Metodyki zwinne wytwarzania oprogramowania

Wykład 10

Marcin Młotkowski

14 grudnia 2016

Plan wykładu

Definicja

Jest to proces ulepszania kodu źródłowego bez zmiany jego działania.

Definicja

Jest to proces ulepszania kodu źródłowego bez zmiany jego działania.

Komentarz

Podczas refaktoryzacji nie dodajemy nowego kodu, tylko go poprawiamy.

Dlaczego refaktoryzujemy

- ulepszanie projekt programu (po dodaniu funkcjonalności etc.) np. usuwanie powtórzeń;

Dlaczego refaktoryzujemy

- ulepszanie projekt programu (po dodaniu funkcjonalności etc.) np. usuwanie powtórzeń;
- Zwiększenie czytelności, tak aby rozumiał to nie tylko komputer ale też inny programista.

Dlaczego refaktoryzujemy

- ulepszanie projekt programu (po dodaniu funkcjonalności etc.) np. usuwanie powtórzeń;
- Zwiększenie czytelności, tak aby rozumiał to nie tylko komputer ale też inny programista.
- Czytelny kod pomaga znaleźć błędy.

Dlaczego refaktoryzujemy

- ulepszanie projekt programu (po dodaniu funkcjonalności etc.) np. usuwanie powtórzeń;
- Zwiększenie czytelności, tak aby rozumiał to nie tylko komputer ale też inny programista.
- Czytelny kod pomaga znaleźć błędy.
- Przyspiesza pisanie programu.

Brzydkie zapachy w kodzie

- Długi kod metody;

Brzydkie zapachy w kodzie

- Długi kod metody;
- duża klasa (wiele odpowiedzialności);

Brzydkie zapachy w kodzie

- Długi kod metody;
- duża klasa (wiele odpowiedzialności);
- Zazdrość o kod (jeśli klasa intensywnie korzysta z metody innej klasy, to metoda powinna być przeniesiona);

Brzydkie zapachy w kodzie

- Długi kod metody;
- duża klasa (wiele odpowiedzialności);
- Zazdrość o kod (jeśli klasa intensywnie korzysta z metody innej klasy, to metoda powinna być przeniesiona);
- zbytnia intymność: dostępność do implementacji innej klasy;

Brzydkie zapachy w kodzie

- Długi kod metody;
- duża klasa (wiele odpowiedzialności);
- Zazdrość o kod (jeśli klasa intensywnie korzysta z metody innej klasy, to metoda powinna być przeniesiona);
- zbytnia intymność: dostępność do implementacji innej klasy;
- leniwa klasa (ma zbyt mało odpowiedzialności);

Brzydkie zapachy w kodzie

- Długi kod metody;
- duża klasa (wiele odpowiedzialności);
- Zazdrość o kod (jeśli klasa intensywnie korzysta z metody innej klasy, to metoda powinna być przeniesiona);
- zbytnia intymność: dostępność do implementacji innej klasy;
- leniwa klasa (ma zbyt mało odpowiedzialności);
- powielony kod;

Brzydkie zapachy w kodzie

- Długi kod metody;
- duża klasa (wiele odpowiedzialności);
- Zazdrość o kod (jeśli klasa intensywnie korzysta z metody innej klasy, to metoda powinna być przeniesiona);
- zbytnia intymność: dostępność do implementacji innej klasy;
- leniwa klasa (ma zbyt mało odpowiedzialności);
- powielony kod;
- nadmierne skomplikowanie kodu.

Przykłady zmian

Rozszerzanie funkcjonalności metod/procedur

Dodawanie nowych możliwości do istniejącej metody, na przykład metodę

OpenWindow(int x, int y, int width, int height)

uzupełniamy o możliwość wyboru koloru

OpenWindow(int x, int y, int width, int height, color border)

Dodawanie kodu

Jeden programista dopisał

```
class Window : Widget
    void ChangeFontColor();
end
```

Dodawanie kodu

Jeden programista dopisał

```
class Window : Widget
    void ChangeFontColor();
end
```

Drugi programista dopisał

```
class Button : Widget
    void ChangeTextColor();
end
```

Dodawanie kodu

Jeden programista dopisał

```
class Window : Widget
    void ChangeFontColor();
end
```

Drugi programista dopisał

```
class Button : Widget
    void ChangeTextColor();
end
```

Powinno być

```
class Widget
    void ChangeForegroundColor()
end
```

Ważne

1.

Refaktoryzacja nie może nic popsuć.

2.

Po każdej zmianie program należy testować.

Plan wykładu

Wypożyczalnia filmów

```
class Film {
    public String tytuł;
    int cena();
}

class Wypożyczenie {
    Film film;
    int liczba_dni;
}
```

```
class Klient {
    Set<Wypożyczenia> zbior;

    public String zestawienie()
    {
        int suma = 0;
        Sting wynik = "Zestawienie dla " + self.Nazwisko + "\n";
        foreach(elem in  zbior)
        {
            suma += elem.film.cena() * elem.liczba_dni;
            wynik += "Tytuł: " + elem.film.tytuł + "\n";
        }
        wynik += "Razem: " + suma.toString();
    }
}
```

Rozwój wypożyczalni

Rozszerzenie funkcjonalności

- wyższa cena dla nowości;
- zniżki dla stałych klientów;

Rozwój wypożyczalni

Rozszerzenie funkcjonalności

- wyższa cena dla nowości;
- zniżki dla stałych klientów;

Konieczne zmiany

- rozszerzenie klasy `Klient` o informacje czy jest stałym klientem;

Rozwój wypożyczalni

Rozszerzenie funkcjonalności

- wyższa cena dla nowości;
- zniżki dla stałych klientów;

Konieczne zmiany

- rozszerzenie klasy `Klient` o informacje czy jest stałym klientem;
- rozszerzenie klasy `Film` o informację, czy film jest nowością;

Rozwój wypożyczalni

Rozszerzenie funkcjonalności

- wyższa cena dla nowości;
- zniżki dla stałych klientów;

Konieczne zmiany

- rozszerzenie klasy `Klient` o informacje czy jest stałym klientem;
- rozszerzenie klasy `Film` o informację, czy film jest nowością;
- modyfikacja metody `Klient.zestawienie`.

Implementacja nowości

class Klient

```
public String zestawienie() {
    int suma = 0;
    Sting wynik = "Zestawienie dla " + self.Nazwisko + "\n";
    foreach(elem in zbior) {
        if elem.nowosc()
            suma += elem.film.cena() * elem.liczba_dni * 2;
        else
            suma += elem.film.cena() * elem.liczba_dni;
        wynik += "Tytuł: " + elem.film.tytuł + "\n";
    }
    if self.stalyKlient()
        suma = suma*0.7;
    wynik += "Razem: " + suma.toString();
}
```

Kolejna zmiana

Do klasy

```
class Klient {
    public String zestawienie() { ... }
}
```

dorzucamy metodę generującą zestawienia w formacie HTML

```
class Klient {
    public String zestawienie() { ... }
    public String zestawienieHTML() { ... }
}
```

Ocena trudności

Optymista

Łatwe! Wystarczy przekopiować kod i poprawić!

Ocena trudności

Optymista

Łatwe! Wystarczy przekopiować kod i poprawić!

Pesymista

A jeśli trafi się kolejna modyfikacja:

- najpierw dopisanie metody `zestawienieHTML()`
- potem modyfikacja sposobu naliczania kwoty za wypożyczenie

Analiza kodu

Przypomnienie

```
public String zestawienie() {
    int suma = 0;
    Sting wynik = "Zestawienie dla " + self.Nazwisko + "\n";
    foreach(elem in zbior) {
        if elem.nowosc()
            suma += elem.film.cena() * elem.liczba_dni * 2;
        else
            suma += elem.film.cena() * elem.liczba_dni;
        wynik += "Tytuł: " + elem.film.tytuł + "\n";
    }
    if self.stalyKlient()
        suma = suma*0.7;
    wynik += "Razem: " + suma.toString();
}
```

Analiza

Metoda robi dwie rzeczy:

- oblicza cenę za wypożyczenie za 1 dzień
- generuje raport

Analiza

Metoda robi dwie rzeczy:

- oblicza cenę za wypożyczenie za 1 dzień
- generuje raport

Potencjalne rozwiązania

Wydzielenie kodu obliczania należności za film

- jako metodę klasy `Klient`
- rozszerzenie metody `Film.cena()`

Obliczanie ceny przez klasę `Klient`

```
class Film {
    int cena() {
        if (self.nowosc)
            return 2*self._cena;
        else
            return self._cena;
    }
}
```

Obliczanie ceny przez klasę `Klient`

```
class Film {
    int cena() {
        if (self.nowosc)
            return 2*self._cena;
        else
            return self._cena;
    }
}

class Klient {
    public int cena(Film f) {
        int res = f.cena();
        if (self.staly)
            return 0.7*f.cena();
        else
            return f.cena();
    }
}
```

Cenę oblicza klasa `Film`

```
class Film {
    public String tytuł;
    int cena() {
        return _cena;
    }
}
```

Cenę oblicza klasa `Film`

```
class Film {
    public String tytuł;
    int cena(Klient k) {
        int kwota = self._cena;
        if self.nowosc()
            kwota = 2 * kwota;
        if k.stały_Klient()
            kwota = 0.7*kwota;
        return kwota;
    }
}
```

Kolejna modyfikacja

Rodzaje filmów

Tworzymy klasyfikację filmów i uzależniamy cenę od rodzaju.

Tabela opłat

rodzaj filmu	mnożnik
sensacyjny	2.71
dla dzieci	0.9
teatr TV	.4

Pierwsze podejście

```
switch self.rodzaj {
    case SENSACJA:
        kwota = 2.71*self._cena;
        break ;
    case DZIECIECY:
        kwota = 0.9*self._cena;
        break ;
    case TEATRTV:
        kwota = 0.4*self._cena;
        break ;
}
```

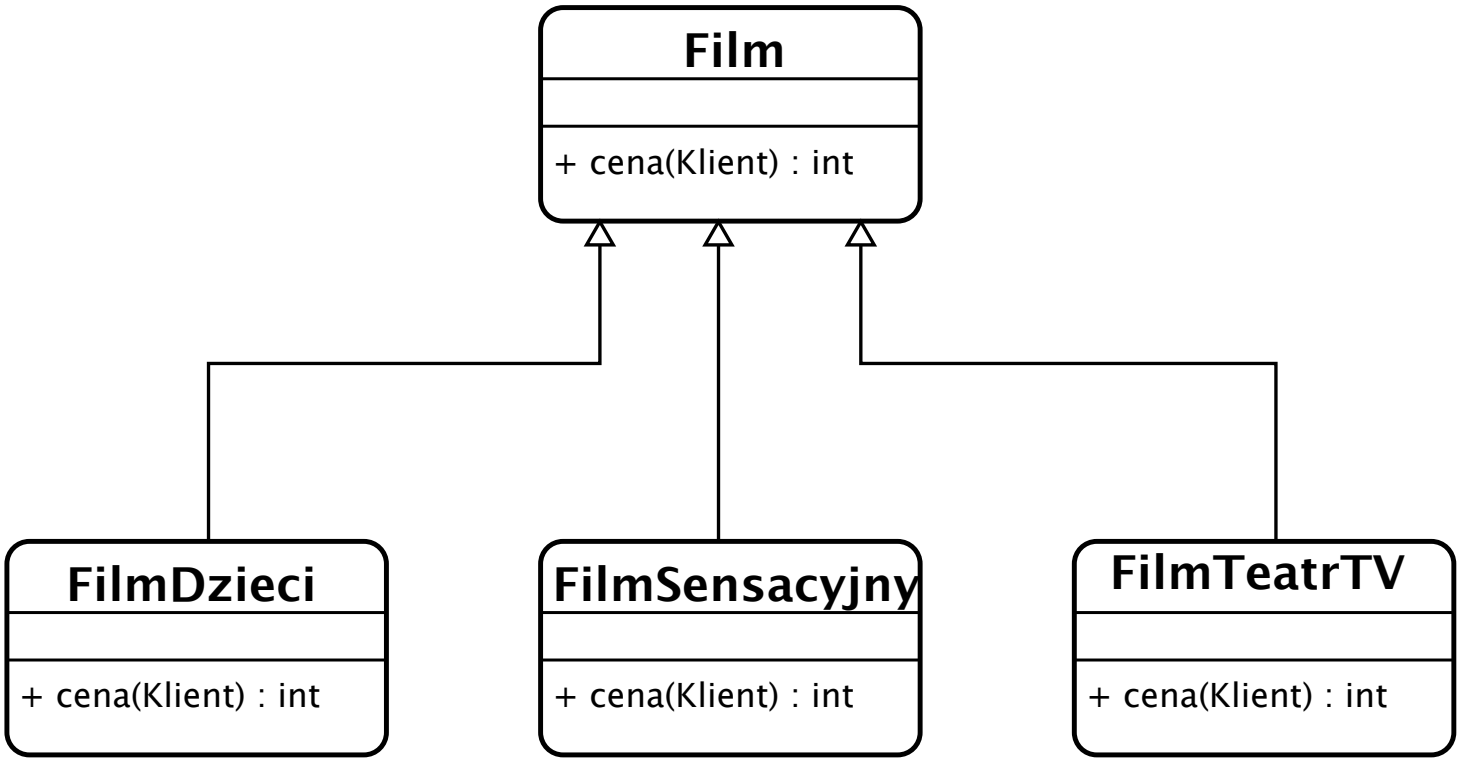
Przypomnienie

Zasada otwarte–zamknięte

Kod powinien być otwarty na rozszerzanie a zamknięty na modyfikacje.

Czy ten program spełnia tę zasadę?

Podejście drugie



Ocena

Wady tego rozwiązania

Nie ma możliwości zmiany typu filmu.

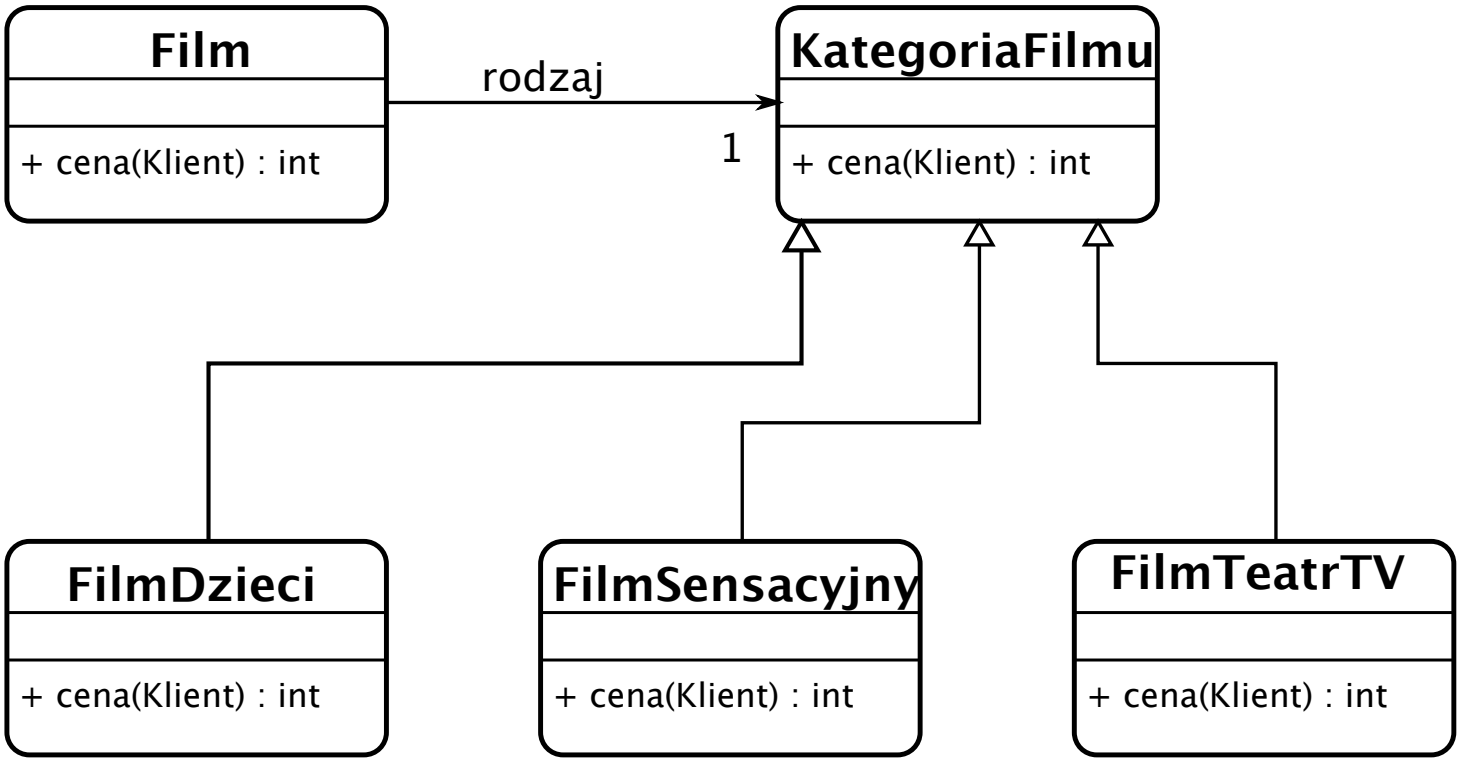
Podejście trzecie

Wydzielenie z klasy `Film` klasyfikacji i cennika.

Narzędzie

Wzorzec projektowy *Stan*

Podejście trzecie



Plan wykładu

Zasady refaktoryzacji

Jak NIE refaktoryzować

NIGDY nie wykonuje się na raz wszystkich zmian.

Zasady refaktoryzacji

Jak NIE refaktoryzować

NIGDY nie wykonuje się na raz wszystkich zmian.

Jak refaktoryzować

Jeśli jest kilka zmian do zrobienia, to wykonujemy je w etapach:

1. zmiana niedużego fragmentu;
2. testy;
3. jeśli testy przechodzą, wykonujemy kolejną zmianę.

Przykład

Zmiana pierwsza

Zróżnicowanie cennika.

Przykład

Zmiana pierwsza

Zróżnicowanie cennika.

Zmiana druga

Dodanie zestawienia w html'u.

Krok pierwszy

Refaktoryzacja

Wydzielenie samej funkcjonalności obliczania należności.

Krok pierwszy

Refaktoryzacja

Wydzielenie samej funkcjonalności obliczania należności.

Testowanie

Uruchomienie testów jednostkowych dla tego modułu.

Krok pierwszy

Refaktoryzacja

Wydzielenie samej funkcjonalności obliczania należności.

Testowanie

Uruchomienie testów jednostkowych dla tego modułu.

Dodanie funkcjonalności

Dodanie nowej metody `zestawienieHTML`.

Krok drugi

Refaktoryzacja

Wydzielenie klasy `KategoriaFilmu`

Krok drugi

Refaktoryzacja

Wydzielenie klasy `KategoriaFilmu`

Testowanie

Uruchomienie testów jednostkowych dla tego modułu.

Krok drugi

Refaktoryzacja

Wydzielenie klasy `KategoriaFilmu`

Testowanie

Uruchomienie testów jednostkowych dla tego modułu.

Dodanie funkcjonalności

Dodanie nowych kategorii.

Ważne

Co jest potrzebne

Musimy mieć testy jednostkowe.