Zaawansowane programowanie obiektowe - wykład 3

dr Piotr Jastrzębski

Wstęp do programowania obiektowego - cd.

Założenia/filary programowania obiektowego

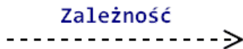
- Abstrakcja
- Hermetyzacja
- Dziedziczenie
- Polimorfizm

Ostatnia aktualizacja pliku: 2019-03-12 20:17:29.

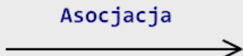
Kopiowanie

Temat klonowania, kopiowania a interfejs `ICloneable` - gist.

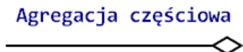
Związki pomiędzy klasami a UML



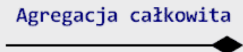
Gdy jedna klasa chwilowo wykorzystuje drugą, lub wie o jej istnieniu



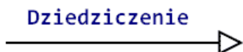
Gdy jedna klasa wykorzystuje drugą, ale nie są zależne



Gdy klasa zawiera drugą klasę, ale współdzieli odwołanie do niej z inną



Gdy klasa zawiera drugą klasę, i są od siebie zależne

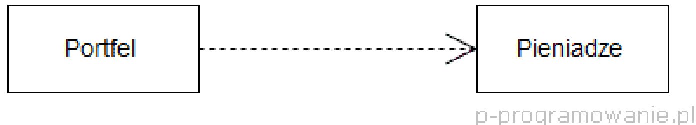


Gdy jedna klasa jest rozszerzeniem drugiej, i współdzielą swoje funkcjonalności

p-programowanie.pl

Zależność

Zależność – najsłabszy związek znaczeniowy między klasami, gdy jedna z nich używa innych klas. Na diagramie klas oznaczana przerywaną linią zakończoną strzałką wskazującą kierunek zależności.



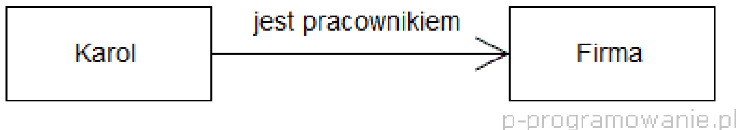
Rysunek 2:

```
class Portfel
{
   void Dodaj(Pieniadze p)
   {

   }
}
```

Asocjacja

Asocjacja wskazuje na trwałe powiązanie pomiędzy obiektami danych klas (np. firma zatrudnia pracowników). Na diagramie asocjację oznacza się za pomocą linii, która może być zakończona strzałką (oznaczającą kierunek powiązania klas). Nazwę cechy wraz z krotnością umieszcza się w punkcie docelowym asocjacji.



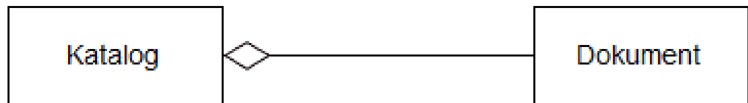
Rysunek 3:

Agregacja

Agregacja (inaczej zawieranie się, gromadzenie) – sytuacja, w której tworzy się nową klasę, używając klas już istniejących (często nazywa się to "tworzeniem obiektu składowego"). Nowa klasa może być zbudowana z dowolnej liczby obiektów (obiekty te mogą być dowolnych typów) i w dowolnej kombinacji, by uzyskać żądany efekt. Agregacja jest często określana jako relacja typu "zawiera" np. "samochód zawiera silnik" - gdzie "samochód" i "silnik" są klasami, oraz klasa "samochód" zawiera w sobie obiekt (czasami referencję czy wskaźnik) typu "silnik".

Agregacja częściowa

Agregacja częściowa - sytuacja, w której element częściowy może należeć do elementu głównego, jednak nie jest od niego zależny. Usunięcie elementu głównego nie wpływa na usunięcie elementu częściowego. Element częściowy może także należeć do wielu elementów głównych.



p-programowanie.pl

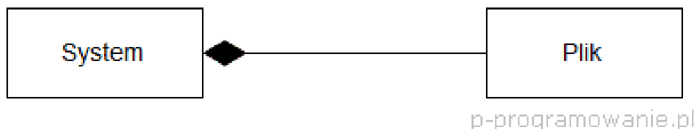
Rysunek 4:

```
class Katalog
{
   private Dokument swiadectwo;

   public void SetSwiatectwo(Dokument swiadectwo)
   {
      swiadectwo = swiadectwo;
   }
}
```

Agregacja całkowita

Agregacja całkowita - sytuacja, w której po usunięciu klasy głównej, zostanie usunięta klasa częściowa.



Rysunek 5:

```
class System
{
    private Plik _plik1 = new Plik();
}
```

```
class System
{
   private Plik plik1;

   public System()
   {
      plik1 = new Plik();
   }
}
```

Polimofizm

Definicja polimorfizmu wg wikipedii

Polimorfizm (z gr. wielopostaciowość) – mechanizmy pozwalające programiście używać wartości, zmiennych i podprogramów na kilka różnych sposobów. Inaczej mówiąc jest to możliwość wyabstrahowania wyrażeń od konkretnych typów.

Polimorfizm w C#

- statyczny
  - przeciążenie funkcji
  - przeciążenie operatorów
- dynamiczny
  - funkcje wirtualne
  - funkcje abstrakcyjne

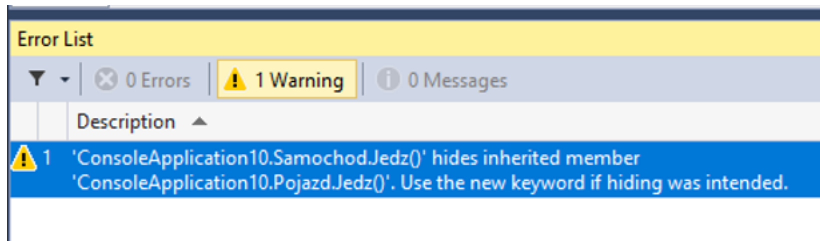
Napisanie metody - to nie jest polimorfizm

```
class Pojazd
{
    public void Jedz()
    {
        Console.WriteLine("Jadę pojazdem");
    }
}
class Samochod: Pojazd
{
    public void Jedz()
    {
        Console.WriteLine("Jadę samochodem");
    }
}
```

```
Pojazd p1 = new Pojazd();
p1.Jedz();
Samochod s1 = new Samochod();
s1.Jedz();
Pojazd sp1 = new Samochod();
sp1.Jedz();
```

Wykonanie kodu kończy się ostrzeżeniem kompilatora.

Podpowiedź sugeruje użycie `new` (czyli nadpisanie metody), jednak takie rozwiązanie nie jest najlepsze - nie daje elastyczności kodu.



Rysunek 6:

Metoda wirtualna

- oznaczona słowem kluczowym `virtual`
- włącza mechanizm polimorfizmu dynamicznego
- tworzymy ją w klasie bazowej
- w klasach pochodnych przesłaniamy metody wirtualne za pomocą słowa kluczowego `override`
- przesłonięcie nie jest obowiązkowe, w razie jego braku zostanie wywołana metoda klasy bazowej
- metody statyczne ani prywatne nie mogą być wirtualne
- metody, które nie są wirtualne, nie można przesłonić za pomocą `override`

```
class Pojazd
{
    public virtual void Jedz()
    {
        Console.WriteLine("Jadę pojazdem");
    }
}

class Samochod: Pojazd
{
    public override void Jedz()
    {
        Console.WriteLine("Jadę samochodem");
    }
}
```

```
Pojazd p1 = new Pojazd();
p1.Jedz();
Samochod s1 = new Samochod();
s1.Jedz();
Pojazd sp1 = new Samochod();
sp1.Jedz();
```

Po co taka konstrukcja?

- dajemy informację dla innej osoby zajmującej się kodem
- przy tworzeniu klasy potomnych mamy elastyczność: możemy zostawić to jak jest w klasie bazowej lub przesłonić

Metody wirtualne w klasie `Object`

- w C# każda klasa dziedziczy niejawnie z klasy `Object`

```
public class Object
{
    /* składowe wirtualne */
    virtual public bool Equals(object o);
    virtual protected void Finalize();
    virtual public string ToString();
    /* itp.. */
}
```

Metody abstrakcyjne

- poprzedzone słowem kluczowym `abstract`
- zdefiniowane w klasie bazowej
- nie zawierają ciała metody (podobnie jak przy interfejsach)
- mogą być zadeklarowane tylko w klasie abstrakcyjnej (poprzedzonej słowem `abstract`)
- nie możemy stworzyć egzemplarza (obiektu) klasy abstrakcyjnej (podobnie jak przy interfejsach), ale możemy dziedziczyć po klasie abstrakcyjnej
- klasa abstrakcyjna może posiadać zwykłe metody (z implementacją)
- klasa pochodna do klasy abstrakcyjnej musi przesłonić wszystkie metody abstrakcyjne

```
abstract class Pojazd
{
    public abstract void Jedz();
}

class Samochod: Pojazd
{
    public override void Jedz()
    {
        Console.WriteLine("Jadę samochodem");
    }
}
```

Wzorce projektowe

Wzorce projektowe

Wzorzec projektowy (ang. design pattern) – uniwersalne, sprawdzone w praktyce rozwiązanie często pojawiających się, powtarzalnych problemów projektowych. Pokazuje powiązania i zależności pomiędzy klasami oraz obiektami i ułatwia tworzenie, modyfikację oraz pielęgnację kodu źródłowego. Jest opisem rozwiązania, a nie jego implementacją. Wzorce projektowe stosowane są w projektach wykorzystujących programowanie obiektowe.

Elementy wzorca projektowego

Wzorzec projektowy składa się z czterech podstawowych elementów:

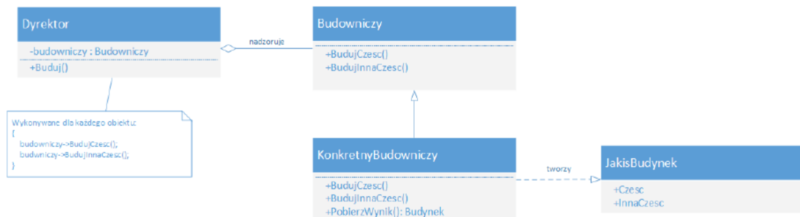
- nazwy wzorca;
- problemu – opisuje sposoby rozpoznawania sytuacji, w których możemy zastosować dany wzorzec oraz warunki jakie muszą zostać spełnione, by jego zastosowanie miało sens;
- rozwiązania – opisuje elementy rozwiązania: ich relacje, powiązania oraz obowiązki, zawiera także wskazówki implementacyjne dla różnych technologii;
- konsekwencji – zestawienie wad i zalet stosowania wzorca, uwzględniające informacje o jego brakach oraz kosztach rozwoju i utrzymania systemu wykorzystującego dany wzorzec.

Wzorce kreacyjne

Wzorce konstrukcyjne (kreacyjne) pozwalają w sposób abstrakcyjny tworzyć i konfigurować obiekty w celu ich wielokrotnego użycia i zachowania niezależności systemu od sposobu ich tworzenia.

Budowniczy

- cel: rozdzielenie sposobu tworzenia obiektów od ich reprezentacji
- proces tworzenia obiektu podzielony jest na kilka mniejszych etapów a każdy z tych etapów może być implementowany na wiele sposobów
- zastosowanie:
  - węzły XMLa
  - konwertowanie tekstu, zdjęć, filmów z jednego formatu na drugi.



Rysunek 7: Diagram UML Budowniczego

Kod do analizy: https://github.com/pjastr/ZPO_programy/tree/master/Budowniczy/Budowniczy

Fabryka abstrakcyjna

Przeanalizujmy przykład: http://tomaszjarzynski.pl/fabryka-abstrakcyjna-wzorzec-projektowy-abstract-factory/

Zastosowania:

- komunikacja z różnymi api
- wyświetlanie tekstu z różnych formatów plików
- obsługa różnych typów baz danych

Metoda wytwórcza

Przykład do przeanalizowania: http://tomaszjarzynski.pl/metoda-wytworcza-wzorzec-projektowy-factory-method/

Zastosowania:

- ukrycie przed klientem implementacji
- sterowniki

Prototyp

Prototyp – kreacyjny wzorzec projektowy, którego celem jest umożliwienie tworzenia obiektów danej klasy bądź klas z wykorzystaniem już istniejącego obiektu, zwanego prototypem.

Przykład do analizy: https://pl.wikibooks.org/wiki/Kody_%C5%BAr%C3%B3d%C5%82owe/Prototyp_(wzorzec_projektowy)#C#

Zastosowania

- duża liczba obiektów tego samego typu
- chcemy otrzymać prawidłową kopię obiektu

Singleton

Singleton - kreacyjny wzorzec projektowy, którego celem jest ograniczenie możliwości tworzenia obiektów danej klasy do jednej instancji oraz zapewnienie globalnego dostępu do stworzonego obiektu

```
public sealed class Singleton
{
    private static Singleton instance = null;

    private Singleton()  {  }

    public static Singleton Instance()
    {
        if (instance == null)
        {
            instance = new Singleton();
        }
        return instance;
    }

}
```

Bibliografia

- https://www.p-programowanie.pl/uml/diagramy-klas-uml/#Zwiazki-pomiedzy-klasami-w-UML, dostęp online 01.03.2019.
- Daniel Krasnokucki, Wzorce projektowe. Leksykon kieszonkowy, Wyd. Helion 2017.
- http://tomaszjarzynski.pl/metoda-wytworcza-wzorzec-projektowy-factory-method/, dostęp online 1.03.2019.
- http://tomaszjarzynski.pl/fabryka-abstrakcyjna-wzorzec-projektowy-abstract-factory/