

## Wprowadzenie

*Inżynieria oprogramowania zajmuje się metodami wytwarzania oprogramowania na skalę przemysłową*

określenie wymagań	15%
projektowanie programów	25%
programowanie	20%
integracja i testowanie	40%
odbiór (zatwierdzanie)	

- Oprogramowanie
- Wielki projekt programistyczny
- Metodyka
- Cel inżynierii oprogramowania
- Główny problem

---

### ***Stan bieżący (Chaos report, Standish Group)***

<b>Rok</b>	<b>Sukces</b>	<b>Przekroczenie</b>	<b>Porażka</b>
1995	16%	53%	31%
2003	34%	51%	15%

## **Procesy projektowe, metody i narzędzia**

- Procesy projektowe
  - Metody
    - obiektowe
    - strukturalne
    - metodyki lekkie
  - Narzędzia
    - systemy *upper CASE*
    - systemy *lower CASE*
    - środowiska wspomagające testowanie
- 

## **Treść wykładu**

1. Metody obiektowe (50% wykładu + laboratorium).
  2. Metody strukturalne (20%).
  3. Organizacja i zarządzanie projektem (30%).
- 

## **Literatura**

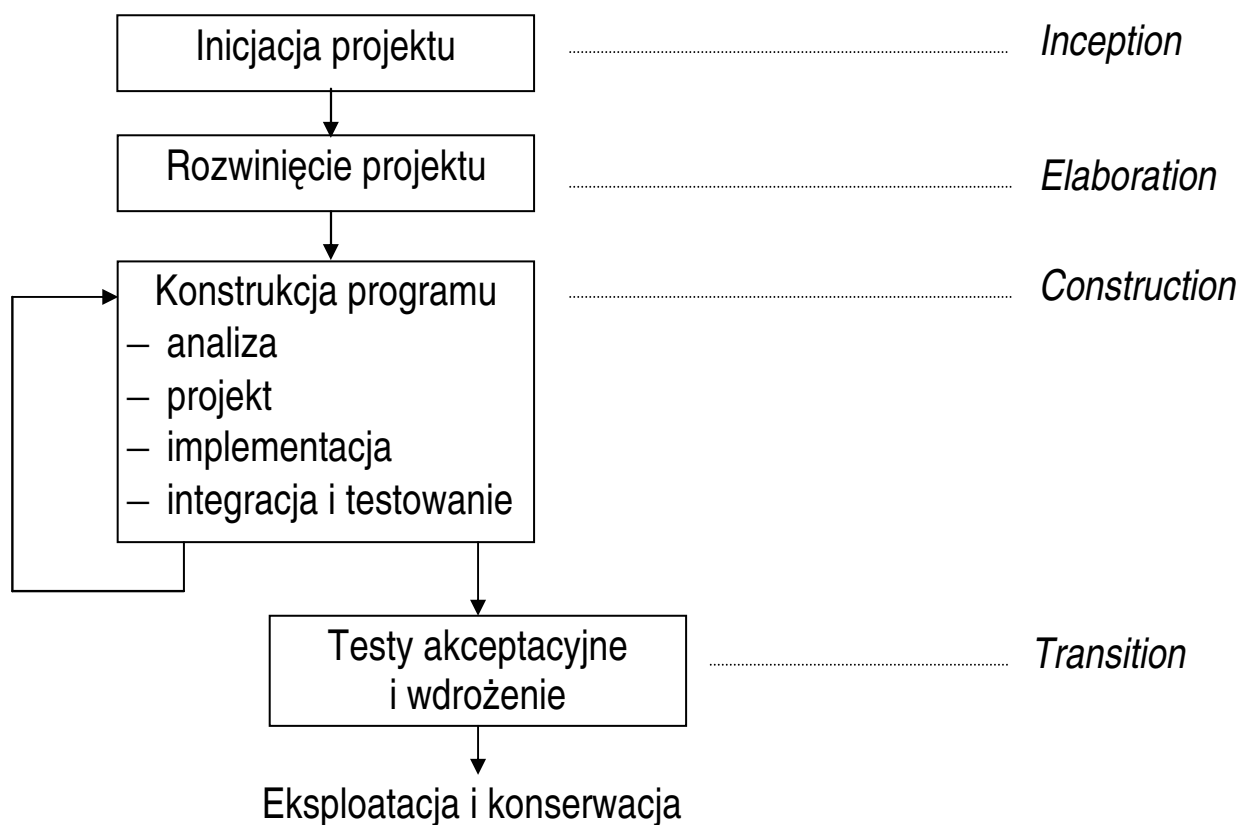
Fowler M., Scot K.: UML w kropelce; LTP 2002, 2005

Jaskiewicz A.: Inżynieria oprogramowania, Helion, Gliwice 1997

## Metody obiektowe

- Dekompozycja obiektowa
- Hierarchiczna klasyfikacja obiektów
- Zachowanie systemu jako wynik interakcji obiektów

### ***Proces projektowy RUP (Rational Unified Process)***



## **Modele obiektowe**

- Model przypadków użycia
- Diagramy aktywności
- Diagramy klas
- Diagramy stanu
- Diagramy współdziałania
- Diagramy implementacyjne

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

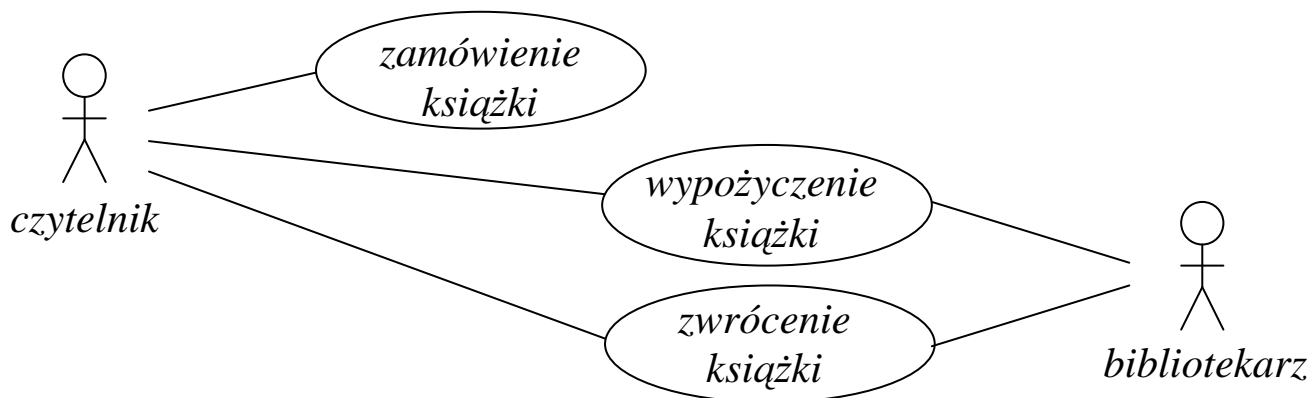
- Modelowanie procedur biznesowych
- Identyfikacja uczestników
- Punkt widzenia użytkowników

### ***Działania wstępne***

- Rozpoznanie dziedziny zastosowania
  - źródła danych
  - wywiady z użytkownikami
  - procesy biznesowe
  - obszary tematyczne
- Wstępna analiza obszarów tematycznych
  - identyfikacja zadań i danych
  - charakterystyki ilościowe
  - model przepływu zadań
  - wizja obszaru

## **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.

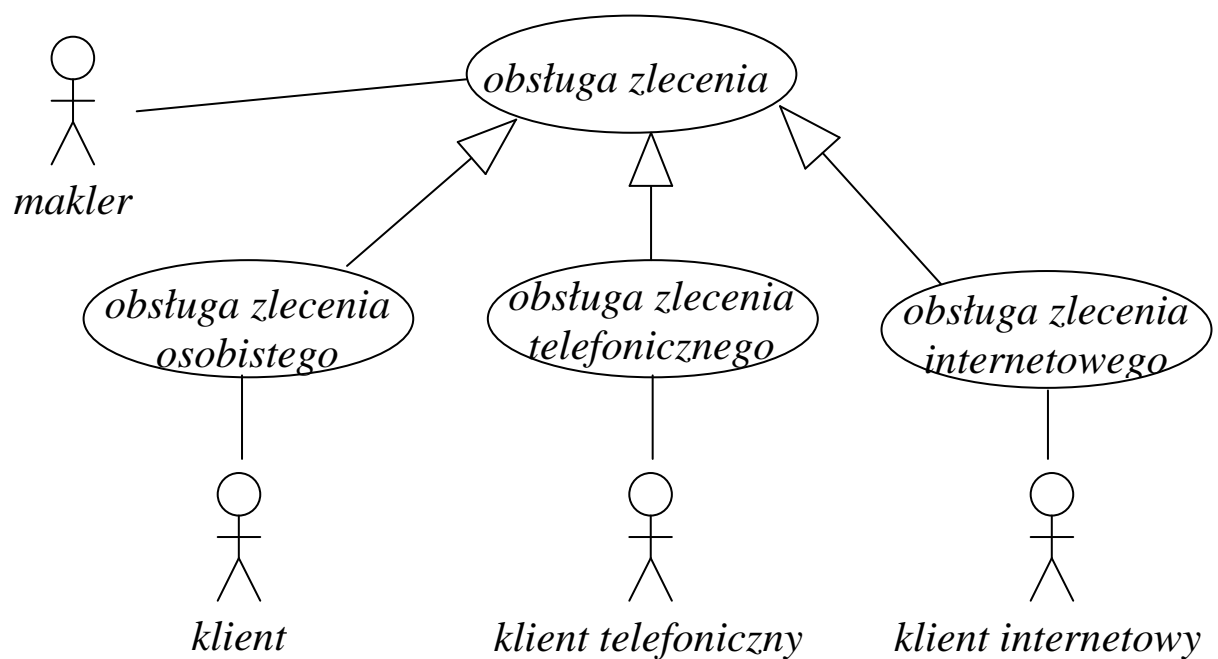
- **Scenariusz alternatywny 2 (brak wolnej książki)**

- 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

.....

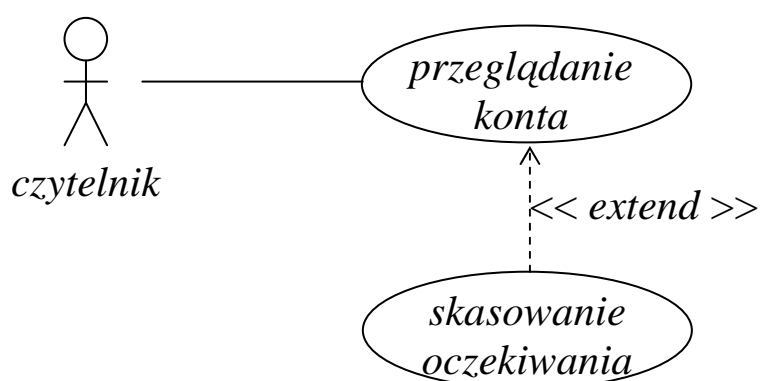
obsługa zlecenia osobistego

.....

## 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

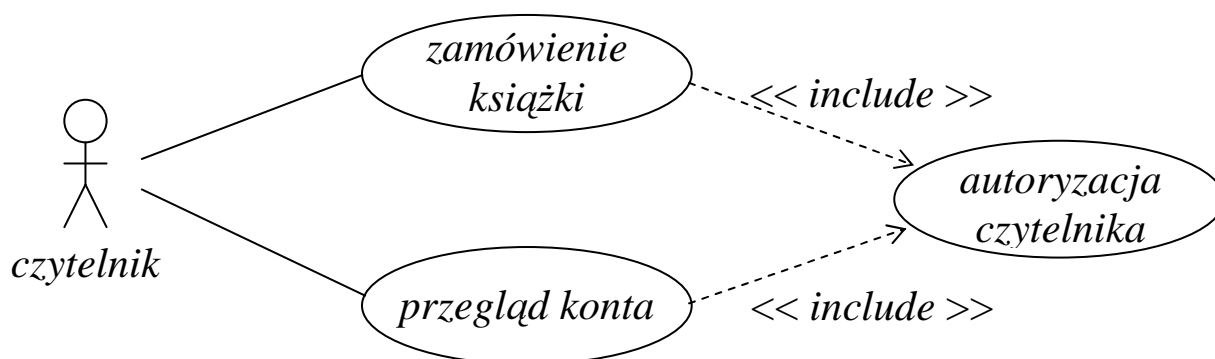
wstawić w punkcie rozszerzenia *kasowanie*:

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

- dostęp internetowy
- zamówienie wolnej książki przez Internet
- kolejka do zwolnienia pozycji

- **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

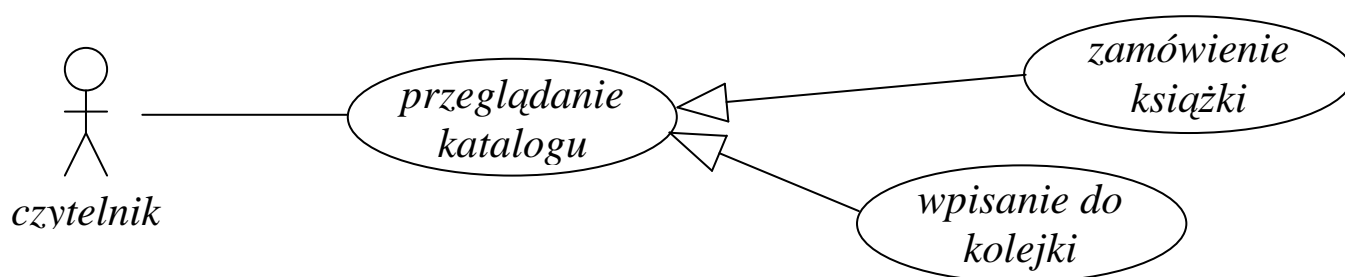
1. System prezentuje ekran wyszukiwania.
2. Czytelnik wprowadza dane bibliograficzne.
3. System przeszukuje katalog i wyświetla listę pozycji (tytułów).
4. Czytelnik przegląda listę i wybiera pozycję.
5. System wyświetla listę książek.

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

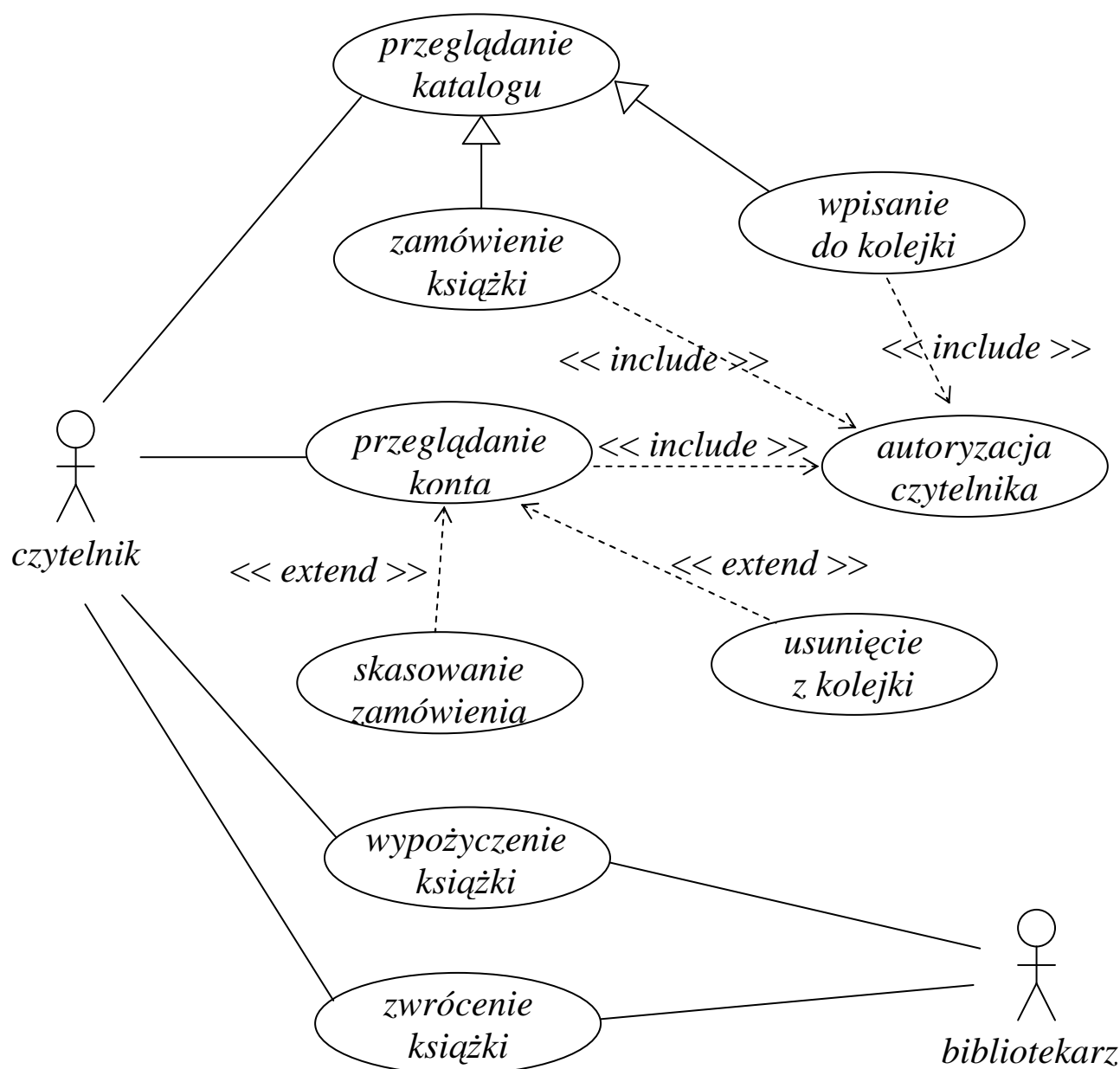
1. Identyfikacja karty czytelnika.
2. Identyfikacja książki.
3. System usuwa zamówienie.
4. System zapisuje wypożyczenie.

### zwrócenie książki

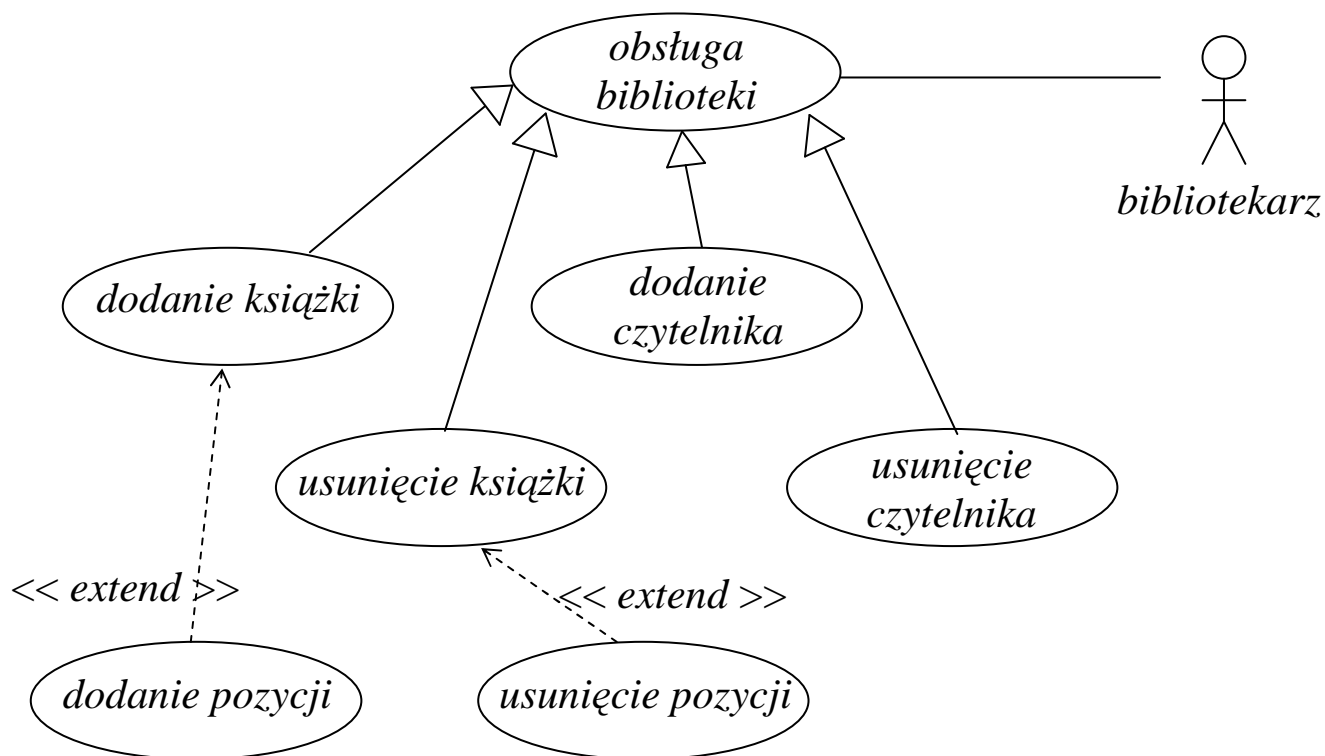
1. Identyfikacja książki.
2. System usuwa wypożyczenie.



## Model z punktu widzenia czytelnika



## Model z punktu widzenia utrzymania biblioteki



## **Podsumowanie: metoda przypadków użycia**

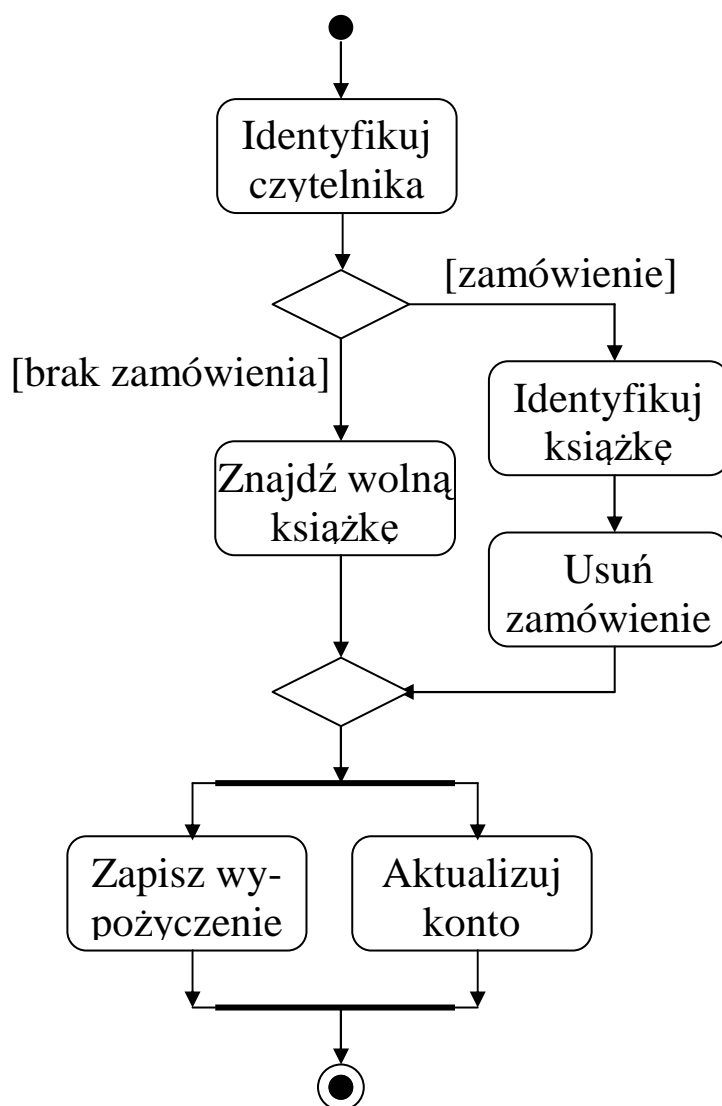
1. Identyfikacja aktorów
2. Określenie przypadków użycia
3. Uporządkowanie modelu
4. Dokumentacja modelu
5. Budowa modelu dziedziny

### Uwagi:

- Scenariuszom towarzyszy prototyp
- Przypadki użycia nie dotyczą problemów projektowych
- Dalsze zastosowania modelu

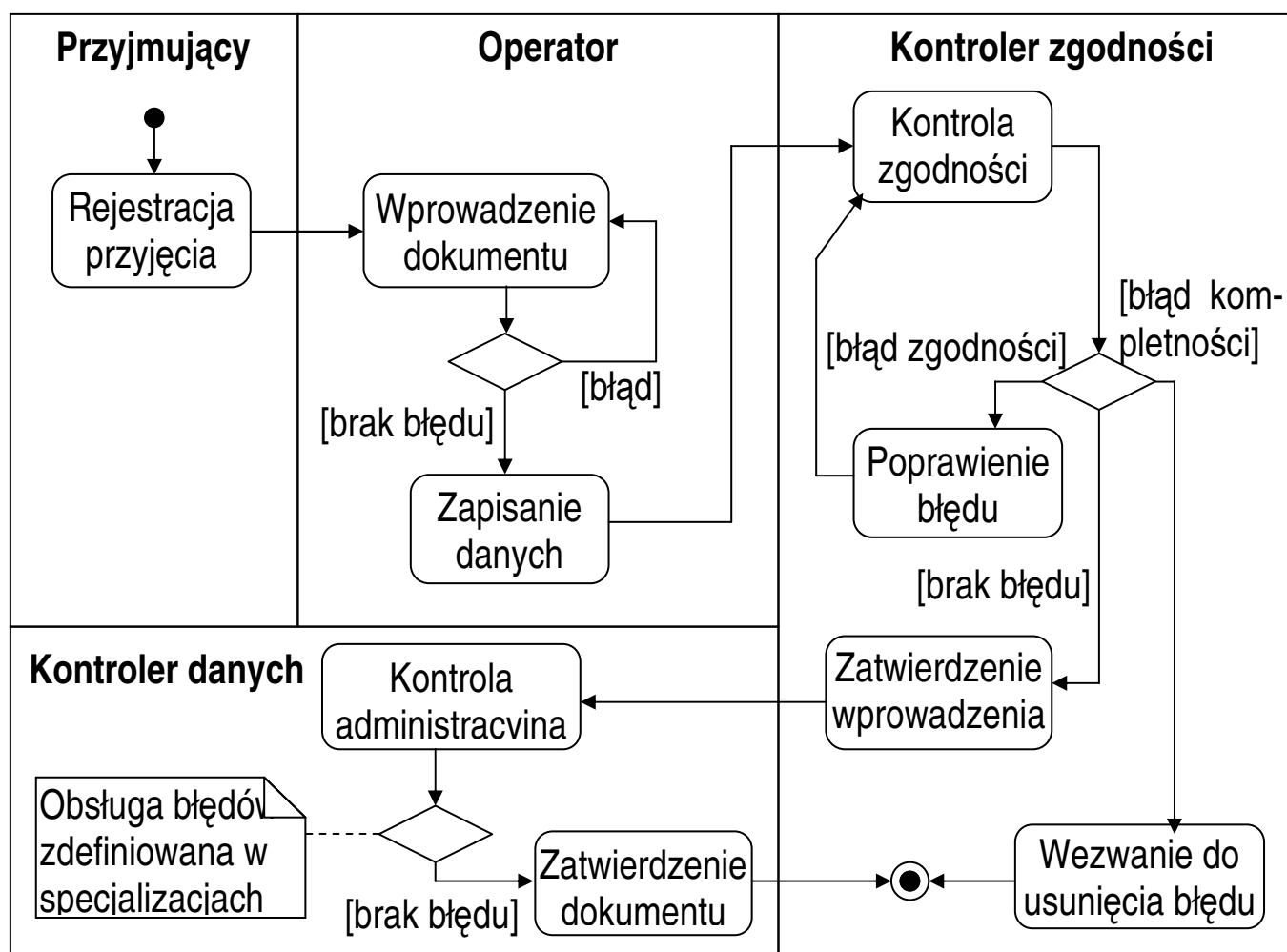
## Diagram aktywności (*activity diagram*)

- Model przepływu sterowania
- Rozszerzona sieć działań



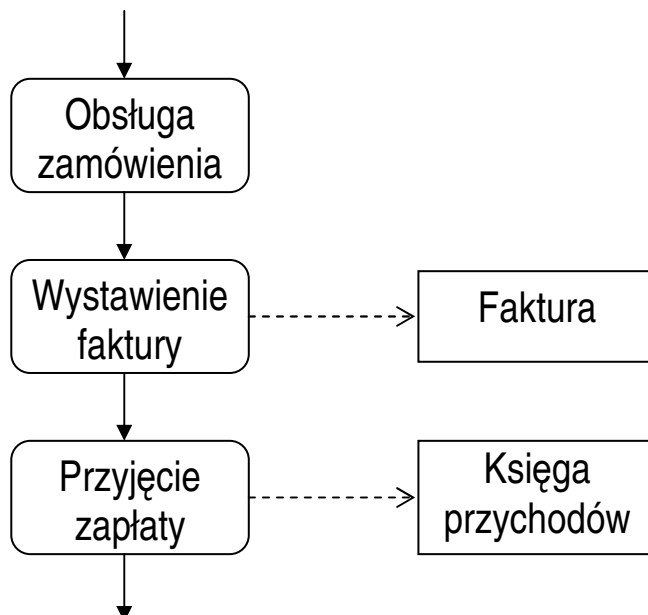
## Wskazanie odpowiedzialności

- Notki
- Boksy (pasy)



## **Wskazanie obiektu czynności**

- Relacja zależności

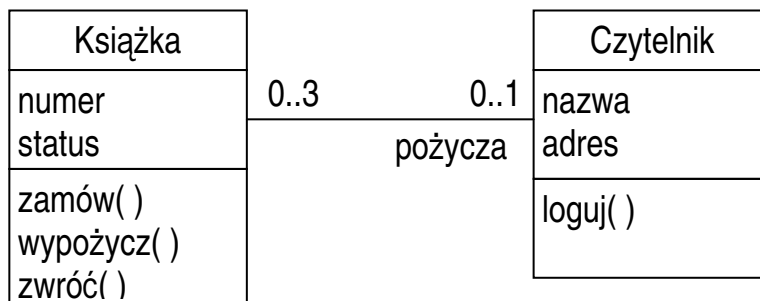


## **Zastosowanie diagramów aktywności**

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

## Diagram klas

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

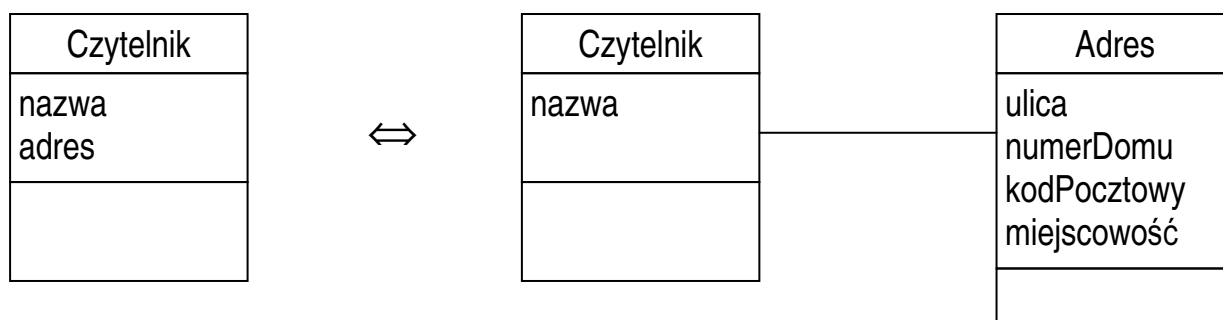




## Perspektywy widzenia modelu

- **Model pojęciowy**

- model analizy
- pokazuje klasy i atrybuty, asocjacje i generalizacje
- brak szczegółów implementacyjnych

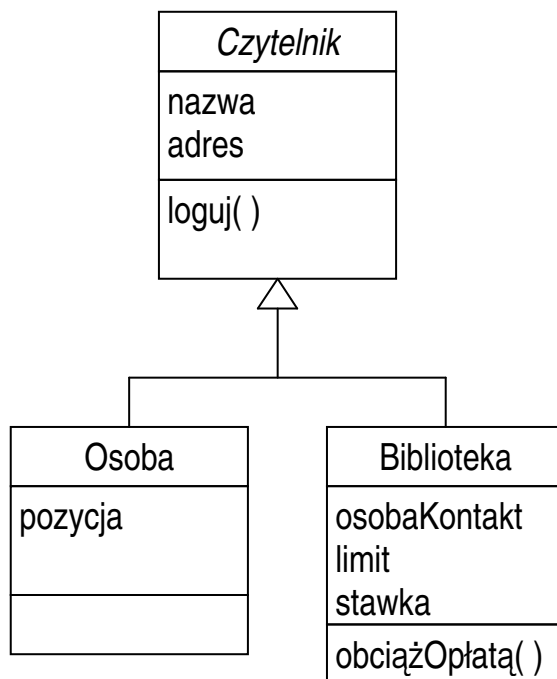


- **Model projektowy (programistyczny)**

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

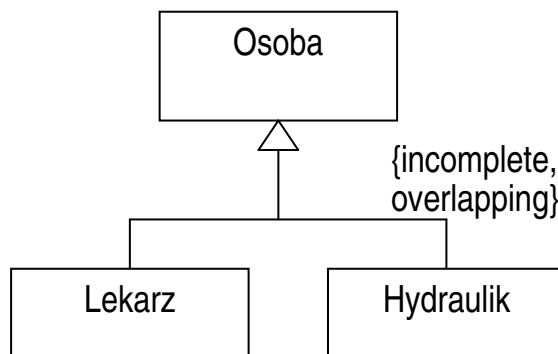
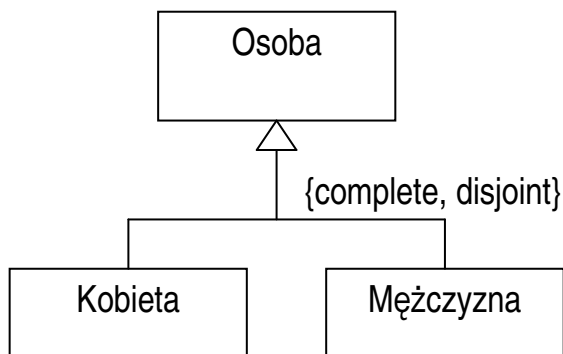
## Uogólnienie

Klasyfikacja, modele na różnych poziomach abstrakcji



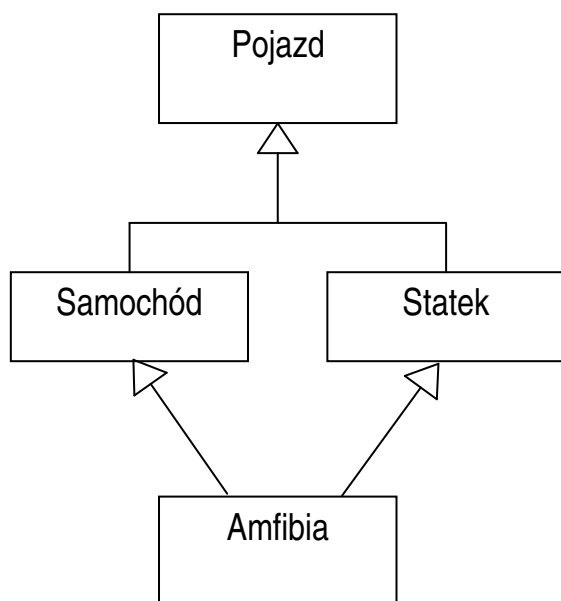
- **Model pojęciowy**
  - klasyfikacja bytów
- **Model projektowy**
  - hierarchia interfejsów o dziedziczenia

- Ograniczenia klasyfikacji



domyślnie: incomplete, disjoint

?? Spójność diagramu



## **Atrybuty**

- **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

## Operacje

- **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:

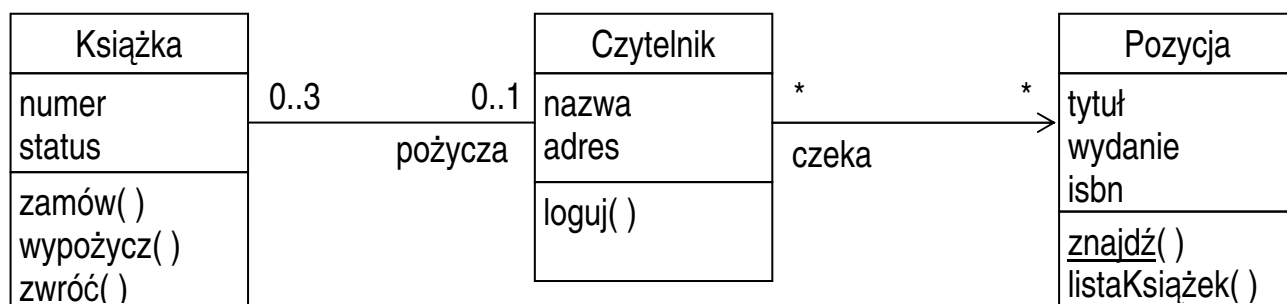
- operacje klasowe: podkreślenie
- operacje abstrakcyjne: *kursywa*

## Asocjacja

Trwałe powiązania między obiektami, niekoniecznie typu 1–1

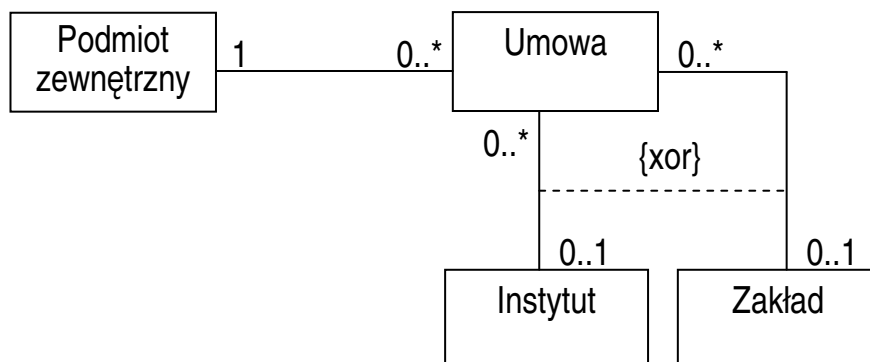
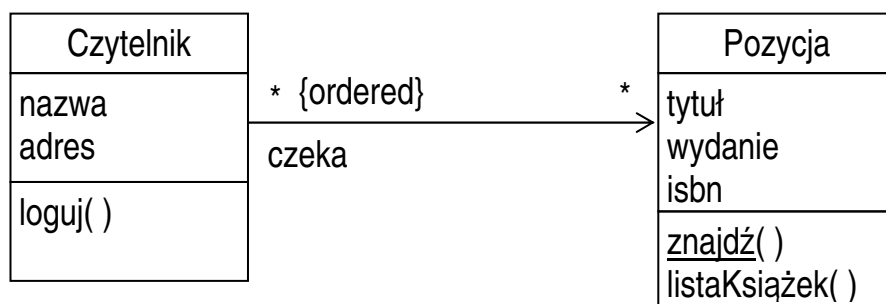
Opis relacji:

- krotność
- kierunek
- etykieta objaśniająca rolę



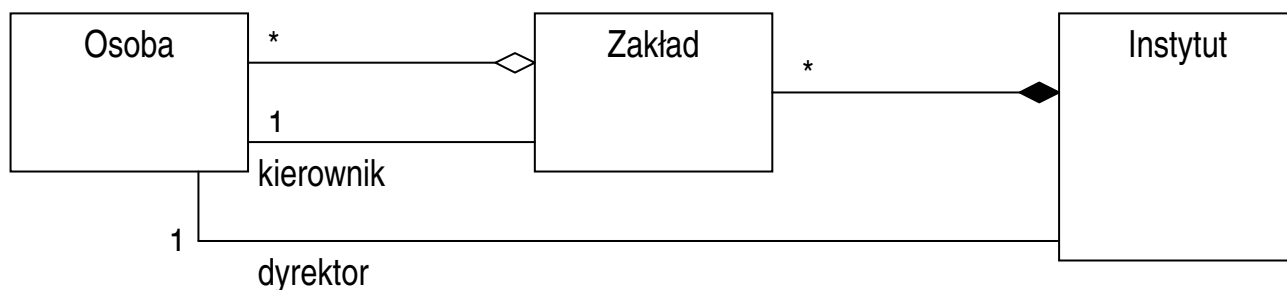
- **Model pojęciowy**
  - związek pojęciowy między klasami
- **Model projektowy**
  - operacje nawigowania między obiektami
  - operacje umożliwiające tworzenie relacji
  - pola klas wskazujące powiązania

- Ograniczenia asocjacji

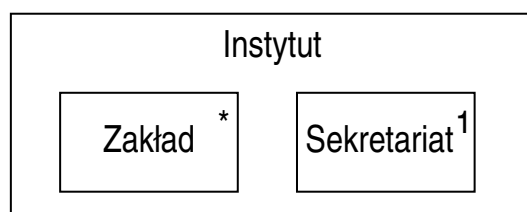


## Agregacja

Szczególny przypadek asocjacji: związek typu całość – część



Złożenie lub zawieranie (*composition*)

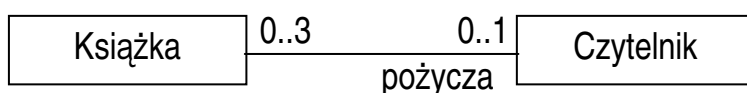




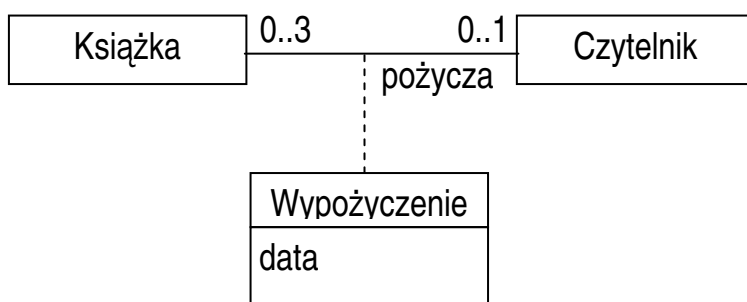
## Klasy asocjacyjne

- 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.

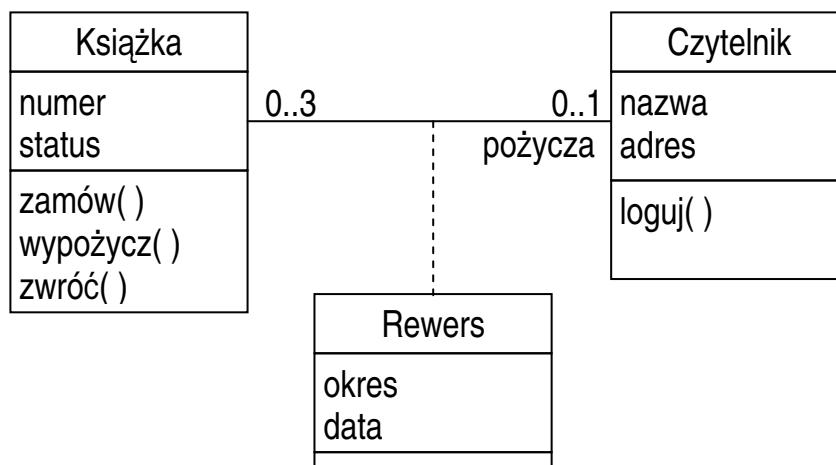
**Asocjacja nie ma atrybutów.**



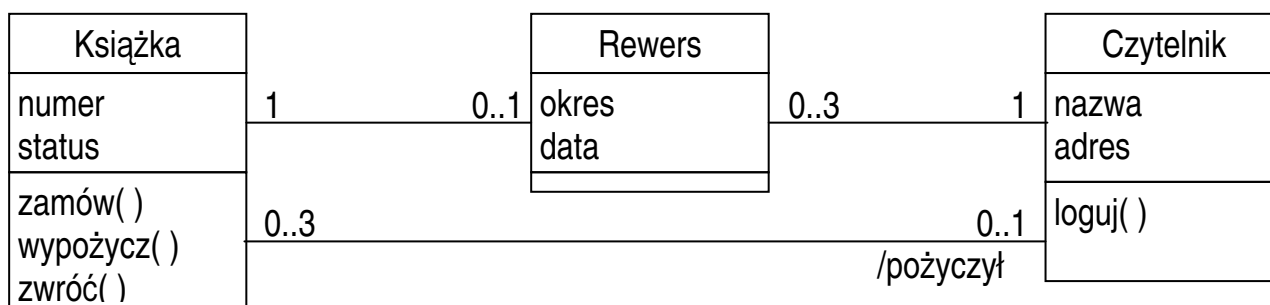
→ do kogo należy atrybut **data wypożyczenia** ?



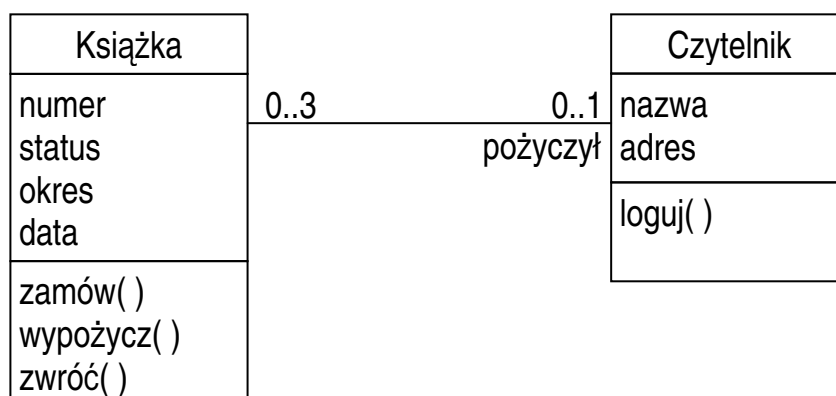
- Klasa asocjacyjna**



- Wariant alternatywny — zwykła klasa.**



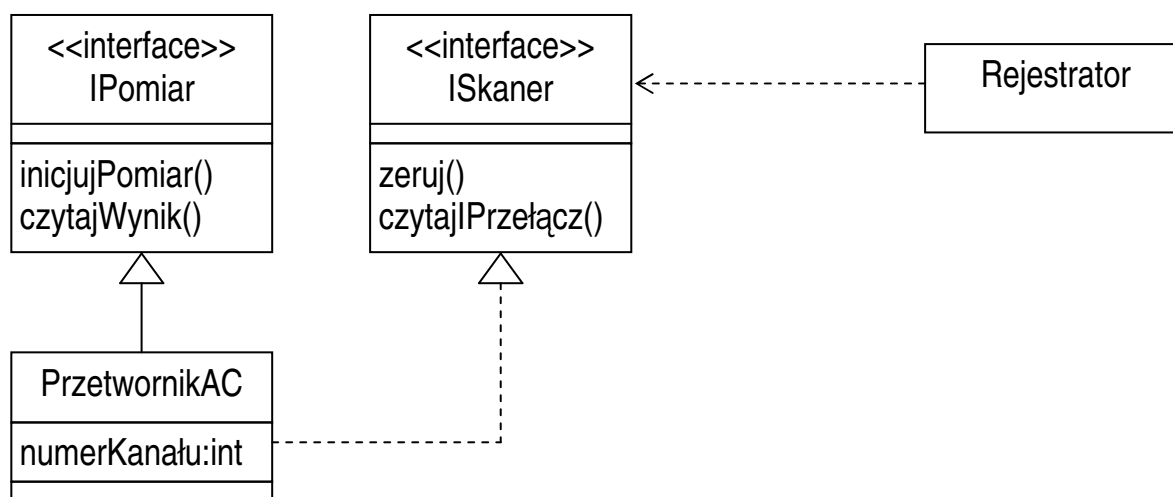
- Trzeci wariant — dodatkowe atrybuty**



## Interfejsy

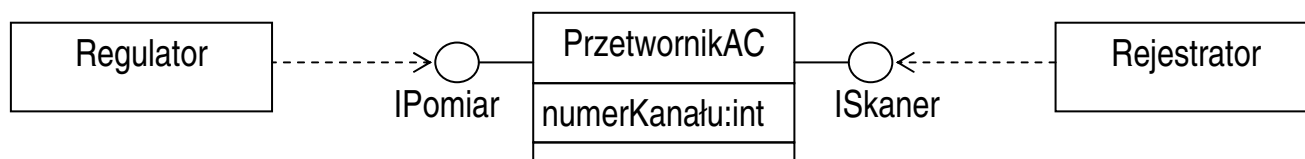
Klasa definiująca operacje udostępniane innym obiektom

- 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**
  - zmiana interfejsu może wymagać zmiany w kodzie klienta
  - zmiana metody wykonania operacji nie wpływa na klienta

Notacja alternatywna (bez pokazania operacji):



### Pojęcia podobne

- klasa abstrakcyjna {abstract}
- typ <<type>>

## **Zastosowanie diagramów klas**

### **1. Model dziedziny (model pojęciowy)**

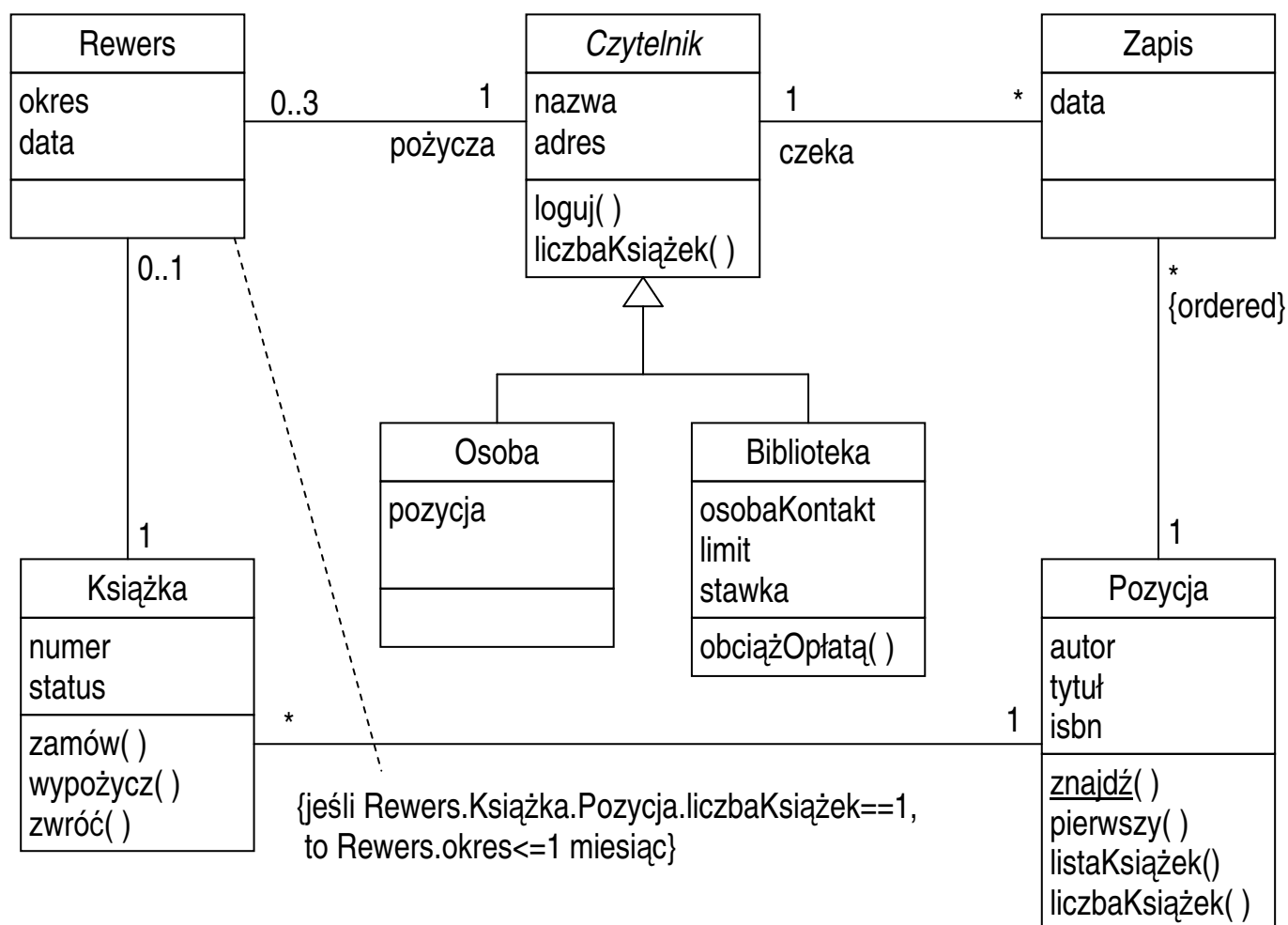
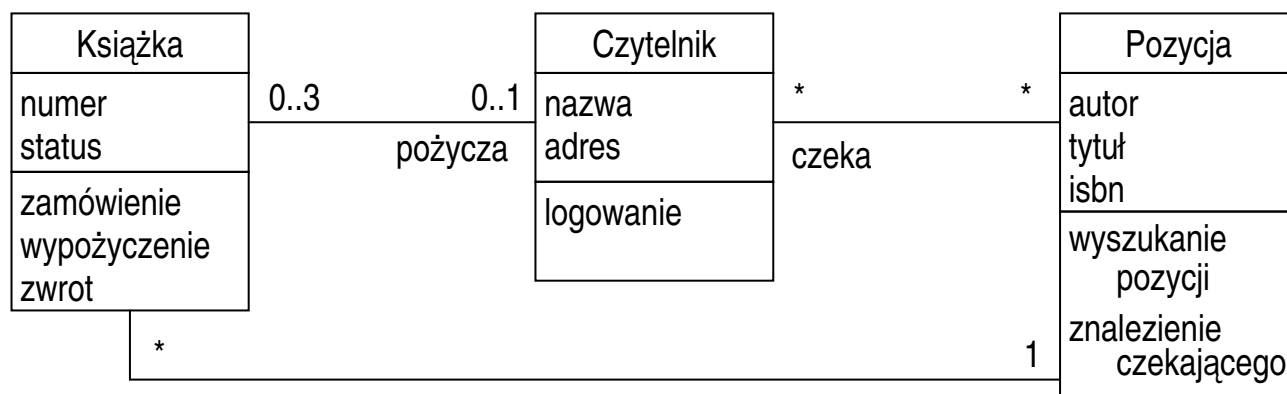
- klasy z dziedziny problemu
- nieformalny opis odpowiedzialności

### **2. Model analizy**

- uzupełnienie opisu klas
- uzupełnienie hierarchii klas
- dodawanie klas pomocniczych
- łączenie modeli różnych obszarów tematycznych
- budowa modeli zachowania

### **3. Model projektu i implementacji**

- określenie komponentów i ich interfejsów
- uporządkowanie hierarchii dziedziczenia
- określenie klas implementujących operacje
- wyodrębnienie modelu bazy danych

Przykład: system biblioteczny

## Dodatkowe elementy opisu klas

- stereotypy

<<interface>>	tylko operacje (brak atrybutów i metod)
<<type>>	tylko atrybuty i operacje (brak metod)
<<business>>	klasa bierna, nie inicjuje działań

- metki

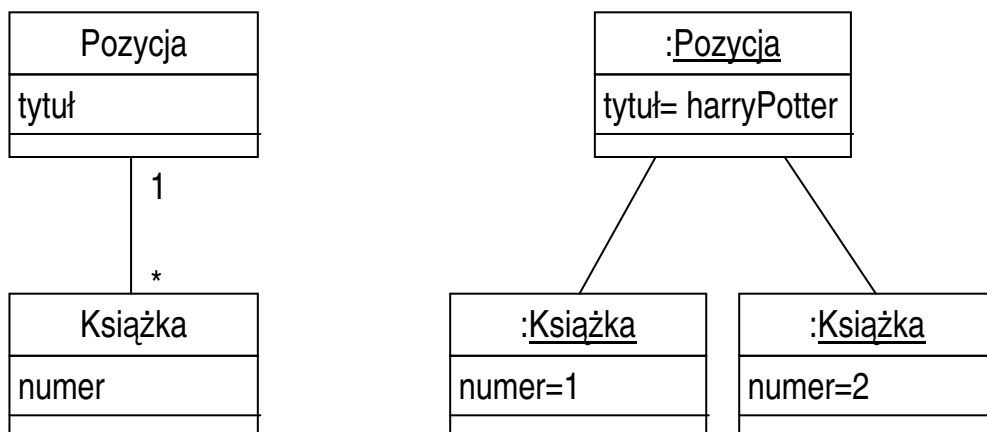
{ abstrakt }	klasa abstrakcyjna (też <i>kursywa</i> )
{ root }	klasa najwyższa w hierarchii generalizacji
{ leaf }	klasa najniższa w hierarchii generalizacji
{ persistent }	klasa reprezentująca obiekty bazy danych

## 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

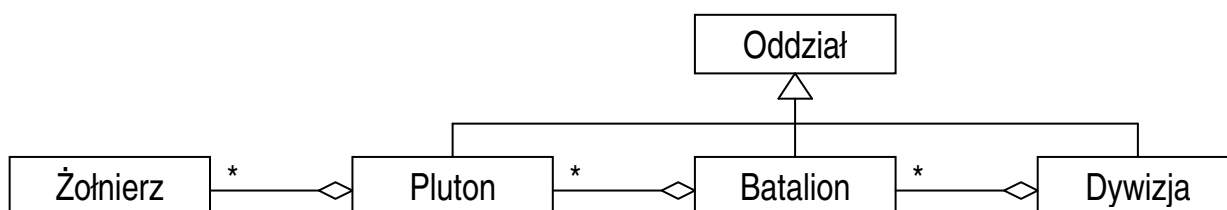
## Przykład: gra komputerowa

- 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

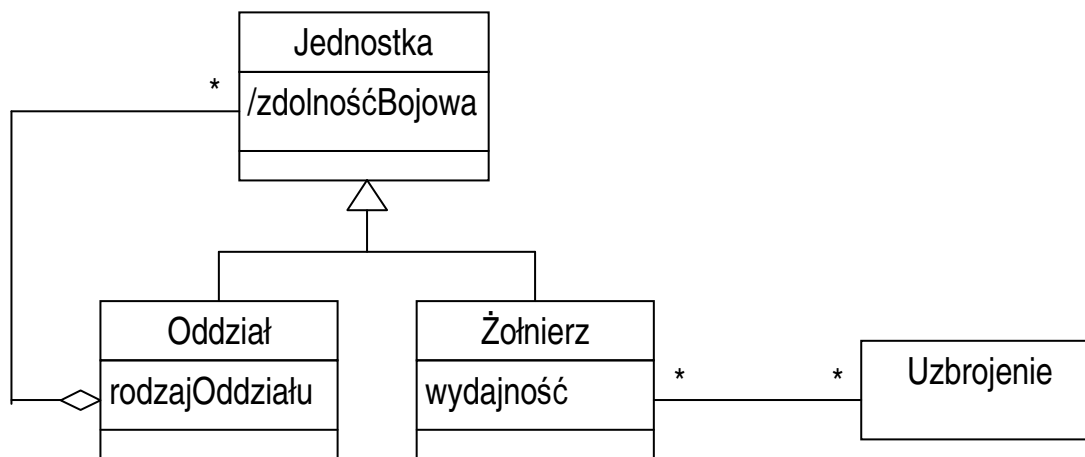
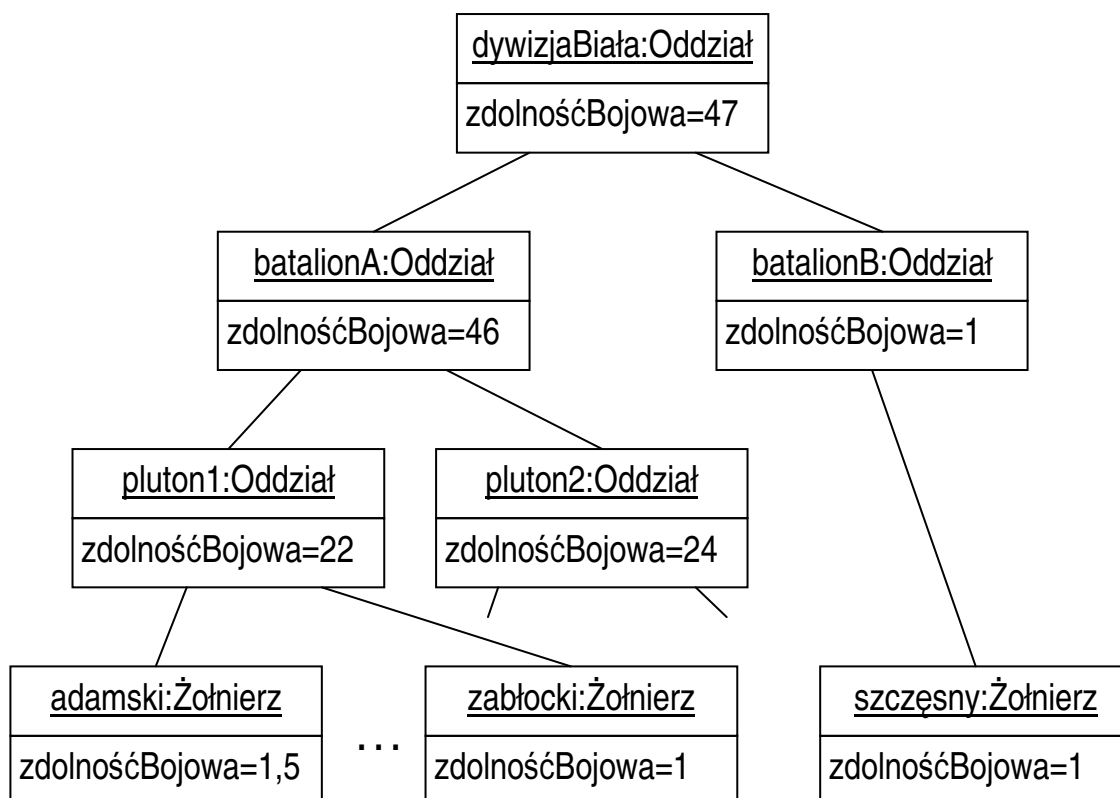


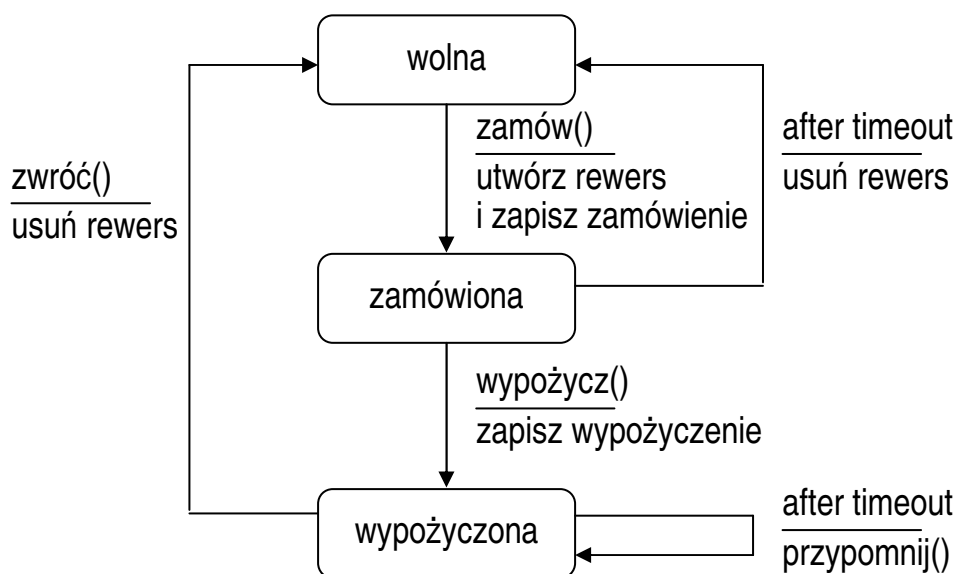
Diagram obiektów pokazuje chwilową konfigurację



→ Konfiguracja obiektów a reprezentacja to nie jest to samo!

## 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

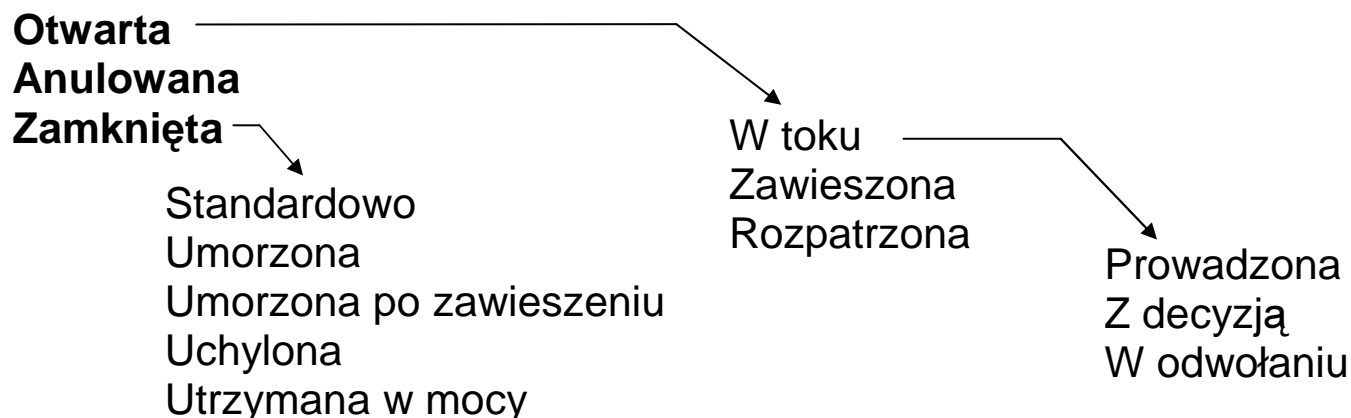
Zdarzenie:

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

## Hierarchia stanów

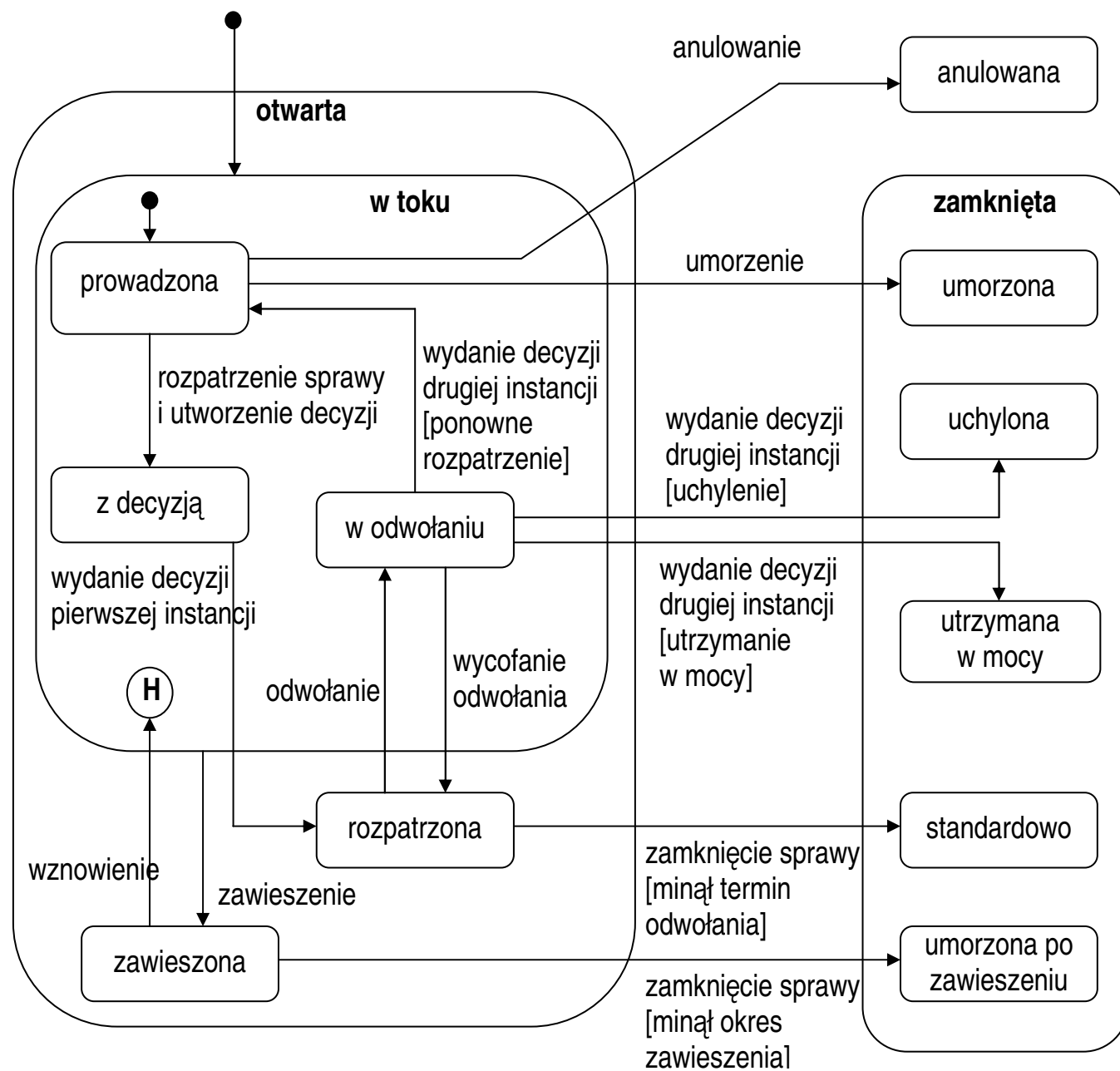
- Modelowanie na różnych poziomach abstrakcji
- Grupa pokrewnych stanów → nadstan (stan złożony)

Przykład: stany sprawy administracyjnej w systemie IACS.



Reguły przetwarzania sprawy są w różnych stanach różne

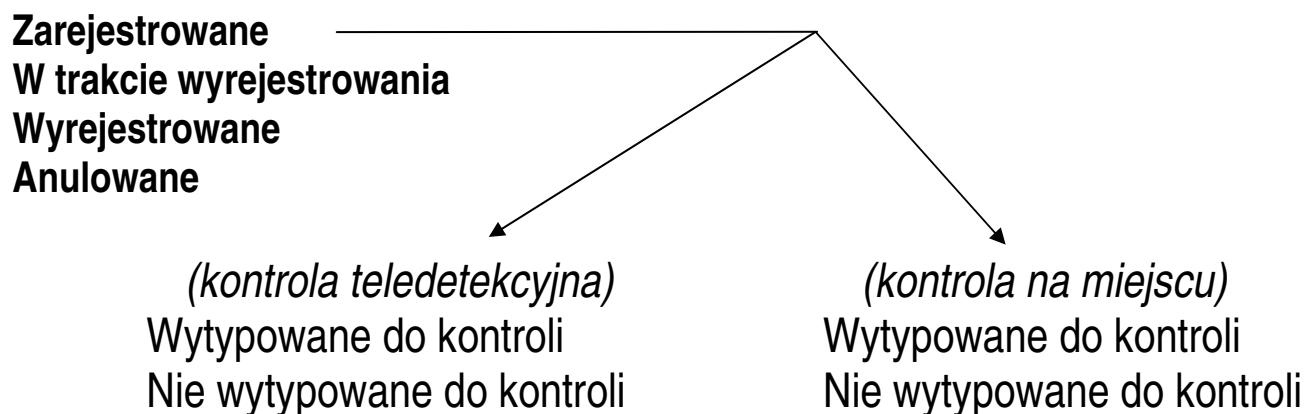
## Diagram stanów sprawy



## Stany równoległe

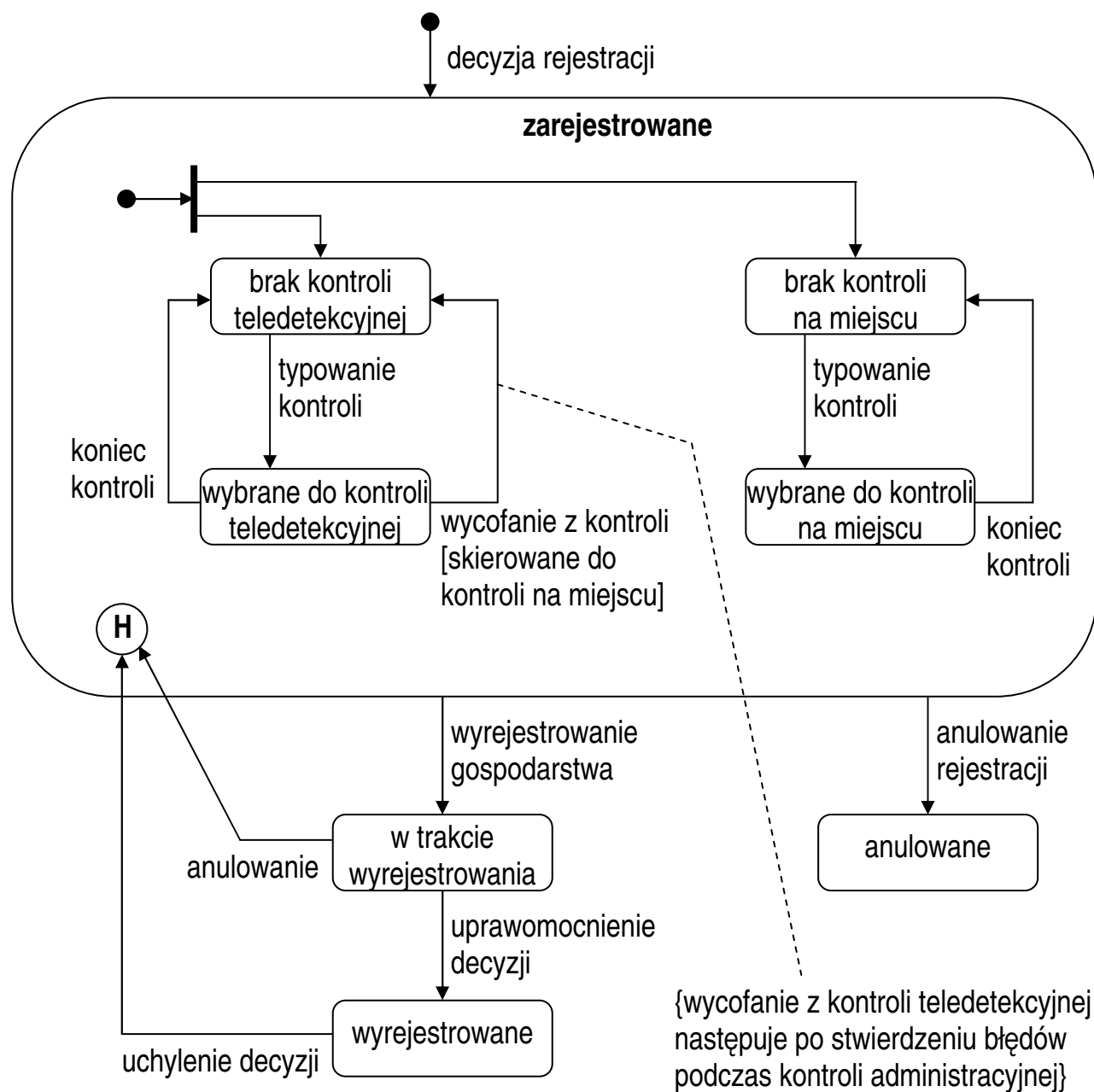
- Różne wymiary zachowania → niezależne pod-diagramy

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*



## Modelowanie współpracy obiektów

- Karty CRC
- Diagram sekwencji
- Diagram współdziałania

### Karty CRC

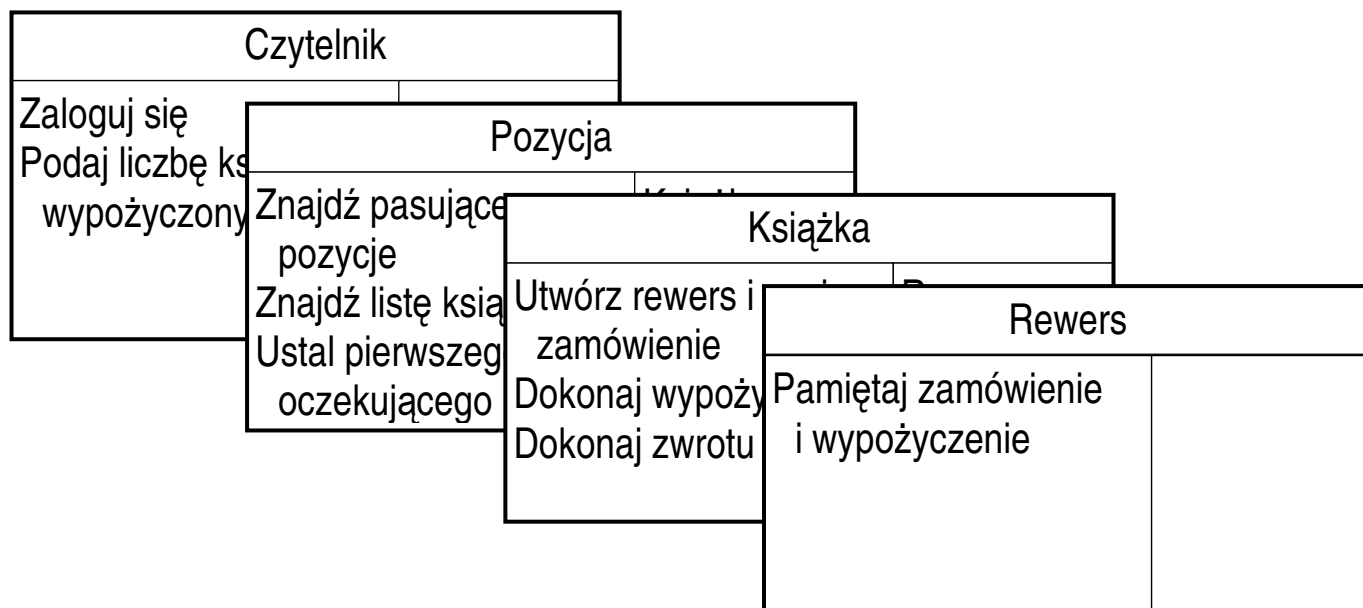
- Nieformalny opis zachowania klas
- Elementy opisu klasy
  - nazwa
  - lista zobowiązań
  - lista klas współpracujących
- Postać

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

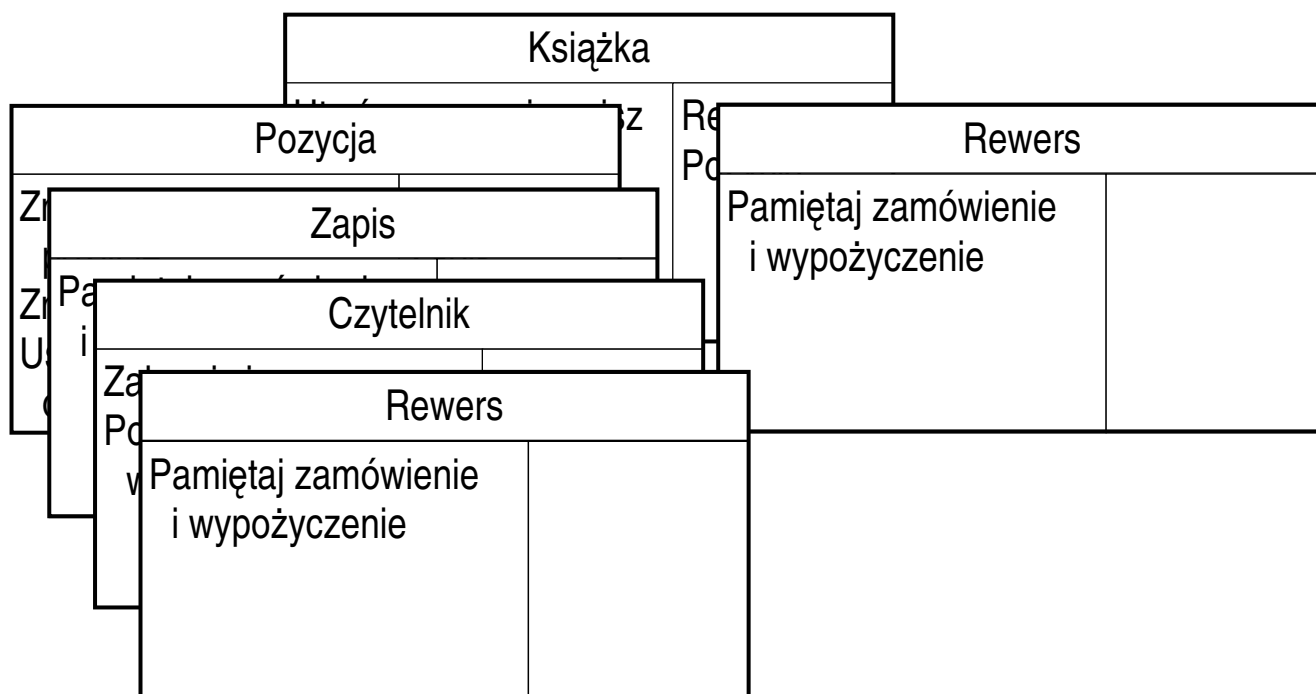
Pozycja	
Znajdź pasujące pozycje	Książka
Znajdź listę książek	Zapis
Ustal pierwszego oczekującego	

Przykład: system biblioteczny

- Zamówienie książki

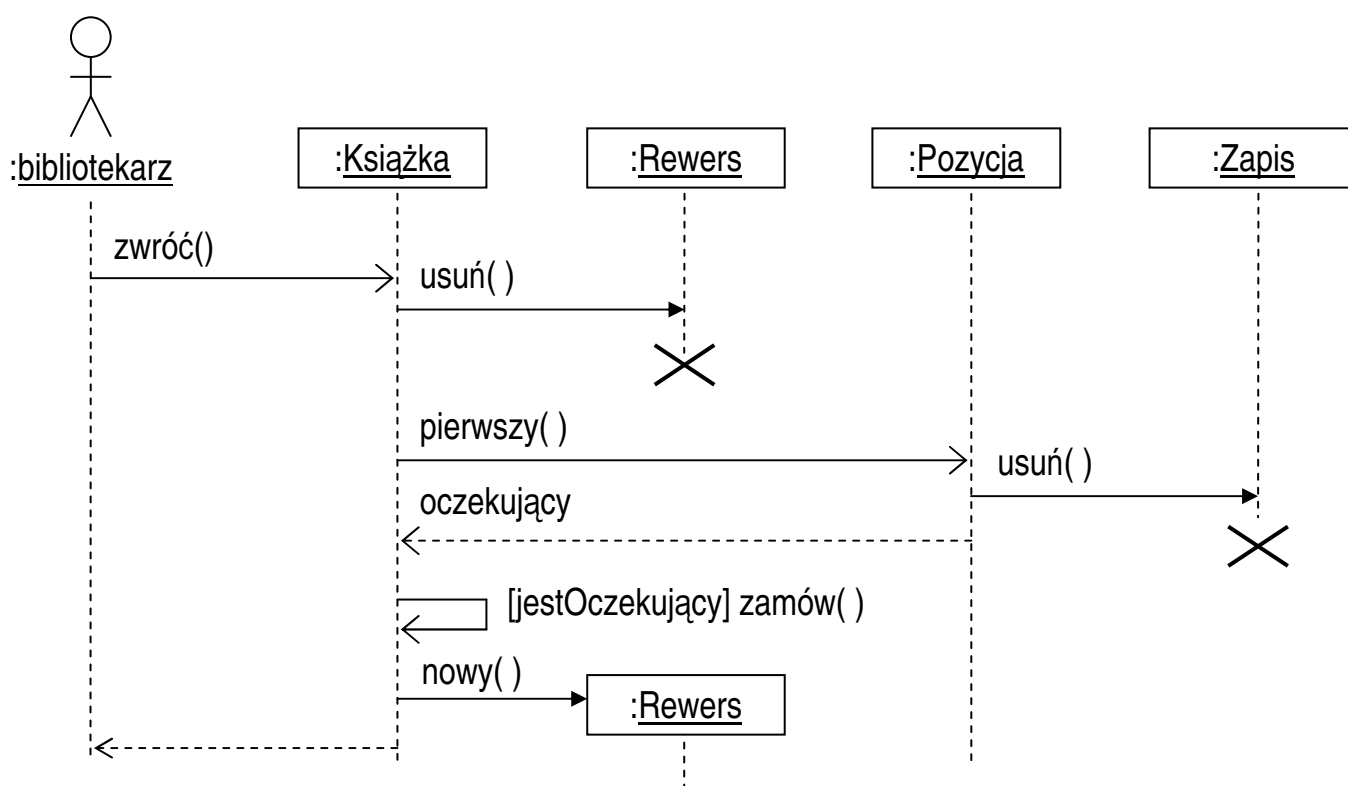


- Zwrot książki

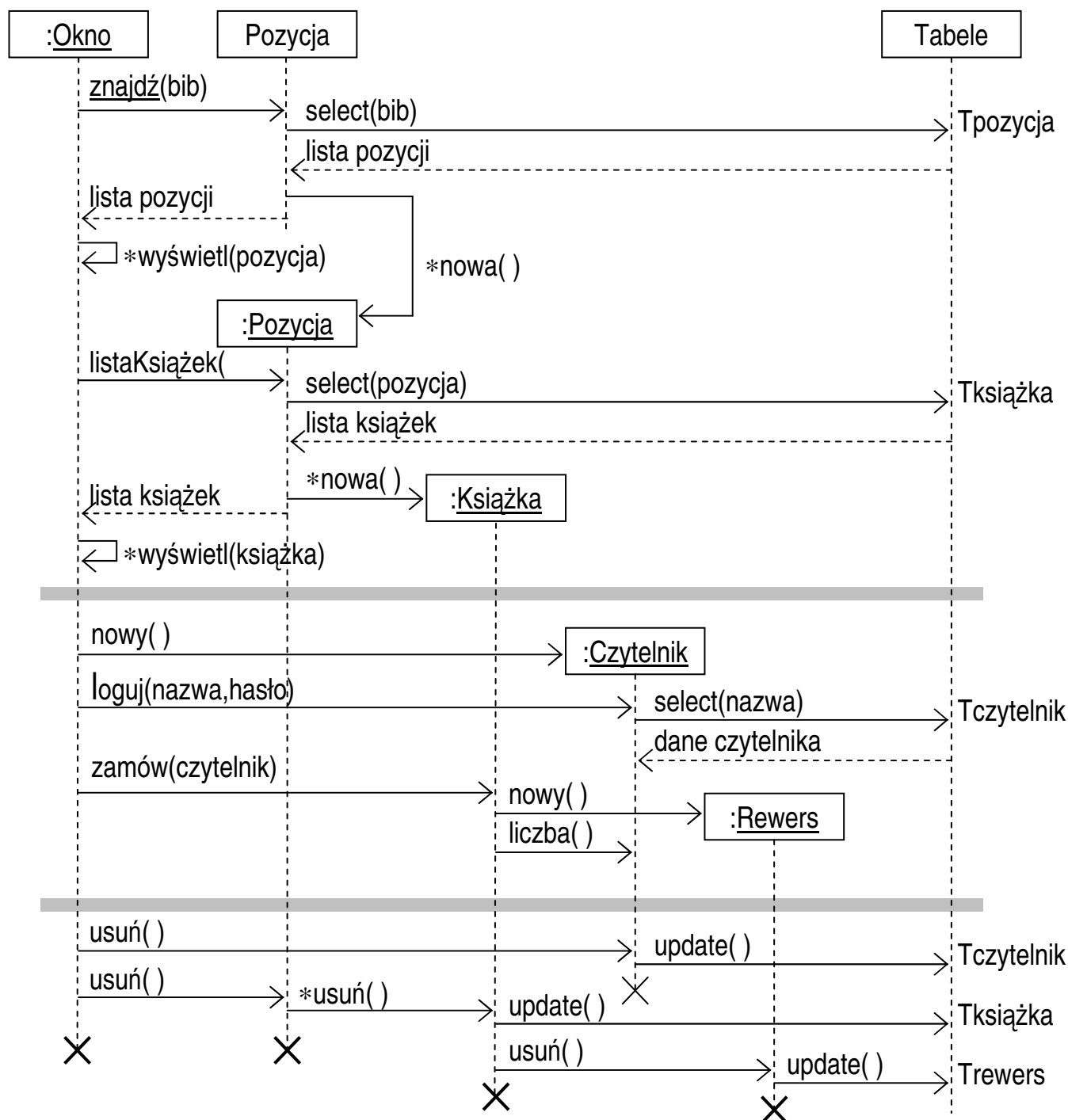


## Diagram sekwencji

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

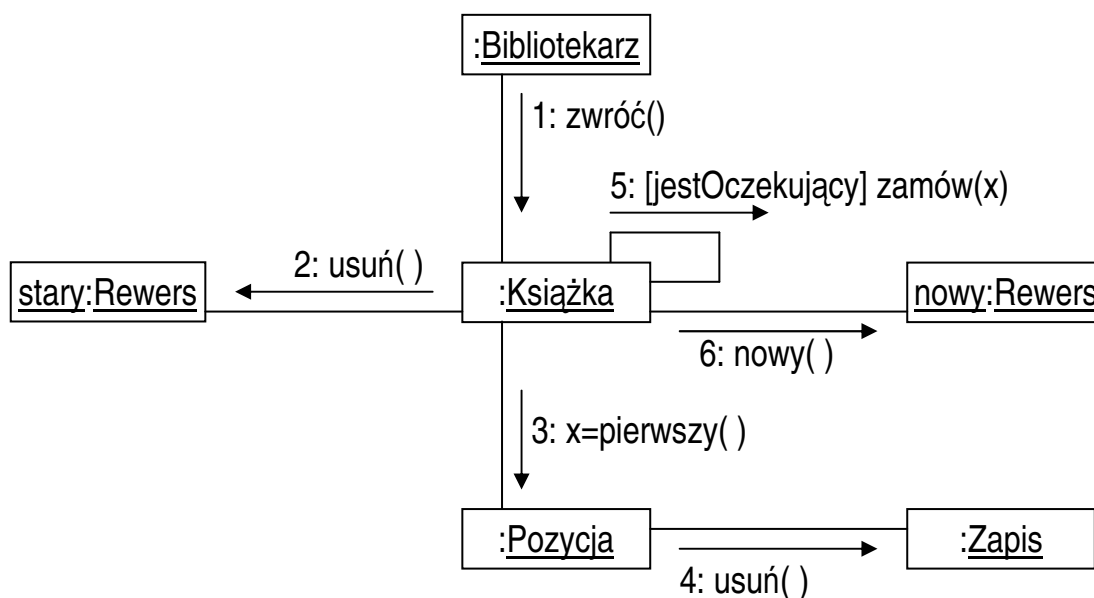


## Przykład: system biblioteczny (zamówienie książki)



## Diagram współdziałania

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



- Obowiązkowa numeracja komunikatów:
  - chronologiczna
  - dziesiętna

## **Zastosowanie modeli współpracy obiektów**

### Modele analityczne

1. Budowa modelu zaczyna się po zdefiniowaniu przypadków użycia i zdefiniowaniu podstawowych klas
    - nieformalny opis na kartach CRC
    - szybka weryfikacja wariantów
  2. Później formalizacja opisu w postaci diagramu
- *Obydwa rodzaje diagramów zawierają tę samą informację*

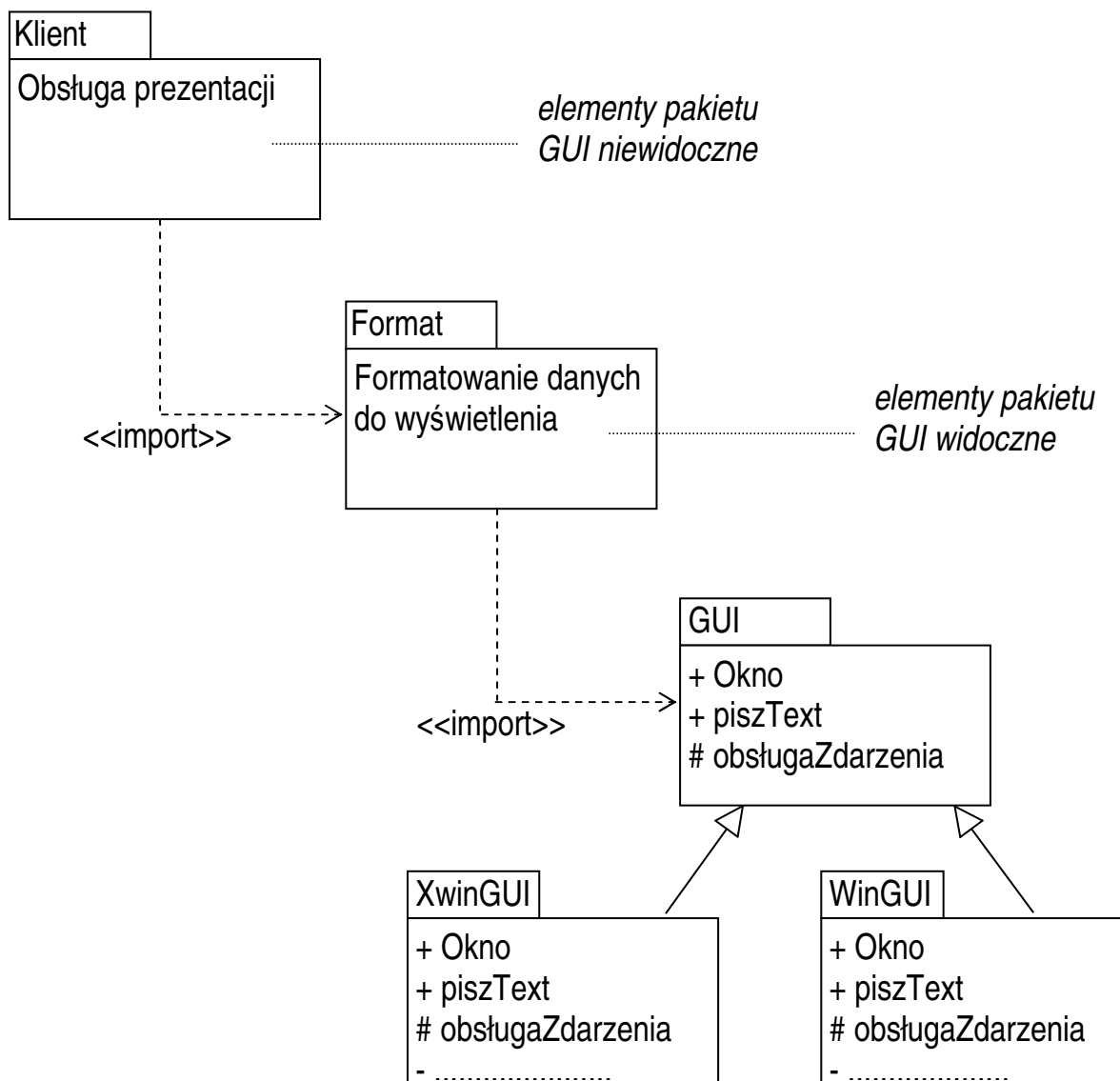
## Pakiety

- Pojemnik zawierający byty dowolnego rodzaju (np. klasy, interfejsy, przypadki użycia, inne pakiety itd.)
- Symbol graficzny

Nazwa
– opis tekstowy
– lista elementów
– diagramy

- Cechy
  - jest grupą innych bytów: bez tożsamości i egzemplarzy
  - określa przestrzeń nazw  
*pakiet\_zewnetrzny::.....::pakiet\_wewnetrzny::element*
  - ogranicza widoczność: publiczne, prywatne, chronione

- Import pakietu: operacja nie jest przechodnia





## **Pakiety klas**

- Najczęściej spotykany rodzaj pakietu — narzędzie strukturalizacji kodu (np. pakiety Javy, pliki C++).
- Kryterium grupowania klas w pakiety:
  - liczba asocjacji?
  - hierarchia dziedziczenia?
  - wspólny interfejs?

### *Zależność*

potrzeba dostosowania jednego elementu do drugiego:

A zależy od B, jeśli modyfikacja B wymusza modyfikację A

Prawidłowa struktura systemu minimalizuje zależności.

- Zalecenie:  
oddzielać interfejs pakietu od realizacji (np. warstwy)
- Diagramy, na których mogą wystąpić pakiety:
  - diagram współdziałania
  - diagram zależności

## **Zastosowanie pakietów**

### 1. Zarządzanie projektem

→ podział systemu na heterogeniczne pakiety, grupujące np. przypadki użycia, klasy i diagramy stanu.

Pakiet zawiera artefakty związane z budową części systemu

### 2. Dekompozycja systemu

→ podział systemu na jednorodne pakiety (klas) opisuje 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

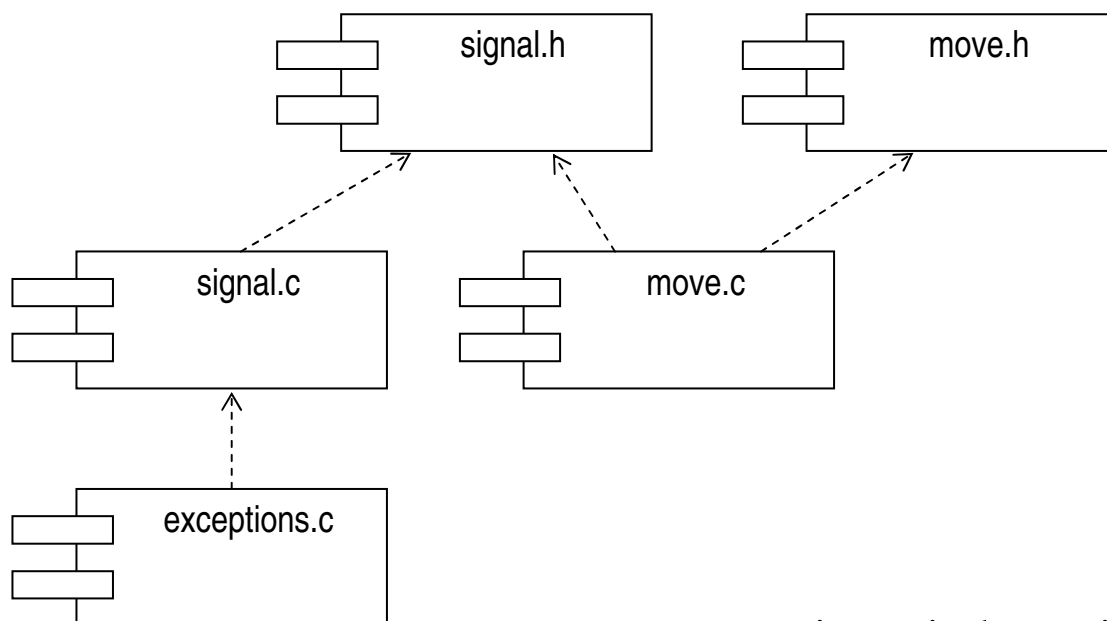
## Diagramy implementacyjne

Diagram komponentów

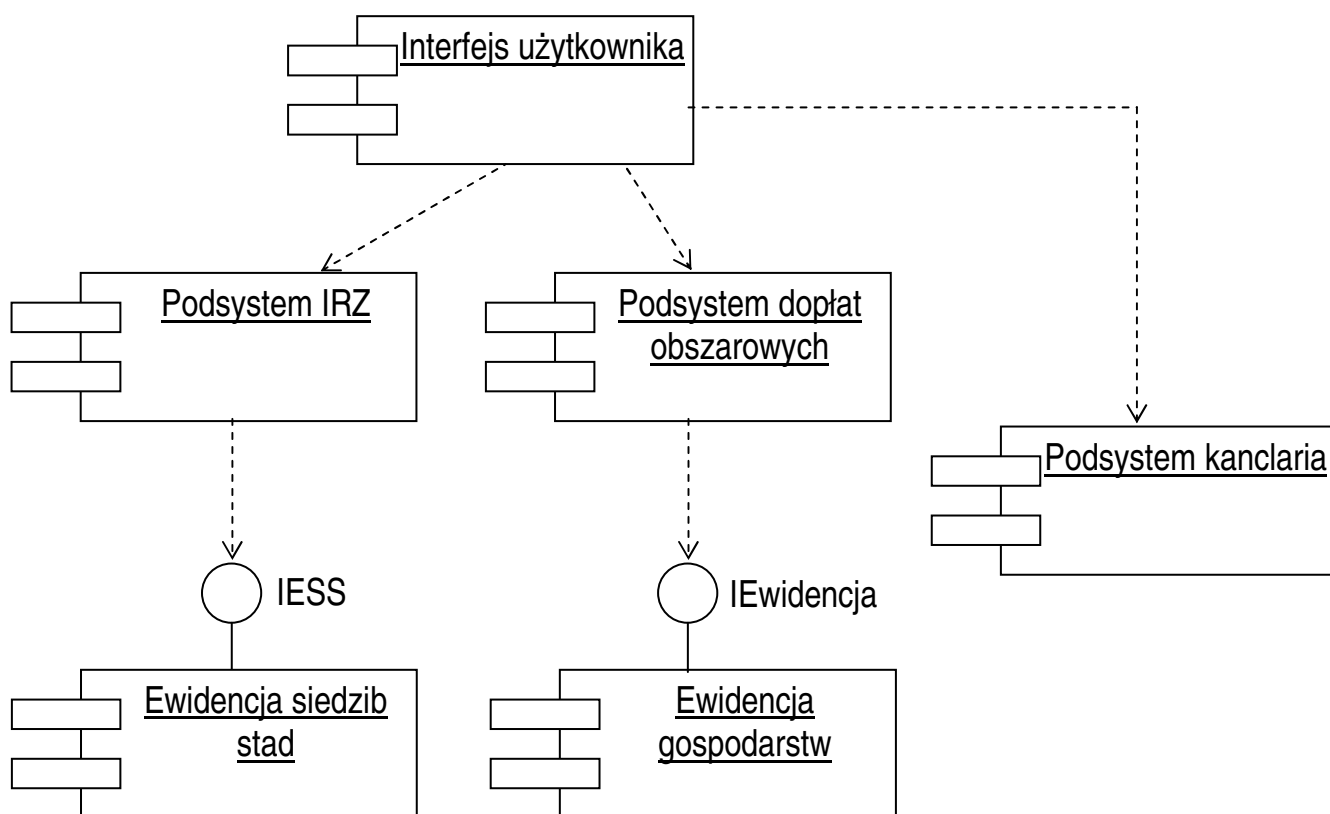
Diagram rozmieszczenia

### ***Diagram komponentów***

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



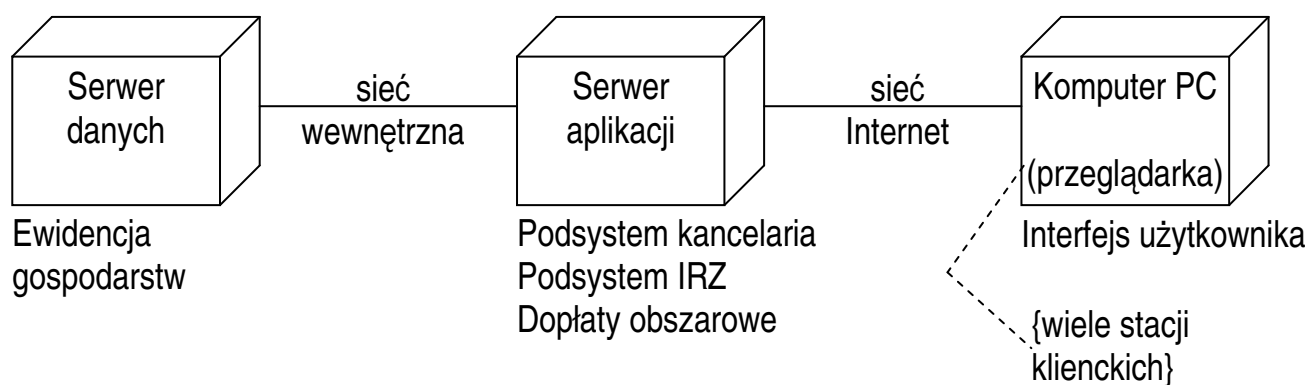
– powiązania kompilacyjne



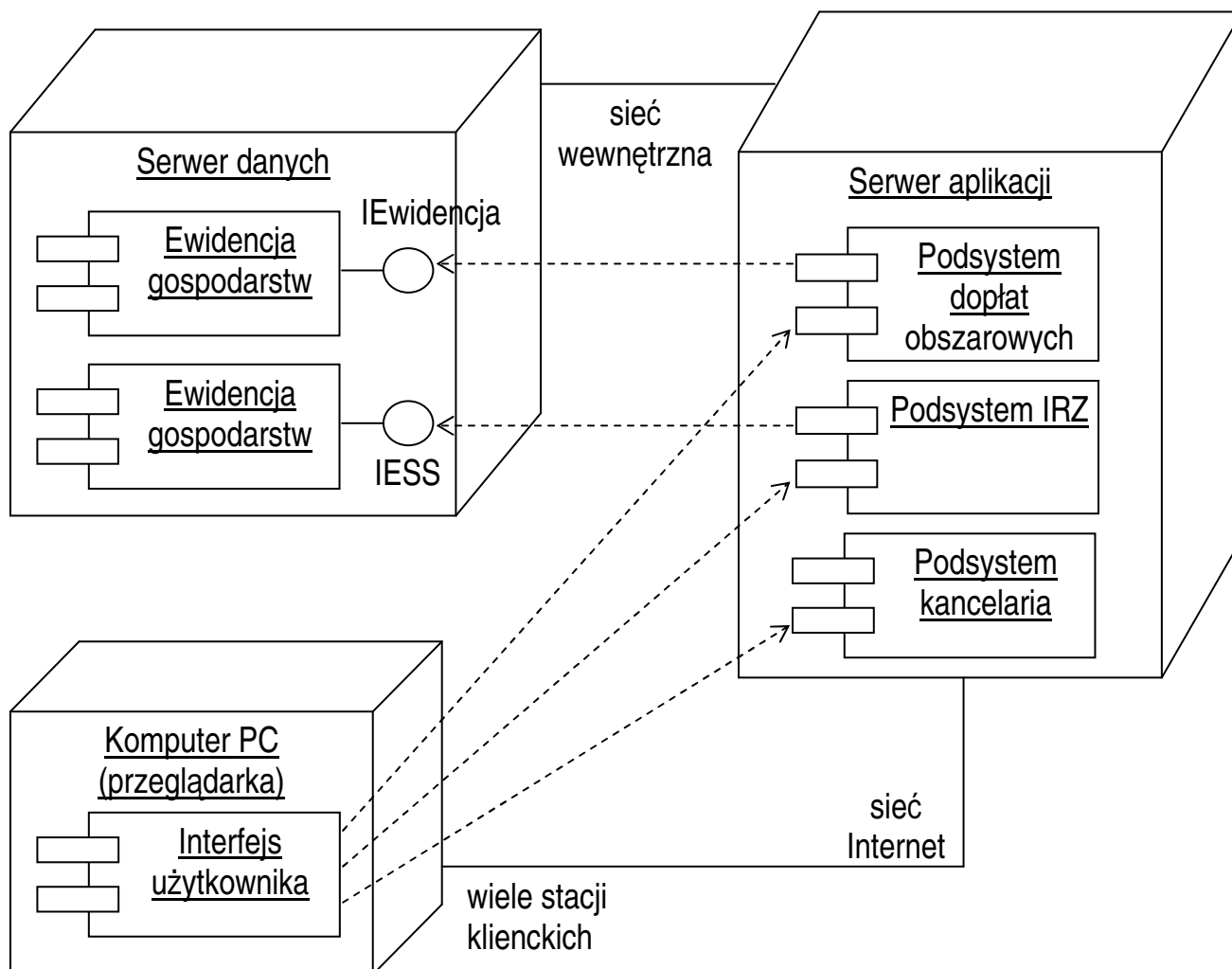
– związki współpracy komponentów.

## 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**

Modele projektowe i implementacyjne

1. Pierwsze wersje diagramów rozmieszczenia powstają w fazie rozwinięcia (koncepcja systemu)
  2. Diagramy komponentów powstają podczas projektu
  3. Wersja ostateczna po określeniu struktury implementacji
- *Diagramy powinny określać numer wersji komponentów*

## Metodologia

- Brak spójnej i szeroko akceptowanej metodyki
- Proces RUP określa ogólny schemat przebiegu projektu
- Język UML definiuje notację i zestaw modeli

### 1. Analiza dziedziny problemu

*(faza inicjacji i rozwinięcia)*

#### Cel:

- poznanie dziedziny aplikacji i wymagań użytkownika
- wyodrębnienie działań wspieranych przez system

#### Metoda:

- studiowanie dokumentów
- udział w prezentacjach i rozmowach z ekspertami
- klasyfikacja i dokumentowanie wymagań

#### Wyniki:

- dokumentacja regulacji prawnych
- określenie sprzęgów zewnętrznych
- biznesowy model przypadków użycia



## **2. Wyodrębnienie funkcji systemu**

*(faza rozwinięcia)*

### Cel:

- zdefiniowanie funkcji systemu
- określenie interfejsu użytkownika

### Metoda:

- dalsza analiza dziedziny aplikacji i wymagań użytkownika
- analiza biznesowych przypadków użycia
- implementacja prototypu interfejsu użytkownika
- wywiady z przyszłymi użytkownikami

### Wyniki:

- systemowy model przypadków użycia
- prototyp interfejsu użytkownika

### **3. Opracowanie modelu dziedziny problemu.**

*(faza rozwinięcia)*

#### Cel:

- identyfikacja i klasyfikacja podstawowych bytów dziedziny
- określenie zachowania kluczowych obiektów

#### Metoda:

- dalsza 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

## **4. Analiza wybranego fragmentu systemu.**

*(faza konstrukcji)*

Proces RUP nie dzieli fazy konstrukcji na analizę i projekt — przeciwnie, traktuje łącznie czynności analizy i projektu

Cel:

- określenie sposobu działania systemu

Metoda:

- weryfikacja i uzupełnienie diagramu klas
  - analiza asocjacji
  - analiza generalizacji
- określenie modelu danych
  - obiekty trwałe
  - ograniczone atrybuty i metody
- budowa modelu zachowania
  - analiza przypadków użycia
  - określenie sposobu realizacji

Wyniki:

- diagramy klas
- diagramy współpracy

## **5. Projektowanie wybranego fragmentu systemu.**

*(faza konstrukcji)*

### Cel:

- określenie budowy systemu

### Metoda:

- modyfikacja modelu klas (logika biznesowa)
  - dodanie elementów prywatnych i chronionych
  - uściślenie kierunków asocjacji
  - dostosowanie hierarchii klas do reguł dziedziczenia
- projektowanie bazy danych
  - określenie tabel i ich powiązań
  - optymalizacja wydajności
  - określenie planu archiwizacji i odtwarzania
- określenie struktury programów
  - podział na komponenty
  - wykorzystanie technologii systemowych
- projektowanie interfejsu (GUI)

### Wyniki:

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

## **6. Implementacja wybranego fragmentu systemu.**

*(faza konstrukcji)*

### Cel:

- opracowanie programów

### Metoda:

- definicja klas programu
  - pola
  - metody
  - współpraca obiektów
- implementacja bazy danych
  - implementacja operacji
  - rozmieszczenie plików na dyskach
  - integracja z narzędziami systemu operacyjnego
- opracowanie podręczników
- wykorzystanie narzędzi CASE
  - automatyczna generacja kodu
  - automatyczna generacja dokumentacji

### Wyniki:

- kod programu
- dokumentacja techniczna
- podręczniki użytkownika