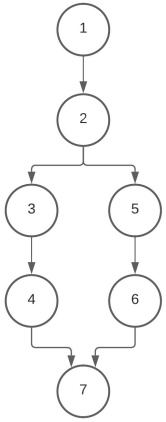
Tomando como base o código correspondente ao método em questões, a análise do fluxo de dados resultou em:

- **placa** : ~d → dd → du → uu → ud → du → uu → uk
- **renavam** : ~d → dd → du → uu → ud → du → uu → uk
- **letras** : ~d → dd → du → uk
- **nums** : ~d → dd → du → uk

Nenhuma anormalidade observada.

Método **verDadosVeiculo() {}**

```
/*  1  */ public static void verDadosVeiculo(){
/*  2  */        Veiculo veiculo = buscarVeiculo();

/*  3  */        if(veiculo != null){
/*  4  */              System.out.println("\n\tVeículo ->" + veiculo);
/*  5  */        } else {
/*  6  */              System.out.println("\n\t\t Nenhum veículo cadastrado com essas
                      informações!\n");
/*  6  */        }
/*  7  */ }
```



Analise do fluxo de controle do método *verDadosVeiculo()*

Considerando o grafo de fluxo controle apresentado anteriormente a respeito do método *verDadosVeiculo()* é possível realizar o cálculo da complexidade cíclomática.

- O grafo de fluxo contém **2 regiões;**
- V(G) = 7 - 7 + 2 = **2;**
- V(G) = 1 nós predicados + 1 = **2;**

A complexidade ciclomática do método *verDadosVeiculo()* é de 2 e com isso o seu valor nos fornece um limite superior para o número de caminhos independentes que podem formar o conjunto base, ou seja, 2 é número de testes que devem ser projetados e executados para garantir uma cobertura 100% do método. Caminhos esses seriam:

- 1 2 3 4 7
- 1 2 5 6 7

Análise do fluxo de dados de método *verDadosVeiculo()*

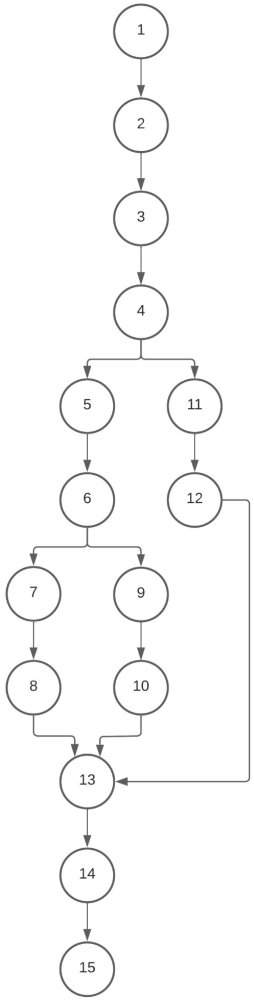
Tomando como base o código correspondente ao método em questões, a análise do fluxo de dados resultou em:

- **veiculo**: ~d → du → uu → uk

Nenhuma anormalidade observada.

Método **buscarMulta() {}**

```
/*  1  */ public static Multa buscarMulta(){
/*  2  */          int id = 0;
/*  2  */          String codigo = "";

/*  2  */          System.out.println("\nInsira as informações da multa para realizar a
                    busca:\n");

/*  3  */          do {
/*  4  */                  String aux1 = "";
/*  4  */                  String aux2 = "";

/*  5  */                  try {
/*  6  */                          System.out.print("\t- Informe o código da multa (apenas
                                    números): ");
/*  6  */                          id = Integer.parseInt(input.nextLine());

/*  7  */                          if (id < 10000 || id > 99999){
/*  8  */                                  System.out.println("\n\t\t Tamanho inválido do campo de
                                        código!\n");
/*  9  */                          } else {
/* 10  */                                  codigo = "" + id;
/* 10  */                                  aux1 = codigo.substring(0,3);
/* 10  */                                  aux2 = codigo.substring(3, 5);

/* 10  */                                  codigo = aux1 + "-" + aux2;
/* 10  */                          }
/* 11  */                  } catch (NumberFormatException e) {
/* 12  */                          System.out.println("\n\t\t O campo código aceita somente
                                    números!\n");
/* 12  */                  }

/* 13  */          } while (codigo.isEmpty());

/* 14  */          return buscaMulta(codigo);
/* 15  */ }
```



Analise do fluxo de controle do método *buscarMulta()*

Considerando o grafo de fluxo controle apresentado anteriormente a respeito do método *buscarMulta()* é possível realizar o cálculo da complexidade cíclomática.

- O grafo de fluxo contém **3 regiões;**
- V(G) = 16 - 15 + 2 = **3;**
- V(G) = 2 nós predicados + 1 = **3;**

A complexidade ciclomática do método *buscarMulta()* é de 3 e com isso o seu valor nos fornece um limite superior para o número de caminhos independentes que podem formar o conjunto base, ou seja, 3 é número de testes que devem ser projetados e executados para garantir uma cobertura ideal do método. Caminhos esses seriam:

- 1 2 3 4 5 6 7 8 13 14 15
- 1 2 3 4 5 6 9 10 13 14 15
- 1 2 3 4 11 12 13 14 15

Análise do fluxo de dados de método *buscarMulta()*

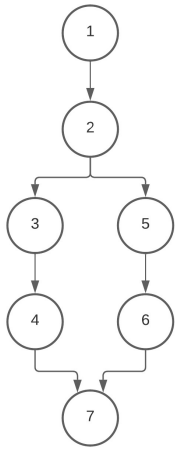
Tomando como base o código correspondente ao método em questões, a análise do fluxo de dados resultou em:

- **id** : ~d → dd → du → uu → uu → uk
- **codigo** : ~d → dd → dd → du → uu → uk
- **aux1** : ~d → dd → du → uk
- **aux2** : ~d → dd → du → uk

Nenhuma anormalidade observada.

Método **verDadosMulta() {}**

```
/*  1  */ public static void verDadosMulta(){
/*  2  */          Multa multa = buscarMulta();
/*  2  */
/*  3  */          if (multa != null){
/*  4  */                  System.out.println("\n\tMulta -> " + multa);
/*  5  */          } else {
/*  6  */                  System.out.println("\n\t\t Nenhuma multa cadastrada com essas
                           informações!\n");
/*  6  */          }
/*  7  */ }
```



Análise do fluxo de controle do método *verDadosMulta()*

Considerando o grafo de fluxo controle apresentado anteriormente a respeito do método *verDadosMulta()* é possível realizar o cálculo da complexidade cíclomática.

- O grafo de fluxo contém **2 regiões;**
- V(G) = 7 - 7 + 2 = **2;**
- V(G) = 1 nós predicados + 1 = **2;**

A complexidade ciclomática do método *verDadosMulta()* é de 2 e com isso o seu valor nos fornece um limite superior para o número de caminhos independentes que podem formar o conjunto base, ou seja, 2 é número de testes que devem ser projetados e executados para garantir uma cobertura confiável do método. Caminhos esses seriam:

- 1 2 3 4 7
- 1 2 5 6 7

Análise do fluxo de dados de método *verDadosMulta()*

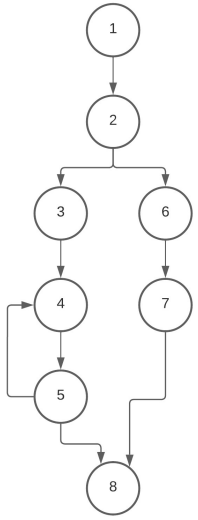
Tomando como base o código correspondente ao método em questões, a análise do fluxo de dados resultou em:

- **multa** : ~d → du → ii → uk

Nenhuma anormalidade observada.

Método **listaMultas() {}**

```
/*  1  */ public static void listaMultas(String classificacao){
/*  2  */         ArrayList<Multa> multas = new ArrayList<>();
/*  2  */         multas = buscaMultaClassificacao(classificacao);

/*  2  */         System.out.print("\n\tMultas " + classificacao + " -> ");
/*  3  */         if (multas != null){
/*  4  */                 for (Multa multa : multas) {
/*  5  */                         System.out.println(multa);
/*  5  */                 }
/*  6  */         } else {
/*  7  */                 System.out.println("\n\t\t Nenhuma multa cadastrada com essa
                          classificação!\n");
/*  7  */         }
/*  8  */ }
```



Análise do fluxo de controle do método *listaMultas()*

Considerando o grafo de fluxo controle apresentado anteriormente a respeito do método *listaMultas()* é possível realizar o cálculo da complexidade cíclomática.

- O grafo de fluxo contém **3 regiões;**
- V(G) = 9 - 8 + 2 = **3;**
- V(G) = 2 nós predicados + 1 = **3;**

A complexidade ciclomática do método *listaMultas()* é de 3 e com isso o seu valor nos fornece um limite superior para o número de caminhos independentes que podem formar o conjunto base, ou seja, 3 é número de testes que devem ser projetados e executados para garantir uma cobertura decente do método. Caminhos esses seriam:

- 1 2 3 4 5 8
- 1 2 6 7 8
- 1 2 3 4 5 4 5 8

Análise do fluxo de dados de método *listaMultas()*

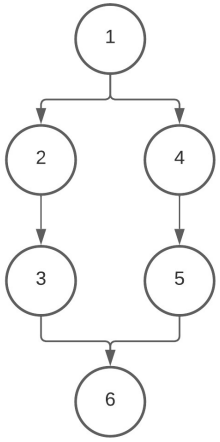
Tomando como base o código correspondente ao método em questões, a análise do fluxo de dados resultou em:

- **multas** : ~d → dd → du → uu → uk
- **multa** : ~d → du → uk
- **classificação** : ~d → du → uk

Nenhuma anormalidade observada.

Método **criarRelatorio() {}**

```
/*  1  */ public static boolean criarRelatorio(Administrador autor){
/*  2  */         Relatorio relatorio = new Relatorio(autor);

/*  2  */         if (insereRelatorio(relatorio)){
/*  3  */                 System.out.println("\n\t\tRelatorio -> " + relatorio);
/*  3  */                 return true;
/*  4  */         } else {
/*  5  */                 return false;
/*  5  */         }
/*  6  */ }
```



Análise do fluxo de controle do método *criarRelatorio()*

Considerando o grafo de fluxo controle apresentado anteriormente a respeito do método *criarRelatorio()* é possível realizar o cálculo da complexidade cíclomática.

- O grafo de fluxo contém **2 regiões;**
- V(G) = 6 - 6 + 2 = **2;**
- V(G) = 1 nós predicados + 1 = **2;**

A complexidade ciclomática do método *criarRelatorio()* é de 2 e com isso o seu valor nos fornece um limite superior para o número de caminhos independentes que podem formar o conjunto base, ou seja, 2 é número de testes que devem ser projetados e executados para garantir uma cobertura decente do método. Caminhos esses seriam:

- 1 2 3 6
- 1 4 5 6

Análise do fluxo de dados de método *criarRelatorio()*

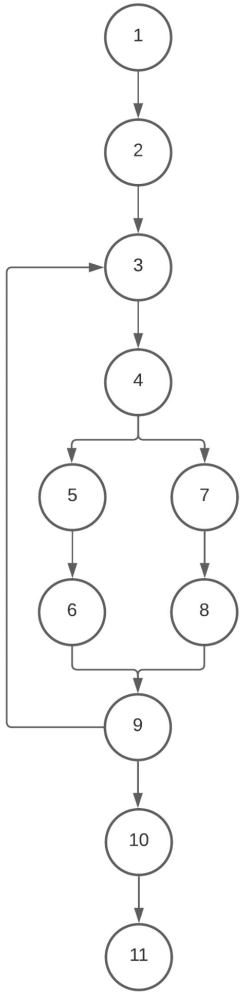
Tomando como base o código correspondente ao método em questões, a análise do fluxo de dados resultou em:

- **relatorio** : ~d → du → uu → uk

Nenhuma anormalidade observada.

Método **logar() {}**

```
/*  1  */ public static Administrador logar(){
/*  2  */          String login = "";
/*  2  */          String senha = "";

/*  2  */          int tentativas = 0;
/*  2  */          boolean acesso = false;

/*  3  */          do {
/*  4  */                  System.out.print("\t- Login: ");
/*  4  */                  login = input.nextLine();

/*  4  */                  System.out.print("\t- Senha: ");
/*  4  */                  senha = input.nextLine();

/*  4  */                  String senhaCodificada =
                           Objects.requireNonNull(buscaAdministrador(login)).getSenha();

/*  5  */                  if(!senhaCodificada.isEmpty() && BCrypt.checkpw(senha,
                           senhaCodificada)){
/*  6  */                          System.out.println("\n\t\t Logando no sistema!");
/*  6  */                          return buscaAdministrador(login);
/*  7  */                  } else {
/*  8  */                          System.out.println("\n\t\t Credenciais incorretas! \n");
/*  8  */                          tentativas++;
/*  8  */                  }
/*  9  */          } while (tentativas < 3 && !acesso);

/* 10  */          System.out.println("\n\t\t Máximo de tentativas atingido.\n\t\t\t  Acesso
                   bloqueado!!!\n\n");
/* 10  */          System.exit(0);

/* 10  */          return null;
/* 11  */ }
```



Análise do fluxo de controle do método *logar()*

Considerando o grafo de fluxo controle apresentado anteriormente a respeito do método *logar()* é possível realizar o cálculo da complexidade cíclomática.

- O grafo de fluxo contém **3 regiões;**
- V(G) = 12 - 11 + 2 = **3;**
- V(G) = 2 nós predicados + 1 = **3;**

A complexidade ciclomática do método *logar()* é de 3 e com isso o seu valor nos fornece um limite superior para o número de caminhos independentes que podem formar o conjunto base, ou seja, 3 é número de testes que devem ser projetados e executados para garantir uma cobertura decente do método. Caminhos esses seriam:

- 1 2 3 4 5 6 9 10 11
- 1 2 3 4 7 8 9 10 11
- 1 2 3 4 7 8 9 3 4 7 8 9 10 11

Análise do fluxo de dados de método cadastrarAdministrador*()*

Tomando como base o código correspondente ao método em questões, a análise do fluxo de dados resultou em:

- **login** : ~d → dd → du → uu → uk
- **senha** : ~d → dd → du → uk
- **senhaCodificada** : ~d → du → uk
- **tentativas** : ~d → du → uu → uk
- **acesso** : ~d → du → uk

Apesar de não ser possível observar nenhuma anormalidade tomando como base no esquema acima, através da análise do código, a variável acesso não tem nenhuma importância.

Método **cadastrarAdministrador() {}**

```
/*  1  */ public static boolean cadastrarAdministrador(@NotNull Administrador
          administrador){
/*  2  */        if(administrador.getId() == 1){
/*  3  */                String nome = "";
/*  3  */                String email = "";
/*  3 */                 String login = "";
/*  3  */                String senha = "";

/*  3  */                do {
/*  4  */                        System.out.print("\n\t- Informe o nome do administrador: ");
/*  4  */                        nome = input.nextLine();

/*  5  */                        if(nome.isEmpty()) {
/*  6  */                            System.out.println("\n\t\t É necessário informar o nome
                                      do administrador!\n");
/*  7  */                        } else if(nome.length() > 80){
/*  8  */                            System.out.println("\n\t\t O nome do administrador deve
                                      ter no máximo 80 caracteres!\n");
/*  8  */                        }

/*  9  */                } while (nome.isEmpty() || nome.length() > 80);

/* 10  */                do {
/* 11  */                        System.out.print("\t- Informe um email para a conta: ");
/* 11  */                        email = input.nextLine();

/* 12  */                        if(email.isEmpty()) {
/* 13  */                                System.out.println("\n\t\t É necessário informar um
                                          email para a conta!\n");
/* 14  */                        } else if(email.length() > 80){
/* 15  */                                System.out.println("\n\t\t O email do administrador deve
                                          ter no máximo 80 caracteres!\n");
/* 15  */                        }

/* 16  */                } while (email.isEmpty() || email.length() > 80);

/* 17  */                do {
/* 18  */                        System.out.print("\t- Informe um login para a conta: ");
/* 18  */                        login = input.nextLine();

/* 19  */                        if(login.isEmpty()) {
/* 20  */                                System.out.println("\n\t\t É necessário informar um
                                          login para a conta!\n");
/* 21  */                        } else if(login.length() > 30){
/* 22  */                                System.out.println("\n\t\t O login do administrador
                                          deve ter no máximo 30 caracteres!\n");
/* 22  */                        }

/* 23  */                } while (login.isEmpty() || login.length() > 30);

/* 24  */                do {
/* 25  */                        System.out.print("\t- Informe uma senha para a conta: ");
/* 25  */                        senha = input.nextLine();

/* 26  */                        if(senha.length() < 8) {
/* 27  */                                System.out.println("\n\t\t A senha deve conter no
                                          mínimo 8 dígitos!\n");
/* 27  */                        }
/* 28  */                } while (senha.length() < 8);

/* 29  */                senha = BCrypt.hashpw(senha, BCrypt.gensalt());
/* 29  */                Administrador adm = new Administrador(nome, email, login,
                          senha);

/* 29  */                return insereAdministrador(adm);
/* 30  */        } else {
/* 31  */                System.out.println("\n\t\t Você não possui permissão para
                          isso!\n");
/* 31  */                return false;
/* 31  */        }
/* 32  */ }
```

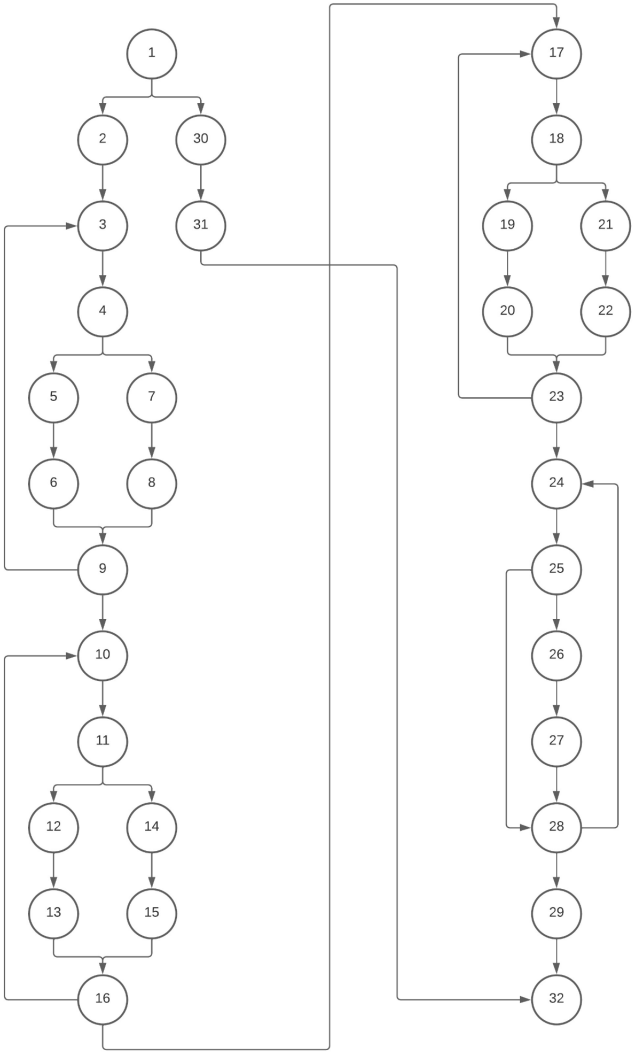
Análise do fluxo de controle do método cadastrarAdministrador()

Considerando o grafo de fluxo controle apresentado anteriormente a respeito do método *cadastrarAdministrador()* é possível realizar o cálculo da complexidade cíclomática.

- O grafo de fluxo contém **10 regiões;**
- V(G) = 40 - 32 + 2 = **10;**
- V(G) = 9 nós predicados + 1 = **10;**

A complexidade ciclomática do método *cadastrarAdministrador()* é de 10 e com isso o seu valor nos fornece um limite superior para o número de caminhos independentes que podem formar o conjunto base, ou seja, 10 é número de testes que devem ser projetados e executados para garantir uma sensação de cobertura boa do método. Caminhos esses seriam:

- 1 2 3 4 5 6 9 10 11 12 13 16 17 18 19 20 23 24 25 26 7 28 29 32
- 1 2 3 4 5 6 9 10 11 12 13 16 17 18 19 20 23 24 25 26 27 28 24 25 28 29 32
- 1 2 3 4 5 6 9 10 11 12 13 16 17 18 19 20 23 24 25 28 29 32
- 1 2 3 4 5 6 9 10 11 12 13 16 17 18 19 20 23 17 18 19 20 23 24 25 28 29 32
- 1 2 3 4 5 6 9 10 11 12 13 16 17 18 21 22 23 24 25 28 29 32
- 1 2 3 4 5 6 9 10 11 12 13 16 10 11 12 13 16 17 18 21 22 23 24 25 28 29 32
- 1 2 3 4 5 6 9 10 11 14 15 16 17 18 21 22 23 24 25 28 29 32



- 1 2 3 4 5 6 9 3 4 5 6 9 10 11 12 13 16 17 18 21 22 23 24 25 28 29 32
- 1 2 3 4 7 8 9 10 11 12 13 16 17 18 21 22 23 24 25 28 29 32
- 1 30 31 32

Análise do fluxo de dados de método cadastrarAdministrador*()*

Tomando como base o código correspondente ao método em questões, a análise do fluxo de dados resultou em:

- **nome** : ~d → dd → du → uu → uu → uu → uu → uk
- **email** : ~d → dd → du → uu → uu → uu → uu → uk
- **login** : ~d → dd → du → uu → uu → uu → uu → uk
- **senha** : ~d → dd → du → uu → uu → uu → ud → du → uk
- **adm :** ~d → du → uk

Nenhuma anormalidade

Método **cadastrarAutuação() {}**

```
/*  1  */ public static boolean cadastrarAutuacao(Administrador autor){
/*  2  */         int id = 0;
/*  2  */         int opcao = 0;

/*  2  */         String codigo = "";
/*  2  */         String placa = "";
/*  2  */         String renavam = "";
/*  2  */         String classificacao = "";
/*  2  */         String estado = "";
/*  2  */         String municipio = "";

/*  2  */         Veiculo veiculo = null;
/*  2  */         Multa multa = null;

/*  2  */         System.out.println("\nInforme a Placa e o Renavam do veículo multado:");

/*  3  */         do {
/*  4  */                 do {
/*  5  */                         System.out.print("\t- Escreva a PLACA do veículo: ");
/*  5  */                         placa = input.nextLine().toUpperCase();

/*  5  */                         Pattern letras = Pattern.compile("^[A-Z]+$");
/*  5  */                         Pattern nums = Pattern.compile("^[0-9]+$");

/*  6  */                         if(placa.length() == 7){
/*  7  */                                 if(!letras.matcher(placa.substring(0, 3)).find() &&
                                        !nums.matcher(("" + placa.charAt(3))).find() &&
                                        !letras.matcher(("" + placa.charAt(4))).find() &&
                                        !nums.matcher(placa.substring(5, 7)).find()){
/*  8  */                                         System.out.println("\n\t\t Formato da PLACA do
                                                veículo incorreta!\n");
/*  8  */                                         placa = "";
/*  8  */                                 }
/*  9  */                         } else {
/* 10  */                                 System.out.println("\n\t\t A PLACA do veículo deve ser
                                                formada por 7 dígitos!\n");
/* 10  */                         }

/* 11  */                 } while (placa.length() != 7);

/* 12  */                 do {
/* 13  */                         System.out.print("\t- Informe o RENAVAM do veículo: ");
/* 13  */                         renavam = input.nextLine();

/* 13  */                         Pattern letras = Pattern.compile("[a-zA-Z]+");

/* 14 */                         if(renavam.length() == 9 || renavam.length() == 11) {
/* 15  */                                 if (letras.matcher(renavam).find()){
/* 16  */                                         System.out.println("\n\t\t O RENAVAM deve conter
                                                apenas números!\n");
/* 16  */                                         renavam = "";
/* 16  */                                 }
/* 17  */                         } else {
/* 18  */                                 System.out.println("\n\t\t É necessário informar o
                                                RENAVAM válido (com 9 ou 11 algarismos)!\n");
/* 18  */                         }

/* 19  */                 } while (renavam.length() != 9 && renavam.length() != 11);

/* 20  */                 veiculo = buscaVeiculo(placa, renavam);

/* 21  */                 if (veiculo == null){
/* 22  */                         System.out.println("\n\t\t Esse veículo não existe!!!\n");
/* 22  */                 }

/* 23  */         } while (veiculo == null);

/* 24  */         System.out.println(veiculo);

/* 24  */         System.out.println("\nAgora informe a multa a ser inserida no veículo
                multado:");
/* 25  */         do {
/* 26  */                 System.out.println("\t- Informe a classificação da multa: \n");
/* 26  */                 System.out.print("\t\t1 - LEVE\n\t\t2 - MÉDIA\n\t\t3 - GRAVE\n\t\t4 -
                        GRAVÍSSIMA\n\t Opção: ");

/* 27  */                 try {
/* 28  */                         opcao = Integer.parseInt(input.nextLine());
```

```
/* 29 */                    switch (opcao) {
/* 30 */                        case 1:
/* 31 */                            classificacao = "LEVE";
/* 31 */                            break;
/* 32 */                        case 2:
/* 33 */                            classificacao = "MÉDIA";
/* 33 */                            break;
/* 34 */                        case 3:
/* 35 */                            classificacao = "GRAVE";
/* 35 */                            break;
/* 36 */                        case 4:
/* 37 */                            classificacao = "GRAVÍSSIMA";
/* 37 */                            break;
/* 38 */                        default:
/* 39 */                            System.out.println("\n\t\t Opção inválida!\n");
/* 39 */                            break;
/* 39 */                    }
/* 40 */                } catch (NumberFormatException e) {
/* 41 */                    System.out.println("\n\t\t O campo opção aceita somente
                            números!\n");
/* 41 */                }

/* 42 */            } while (classificacao.isEmpty());

/* 43 */            listaMultas(classificacao);

/* 44 */            do {
/* 45 */                do {
/* 46 */                    String aux1 = "";
/* 46 */                    String aux2 = "";

/* 47 */                    try {
/* 48 */                        System.out.print("\n\t- Informe o código da multa
                                (apenas números): ");
/* 48 */                        id = Integer.parseInt(input.nextLine());

/* 48 */                        codigo = "" + id;
/* 48 */                        aux1 = codigo.substring(0, 3);
/* 48 */                        aux2 = codigo.substring(3, 5);

/* 48 */                        codigo = aux1 + "-" + aux2;

/* 49 */                        if (buscaMulta(codigo) == null) {
/* 50 */                            System.out.println("\n\t\t Não existe nenhuma
                                    multa com esse código!");
/* 50 */                            codigo = "";
/* 50 */                        }
/* 51 */                    } catch (NumberFormatException e) {
/* 52 */                        System.out.println("\n\t\t O campo código aceita
                                somente números!");
/* 52 */                    }
/* 53 */                } while (codigo.isEmpty());

/* 54 */                multa = buscaMulta(codigo);

/* 55 */                if (multa == null){
/* 56 */                    System.out.println("\n\t\t Essa multa não existe!!!\n");
/* 56 */                }

/* 57 */            } while (multa == null);

/* 58 */            do {
/* 59 */                System.out.print("\t- Informe o estado da autuação de trânsito: ");
/* 59 */                estado = input.nextLine();

/* 60 */                if (estado.isEmpty()) {
/* 61 */                    System.out.println("\n\t\t É necessário informar a estado da
                            autuação!\n");
/* 61 */                }
/* 62 */            } while (estado.isEmpty());

/* 63 */            do {
/* 64 */                System.out.print("\t- Informe o município da autuação de trânsito: ");
/* 64 */                municipio = input.nextLine();

/* 65 */                if (municipio.isEmpty()) {
/* 66 */                    System.out.println("\n\t\t É necessário informar o município da
                            autuação!\n");
/* 66 */                }
```

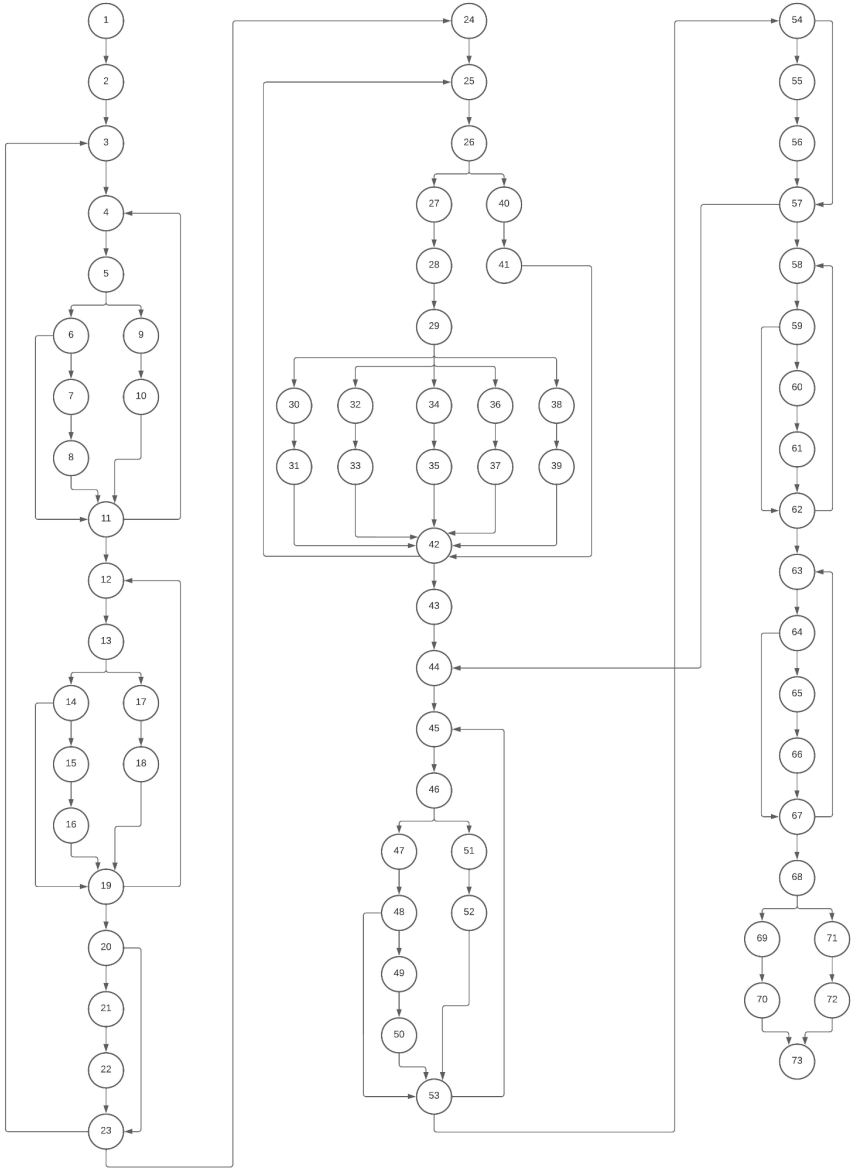
```
/* 67  */        } while (municipio.isEmpty());

/* 68  */        AutuacaoTransito autuacao = new AutuacaoTransito(autor, veiculo, multa,
                 estado, municipio);

/* 69  */        if (insereAutuacao(autuacao)){
/* 70  */                autuacao.setId(buscaIdAutuacao(autor.getId(),
                          autuacao.getDataHora()));
/* 70  */                autor.setAcoes(autuacao);
/* 70  */                return true;
/* 71  */        } else {
/* 72  */                return false;
/* 72  */        }
/* 73  */ }
```

Análise do fluxo de controle do método cadastrarAutuação()

Considerando o grafo de fluxo controle apresentado anteriormente a respeito do método *cadastrarAutuação()* é possível realizar o cálculo da complexidade cíclomática.

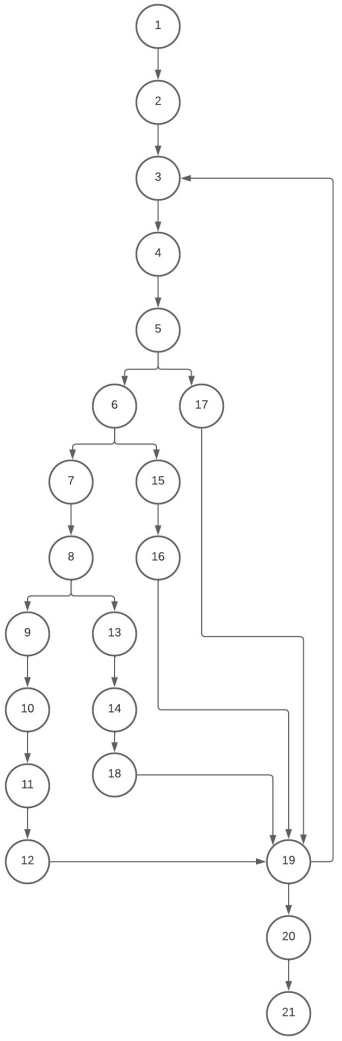
- O grafo de fluxo contém **26 regiões;**
- V(G) = 97 - 73 + 2 = **26;**
- V(G) = 25 nós predicados + 1 = **26;**

A complexidade ciclomática do método *cadastrarAutuação()* é de 26 e com isso o seu valor nos fornece um limite superior para o número de caminhos independentes que podem formar o conjunto base, ou seja, 26 é número de testes que devem ser projetados e executados para garantir uma sensação de cobertura boa do método.



Método **removerAutuação() {}**

```
/*  1  */ public static boolean removerAutuacao(@NotNull Administrador autor){
/*  2  */          int id = 0;
/*  2  */          int opcao = 0;
/*  2  */          boolean valid = true;

/*  2  */          AutuacaoTransito autuacao = null;
/*  2  */          LinkedList<AutuacaoTransito> acoes = autor.getAcoes();
/*  2  */          System.out.println("\nInforme o Id da autuação de trânsito que deseja
                    excluir:");
/*  3  */          do {
/*  4  */                  System.out.print("\t- Escreva o ID da autuacão: ");
/*  5  */                  try {
/*  6  */                          id = Integer.parseInt(input.nextLine());
/*  6  */                          autuacao = buscaAutuacao(id);
/*  7  */                          if(autuacao != null){
/*  8  */                                  System.out.println("\n\tAutuação de trânsito -> " +
                                         autuacao);
/*  8  */                                  System.out.print("\n\t\t Deseja mesmo desvincular essa
                                         autuação (1 - Sim, 0 - Não)? ");
/*  8  */                                  opcao = Integer.parseInt(input.nextLine());

/*  9  */                                  if(opcao == 1){
/* 10  */                                          valid = false;
/* 11  */                                          if(acoes != null) {
/* 12 */                                                   acoes.remove(autuacao);
/* 12 */                                           }
/* 13 */                                   } else {
/* 14 */                                           System.out.println("\n\n");
/* 14  */                                   }
/* 15  */                          } else {
/* 16  */                                  System.out.println("\n\t\t Autuação de trânsito com ID "
                                         + id + " não existe! \n");
/* 16  */                          }
/* 17 */                  } catch (NumberFormatException e){
/* 18  */                          System.out.println("\n\t\t O campo opção aceita somente
                                    números!\n");
/* 18  */                  }
/* 19  */          } while (valid);

/* 20  */          return removeAutuacao(id);
/* 21  */ }
```



Análise do fluxo de controle do método removerAutuação()

Considerando o grafo de fluxo controle apresentado anteriormente a respeito do método *removerAutuação()* é possível realizar o cálculo da complexidade cíclomática.

- O grafo de fluxo contém **5 regiões;**
- V(G) = 24 - 21 + 2 = **5;**
- V(G) = 4 nós predicados + 1 = **5;**

A complexidade ciclomática do método *removerAutuação()* é de 5 e com isso o seu valor nos fornece um limite superior para o número de caminhos independentes que podem formar o conjunto base, ou seja, 5 é número de testes que devem ser projetados e executados para garantir a abrangência de todos os comandos do programa. Caminhos esses seriam:

- 1 2 3 4 5 6 7 8 9 10 11 12 19 20 21
- 1 2 3 4 5 6 7 8 13 14 16 19 20 21
- 1 2 3 4 5 6 15 16 19 20 21
- 1 2 3 4 5 17 19 20 21
- 1 2 3 4 5 17 19 3 4 5 17 19 20 21

Análise do fluxo de dados de método removerAutuação*()*

Tomando como base o código correspondente ao método em questões, a análise do fluxo de dados resultou em:

- **id** : ~d → dd → du → uu → uu → uk
- **opcao** : ~d → dd → du → uk
- **valid** : ~d → dd → du → uk
- **autuacao** : ~d → dd → du → uu → uk
- **acoes**: ~d → du → uk

Nenhuma anormalidade.