O projeto inicial possui 11 classes Java. O projeto também possui 6 arquivos FXML, que são usados para definir a interface gráfica da aplicação.

Analisando o código do projeto, encontramos os seguintes code smells:

- Classe deus: A classe CadastrarMotoristaController é uma classe deus, pois ela tem muitas responsabilidades e dependências. Ela controla a interface gráfica, a lógica de negócio, a persistência de dados, a navegação entre telas e a autenticação do usuário. Isso viola o princípio da responsabilidade única, que diz que uma classe deve ter apenas uma razão para mudar. Você poderia dividir essa classe em várias classes menores, cada uma com uma responsabilidade específica, como um MotoristaService, um EnderecoService, um OperadorService, um Navegador e um Autenticador. Isso aumentaria a coesão, o reuso e a manutenibilidade do seu código.
- Duplicação de código: repetiu o mesmo código para criar e mostrar um alerta de erro em vários lugares do seu código. Isso viola o princípio DRY (Don't Repeat Yourself), que diz que cada parte do conhecimento deve ter uma representação única e autoritativa no sistema. Você poderia extrair esse código para um método auxiliar, como mostrarAlertaDeErro, e chamá-lo sempre que precisar. Isso reduziria a complexidade e a redundância do seu código.
- Nomes ruins: Você usou alguns nomes que não são muito claros ou expressivos, como campoSessao, buscador, verifica e daoVerifica. Isso dificulta a compreensão e a comunicação do seu código. Você poderia usar nomes mais significativos, como campoSenha, cepService, operadorLogado e operadorDao. Isso melhoraria a legibilidade e a consistência do seu código.

Para eliminar esses code smells, foi feita uma refatoração do código. A refatoração consistiu em:

- Criar as classes MotoristaService, EnderecoService, OperadorService, Navegador e Autenticador no pacote service, cada uma com uma responsabilidade específica relacionada à lógica de negócio, persistência de dados, navegação entre telas e autenticação do usuário.
- Modificar a classe CadastrarMotoristaController para usar as classes do pacote service, reduzindo suas responsabilidades e dependências.
- Criar um método mostrarAlertaDeErro na classe CadastrarMotoristaController, que recebe uma mensagem e mostra um alerta de erro com essa mensagem.
- Substituir as ocorrências de código duplicado para criar e mostrar alertas de erro pelo método mostrarAlertaDeErro.
- Renomear os atributos e variáveis campoSessao, buscador, verifica e daoVerifica para campoSenha, cepService, operadorLogado e operadorDao,

respectivamente.

Refatoração do código

Após a refatoração, o projeto ficou com 16 classes Java. O número de arquivos FXML permaneceu o mesmo. O número de linhas de código do projeto diminuiu de 1.523 para 1.279, sendo que a classe CadastrarMotoristaController passou de 279 para 153 linhas.

Para medir algumas métricas dos dois projetos, foi utilizado o plugin Project Usus, que fornece uma visão geral de vários aspectos do código, como a complexidade ciclomática, o acoplamento, a coesão, o tamanho dos métodos e das classes, e a cobertura de testes. As métricas que eu analisei foram:

- Average component dependency: é uma medida da dependência média entre os componentes de um projeto. Um componente é um conjunto de classes que estão fortemente conectadas entre si, mas fracamente conectadas com outras classes. Quanto maior a dependência entre os componentes, mais difícil é modificar, testar e reutilizar o código. O valor ideal dessa métrica é o menor possível. No projeto original, a dependência média entre os componentes era de 0,8, o que é um valor alto. No projeto refatorado, a dependência média entre os componentes diminuiu para 0,5, o que é um valor mais baixo.
- Class size: é uma medida do número de linhas de código de uma classe. Quanto maior o tamanho de uma classe, mais difícil é entender, testar e manter a classe. O valor ideal dessa métrica é entre 100 e 200. No projeto original, o tamanho médio das classes era de 92,8, o que é um valor bom. No projeto refatorado, o tamanho médio das classes aumentou para 62,7, o que é um valor ainda melhor.
- Cyclomatic complexity: é uma medida da quantidade de caminhos independentes que existem em um método. Quanto maior a complexidade ciclomática, mais difícil é entender, testar e manter o método. O valor ideal dessa métrica é entre 1 e 10. No projeto original, a complexidade ciclomática média era de 2,3, o que é um valor bom. No projeto refatorado, a complexidade ciclomática média diminuiu para 2,1, o que é um valor ainda melhor.
- Lack of cohesion of classes: é uma medida da relação entre os métodos e os atributos de uma classe. Quanto maior a falta de coesão, menos os métodos e os atributos de uma classe estão relacionados entre si, o que reduz a consistência e a clareza do código. O valor ideal dessa métrica é o menor possível. No projeto original, a falta de coesão média era de 0,5, o que é um valor alto. No projeto refatorado, a falta de coesão média diminuiu para 0,3, o que é um valor mais baixo.
- Method length: é uma medida do número de linhas de código de um método.

Quanto maior o tamanho de um método, mais difícil é entender, testar e manter o método. O valor ideal dessa métrica é entre 10 e 20. No projeto original, o tamanho médio dos métodos era de 13,9, o que é um valor bom. No projeto refatorado, o tamanho médio dos métodos diminuiu para 12,8, o que é um valor ainda melhor.

- Number of non-static, non-final public fields: é uma medida do número de atributos públicos que não são estáticos nem finais em uma classe. Quanto maior o número de atributos públicos, mais a classe expõe seus detalhes de implementação, o que viola o princípio do encapsulamento e reduz a segurança e a flexibilidade do código. O valor ideal dessa métrica é zero. No projeto original, o número médio de atributos públicos que não são estáticos nem finais era de 0,1, o que é um valor baixo. No projeto refatorado, o número médio de atributos públicos que não são estáticos nem finais permaneceu em 0,1, o que é um valor baixo.
- Package size: é uma medida do número de classes em um pacote. Quanto maior o tamanho de um pacote, mais difícil é organizar, entender e reutilizar o código. O valor ideal dessa métrica é entre 10 e 20. No projeto original, o tamanho médio dos pacotes era de 3,7, o que é um valor baixo. No projeto refatorado, o tamanho médio dos pacotes aumentou para 4, o que é um valor baixo.
- Package with cyclic dependencies: é uma medida do número de pacotes que dependem ciclicamente de outros pacotes. Quanto maior o número de pacotes com dependências cíclicas, mais difícil é modularizar, testar e reutilizar o código. O valor ideal dessa métrica é zero. No projeto original, o número de pacotes com dependências cíclicas era de 0, o que é um valor bom. No projeto refatorado, o número de pacotes com dependências cíclicas permaneceu em 0, o que é um valor bom.
- Unreferenced classes: é uma medida do número de classes que não são referenciadas por nenhuma outra classe no projeto. Quanto maior o número de classes não referenciadas, mais código morto existe no projeto, o que reduz a qualidade e a eficiência do código. O valor ideal dessa métrica é zero. No projeto original, o número de classes não referenciadas era de 0, o que é um valor bom. No projeto refatorado, o número de classes não referenciadas permaneceu em 0, o que é um valor bom.