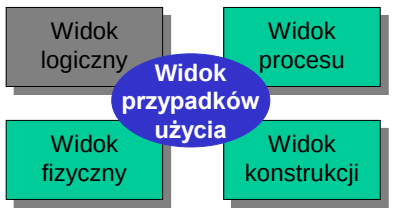
Projektowanie oprogramowania

Wykład 3

1

1

Widok logiczny

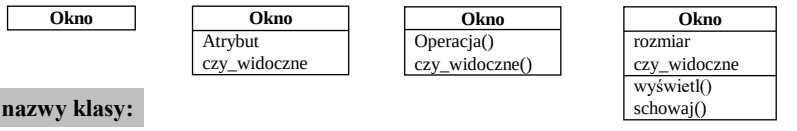


- Używany do modelowania części systemu oraz sposobów, w jaki one ze sobą współdziałają.
- Ten widok zazwyczaj tworzą diagramy:
  - Klas,
  - Obiektów,
  - Maszyny stanowej,
  - Interakcji.

2

2

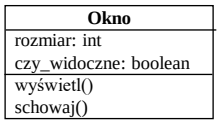
Podstawowe sposoby przedstawiania klas, różne poziomy szczegółowości



Pole nazwy klasy:

nazwa_ klasy

Pole atrybutów:



dostępność nazwa_atrybutu : typ = wart_początkowa

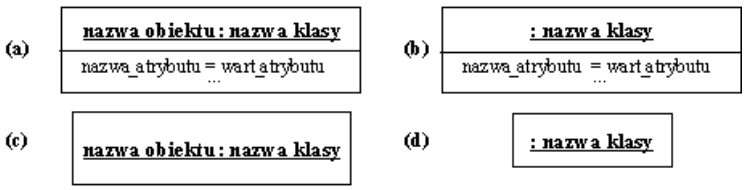
Pole metod:

dostępność nazwa_metody (lista_arg) : typ_wart_zwracanej

3

3

Wystąpienie klasy - obiekty



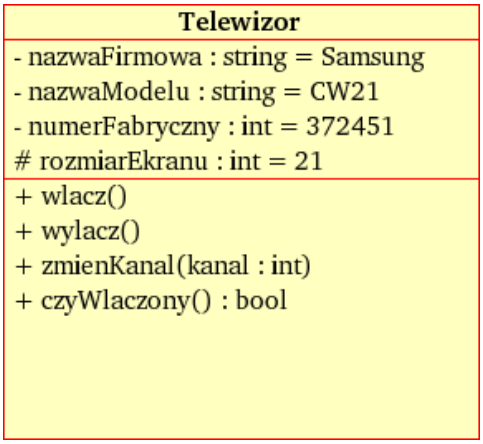
4

4

Poziomy dostępu

Dostepności:

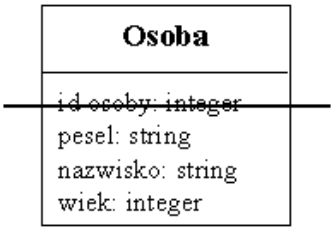
+ publiczna
- prywatna
chroniona
~ zakres pakietu



5

5

Klucz

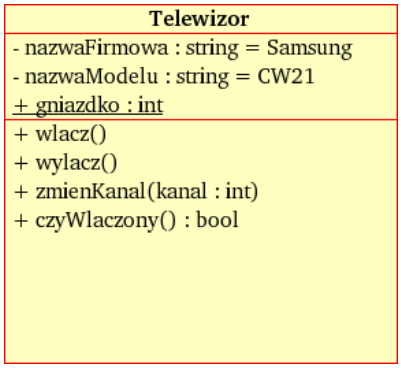


- Ponieważ jedną z podstawowych własności obiektu jest jego tożsamość, nie zachodzi potrzeba definiowania na etapie analizy *klucza*, czyli specjalnego atrybutu (atrybutów) unikalnie identyfikującego (identyfikujących) obiekt.

6

6

Statyczne części klasy

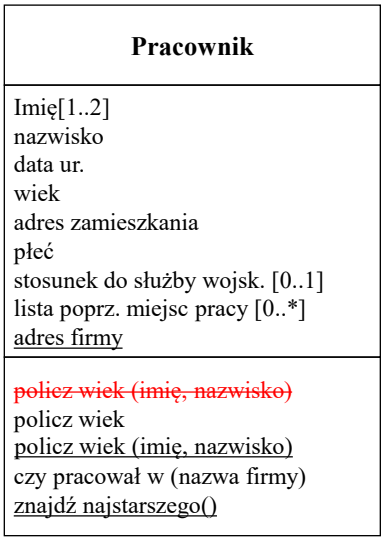


Podkreślenie!

7

7

Liczebność



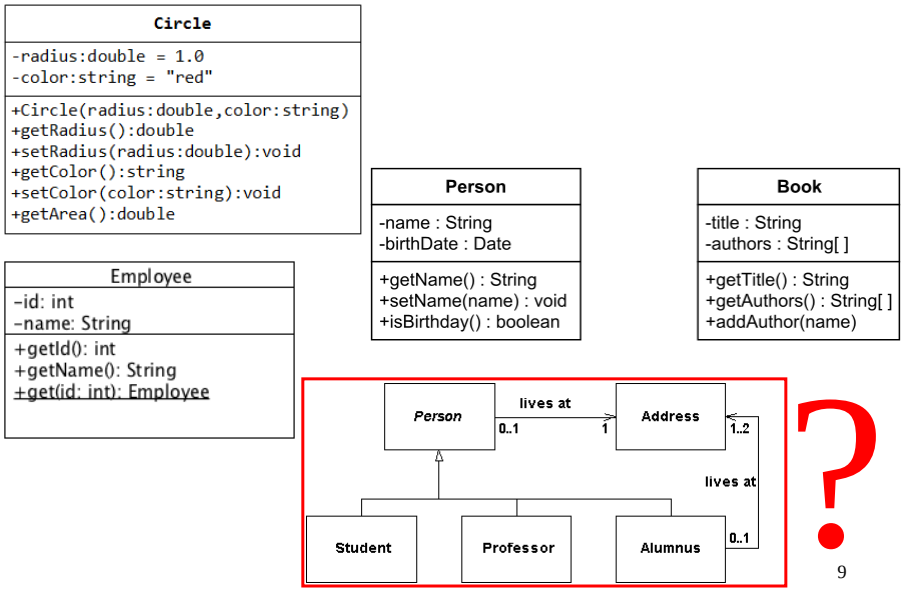
Pozwala określić, że atrybut w rzeczywistości reprezentuje zbiór obiektów.

Dlaczego pierwsza metoda jest przekreślona (niepoprawna)?

8

8

Przykłady klas

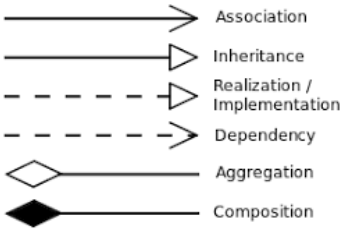


9

9

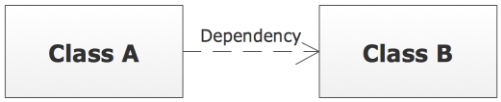
Związki pomiędzy klasami

- Wymieniane od najsłabszych:
  - Zależność (strzałka przerywana)
  - Asocjacja (pojedyncza linia)
  - Agregacja częściowa (pusta strzałka zakończona rombem)
  - Agregacja całkowita (pełna strzałka zakończona rombem)
  - Dziedziczenie (pusta strzałka)



10

Zależność (1)



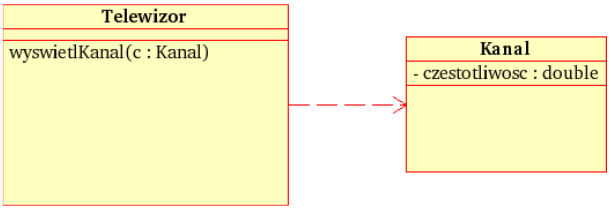
- **Zależność** pomiędzy dwiema klasami informuje, że jedna z nich, aby używać obiektów innej, musi mieć o niej informacje.
- Zależność występuje gdy zmiana specyfikacji jednej klasy, może powodować konieczność wprowadzenia zmiany w innej klasie.

11

11

Zależność (2)

- Najczęściej używa się zależności do pokazania, że jedna klasa używa innej klasy jako parametru jakiejś operacji:



- Obie klasy są zależne od siebie nawzajem w celu zapewnienia poprawnej pracy!

12

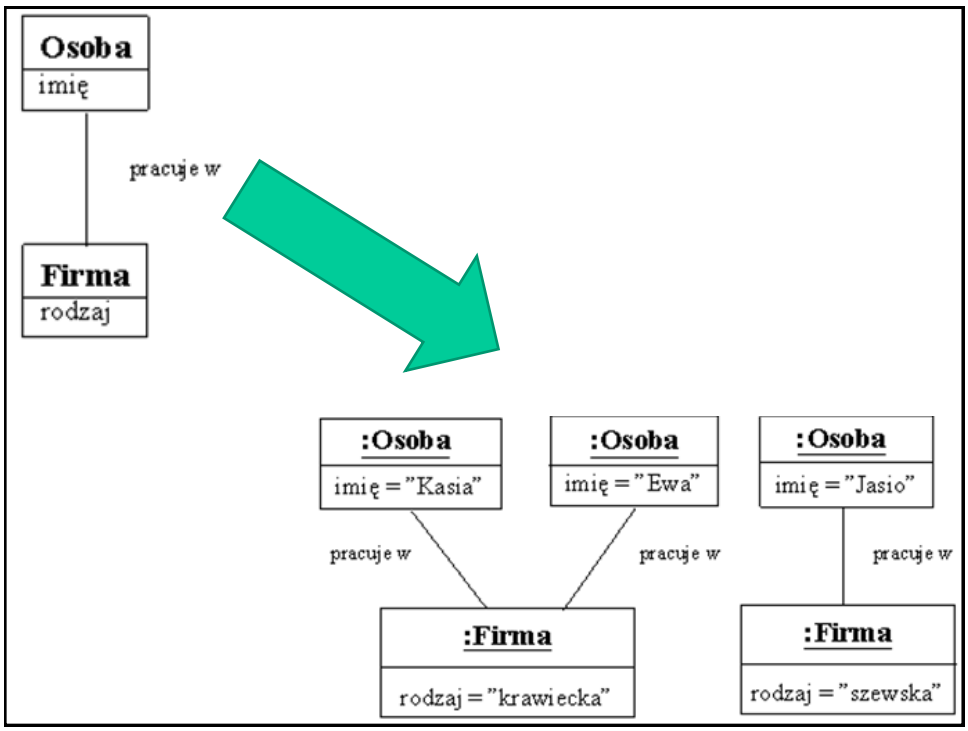
12

Asocjacja (1)

- Związek asocjacji pozwala jednej klasie na używanie obiektów innej klasy.
- Asocjacja oznacza, że klasa będzie w rzeczywistości:
  - Zawierać w postaci atrybutu odwołanie do obiektu,
  - Zawierać w postaci atrybutu same obiekty,
  - Zawierać inną klasę.

13

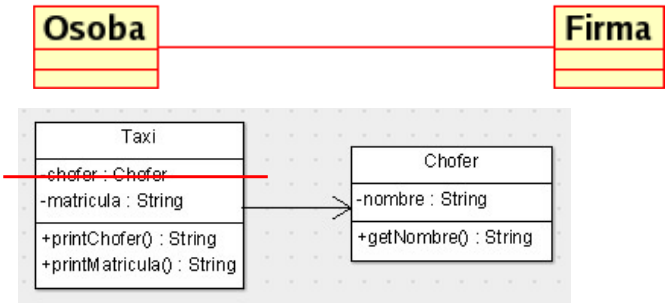
13



14

Asocjacja (2)

- Domyślnie powiązanie jest dwukierunkowe (jeśli tego nie chcemy, to trzeba do linii dodać strzałkę)

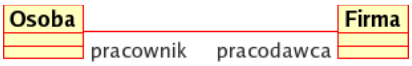


15

15

Asocjacje (3)

- Na diagramach można umieszczać dodatkowe informacje o powiązaniach
- **Rola**:



- **Krotność**:
- Przykładowe krotności: 1, 2, 5, 1..3, *, 3..*, 0,1, 0..6

16

16

Asocjacje (4)

- Za pomocą dodatkowego atrybutu można określać, czy odwołanie się do powiązania jest dostępne dla innych obiektów nie biorących w powiązaniu udziału (czy jest publiczne):

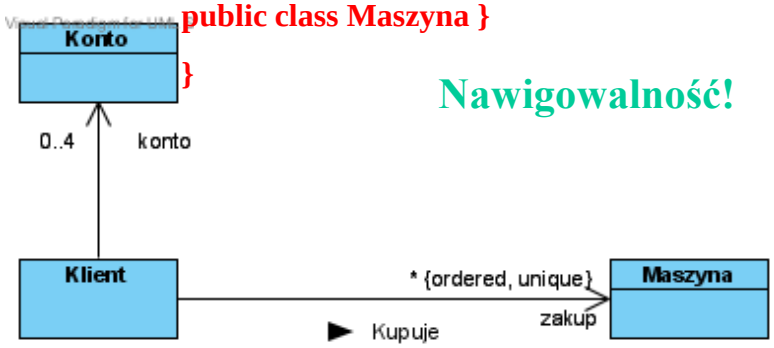
17

17

Asocjacje (5) – przykład implementacji

```
public class Klient {
        public Maszyna[] zakup;
        public Konto[] konto;
}
public class Maszyna }
}
```

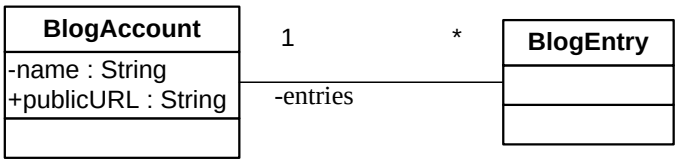
Nawigowalność!



18

18

Atrybuty

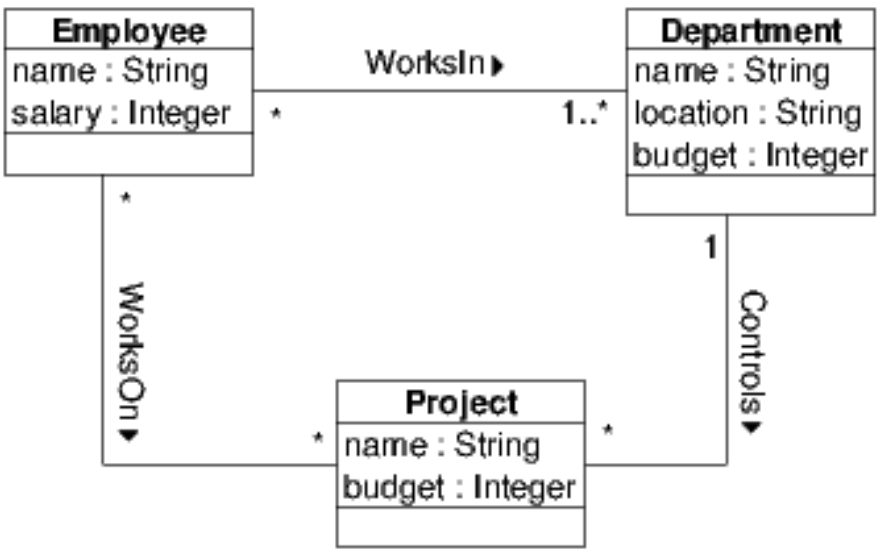


Dwa sposoby (należy używać jednego z dwóch, nie oba na raz):

- Atrybuty wpisane
- Atrybuty zadeklarowane poprzez asocjację

19

19

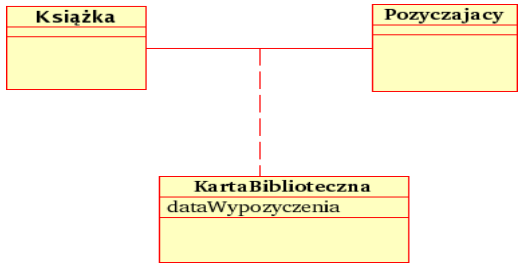


20

20

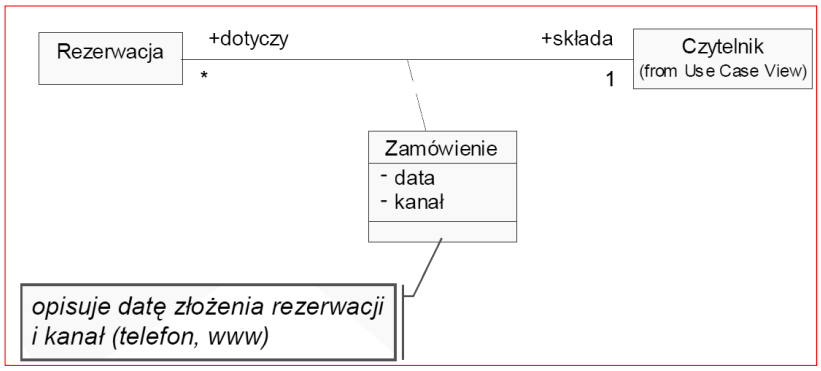
Klasy asocjacyjne

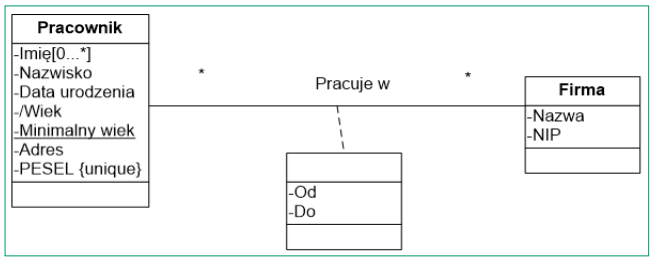
- Sama asocjacja może powodować powstawanie nowych klas.
- Powiązanie może mieć atrybuty i operacje, tak samo jak każda inna klasa.



21

21

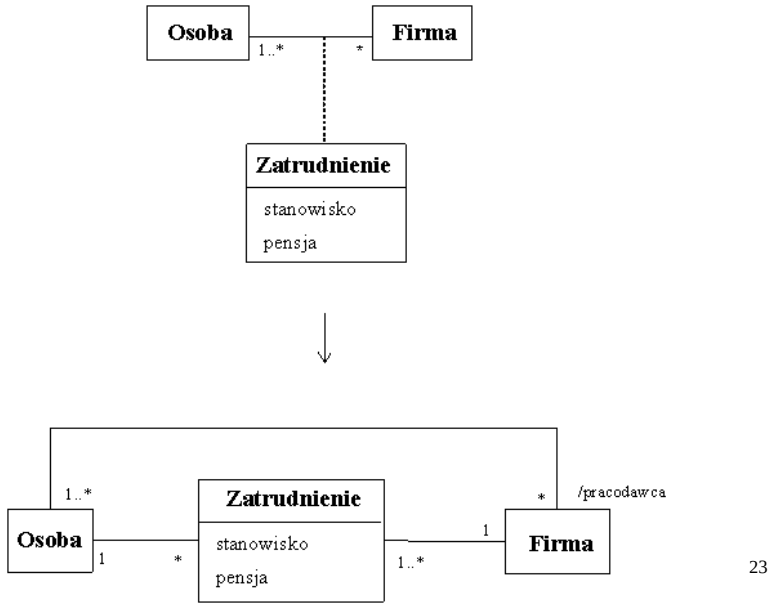




22

22

Obejście klasy asocjacji

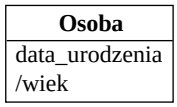


23

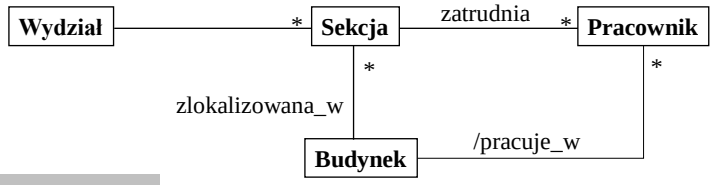
Atrybuty i asocjacje pochodne

Cecha pochodna jest zdefiniowana poprzez funkcję działającą na jednym lub więcej bytach modelu, które też mogą być pochodne. Cecha pochodna oznaczana jest ukośnikiem /.

atrybut pochodny: /wiek



{wiek = data_bieżąca - data_urodzenia}



asocjacja pochodna: /pracuje_w

Asocjacja **pracuje_w** jest **asocjacją pochodną**, którą można wyznaczyć poprzez asocjacje **zatrudnia** i **zlokalizowana_w**. Asocjację pochodną można oznaczyć poprzedzając ukośnikiem nazwę lub rolę asocjacji.

24

Agregacja częściowa

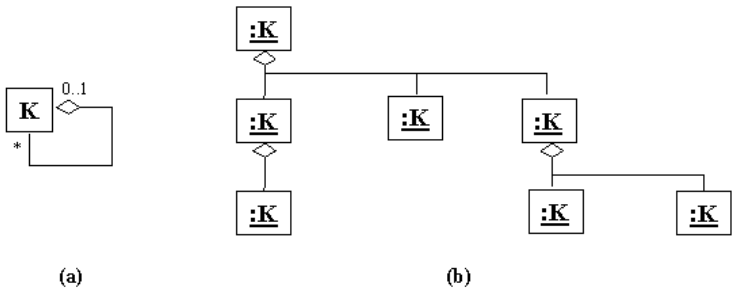
- W rzeczywistości jest silniejszą wersją asocjacji.
- Zaznacza, że jedna klasa w rzeczywistości posiada obiekty innej, ale może je jednocześnie również współdzielić.

25

25

Agregacja rekursywna

- Przykład (jedna z możliwości):



26

26

Agregacja całkowita (kompozycja, złożenie)

- Jeśli obiekt należy tylko do jednej całości, jest tworzony i likwidowany razem z całością, nazywamy to agregacją całkowitą.
- Kompozycja oznacza, że cykl życiowy składowej zawiera się w cyklu życiowym całości, oraz że składowa nie może być współdzielona.

27

27

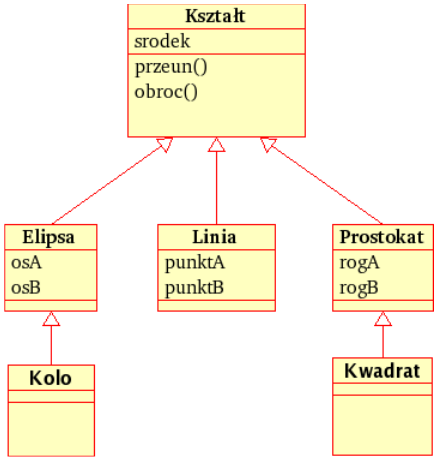
Uogólnienie (dziedziczenie)

- Używa się w celu opisania klasy, która jest **rodzajem** innej klasy.
- Uogólnienie/dziedziczenie jest relacją pomiędzy klasą ogólniejszą (nazywaną klasą rodzicem, klasą bazową itp.) a bardziej szczegółową (nazywaną podklasą, albo klasą dzieckiem) – reprezentuje stwierdzenie: “A jest rodzajem B”.

28

28

Uogólnienie - przykład



Klasa posiadająca klasy potomne, ale nie posiadająca rodziców jest nazywana klasą korzeniem (ang. *root class*)

Klasa bez potomków nazywana jest klasą liściem (ang. *leaf class*)

29

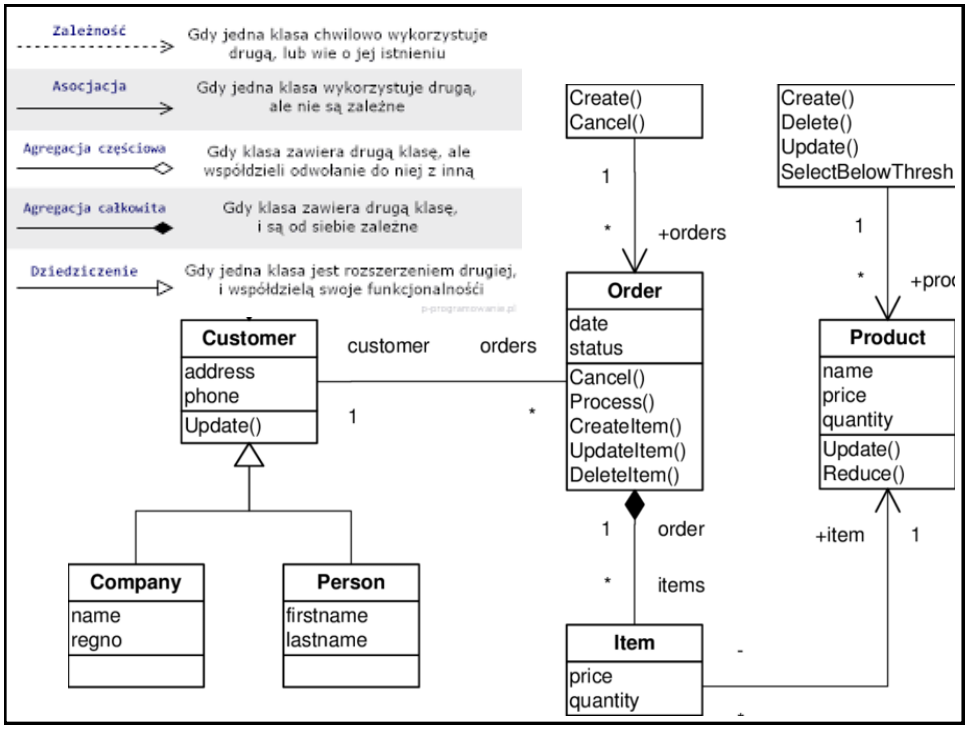
29

Dziedziczenie wielokrotne

- *Dziedziczenie wielokrotne* – zachodzi w sytuacji, gdy klasa dziedziczy z dwu lub więcej klas rodziców.
- Jest oficjalnym terminem UML.
- Jego stosowanie nie jest uznawane za najlepszą praktykę, zwłaszcza w przypadku, gdy **dwie klasy rodziców zawierają wspólne atrybuty lub współdziałają zachowanie.**

30

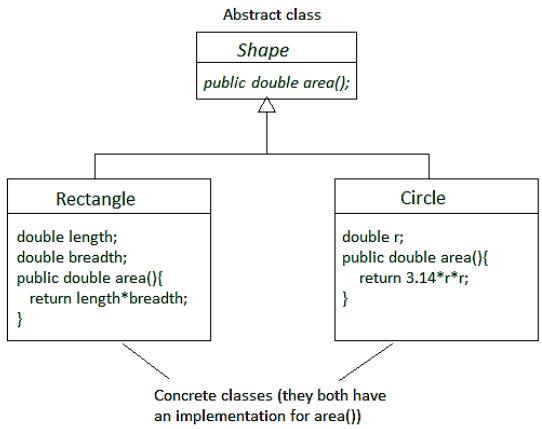
30



31

Klasy abstrakcyjne

- Nazwa klasy pisana *kursywą*.
- Metody abstrakcyjne pisane *kursywą*.



32

32

Interfejsy

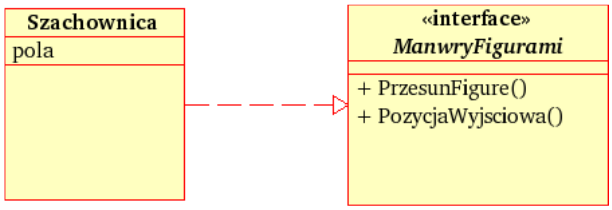
- Interfejs jest zbiorem operacji, które nie mają odpowiadających im metod implementujących.

33

33

Interfejsy, **realizacja**

- Interfejs jest przedstawiany podobnie do klasy – nie ma jednak właściwości.

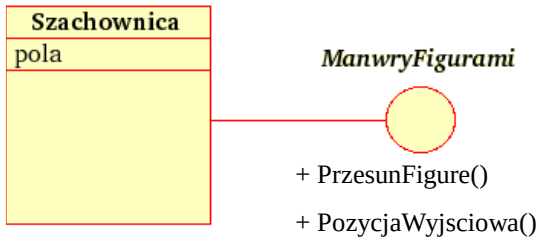


- Związek pomiędzy klasą a interfejsem nazywa się *realizacją*

34

34

Intrfejsy, notacja „lizakowa”

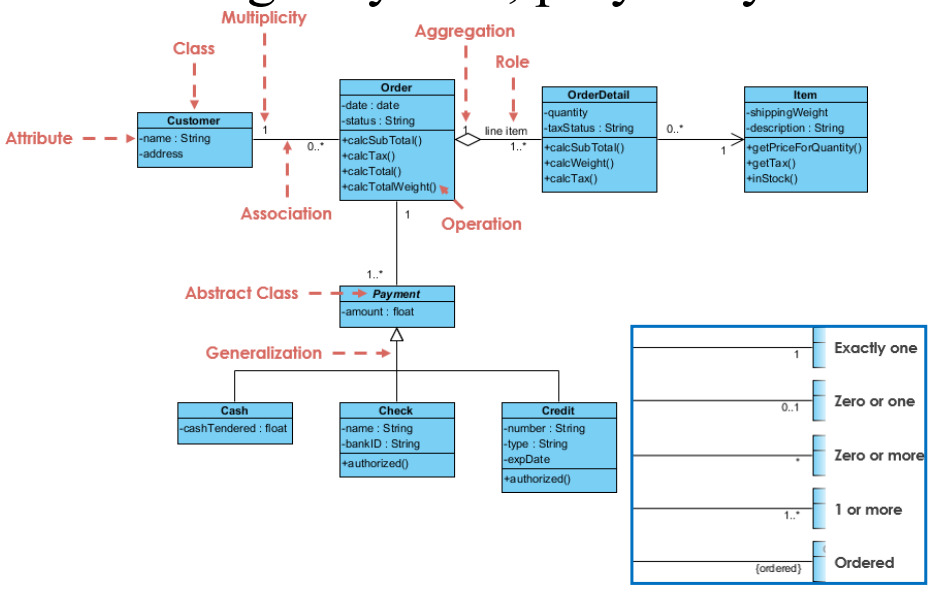


W tej notacji operacje interfejsu podajemy alternatywnie!

35

35

Diagramy klas, przykłady



36

Source code after transformation

```
public class User {
    private String email;
    public String getEmail() {
        return email;
    }
    public void setEmail(String
value){
        email = value;
    }
    public void notify(String msg) {
        // ....
    }
    /* Other methods omitted */
}
```

```
public class LeagueOwner extends User
{
    private int maxNumLeagues;
    public int getMaxNumLeagues() {
        return maxNumLeagues;
    }
    public void setMaxNumLeagues
                        (int value) {
        maxNumLeagues = value;
    }
    /* Other methods omitted */
}
```

37

37

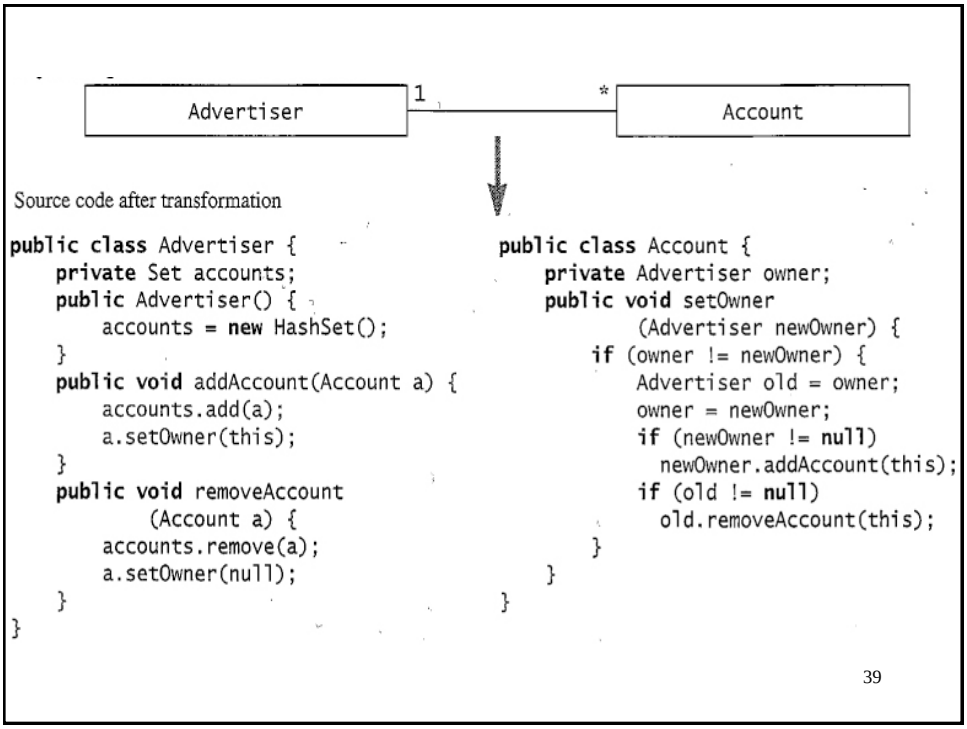
Source code after transformation

```
public class Advertiser {
    /* The account field is initialized
     * in the constructor and never
     * modified. */
    private Account account;

    public Advertiser() {
        account = new Account(this);
    }
    public Account getAccount() {
        return account;
    }
}
```

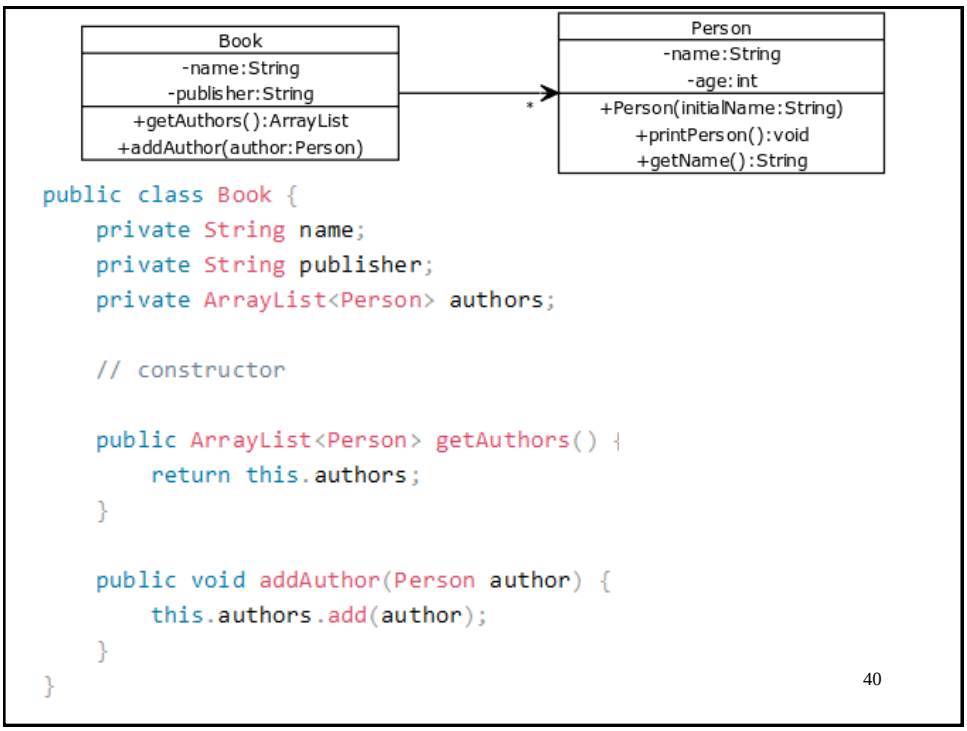
```
public class Account {
    /* The owner field is initialized
     * during the constructor and
     * never modified. */
    private Advertiser owner;

    public Account(Advertiser owner) {
        this.owner = owner;
    }
    public Advertiser getOwner() {
        return owner;
    }
}
```

38

38

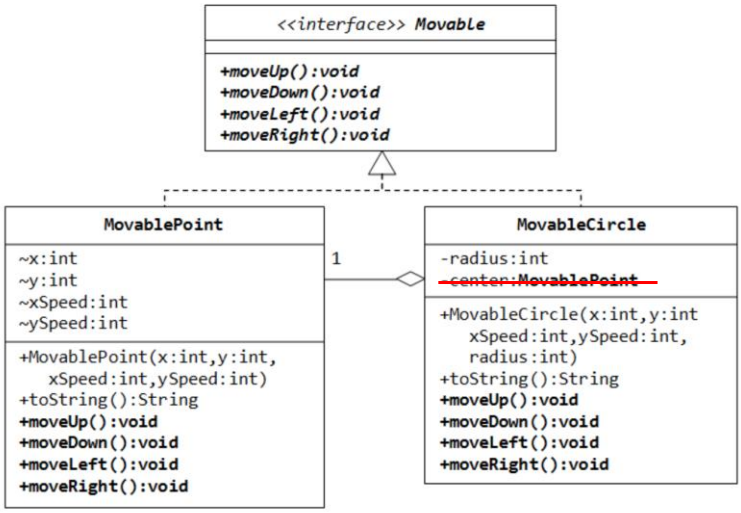

Advertiser
1
*
Account
Source code after transformation
public class Advertiser {
private Set accounts;
public Advertiser() {
accounts = new HashSet();
}
public void addAccount(Account a) {
accounts.add(a);
a.setOwner(this);
}
public void removeAccount
(Account a) {
accounts.remove(a);
a.setOwner(null);
}
}
public class Account {
private Advertiser owner;
public void setOwner
(Advertiser newOwner) {
if (owner != newOwner) {
Advertiser old = owner;
owner = newOwner;
if (newOwner != null)
newOwner.addAccount(this);
if (old != null)
old.removeAccount(this);
}
}
}
39

39


Book
-name:String
-publisher:String
+getAuthors():ArrayList
+addAuthor(author:Person)
*
Person
-name:String
-age:int
+Person(initialName:String)
+printPerson():void
+getName():String
public class Book {
private String name;
private String publisher;
private ArrayList<Person> authors;
// constructor
public ArrayList<Person> getAuthors() {
return this.authors;
}
public void addAuthor(Person author) {
this.authors.add(author);
}
}
40

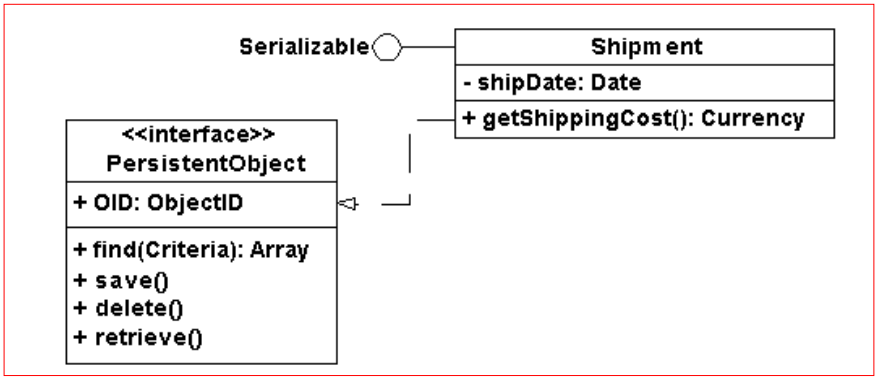
40

Diagramy klas, przykłady



41

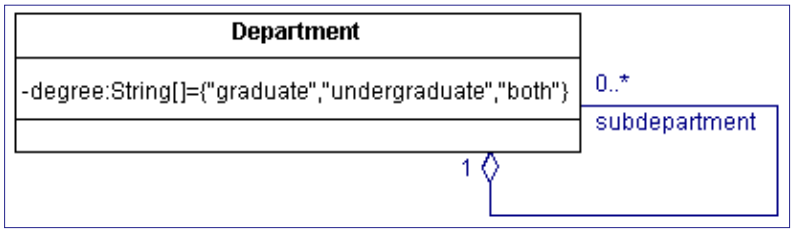
Diagramy klas, przykłady



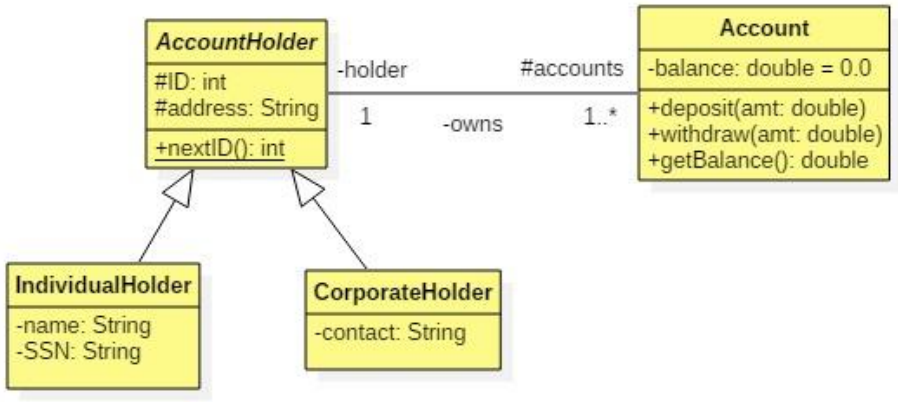
42

42

Diagramy klas, przykłady



43



44