Trabalho 3 - Programação Orientada a Objetos Avançada
UFSCar - ENPE 2020.1 - Bloco B
Prof. Daniel Lucrédio

Urna Eletrônica - Princípios SOLI

Guilherme Locca Salomão
Nathaelly Boni
Renan Dantas Pasquantonio

São Carlos - SP
01/2021

1 - Introdução

O Trabalho 3 consiste em projetar um sistema de *software* para uma urna eletrônica de acordo com os requisitos listados na descrição do projeto, atendendo aos princípios SOLI.
A descrição arquitetural da solução projetada foi realizada utilizando a notação UML, que está disponível para visualização no repositório mantido pelo grupo.
A seguir estão detalhadas as decisões tomadas dentro do projeto e como elas atendem aos princípios SOLI.

2 - Decisões de Projeto e Atendimento dos Princípios

2.1 - Princípio da Responsabilidade Única

O Princípio da Responsabilidade Única diz que uma classe deve ter uma, e apenas uma razão, para ser modificada. Na prática isso nos diz que as responsabilidades do projeto devem ser distribuídas para as entidades capazes de lidar com elas sem a necessidade de assistência externa.
Nesse projeto isso é visível pela separação das principais responsabilidades (enviar comandos, informar o eleitor, receber o voto, registrar o voto) sendo atendidas por diferentes classes. A classe abstrata *Comando* tratando os comandos enviados pelo mesário, *Display* cuidando de informar o eleitor sobre a votação, *RealizaVoto* e *Feedback* recebendo o voto do eleitor e gerando o *feedback* respectivamente, e por fim *RegistroVoto* que salva o voto do eleitor.

2.2 - Princípio Aberto-Fechado

O Princípio Aberto-Fechado diz que objetos ou entidades devem estar abertos para extensão, mas fechados para modificação. Isto é, não devemos alterar uma classe já existente para adicionar novos comportamentos que podem fugir do escopo de suas responsabilidades. Isso foi feito separando o comportamento extensível da classe por trás de uma interface.
Utilizaremos o segundo requisito e sua implementação dentro do projeto para fins de exemplo. O F2 solicita diferentes formas da informação ser exibida na tela, a classe abstrata *Display* que é responsável por isso é apenas uma interface para as diferentes classes que atendem a essa responsabilidade de formas distintas.
Dessa forma todas as diferentes implementações ficam fechadas para modificação, assim como a extensão de novas funcionalidades se dá apenas pela criação de uma nova classe. As classes *DisplayMente*, *EntradaVisual*, *FeedbackVibracao* e

ImprimeVotoBraille representam as possíveis extensões que podem ser solicitadas no projeto.

2.3 - Princípio da Substituição de Liskov

O Princípio da Substituição de Liskov prevê que uma classe derivada deve ser substituível por sua classe base. Isso quer dizer que: se S é uma subclasse de T, então todo objeto do tipo T pode ser substituído por um objeto do tipo S.
Esse princípio foi atendido em conjunto com o ISP para evitar problemas envolvendo generalizações. No projeto, um objeto da classe *DisplayTexto* pode ser instanciado dentro do projeto, imprimindo apenas texto no *display* da urna. A classe *DisplayFoto* é capaz de imprimir texto e imagem no *display* da urna. A substituição de um objeto da classe *DisplayTexto* por um da classe *DisplayFoto* não resultaria em resultados inesperados.

2.4 - Princípio da Segregação de Interface

O Princípio da Segregação de Interface diz que uma classe não deve ser forçada a implementar interfaces que não serão utilizadas. Isto é, é preferível criar interfaces específicas do que uma interface genérica.
Isso pode ser observado nas interfaces criadas para esse projeto, onde todas possuem apenas as funcionalidades utilizadas por todos suas subclasses e consequentemente isolando os comportamentos da classe. Isso pode ser observado principalmente na interface *RegistroVoto* que possui interfaces para poder abrigar diferentes estratégias para as mesmas formas de registro, e consequentemente respeitando o Princípio da Segregação de Interface.

3 - Conclusão

É possível notar melhorias no projeto referente à aplicação de cada um dos princípios. Após a aplicação dos quatro princípios solicitados, é possível ver como o projeto possui suas responsabilidades bem separadas, poucos casos de acoplamentos, além de tornar fácil a refatoração e o reaproveitamento de código. Tornando assim o projeto mais robusto, escalável e flexível, facilitando eventuais mudanças, implementação de novos recursos e uma melhor manutenção do código.

4 - Referências

- O que é SOLID: O guia completo para você entender os 5 princípios da POO
- The Principles of OOD
- SOLID
- Writing Flexible Code with the Single Responsibility Principle
- Maintainable Code and the Open-Closed Principle
- Making the Most of Polymorphism with the Liskov Substitution Principle
- Avoiding Interface Pollution with the Interface Segregation Principle