Laboratório 3.2 – Exercício
Princípios de PPOO – LSP e ISP

Objetivo

Exercícios de fixação para explorar o entendimento dos princípios.

Instruções

Exercício individual ou em duplas.

Estabelecendo regras para hierarquias de interfaces e classes

O princípio da Substituição de Liskov (LSP) apresenta uma regra interessante para validar se as derivações de uma classe ou interface estão cumprindo os comportamentos esperados e planejados na concepção inicial da abstração.

Uma vez que uma violação da LSP seja identificada, a solução pode não envolver apenas a correção da implementação da classe derivada. Um problema com a própria abstração pode ter sido descoberto. Nesse momento, o design da hierarquia e suas dependências precisa ser revisto com cuidado, com foco em dois pontos de vista importantes:

- Ponto de vista dos clientes: objetos que usam (dependem) da abstração em questão.
- Ponto de vista das derivações: objetos que implementam a abstração em questão.

Do ponto de vista das derivações, as mesmas não deveriam ser obrigadas a implementar funções vazias ou funções não importantes ou fora de suas responsabilidades. Quando ocorre alguns desses tipos de situações, podem indicar que o modelo abstrato está inchado e deva ser segregado.

Do ponto de vista dos clientes, mudanças na hierarquia causam impacto e devem ser planejadas com cuidado. Uma refatoração é sempre bem vinda, desde que evolua o design para uma estrutura ainda mais coesa e com baixo acoplamento. Em casos de interfaces inchadas com muitos interesses, será provalvemente fácil distinguir nos clientes as partes que lidam apenas com determinados interesses da interface. A análise desses dois ponto de vista é essencial para uma boa refatoração.

Um dos princípios SOLID é o Princípio da Segregação de Interfaces (ISP) [1], que diz:

Os clientes não devem ser obrigados a depender de métodos que não utilizam.

Quando clientes estão acoplados a classes e métodos que não estão relacionados ao cerne de suas responsabilidades, mudanças nesses métodos podem afetar clientes. Isso gera risco ao desenvolvimento de software, pois mudanças em funcionalidades estão afetando outras funcionalidades não relacionadas.

[1] **Princípios, padrões e práticas ágeis em C#**. MARTIN, Robert. MARTIN, Micah. 1ª edição. Capítulos 12.

Exercício

Considere o código-fonte no arquivo “Recursos.zip”. O teste de unidade está falhando, pois uma classe derivada violou o LSP. A classe “ConfiguracoesEspeciais.java” não deve permitir que alterações sejam salvas. No projeto da hierarquia, planejou-se inicialmente que todos os recursos poderiam ser carregados, lidos, modificados e armazenados. Mas isso não se mostrou eficaz quando surgiu a necessidade de um recurso apenas de leitura (ConfiguracoesEspeciais.java).

Sua tarefa é refatorar o código para acomodar a nova mudança de forma adequada. Isso necessitará uma re-modelagem da interface iRecurso e suas derivações. Nesse processo, os testes também sofrerão refatoração.

Seu objetivo é planejar a aplicação para dar suporte a recursos modificáveis e recursos não-modificáveis. Uma informação revelante é que recursos não mudam de tipo (modificável ou não-modificável) durante tempo de execução.