Projektowanie obiektowe oprogramowania
Zestaw 3

SOLID GRASP

2019-03-12

Liczba punktów do zdobycia: **9/25**
Zestaw ważny do: 2019-03-26

1. (**2p**) (**GRASP**) Pięć wybranych zasad projektowych GRASP zilustrować przykładowym kompilującym się kodem. Umieć uzasadnić, że zaproponowany kod rzeczywiście ilustruje wybrane reguły.

2. (**1p**) (**Single Responsibility Principle**) Dokonać analizy projektu obiektowego pod katem zgodności z zasadą SRP. Zaproponować zmiany. Narysować diagramy klas przed i po zmianach. Zaimplementować działający kod dla przykładu przed i po zmianach.

```
public class ReportPrinter {
   public string GetData();
   public void FormatDocument();
   public void PrintReport();
}
```

   Ile klas docelowo powstanie z takiej jednej klasy? Dlaczego akurat tyle? Czy refaktoryzacja klasy naruszającej SRP oznacza automatycznie, że **każda** metoda powinna trafić do osobnej klasy?

3. (**2p**) (**Open-Closed Principle**) Dokonać analizy projektu obiektowego pod katem zgodności klasy `CashRegister` z zasadą OCP. Zaproponować takie zmiany, które uczynią ją niezmienną a równocześnie rozszerzalną jeśli chodzi o możliwość implementowania różnych taryf podatkowych oraz drukowania paragonów z uwzględnieniem różnego porządkowania towarów (alfabetycznie, według kategorii itp.)

   Narysować diagramy klas przed i po zmianach. Zaimplementować działający kod dla przykładu przed i po zmianach demonstrując kilka różnych rozszerzeń.

```
public class TaxCalculator {
   public Decimal CalculateTax( Decimal Price ) { return Price * 0.22 }
}

public class Item {
   public Decimal Price { get { ... } }
   public string Name { get { ... } }
}

public class CashRegister {
   public TaxCalculator taxCalc = new TaxCalculator();

   public Decimal CalculatePrice( Item[] Items ) {
      Decimal _price = 0;
      foreach ( Item item in Items ) {
```

```
            _price += itemPrice + taxCalc.CalculateTax( item.Price );
        }
        return _price;
    }

    public string PrintBill( Item[] Items ) {
      foreach ( var item in Items )
        Console.WriteLine( "towar {0} : cena {1} + podatek {2}",
           item.Name, item.Price, taxCalc.CalculateTax( item.Price ) );
    }
}
```

4. (**1p**) (**Liskov Substitution Principle**) Zaprojektowano klasy `Rectange` i `Square` i w "naturalny" sposób relację dziedziczenia między nimi (każdy kwadrat jest prostokątem).

```
public class Rectangle
{
    public virtual int Width { get; set; }
    public virtual int Height { get; set; }
}

public class Square : Rectangle
{
    public override int Width
    {
        get { return base.Width; }
        set { base.Width = base.Height = value;}
    }

    public override int Height
    {
        get { return base.Height; }
        set { base.Width = base.Height = value; }
    }
}
```

   Co można powiedzieć o spełnianiu przez taką hierarchię zasady LSP w kontekście poniższego kodu klienckiego?

```
public class AreaCalculator
{
    public int CalculateArea( Rectangle rect )
    {
        return rect.Width * rect.Height;
    }
}

int w = 4, h = 5;

Rectangle rect = new Square() { Width = w, Height = h };

AreaCalculator calc =  AreaCalculator();

Console.WriteLine( "prostokąt o wymiarach {0} na {1} ma pole {2}",
    w, h, calc.CalculateArea( rect ) );
```

   Jak należałoby zmodyfikować przedstawioną hierarchię klas, żeby zachować zgodność z LSP w kontekście takich wymagań? Jak potraktować klasy `Rectangle` i `Square`? Odpowiedź zilustrować działającym kodem.

5. (**1p**) (**Interface Segregation Principle**) Znaleźć w bibliotece standardowej dowolnego języka programowania przykład interfejsu, który łamie zasadę ISP tzn. istnieją zastosowania, w których korzysta się tylko z części tego interfejsu.

   Zaproponować refaktoryzację interfejsu wolną od zaobserwowanej przypadłości.

6. (**1p**) (**SRP vs ISP**) Wytłumaczyć różnicę między SRP a ISP.

7. (**1p**) (**Dependency Inversion Principle**) Przykład z zadania o SRP zrefaktoryzować - wprowadzić klasę `ReportComposer` która ma "wstrzykiwane" zależności do obiektów usługowych.

Wiktor Zychla