

Rozwiązanie zadania 3 z zestawu 3 z ”Projektowania obiektowego oprogramowania”

Mateusz Małowiecki

March 17, 2021

Zaprojektowano klasy `Rectangle` 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.

Rozwiązanie

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

Hierarchia ta łamie zasadę LSP. Powodem jest to, że prostokąty nie mogą być reprezentowane jako kwadraty, gdyż settery kwadratów zachowują się w nieoczekiwany sposób (podstawienie $\text{Width} = w$, $\text{Height} = h$, może spowodować, że pole figury będzie inne niż $w \cdot h$).

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

Ponieważ Square nie rozszerza Rectangle to moglibyśmy usunąć relację dziedziczenia między Rectangle a Square. Dodatkowo ponieważ dla obu tych figur chcielibyśmy móc policzyć ich pola, to, żeby nie musieć obsługiwać osobno dwóch typów, moglibyśmy stworzyć abstrakcyjną klasę Shape, która będzie miała metodę `GetArea`, obliczającą pole danej figury. Takie rozwiązanie pozwala na zachowanie zasady LSP, ponieważ w dowolnym kontekście klasę Shape możemy zastąpić obiektami podklas Rectangle i Square.