Ministerul Educatiei al Republicii Moldova

Universitatea Tehnica a Moldovei

Facultatea Calcuatoare 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 lucrarii de laborator

Implementarea pricipiilor SOLID, Single Responsibility si Interface Segregation

2 Obiectivele lucrarii de laborator

Principiile SOLID:
- Single Resonsibility;
- Interface Segregation;

3 Efectuarea lucrarii de laborator

3.1 Sarcinile propuse pentru efectuare lucrarii de laborator

Link-ul la repozitoriu *here*
1. Sa se implementeze principiile SOLID:
    a) Single Responsibility Principle
    b) Interface Segregation Principle

3.2 Realizarea lucrarii de laborator

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

```
interface IComputerDescription {
    GetDescription();
    GetColor();
}
interface IComputerPrice {
    GetPrice();
}
interface IPriceCalculation {
    CalculatePriceWithDiscount(calculate : IComputerPrice);
}
class Desktop implements IComputerDescription, IComputerPrice {
    public desktopPrice: number
    constructor(desktopPrice : number) {
        this.desktopPrice = desktopPrice
    }
    public GetDescription(): string{
        return "Calculator Desktop";
    }
    public GetColor(){
```

```
        return "Negru";
    }
    public GetPrice(){
        return this.desktopPrice;
    }
}
class Laptop implements IComputerDescription, IComputerPrice {
    public laptopPrice : number
    constructor(laptopPrice : number) {
        this.laptopPrice = laptopPrice
    }
    public GetDescription(): string {
        return "Laptop"
    }
    public GetColor(): string {
        return "Negru, alb"
    }
    public GetPrice(): number {
        return this.laptopPrice
    }
}
class GiftItem implements IComputerDescription {
    public GetDescription(): string {
        return "Stick USB"
    }
    public GetColor(): string {
        return "Rosu, negru, alb"
    }
}
class CalculateComputerPrice implements IPriceCalculation {
    public CalculatePriceWithDiscount(calculate : IComputerPrice) : any {
        return calculate.GetPrice() - calculate.GetPrice() * 0.1;
    }
}
class Shop {
    public GetMyComputer(cmptype : IComputerDescription) {
        var myComp = cmptype.GetDescription();
        return myComp;
    }
    public GetMyComputerPrice(cmpCal : IPriceCalculation, cmpPrice : IComput
```

```
        var myCompprice = "Pretul este " + cmpCal.CalculatePriceWithDiscount(
        return myCompprice;
    }
    public GetAvailableColor(cmptype : IComputerDescription): any {
        var myCompcolor = cmptype.GetColor();
        return myCompcolor;
    }
    public WhatIsTheColorOfGiftItem(cmptype : IComputerDescription) {
        return this.GetAvailableColor(cmptype);
    }
    public IsThereAnyGiftItem(gftType : IComputerDescription) {
        return this.GetMyComputer(gftType);
    }
}
const queryForLaptop = new Laptop(18999);
const queryForDesktop = new Desktop(30000);
const queryForGiftItem = new GiftItem();
const computerShop = new Shop();
const responsibleForCalculation = new CalculateComputerPrice();
const laptopType = computerShop.GetMyComputer(queryForLaptop);
const desktopType = computerShop.GetMyComputer(queryForDesktop);
const priceAnswerLaptop = computerShop.GetMyComputerPrice(responsibleForCalcu
const priceAnswerDesktop = computerShop.GetMyComputerPrice(responsibleForCalc
const anyGiftAnswer = computerShop.IsThereAnyGiftItem(queryForGiftItem);
const colorAnswer = computerShop.WhatIsTheColorOfGiftItem(queryForGiftItem);
const myAnswer = computerShop.GetAvailableColor(queryForLaptop);
console.log("cumparator: Care este pretul unui laptop si calculator desktop?"
console.log("vinzator: " + priceAnswerLaptop + " pentru laptop si pentru desk
console.log("vinzator: " + laptopType + " sau " + desktopType + "?");
console.log("cumparator: Daca procur un laptop voi primi un cadou sau ceva gr
console.log("vinzator: " + anyGiftAnswer);
console.log("cumparator: Care sunt culorile disponibile pentru stick-ul usb?"
console.log("vinzator: " + colorAnswer);
console.log("cumparator: Care sunt culorile disponibile pentru laptop?")
console.log("vinzator: " + myAnswer)
```

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 interfete mai mici și mai specifice, astfel încât clienții vor trebui doar să știe despre metodele care ii 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 IComputerDescription {
    GetDescription();
    GetColor();
}
interface IComputerPrice {
    GetPrice();
}
interface IPriceCalculation {
    CalculatePriceWithDiscount(calculate : IComputerPrice);
}
```

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 Desktop implements IComputerDescription, IComputerPrice {
    public desktopPrice: number
    constructor(desktopPrice : number) {
        this.desktopPrice = desktopPrice
    }
    public GetDescription(): string{
        return "Calculator Desktop";
    }
    public GetColor(){
        return "Negru";
    }
    public GetPrice(){
        return this.desktopPrice;
    }
}
class Laptop implements IComputerDescription, IComputerPrice {
    public laptopPrice : number
    constructor(laptopPrice : number) {
        this.laptopPrice = laptopPrice
    }
    public GetDescription(): string {
        return "Laptop"
    }
    public GetColor(): string {
```

```
        return "Negru, alb"
    }
    public GetPrice(): number {
        return this.laptopPrice
    }
}
class GiftItem implements IComputerDescription {
    public GetDescription(): string {
        return "Stick USB"
    }
    public GetColor(): string {
        return "Rosu, negru, alb"
    }
}
class CalculateComputerPrice implements IPriceCalculation {
    public CalculatePriceWithDiscount(calculate : IComputerPrice) : any {
        return calculate.GetPrice() - calculate.GetPrice() * 0.1;
    }
}
class Shop {
    public GetMyComputer(cmptype : IComputerDescription) {
        var myComp = cmptype.GetDescription();
        return myComp;
    }
    public GetMyComputerPrice(cmpCal : IPriceCalculation, cmpPrice : IComput
        var myCompprice = "Pretul este " + cmpCal.CalculatePriceWithDiscount(
        return myCompprice;
    }
    public GetAvailableColor(cmptype : IComputerDescription): any {
        var myCompcolor = cmptype.GetColor();
        return myCompcolor;
    }
    public WhatIsTheColorOfGiftItem(cmptype : IComputerDescription) {
        return this.GetAvailableColor(cmptype);
    }
    public IsThereAnyGiftItem(gftType : IComputerDescription) {
        return this.GetMyComputer(gftType);
    }
}
```

3.3 Imagini

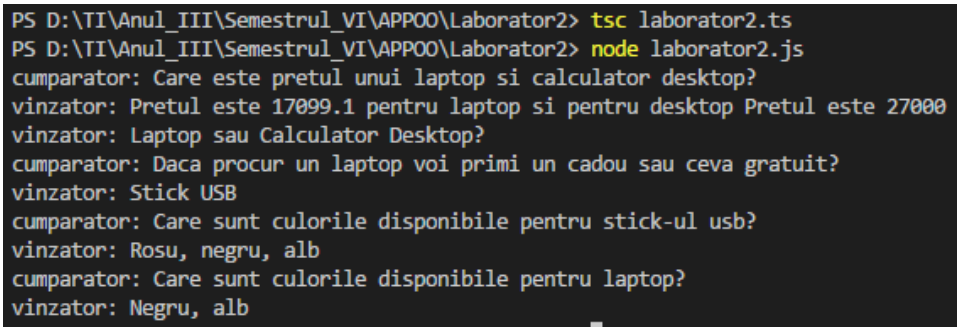

Figure 3.1 – Final result

Concluzie

In lucrarea data, au fost implementate 2 principii SOLID precum Single Responsibility principle si Interface Segrgation 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 sa analizam codul folosind modele standard și terminologii și să evaluam calitativ "curățenia" codului.

Prin urmare, este important să înțelegem că respectarea acestor principii nu ne va face în mod automat codul curat, insa 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.oodesign.com/single-responsibility-principle.html
2. Interface Segregation Principle [Resursă electronică]. Regim de acces: http://www.oodesign.com/interface-segregation-principle.html