

MINISTERUL EDUCATIEI AL REPUBLICII MOLDOVA

UNIVERSITATEA TEHNICA A MOLDOVEI

FACULTATEA CALCULATOARE INFORMATICA SI MICROELECTRONICA

DEPARTAMENTUL INGINERIA SOFTWARE SI AUTOMATICA

RAPORT

Lucrarea de laborator nr.2

Disciplina: Analiza, proiectarea si programarea orientata pe obiecte

Tema: Pricipiile Single Responsibility si Interface Segregation

A efectuat
st.gr. TI-153

Popusoi Victor

A controlat
asis., univ.

Pecari Mihail

Chisinau 2018

1 Scopul lucrării de laborator

Implementarea principiilor SOLID, Single Responsibility si Interface Segregation

2 Obiectivele lucrării de laborator

Principiile SOLID:

- Single Responsibility;
- Interface Segregation;

3 Efectuarea lucrării de laborator

3.1 Sarcinile propuse pentru efectuare lucrării de laborator

Link-ul la repository *here*

1. Sa se implementeze principiile SOLID:

- a) Single Responsibility Principle
- b) Interface Segregation Principle

3.2 Realizarea lucrării de laborator

In urmatorul program au fost implementate principiile SOLID si anume Single Responsibility si Interface Segregation

```
interface ISquare {  
    SquareArea(a: number)  
}  
interface IRectangle {  
    RectanglePerimeter(a: number, b: number)  
}  
interface ICircle {  
    CircleArea(r: number)  
}  
interface ITriangle {  
    TrianglePerimeter(a: number, b: number, c: number)  
}  
class Square implements ISquare {  
    public a: number  
    constructor(a: number) {  
        this.a = a  
    }  
    SquareArea(a): void {  
        let result = a * a
```

```

        console.log("-----SQUARE-----")
        console.log("Square sides: a = " + a)
        console.log("Square area: " + result)
    }
}

class Rectangle implements IRectangle {
    public a: number
    public b: number
    constructor(a: number, b: number) {
        this.a = a
        this.b = b
    }
    RectanglePerimeter(a, b): void {
        let result = (2 * a) + (2 * b)
        console.log("-----RECTANGLE-----")
        console.log("Rectangle sides: a = " + a + ", b = " + b)
        console.log("Rectangle perimeter: " + result)
    }
}

class Circle implements ICircle {
    public r: number
    constructor(r: number) {
        this.r = r
    }
    CircleArea(r): void {
        let pi = 3.14
        let result = pi * Math.pow(r, 2)
        console.log("-----CIRCLE-----")
        console.log("Circle radius: r = " + r)
        console.log("Circle area: " + result)
    }
}

class Triangle implements ITriangle {
    public a: number
    public b: number
    public c: number
    constructor(a: number, b: number, c: number) {
        this.a = a
        this.b = a
        this.c = a
    }
}

```

```

    }
    TrianglePerimeter(a, b, c): void {
        let result = a + b + c
        console.log("-----TRIANGLE-----")
        console.log("Triangle sides: a = " + a + ", b = " + b + ", c = " + c)
        console.log("Triangle perimeter: " + result)
    }
}

class Type extends Triangle {
    public a: number
    public b: number
    public c: number
    TriangleType(a, b, c): void {
        if (a === b === c) {
            console.log("Triangle is equilateral")
        }
        else if (a !== b !== c) {
            console.log("Triangle is scalene")
        }
        else if (a === b || a === c || b === c) {
            console.log("Triangle is isosceles")
        }
    }
}

const square = new Square(5)
square.SquareArea(5)
const rectangle = new Rectangle(5, 7)
rectangle.RectanglePerimeter(5, 7)
const circle = new Circle(4)
circle.CircleArea(4)
const triangle = new Type(5, 6, 7)
triangle.TrianglePerimeter(5, 6, 7)
triangle.TriangleType(5, 6, 7)

```

Principiul Interface Segregation(ISP) prevede că niciun client nu ar trebui să fie obligat să depindă de metodele pe care nu le utilizează. ISP împarte interfețe care sunt foarte mari în interfețe mai mici și mai specifice, astfel încât clienții vor trebui doar să știe despre metodele care îi interesează. Astfel de interfețe sunt de asemenea numite interfețe de rol. ISP are rolul de a menține un sistem decuplat și, prin urmare, mai ușor de refăcut, de schimbare și de redistribuire

```

interface ISquare {

```

```

    SquareArea(a: number)
  }
  interface IRectangle {
    RectanglePerimeter(a: number, b: number)
  }
  interface ICircle {
    CircleArea(r: number)
  }
  interface ITriangle {
    TrianglePerimeter(a: number, b: number, c: number)
  }

```

Principiul Single Responsibility este un principiu de programare care prevede că fiecare modul sau clasă trebuie să aibă responsabilitatea asupra unei singure părți a funcționalității furnizate de software și că această responsabilitate ar trebui să fie în întregime încapsulată de clasă. Toate serviciile sale ar trebui să fie aliniate strâns la această responsabilitate. O clasă ar trebui să aibă doar un singur motiv să se schimbe.

```

class Square implements ISquare {
  public a: number
  constructor(a: number) {
    this.a = a
  }
  SquareArea(a): void {
    let result = a * a
    console.log("-----SQUARE-----")
    console.log("Square sides: a = " + a)
    console.log("Square area: " + result)
  }
}

class Rectangle implements IRectangle {
  public a: number
  public b: number
  constructor(a: number, b: number) {
    this.a = a
    this.b = b
  }
  RectanglePerimeter(a, b): void {
    let result = (2 * a) + (2 * b)
    console.log("-----RECTANGLE-----")
    console.log("Rectangle sides: a = " + a + ", b = " + b)
  }
}

```

```

        console.log("Rectangle perimeter: " + result)
    }
}
class Circle implements ICircle {
    public r: number
    constructor(r: number) {
        this.r = r
    }
    CircleArea(r): void {
        let pi = 3.14
        let result = pi * Math.pow(r, 2)
        console.log("-----CIRCLE-----")
        console.log("Circle radius: r = " + r)
        console.log("Circle area: " + result)
    }
}
class Triangle implements ITriangle {
    public a: number
    public b: number
    public c: number
    constructor(a: number, b: number, c: number) {
        this.a = a
        this.b = a
        this.c = a
    }
    TrianglePerimeter(a, b, c): void {
        let result = a + b + c
        console.log("-----TRIANGLE-----")
        console.log("Triangle sides: a = " + a + ", b = " + b + ", c = " + c)
        console.log("Triangle perimeter: " + result)
    }
}
class Type extends Triangle {
    public a: number
    public b: number
    public c: number
    TriangleType(a, b, c): void {
        if (a === b === c) {
            console.log("Triangle is equilateral")
        }
    }
}

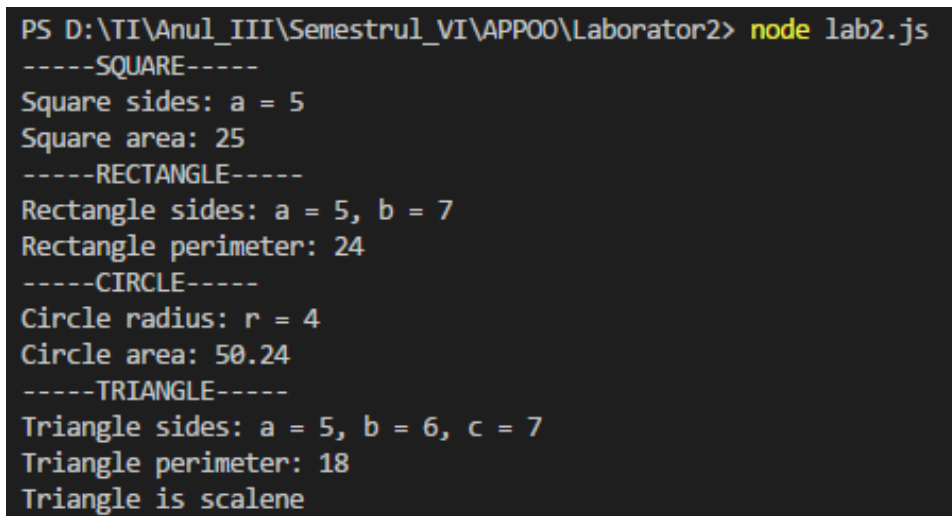
```

```

    else if (a != b != c) {
        console.log("Triangle is scalene")
    }
    else if (a == b || a == c || b == c) {
        console.log("Triangle is isosceles")
    }
}
}

```

3.3 Imagini



```

PS D:\TI\Anul_III\Semestrul_VI\APP00\Laborator2> node lab2.js
-----SQUARE-----
Square sides: a = 5
Square area: 25
-----RECTANGLE-----
Rectangle sides: a = 5, b = 7
Rectangle perimeter: 24
-----CIRCLE-----
Circle radius: r = 4
Circle area: 50.24
-----TRIANGLE-----
Triangle sides: a = 5, b = 6, c = 7
Triangle perimeter: 18
Triangle is scalene

```

Figure 3.1 – Final result

Concluzie

În lucrarea dată, au fost implementate 2 principii SOLID precum Single Responsibility principle și Interface Segregation principle. Importanța SOLID provine din faptul că este practic imposibil să scriem un cod "curat" dacă nu stim ce diferențiază un cod "curat" de un cod "murdar". Înțelegerea acestor cinci principii ne permite să analizăm codul folosind modele standard și terminologii și să evaluăm calitativ "curățenia" codului.

Prin urmare, este important să înțelegem că respectarea acestor principii nu ne va face în mod automat codul curat, însă dacă vom urma principiile SOLID, putem produce coduri care sunt mai flexibile și mai robuste și care au o posibilitate mai mare de reutilizare.

Bibliografie

1. Single Responsibility Principle [Resursă electronică]. Regim de acces:
<http://www.oodeesign.com/single-responsibility-principle.html>
2. Interface Segregation Principle [Resursă electronică]. Regim de acces:
<http://www.oodeesign.com/interface-segregation-principle.html>