BSI-MC FACOM UFU

Programação Orientada a Objetos 2

Laboratório 1 - Propriedades e Princípios de Projetos

Profa. Fabíola S. F. Pereira

Instruções

- O objetivo desse laboratório é tornar o(a) aluno(a) familiarizado(a) com conceitos de Propriedades e Princípios de Projetos. Em especial, esse laboratório tem o foco nos princípios SOLID.

- As atividades deste laboratório **NÃO PRECISAM SER ENTREGUES**. Portanto, **não valerão nota**.

- É recomendável que você crie o hábito de armazenar seus códigos em uma ferramenta de versionamento (ex.: GitHub).

- A professora irá compartilhar as respostas de todas as questões desse laboratório após a aula, via MS Teams.

- Os códigos-fonte referentes aos enunciados deste laboratório estão disponíveis em: https://github.com/fabsfernandes/poo2-20241/

QUESTÃO 1

Como o nome sugere, o **Princípio da Responsabilidade Única** (Single Responsibility) enuncia que cada classe deve ter uma responsabilidade, um único propósito. Ou seja, a classe irá fazer um único trabalho, o que nos leva a concluir que ela deve mudar por uma única razão, caso necessário.

a) Execute o código abaixo e verifique por que ele fere o Princípio da Responsabilidade Única.

b) Refatore o código abaixo para que ele esteja compatível com o Princípio da Responsabilidade Única.

```java
package singleresponsibility;


public class SingleResponsibility {

    public SingleResponsibility() {
        doIncorrectImplementation();
        doCorrectImplementation();
    }

    public void doIncorrectImplementation() {
        /**
         * Considere uma classe que contenha informações sobre um livro.
         * A única função dessa classe deve ser conter dados relacionados a livros e operações relacionadas a livros.
         */
        singleresponsibility.incorrect.Book book = new singleresponsibility.incorrect.Book("Clean Architecture", "Robert
            Martin", "Science");

        /**
         * Ter um método que imprime o nome do autor nessa classe viola o Princípio da Responsabilidade Única
         */
        book.printAuthorName();
    }

    public void doCorrectImplementation() {

        /**
         * Adicione aqui seu código refatorado com referências a métodos e classes contidas no pacote
         singleresponsibility.correct
         */

    }

    public static void main(String [] args ) {
        SingleResponsibility sr = new SingleResponsibility();

    }

}
```

```java
package singleresponsibility.incorrect;


public class Book {

    private String title;
    private String author;
    private String genre;

    public Book(String title, String author, String genre) {
        this.title = title;
        this.author = author;
        this.genre = genre;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getAuthor() {
        return author;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    public String getGenre() {
        return genre;
    }

    public void setGenre(String genre) {
        this.genre = genre;
    }

    public void printAuthorName() {
        System.out.println("Nome do autor -> "+author);
    }

}
```

QUESTÃO 2

Entidades de software (classes, módulos, funções etc) devem ser abertas para extensão, mas fechadas para modificação. A ideia geral do Princípio Aberto-Fechado enuncia que você deve escrever seu código para que seja possível adicionar uma nova funcionalidade sem modificar o código existente.

a) Execute o código abaixo e verifique por que ele fere o Princípio Aberto-Fechado.

b) Refatore o código abaixo para que ele esteja compatível com o Princípio Aberto-Fechado.

```java
package openclosed;

public class OpenClosed {

    public OpenClosed(){
        doIncorrectImplementation();
        doCorrectImplementation();
    }

    public void doIncorrectImplementation(){

        /**
         * Aqui temos classes Rectangle e Circle que utilizam a classe GeometryOperation para calculo da area
         */
        openclosed.incorrect.Rectangle rectangle = new openclosed.incorrect.Rectangle(4,3);
        openclosed.incorrect.Circle circle = new openclosed.incorrect.Circle(5);

        /**
         * Toda vez que uma nova forma geometrica eh adicionada, precisaremos de uma nova logica para calculo da area
         * Como precisaremos mudar o codigo existente (GeometryOperation), essa abordagem fere o Principio do
         Aberto-Fechado
         */
        openclosed.incorrect.GeometryOperation op = new openclosed.incorrect.GeometryOperation();
        System.out.println("Area do retangulo -> " + op.getArea(rectangle));
        System.out.println("Area do circulo -> " + op.getArea(circle));
    }

    public void doCorrectImplementation(){

        /**
         * Adicione aqui seu código refatorado com referências a métodos e classes contidas no pacote openclosed.correct
         */
    }

    public static void main(String [] args ) {
        OpenClosed opClo = new OpenClosed();

    }
}
```

```java
package openclosed.incorrect;

public class Circle {

    private int radius;

    public Circle(int radius) {
        this.radius = radius;
    }

    public int getRadius() {
        return radius;
    }

}
```

```java
package openclosed.incorrect;

public class Rectangle {

    private int width;
    private int height;

    public Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }

    public int getWidth() {
        return width;
    }

    public int getHeight() {
        return height;
    }
}
```

```java
package openclosed.incorrect;

public class GeometryOperation {

    public int getArea(Object object) {
        if(object instanceof Rectangle) {
            Rectangle rectangle = (Rectangle) object;
            return rectangle.getHeight() * rectangle.getWidth();
        } else {
            Circle circle = (Circle) object;
            return (int) (Math.PI * circle.getRadius() * circle.getRadius());
        }
    }

}
```

QUESTÃO 3

Clientes não devem ser forçados a depender de interfaces que eles não usam. Similar ao Princípio da Responsabilidade Única, o objetivo do Princípio da Segregação de Interfaces é reduzir os efeitos colaterais e frequência das mudanças, dividindo o software em múltiplas partes independentes.

a) Execute o código abaixo e verifique por que ele fere o Princípio da Segregação de Interfaces.

b) Refatore o código abaixo para que ele esteja compatível com o Princípio da Segregação de Interfaces.

```java
package interfacesegregation;

public class InterfaceSegregation {

    public InterfaceSegregation() {
        doIncorrectImplementation();
        doCorrectImplementation();
    }


    public void doIncorrectImplementation() {
        /**
         * Crow e Penguin classes implementam a interface Bird
         */
        interfacesegregation.incorrect.Crow crow = new interfacesegregation.incorrect.Crow();
        interfacesegregation.incorrect.Penguin penguin = new interfacesegregation.incorrect.Penguin();

        /**
         * Os metodos funcionam bem para um passaro que pode comer, dormir e voar
         */
        crow.eat();
        crow.sleep();
        crow.fly();

        /**
         * Os metodos nao funcionam bem para o Penguin que pode comer e dormir, mas nao pode voar
         */
        penguin.eat();
        penguin.sleep();
        penguin.fly();
    }

    public void doCorrectImplementation() {
        /**
         * Adicione aqui seu código refatorado com referências a métodos e classes contidas no pacote
         interfacesegregation.correct
         */
    }

    public static void main(String [] args) {
        InterfaceSegregation is = new InterfaceSegregation();
    }

}
```

```java
package interfacesegregation.incorrect;

public interface Bird {
    public void eat();
    public void sleep();
    public void fly();
}
```

```java
package interfacesegregation.incorrect;

public class Crow implements Bird{

    @Override
    public void eat() {

    }

    @Override
    public void sleep() {

    }

    @Override
    public void fly() {

    }

}
```

```java
package interfacesegregation.incorrect;

public class Penguin implements Bird{

    @Override
    public void eat() {

    }

    @Override
    public void sleep() {

    }

    @Override
    public void fly() {
        throw new UnsupportedOperationException("Nao suportado");
    }
```