



Modelowanie i Analiza Systemów informacyjnych (MAS)

dr inż. Mariusz Trzaska
mtrzaska@pjwstk.edu.pl

Wykład 5

Realizacja asocjacji w obiektowych językach programowania (1)

Zagadnienia

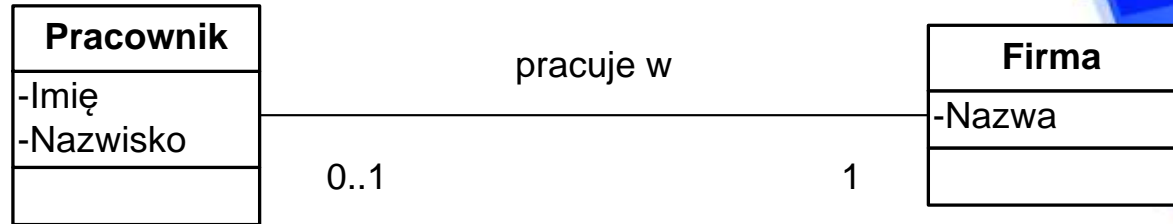
- Wstęp teoretyczny
- Implementacja asocjacji:
 - Przy pomocy identyfikatorów,
 - Korzystając z natywnych referencji.
- Implementacja asocjacji:
 - ze względu na licznosci,
 - binarnych,
 - z atrybutem,
 - kwalifikowanych,
 - n-arnych,
- Implementacja agregacji,
- Implementacja kompozycji,
- Uniwersalne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

Powiązania i Asocjacje

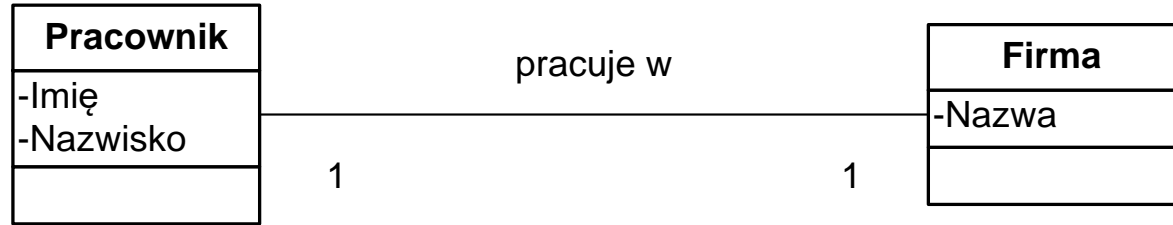
- Powiązanie. Zależność łącząca obiekty.
- Powiązania binarne.
- Asocjacja. Grupa powiązań o tej samej semantyce i strukturze.
- Służą do opisu zależności pomiędzy klasami.
- Powiązanie jest wystąpieniem asocjacji (tak jak obiekt jest wystąpieniem klasy).

Liczności asocjacji

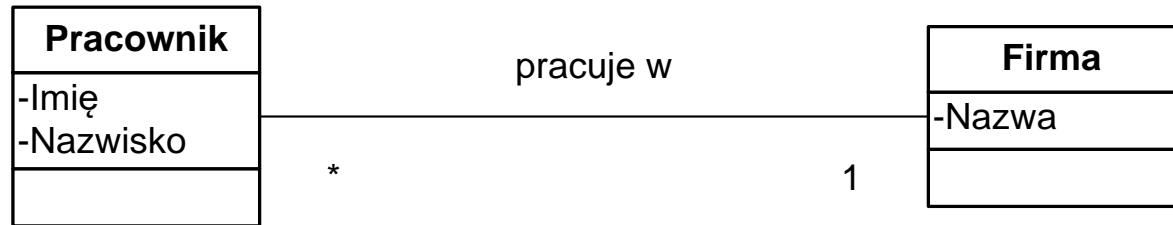
- 1 do 0, 1



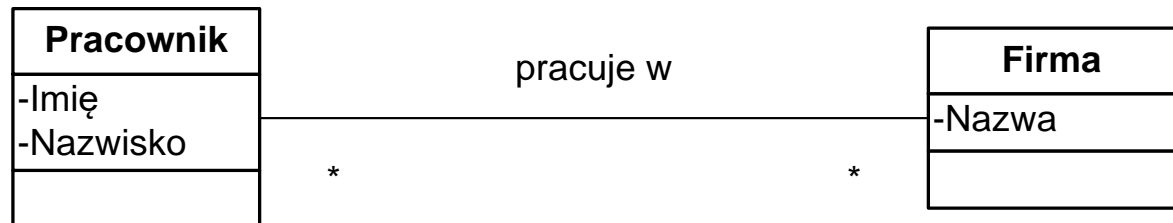
- 1 do 1



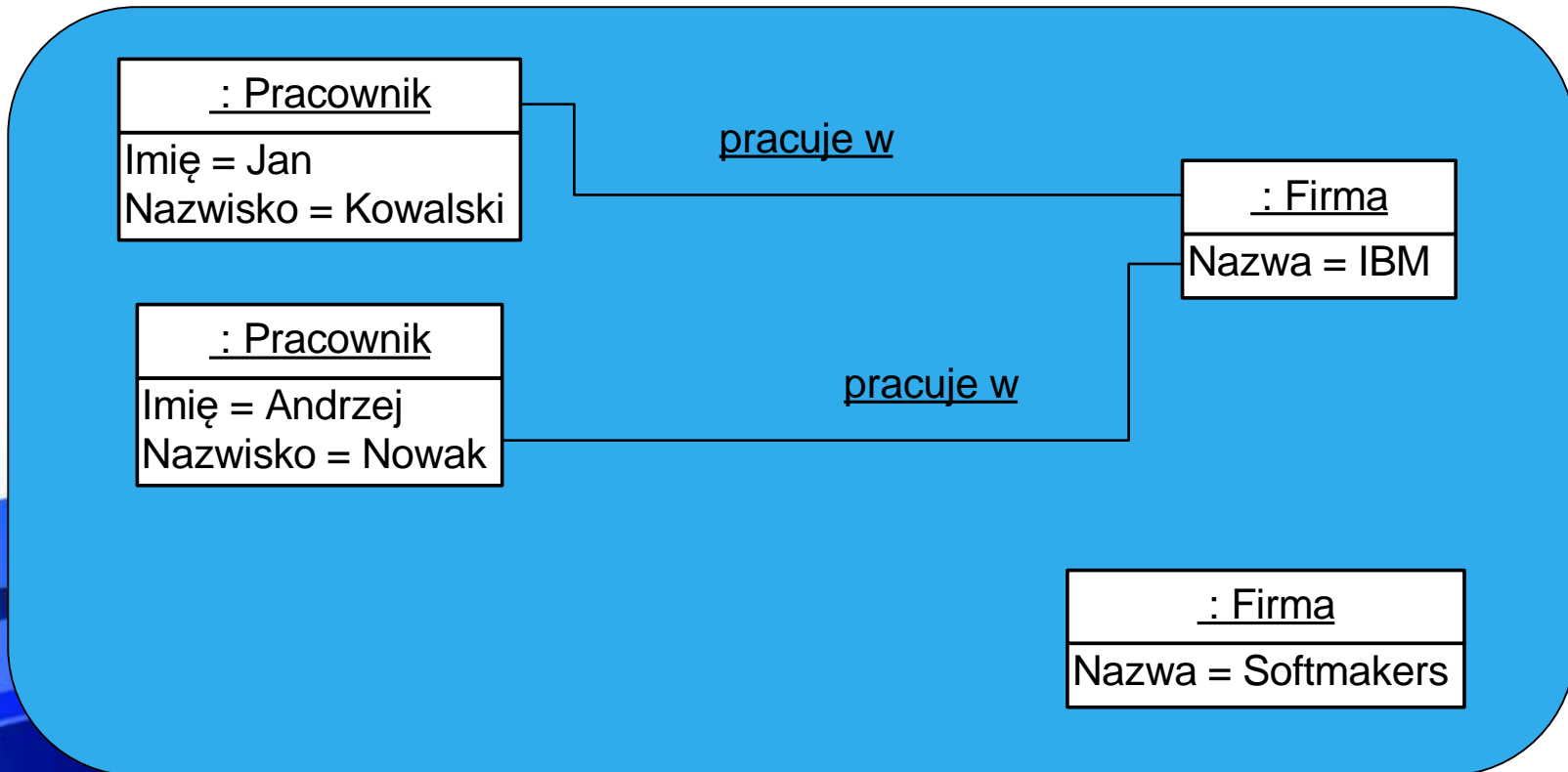
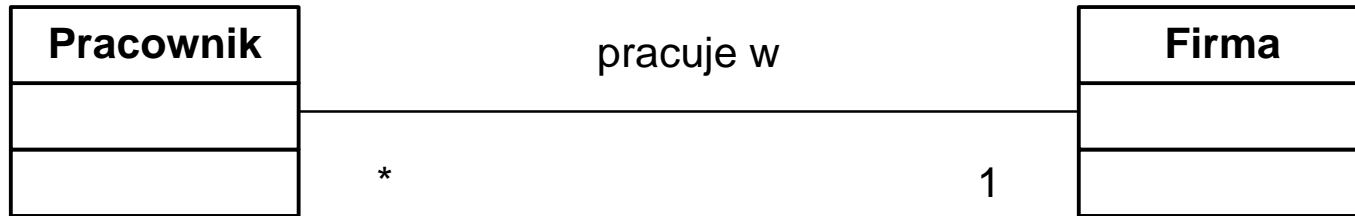
- 1 do *



- * do *

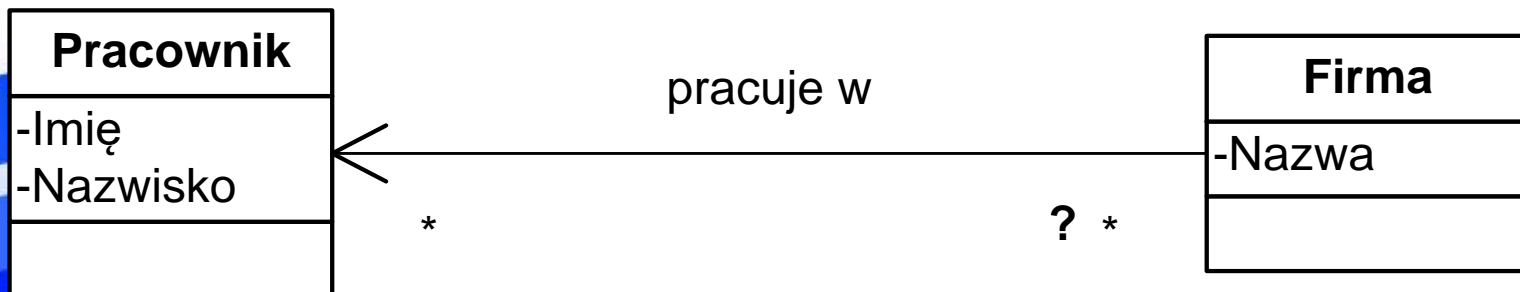


Powiązania i Asocjacje - przykład



Asocjacja skierowana

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

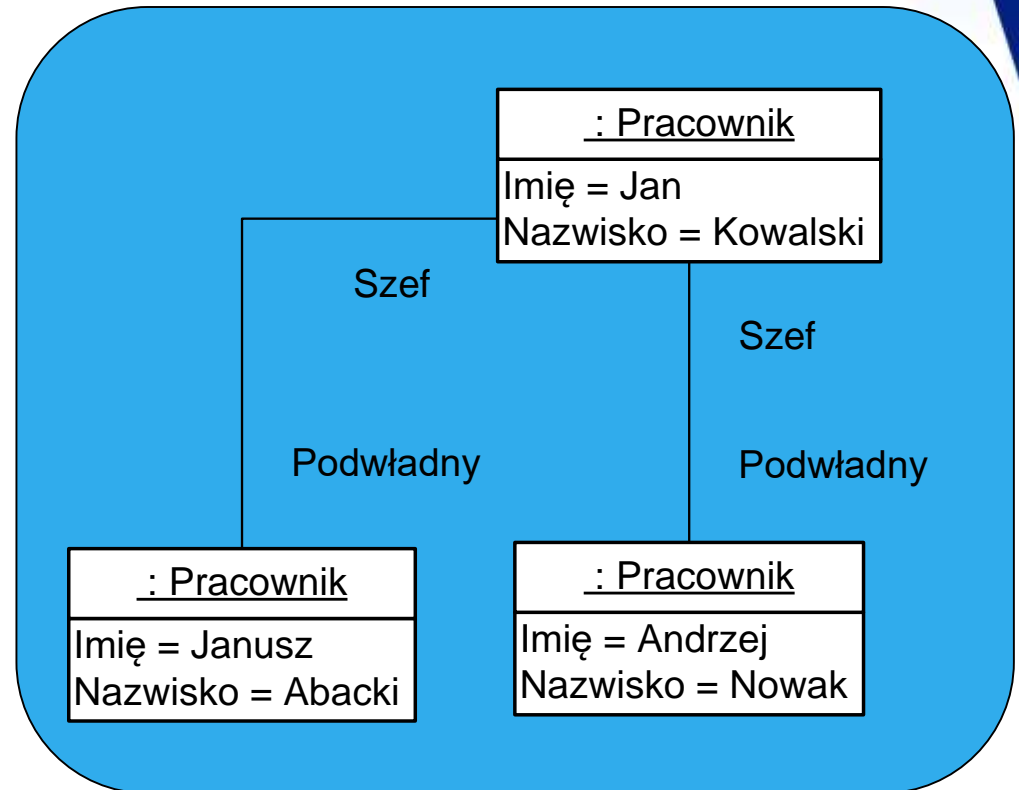
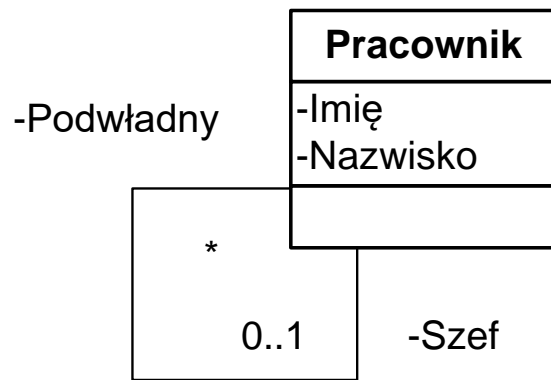


Role asocjacji

- 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.

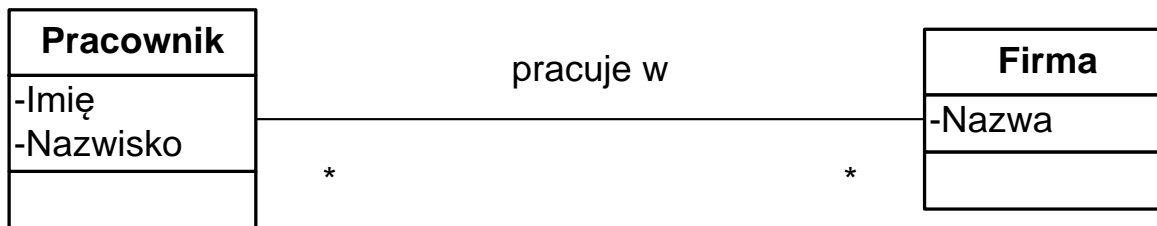
Asocjacja rekurencyjna

- Zachodzi w ramach tej samej klasy



Asocjacja z atrybutem

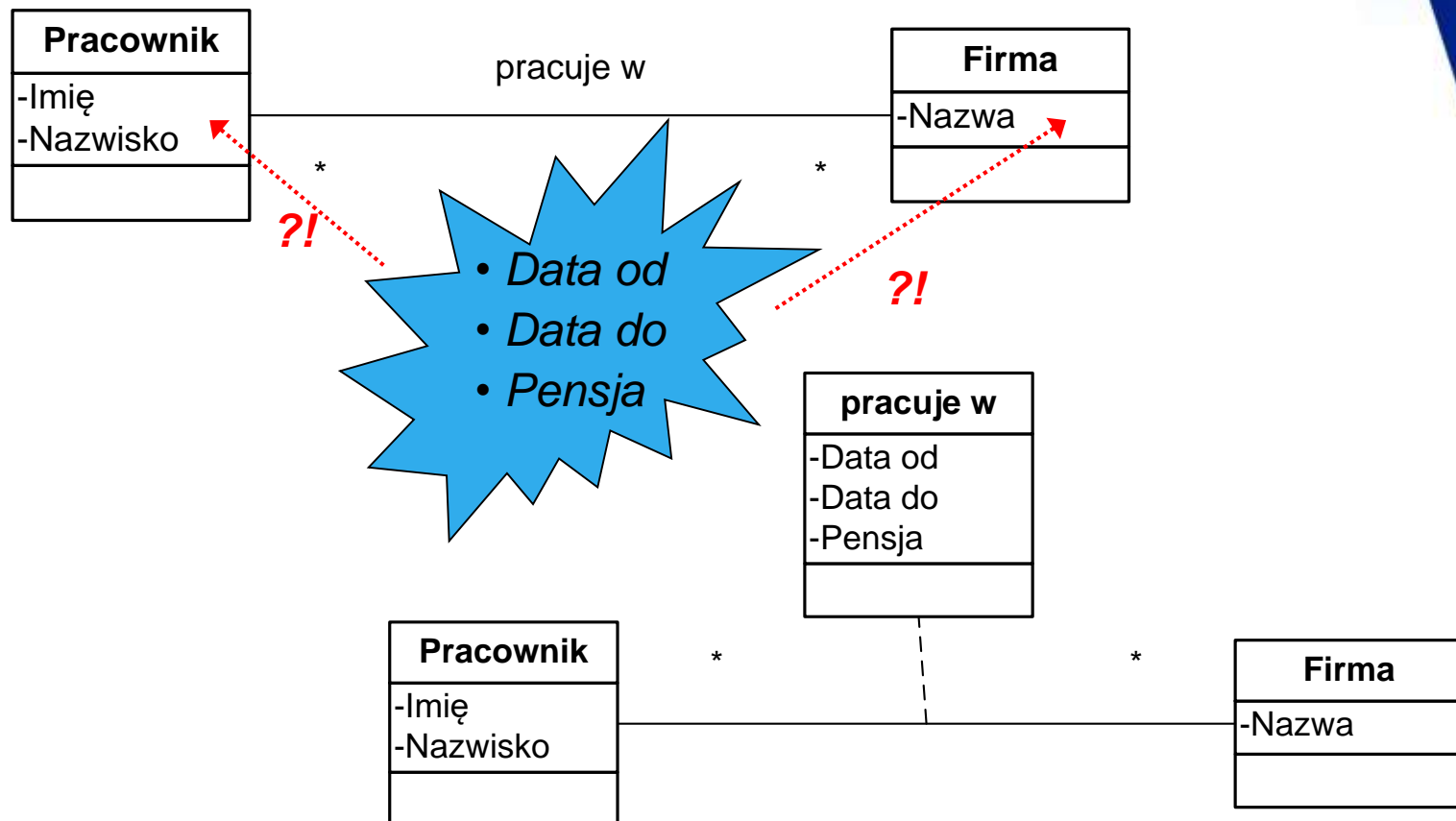
- Załóżmy, że mamy przypadek biznesowy opisany poniższym diagramem



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

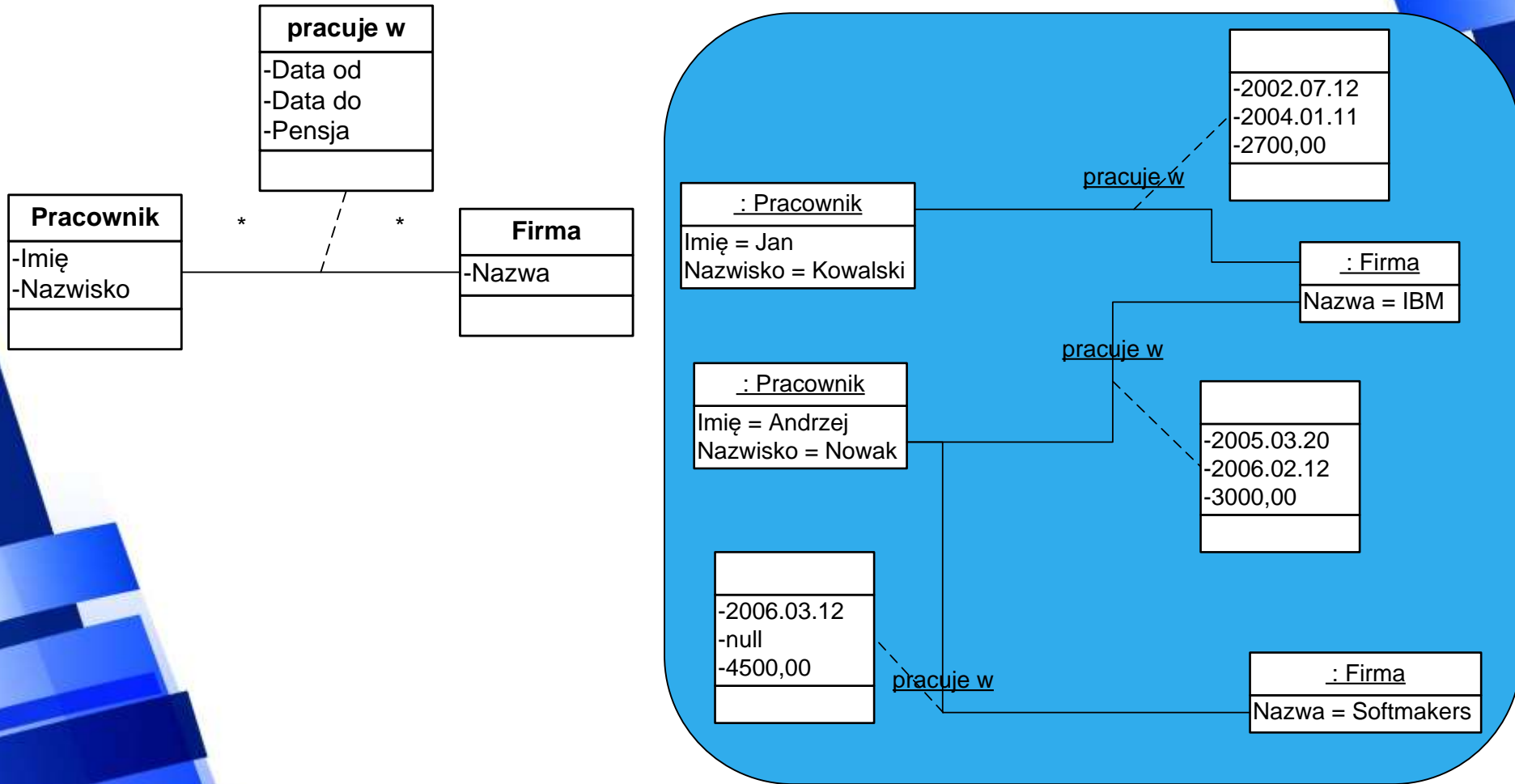
Asocjacja z atrybutem (2)

- Rozwiązanie: asocjacja z atrybutem (klasa asocjacji)



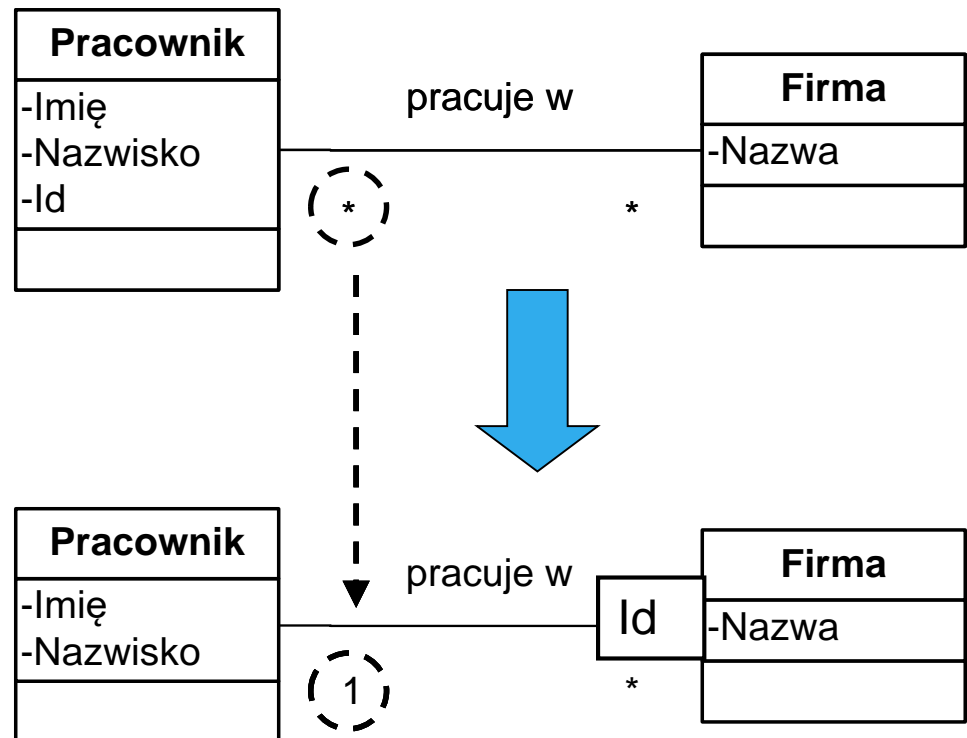
Asocjacja z atrybutem (3)

- Diagram klas oraz diagram obiektów



Asocjacja kwalifikowana

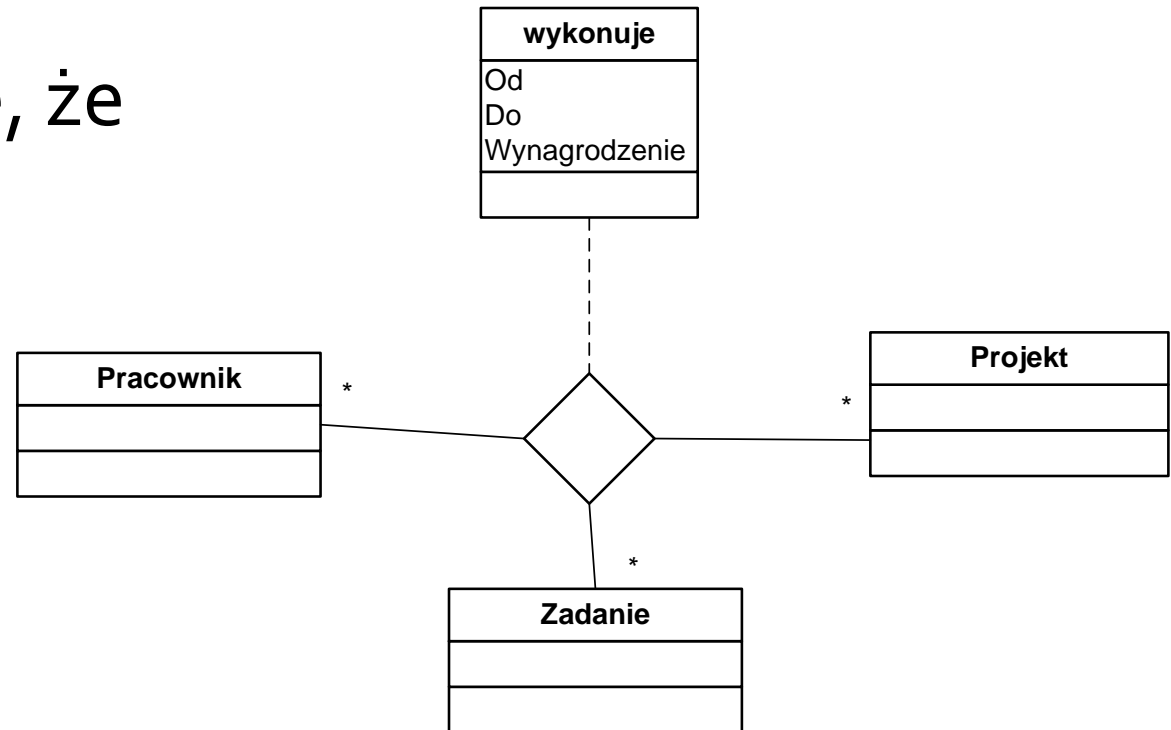
- Kwalifikator: atrybut lub zestaw atrybutów jednoznacznie identyfikujący obiekt w ramach asocjacji.
- Umożliwia szybkie otrzymanie obiektu z drugiej strony powiązania na podstawie kwalifikatora
- Indeksowanie (?)



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

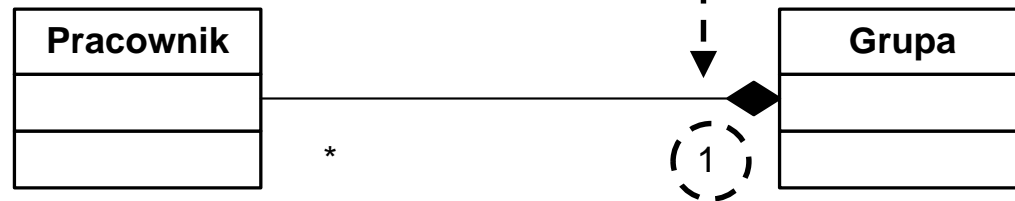
- Kompozycja jest mocniejszą formą agregacji.
- Czyli jest również asocjacją, a zatem posiada wszystkie jej cechy.
- Inaczej niż w przypadku agregacji, wykorzystanie kompozycji niesie pewne konsekwencje:
 - Część nie może być współdzielona (liczności),
 - Część nie może istnieć bez całości (całość może),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

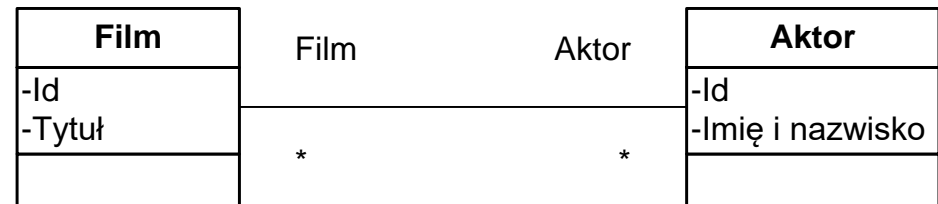
Implementacja asocjacji

- 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 licznosci).
- Konieczność tworzenia

par identyfikatorów (dla
asocjacji dwukierunkowych)



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

:Film
Id = 2
Tytuł = „T3”
Aktor = [3, 5]

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

:Aktor
Id = 4
Imię i Nazwisko = „MB”
Film = [1]

:Aktor
Id = 5
Imię i Nazwisko = „KL”
Film = [2]

Wykorzystanie identyfikatorów (2)

```
public class Actor {
    private int id;
    public String name;    // public for simplicity

    public int[] movieIds;

    private static List<Actor> extent = new ArrayList<>();

    public Actor(int id, String name, int[] movieIds) {
        // Add to the extent
        extent.add(this);

        this.id = id;
        this.name = name;
        this.movieIds = movieIds;
    }

    public static Actor findActor(int id) throws Exception {
        for(Actor actor : extent) {
            if(actor.id == id) {
                return actor;
            }
        }

        throw new Exception("Unable to find an actor with the id = " + id);
    }
}
```

Wykorzystanie identyfikatorów (3)

```
public class Movie {
    public int id;
    public String title; // public for simplicity

    public int[] actorIds;

    private static ArrayList<Movie> extent = new ArrayList<Movie>();

    public Movie(int id, String title, int[] actorIds) {
        // Add to the extent
        extent.add(this);

        this.id = id;
        this.title = title;
        this.actorIds = actorIds;
    }

    public static Movie findMovie(int id) throws Exception {
        for(Movie movie : extent) {
            if(movie.id == id) {
                return movie;
            }
        }

        throw new Exception("Unable to find a movie with the id = " + id);
    }
}
```

Wykorzystanie identyfikatorów (4)

```
public static void testIdAssociations() throws Exception {
    var movie1 = new mt.mas.associationsid.Movie(1, "T1", new int[]{3, 4}); // The 'var' requires
Java 10+
    var movie2 = new mt.mas.associationsid.Movie(2, "T3", new int[]{3});

    var actor1 = new mt.mas.associationsid.Actor(3, "AS", new int[]{1, 2});
    var actor2 = new mt.mas.associationsid.Actor(4, "MB", new int[]{1});
    var actor3 = new mt.mas.associationsid.Actor(5, "KL", new int[]{2});

    // Show information about the movie1
    System.out.println(movie1.title);
    for(int i = 0; i < movie1.actorIds.length; i++) {
        System.out.println("    " + mt.mas.associationsid.Actor.findActor(movie1.actorIds[i]).name);
    }

    // Show information about the actor1
    System.out.println(actor1.name);
    for(int i = 0; i < actor1.movieIds.length; i++) {
        System.out.println("    " + mt.mas.associationsid.Movie.findMovie(actor1.movieIds[i]).title);
    }
}
```

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 Movie {
    private static Map<Integer, Movie> extent = new TreeMap<>();

    // [...]

    public Movie(int id, String title, int[] actorIds) {
        // Add to the extent
        extent.put(id, this);

        this.id = id;
        this.title = title;
        this.actorIds = actorIds;
    }

    public static Movie findMovie(int id) throws Exception {
        return extent.get(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 List<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 List<Film> film; // impl. asocjacji, licznosc *

    public Aktor(String imieNazwisko) {
        this.imieNazwisko = imieNazwisko;
    }
}

public class Film {
    public String tytul; // public dla uproszczenia
    public List<Aktor> aktor; // impl. asocjacji, licznosc *

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

Ulepszone zarządzanie powiązaniem

- 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 powiązaniem

(2)

```
public class Actor {
    public String name;    // public for simplicity

    private List<Movie> movies = new ArrayList<>(); // implementation of the association, cardinality *

    public Actor(String name) {
        this.name = name;
    }

    public void addMovie(Movie newMovie) {
        // Check if we already have the info
        if(!movies.contains(newMovie)) {
            movies.add(newMovie);

            // Add the reverse connection
            newMovie.addActor(this);
        }
    }

    @Override
    public String toString() {
        var info = "Actor: " + name + "\n";

        // Add info about his/her movies
        for(Movie movie : movies) {
            info += "    " + movie.title + "\n";
        }

        return info;
    }
}
```

Ulepszone zarządzanie powiązaniem

(3)

```
public class Movie {
    public String title; // public for simplicity

    private List<Actor> actors = new ArrayList<>(); // implementation of the association, cardinality: *

    public Movie(String title) {
        this.title = title;
    }

    public void addActor(Actor newActor) {
        // Check if we have the information already
        if(!actors.contains(newActor)) {
            actors.add(newActor);

            // Add the reverse connection
            newActor.addMovie(this);
        }
    }

    @Override
    public String toString() {
        var info = "Movie: " + title + "\n";

        // Add info about titles of his/her movies
        for(Actor a : actors) {
            info += "    " + a.name + "\n";
        }

        return info;
    }
}
```

Ulepszone zarządzanie powiązaniem

(4)

- Uwaga na licznosc 1 - *:
 - Tam gdzie jest licznosc 1 bedzie setter, np. setFilm (zamiast addFilm).
 - W efekcie zanim wstawimy nowa wartosc, musimy sprawdzic czy aktualna jest rozna od null.
 - Jezeli tak, to trzeba usunac istniejace powiazanie z obu stron.
 - Nastepnie mozemy utworzyc nowe poprzez przypisanie (licznosc 1) i dodanie (licznosc *).

Ulepszone zarządzanie powiązaniem

(4)

```
public static void testRefAssociations() throws Exception {  
    // Create new business objects (without connections)  
    var movie1 = new mt.mas.asocjacjeref.Movie("T1");  
    var movie2 = new mt.mas.asocjacjeref.Movie("T3");  
  
    var actor1 = new mt.mas.asocjacjeref.Actor("AS");  
    var actor2 = new mt.mas.asocjacjeref.Actor("MB");  
    var actor3 = new mt.mas.asocjacjeref.Actor("KL");  
  
    // Add info about connections  
    movie1.addActor(actor1);  
    movie1.addActor(actor2);  
    movie2.addActor(actor1);  
    movie2.addActor(actor3);  
  
    // Show info about movies  
    System.out.println(movie1);  
    System.out.println(movie2);  
  
    // Show info about actors  
    System.out.println(actor1);  
    System.out.println(actor2);  
    System.out.println(actor3);  
}
```

Movie: T1

AS

MB

Movie: T3

AS

KL

Actor: AS

T1

T3

Actor: MB

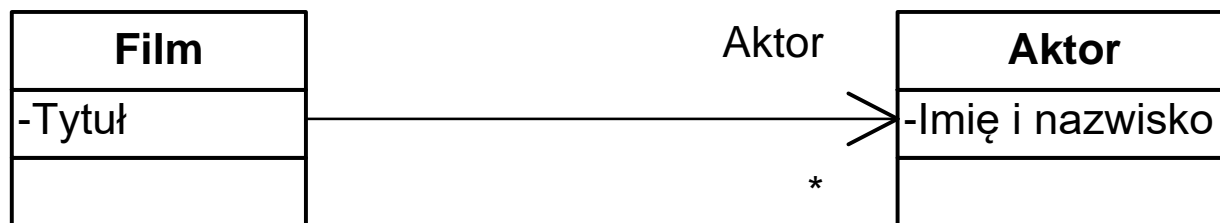
T1

Actor: KL

T3

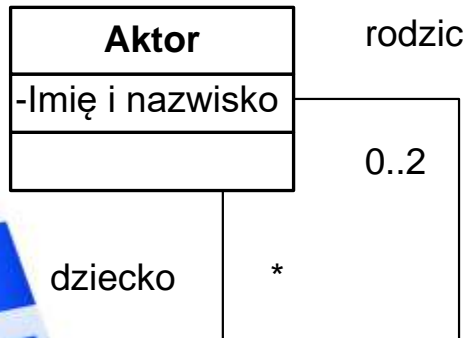
Implementacja asocjacji skierowanej

- Poniższy 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.



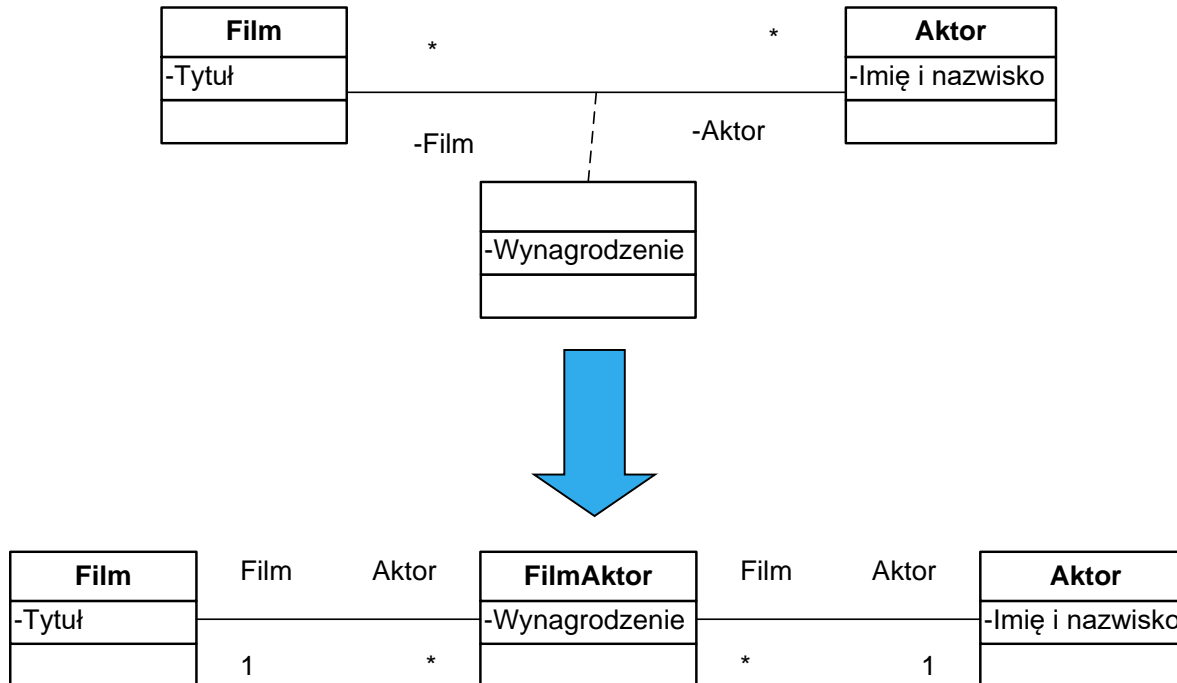
```
public class Actor {
    // [...]

    private ArrayList<Actor> parents = new ArrayList<Actor>();
    private ArrayList<Actor> children = new ArrayList<Actor>();

    // [...]
}
```

Implementacja asocjacji z atrybutem

- Najpierw musimy zamienić:
 - jedną konstrukcję UML (asocjacja z atrybutem)
 - na inną konstrukcję UML (asocjacja 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)
- Warto utworzyć specjalny konstruktor dla 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 `List`,
 - Stosujemy kontener mapujący, gdzie kluczem jest *tytuł*, a wartością obiekt opisujący *film*.
 - Informacja zwrotna w dotychczasowy sposób.



Implementacja asocjacji kwalifikowanej (2)

```
public class Actor {  
  
    // [...]  
  
    private Map<String, Movie> moviesQualif = new TreeMap<>();  
  
    public void addMovieQualif(Movie newMovie) {  
        // Check if we already have the info  
        if(!moviesQualif.containsKey(newMovie.title)) {  
            moviesQualif.put(newMovie.title, newMovie);  
  
            // Add the reverse connection  
            newMovie.addActor(this);  
        }  
    }  
  
    public Movie findMovieQualif(String title) throws Exception {  
        // Check if we have the info  
        if(!moviesQualif.containsKey(title)) {  
            throw new Exception("Unable to find a movie: " + title);  
        }  
  
        return moviesQualif.get(title);  
    }  
  
    // [...]  
}
```

Implementacja asocjacji kwalifikowanej (3)

```
public static void testQualifiedAssociations() throws Exception {
    // Create new business objects (without connections)
    var movie1 = new mt.mas.associationsref.Movie("T1");
    var movie2 = new mt.mas.associationsref.Movie("T3");

    var actor1 = new mt.mas.associationsref.Actor("AS");
    var actor2 = new mt.mas.associationsref.Actor("MB");
    var actor3 = new mt.mas.associationsref.Actor("KL");

    // Add info about connections
    actor1.addMovieQualif(movie1);
    actor1.addMovieQualif(movie2);
    actor2.addMovieQualif(movie1);
    actor3.addMovieQualif(movie2);

    // Show info about actors
    System.out.println(actor1);
    System.out.println(actor2);
    System.out.println(actor3);

    // Get the info about the "T1" movie for the actor1
    var movie = actor1.findMovieQualif("T1");
    System.out.println(movie);
}
```

Actor: AS

T1

T3

Actor: MB

T1

Actor: KL

T3

Movie: T1

AS

MB

- Ciąg dalszy na następnym wykładzie...

Pliki źródłowe

- Pobierz pliki źródłowe do wszystkich wykładów MAS



<http://www.mtrzaska.com/plik/mas/mas-source-files-lectures>