Modelowanie Systemów Informacyjnych (MSI)

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

Wykład 11

Realizacja pozostałych, wybranych konstrukcji UML w obiektowych językach programowania

http://www.mtrzaska.com

Zagadnienia

- Omówienie różnych rodzajów ograniczeń występujących w UML
- Implementacja ograniczeń w obiektowych językach programowania:
  - Dotyczące atrybutów
  - Subset
  - Ordered
  - Bag
  - History
  - Xor
  - Dowolne
- Podsumowanie

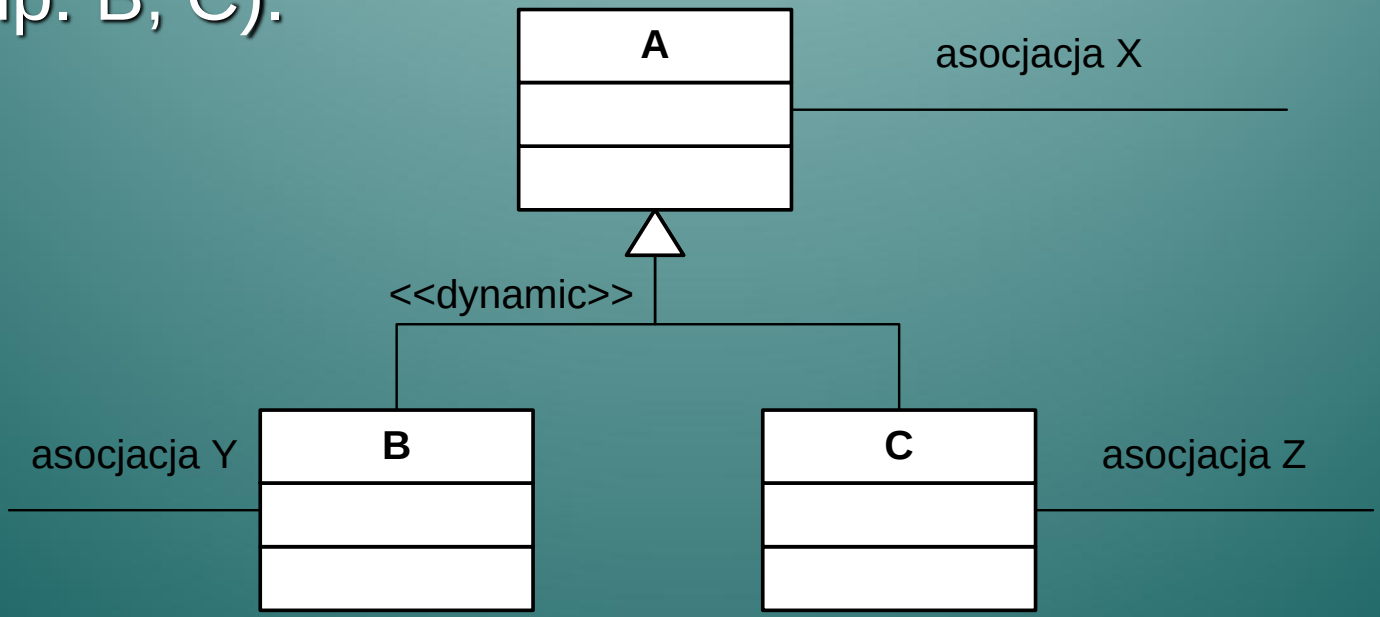
Praca domowa

- Modyfikacja klasy `ObjectPlusPlus` ułatwiająca realizację dziedziczenia dynamicznego.

Prezentacje studentów…

Praca domowa (2)

- Dodatkowe problemy
  - Ze „starego” obiektu powinniśmy skopiować tylko powiązania należące do „nadklasy” (np. A). Nie kopiujemy powiązań należących do jego asocjacji (np. B, C).



Praca domowa (3)

- Dodatkowe problemy – c. d.
  - Aby poprawnie uaktualnić powiązania pokazujące na „nowy” obiekt, potrzebujemy informacje o wzajemnych powiązaniach ról.
    - Która `nazwaRoli` odpowiada `odwrotnaNazwaRoli`,
    - Teoretycznie możemy zastąpić wszystkie wystąpienia (we wszystkich rolach) `obiektDocelowy` pokazujące na „stary” obiekt, ale część z tych uaktualnień jest niepotrzebna, ponieważ część z powiązań „przepada” (patrz poprzedni slajd).

Powrót do aktualnego wykładu

Ograniczenia w UML

- Jeden z mechanizmów rozszerzalności UML.
- Umożliwiają doprecyzowanie modelu.
- Wykorzystanie:
  - OCL
  - Wyrażenia matematyczne (arytmetyka, logika)
  - Tekst
- Istnieje pewna predefiniowana grupa ograniczeń
- Notacja: `{treść ograniczenia}`

Rodzaje ograniczeń

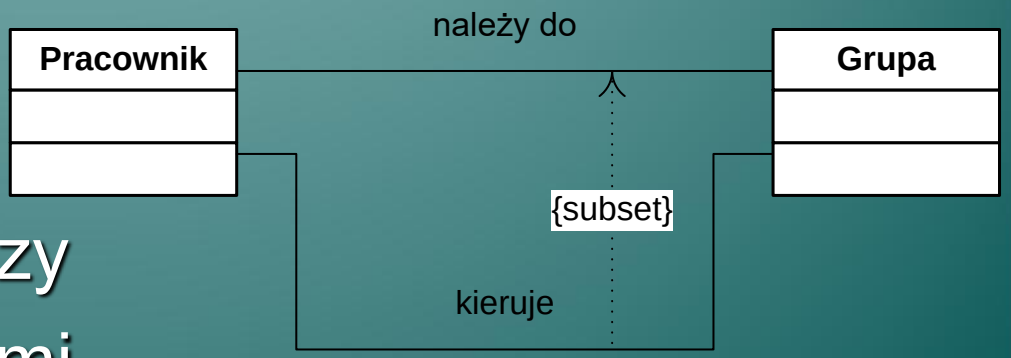
- **Dynamiczne**. Przy sprawdzaniu warunku, istotny jest poprzedni stan elementu, którego ograniczenie dotyczy.
- **Statyczne**. Przy sprawdzaniu warunku, poprzedni stan elementu, którego ograniczenie dotyczy, nie ma znaczenia.

Rodzaje ograniczeń (2)

- Przykłady
  - Ograniczenia dynamiczne
    - Pensja nie może zmaleć,
    - Wzrost pensji nie może być większy niż 10%.
  - Ograniczenia statyczne
    - Pensja > 2000
    - Pensja < 5000

Predefiniowane ograniczenia

- {Subset}
  - Nakładane na dwie asocjacje (agregacje).
  - Aby można było utworzyć powiązanie w ramach asocjacji B („kieruje”), musi już istnieć powiązanie w ramach asocjacji A („należy do”).
  - Obydwie asocjacje powinny być pomiędzy tymi samymi klasami.
  - Obydwa powiązania powinny być pomiędzy tymi samymi obiektami.

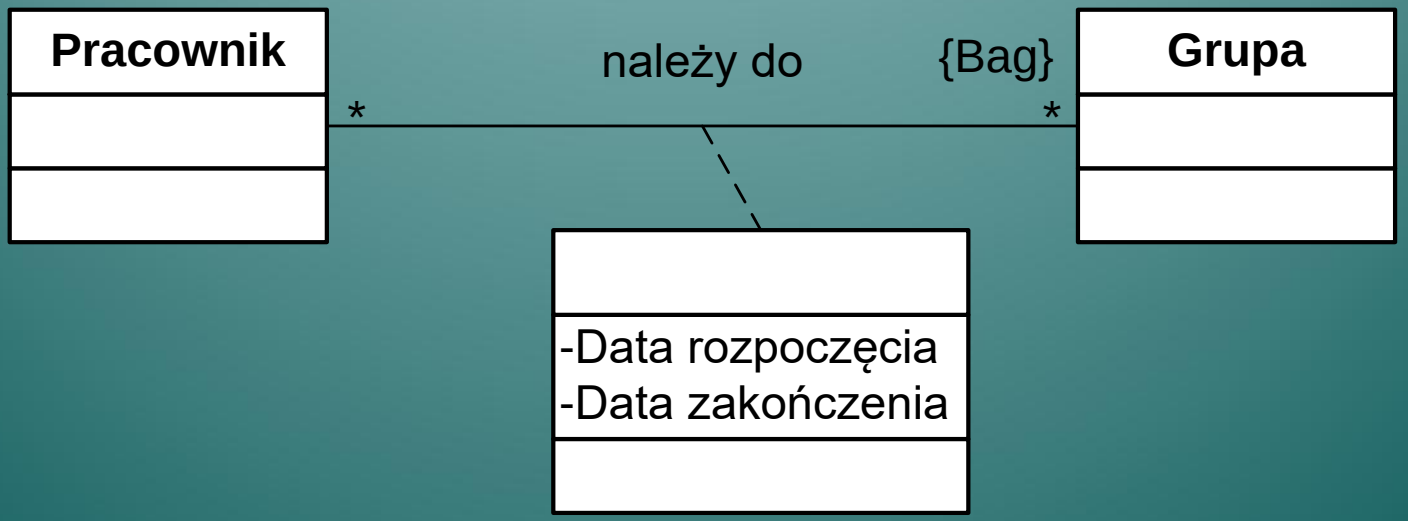


Predefiniowane ograniczenia (2)

- `{Ordered}`
  - Może dotyczyć:
    - **Asocjacji**. Oznacza, że powiązania są przechowywane (oraz otrzymywane i przetwarzane) w pewnej ustalonej kolejności.
    - **Klasy**. Obiekty w ekstensji są przechowywane (oraz otrzymywane i przetwarzane) w pewnej ustalonej kolejności.

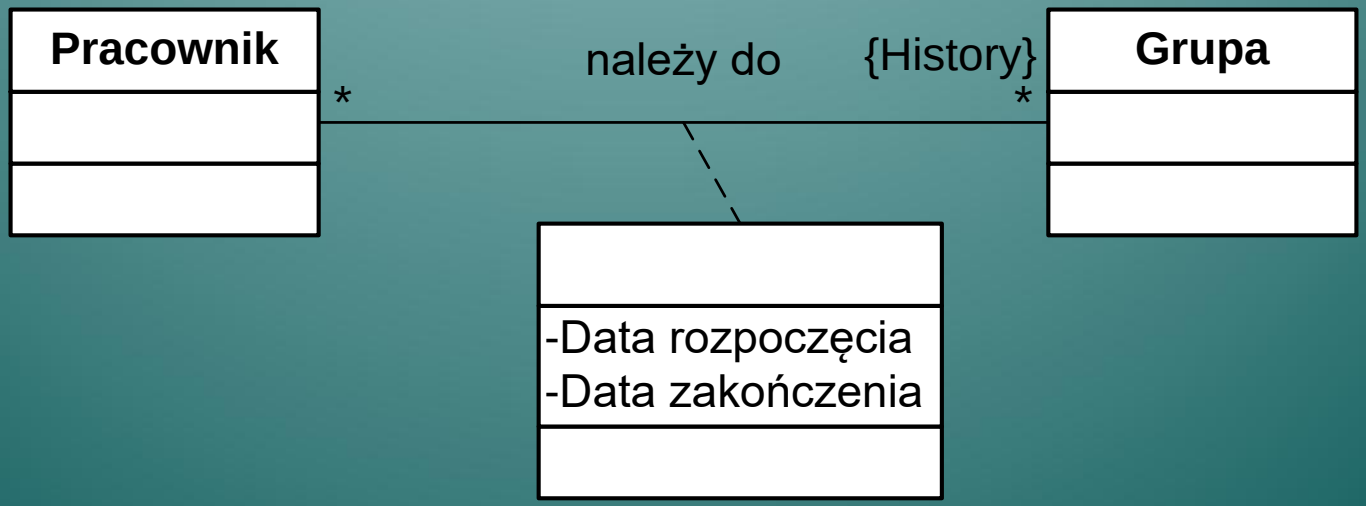
Predefiniowane ograniczenia (3)

- {Bag}
  - Umożliwia przechowywanie duplikatów elementów.
  - W przypadku asocjacji, oznacza, że może istnieć wiele powiązań pomiędzy tymi samymi obiektami.



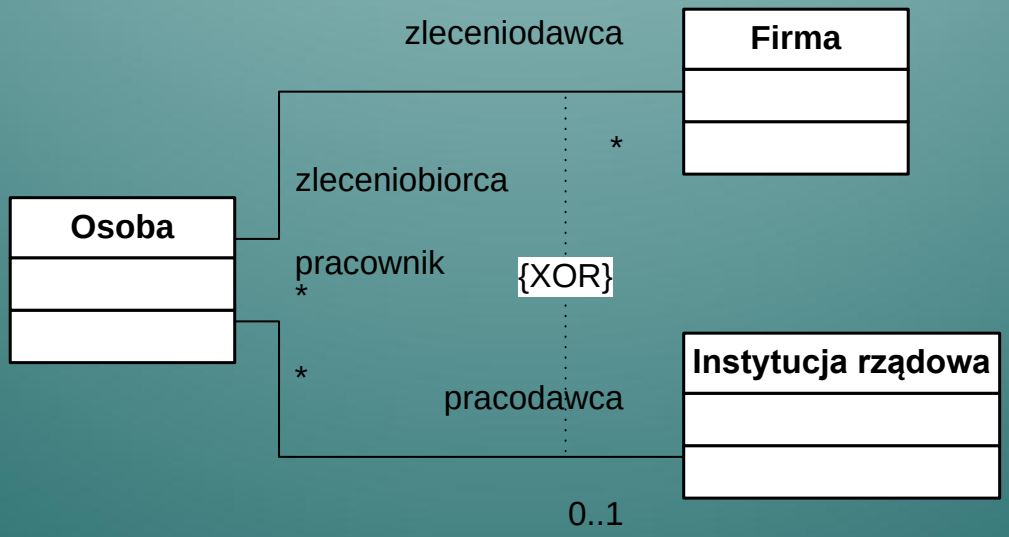
Predefiniowane ograniczenia (4)

- `{History}`
  - Podobny do ograniczenia `{bag}`.
  - Umożliwia utworzenie wielu powiązań pomiędzy tymi samymi obiektami.
  - Podkreśla aspekt czasowy opisywanej asocjacji.



Predefiniowane ograniczenia (5)

- {XOR}
  - Dotyczy co najmniej dwóch asocjacji.
  - Zapewnia, że będzie istniało tylko jedno powiązanie w ramach asocjacji, które ogranicza.



  - Jeżeli osoba pracuje w instytucji rządowej to nie może współpracować z firmą.

Ograniczenia UML, a języki programowania

- W popularnych językach programowania:
  - Java
  - C#
  - C++

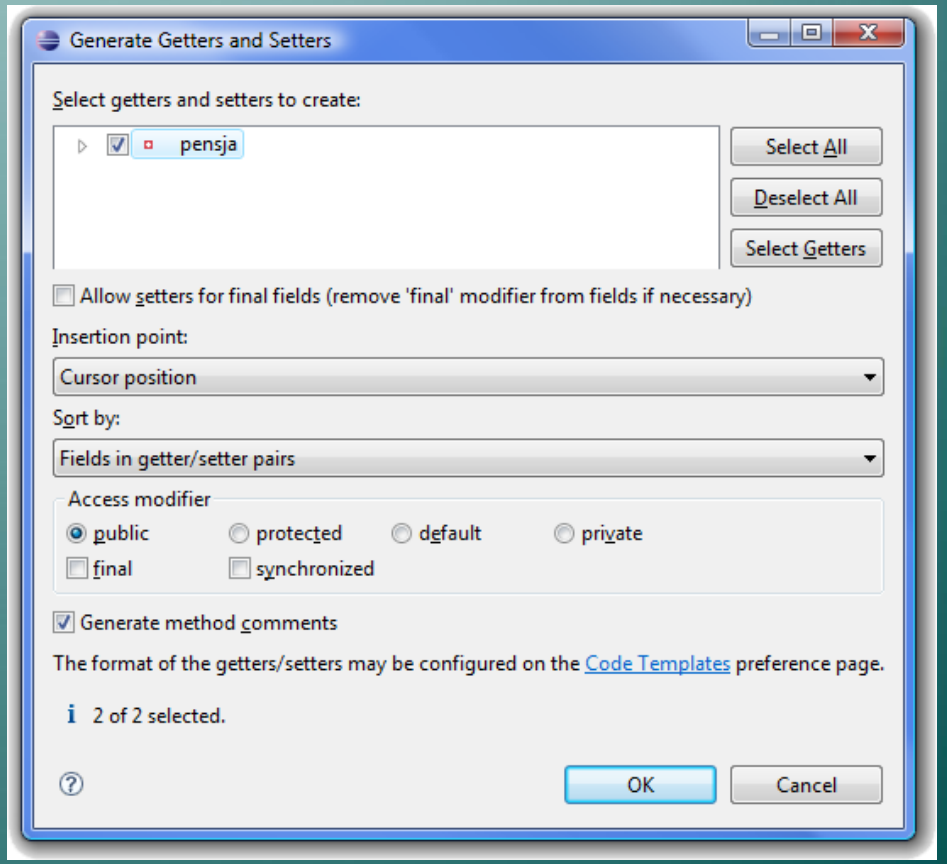
  ograniczenia **nie** występują bezpośrednio.
- Możemy je uwzględnić:
  - ręcznie implementując odpowiednie metody.
  - korzystając z dedykowanej biblioteki (np. dla OCL).
- Stopień skomplikowania takich implementacji zależy od rodzaju ograniczeń.

Implementacja ograniczeń dotyczących atrybutów

- Podstawowe założenia
  - Atrybuty w klasie są ukryte (najlepiej jako `private`),
  - Wszelka aktywność związana z atrybutami odbywa się przez dedykowane metody
    - get…
    - set…
  - Powyższa reguła obowiązuje również wewnątrz klasy w której są atrybuty (chociaż tego nie da się kontrolować).

Implementacja ograniczeń dotyczących atrybutów (2)

- Współczesne środowiska programistyczne (IDE) wspierają proces tworzenia metod służących do operacji na „ukrytych” atrybutach.
  - W Eclipse jest do tego dedykowana opcja: *Source ➔ Generate Getters and Setters*.



Implementacja ograniczeń dotyczących atrybutów (3)

- Pensja nie może zmaleć,

```
public class Pracownik {
	private float pensja;

	public void setPensja(float pensja) throws Exception {
		// Sprawdzenie ograniczenia
		if(pensja < this.pensja) {
			throw new Exception("Pensja nie moze zmalec!");
		}

		this.pensja = pensja;
	}
}
```

- Wzrost pensji nie może być większy niż 10%.

```
public void setPensja(float pensja) throws Exception {
	// Sprawdzenie ograniczenia
	if(this.pensja * wspPensji < pensja) {
		throw new Exception("Wzrost pensji nie moze byc wiekszy niz " + wspPensji);
	}

	this.pensja = pensja;
}
```

Implementacja ograniczeń dotyczących atrybutów (4)

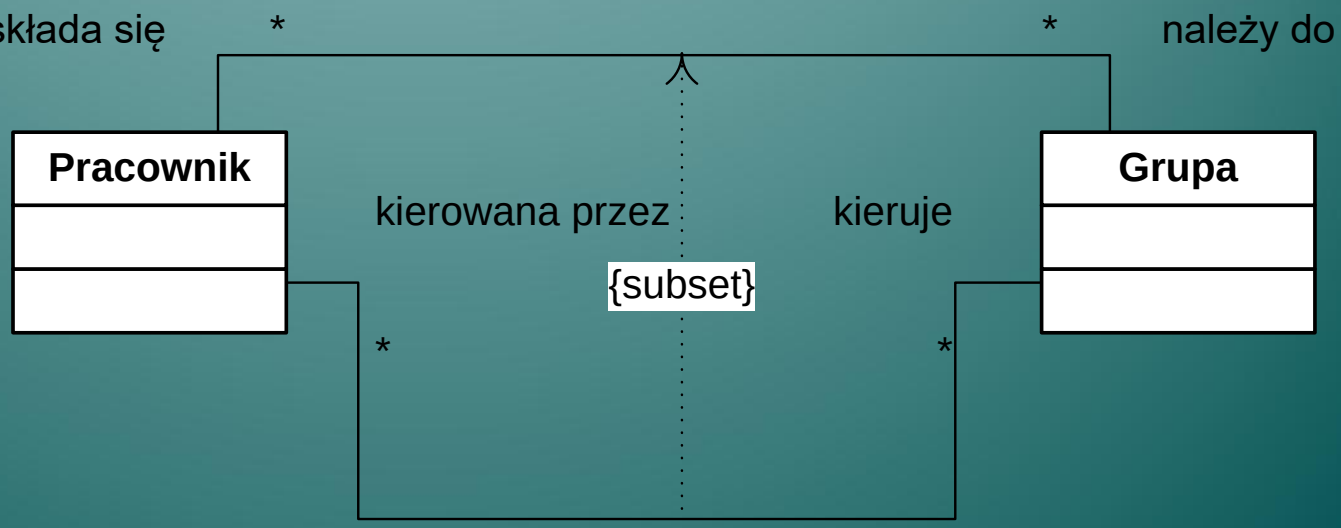
- Pensja > 2000

```
public class Pracownik {
     private float pensja;

     public void setPensja(float pensja) throws Exception {
          // Sprawdzenie ograniczenia
          if(pensja < minimalnaPensja) {
               throw new Exception("Pensja musi wynosic co najmniej " +
                    minimalnaPensja);
          }

          this.pensja = pensja;
     }

     final static float minimalnaPensja = 2000;
     final static float maksymalnaPensja = 5000;
}
```

- Pensja < 5000
  - Analogicznie jak powyżej.

Implementacja ograniczenia {subset}

- Powiązania w ramach asocjacji opisanej przez role „*składa się*”, „*należy do*” dodajemy „normalnie”.
- Przed dodaniem powiązania w ramach asocjacji opisanej przez role „*kierowana przez*”, „*kieruje*”, sprawdzamy czy istnieje powiązanie do dodawanego obiektu w ramach „nadrzędnej” asocjacji.



Implementacja ograniczenia {subset} (2)

- Zmodyfikujemy klasę `ObjectPlusPlus`
  - Dodamy metodę, która sprawdzi czy istnieje powiązanie do podanego obiektu w ramach danej roli.
- Utworzymy nową klasę `ObjectPlus4`
  - dziedziczącą z `ObjectPlusPlus`
  - enkapsulującą funkcjonalność związaną z realizacją ograniczeń.

Implementacja ograniczenia {subset} (3)

- Metoda w ObjectPlusPlus, która sprawdza czy istnieje powiązanie do podanego obiektu w ramach danej roli.

```
public boolean czyIstniejePowiazanie(String nazwaRoli, ObjectPlusPlus obiektDocelowy)
          throws Exception {
     HashMap<Object, ObjectPlusPlus> powiazaniaObiektu;

     if(!powiazania.containsKey(nazwaRoli)) {
          // Brak powiazan dla tej roli
          throw new Exception("Brak powiazan dla roli: " + nazwaRoli);
     }
     powiazaniaObiektu = powiazania.get(nazwaRoli);
     if(powiazaniaObiektu.containsValue(obiektDocelowy)) {
          // Obiekt docelowy istnieje
          return true;
     }
     // brak obiektu
     return false;
}
```

Implementacja ograniczenia {subset} (4)

- Metoda w ObjectPlus4, która dodaje nowe powiązanie uwzględniając ograniczenie {subset}

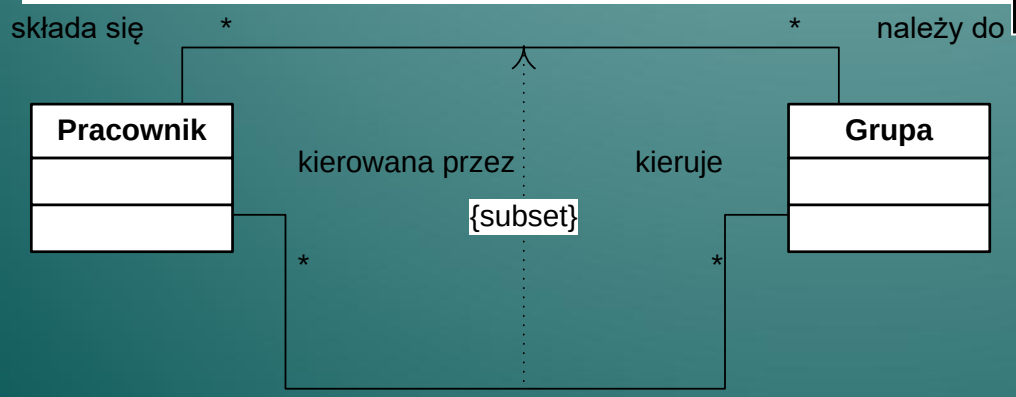
```
public class ObjectPlus4 extends ObjectPlusPlus {
     public ObjectPlus4() {
          super();
     }
     public void dodajPowiazanie_subset(String nazwaRoli, String odwrotnaNazwaRoli,
          String nazwaRolinadrzednej, ObjectPlusPlus obiektDocelowy) throws Exception {
          if(czyIstniejePowiazanie(nazwaRoliNadrzednej, obiektDocelowy)) {
               // Istnieje powiazanie do dodawanego obiektu w ramach roli nadrzednej
               // Mozemy utworzyc nowe powiazanie
               dodajPowiazanie(nazwaRoli, odwrotnaNazwaRoli, obiektDocelowy);
          }
          else {
               // Brak powiazania do dodawanego obiektu w ramach roli nadrzednej ==>
               wyjatek
               throw new Exception("Brak powiazania do dodawanego obiektu '"
                    + obiektDocelowy + "'w ramach roli nadrzednej '" +
                    nazwaRoliNadrzednej + "'!");
          }
     }
}
```

Implementacja ograniczenia {subset} (5)

- Przykład użycia – wersja automatyczna

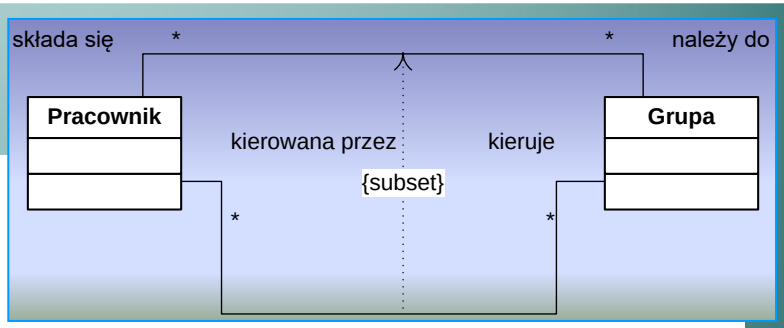
```
Pracownik p = new Pracownik("Jan", "Kowalski");
Grupa g = new Grupa("Grupa nr 1");
try {
      // Dodaj "zwykle" powiazanie
      p.dodajPowiazanie(Pracownik.rolaNalezyDo, Grupa.rolaSkladaSie, g);

      p.dodajPowiazanie_subset(Pracownik.rolaKieruje, Grupa.rolaKierowanaPrzez,
      Pracownik.rolaNalezyDo, g);

      // Wyswietl informacje o wszystkich powiazaniach tego obiektu
      p.wyswietlPowiazania(System.out);
} catch (Exception e) {
      e.printStackTrace();
}
```

```
Pracownik powiazania w roli kieruje:
   Grupa: Grupa nr 1
Pracownik powiazania w roli nalezy do:
   Grupa: Grupa nr 1
```



Implementacja ograniczenia {subset} (5)

- Przykład użycia - ręczne




```
Pracownik p = new Pracownik("Jan", "Kowalski");
Grupa g = new Grupa("Grupa nr 1");

try {
     // Dodaj "zwykle" powiazanie
     p.dodajPowiazanie(Pracownik.rolaNalezyDo, Grupa.rolaSkladaSie, g);

     // Sprawdz czy istnieje "nadrzedne" powiazanie
     if(p.czyIstniejePowiazanie(Pracownik.rolaNalezyDo, g)) {
          // Powiazanie nadrzedne istnieje => mozna dodac nowe powiazanie
          p.dodajPowiazanie(Pracownik.rolaKieruje, Grupa.rolaKierowanaPrzez, g);
     }
     else {
          // Brak powiazania nadrzednego
          System.out.println("Brak powiazania nadrzednego w roli: " +
                    Pracownik.rolaNalezyDo);
     }

     // Wyswietl informacje o wszystkich powiazaniach tego obiektu
     p.wyswietlPowiazania(System.out);
} catch (Exception e) {
     // Blad
     e.printStackTrace();
}
```

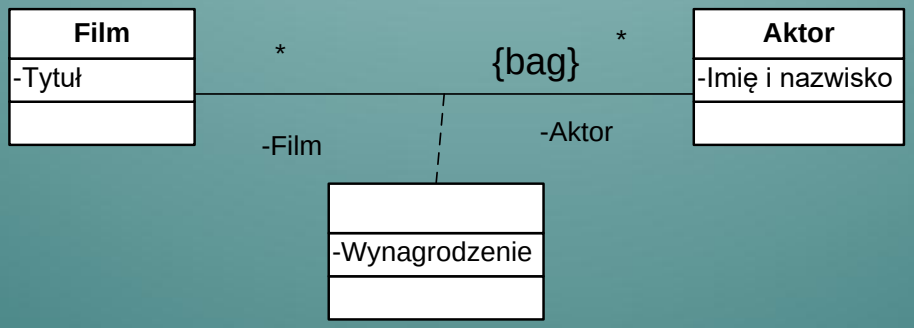
```
Pracownik powiazania w roli kieruje:
  Grupa: Grupa nr 1
Pracownik powiazania w roli nalezy do:
  Grupa: Grupa nr 1
```

Implementacja ograniczenia `{ordered}`

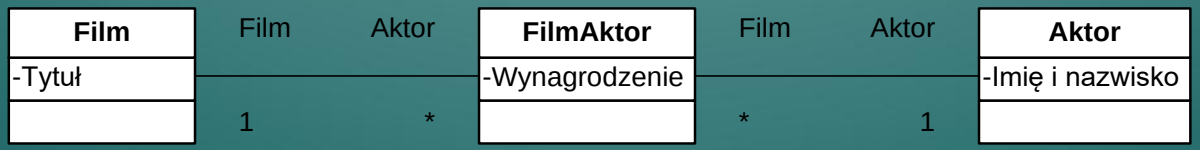
- Dla asocjacji
  - Musimy użyć kontenera gwarantującego kolejność przechowywanych elementów. W przypadku:
    - kolejności dodawania - np. `List`
    - Indywidualnie zdefiniowanej kolejności, np. `TreeSet` razem z odpowiednim komparatorem (`Comparator`).
- Dla ekstensji
  - Analogicznie jak dla asocjacji.
- Klasa `ObjectPlusPlus` domyślnie używa klasy `Vector` dla ekstensji (kolejność gwarantowana) oraz `HashMap` dla powiązań.

Implementacja ograniczenia {bag}

- W przypadku asocjacji, musimy użyć kontenera umożliwiającego przechowywanie duplikatów elementów.
- Duplikaty chcemy przechowywać najczęściej gdy korzystamy z atrybutu asocjacji podającego dodatkowe informacje o „duplikacie”.



- A w takiej sytuacji i tak wprowadzamy klasę pośredniczącą, więc problem „duplikatów” znika.



Implementacja ograniczenia `{bag}` (2)

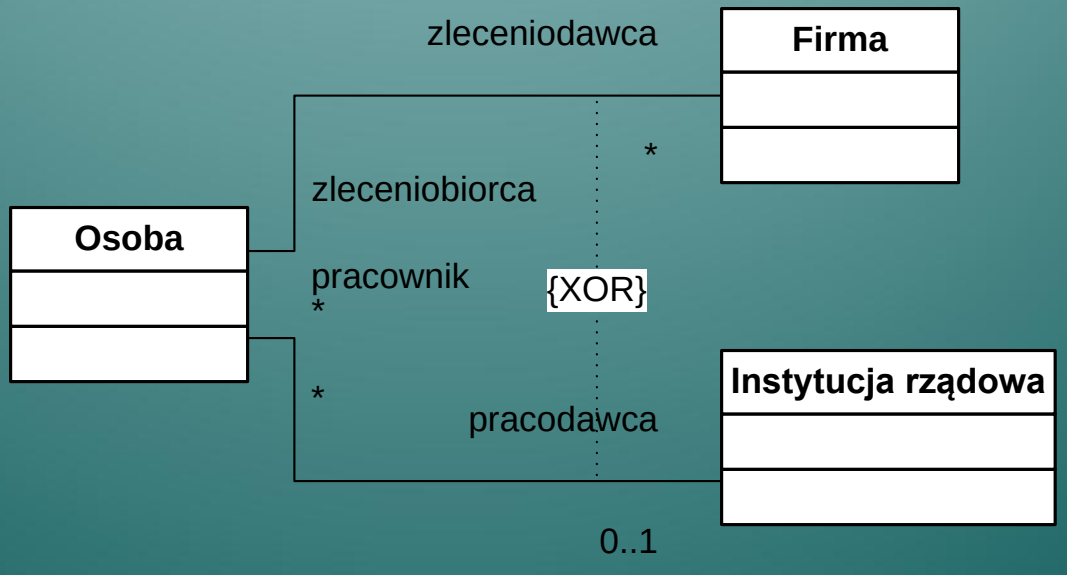
- Gdyby jednak ktoś chciał trzymać duplikaty to trzeba wybrać kontener, który na to pozwala, np. Vector.
- Klasa `ObjectPlusPlus` domyślnie używa klasy `HashMap` dla powiązań, która pilnuje unikalności kluczy. Można dodawać duplikowane wartości, ale z unikalnymi kluczami.

Implementacja ograniczenia {history}

- Czy użycie ograniczenia {history} oznacza jakieś szczególne konsekwencje (w porównaniu z {bag})?
- Wydaje się, że nie.
- Wniosek: Implementacja ograniczenia {history} jest analogiczna do implementacji ograniczenia {bag}.

Implementacja ograniczenia {XOR}

- Musimy zadbać aby w zdefiniowanej grupie asocjacji (ról) występowało tylko jedno powiązanie.
- Wykorzystamy odpowiednio zmodyfikowaną klasę `ObjectPlus4`.



Implementacja ograniczenia {XOR} (2)

```
public class ObjectPlus4 extends ObjectPlusPlus {
     private List<String> roleXOR = new LinkedList<String>();

     // [...]

     public void dodajRoleXOR(String nazwaRoliXor) {
          roleXOR.add(nazwaRoliXor);
     }

     public void dodajPowiazanie_xor(String nazwaRoli, String odwrotnaNazwaRoli,
               ObjectPlusPlus obiektDocelowy) throws Exception {
          if(roleXOR.contains(nazwaRoli)) {
               // Aktualnie dodawana rola jest objeta ograniczeniem XOR

               // Sprawdz czy jest juz jakies powiazanie w ramach rol objetych ograniczeniem
               if(czyIstniejePowiazanie()) {
                    throw new Exception("Istnieja juz powiazanie w ramach rol objetych
                              ograniczeniem {XOR}!");
               }

          // Nie ma powiazan, wiec ponizej dodajemy powiazanie korzystajac z "normalnej"
          metody z nadklasy
          }

          // Dodaj powiazanie
          super.dodajPowiazanie(nazwaRoli, odwrotnaNazwaRoli, obiektDocelowy);
     }
     // [...]
```

Implementacja ograniczenia {XOR} (3)

```
public class ObjectPlus4 extends ObjectPlusPlus {

     // [...]

     private boolean czyIstniejePowiazanie() {
          for(String rola : roleXOR) {
               if(this.czySaPowiazania(rola)) {
                    return true;
               }
          }

          return false;
     }
}
```

- Ewentualnie do zrobienia:
  - Uwzględnienie ograniczenia {xor} w przypadku dodawania powiązań z „drugiej strony”

Implementacja dowolnych ograniczeń

- W przypadku dowolnych, innych ograniczeń, np. pisanych „zwykłym tekstem”
  - Niestety trzeba zrozumieć „co autor miał na myśli”,
  - Zaimplementować to wykorzystując odpowiednie metody.
  - Czasami, może być wymagana zmiana podejścia do tworzenia programu, np. zastosowanie ortodoksyjnej hermetyzacji.
  - Brak gotowych rozwiązań.

Podsumowanie

- W popularnych językach programowania ograniczenia nie występują bezpośrednio.
- Pewne ich rodzaje można implementować korzystając z gotowych rozwiązań, np.
  - Dotyczące atrybutów,
  - Predefiniowane: `{subset}`, `{xor}`.
- W ogólnym przypadku (ograniczenia pisane zwykłym tekstem) konieczna jest ich ręczna implementacja.