
Polsko-Japońska Wyższa Szkoła Technik Komputerowych
Modelowanie Systemów Informacyjnych (MSI)
dr inż. Mariusz Trzaska
mtrzaska@pjwstk.edu.pl
Wykład 8
Realizacja asocjacji w obiektowych językach programowania (1)
http://www.mtrzaska.com

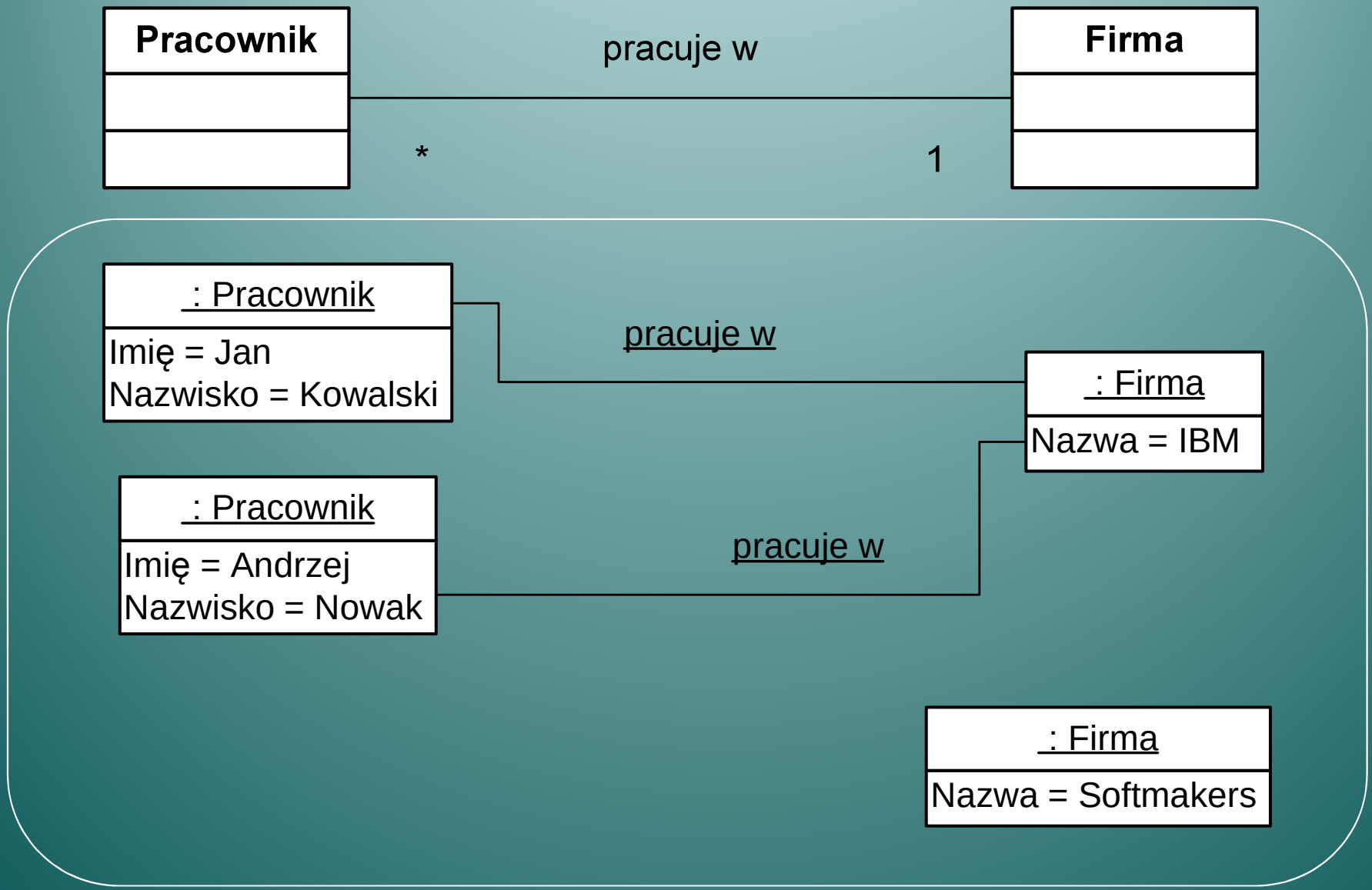
Zagadnienia

- Wstęp teoretyczny
- Implementacja asocjacji:
  - Przy pomocy identyfikatorów,
  - Korzystając z natywnych referencji.
- Implementacja asocjacji:
  - ze względu na liczności,
  - binarnych,
  - z atrybutem,
  - kwalifikowanych,
  - *n-arnych,*
- *Implementacja agregacji,*
- *Implementacja kompozycji,*
- *Uniwersalne zarządzanie asocjacjami,*
- *Podsumowanie*

Powiązania i Asocjacje

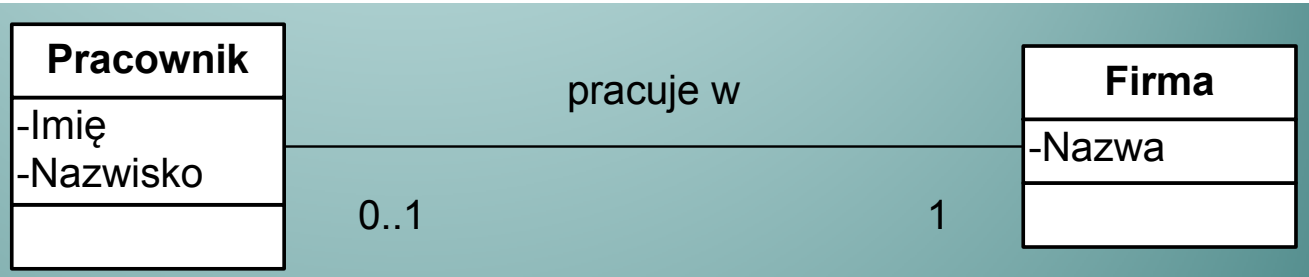
- **Powiązanie**. Zależność łącząca obiekty.
- Powiązania binarne.
- **Asocjacja**. Grupa powiązań o tej samej semantyce i strukturze.
- Służą do opisu zależności pomiędzy klasami.
- Powiązanie jest wystąpieniem asocjacji (tak jak obiekt jest wystąpieniem klasy).

Powiązania i Asocjacje (2)

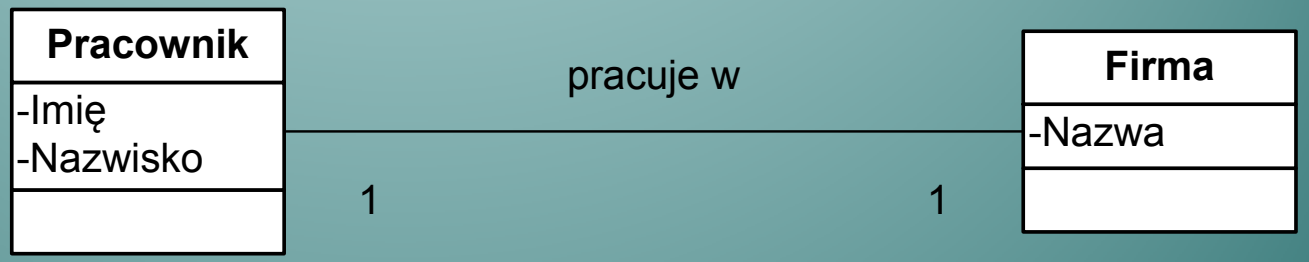


Liczności asocjacji

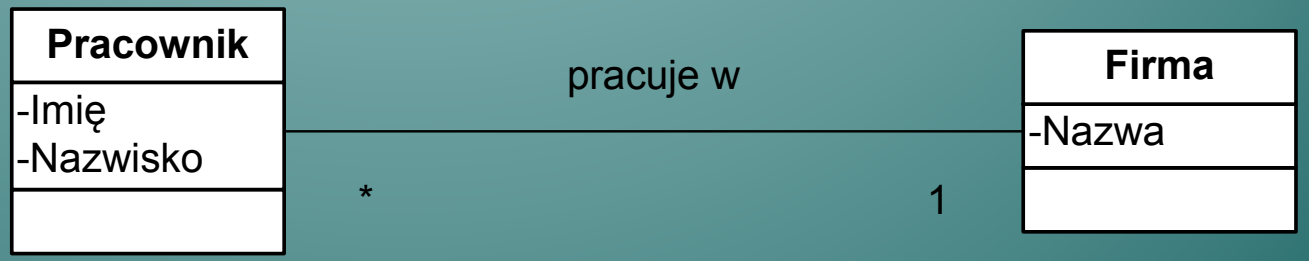
- 1 do 0, 1



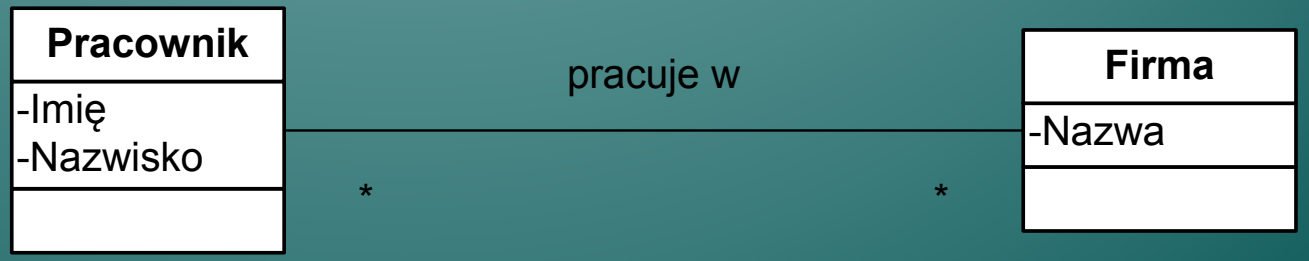
- 1 do 1



- 1 do *

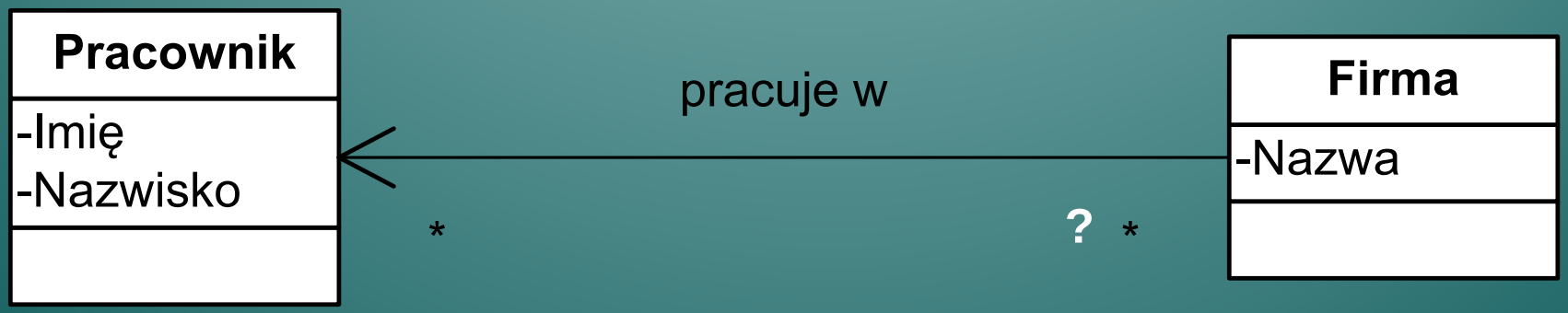


- * do *



Asocjacja skierowana

- Informacja biznesowa (zależność) jest przechowywana tylko w jednym kierunku.
- Zastosowania?
- Użyteczność?

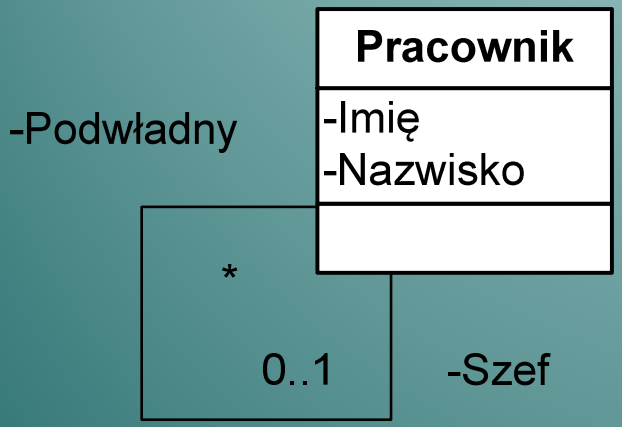


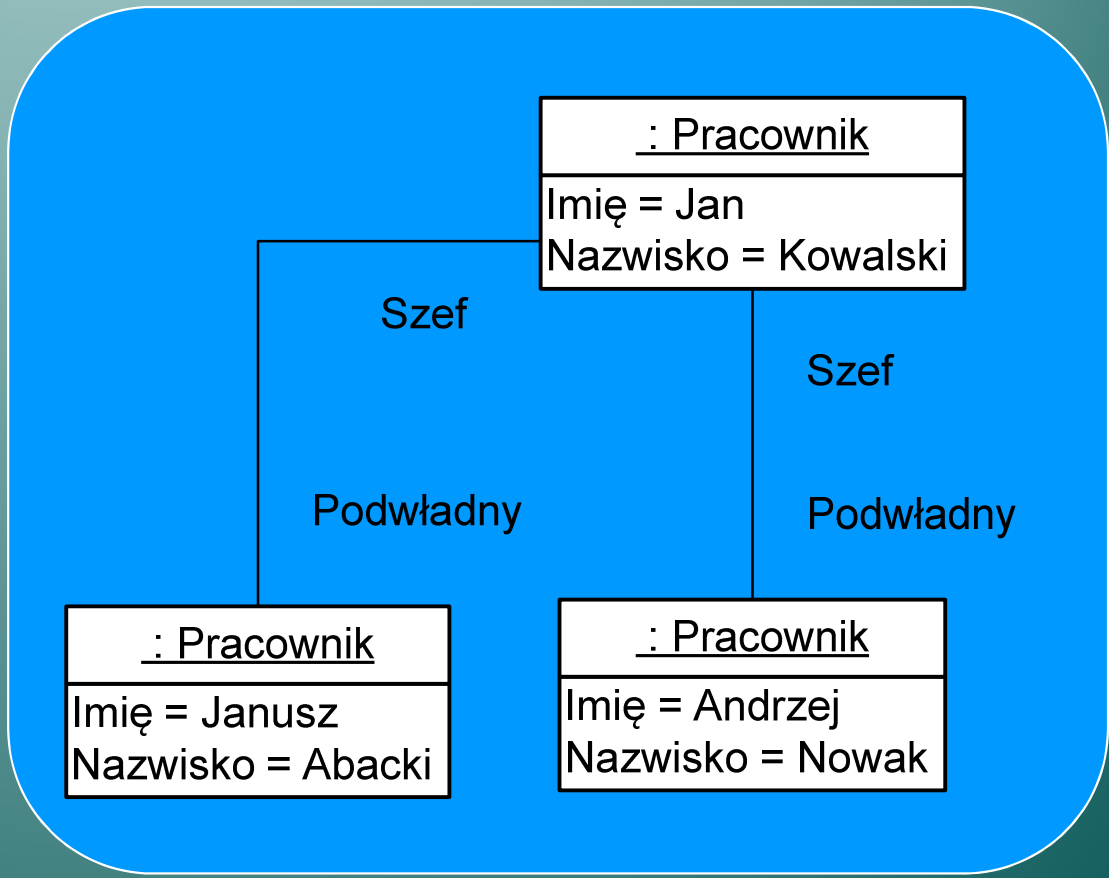
Role asocjacji

- Oprócz nazwy asocjacja może posiadać nazwy ról.
- Nazewnictwo
  - Nazwa asocjacji: pracuje w,
  - Nazwa roli: pracodawca
- Role
  - kiedy opcjonalne,
  - a kiedy wymagane?
- Użyteczność z punktu widzenia implementacji.

Asocjacja rekurencyjna

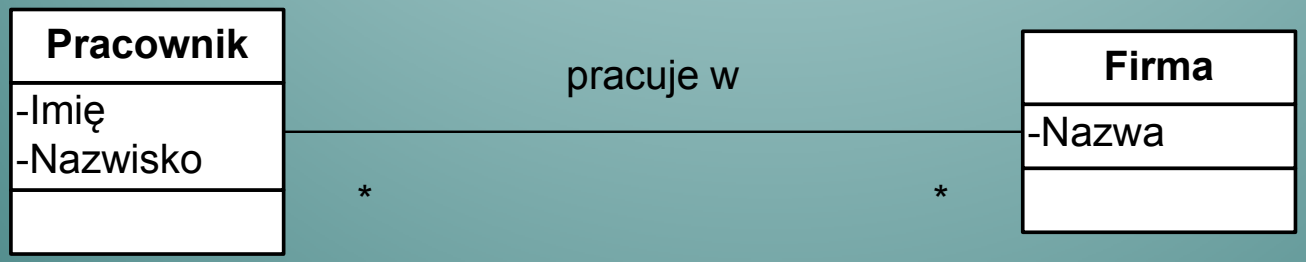
- Zachodzi w ramach tej samej klasy





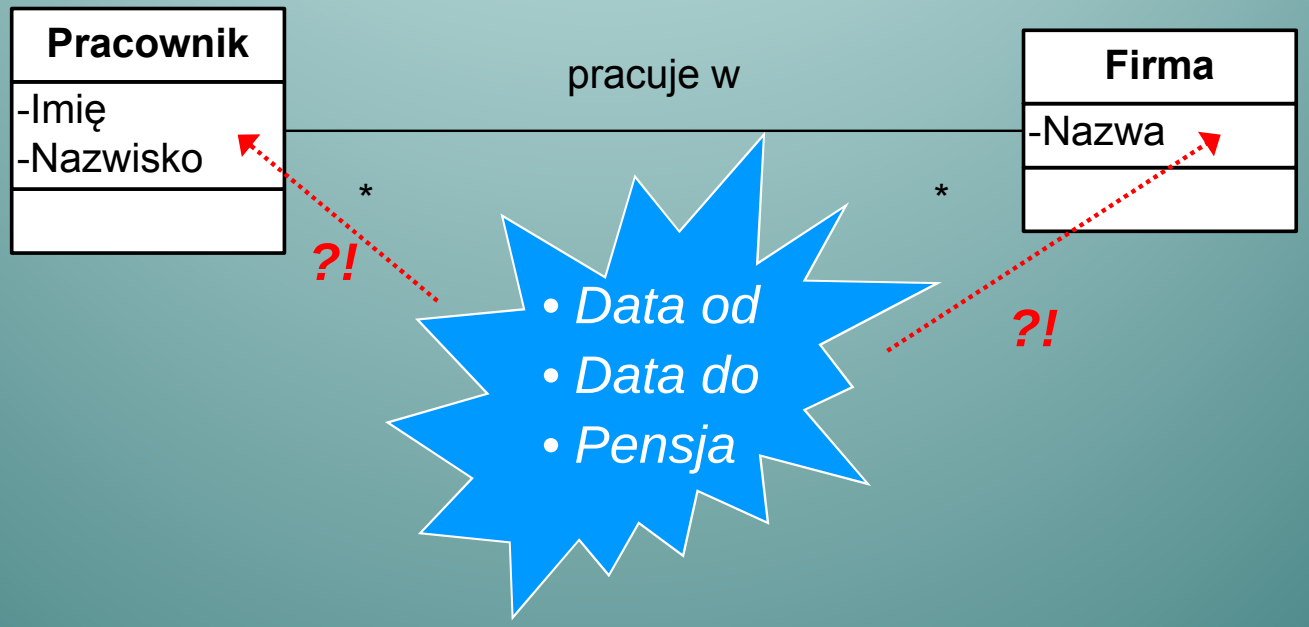
Asocjacja z atrybutem

- Załóżmy, że mamy przypadek biznesowy opisany poniższym diagramem

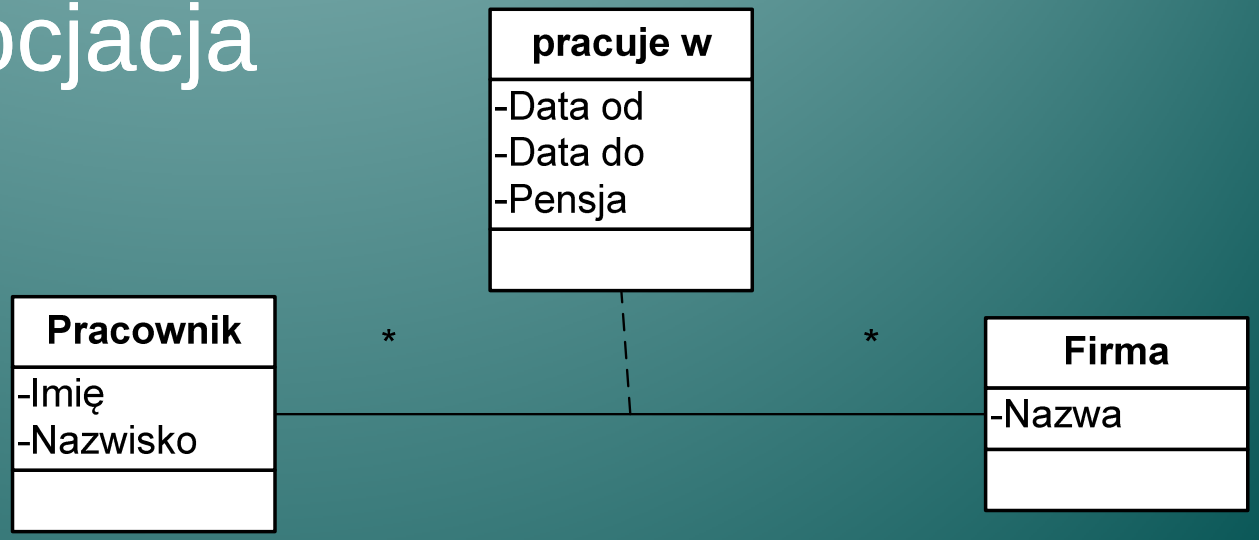


- Chcemy zapamiętać:
  - Kiedy pracownik pracował w danej firmie,
  - Ile tam zarabiał.
- Jak i gdzie umieścić takie informacje na diagramie?

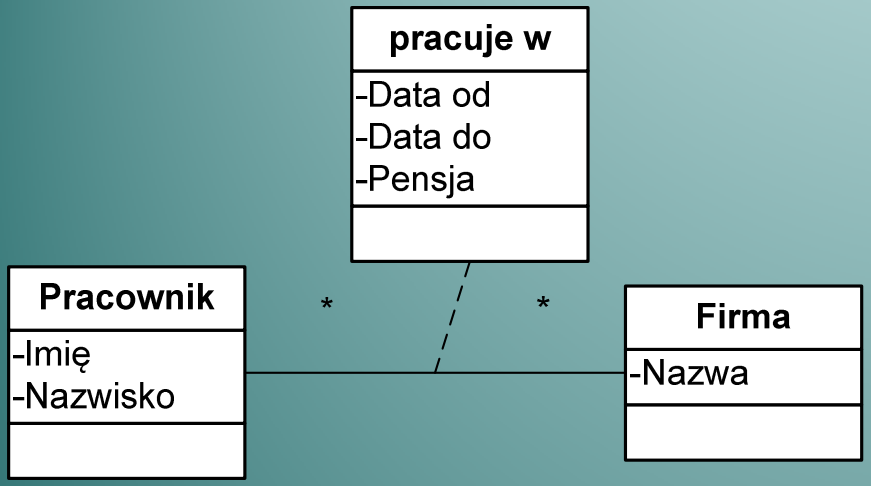
Asocjacja z atrybutem (2)



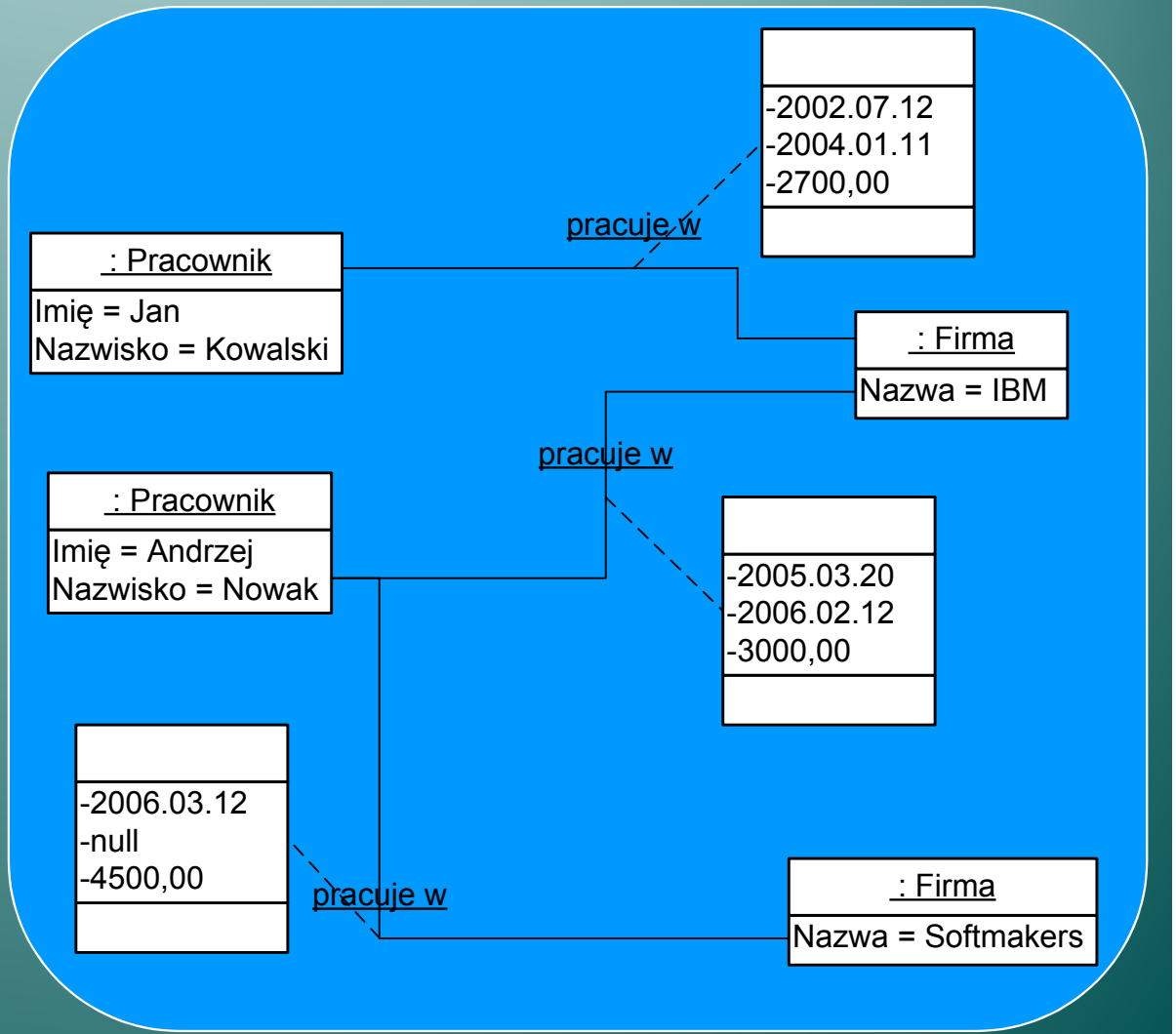
- Rozwiązanie asocjacja z atrybutem (klasa asocjacji)



Asocjacja z atrybutem (3)

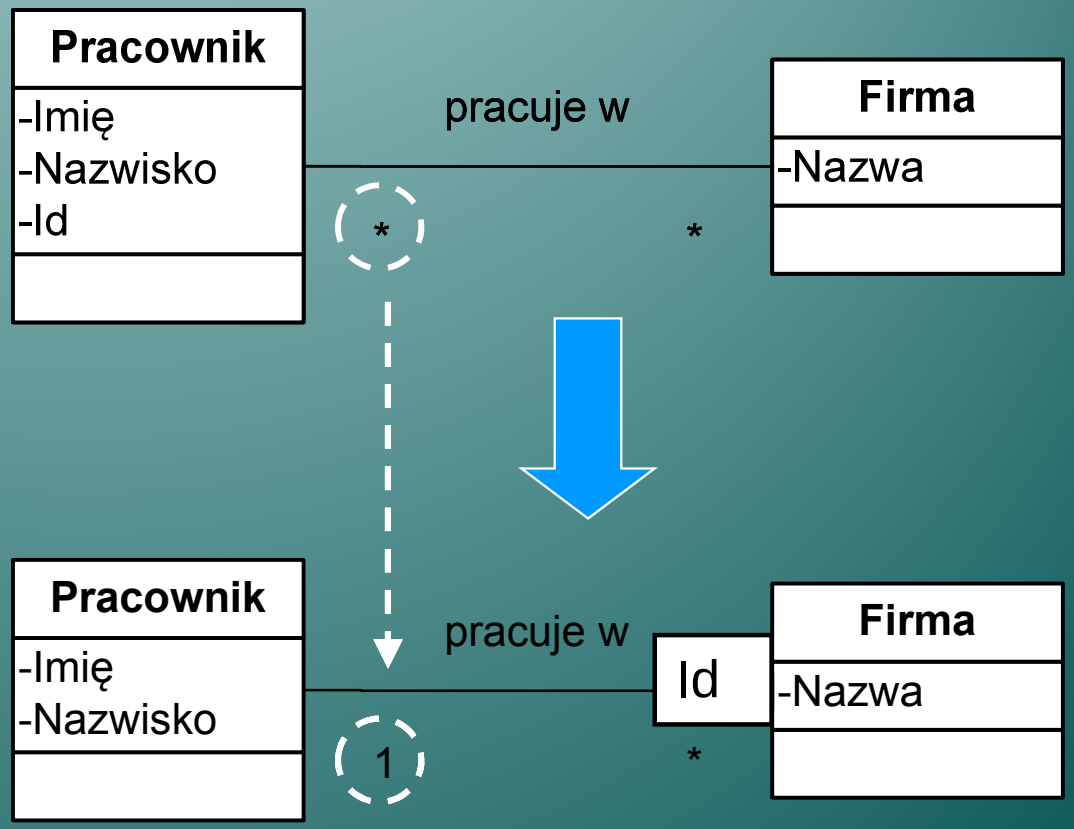


- Diagram klas oraz diagram obiektów



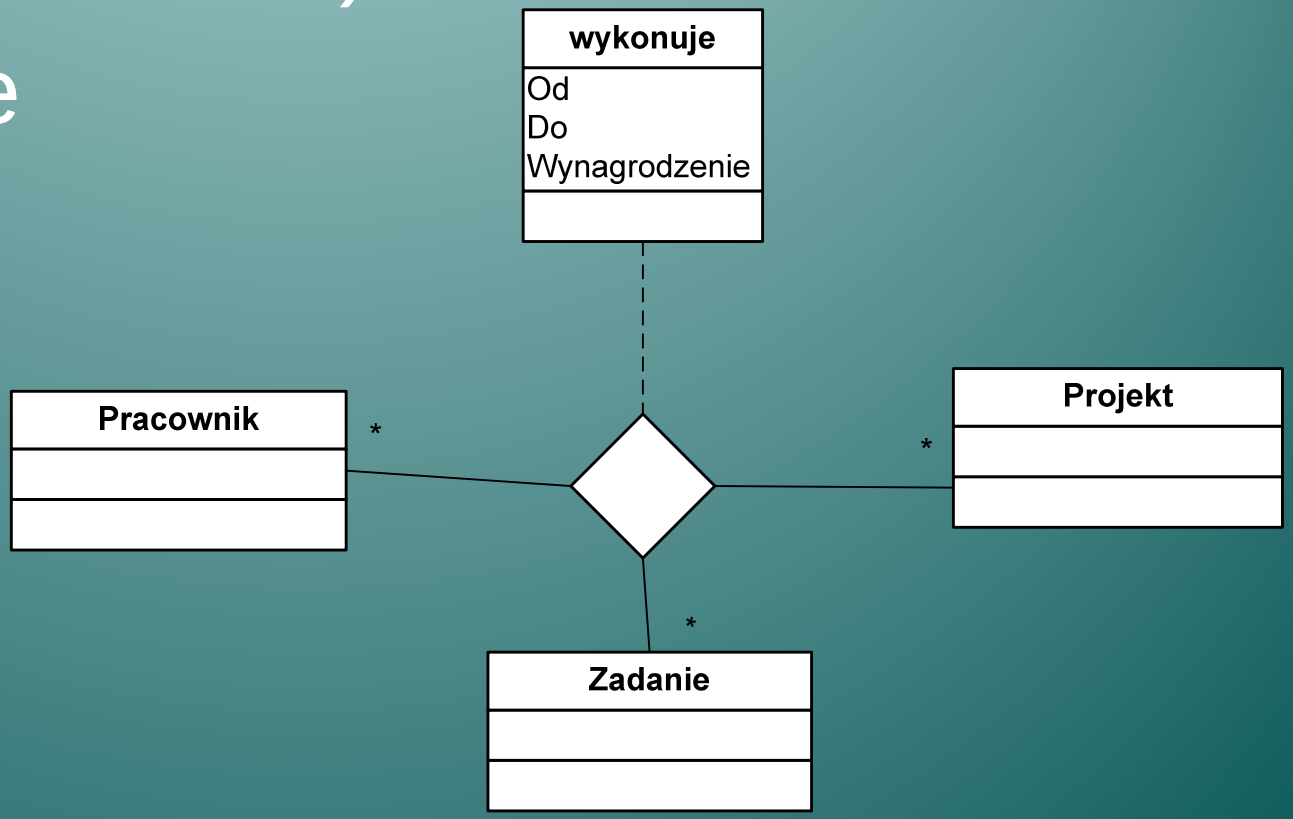
Asocjacja kwalifikowana

- Kwalifikator: atrybut lub zestaw atrybutów jednoznacznie identyfikujący obiekt w ramach asocjacji.
- Umożliwia szybkie otrzymanie obiektu z drugiej strony powiązania na podstawie kwalifikatora
- Indeksowanie (?)



Asocjacja n-arna

- Asocjacja łącząca n klas.
- Problemy koncepcyjne (w tym z szacowaniem liczności)
- Zakłada się, że liczności powinny być wiele.



Agregacja

- Asocjacja opisująca zależność typu część – całość:
  - Składa się z,
  - Należy do,
  - Jest członkiem,
  - Itp..
- Posiada wszystkie cechy „zwykłej” asocjacji.
- Ze względu na opisywanie specyficznej zależności warto zrezygnować z nazwy.
- Wykorzystanie asocjacji nie narzuca żadnych konsekwencji na związek łączący klasy.

Kompozycja

- Kompozycja jest mocniejszą forma agregacji.
- Czyli jest również asocjacją, a zatem posiada wszystkie jej cechy.
- Inaczej niż w przypadku agregacji, wykorzystanie kompozycji niesie pewne konsekwencje:
  - Część nie może być współdzielona (liczności),
  - Część nie może istnieć bez całości (całość może),
  - Usunięcie całości oznacza usunięcie również wszystkich jej części.

Kompozycja (2)

- Agregacja

- Kompozycja

Asocjacje, a języki programowania

- Jak się mają podane definicje do popularnych, obiektowych języków programowania?
- W językach
  - Java,
  - MS C#,
  - C++

  asocjacje **nie występują**.
- Oczywiście można je samodzielnie zaimplementować.

Implementacja asocjacji

- Można zastosować dwa podejścia. Zasadnicza różnica polega na sposobie przechowywania informacji o powiązaniu obiektów:
  - Identyfikatory, np. liczbowe,
  - Natywne referencje języka (Java, C#) lub wskaźniki (C++).
- Które podejście jest lepsze?

Wykorzystanie identyfikatorów

- Do każdej klasy dodajemy atrybut będący identyfikatorem, np. liczbę `int`.
- Informacje o powiązanych obiektach przechowujemy pamiętając ich identyfikatory (uwaga na liczności).
- Konieczność tworzenia par identyfikatorów (dla asocjacji dwukierunkowych)

:Film	:Film
Id = 1	Id = 2
Tytuł = „T1”	Tytuł = „T3”
Aktor = [3, 4]	**Aktor = [3]**

:Aktor	:Aktor
Id = 3	Id = 4
Imię i Nazwisko = „AS”	Imię i Nazwisko = „MB”
Film = [1, 2]	**Film = [1]**

:Aktor
Id = 3
Imię i Nazwisko = „KL”
Film = [2]

Wykorzystanie identyfikatorów (2)

```
public class Aktor {
     private int id;
     public String imieNazwisko;// public dla uproszczenia
     public int[] film;
     private static ArrayList<Aktor> ekstensja = new ArrayList<Aktor>();

     public Aktor(int id, String imieNazwisko, int[] filmId) {
          // Dodaj do ekstensji
          ekstensja.add(this);

          this.id = id;
          this.imieNazwisko = imieNazwisko;
          film = filmId;
     }

     public static Aktor znajdzAktora(int id) throws Exception {
          for(Aktor aktor : ekstensja) {
               if(aktor.id == id) {
                    return aktor;
               }
          }
          throw new Exception("Nie znaleziono aktora o id = " + id);
     }
}
```

Wykorzystanie identyfikatorów (3)

```
public class Film {
     public int id;
     public String tytul; // public dla uproszczenia
     public int[] aktor;
     private static ArrayList<Film> ekstensja = new ArrayList<Film>();

     public Film(int id, String tytul, int[] aktorId) {
          // Dodaj do ekstensji
          ekstensja.add(this);

          this.id = id;
          this.tytul = tytul;
          aktor = aktorId;
     }

     public static Film znajdzFilm(int id) throws Exception {
          for(Film film : ekstensja) {
               if(film.id == id) {
                    return film;
               }
          }
          throw new Exception("Nie znaleziono filmu o id = " + id);
     }
}
```

Wykorzystanie identyfikatorów (4)

```
Film film1 = new Film(1, "T1", new int[]{3, 4});
Film film2 = new Film(2, "T3", new int[]{3});

Aktor aktor1 = new Aktor(3, "AS", new int[]{1, 2});
Aktor aktor2 = new Aktor(4, "MB", new int[]{1});
Aktor aktor3 = new Aktor(5, "KL", new int[]{3});

// Wyswietl info o filmie1
System.out.println(film1.tytul);
for(int i = 0; i < film1.aktor.length; i++) {
     System.out.println("   " + Aktor.znajdzAktora(film1.aktor[i]).imieNazwisko);
}

// Wyswietl info o aktor1
System.out.println(aktor1.imieNazwisko);
for(int i = 0; i < aktor1.film.length; i++) {
     System.out.println("   " + Film.znajdzFilm(aktor1.film[i]).tytul);
}
```

```
T1
  AS
  MB
AS
  T1
  T3
```

Wykorzystanie identyfikatorów (5)

- Wady:
  - Konieczność wyszukiwania obiektu na podstawie jego numeru identyfikacyjnego – problemy z wydajnością. Wydajność wyszukiwania można zdecydowanie poprawić używając kontenerów mapujących zamiast zwykłych, np.:

```
public class Film {
     private static Map<Integer, Film> ekstensja = new TreeMap<Integer, Film>();
     // [...]
     public Film(int id, String tytul, int[] aktorId) {
          // Dodaj do ekstensji
          ekstensja.put(new Integer(id), this);
     }
     // [...]
     public static Film znajdzFilm(int id) throws Exception {
          return ekstensja.get(new Integer(id));
     }
}
```

  - Mimo wszystko i tak trzeba wyszukiwać…

Wykorzystanie identyfikatorów (6)

- Zalety (właściwie jedna (?), ale czasami ważna):
  - Uniezależnienie poszczególnych obiektów od siebie.
  - Dzięki temu, że nie używamy referencji języka Java, maszyna wirtualna nic nie wie o naszych powiązaniach.
  - Jest to bardzo ważne, gdy np. chcemy odczytać jeden obiekt z bazy danych lub przesłać przez sieć:
    - Tworzony jest obiekt języka Java na podstawie zawartości BD, ale nie są tworzone obiekty z nim powiązane,
    - Całość można tak zaprojektować, aby dopiero przy próbie dostępu do obiektu wskazywanego przez id, pobrać go z BD czy lokalizacji sieciowej.
    - Dzięki temu nie musimy rekonstruować od razu całego grafu obiektów, który nie musi być nam potrzebny.

Wykorzystanie natywnych referencji

- W celu „pokazywania” na powiązane obiekty wykorzystujemy natywne referencje języka (Java, C#) lub wskaźniki (C++).
- Dzięki temu nie musimy odszukiwać obiektów;
- Mając jego referencje (lub wskaźnik) mamy do niego natychmiastowy dostęp (szybciej już się nie da).
- Konieczność tworzenia par referencji (jeżeli chcemy nawigować w dwie strony – a przeważnie chcemy).

Wykorzystanie natywnych referencji (2)

- W zależności od liczności asocjacji, wykorzystujemy różne konstrukcje:
  - 1 do 1. Pojedyncza referencja z każdej strony.

```
public class Aktor {
	public String imieNazwisko;// public dla uproszczenia
	public Film film; // impl. asocjacji, licznosc 1
	public Aktor(String imieNazwisko, Film film) {
		this.imieNazwisko = imieNazwisko;
		this.film = film;
	}
}
public class Film {
	public String tytul; // public dla uproszczenia
	public Aktor aktor; // impl. asocjacji, licznosc 1
	public Film(String tytul, Aktor aktor) {
		this.tytul = tytul;
		this.aktor = aktor;
	}
}
```

Wykorzystanie natywnych referencji (3)

- W zależności od liczności asocjacji, wykorzystujemy różne konstrukcje:
  - 1 do *. Pojedyncza referencja oraz kontener.

```
public class Aktor {
	public String imieNazwisko;// public dla uproszczenia
	public Film film; // impl. asocjacji, licznosc 1
	public Aktor(String imieNazwisko, Film film) {
		this.imieNazwisko = imieNazwisko;
		this.film = film;
	}
}
public class Film {
	public String tytul; // public dla uproszczenia
	public ArrayList<Aktor> aktor; // impl. asocjacji, licznosc *

	public Film(String tytul, Aktor aktor) {
		this.tytul = tytul;
		this.aktor.add(aktor);
	}
}
```

Wykorzystanie natywnych referencji (4)

- W zależności od liczności asocjacji, wykorzystujemy różne konstrukcje:
  - * do *. Dwa kontenery.

```
public class Aktor {
     public String imieNazwisko;// public dla uproszczenia
     public ArrayList<Film> film; // impl. asocjacji, licznosc *

     public Aktor(String imieNazwisko) {
          this.imieNazwisko = imieNazwisko;
     }
}
public class Film {
     public String tytul; // public dla uproszczenia
     public ArrayList<Aktor> aktor; // impl. asocjacji, licznosc *

     public Film(String tytul) {
          this.tytul = tytul;
     }
}
```

Ulepszone zarządzanie powiązaniami

- Poprzednie podejście wymagało ręcznego dodawania informacji o powiązaniu zwrotnym.
- Warto to jakoś zautomatyzować.
- Stworzymy metodę, która doda informacje o powiązaniu:
  - W klasie głównej,
  - W klasie z nią powiązanej.
- Całość musi być tak zaprojektowana aby nie dochodziło do zapętlenia.

Ulepszone zarządzanie powiązaniami (2)

```
public class Aktor {
     public String imieNazwisko;// public dla uproszczenia
     private ArrayList<Film> film = new ArrayList<Film>(); // impl. asocjacji, licznosc *
     public Aktor(String imieNazwisko) {
          this.imieNazwisko = imieNazwisko;
     }
     public void dodajFilm(Film nowyFilm) {
          // Sprawdz czy nie mamy juz informacji o tym filmie
          if(!film.contains(nowyFilm)) {
               film.add(nowyFilm);

               // Dodaj informacje zwrotna
               nowyFilm.dodajAktor(this);
          }
     }
     public String toString() {
          String info = "Aktor: " + imieNazwisko + "\n";
          // Dodaj info o tytulach jego filmow
          for(Film f : film) {
               info += "   " + f.tytul + "\n";
          }
          return info;
     }
}
```

Ulepszone zarządzanie powiązaniami (3)

```
public class Film {
     public String tytul; // public dla uproszczenia
     private ArrayList<Aktor> aktor = new ArrayList<Aktor>(); // impl. asocjacji, licznosc *
     public Film(String tytul) {
          this.tytul = tytul;
     }
     public void dodajAktor(Aktor nowyAktor) {
          // Sprawdz czy nie mamy juz takiej informacji
          if(!aktor.contains(nowyAktor)) {
               aktor.add(nowyAktor);
               // Dodaj informacje zwrotna
               nowyAktor.dodajFilm(this);
          }
     }
     public String toString() {
          String info = "Film: " + tytul + "\n";
          // Dodaj info o tytulach jego filmow
          for(Aktor a : aktor) {
               info += "   " + a.imieNazwisko + "\n";
          }

          return info;
     }
}
```

Ulepszone zarządzanie powiązaniami (4)

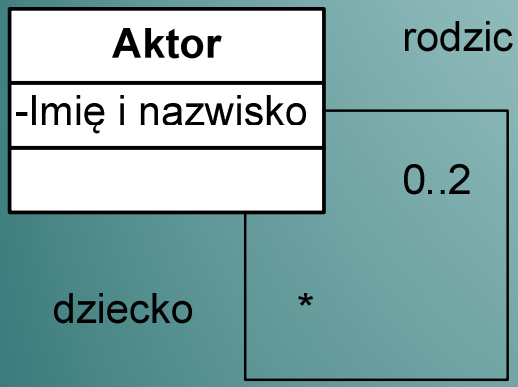
```
// Utworz nowe obiekty (na razie bez informacji o powiazaniach)
Film film1 = new Film("T1");
Film film2 = new Film("T3");

Aktor aktor1 = new Aktor("AS");
Aktor aktor2 = new Aktor("MB");
Aktor aktor3 = new Aktor("KL");

// Dodaj informacje o powiazaniach
film1.dodajAktor(aktor1);
film1.dodajAktor(aktor2);
film2.dodajAktor(aktor1);
film2.dodajAktor(aktor3);

// Wyswietl info o filmach
System.out.println(film1);
System.out.println(film2);

// Wyswietl info o aktorach
System.out.println(aktor1);
System.out.println(aktor2);
System.out.println(aktor3);
```

```
Film: T1
  AS
  MB

Film: T3
  AS
  KL

Aktor: AS
  T1
  T3

Aktor: MB
  T1

Aktor: KL
  T3
```

Implementacja asocjacji skierowanej

- Powyższy diagram oznacza, że dla:
  - konkretnego filmu chcemy znać jego aktorów,
  - Konkretnego aktora nie chcemy znać jego filmów.
- Implementacja jest analogiczna do poprzednich przypadków, z tym, że pamiętamy tylko informacje w jednej z klas:
  - W klasie film jest odpowiedni kontener (pokazujący na filmy),
  - W klasie aktor brak kontenera przechowującego info o filmach.

Implementacja asocjacji rekurencyjnej

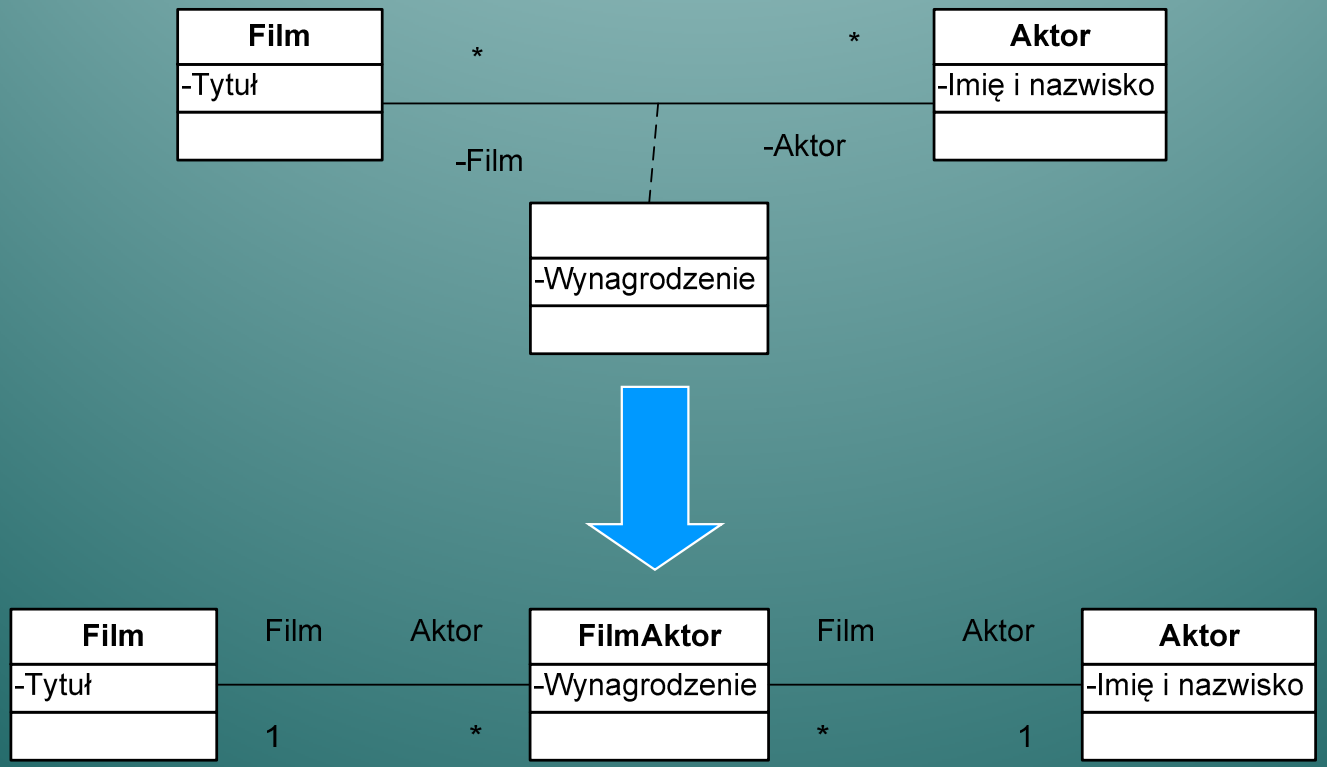


- Implementacja takiej asocjacji bazuje na poprzednio poznanych zasadach.
- W klasie mamy dwa kontenery (zamiast jednego), przechowujące informacje z punktu widzenia każdej z ról.

```
public class Aktor {
     public String imieNazwisko;// public dla uproszczenia
     private ArrayList<Film> film = new ArrayList<Film>(); // impl. asocjacji, licznosc

     private ArrayList<Aktor> rodzic = new ArrayList<Aktor>();
     private ArrayList<Aktor> dziecko = new ArrayList<Aktor>();
     // [...]
}
```

Implementacja asocjacji z atrybutem

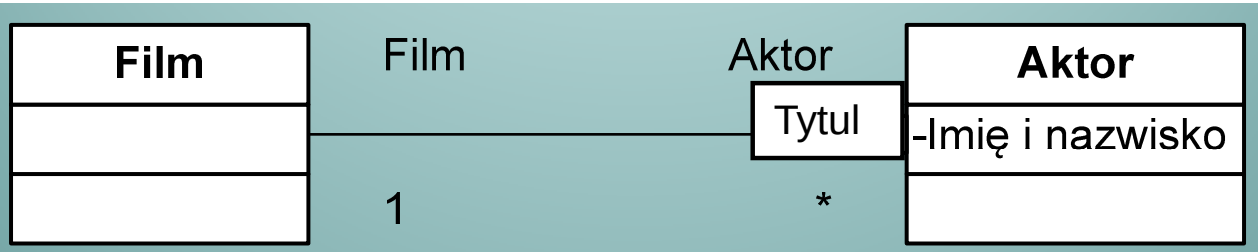
- Najpierw musimy zamienić:
  - jedną konstrukcję UML (asocjacja z atrybutem)
  - na inną konstrukcję UML (asocjacją z klasą pośredniczącą).



Implementacja asocjacji z atrybutem (2)

- Dzięki zastąpieniu atrybutu asocjacji, klasą pośredniczącą otrzymaliśmy dwie „zwykłe” asocjacje.
- „Nowe” asocjacje implementujemy na jeden ze znanych sposobów.
- Problem z semantyką tej nowej klasy
  - Nazwa?
  - Nazwy ról: „starych” oraz „nowych”
- Utrudniony dostęp do obiektów docelowych (poprzez obiekt klasy pośredniczącej)

Implementacja asocjacji kwalifikowanej



- Najprostsza implementacja:
  - Korzystamy z istniejącego podejścia,
  - Dodajemy metodę, która na podstawie tytułu zwróci nam obiekt klasy Film,
  - Słaba wydajność.
- Lepsze rozwiązanie:
  - Nie korzystamy z `ArrayList`,
  - Stosujemy kontener mapujący, gdzie kluczem jest tytuł, a wartością obiekt opisujący film.
  - Informacja zwrotna w dotychczasowy sposób.

Implementacja asocjacji kwalifikowanej (2)

```
public class Aktor {
     // [...]

     private TreeMap<String, Film> filmKwalif = new TreeMap<String, Film>();

     public void dodajFilmKwalif(Film nowyFilm) {
          // Sprawdz czy nie mamy juz informacji o tym filmie
          if(!filmKwalif.containsKey(nowyFilm.tytul)) {
               filmKwalif.put(nowyFilm.tytul, nowyFilm);
               // Dodaj informację zwrotną
               nowyFilm.dodajAktor(this);
          }
     }
     public Film znajdzFilmKwalif(String tytul) throws Exception {
          // Sprawdz czy nie mamy juz informacji o tym filmie
          if(!filmKwalif.containsKey(tytul)) {
               throw new Exception("Nie odnaleziono filmu o tytule: " + tytul);
          }
          return filmKwalif.get(tytul);
     }
     // [...]
}
```

Implementacja asocjacji kwalifikowanej (3)

```
// Utworz nowe obiekty (na razie bez informacji o powiazaniach)
Film film1 = new Film("T1");
Film film2 = new Film("T3");

Aktor aktor1 = new Aktor("AS");
Aktor aktor2 = new Aktor("MB");
Aktor aktor3 = new Aktor("KL");

// Dodaj informacje o powiazaniach
aktor1.dodajFilmKwalif(film1);
aktor1.dodajFilmKwalif(film2);
aktor2.dodajFilmKwalif(film1);
aktor3.dodajFilmKwalif(film2);

// Wyswietl info o aktorach
System.out.println(aktor1);
System.out.println(aktor2);
System.out.println(aktor3);

// Pobierz info o filmie "T1" dla aktora aktor1
Film f = aktor1.znajdzFilmKwalif("T1");
System.out.println(f);
```

```
Aktor: AS
  T1
  T3

Aktor: MB
  T1

Aktor: KL
  T3

Film: T1
  AS
  MB
```

Ciąg dalszy na następnym wykładzie...