+Devs2Blu

Linguagem de programação JAVA

Profª. Heloisa Moura

+Devs2Blu

Fundamentos avançados OOP

O que vamos ver:

- Princípio Aberto-Fechado (O - Open/Closed Principle)
- Princípio de Substituição de Liskov (L - Liskov Substitution Principle)

Princípio de Substituição de Liskov (L - Liskov Substitution Principle)

Princípio de Substituição de Liskov (L - Liskov Substitution Principle)

É uma variação do princípio aberto fechado.

- Afirma que "Se S é um subtipo de T, então os objetos do tipo T podem ser substituídos pelos objetos do tipo S sem alterar as propriedades desejáveis do programa.
- Em outras palavras, subclasses devem poder substituir suas classes base sem que o comportamento do programa seja modificado.

Temos o código:

Princípio de Substituição de Liskov (L - Liskov Substitution Principle)

Código 1: Exemplo onde não respeita o LSP.

```
class Retangulo {
    private int largura;
    private int altura;

    public void setLargura(int largura) {
        this.largura = largura;
    }

    public void setAltura(int altura) {
        this.altura = altura;
    }

    public int getArea() {
        return largura * altura;
    }
}
```

Princípio de Substituição de Liskov (L - Liskov Substitution Principle)

Código 1: Exemplo onde não respeita o LSP.

```
class Quadrado extends Retangulo {
    @Override
    public void setLargura(int largura) {
        super.setLargura(largura);
        super.setAltura(largura);
    }

    @Override
    public void setAltura(int altura) {
        super.setLargura(altura);
        super.setAltura(altura);
    }
}
```

O problema que temos: Quadrado quebra a definição do Retângulo, onde a largura e a altura podem ser diferentes.

Princípio de Substituição de Liskov (L - Liskov Substitution Principle)

Solução

Código 2: Refatorando para respeitar o LSP.

```
public interface Forma {
    int getArea();
}
```

Princípio de Substituição de Liskov (L - Liskov Substitution Principle)

Código 2: Refatorando para respeitar o LSP.

```
public class Retangulo implements Forma {

    private int largura;
    private int altura;

        public Retangulo(int largura, int altura) {
            this.largura = largura;
            this.altura = altura;
        }

        public int getArea() {
            return largura * altura;
        }
}
```

Princípio de Substituição de Liskov (L - Liskov Substitution Principle)

Código 2: Refatorando para respeitar o LSP.

```
public class Quadrado implements Forma {
    private int lado;

    public Quadrado(int lado) {
        this.lado = lado;
    }

    public int getArea() {
        return lado * lado;
    }
}
```

Agora, tanto **Retangulo** quanto **Quadrado** implementam a interface **Forma** de maneira independente e coerente, respeitando o LSP.

Princípio de Substituição de Liskov (L - Liskov Substitution Principle)

Outro exemplo para clarear mais as ideias:

Pensamos no seguinte cenário: o desenvolvimento de um sistema de uma faculdade.

Dentro do sistema, há uma classe-mãe **Estudante**, que representa um estudante de graduação, e a filha dela, **EstudantePosGraduacao**.

Temos o seguinte código:

Princípio de Substituição de Liskov (L - Liskov Substitution Principle)

```java
public class Estudante {
    String nome;

    public Estudante(String nome) {
        this.nome = nome;
    }

    public void estudar() {
        System.out.println(nome + " está estudando.");
    }
}
```

```java
public class EstudanteDePosGraduacao extends Estudante {


    public EstudanteDePosGraduacao(String nome) {
        super(nome);

    }

    @Override
    public void estudar() {
        System.out.println(nome + " está estudando e pesquisando.");
    }
}
```

Princípio de Substituição de Liskov (L - Liskov Substitution Principle)

Para adicionar a funcionalidade **entregarTCC()** ao sistema, basta colocar esse método na classe **Estudante**.

O código fica assim:

```
public class Estudante {
    String nome;

    public Estudante(String nome) {
        this.nome = nome;
    }

    public void estudar() {
        System.out.println(nome + " está estudando.");
    }

    public void entregarTCC(){
        //…
    }
}
```

Princípio de Substituição de Liskov (L - Liskov Substitution Principle)

Problemática:

Observando o código, percebemos algo. Normalmente, estudantes de pós-graduação não entregam TCCs.

- Só que a classe **EstudanteDePosGraduacao** é filha de Estudante, e portanto, deve apresentar todos os comportamentos dela.
- Uma alternativa seria sobrescrever o método **entregarTCC()** na classe **EstudanteDePosGraduacao** lançando uma exceção.

Princípio de Substituição de Liskov (L - Liskov Substitution Principle)

Problemática:

- No entanto, continuaria sendo problemático: a classe **EstudanteDePosGraduacao** ainda não teria os comportamentos iguais aos de Estudante.
- O ideal é que, nos lugares que estiver a classe Estudante, seja possível usar uma classe **EstudanteDePosGraduacao**, já que pela herança, um estudante de pós-graduação é um estudante.

Princípio de Substituição de Liskov (L - Liskov Substitution Principle)

Solução

Para este problema é modificar a nossa modelagem. Podemos criar uma nova classe **EstudanteDeGraduacao**, que também herdará de Estudante. Essa classe terá o método **entregarTCC().**

Vamos ver como fica o código na IDE

Resumo

- Aplicar esse princípio nos traz diversos benefícios, especialmente para ter uma modelagem mais fiel à realidade;
- Substituição Segura: As subclasses podem ser usadas no lugar da classe base sem quebrar o sistema.
- Reduzir erros inesperados no programa e simplificar a manutenção do código.

Pra não confundir: diferença entre os dois princípios OCP e LSP

Princípio	OCP (Open-Closed Principle)	LSP (Liskov Substitution Principle)
Foco	Foca na **extensibilidade** do sistema sem modificar o código existente.	Foca na **substituição** correta de classes base por suas subclasses.
Objetivo	Facilitar a adição de novas funcionalidades sem alterar o código já implementado.	Garantir que subclasses possam ser usadas de maneira intercambiável com a classe base.
Preocupação	Evitar a modificação do código base ao adicionar novos comportamentos.	Assegurar que as subclasses mantenham a coerência com o comportamento da classe base.
Exemplo de violação	Modificar a classe **`Círculo`** quando você quiser adicionar uma nova forma **`Triângulo`**.	Criar uma subclasse **`Pinguim`** de **`Pássaro`**, onde **`Pinguim`** não pode voar, mas **`Pássaro`** pode.

Princípio de Substituição de Liskov (L - Liskov Substitution Principle)

Exercícios