Metodyki zwinne wytwarzania oprogramowania

Wykład 11

Marcin Młotkowski

4 stycznia 2017

Plan wykładu

Kiedy refaktoryzować

Brzydkie zapachy w kodzie

- powielony kod;
- długa metoda;
- duża klasa;
- długa lista paramterów;
- rozbieżna zmiana;
- poszatkowanie;
- zazdrość o kod;
- zbitki danych;
- obsesja typów podstawowych;
- instrukcje `switch`;
- ...

Plan wykładu

Powielony kod

Fragment kodu jest w kilku miejscach.

Rodzaje powielenia

- wspólny fragment kodu w kilku metodach tej samej klasy;
- wspólny fragment kodu w metodach podklas;
- wspólny fragment kodu w różnych częściach programu.

Schemat postępowania

- utworzenie nowej metody;
- skopiowanie kodu;
- analiza wykorzystania zmiennych lokalnych (deklaracje, parametry);
- kompilacja;
- zastąpienie starego kodu nową metodą;
- kompilacja i testowanie.

Prosty przykład

```
void raport(DataList data)
{
  Console.WriteLine('*************************');
  Console.WriteLine(' Raport z dn. {0} ', DateTime.Now);
  Console.WriteLine('*************************');

  foreach(DataElement d in data)
  {
    Console.WriteLine(d.toString());
  }

  Console.WriteLine('************************');
  Console.WriteLine('******** Koniec *******');
  Console.WriteLine('************************');
}
```

Wydzielenie metody

```
void naglowek() {
  Console.WriteLine('*****************************');
  Console.WriteLine(' Raport z dn. {0} ', DateTime.Now);
  Console.WriteLine('*****************************');
}

void stopka() {
  Console.WriteLine('*****************************');
  Console.WriteLine('********** Koniec *********');
  Console.WriteLine('*****************************');
}

void raport(DataList data) {
  naglowek();
  foreach(DataElement d in data)
  {
    Console.WriteLine(d.toString());
  }
```

Łatwe! Bo nie było zmiennych!

Wydzielenie metody ze zmiennymi do odczytu

```
void raport(DataList data, string title)
{
  Console.WriteLine('***** {0} *****', title);
  Console.WriteLine(' Raport z dn. {0} ', DateTime.Now);
  Console.WriteLine('**************************');

  foreach(DataElement d in data)
  {
    Console.WriteLine(d.toString());
  }

  Console.WriteLine('**************************');
  Console.WriteLine('********** Koniec *********');
  Console.WriteLine('**************************');
}
```

Wydzielenie metody ze zmienną

```
void naglowek(string title) {
  Console.WriteLine('********* {0} *******', title);
  Console.WriteLine(' Raport z dn. {0} ', DateTime.Now);
  Console.WriteLine('************************');
}
void stopka() {
  Console.WriteLine('****************************');
  Console.WriteLine('*********** Koniec *********');
  Console.WriteLine('****************************');
}

void raport(DataList data, string title) {
  naglowek(title);
  foreach(DataElement d in data)
    Console.WriteLine(d.toString());
}
```

Przykład

```
void należność(DataList data, string title)
{
  double suma = 0;

  nagłówek(title);
  foreach(DataElement d in data)
  {
    suma += d.kwota;
  }
  Console.WriteLine("Suma: {0}", suma);
}
```

Przykład po refaktoryzacji

```
double podsumowanie(DataList data) {
  double suma
  foreach(DataElement d in data)
  {
    suma += d.kwota;
  }
  return suma;
}

void należność(DataList data, string title) {
  double suma = 0;
  nagłówek(title);
  suma = podsumowanie(data);
  Console.WriteLine("Suma: {0}", suma);
}
```

Kłopoty

	misie	mikołaje	renifery
styczeń	10	14	5
luty	34	12	7
...	...	...	...
podsumowanie	1024	512	2048

Opis

Kiedy stosujemy

W podklasach występują pola obiektu o podobnym znaczeniu i takim samym typie.

Opis

Kiedy stosujemy

W podklasach występują pola obiektu o podobnym znaczeniu i takim samym typie.

Przykład

```
class Osoba { }

class Pracownik : Osoba {
  private string nazw;
}

class Student : Osoba {
  protected string Name;
}
```

Opis procedury

1. analiza wszystkich odwołań do pól (kandydatów do referencji);
2. ustalenie wspólnej nazwy dla obydwu zmiennych i zmiana nazw;
3. kompilacja i testy;
4. utworzenie nowego pola w nadklasie (o widoczności w podklasach);
5. usunięcie pól z podklas;
6. kompilacja i testowanie;
7. rozważenie samoenkapsulacji pola.

Opis

Metoda szablonowa przydaje się w sytuacji, gdy mamy podobny (ale nie identyczny) kod w dwóch metodach.

Przykład (był na poprzednim wykładzie)

Wypożyczalnia filmów

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

Sposób postępowania

Krok 1. – podzielenie metod na mniejsze tak, aby były identyczne lub różne

Wydzielenie obliczania należności

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

Krok 2

Przemieszczenie metody w górę.

Krok 2

Przemieszczenie metody w górę.

W naszym przykładzie nie jest to potrzebne.

Krok 3

Ujednolicenie *sygnatur* metod.

Krok 3

Ujednolicenie *sygnatur* metod.

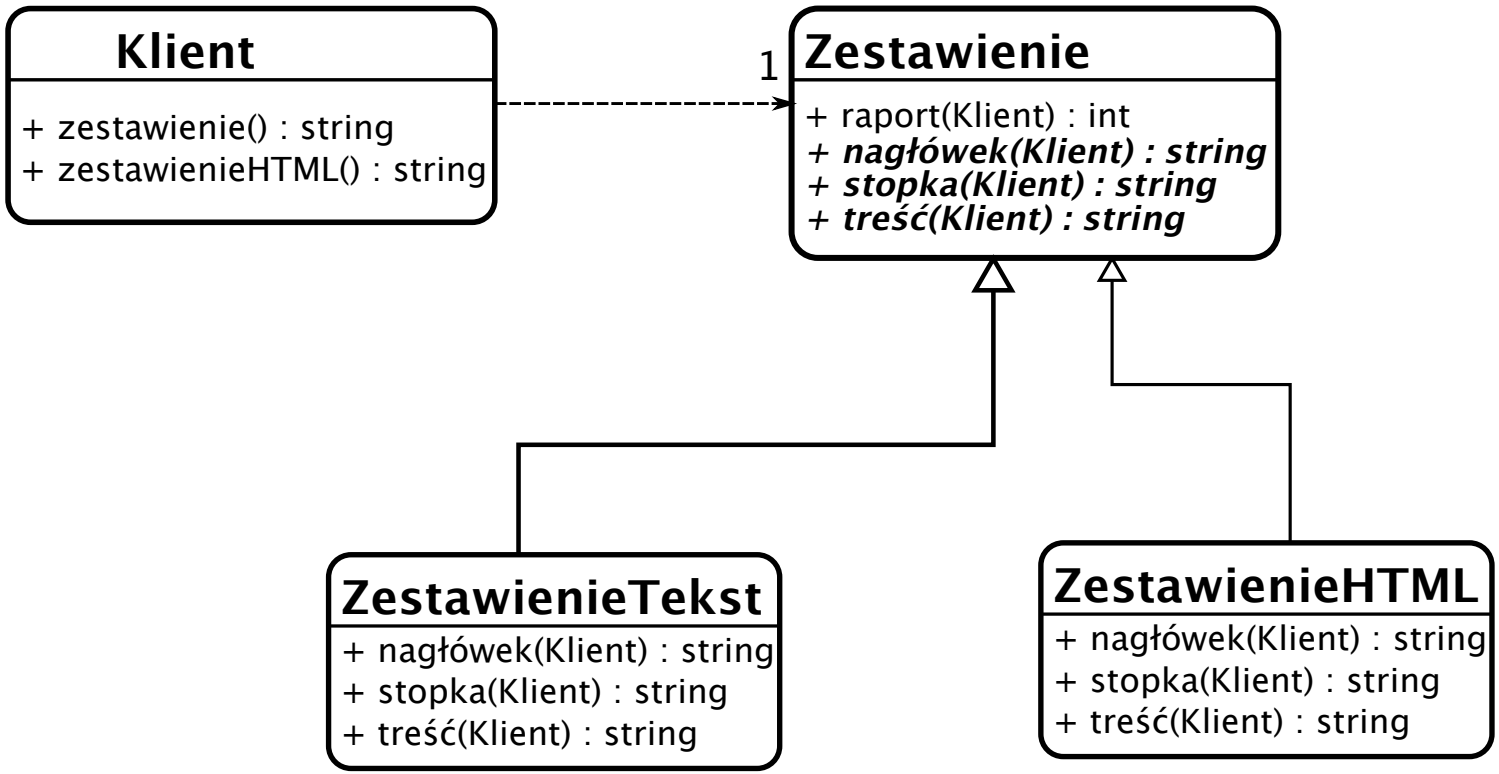
Krok 4

Kompilacja i testowanie

Krok 5: tworzenie metody szablonowej

```
class Klient {
  String raport(void -> string : nagłowek,
                void -> string : zestawienie) {
    String wynik = "";
    wynik += nagłówek();
    wynik += zestawienie();
    wynik += this.podsumowanie();
    return wynik;
  }
  public String zestawienie() {
    return raport(nagłowekTekst, zestawienieTekst);
  }
  public String zestawienieHTML() {
    return raport(nagłowekHTML, zestawienieHTML);
  }
}
```

Wzorzec strategia



Jak używać takiej implementacji

```
class Klient {
  public String zestawienie()
  {
    return new ZestawienieTekst().raport(this);
  }

  public String zestawienieHTML()
  {
    return new ZestawienieHTML().raport(this);
  }
}
```

Kiedy wydzielić klasę

Z klasy wydzielamy odrębną klasę aby skrócić klasę podstawową.

Przykład

```
class Klient
{
  String adres;
}
```

Rozwój oprogramowania

```
class Klient
{
  String kraj;
  String ulica;
  String miejscowość;
  String poczta;
  String kod_poczt;

  String adresTekst() {
    return kraj + ulica + miejscowość +
           poczta + kod_poczt;
  }
}
```

Wydzielenie klasy

```
class Adres {
  public String toString() { ... }
}

class Klient {
  Adres adres;
}
```

Plan wykładu

Długa metoda

Dobra metoda to zazwyczaj krótka metoda.

Jak i kiedy wydzielić metodę

To już było ;-)

Kiedy wykonujemy tę operację

Operację wykonujemy dla lokalnych zmiennych tymczasowych, np.

```
...
// obliczanie
tmp = wyrażenie
wynik = ... tmp ...
...
```

Kiedy wykonujemy tę operację

Operację wykonujemy dla lokalnych zmiennych tymczasowych, np.

```
...
// obliczanie
tmp = wyrażenie
wynik = ... tmp ...
...
```

```
int wynik_pośredni() { ... }
```

```
...
wynik = ... wynik_pośredni() ...
...
```

Jak to zrobić

- **Sprawdzenie, że jest tylko jedno przypisanie**

  Np. zadeklarowanej zmiennej jako *final* i skompilowanie programu.

- wydzielenie obliczenia zmiennej jako odrębną metodę i skompilowanie;
- wstawienie wywołania nowej metody zamiast zmiennej tymczasowej.

Warianty metody

Zmiennej przypisywana jest wartość więcej niż dwa razy. I nie jest to zmienna wykorzystywana w pętli.

Warianty metody

Zmiennej przypisywana jest wartość więcej niż dwa razy. I nie jest to zmienna wykorzystywana w pętli.

Obserwacja

Jeśli zmiennej przypisywana jest wartość więcej niż dwa razy to pewnie jest użyta w dwóch rolach.

Warianty metody

Zmiennej przypisywana jest wartość więcej niż dwa razy. I nie jest to zmienna wykorzystywana w pętli.

Obserwacja

Jeśli zmiennej przypisywana jest wartość więcej niż dwa razy to pewnie jest użyta w dwóch rolach.

Przykład

```
int podsuma;
...
podsuma = ...
...
podsuma = ...
...
```

Rozdzielenie zmiennej tymczasowej

Przykład

```
int podsuma1;
int podsuma2
...
podsuma1 = ...
...
podsuma2 = ...
...
```

Sprawdzeniem może być zadeklarowanie zmiennej jako `final`.

Przykład

```
String raport(
      DataSource źródło,
      Data od, Data do,
      String nagłówek, String stopka,
      bool numerowanie_stron, bool posumowanie,
      String porządkowanie, bool filtr)
{
   ...
}
```

Rozwiązanie

Utworzenie kilku klas

```
class CzasZakres {
  Data od;
  Data do;
}
class CechyRaportu {
  String nagłówek;
  String stopka;
  bool numerowanie_stron;
  bool podsumowanie;
}
```

Utworzenie jednej klasy

```
class ParametryRaportu
{
  Data od;
  Data do;
  String nagłowek;
  String stopka;
  bool numerowanie_stron;
  bool podsumowanie;
}
```

Zalety rozwiązania

- krótsza liczba parametrów;
- kontrola poprawności parametrów na poziomie klasy;

Zalety rozwiązania

- krótsza liczba parametrów;
- kontrola poprawności parametrów na poziomie klasy;
- przeniesienie do klasy pewnych operacji

Kod z wnętrza raportu

```
if (dana.data >= od and dana.data <= do)
  ...
```

Nowa implementacja

```
class Zakres {
  public bool zawiera(Data d) { ... }
}


...
if (zakres.zawiera(dana.data))
```

Obiekt jest parametrem metody

Jest to w pewnym sensie metoda odwrotna do poprzedniej.

Przykład

```
public String raport(Finanse dane) { ... }
```

Komentarz

- tę faktoryzację można stosować również gdy przekazujemy tylko jeden argument;
- taka operacja uodparnia na zmiany, np. potrzebę nowego argumentu do wykonania metody.

Zastosowanie refaktoryzacji

- mamy (zbyt) długą metodę;
- w metodzie jest mnóstwo zmiennych lokalnych, które występują w różnych miejscach w procedurze;
- podział metody na mniejsze jest kłopotliwe, bo: wymaga przekazywania mnóstwa parametrów do mniejszych metod; w wielu językach programowania przekazywanie argumentów jest przez wartość.

Przykład

Metoda `float sumowanie()`, klasa `Numeryka`

```
sumaCałkowita = 0.0;
for(int i = 0; i < wiersze; i++)
{
  podsuma = 0.0;
  for(int j = 0; j < kolumny; j++)
  {
    sumaCałkowita += M[i,j];
    podsuma += M[i,j];
  }
  wydruk(podsuma);
}
```

```
float podsuma(Macierz M, wiersz, sumaCałkowita)
{
  podsuma = 0.0;
  for(int j = 0; j < kolumny; j++)
  {
    sumaCałkowita += M[wiersz,j];
    podsuma += M[wiersz,j];
  }
  wydruk(podsuma);
  return sumaCałkowita;
}
...
sumaCałkowita = 0.0;
for(int i = 0; i < wiersze; i++)
{
  sumaCałkowita = podsuma(M, i, sumaCałkowita)
}
```

```
float podsuma(Macierz M, wiersz, sumaCałkowita)
{
  podsuma = 0.0;
  for(int j = 0; j < kolumny; j++)
  {
    sumaCałkowita += M[wiersz,j];
    podsuma += M[wiersz,j];
  }
  wydruk(podsuma);
  return sumaCałkowita;
}
...
sumaCałkowita = 0.0;
for(int i = 0; i < wiersze; i++)
{
  sumaCałkowita = podsuma(M, i, sumaCałkowita)
}
```

Co będzie, gdy musimy zwrócić więcej niż jeden parametr?

Zastąpienie metody obiektem

Procedura

- utworzenie klasy o nazwie takiej jak metoda;
- utworzenie w nowej klasie pola na obiekt z którego pochodzi metoda;
- utworzenie w nowej klasie pól odpowiadających zmiennym lokalnym ze starej metody;
- utworzenie w nowej klasie metody `oblicz` o kodzie takim samym jak stara metoda;
- zastąpienie ciała starej metody wywołaniem metody `oblicz` starej metody.

Oczywiście w międzyczasie kompilujemy i testujemy.

```
class Sumowanie  {
    Numeryka obiekt; Macierz M;
    float sumaCałkowita = 0; float podsuma = 0;
    int i;
    public Sumowanie(Numeryka obj, .... ) { M = obj.M }

    private void podsuma() {
      this.podsuma = 0.0;
      for(int j = 0; j < kolumny; j++) {
        this.sumaCałkowita += M[i,j];
        this.podsuma += M[i,j];
      }
      obj.wydruk(this.podsuma);
    }
    public void oblicz() {
      this.sumaCałkowita = 0.0;
      for(int i = 0; i < wiersze; i++)
           this.podsuma()
    }
}
```

Złe wyrażenie warunkowe: przykład

```
if (DateTime.Now < DateTime("20-02-2011") and
    DateTime("02-10-2010") < DateTime.Now)
        conn = new Connection("..")
        conn.query("SELECT * FROM Tabela
            WHERE czas < '20-02-2011' AND '02-10-2010'")
else
    ...
```

Propozycja rozwiązania

Wydzielenie wyrażenia w postaci metody

```
bool semestrZimowy(int rok) { ... }
```

Wydzielenie gałęzi if

```
void ofertaZima(int rok) { ... }
```

Zastąpienie wyrażenia metodą

```
if semestrZimowy(2010)
  ofertaZima(2010);
else
  ofertaLato(2010);
```

Plan wykładu

Wydzielenie klasy

Już było przy okazji refakoryzacji powielonego kodu.

Kiedy stosujemy

Stosujemy, gdy z pewnych metod korzystają tylko niektóre obiekty.

Schemat postępowania

- dodanie nowej podklasy z identycznym konstruktorem;
- przeniesienie wybranych pól i metod do podklasy;
- zastąpienie wystąpień typu obiektu przez polimorfizm.

Zastosowanie

Sytuacja, w której w jednej klasie jest zarówno logika biznesowa jak i obsługa interfejsu graficznego.

Podejście 1.

Rozdzielenie kodu na dwie klasy

Model

Zawiera dane dziedzinowe oraz zawiera implementację logiki biznesowej.

Klasa implementująca interfejs graficzny

Implementacja obsługi przez GUI (np. edycja w AWT).

Podejście 1.

Rozdzielenie kodu na dwie klasy

Model

Zawiera dane dziedzinowe oraz zawiera implementację logiki biznesowej.

Klasa implementująca interfejs graficzny

Implementacja obsługi przez GUI (np. edycja w AWT).

Problem: rozdzielamy dane, które są w dwóch miejscach: w edytorze i w modelu. Trzeba je synchronizować.

Krok pierwszy

Wydzielenie klasy modelu

Tworzymy odrębną klasę zawierającą dane i logikę biznesową.

W starej klasie

Tworzymy referencję do obiektu nowej klasy.

Kompilacja i testowanie

Krok 2.

Dodajemy do edytora metodę

```
aktualizacja(Model m, Object dane) { ... }
```

która zaktualizuje dane w modelu

Krok 3.

Podłączenie aktualizacji do zdarzeń AWT.

Krok 4.

Zamiana odwołań do danych lokalnych na odwołania do pól modelu.

Plan wykładu

Przykład

```
void metoda(param1, param2) {
  zmienna = funkcja(param2);
}
```

Przykład

```
void metoda(param1, param2) {
  zmienna = funkcja(param2);
}

funkcja() {
  p1 = pobierzParam2();
  return p1;
}
void metoda(param1) {
  zmienna = funkcja();
}
```

Przekazanie całego obiektu

Już było ;-)

Utworzenie klasy dla grupy parametrów

Już było ;-)