

## **Analiza i modelowanie wymagań**

Przed kontraktem i po jego podpisaniu

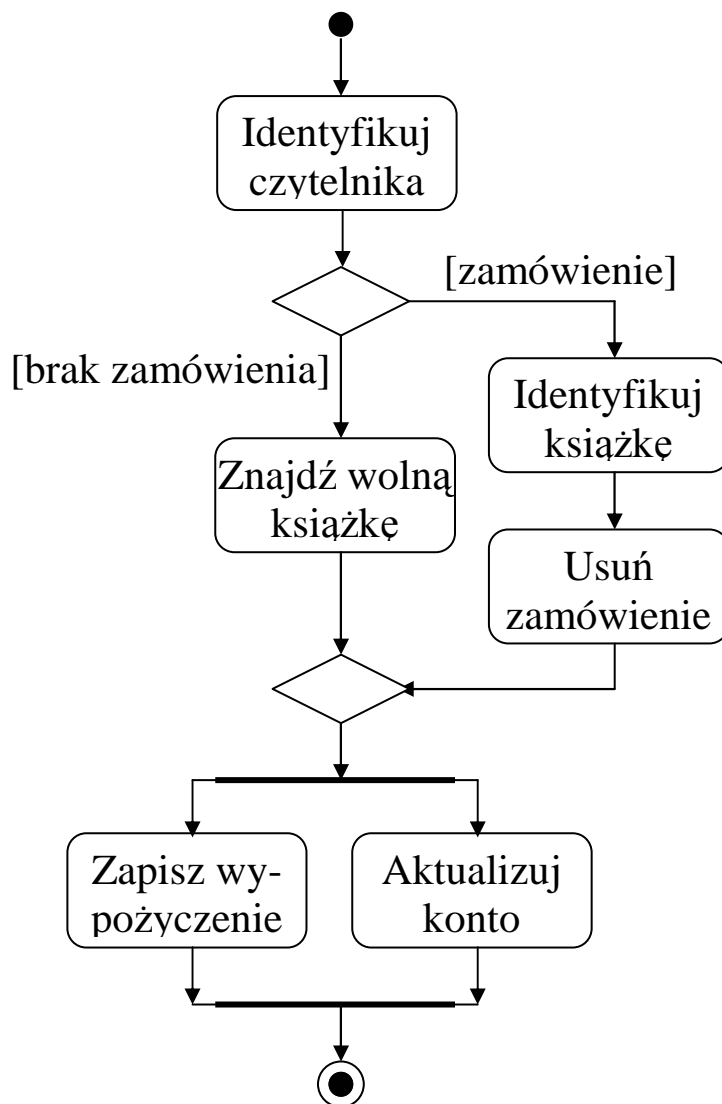
– różne poziomy szczegółowości

### ***Proces analizy wymagań***

- Rozpoznanie dziedziny zastosowania
  - pozyskanie danych
  - procesy biznesowe
  - obszary tematyczne
- Wstępna analiza obszarów tematycznych
  - identyfikacja zadań i danych
  - model przepływu zadań
  - wizja obszaru
- Identyfikacja i dokumentacja przypadków użycia
  - model przypadków użycia
  - model dziedziny problemu
  - pozostałe wymagania

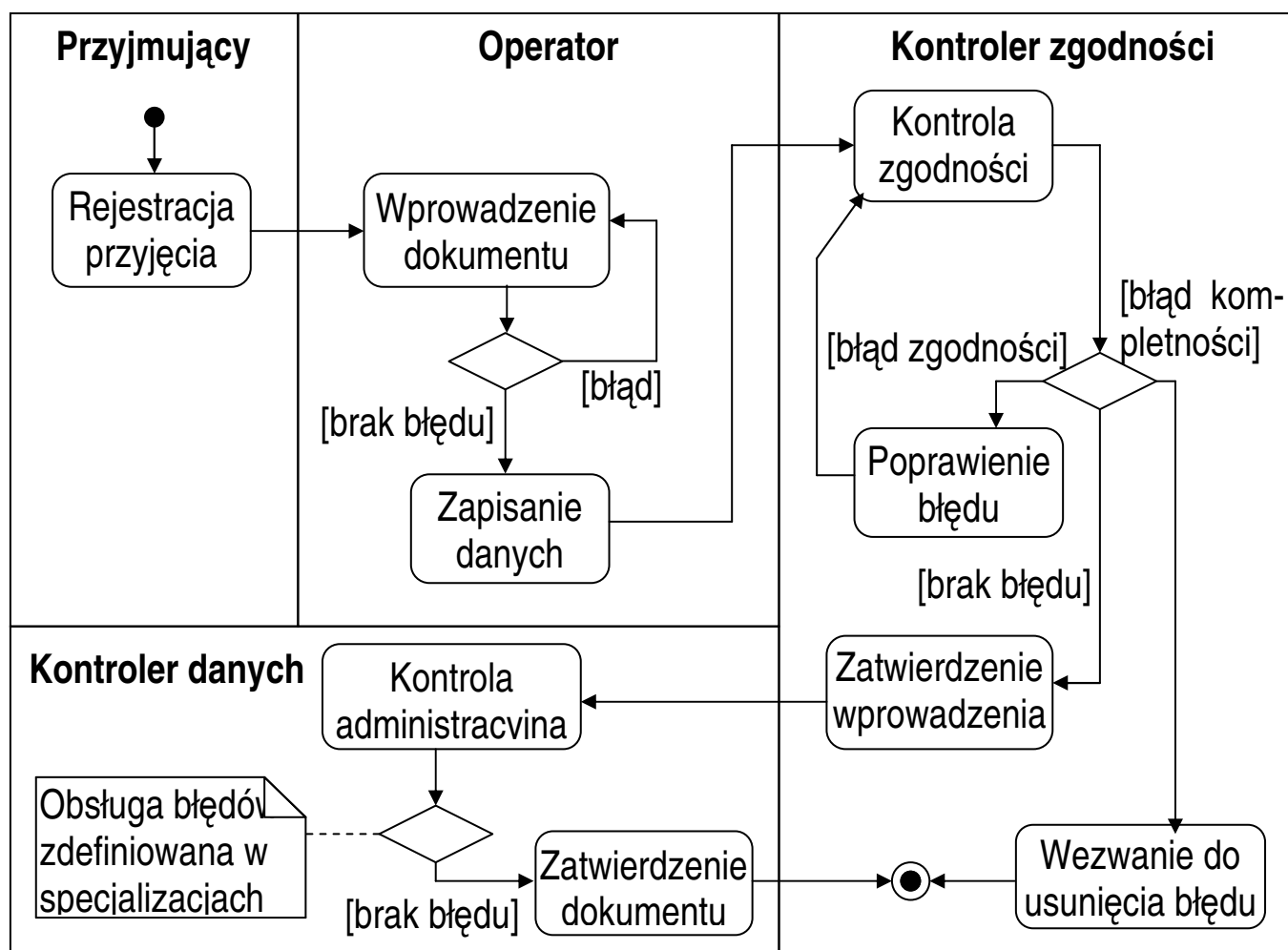
## Diagram aktywności (activity diagram)

- Model przepływu sterowania
- Rozszerzona sieć działań



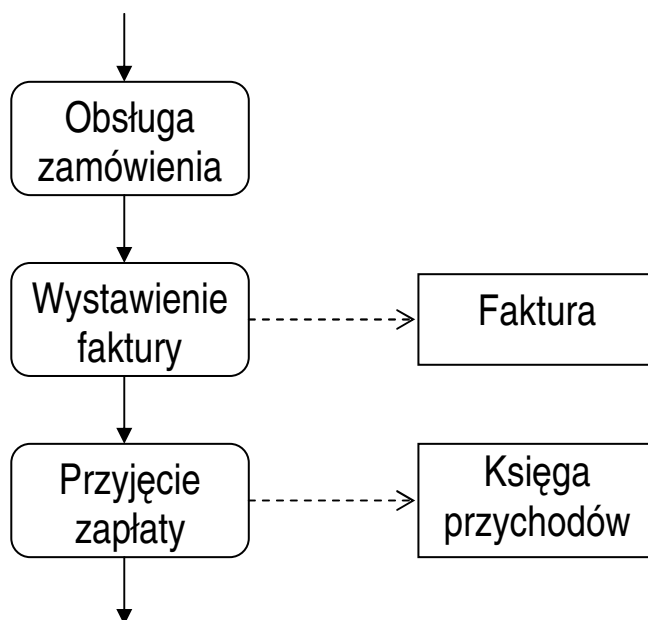
## Wskazanie odpowiedzialności

- Notki
- Boksy (pasy)



## Wskazanie obiektu czynności

- Relacja zależności



## **Zastosowanie diagramów aktywności**

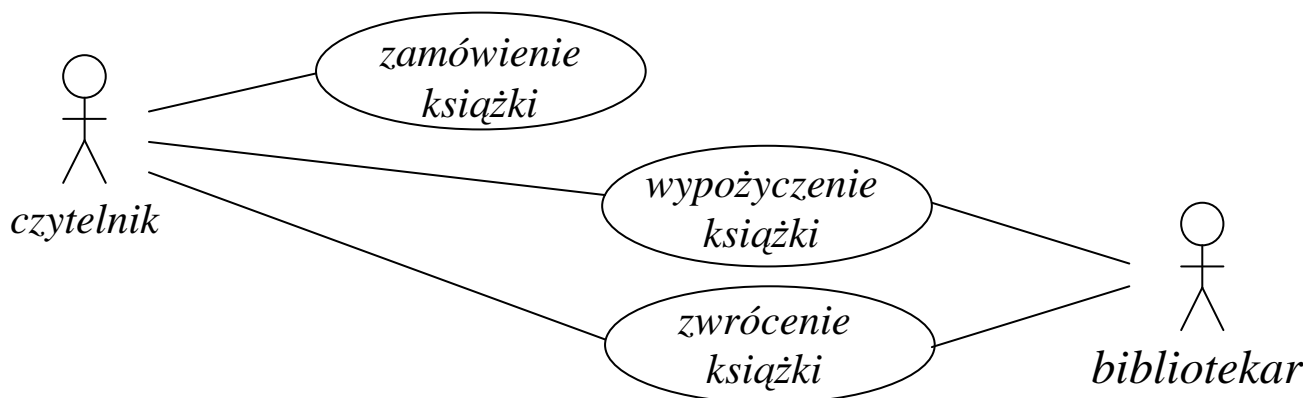
- Kolejność wykonania czynności
  - procesy biznesowe (*workflow*)
  - scenariusze przypadków użycia
  - algorytmy
- Reprezentacja hierarchiczna

## **Metoda przypadków użycia (use case)**

- Identyfikacja użytkowników
- Modelowanie procedur biznesowych

### **Model przypadków użycia**

- Elementy modelu
  - aktor
  - przypadek użycia
  - relacje
- Reprezentacja graficzna



## **Scenariusz przypadku użycia**

### zamówienie książki

- **Scenariusz główny**

1. System prezentuje ekran wyszukiwania.
2. *Czytelnik* wprowadza dane bibliograficzne.
3. System przeszukuje katalog i wyświetla listę pozycji (tytułów).
4. *Czytelnik* przegląda listę i wybiera pozycję.
5. System wyświetla listę książek.
6. *Czytelnik* zamawia wolną książkę.
7. System wyświetla okno logowania.
8. *Czytelnik* wprowadza nazwę i hasło.
9. System autoryzuje czytelnika.
10. System potwierdza przyjęcie zamówienia.

- **Scenariusz alternatywny 1 (odmowa autoryzacji)**

- 1 – 8. Jak w scenariuszu głównym.
9. System odmawia autoryzacji: powrót do kroku 7.

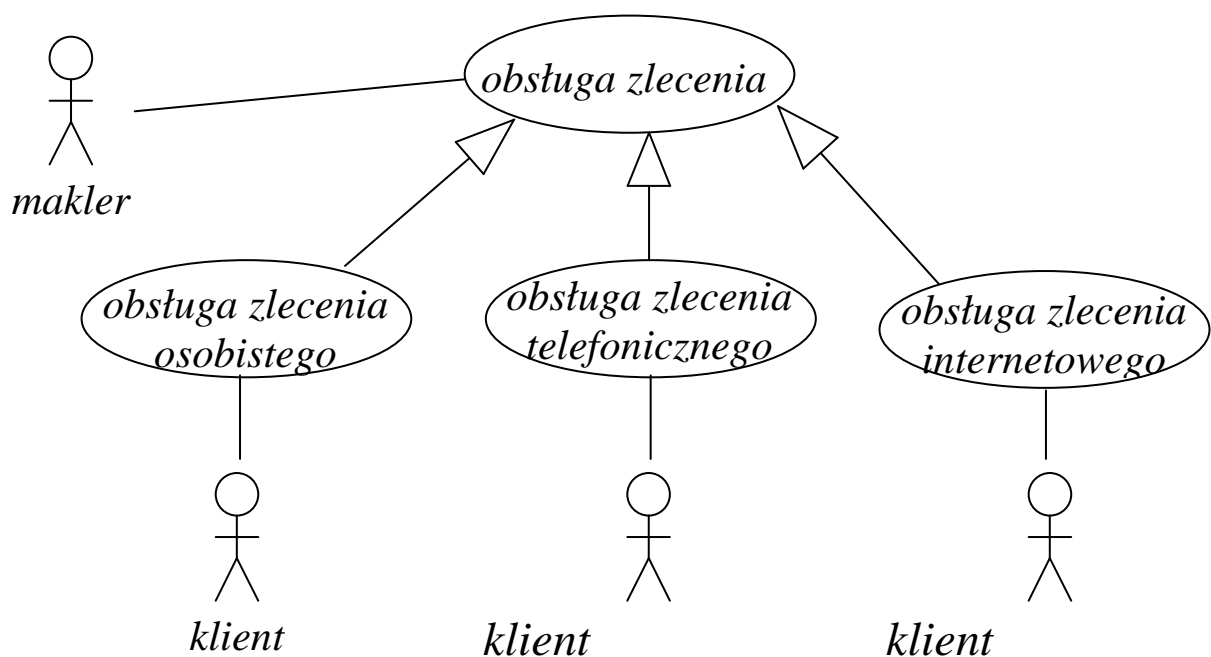
- **Scenariusz alternatywny 2 (brak wolnej książki)**

- 1 – 5. Jak w scenariuszu głównym.
6. Brak wolnej książki: czytelnik zapisuje się do kolejki.
7. System wyświetla okno logowania.
8. *Czytelnik* wprowadza nazwę i hasło.
9. System autoryzuje czytelnika.
10. System potwierdza zapisanie do kolejki.

## Generalizacja

- Ogólny schemat i specyficzne realizacje
- Ten sam cel
- Różni aktorzy

Przykład: biuro maklerskie





obsługa zlecenia

1. Makler przyjmuje zlecenie i wprowadza do systemu BM.
2. System BM blokuje środki na rachunku klienta.
3. System BM przekazuje zlecenie do systemu Warset.

obsługa zlecenia telefonicznego

- 1.1. Makler odbiera i wprowadza do systemu BM dane klient .
- 1.2. Makler odbiera i wprowadza do systemu hasło klienta.
- 1.3. System potwierdza autoryzację klienta.
- 1.4. Makler odbiera i wprowadza zlecenie do systemu BM.
- 2 – 3. Jak w przypadku generalizującym.

obsługa zlecenia internetowego

.....

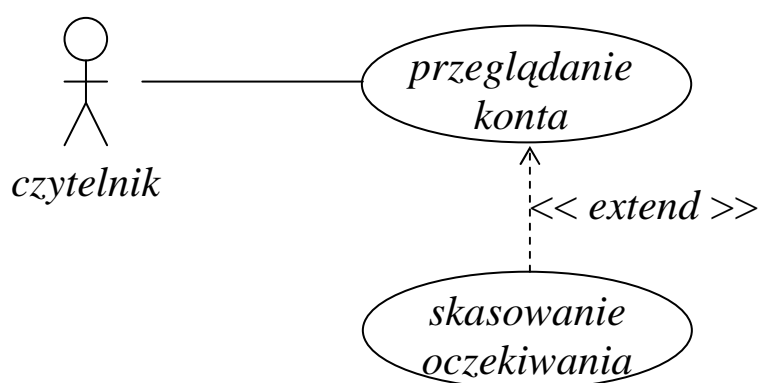
obsługa zlecenia osobistego

.....

## Rozszerzenie

- Typowy schemat i dodatkowe czynności
- Określone punkty rozszerzenia
- Brak odrębnych aktorów

Przykład: system biblioteczny



### przeglądanie konta

1. Czytelnik wybiera opcję przeglądania konta.
2. System wyświetla okno logowania.
3. Czytelnik wprowadza nazwę i hasło.
4. System autoryzuje czytelnika.
5. System wyświetla stan konta (*kasowanie*).
6. Czytelnik wybiera następną opcję.

### skasowanie zamówienia

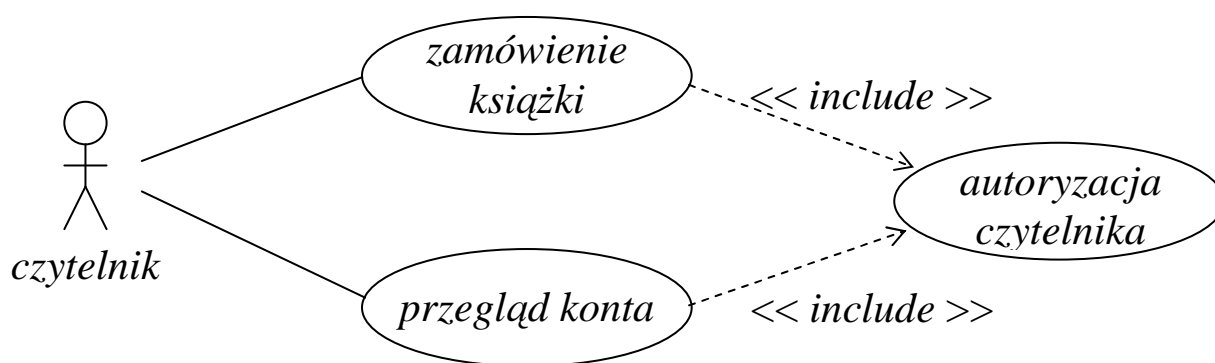
wstawić w punkcie rozszerzenia *kasowanie*:

1. Czytelnik kasuje zamówienie.
2. System potwierdza skasowanie zamówienia.

## Zawieranie

- Wyodrębnienie fragmentu przypadku użycia
- Możliwe fragmenty wspólne
- Brak odrębnych aktorów

Przykład: system biblioteczny.



zamówienie książki

1. System prezentuje ekran wyszukiwania.
2. *Czytelnik* wprowadza dane bibliograficzne.
3. System przeszukuje katalog i wyświetla listę pozycji (tytułów).
4. *Czytelnik* przegląda listę i wybiera pozycję.
5. System wyświetla listę książek.
6. *Czytelnik* zamawia wolną książkę.
7. **Wykonaj autoryzację czytelnika.**
8. System potwierdza przyjęcie zamówienia.

autoryzacja czytelnika

1. System wyświetla okno logowania.
2. *Czytelnik* wprowadza nazwę i hasło.
3. System autoryzuje czytelnika.

## Przykład: system biblioteczny

- przeglądanie katalogu i zamawianie przez Internet
- kolejka do zwolnienia pozycji
- różne kategorie książek i czytelników

- **Aktorzy**

*czytelnik, bibliotekarz.*

- **Przypadki użycia** (istotne dla czytelnika)

<i>przeglądanie katalogu,</i>	<i>zamówienie książki,</i>
<i>wypożyczenie książki,</i>	<i>zwrócenie książki,</i>
<i>zapisanie do kolejki,</i>	<i>przeglądanie konta,</i>
<i>usunięcie zamówienia,</i>	<i>usunięcie z kolejki.</i>

- **Przypadki użycia** (istotne dla bibliotekarza)

<i>dodanie czytelnika,</i>	<i>usunięcie czytelnika,</i>
<i>dodanie książki,</i>	<i>usunięcie książki.</i>

### Ograniczenie zakresu modelu

Inni aktorzy, np.: *konserwator,*  
*magazynier* — pomijamy.

Inne przypadki, np.: *wycofanie książki,*  
*przywrócenie książki,*  
*przeniesienie książki* — pomijamy.

przeglądanie katalogu

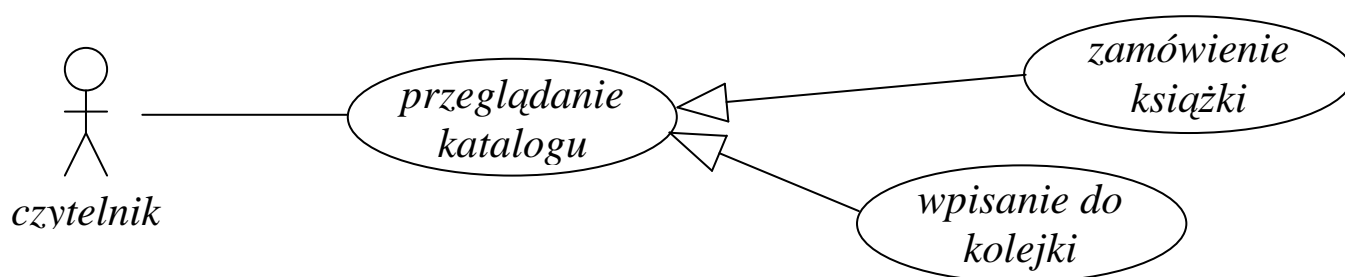
1. System prezentuje ekran wyszukiwania.
2. Czytelnik wprowadza dane bibliograficzne.
3. System przeszukuje katalog i wyświetla listę pozycji (tytułów).
4. Czytelnik przegląda listę i wybiera pozycję.
5. System wyświetla listę książek.

zamówienie książki (specjalizacja przeglądania):

- 1–5. Jak w przypadku generalizującym.
6. Czytelnik zamawia wolną książkę.
7. Wykonaj autoryzację czytelnika.
8. System potwierdza przyjęcie zamówienia.

wpisanie do kolejki (specjalizacja przeglądania):

- 1–5. Jak w przypadku generalizującym.
6. Brak wolnej książki: czytelnik zapisuje się do kolejki oczekiwania na daną pozycję (tytuł).
7. Wykonaj autoryzację czytelnika.
8. System potwierdza zapisanie do kolejki.

autoryzacja czytelnika — b.z. jak poprzednio.

**Alternatywa:** rozszerzenie?

### przeglądanie konta

1. Czytelnik wybiera opcję przeglądania konta.
2. Wykonaj autoryzację czytelnika.
3. System wyświetla stan konta (*operacje*).

### skasowanie zamówienia

wstawić w punkcie rozszerzenia *operacje*:

1. Czytelnik kasuje zamówienie.
2. System potwierdza skasowanie zamówienia.

### usunięcie z kolejki

wstawić w punkcie rozszerzenia *operacje*:

1. Czytelnik usuwa wpis w kolejce.
2. System potwierdza usunięcie z kolejki.

### wypożyczenie książki

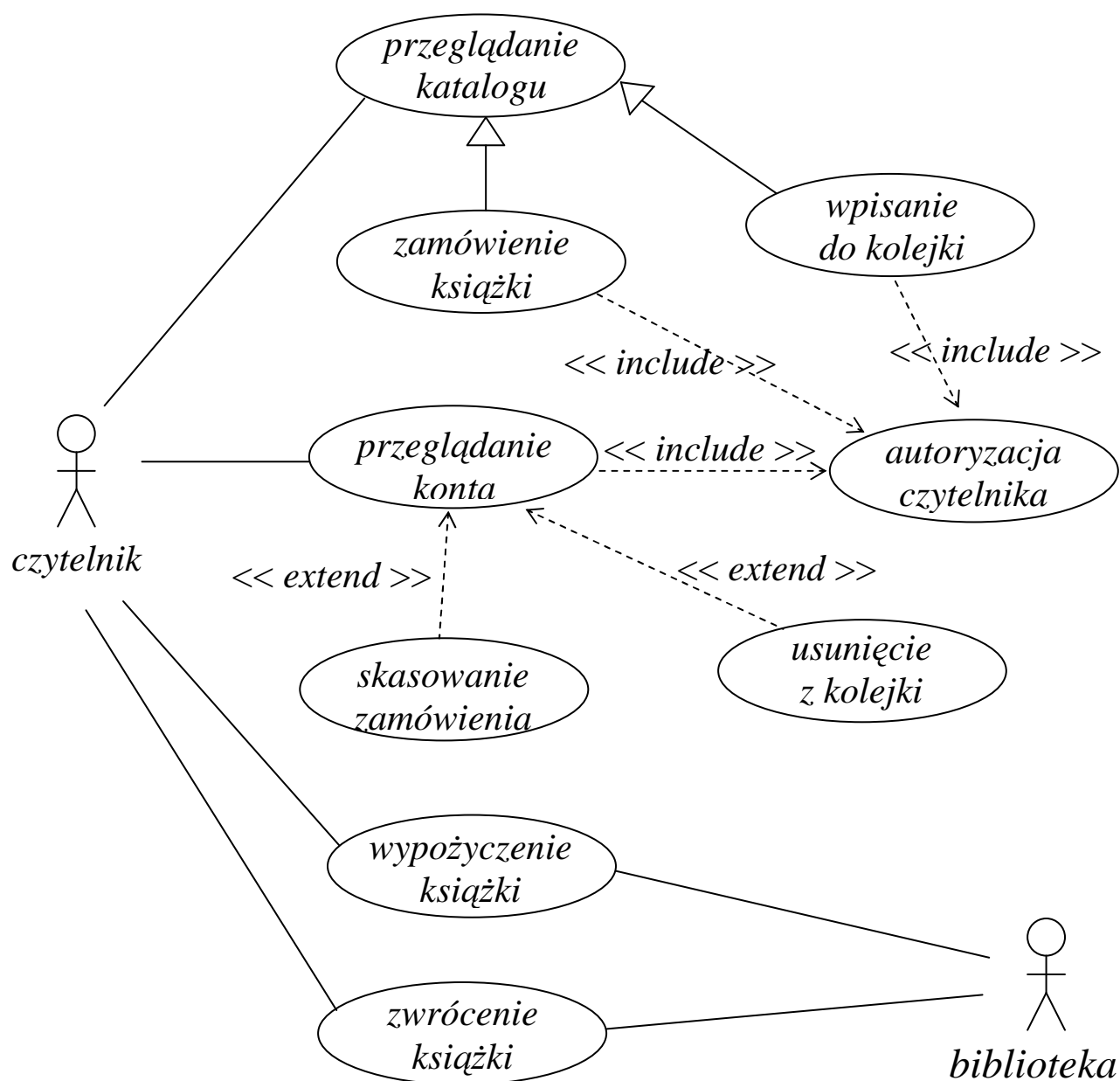
1. Identyfikacja karty czytelnika.
2. Identyfikacja książki.
3. System usuwa zamówienie.
4. System zapisuje wypożyczenie.

### zwrócenie książki

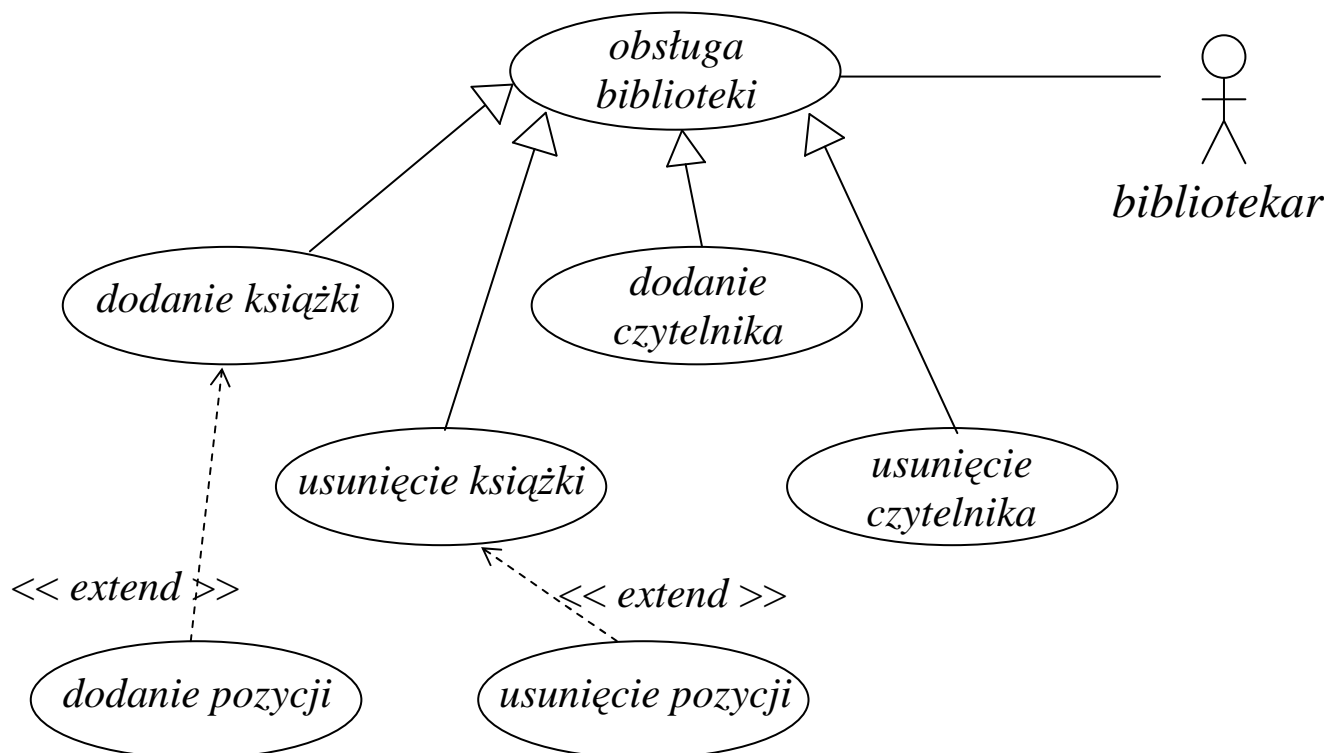
1. Identyfikacja książki.
2. System usuwa wypożyczenie.
3. System wyszukuje oczekującego i rezerwuje dla niego książkę



## Model z punktu widzenia czytelnika



## Model z punktu widzenia utrzymania biblioteki



## Specyfikacja modelu

- Lista przypadków użycia (model graficzny)
- Scenariusze
- Reguły biznesowe

Przykład: system biblioteczny

### 1. Okres wypożyczenia:

- książki z wypożyczalni studenckiej — na 2 miesiące
- książki z antresoli — na 2 tygodnie
- książki swobodnego dostępu — na 1 tydzień

### 2. Liczba książek wypożyczonych:

- pracownicy — 3
- doktoranci i studenci — 2

### 3. Książki z czytelni:

- pracownicy — na 1 miesiąc
- doktoranci — na 1 tydzień
- podczas wakacji ?

### 4. Opłata za opóźnienie zwrotu:

- czy wszyscy płacą tyle samo ?
- jeśli termin w niedzielę ?

### 5. Biblioteki płacą za wypożyczenie:

- okres wypożyczenia ?
- stawka ?

## **Podsumowanie: metoda przypadków użycia**

1. Identyfikacja aktorów
2. Określenie przypadków użycia
3. Określenie reguł biznesowych
4. Uporządkowanie modelu
5. Dokumentacja modelu
6. Budowa modelu dziedziny

### Uwagi:

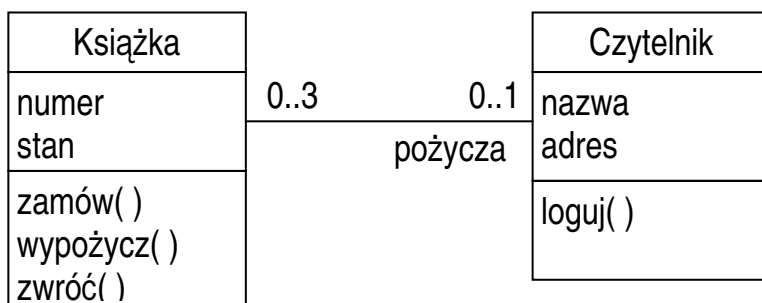
- Pożądany prototyp
- Brak odniesień projektowych
- Dalsze zastosowania modelu

## **Modelowanie dziedziny problemu**

- Wyodrębnienie obiektów dziedziny problemu
  - analiza pojęć
  - analiza scenariuszy
  - określenie atrybutów
- Porządkowanie dziedziny
  - klasyfikacja obiektów
  - identyfikacja związków klas
  - dokumentowanie
- Określenie typów zachowań
  - identyfikacja stanów obiektu
  - identyfikacja zmian
  - dokumentowanie

## Diagram klas

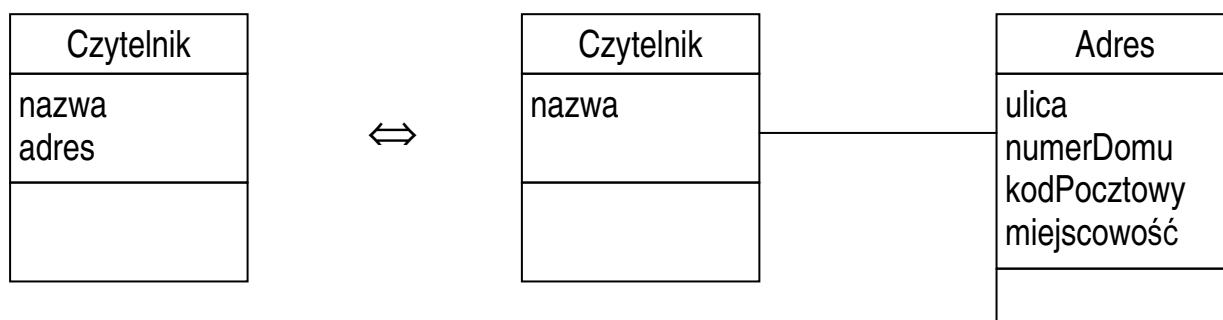
- Model statycznej struktury problemu
- Elementy modelu
  - klasy
  - relacje (*uogólnienie, stowarzyszenie*)
- Reprezentacja graficzna



## Perspektywy widzenia modelu

- **Model pojęciowy**

- model analizy
- pokazuje klasy i atrybuty, relacje klas
- brak szczegółów implementacyjnych

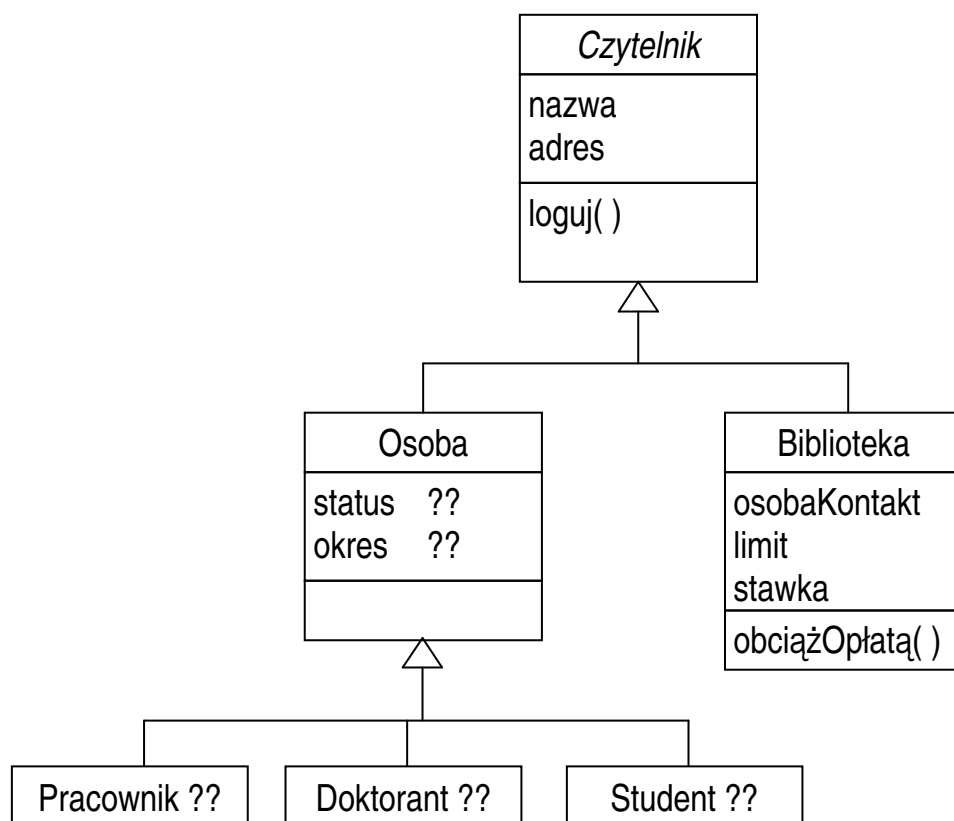


- **Model projektowy (programistyczny)**

- model projektu i implementacji
- określa strukturę danych (pola) i metody klas
- klasyfikacja opisuje hierarchię dziedziczenia

## Uogólnienie

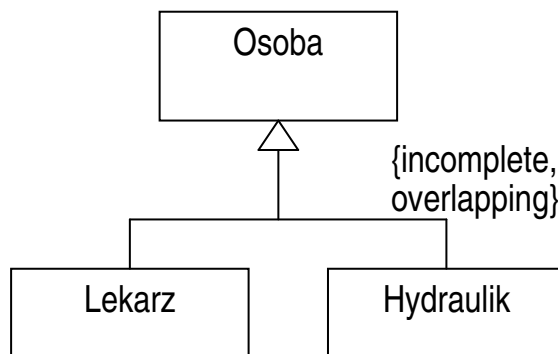
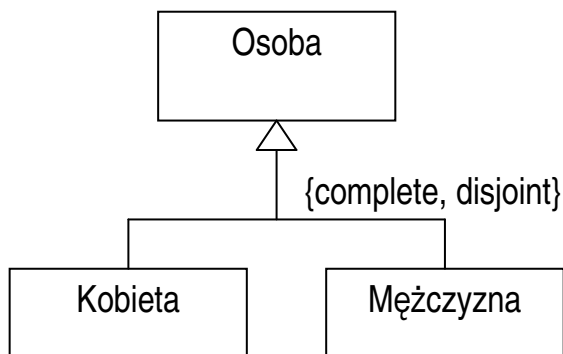
Klasyfikacja, modele na różnych poziomach abstrakcji



- **Model pojęciowy**
  - klasyfikacja bytów
- **Model projektowy**
  - hierarchia interfejsów i dziedziczenia

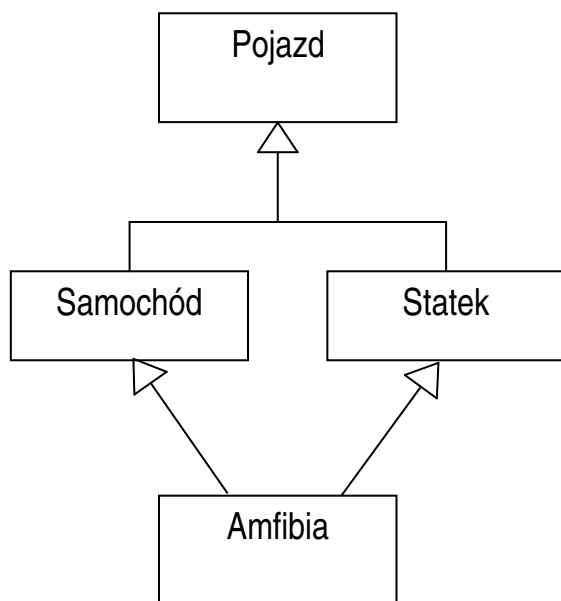


- Ograniczenia klasyfikacji



domyślnie: incomplete, disjoint

?? Spójność diagramu

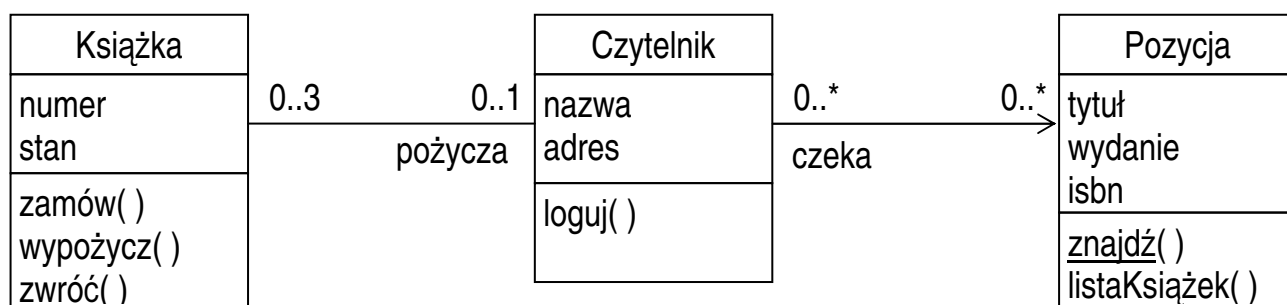


## Asocjacja

Trwałe powiązania między obiektami, niekoniecznie typu 1–1

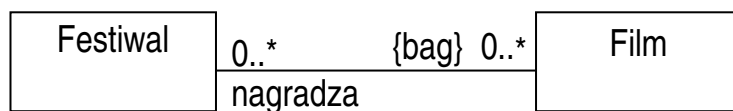
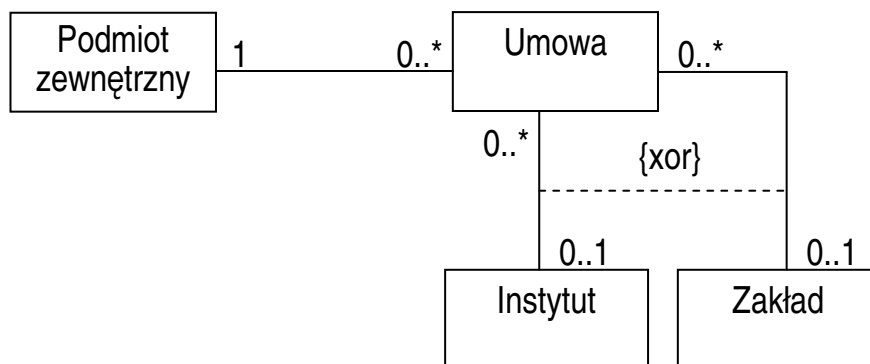
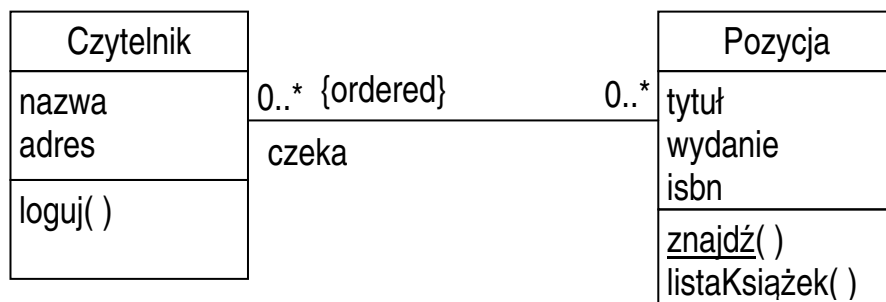
Opis relacji:

- krotność
- nazwa
- kierunek



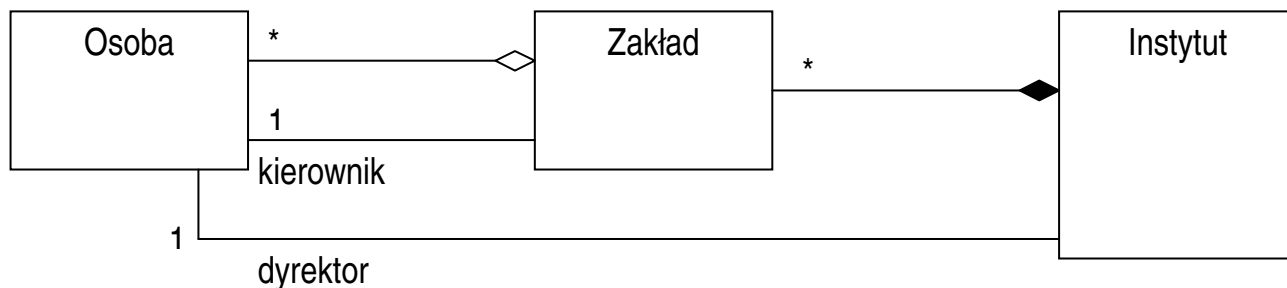
- **Model pojęciowy**
  - związek pojęciowy między klasami
- **Model projektowy**
  - operacje nawigowania między obiektami
  - operacje umożliwiające tworzenie relacji
  - pola klas wskazujące powiązania

- Ograniczenia asocjacji

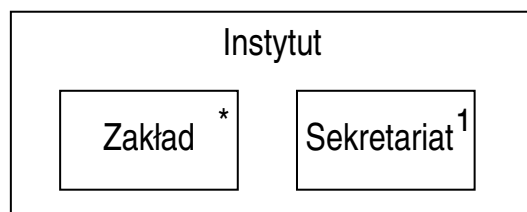


## Agregacja

Szczególny przypadek asocjacji: związek typu całość – część



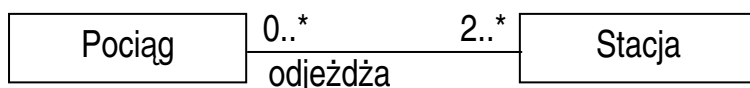
Złożenie lub zawieranie (*composition*)



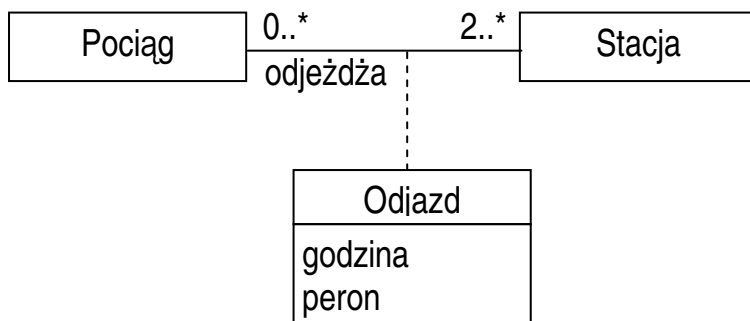
## Klasy asocjacyjne

- Klasa jest zbiorem obiektów
  - element klasy (obiekt) opisują wartości atrybutów.
- Asocjacja dwóch klas jest relacją tzn. zbiorem par obiektów
  - element relacji (parę obiektów) opisują atrybuty obiektów.

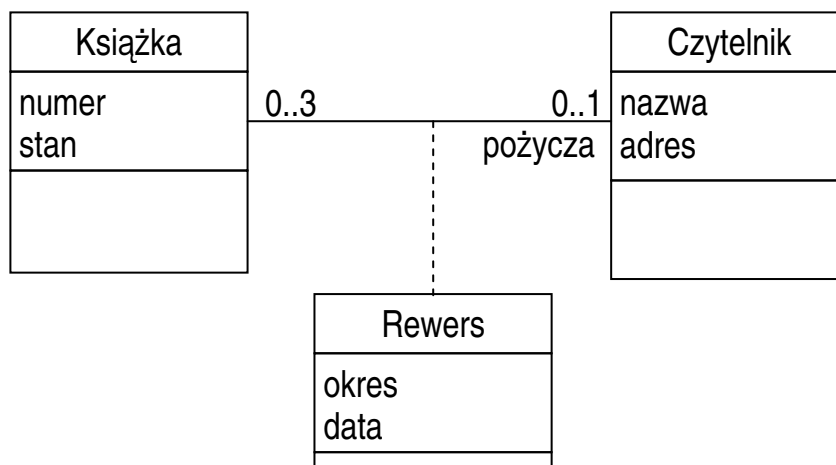
**Asocjacja nie ma atrybutów.**



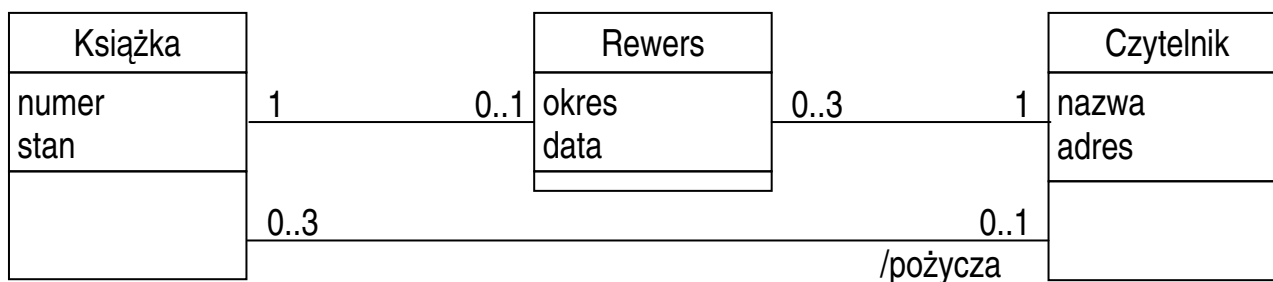
→ do kogo należy atrybut **godzina odjazdu** ?



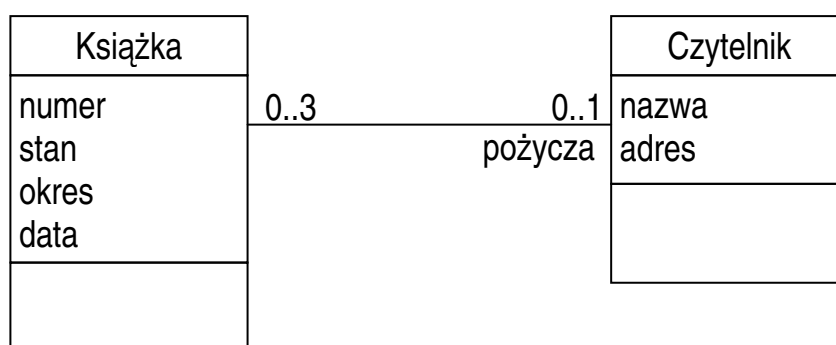
- Klasa asocjacyjna



- Wariant alternatywny — zwykła klasa.



- Trzeci wariant — dodatkowe atrybuty



## **Zastosowanie diagramów klas**

### **1. Modelowanie dziedziny problemu**

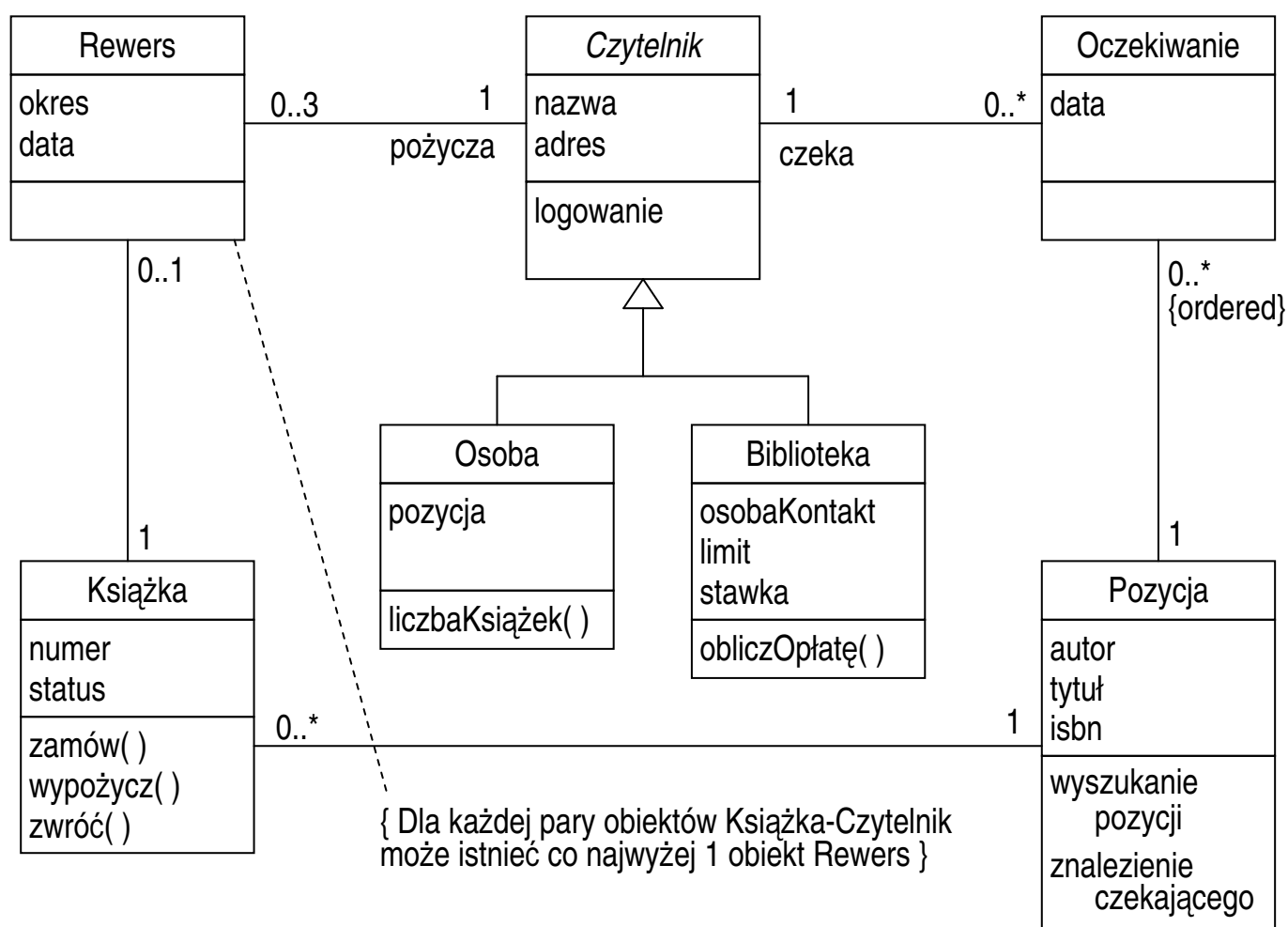
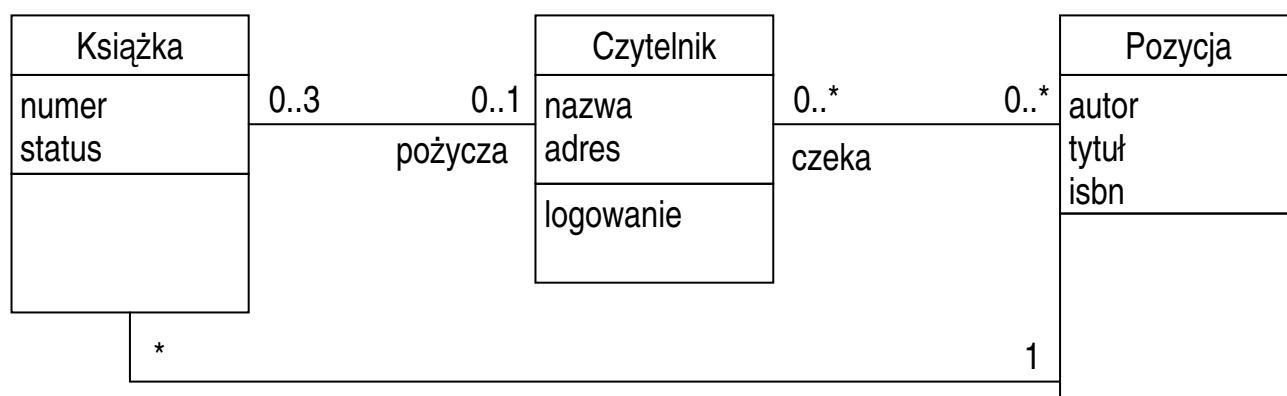
- wyodrębnienie klas w dziedziny problemu
- pokazanie relacji klas (model pojęciowy)
- analiza relacji
- nieformalny opis odpowiedzialności

### **2. Projektowanie logicznej struktury aplikacji**

- określenie zachowań ( $\rightarrow$  operacje i metody, nowe klasy)
- porządkowanie hierarchii klas (model projektowy)
- wyodrębnienie modelu bazy danych
- powiązanie z interfejsem użytkownika

### **3. Projektowanie fizycznej struktury aplikacji**

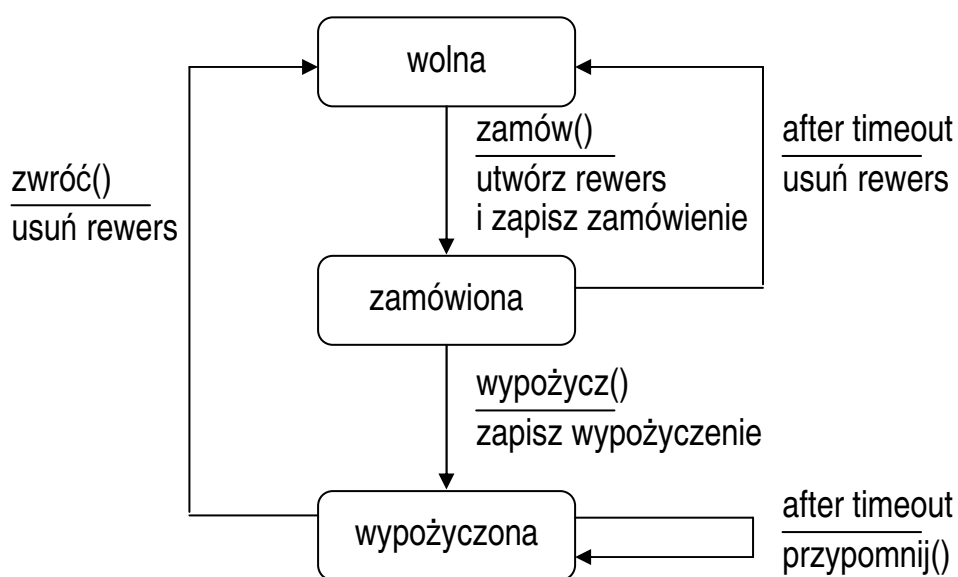
- określenie komponentów i ich interfejsów
- uporządkowanie hierarchii dziedziczenia
- określenie klas implementujących operacje

Przykład: system biblioteczny



## Diagram stanów

- Model zachowania obiektów pewnej klasy
- Elementy modelu
  - stany
  - przejścia między stanami
- Reprezentacja graficzna



- Model formalny — teoria automatów (*finite state machine*)

- **Opis stanu w języku UML**

<i>nazwa</i>	ciąg znaków (mogą istnieć stany bez nazwy)
<i>entry/akcja()</i>	akcja wejściowa
<i>exit/akcja()</i>	akcja wyjściowa
<i>zdarzenie/akcja()</i>	akcja wewnętrzna
<i>zdarzenie/defer</i>	zdarzenie zapamiętywane
<i>do/działanie</i>	czynność (procedura) wykonywana w stanie

- **Opis przejścia**

*zdarzenie[dozór]/akcja*

<i>zdarzenie</i>	zdarzenie wywołujące przejście
<i>dozór</i>	warunek logiczny umożliwiający przejście
<i>akcja()</i>	akcja wykonywana podczas przejścia

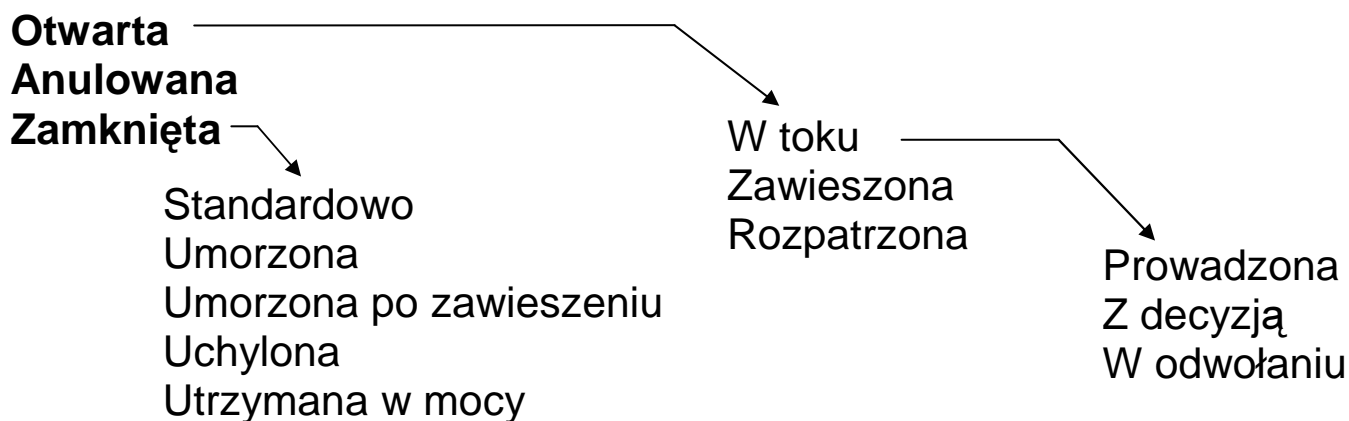
Zdarzenie:

- wywołanie metody
- spełnienie warunku *when warunek*,
- czasowe *after okres*.

## Hierarchia stanów

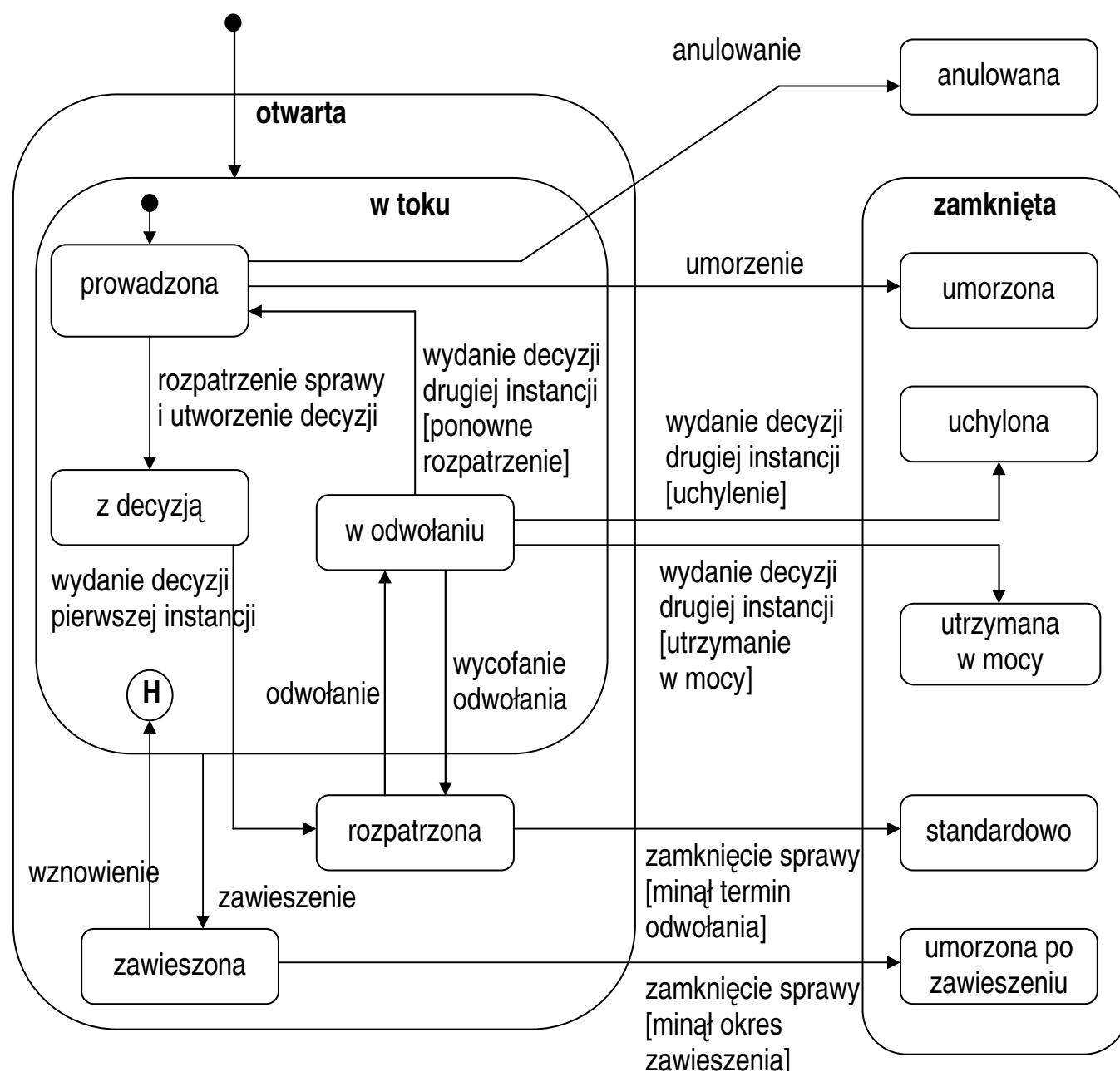
- Modelowanie na różnych poziomach abstrakcji
- Grupa pokrewnych stanów → nadstan (stan złożony)

Przykład: stany sprawy administracyjnej w systemie IACS.



Reguły przetwarzania sprawy są w różnych stanach różne

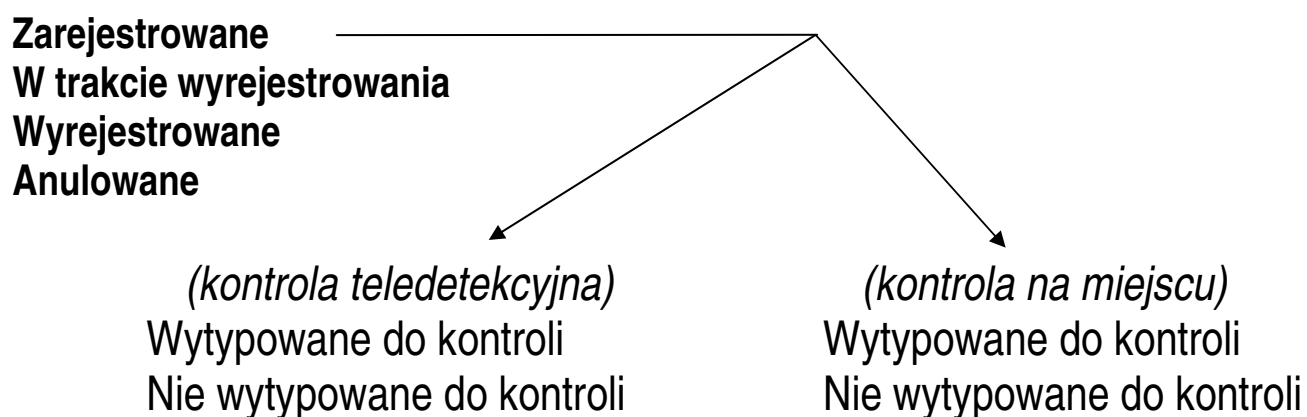
## Diagram stanów sprawy



## Stany równoległe

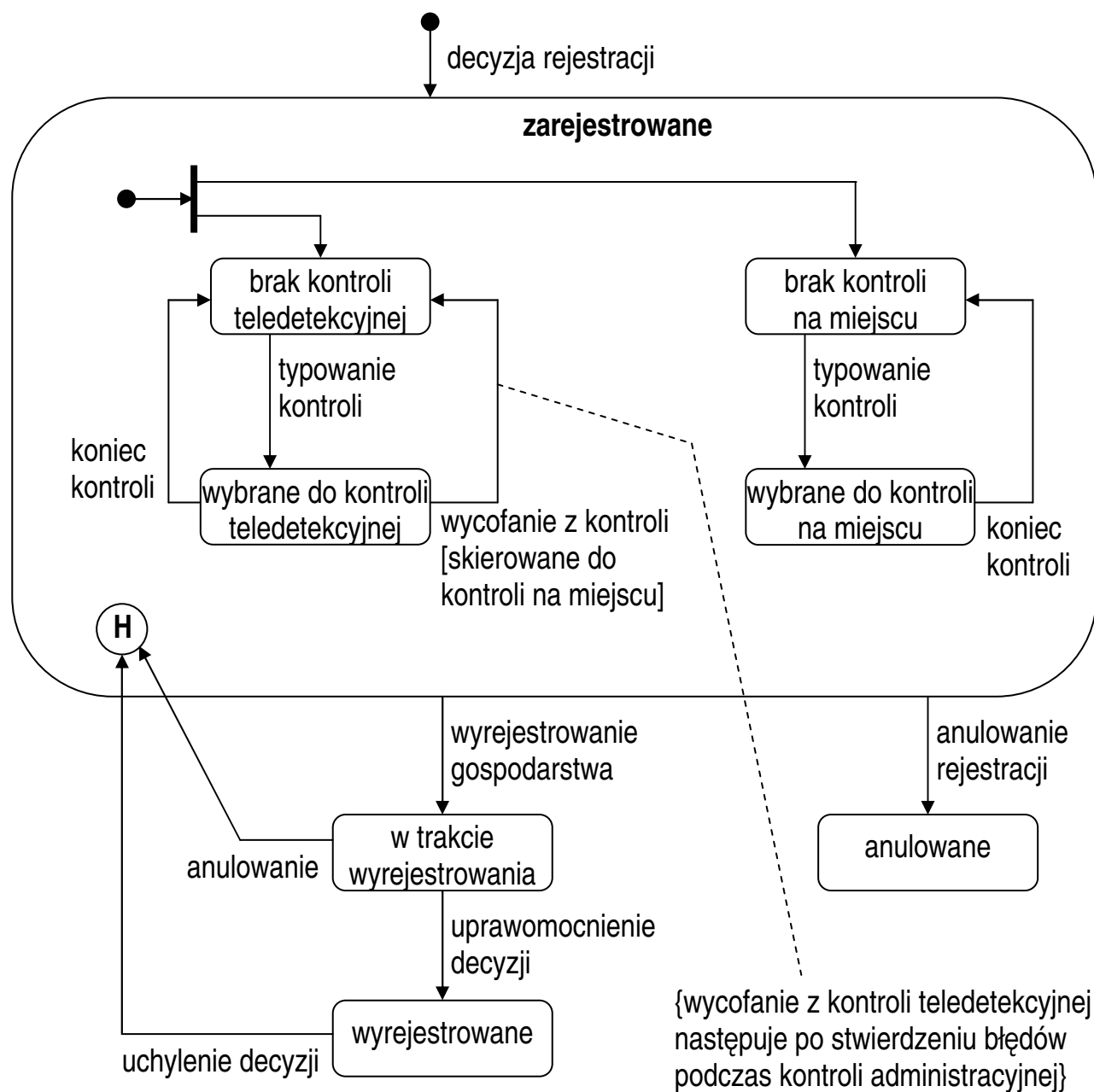
- Różne wymiary zachowania → niezależne pod-diagramy

Przykład: stany gospodarstwa rolnego w systemie IACS.



Typowanie obydwu rodzajów kontroli są niezależne

## Diagram stanów kontroli gospodarstwa



→ Równoległe grafy stanów można formalnie połączyć w jeden automat sekwencyjny.

## **Zastosowanie diagramów stanów**

### Modelowanie dynamiki obiektów

1. Pierwszy model dotyczy obiektów z dziedziny aplikacji (model dziedziny)

Stany opisują różne typy zachowań:

- różne sposoby reagowania obiektu
- możliwość wykonania różnych zbiorów operacji

2. Później obejmuje wszystkie obiekty o ciekawej dynamice (szczególnie takich, które pełnią rolę sterującą)

Opis stanowy ułatwia weryfikację poprawności

3. Podczas implementacji diagramy stanu ułatwiają kodowanie (algorytmiczne przejście do kodu programu)

→ *Opis automatowy jest szczególnie ważny w systemach R-T*

## **Projektowanie logicznej struktury aplikacji**

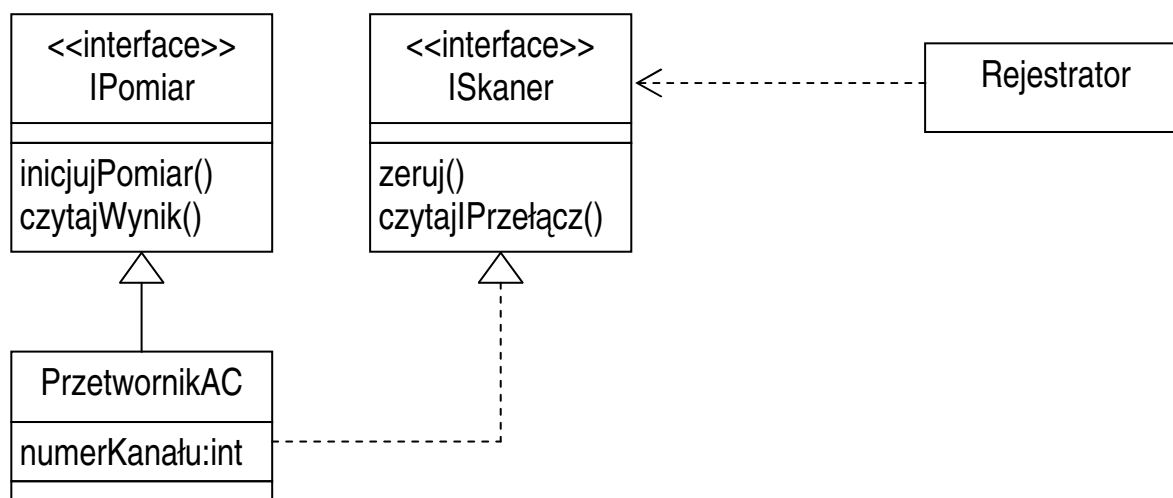
- Weryfikacja diagramu klas
  - hierarchia dziedziczenia
  - analiza asocjacji
  - reguły biznesowe
- Określenie modelu danych
  - obiekty trwałe
  - atrybuty i metody
  - hierarchia dziedziczenia
- Określenie struktury i zachowania aplikacji
  - powiązanie z interfejsem użytkownika
  - powiązanie z bazą danych
  - określenie modelu współdziałania



## Diagram klas

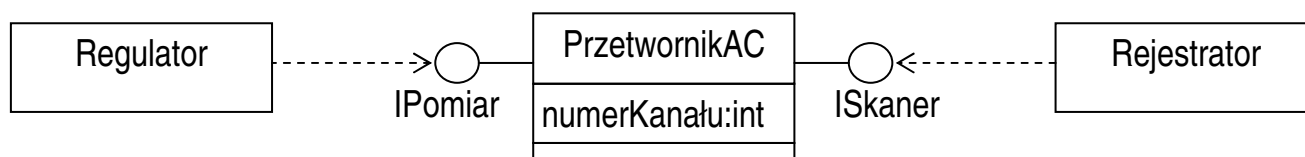
### Interfejsy

- Metod realizujących operacje interfejsu może dostarczyć:
  - klasa pochodna (relacja **generalizacji**)
  - inna klasa lub komponent systemu (relacja **realizacji**)



- Klasa używająca interfejsu jest z nim w relacji **zależności**

### Notacja alternatywna



## Dodatkowe elementy opisu

- stereotypy
  - <<interface>>      tylko operacje (brak atrybutów i metod)
  - <<type>>            tylko atrybuty i operacje (brak metod)
  - <<business>>        klasa bierna, nie inicjuje działań
- metki
  - { abstract }            klasa abstrakcyjna (też *kursywa*)
  - { root }                klasa najwyższa w hierarchii generalizacji
  - { leaf }                klasa najniższa w hierarchii generalizacji
  - { persistent }        klasa reprezentująca obiekty bazy danych

## Opis atrybutów

- **Model pojęciowy** – nazwa i nieformalny opis znaczenia
- **Model projektowy** – pełna specyfikacja postaci:

widoczność nazwa [ krotność ] : typ = wartość-domyślna { właściwości }

**widoczność** określa dostępność atrybutu w modelu:

- +     dostęp publiczny
- #     dostęp chroniony
- dostęp prywatny

**krotność** określa liczbę wartości danego atrybutu, np.:

- 1             atrybut obowiązkowy
- 0..1          atrybut opcjonalny
- k lub k..n    postać ogólna

**właściwości** definiują dodatkowe cechy atrybutu:

- changeable    atrybut modyfikowalny bez ograniczeń
- addOnly       brak możliwości usuwania (krotność > 1)
- frozen         atrybut stały (ustawiany podczas inicjalizacji)

Wyróżnienia:

- atrybuty klasowe:     podkreślenie

## Opis operacji

- **Model pojęciowy** – brak lub odpowiedzialności
- **Model projektowy** – pełna specyfikacja postaci:

widoczność nazwa ( lista-argumentów ) : typ { właściwości }

**widoczność** określa dostępność operacji w całym modelu

**lista-argumentów** określa argumenty operacji:

sposób-przekazywania nazwa : typ = wartość-domyślna

### sposób-przekazywania

in przekazanie przez wartość  
out przekazanie przez referencję  
inout przekazanie przez referencję

**właściwości** definiują dodatkowe cechy operacji:

leaf	operacja nie jest polimorficzna
isQuery	operacja nie zmienia atrybutów
sequential	wymaga sekwencyjnego działania obiektu
guarded	wykonywana rozłącznie z innymi
concurrent	wykonywana współbieżnie z innymi

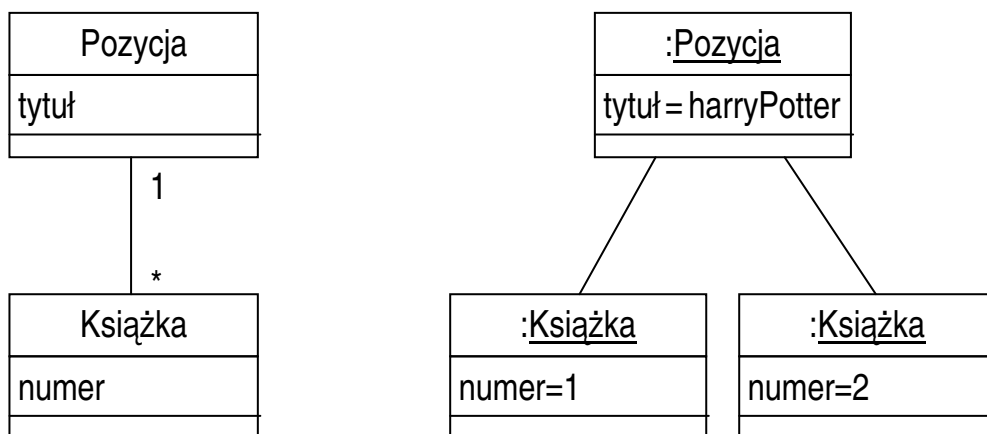
### Wyróżnienia:

- |                          |                     |
|--------------------------|---------------------|
| – operacje klasowe:      | <u>podkreślenie</u> |
| – operacje abstrakcyjne: | <i>kursywa</i>      |

## Diagram obiektów

Diagram klas — ogólne związki między obiektami  
Diagram obiektów — chwilowa konfiguracja obiektów

- Notacja
  - ta sama symbolika, prostokąty reprezentują obiekty
  - nazwy obiektów obiekt: klasa
- Użyteczność
  - w prostych przypadkach niewielka



- w złożonych modelach rekurencyjnych duża

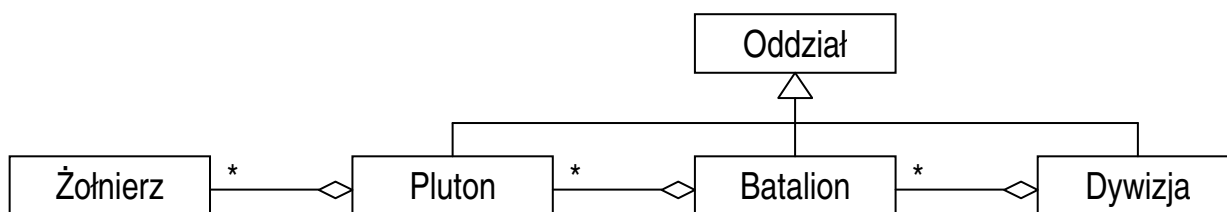
## Przykład: gra komputerowa

- żołnierze, o zdolności bojowej charakteryzowanej przez uzbrojenie i wydajność (0...1),
- oddziały, których zdolność bojowa jest sumą zdolności bojowej żołnierzy.

System zna zdolność bojową żołnierzy i oddziałów i oferuje jednolite metody posługiwania się żołnierzami i oddziałami

**I wariant:** prosty diagram klas

Ogólna klasa Oddział i agregacja oddziałów i żołnierzy



Wady:

- sztywna hierarchia oddziałów (reorganizacja po bitwie)
- brak jednolitego interfejsu oddziałów i żołnierzy

## II wariant: wzorzec projektowy *Composite*

- wspólna nadklasa dla oddziałów i żołnierzy (Jednostka)
- model rekurencyjny

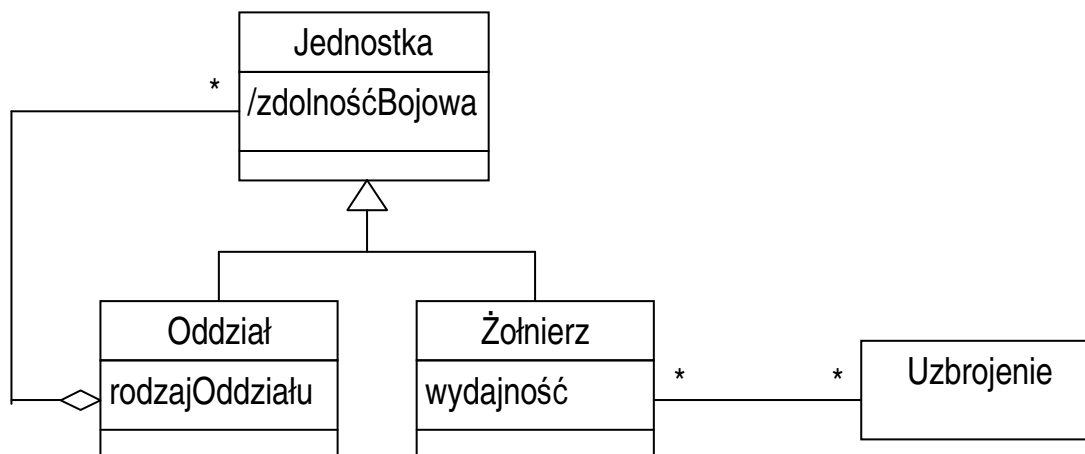
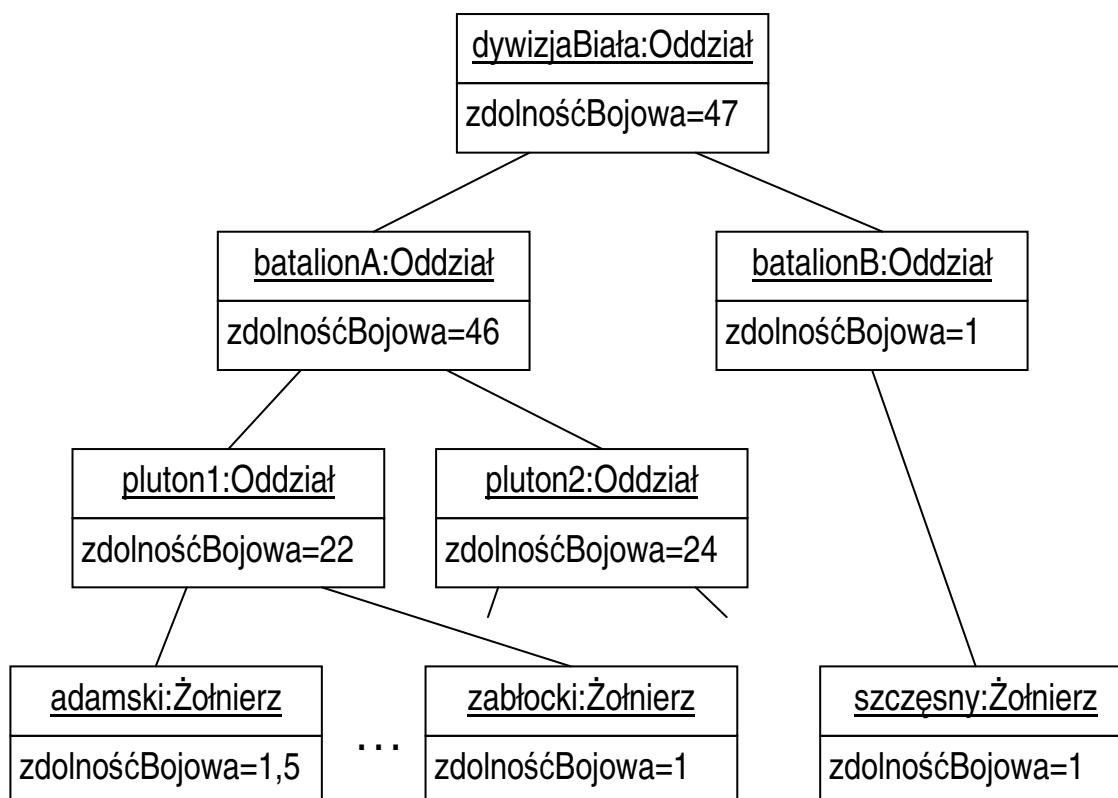


Diagram obiektów pokazuje chwilową konfigurację



→ Konfiguracja obiektów a reprezentacja to nie jest to samo!

## Modelowanie współpracy obiektów

- Karty CRC
- Diagram sekwencji
- Diagram współdziałania

### Karty CRC

- Nieformalny opis zachowania klas
- Elementy opisu klasy
  - nazwa
  - lista zobowiązań
  - lista klas współpracujących
- Postać

Książka	
Utwórz rewers i zapisz zamówienie	Rewers
Dokonaj wypożyczenia	Pozycja
Dokonaj zwrotu	

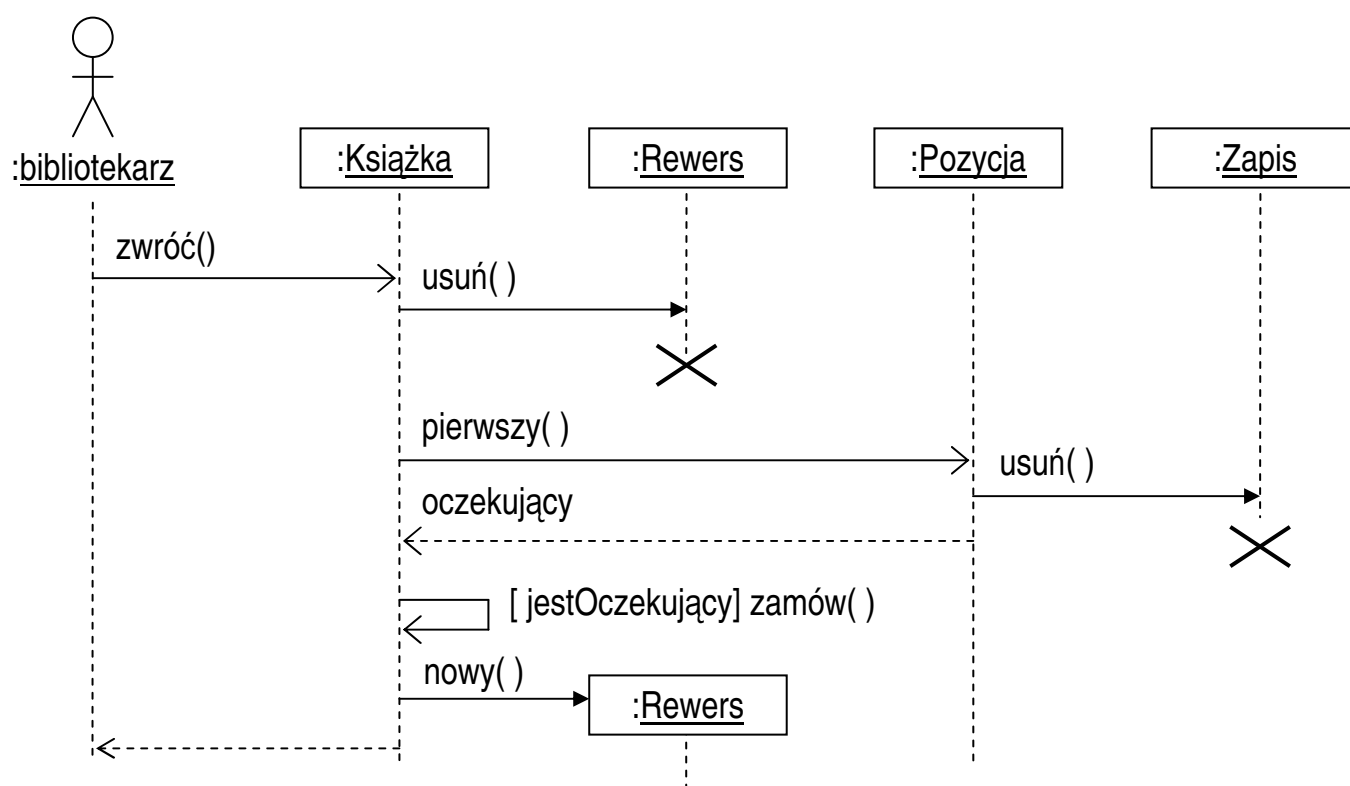
Pozycja	
Znajdź pasujące pozycje	Książka
Znajdź listę książek	Zapis
Ustal pierwszego oczekującego	





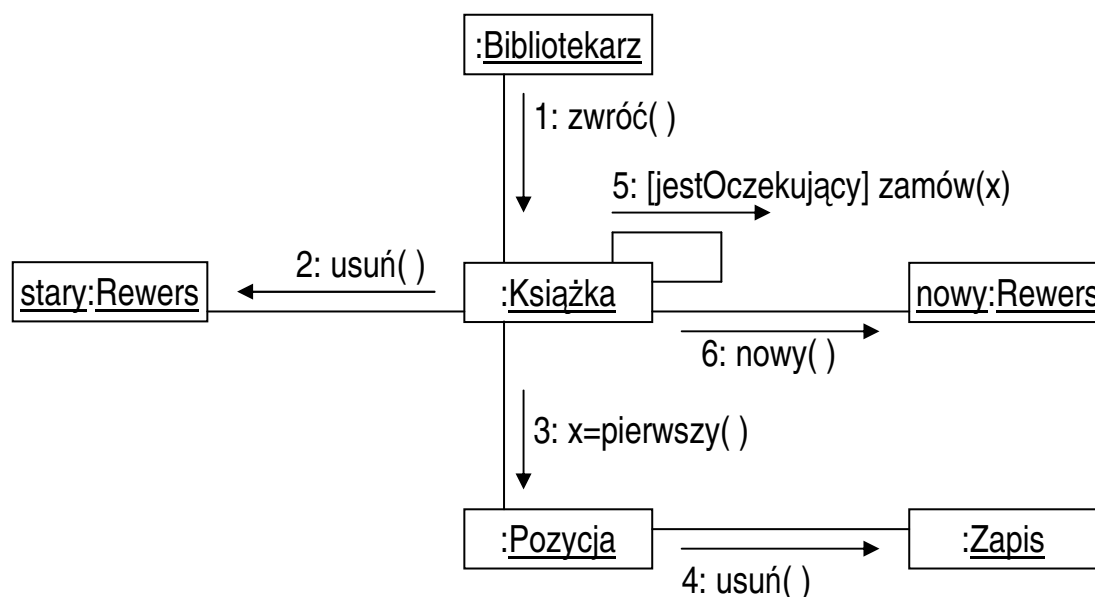
## Diagram sekwencji

- Model współdziałania obiektów
- Elementy modelu:
  - obiekty
  - komunikaty
- Reprezentacja graficzna



## Diagram współdziałania

- Model współdziałania obiektów.
- Elementy modelu:
  - obiekty
  - kanały komunikacji między obiektami
  - komunikaty
- Reprezentacja graficzna



- Obowiązkowa numeracja komunikatów:
  - chronologiczna
  - dziesiętna

## **Zastosowanie modeli współpracy obiektów**

1. Diagramy są narzędziem analizy i projektowania logicznego w fazie konstrukcji
  - wstępna koncepcja na kartach CRC
  - opis w postaci diagramu
2. Później diagramy pokazujące współpracę fizycznych komponentów systemu

## **Przykład: system biblioteczny**

### 1. Dane wejściowe:

- model przypadków użycia (scenariusze + reguły)
- model dziedziny (diagramy klas + stanu)

### 2. Problemy:

- jak określić strukturę bazy danych
- jak розміścić metody (algorytmy)

## **Wariant 1 — procedury transakcji użytkownika**

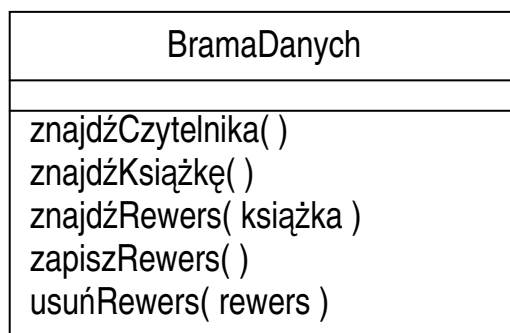
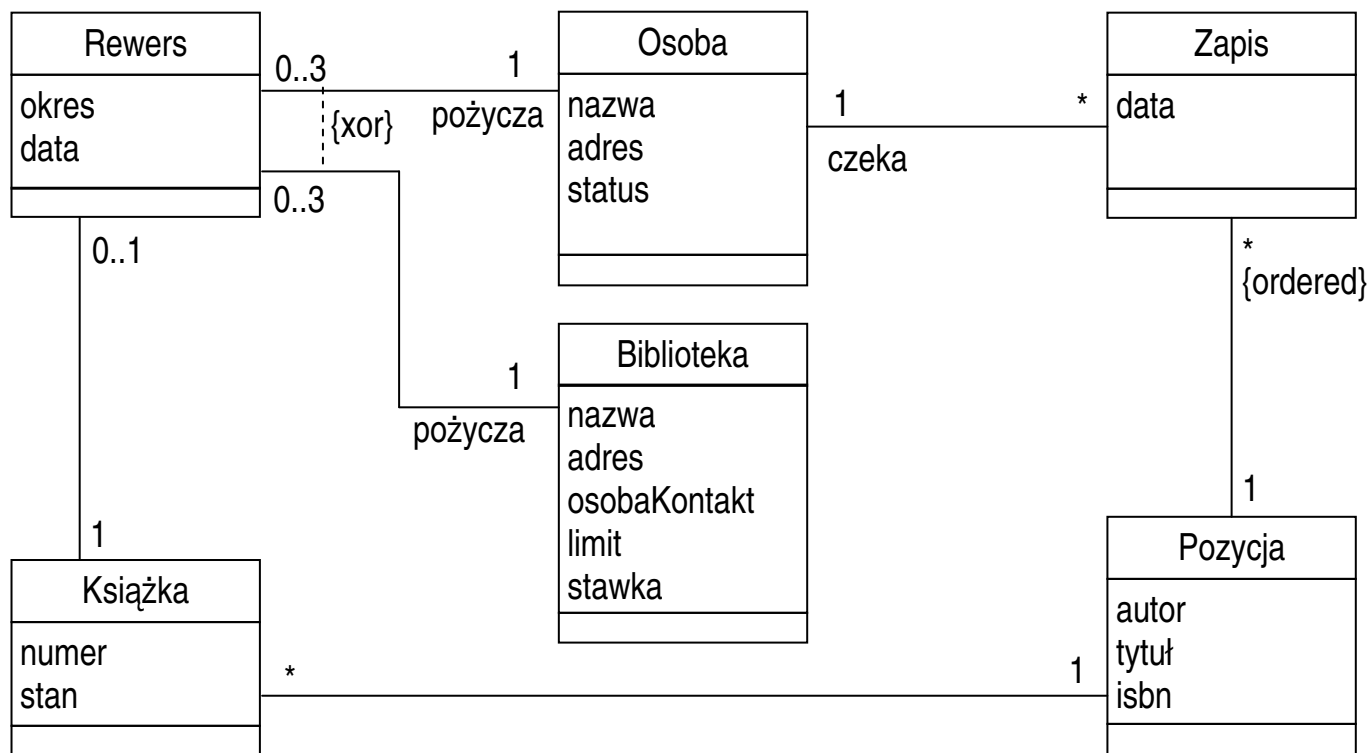
### **wypożycz( )**

identyfikujCzytelnika( )  
identyfikujKsiążkę( )  
obliczOkresWypożyczenia( książka, czytelnik )  
obliczOpłatę( książka, czytelnik )  
znajdźCzytelnika( )  
znajdźKsiążkę( )  
zapiszRewers( )

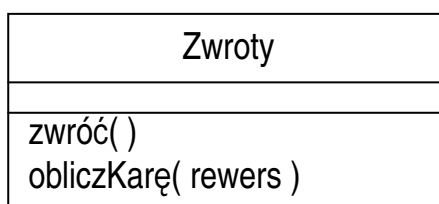
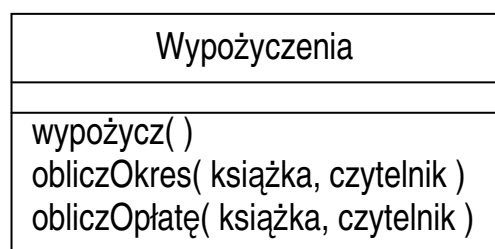
### **zwróć( )**

identyfikujKsiążkę( )  
obliczKaręZaZwłokę( rewers )  
znajdźRewers( książka )  
usuńRewers( rewers )

.....



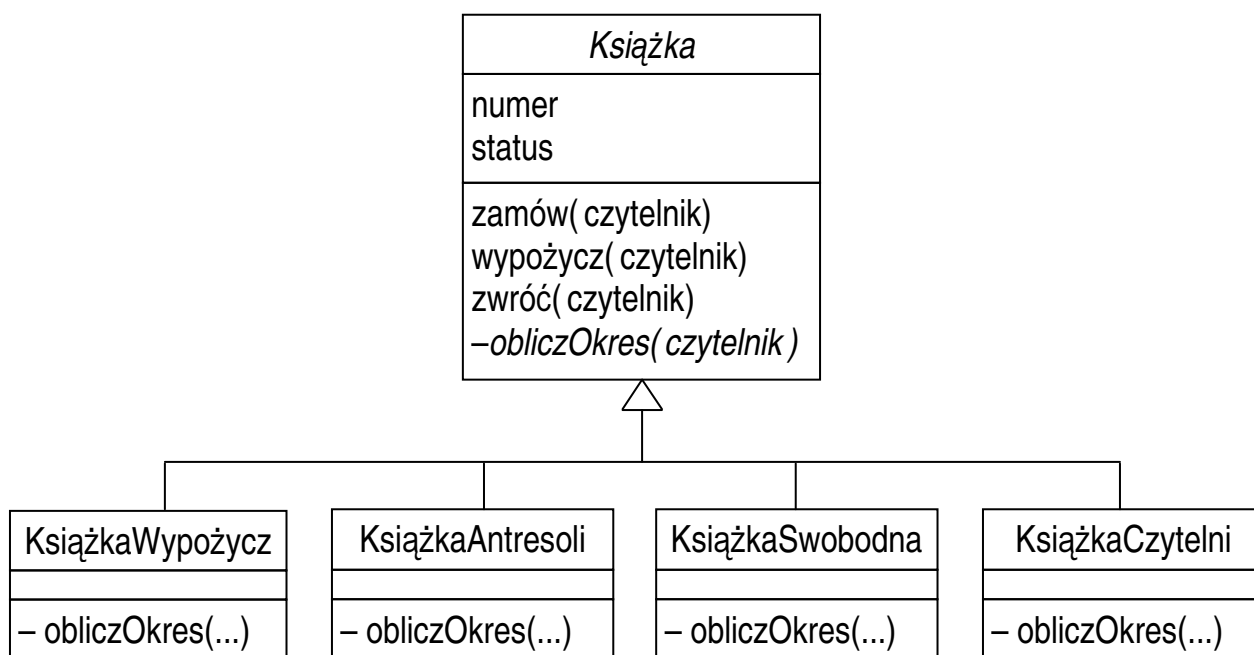
.....



.....

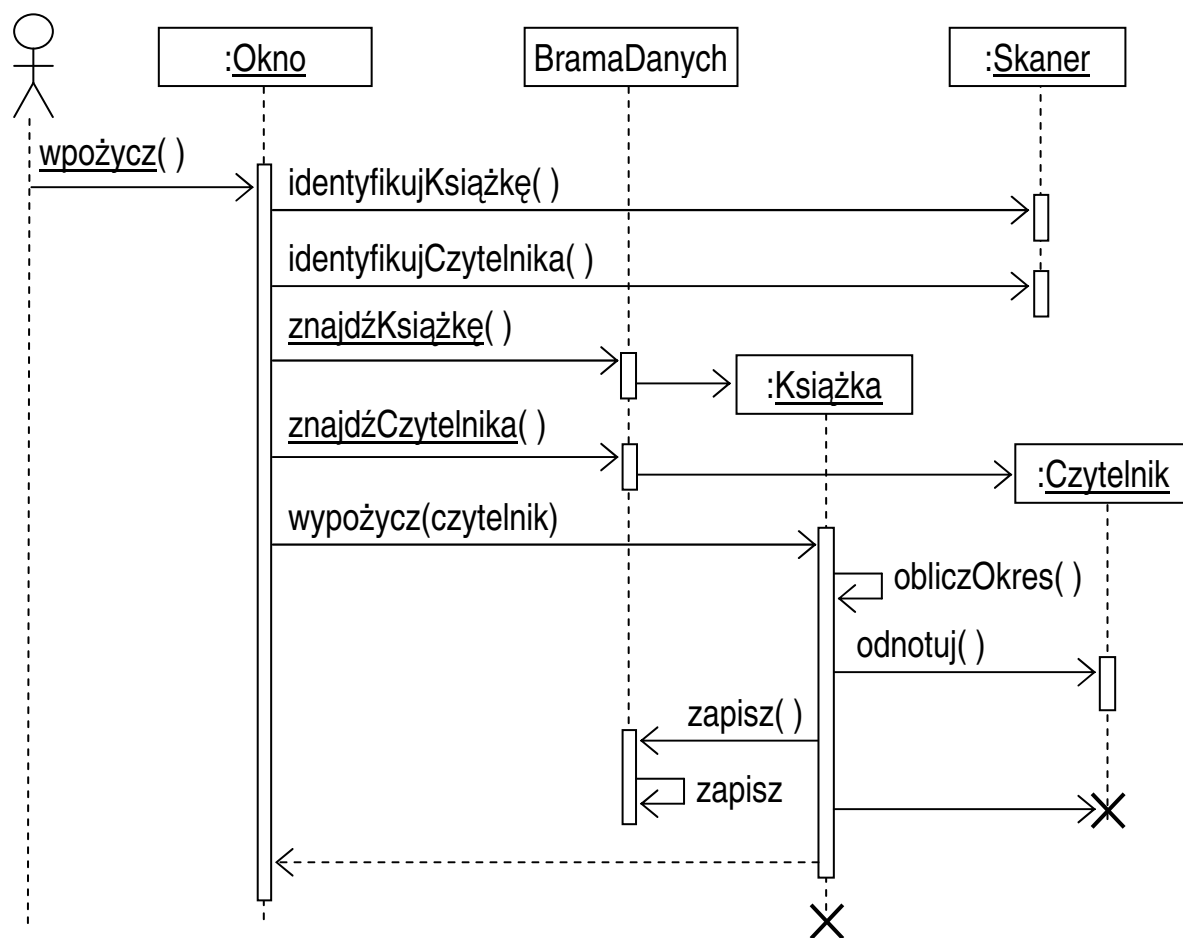


## Wariant 2 — reguły w klasach modelu dziedziny

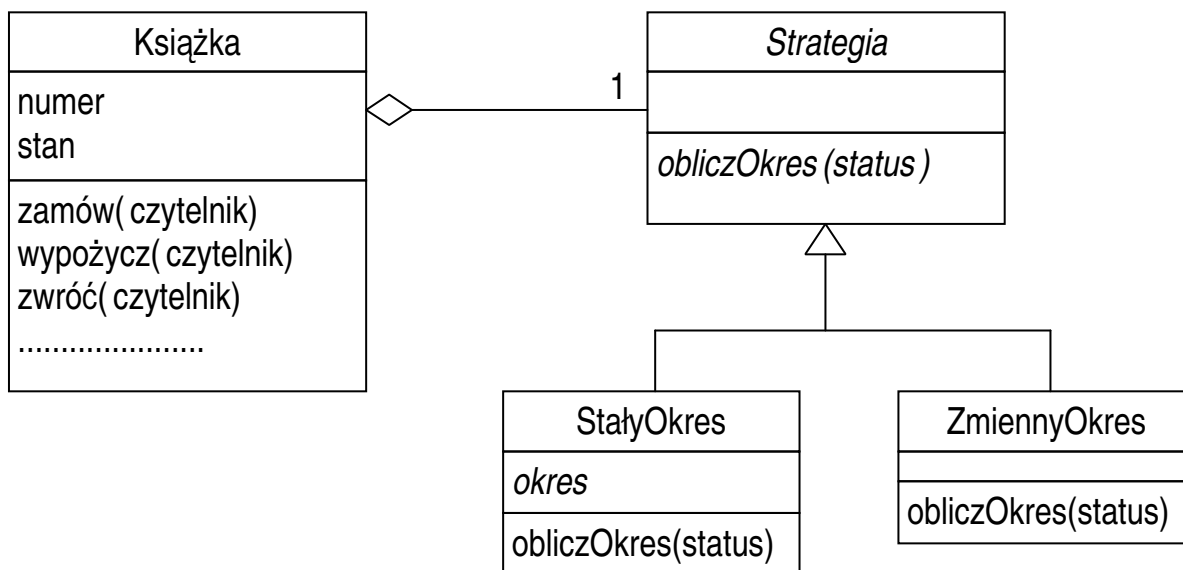


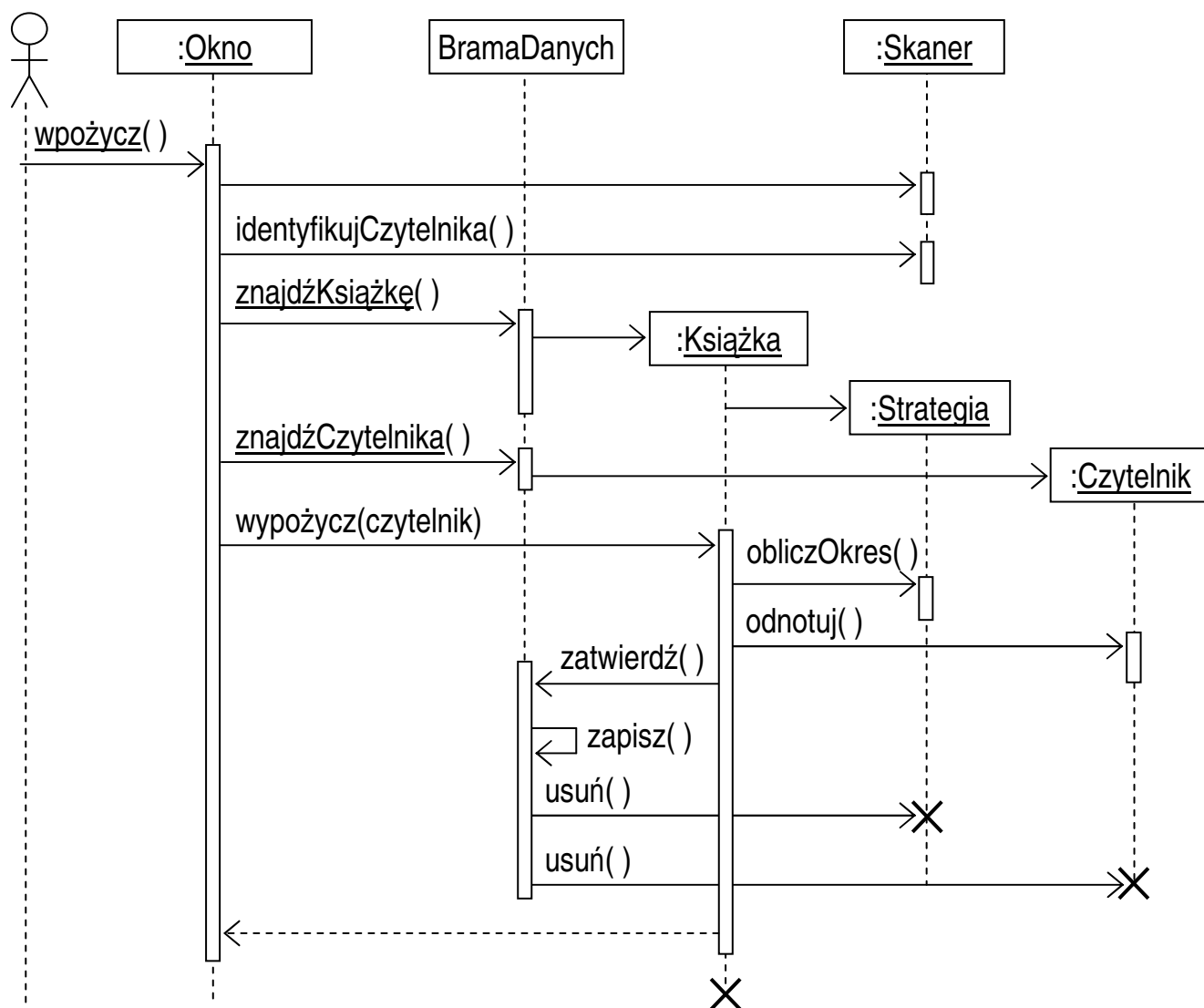
- Model danych jak w wariancie poprzednim
- Struktura klas odmienna od struktury danych trwałych





## Wariant 3 — hierarchia strategii





- Szkielet programów

```
class Strategia {  
    public virtual int obliczOkres(int status)=0;  
};
```

```
class StałyOkres : public Strategia {  
private int okres;  
public:  
    StałyOkres(int okres) {  
        this.okres=okres;  
    }  
    int obliczOkres(int status) {  
        return okres;  
    }  
};
```

```
class ZmiennyOkres : public Strategia {  
public:  
    ZmiennyOkres( );  
    int obliczOkres(int status) {  
        switch ( czytelnik.status ) {  
            case pracownik:  
                return 30;  
                break;  
            case doktorant:  
                return 7;  
                break;  
            default return 0;  
        }  
    }  
};
```

```
class Książka {
private:
    Pozycja *pozycja;
    int numer;
    int stan;
    Strategia *strategia;           // obiekt strategii
public:
    Książka(Pozycja *pozycja, int numer, Strategia *strategia) {
        this.pozycja=pozycja;
        this.numer=numer;
        this.strategia=strategia;
    }
    int Wypożycz (Czytelnik *czytelnik) {
        int okres;
        .....
        okres=strategia->obliczOkres(czytelnik->status);
        .....
    }
};
```

- Stworzenie nowej książki:

```
książka *ksWyp = new Książka ( pozycja, numer, StałyOkres(60) );
książka *ksAntr = new Książka ( pozycja, numer, StałyOkres(14) );
książka *ksSw = new Książka ( pozycja, numer, StałyOkres(7) );
książka *ksCzyt = new Książka ( pozycja, numer, ZmiennyOkres( ) );
```

## **Wzorce projektowe**

- *Composite*
- *Strategy*

E. Gamma, R. Helm, R. Johnson, J. Vlissides,  
Design Patterns

- *Transaction script*
- *Domain model*
- *Data gateway*
- *Data mapper*

M. Fowler i inni,  
Architektura Systemów zarządzania przedsiębiorstwem

## **Modelowanie fizycznej struktury aplikacji**

- Określenie struktury programów
  - podział na komponenty
  - określenie zależności komponentów
  - kontrola wersji
- Rozmieszczenie komponentów
  - przypisanie do serwerów i stacji roboczych
  - wykorzystanie technologii systemowych
  - ustalenie interfejsów
- Dokumentacja
  - komponenty, ich wersje i zależności
  - struktura systemu
  - przypisanie komponentów

## Pakiety

- Pojemnik zawierający dowolne byty
- Symbol graficzny

Nazwa
-------

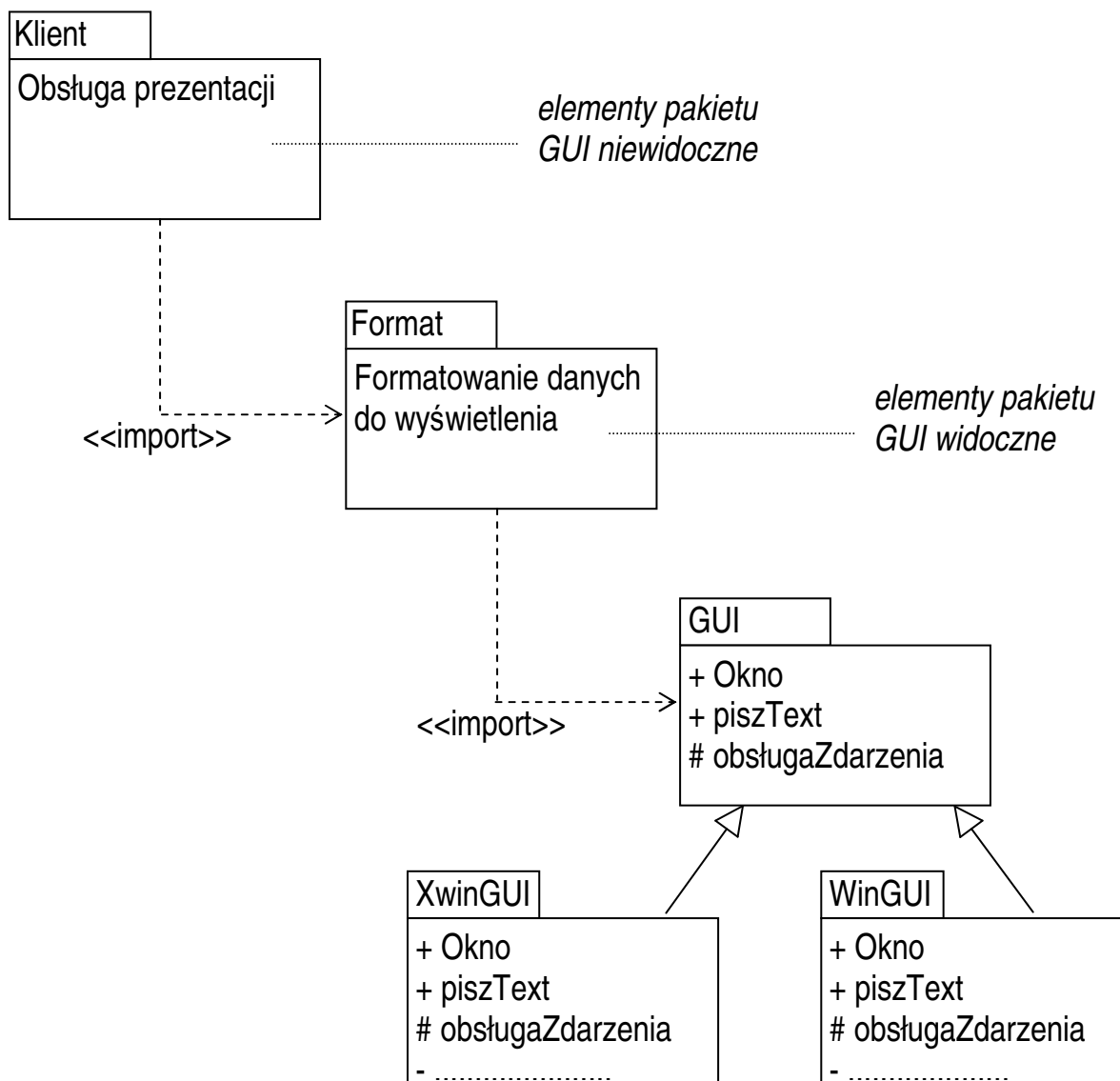
<ul style="list-style-type: none"><li>– opis tekstowy</li><li>– lista elementów</li><li>– diagramy</li></ul>
--

- Cechy
  - jest grupą innych bytów
  - określa przestrzeń nazw

*pakiet\_zewnetrzny::.....::pakiet\_wewnetrzny::element*
  - określa zakres widoczności



- Import pakietu



## **Pakiety klas**

- Strukturalizacja kodu
- Kryterium grupowania klas w pakiety:
  - liczba asocjacji?
  - hierarchia dziedziczenia?
  - wspólny interfejs?
  - minimalizacja zależności?
- Zalecenie:  
oddzielać interfejs pakietu od realizacji (np. warstwy)
- Diagramy, na których mogą wystąpić pakiety:
  - diagram współdziałania
  - diagram zależności

## Zastosowanie pakietów

### 1. Zarządzanie projektem

- heterogeniczne pakiety artefaktów części systemu, np. przypadki użycia, klasy, diagramy stanu.

### 2. Dekompozycja systemu

- jednorodne pakiety (klas) opisujące strukturę systemu na wyższym poziomie abstrakcji

Podział na pakiety może poprawić modyfikowalność

### Dodatkowe elementy opisu pakietów

- stereotypy:

<<system>>	domyślny pakiet obejmujący cały system
<<subsystem>>	niezależna część systemu
<<facade>>	pakiet opisujący interfejs innego pakietu
<<stub>>	reprezentant (symulator) innego pakietu
<<framework>>	pakiet parametryzowanych wzorców

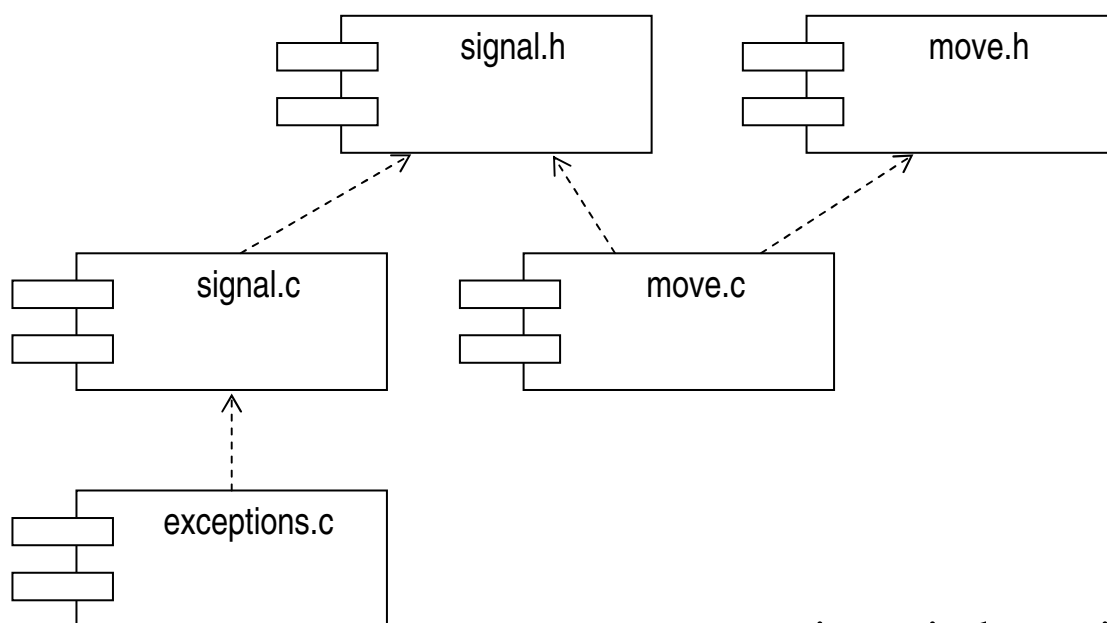
## Diagramy implementacyjne

Diagram komponentów

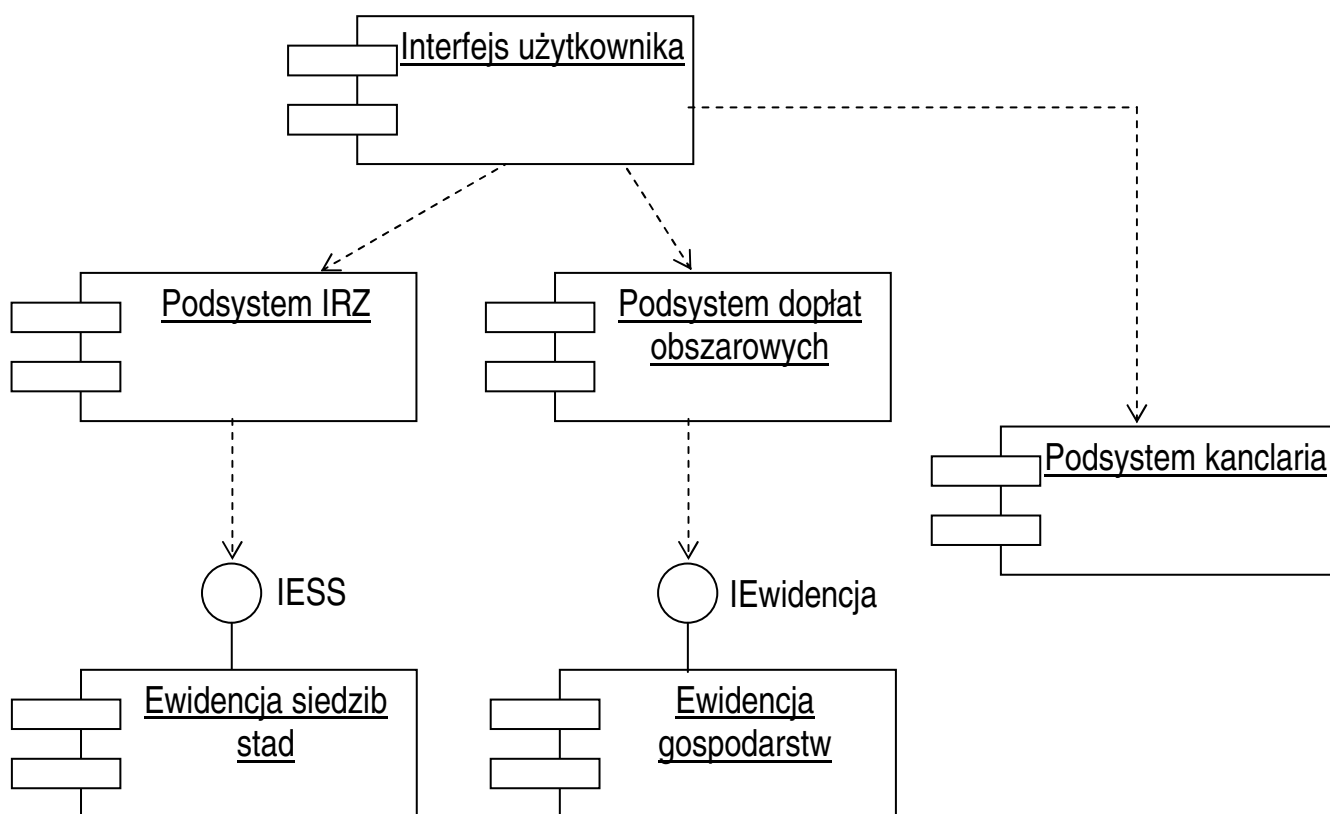
Diagram rozmieszczenia

### Diagram komponentów

- Model zależności komponentów oprogramowania
- Elementy modelu:
  - komponenty  
(np. pliki źródłowe, wykonalne, biblioteki, tabele)
  - zależności
- Reprezentacja graficzna



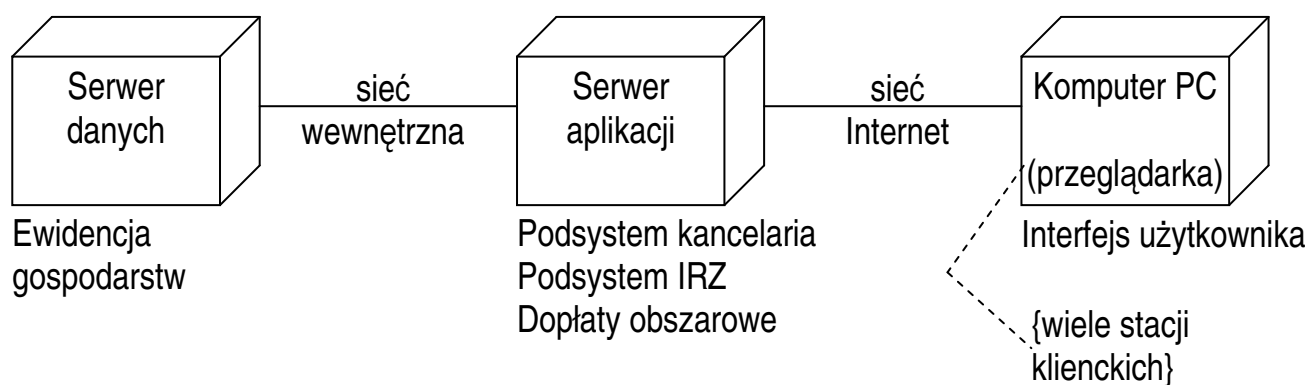
– powiązania kompilacyjne



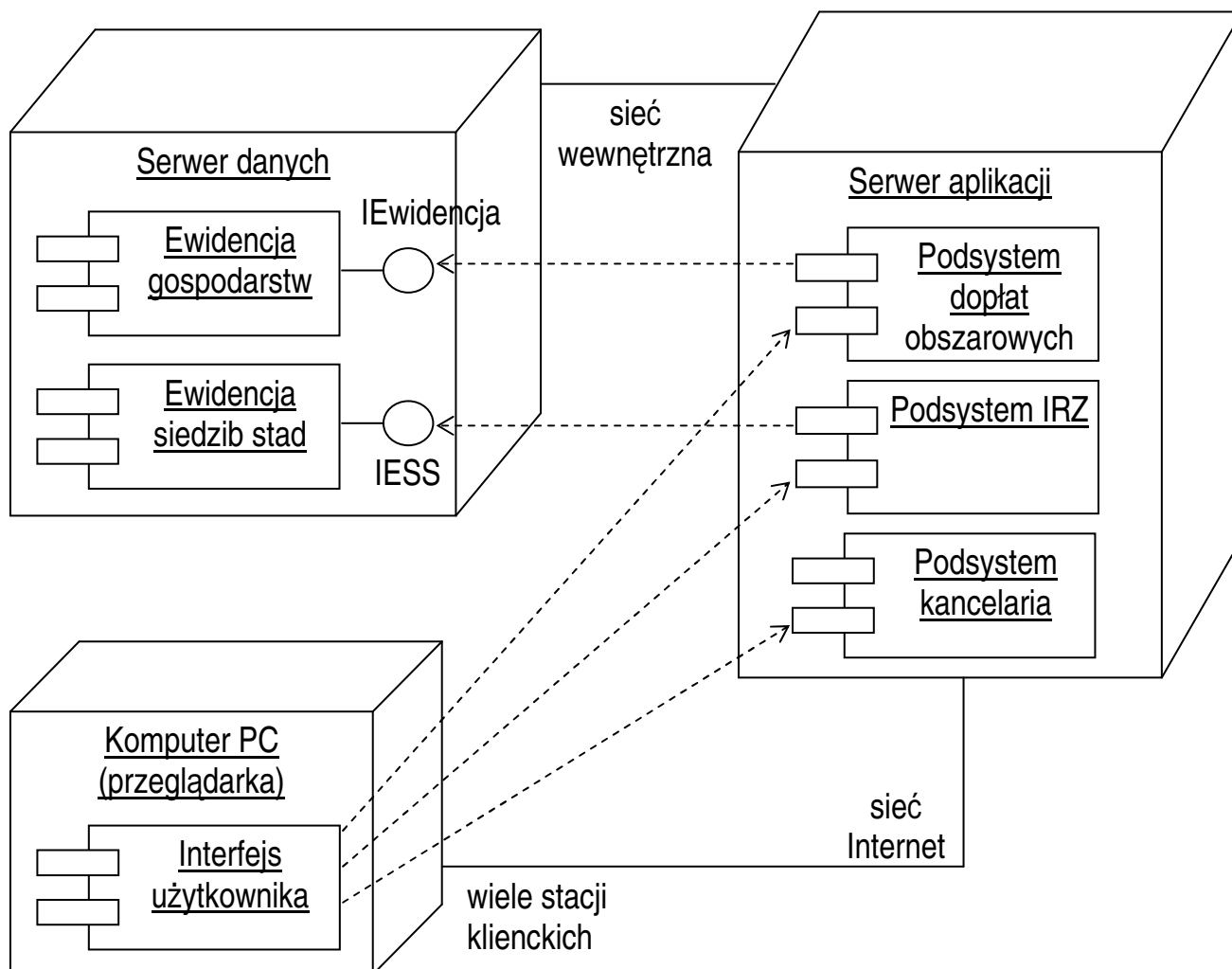
– związki współpracy komponentów.

## Diagram rozmieszczenia (*deployment diagram*)

- Model fizycznej struktury systemu
- Elementy modelu
  - węzły (komputery lub urządzenia pomocnicze)
  - połączenia między węzłami (sieci lub inne sprzęgi)
  - przypisanie komponentów do węzłów
- Reprezentacja graficzna



- Wariant: połączenie diagramów implementacyjnych



## ***Zastosowanie modeli implementacyjnych***

1. Pierwsze wersje diagramów rozmieszczenia powstają w fazie rozwinięcia (koncepcja systemu)
2. Diagramy komponentów po zakończeniu projektowania w fazie konstrukcji
3. Wersja ostateczna – po implementacji



## Podsumowanie: przebieg projektu

- Proces RUP określa ogólny schemat przebiegu projektu
- Język UML definiuje notację i zestaw modeli

### 1. Analiza wymagań

#### Działania:

- poznanie dziedziny aplikacji i wymagań użytkownika
- klasyfikowanie i dokumentowanie wymagań
- budowa prototypu interfejsu użytkownika

#### Wyniki:

- model przypadków użycia
- prototyp interfejsu użytkownika
- określenie sprzęgów zewnętrznych
- określenie wymagań нефunkcjonalnych

---

*Faza inicjacji*

*Faza rozwinięcia*

*Faza konstrukcji*

## 2. Modelowanie dziedziny problemu

### Działania:

- analiza dziedziny aplikacji i wymagań użytkownika
- analiza scenariuszy przypadków użycia
- wykorzystanie pojęć opisywanych w literaturze
- językowa analiza opisu

### Wyniki:

- diagramy klas (model pojęciowy)
  - diagramy stanu
- 

*Faza inicjacji*

*Faza rozwinięcia*

*Faza konstrukcji*

### 3. Określenie logicznej struktury aplikacji

#### Działania:

- rozwijanie diagramu klas → wzorce projektowe
- określenie modelu danych
- określenie realizacji interfejsu użytkownika
- budowa modelu zachowania

#### Wyniki:

- diagramy klas
  - diagramy współpracy
- 

#### *Faza konstrukcji*

## 4. Projektowanie programu

### Działania:

- dostosowanie modelu klas
- projektowanie bazy danych
- określenie struktury programów
- projektowanie interfejsu (GUI)
- refaktoryzacja

### Wyniki:

- diagramy klas (model projektowy)
  - diagramy komponentów i rozmieszczenia
- 

*Faza konstrukcji*

## 5. Implementacja programu

### Działania:

- definiowanie klas programu
- implementacja bazy danych
- opracowanie podręczników
- wykorzystanie narzędzi CASE

### Wyniki:

- kod programu
  - dokumentacja techniczna
  - podręczniki użytkownika
- 

*Faza konstrukcji*

## 6. Integracja i testowanie

### Działania:

- zaplanowanie testowania akceptacyjnego
- zaplanowanie testowania integracyjnego
- integracja i testowanie → usuwanie błędów
- strojenie systemu
- testowanie akceptacyjne → odbiór

### Wyniki:

- działający system
  - raporty testowania
  - poprawiona dokumentacja
- 

*Faza inicjacji*

*Faza rozwinięcia*

*Faza konstrukcji*