Wykład 2 – część pierwsza

Diagramy UML 2 – część pierwsza

Na podstawie

UML 2.0 Tutorial

http://sparxsystems.com.au/resources/uml2_tutorial/

Dwa rodzaje diagramów UML 2

Diagramy UML <u>modelowania strukturalnego</u>

- ***Diagramy pakietów***
- ***Diagramy klas***
- **Diagramy obiektów**
- **Diagramy mieszane**
- **Diagramy komponentów**
- **Diagramy wdrożenia**

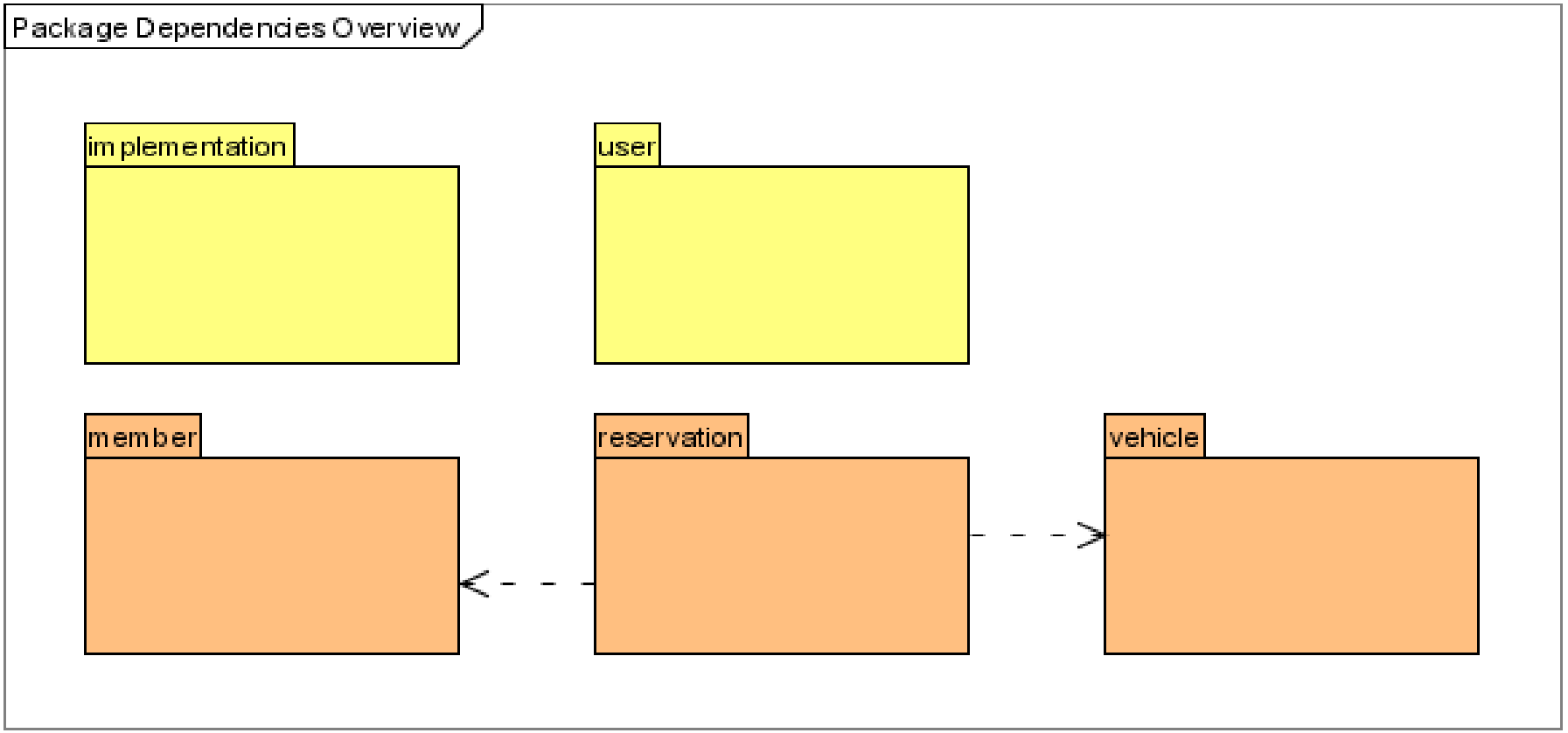
Diagramy UML <u>modelowania zachowania</u>

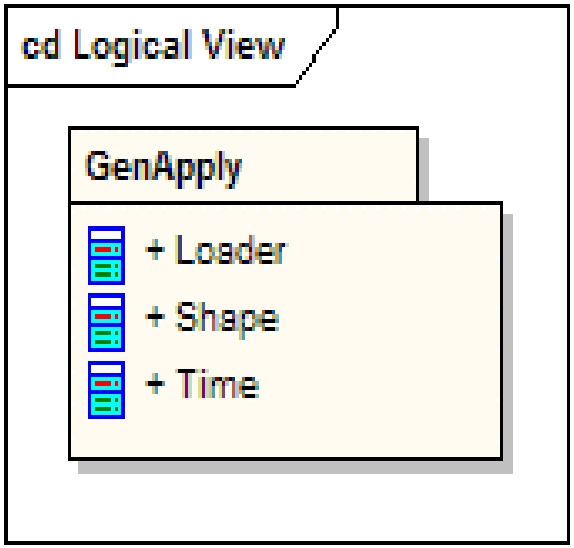
- ***Diagramy przypadków użycia***
- **Diagramy aktywności**
- **Diagramy stanów**
- **Diagramy komunikacji**
- ***Diagramy sekwencji***
- **Diagramy czasu**
- **Diagramy interakcji**

Diagram pakietów

przedstawia:

- ogólną organizację pakietu i jego elementów
- np. organizację diagramu „use-case”
- np. organizację diagramu klas





Właściwości:

- Elementy pakietu dzielą tę samą **namespace** i muszą posiadać unikatową nazwę
- Elementy pakietu reprezentują fizyczne lub logiczne zależności między klasami (powiązania, dziedziczenie, agregacja)
- Pakiety są traktowane jako foldery

Łączenie pakietów «merge» - elementy pakietu docelowego (źródła łączenia) powstają z elementów pakietu importowanego na drodze generalizacji (źródło łączenia w relacji generalizacji do importowanego pakietu)

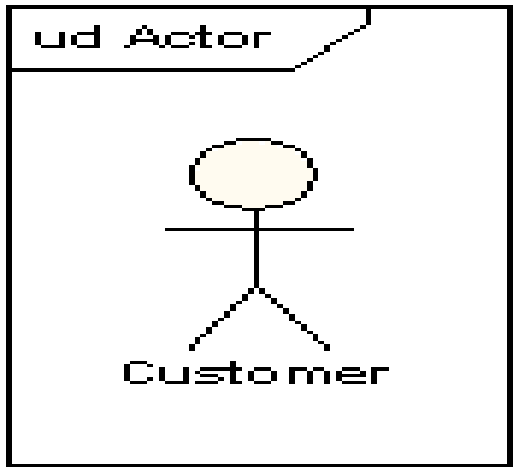
Importowanie pakietów «import» - klasa importowana znajduje się teraz w **namespace** pakietu źródłowego, do którego jest importowana, natomiast **namespace** pakietu, z którego klasa jest importowana staje się nieaktywna

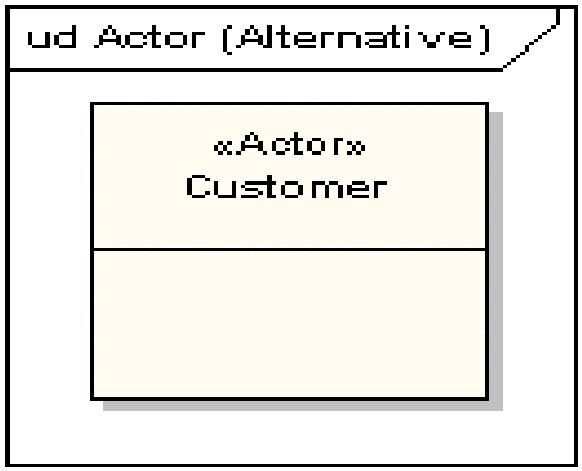
Zagnieżdżanie połączeń – połączenie między zagnieżdżonymi pakietami: źródłowym i docelowym wyraża hierarchię zawartości poszczególnych pakietów

Diagramy przypadków użycia (Use Case Diagram)

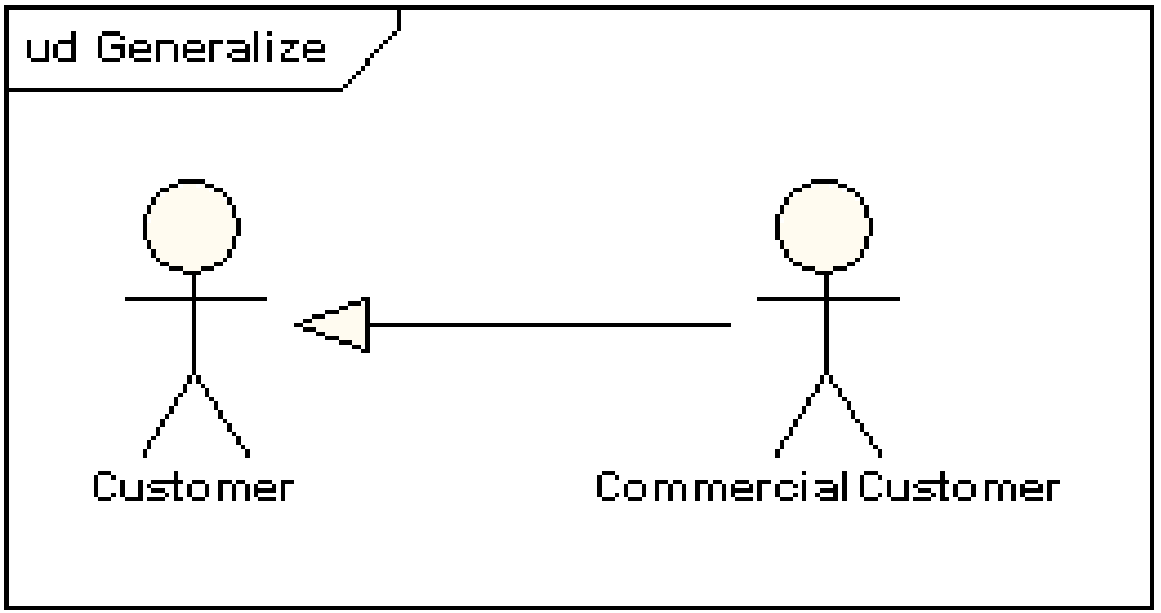
- Opisują wymagania systemu
- Przypadki użycia (Use cases) oznaczają interakcje użytkowników lub innych zewnętrznych systemów (actors) z przedstawianym systemem

Actors
Zewnętrzni użytkownicy: ludzie, sprzęt, system rysowania jako figura lub klasa ze słowem kluczowym «actor» .

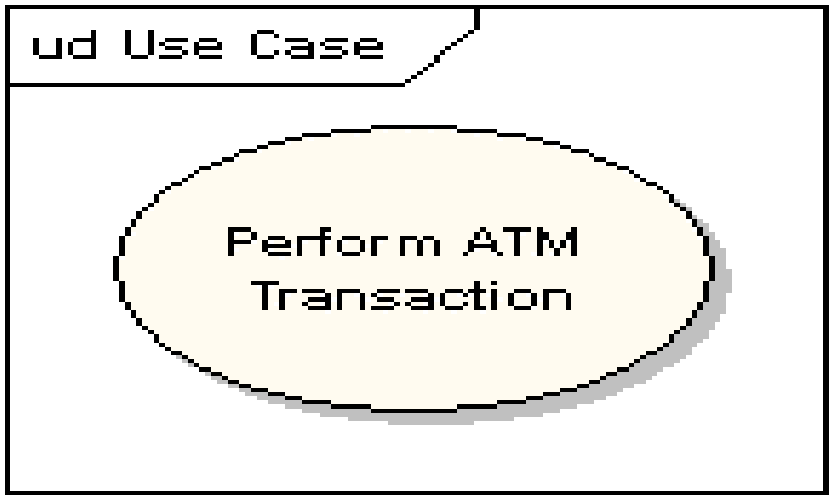




Actors mogą generalizować innych *Actors. Oznacza to rozszerzenie możliwości dostępu do przypadków. Aktor CommercialCustomer rozszerza przypadki użycia aktora dziedziczone od aktora Customer*



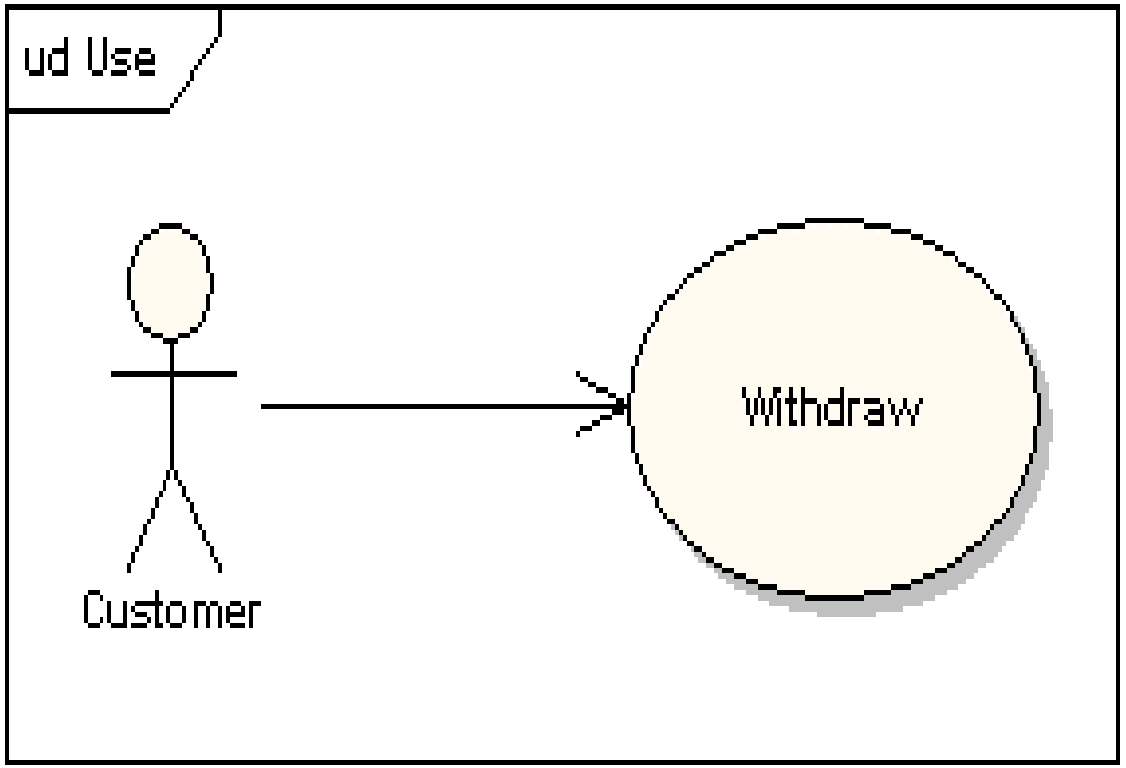
Przypadek użycia (Use Cases)



- Jednostka pracy
- Wysoki poziomom zewnętrznej obserwacji systemu
- Notacja – elipsa
- Np. aktor *Customer* używa przypadku użycia *Withdraw* (pobiera pieniądze np. z konta)

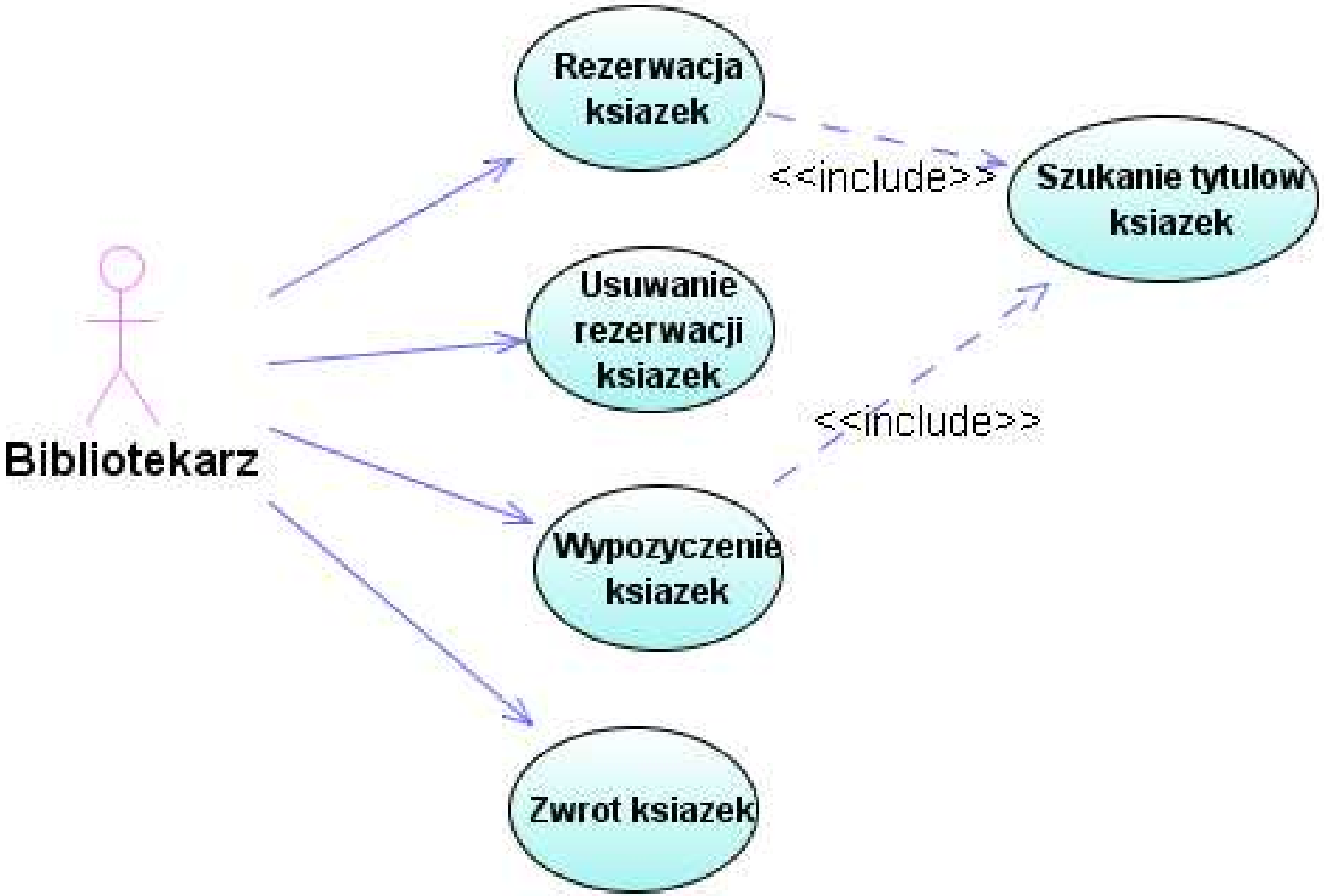
Przypadek użycia zawiera:

- **Nazwę i opis**
- **Wymagania** funkcjonalne spełniane dla użytkownika
- **Ograniczenia** – warunki przed- po- oraz nie zmieniające się na skutek wykonania przypadku użycia
- **Scenariusze** – sekwencja zdarzeń między systemem i zewnętrznymi użytkownikami (opis tekstowy)
- **Diagramy scenariuszy** – diagramy sekwencji
- **Dodatkowe informacje** – np. Identyfikacja karty płatniczej przez dokonaniem wyciągu z konta



Przykład diagramu przypadków użycia

AKTOR	OPIS	PRZYPADKI UŻYCIA
Bibliotekarz	*Bibliotekarz wypożycza książki i przyjmuje zwroty książek. Zajmuje się on również rezerwacją książek i usuwaniem rezerwacji książek*	• *Rezerwacja ksiazek* zawiera PU Szukanie tytulow ksiazek • *Usuwanie rezerwacji ksiazek* • *Wypozyczanie ksiazek zawiera* PU Szukanie tytulow ksiazek • *Zwrot ksiazek*

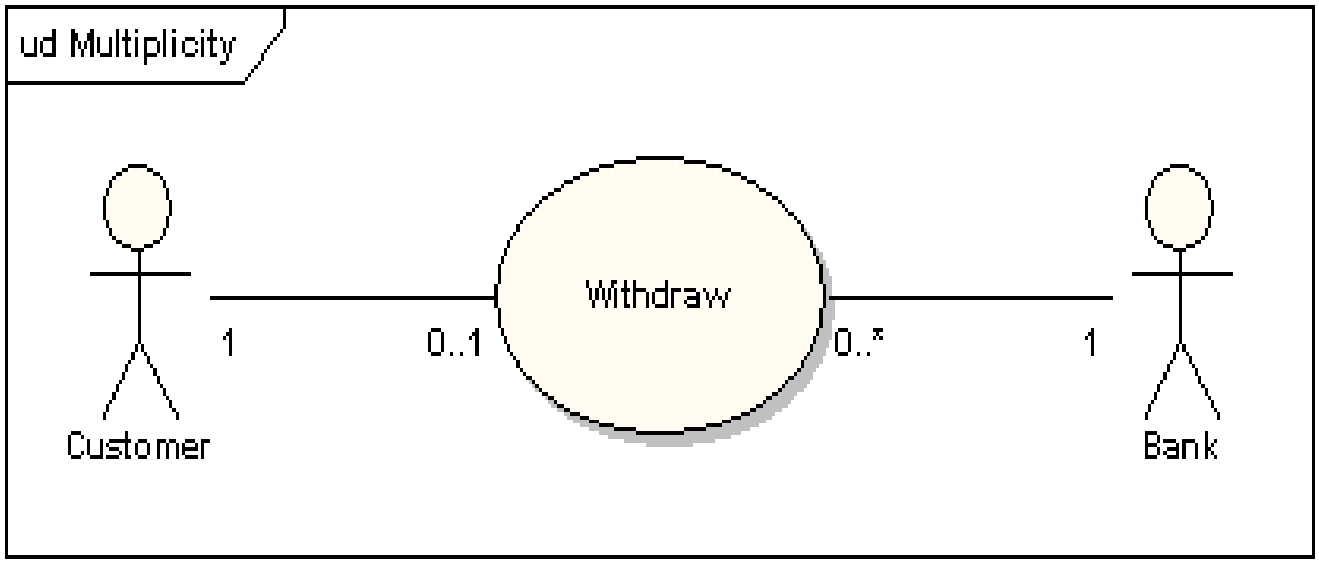

Rezerwacja ksiazek
<<include>>
Szukanie tytulow ksiazek
Usuwanie rezerwacji ksiazek
<<include>>
Bibliotekarz
Wypozyczenie ksiazek
Zwrot ksiazek

PU Wypozyczenie ksiazek
OPIS
CEL: obsługa bibliotekarza
WS (warunki wstępne): inicjalizacja przez uruchomienie programu (np. otwarcie strony WWW, start aplikacji)
WK (warunki końcowe): pojawienie się nowej danej w aplikacji lub komunikat o przyczynach braku wypożyczenia
PRZEBIEG:

1. Pokazanie menu i wybór tytułu z listy
2. Identyfikacja tytułu (wywołanie przypadku użycia **PU Szukanie tytulow ksiazek**) - jeśli tytuł jest w systemie, przejdź do punktu 3 lub automatycznie zakończ wypożyczanie
3. Identyfikacja dostępności egzemplarza (pozycji) – jeśli jest przynajmniej jeden egzemplarz wolny, przejdź do punktu 4 lub zakończ automatycznie wypożyczanie
4. Pokazanie menu i wybór wypożyczającego z listy
5. Identyfikacja wypożyczającego – jeśli jest, przejdź do punktu 6 lub zakończ automatycznie wypożyczanie
6. Rejestracja nowego wypożyczenia

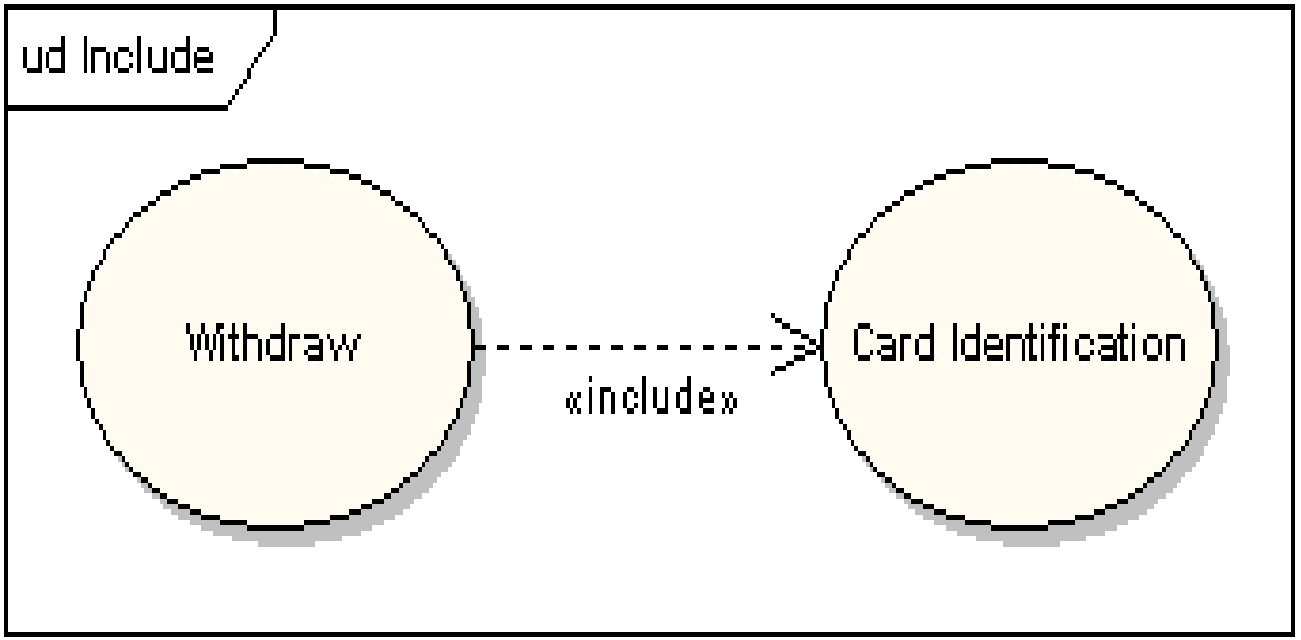
Powiązania:

•Liczność instancji na końcu połączenia
•Np. aktor *Klient (Customer)* ma tylko jedną sesję wypłacania pieniędzy w danym momencie (*Withdraw*) natomiast *Bank* może mieć ich wiele w tym samym czasie



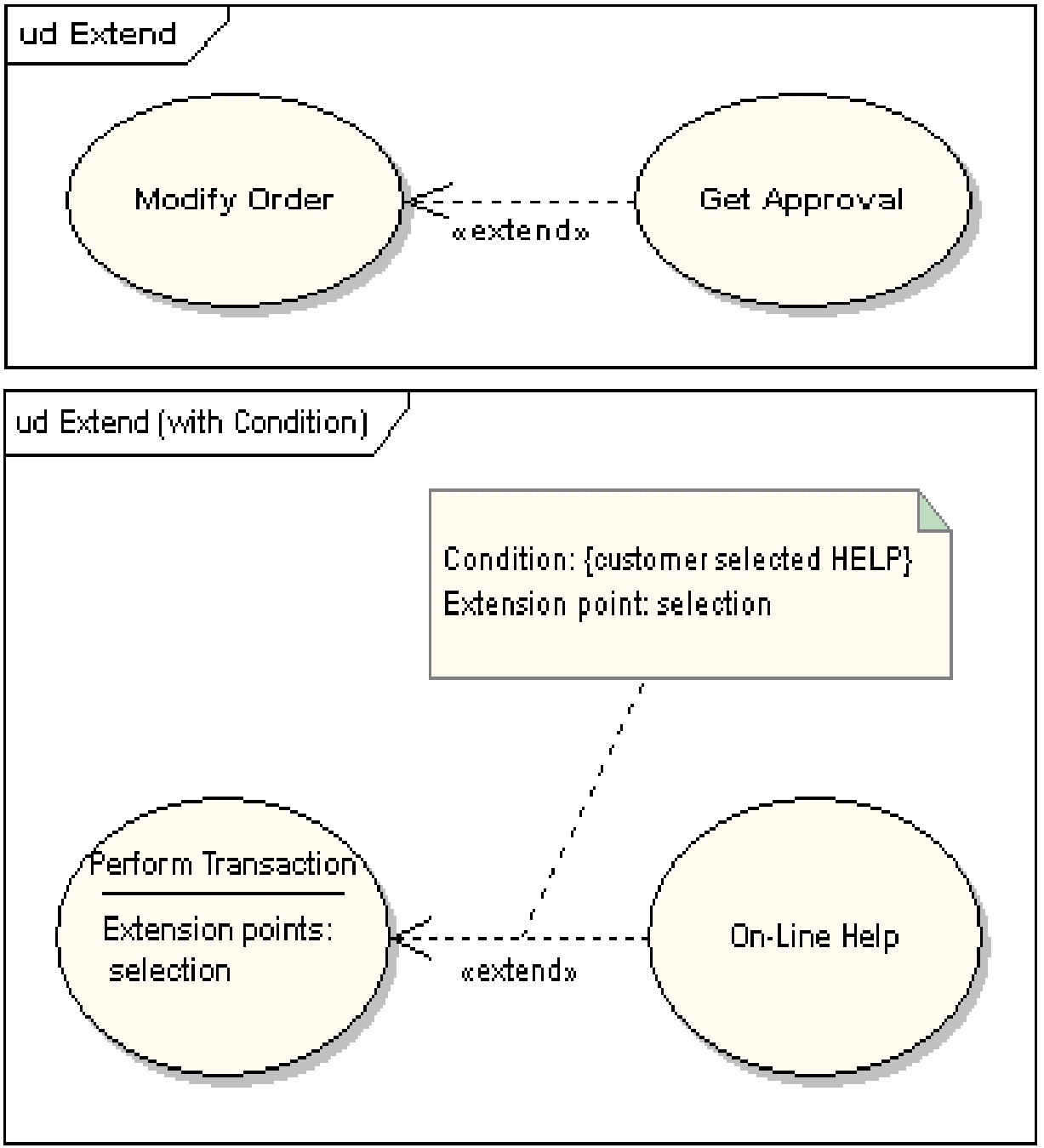
Zawieranie <<includes>>:

Przypadek użycia zawiera jeden lub wiele innych przypadków użycia eliminując powtarzanie funkcjonalności systemu dzięki tej wielużywalności, czyli zawieraniu
np. Pobranie z konta (*Withdraw*) zawsze zawiera identyfikację karty (*Card identification*)



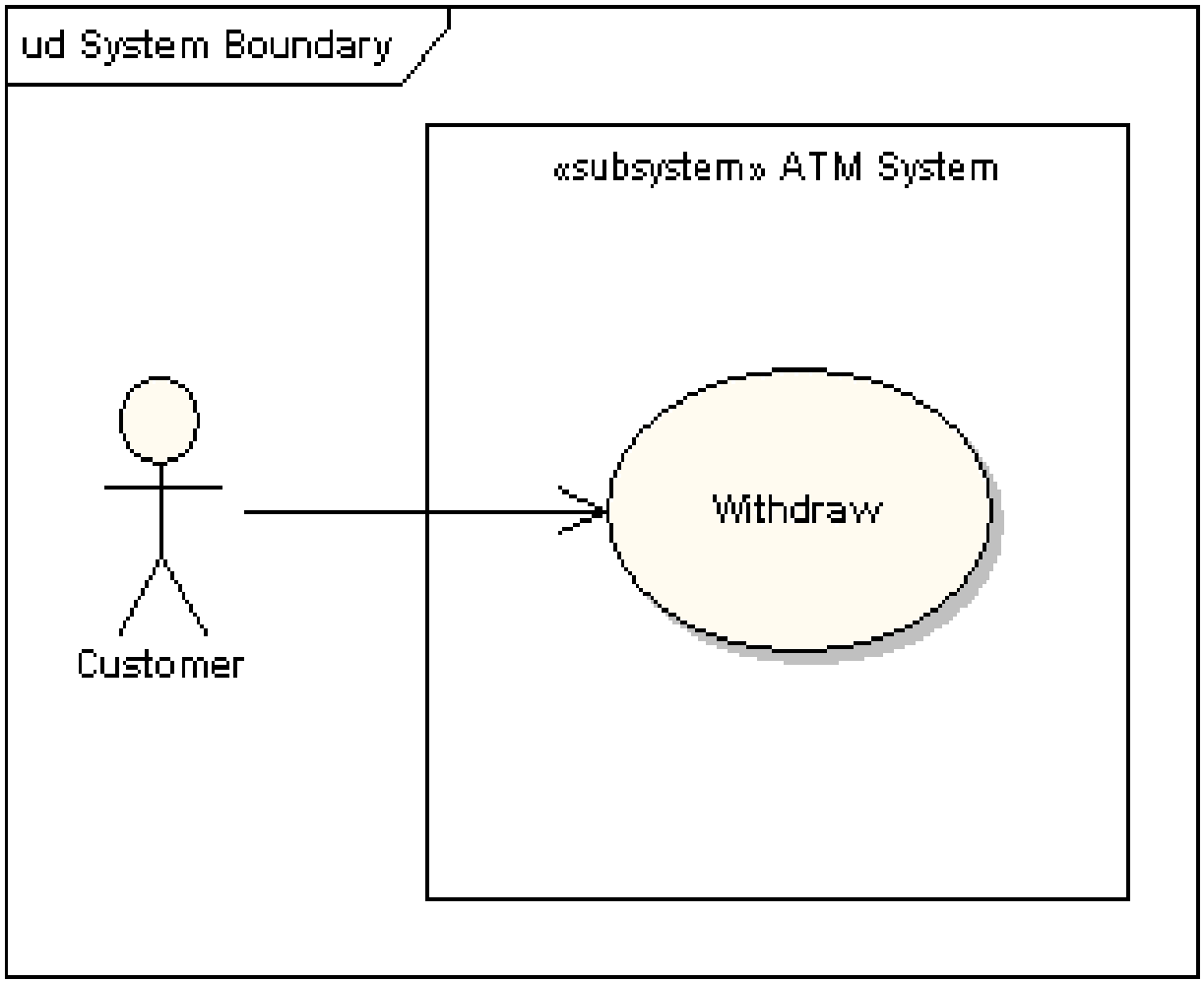
Rozszerzanie <<extends>>:
Jeden przypadek użycia może być użyty do rozszerzenia właściwości drugiego przypadku użycia Np. Przypadek użycia *Zezwolenie (Get Approval)* opcjonalnie rozszerza właściwości przypadku użycia *Modyfikuj zlecenie (Modify Order)*

Punkty rozszerzające (Extension Points)
np..Punkt, w którym rozszerzany przypadek użycia *Wykonanie tranzakcji* (*Perform Transaction*) jest rozszerzany przez rozszerzajacy przypadek użycia *Pomoc* (*On-Line Help*) zgodnie ze znaczeniem punktu rozszerzania np. przez *wybór* (*selection*)



Granice systemu (System Boundary)

Zazwyczaj aktorzy są na zewnątrz systemu, a przypadki użycia wewnątrz systemu.



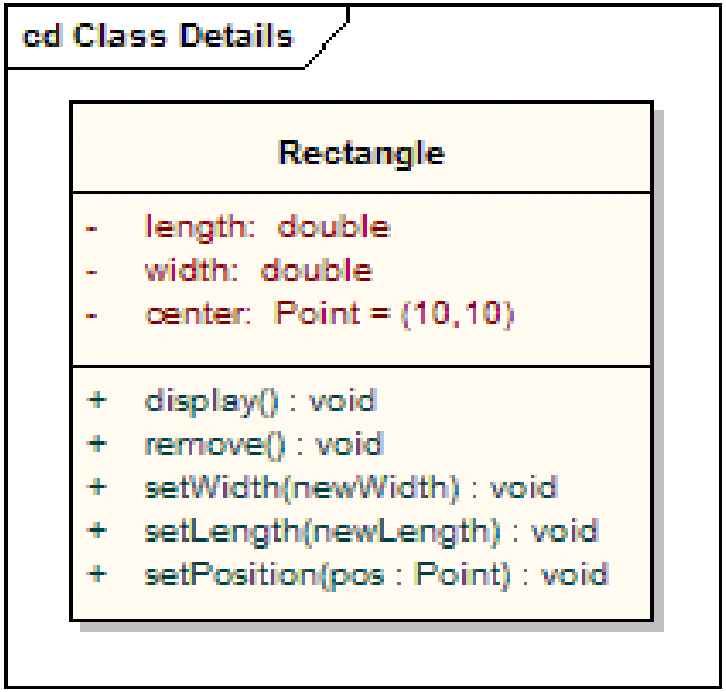
Diagramy klas (Class Diagrams)

•**Diagram klas** reprezentuje statyczny model świata rzeczywistego: jego atrybuty i właściwości oraz powiązania

•**Klasa** reprezentuje model rzeczy konceptualnej i fizycznej i jest powielana w postaci **obiektów**, czyli wystąpień klasy.

•**Atrybuty***:* składowe do przechowywania danych, które posiadają nazwę, typ, zakres wartości oraz określony dostęp.

•**Operacje:** składowe do wykonania operacji na atrybutach, zadeklarowane jako funkcje publiczne lub prywatne posiadające nazwę oraz zdefiniowany sposób wykonania.



Notatacje

Atrybuty: length, width, center**.** Atrybut **center** posiada wartość początkową.

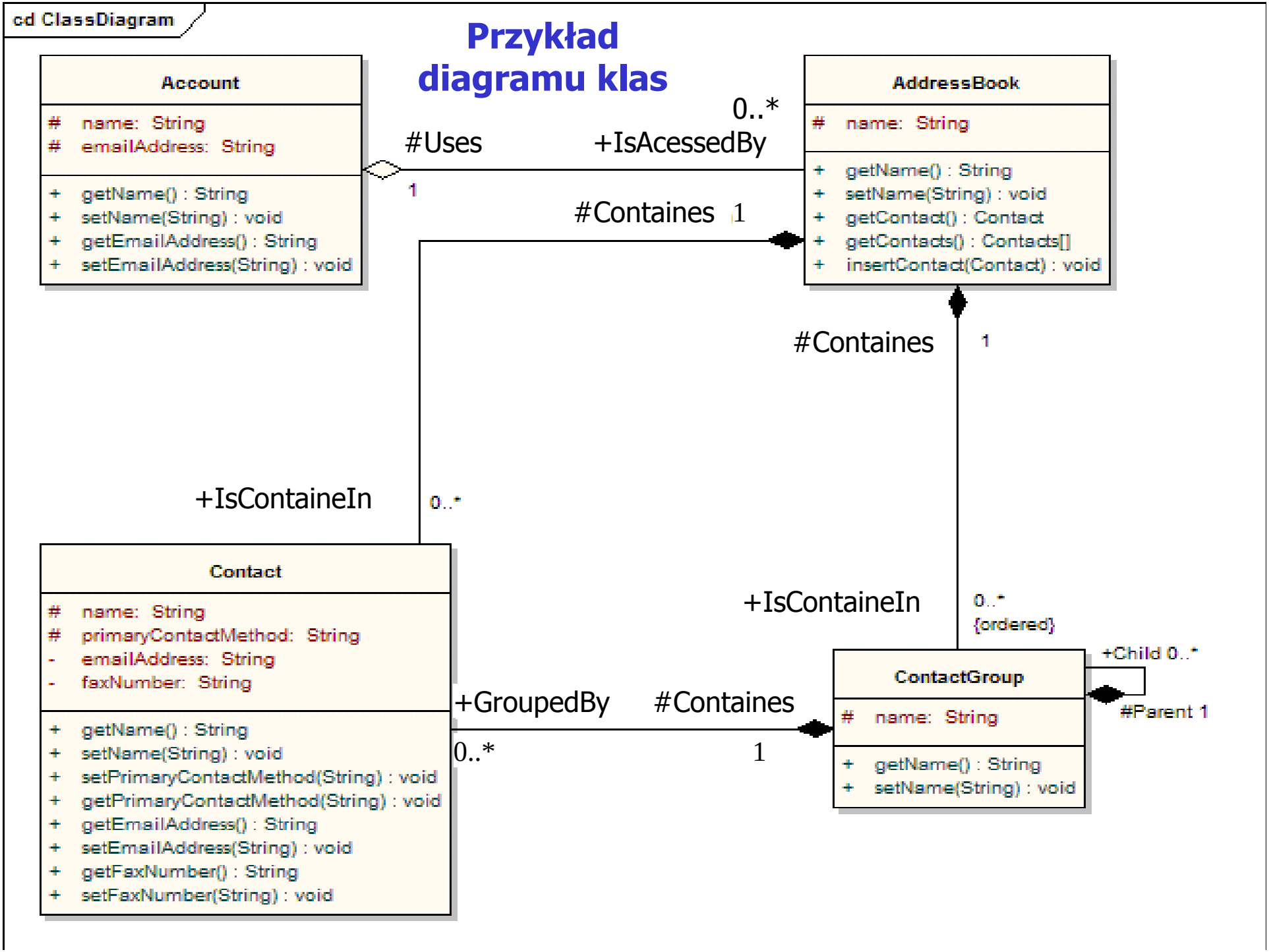
Operacje: setWidth, setLength, setPosition

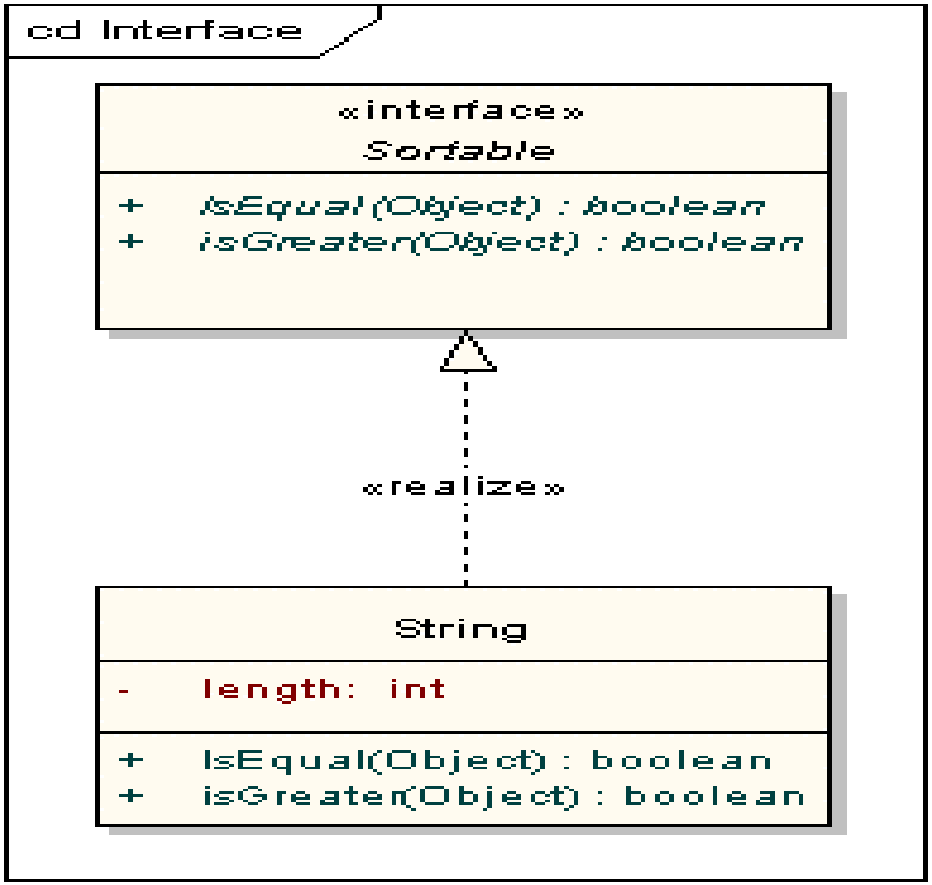
+ składowa publiczna

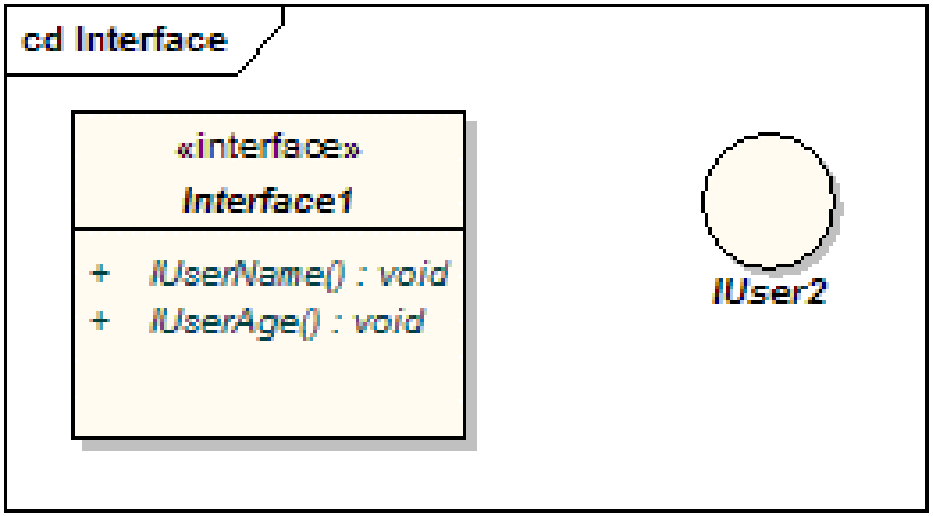
- składowa prywatna

składowa typu protected

~ składowa publiczna w zasięgu pakietu


cd ClassDiagram
Przykład diagramu klas
Account
name: String
emailAddress: String
+ getName() : String
+ setName(String) : void
+ getEmailAddress() : String
+ setEmailAddress(String) : void
AddressBook
name: String
+ getName() : String
+ setName(String) : void
+ getContact() : Contact
+ getContacts() : Contacts[]
+ insertContact(Contact) : void
#Uses
1
0..*
+IsAcessedBy
#Containes
1
#Containes
1
+IsContaineIn
0..*
Contact
name: String
primaryContactMethod: String
- emailAddress: String
- faxNumber: String
+ getName() : String
+ setName(String) : void
+ setPrimaryContactMethod(String) : void
+ getPrimaryContactMethod(String) : void
+ getEmailAddress() : String
+ setEmailAddress(String) : void
+ getFaxNumber() : String
+ setFaxNumber(String) : void
+IsContaineIn
0..*
{ordered}
+Child 0..*
#Parent 1
ContactGroup
name: String
+ getName() : String
+ setName(String) : void
+GroupedBy
0..*
#Containes
1



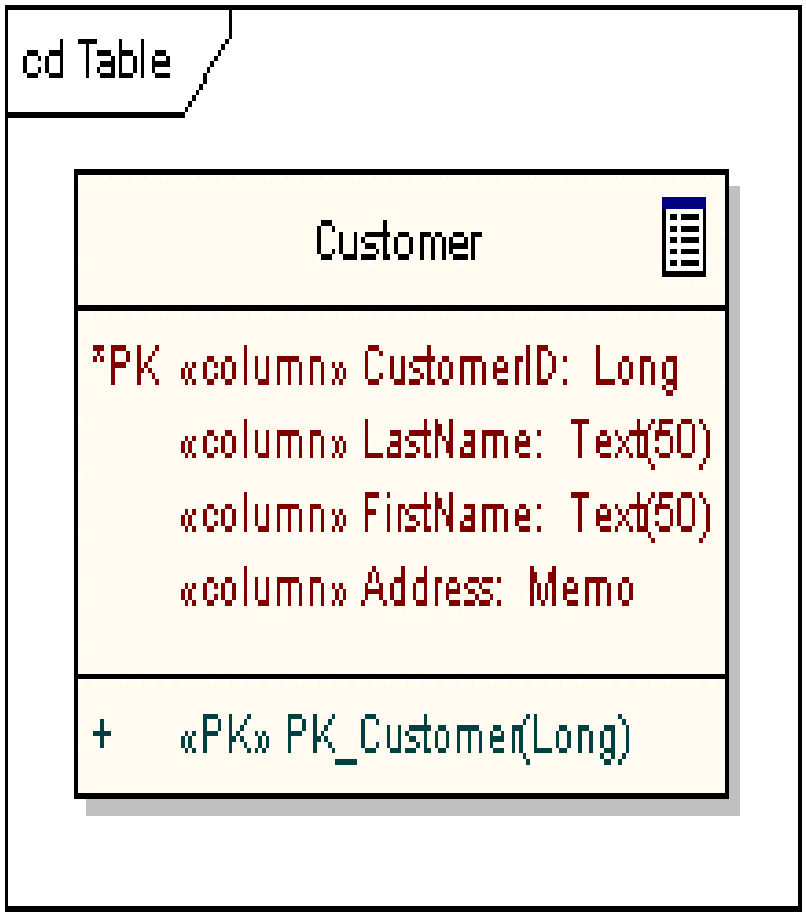


Interfejs (interface)

Jest specyfikacją właściwości (operacji czyli metod), które musi zdefiniować implementująca go klasa

Notacje

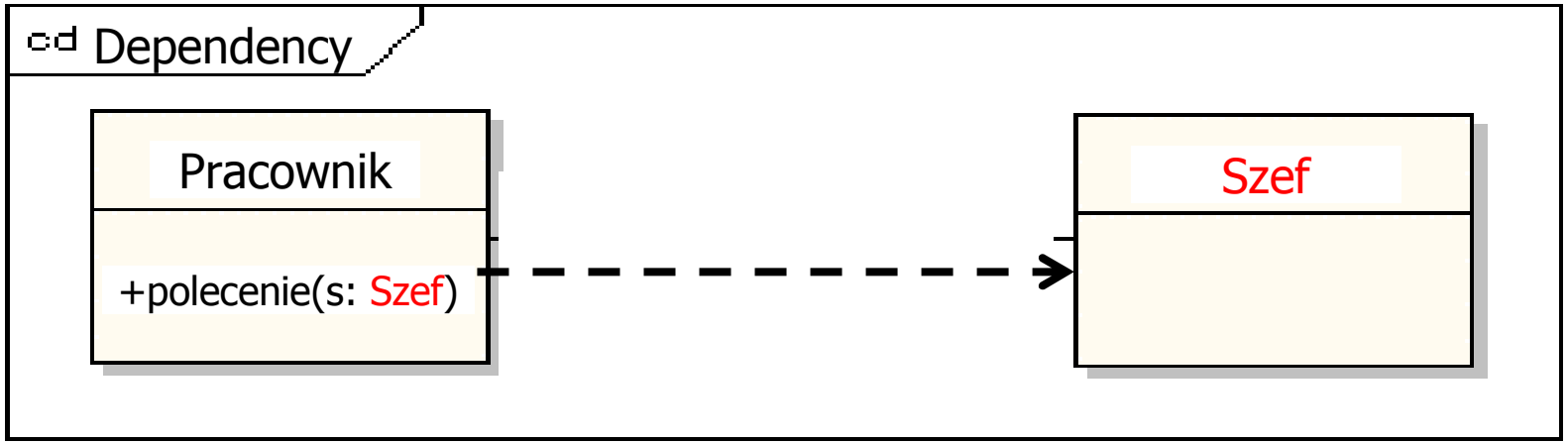
- Interfejsy są rysowane podobnie jak klasy z przerywaną strzałką wychodzącą od klasy, która realizuje metody, ze stereotypem <<realize>>
- lub jako koła bez wyspecyfikowanych metod; połączenia z klasą implementującą nie są oznaczane strzałką



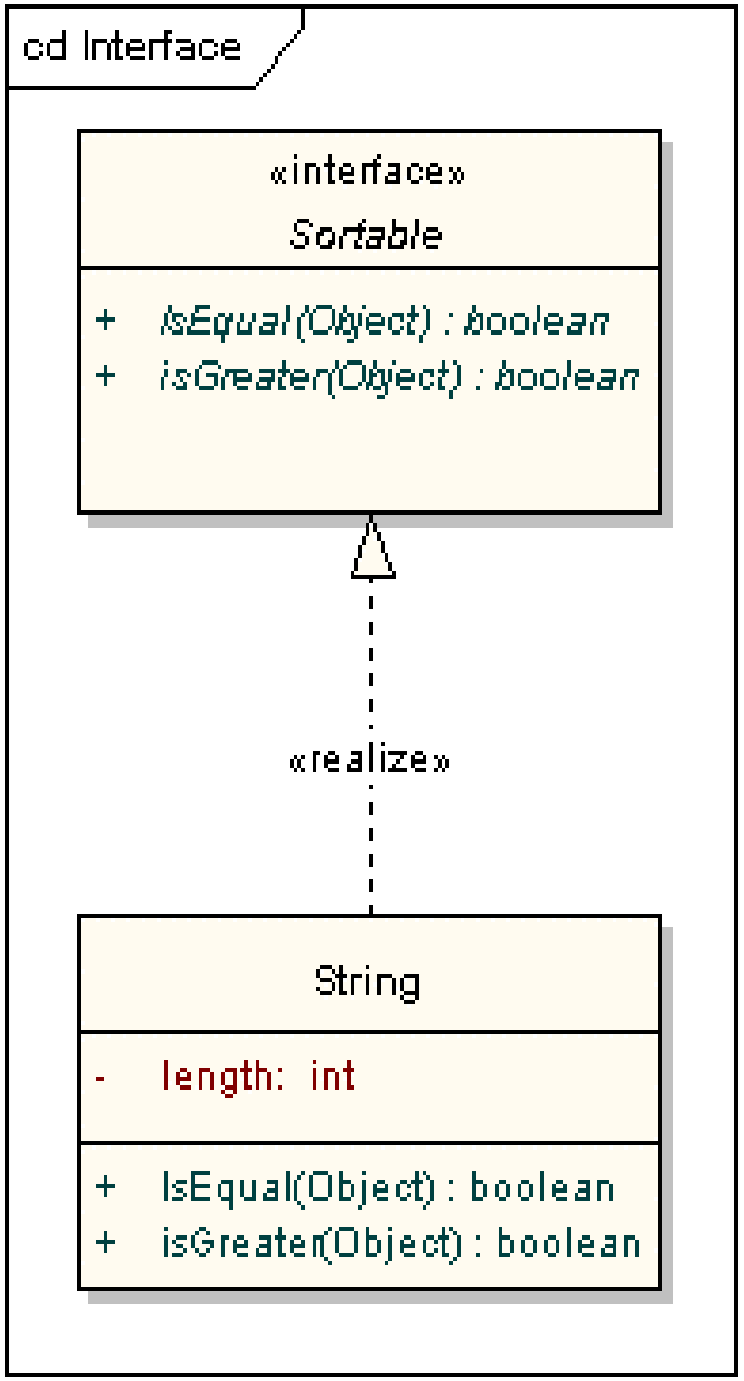
Tabele (table)

- Klasa stereotypowa
- Atrybuty tabeli o stereotypie <<column>>
- Posiada klucz główny (<<PK>> – primary key) obejmujący jedną lub wiele kolumn o unikatowym znaczeniu
- Może posiadać jeden lub wiele kluczy obcych (<<FK>>- foreign key) jako kluczy głównych w powiązanych tabelach po stronie „1”

Zależności (Dependencies)



- Zależności są używane do modelowania powiązań między elementami modelu we wczesnej fazie projektowania, jeśli nie można określić precyzyjnie typu powiązania. Stanowią one wtedy związek użycia **(<<usage>>)**.
- Strałka przerywana wskazuje grotem na klasę, od której coś zależy.
- Później są one uzupełniane o stereotypy: «instantiate», «trace», «import» itp. lub zastąpione innym specjalizowanym połączeniem
- **Implementacja zależności: klasa z operacją jest klasą zależną, natomiast parametr tej operacji jest obiektem typu klasy, od której coś zależy**

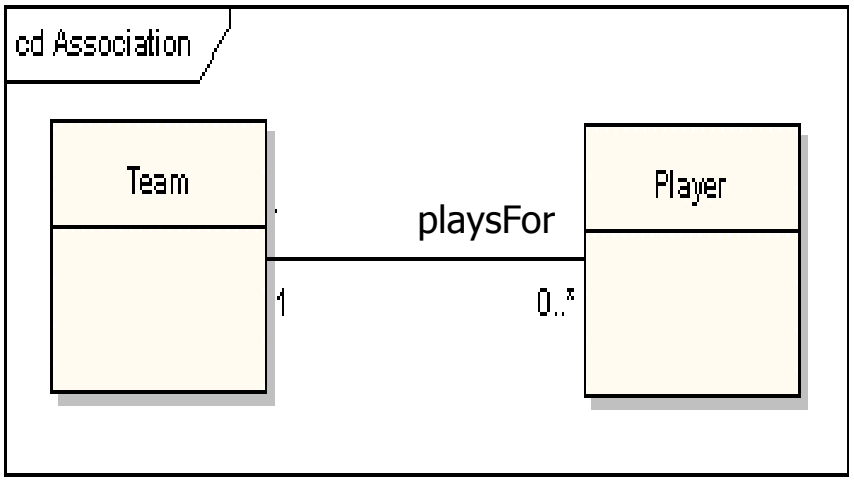


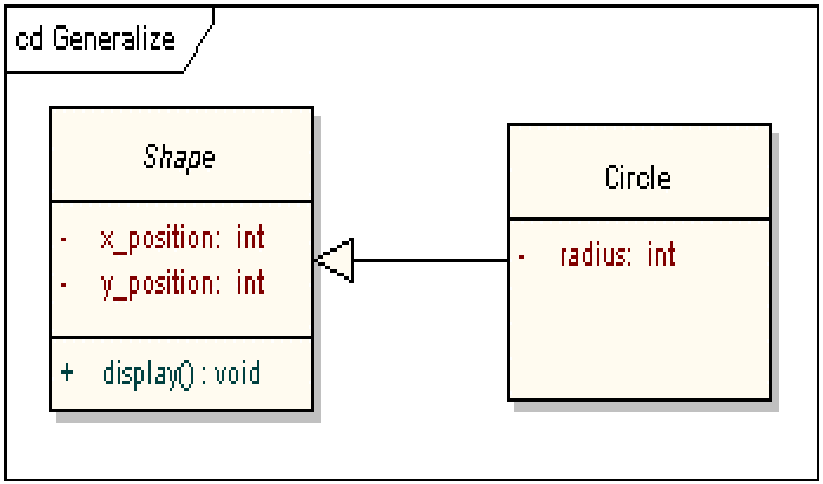
Specjalizacja zależności (Traces)

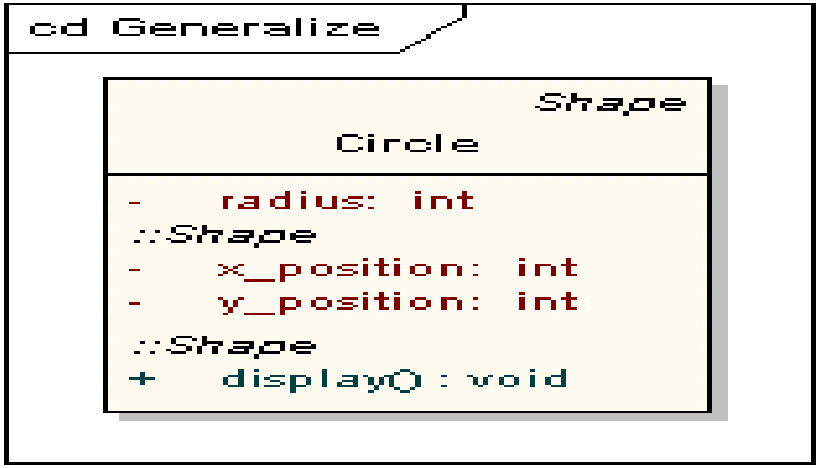
- Łączy elementy modelu o tym samym przeznaczeniu, wymaganiach lub tym samym momencie zmian
- Ma znaczenie informacyjne

Realizacje (Realizations)

- Oznaczane przerywaną strzałką ze stereotypem **<<realize>>:** strzałka wychodzi z klasy implementującej do klasy implementowanej
- Implementacja pewnych właściwości klasy typu *interface*







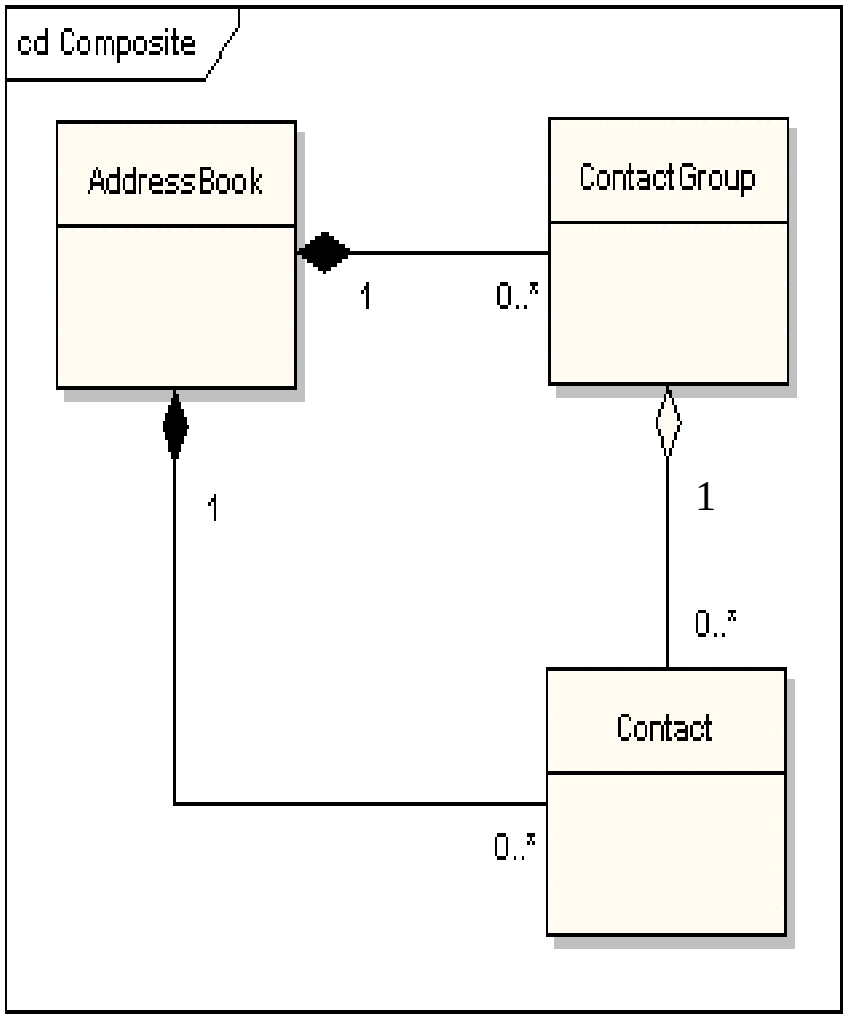
Powiązania (Associations)

- Wiąże dwa elementy modelu
- **Jest implementowana jako:**
  - **relacje wiele do jeden** lub **jeden do jeden: w obiekcie po stronie wiele lub jeden znajduje się referencja do obiektu z przeciwnej strony relacji (strony jeden)**
  - **relacje jeden do wiele: kolekcja referencji instancji obiektów po stronie wiele w obiekcie po stronie jeden** (np. referencja do obiektu typu Team występuje w obiekcie typu Player oraz kolekcja referencji obiektów typu Player w obiekcie klasy Team)
- Połączenie może zawierać nazwy ról na każdym końcu, liczność wystąpień instancji tych elementów, kierunek oraz ograniczenia
- Dla większej liczby powiązanych elementów jest przedstawiana jako romb

Generalizacja (Generalizations)

Używana do oznaczania dziedziczenia

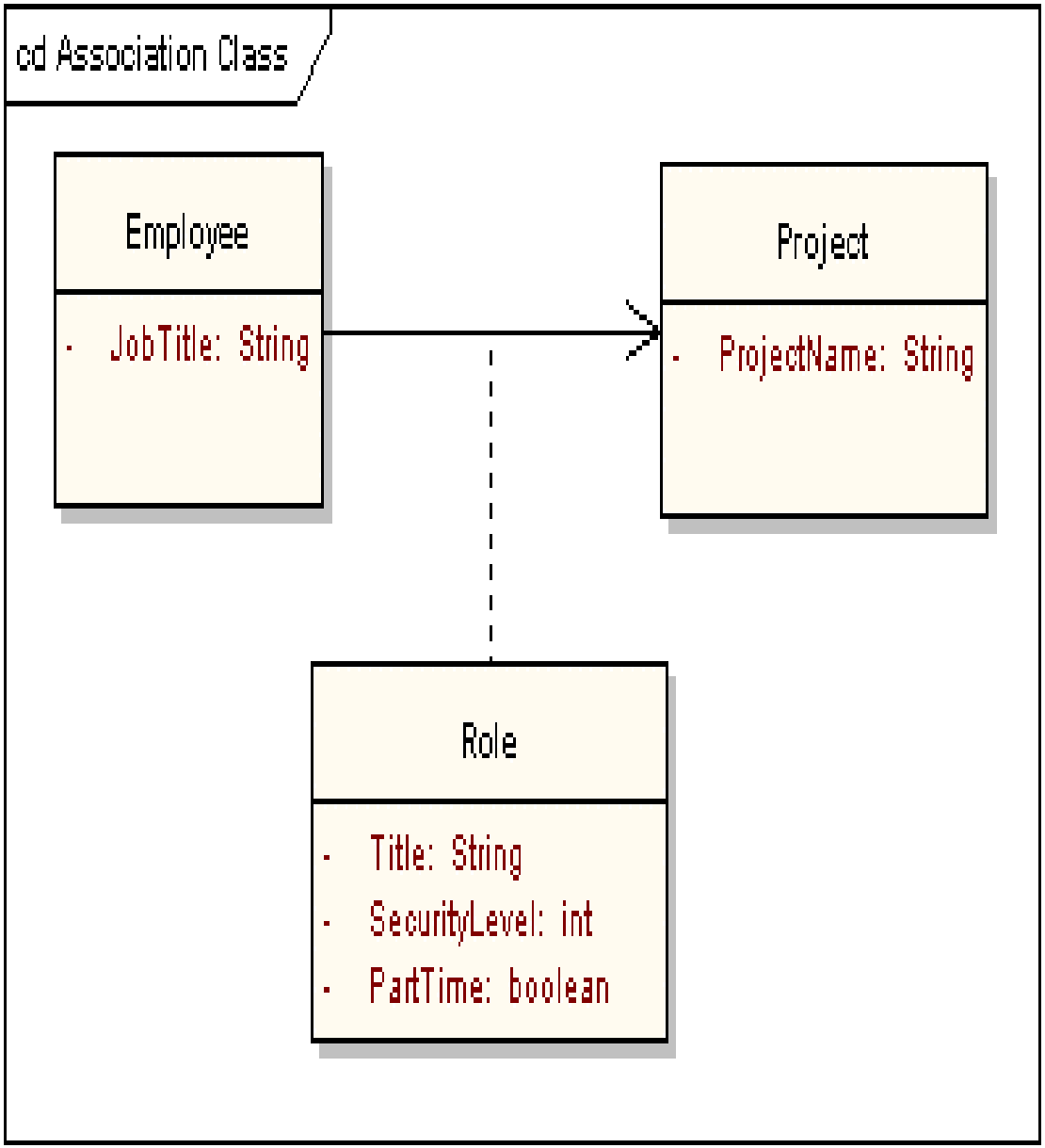
- Strzałka wychodzi z klasy dziedziczącej do klasy, po której dziedziczy
- Np. Klasa Circle dziedziczy atrybuty *x_position, y_position* i metodę display() oraz dodaje atrybut *radius*



Agregacja (Aggregations)

- Oznacza elementy składające się z innych elementów
- Jest tranzytywna, symetryczna,może być rekursywna
- Jest wyrażana za pomocą rombów białych i czarnych, umieszczonych przy klasach agregujących
- Romby czarne - silna agregacja oznaczająca, że przy usuwaniu obiektu klasy agregującej usuwany jest obiekt klasy agregowanej
- Romby białe – słaba agregacja nie pociąga za sobą usuwania z pamięci obiektów agregowanych, gdy usuwany jest obiekt agregujący
- Jest implementowana jak powiązania (associations)

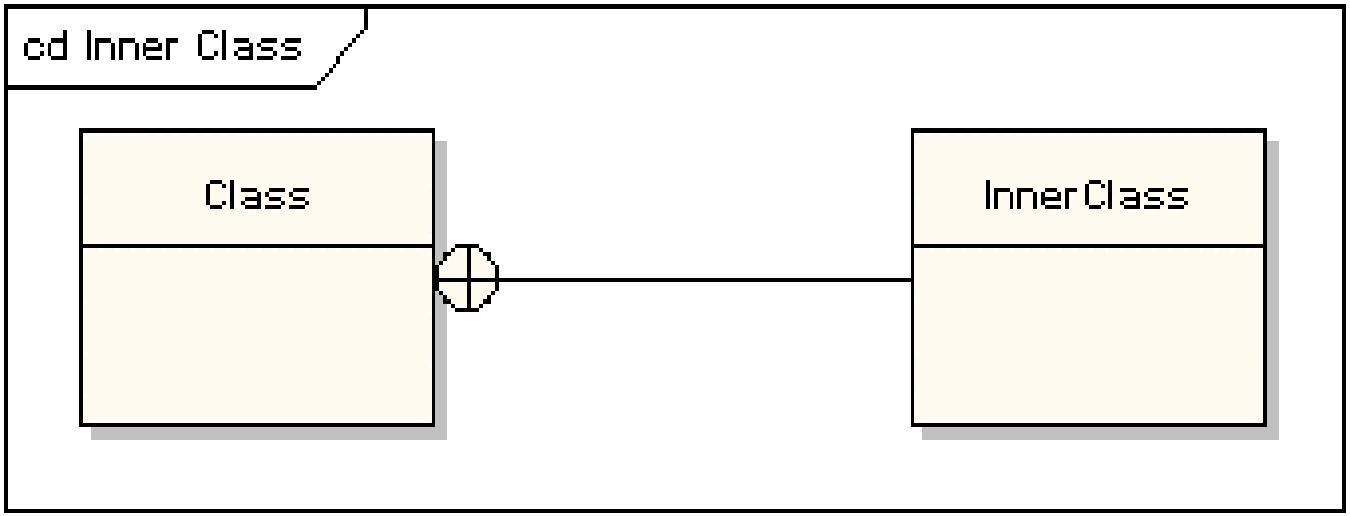
•**Przykład:** agregacja wielu obiektów klasy ***ContactGroup*** *oraz* ***Contact*** w księdze adresowej ***AddressBook*** stanowi silną agregację. Obiekt klasy ***ContactGroup*** agreguje wiele obiektów klasy ***Contact*** w sposób słaby. Usunięcie obiektu klasy *AddressBook* pociąga za sobą usuniecie obiektów klasy ***Contact*** i ***ContactGroup***, usunięcie obiektu klasy ***Contact Gropup*** nie pociąga za sobą usuwania obiektów klasy ***Contact***



Klasy powiązań (Association Classes)

• uzupełnia powiązanie o atrybuty i metody

• np. powiązanie między projektem (obiekt klasy *Project*) a wykonawcą (obiekt klasy *Employee*) dodatkowo jest opisane za pomocą składowych obiektu klasy *Role*. Obiekt klasy *Role* jest przypisany w powiązaniu do jednej pary obiektów klas *Employee* i *Project*, które dodatkowo opisuje jako konkretnego pracownika wykonującego dany projekt



Zagnieżdżenia (Nestings)

•symbol zagnieżdżenia oznacza, że klasa, do której symbol jest dołączony, posiada zagnieżdżoną klasę dołączoną z drugiej strony zagnieżdżenia

•Np. Klasa Class ma zagnieżdżoną klasę *InnerClass*

Diagramy sekwencji (Sequence Diagrams)

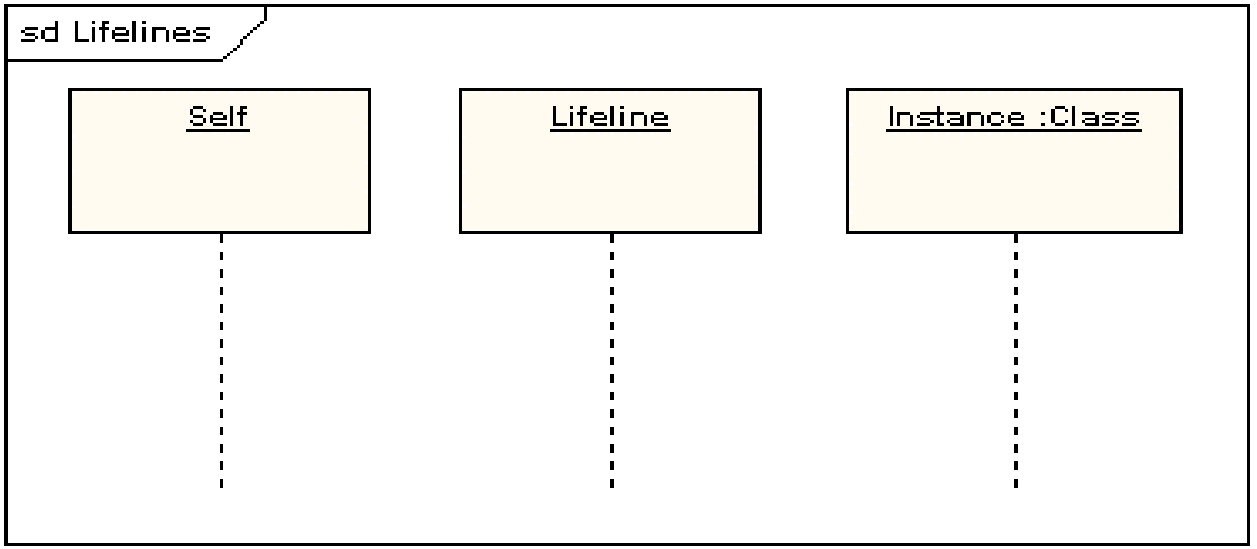
• wyrażają interakcje w czasie (wiadomości wymieniane między obiektami jako poziome strzałki wychodzące od linii życia jednego obiektu i wchodzące do linii życia drugiego obiektu)

•wyrażają dobrze komunikację między obiektami i zarządzanie przesyłaniem wiadomości

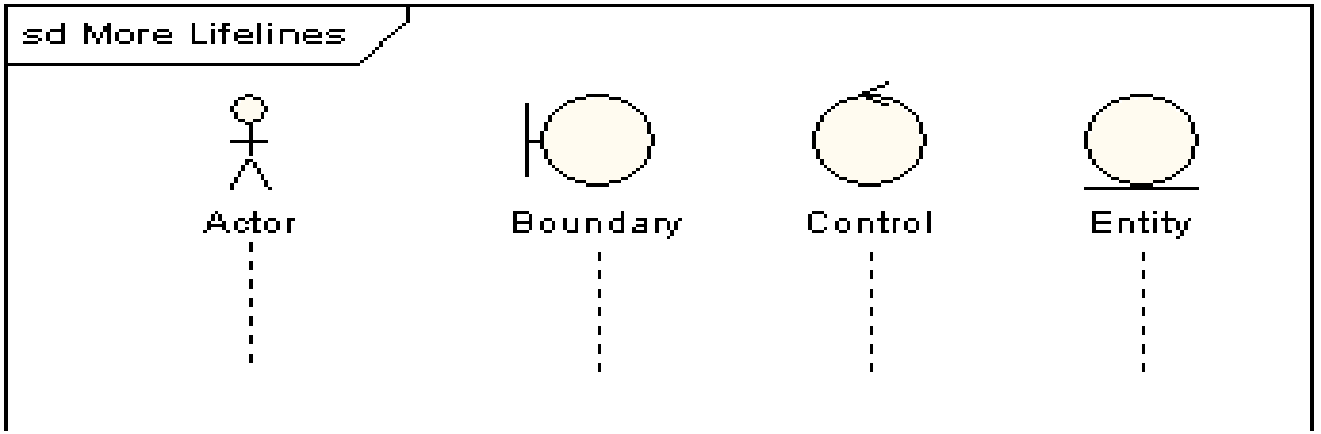
•nie są używane do wyrażania złożonej logiki proceduralnej

Linie życia (Lifelines)

Linie życia reprezentują indywidualne uczestniczenie obiektu w diagramie. Posiadają one często prostokąty zawierające nazwę i typ obiektu.



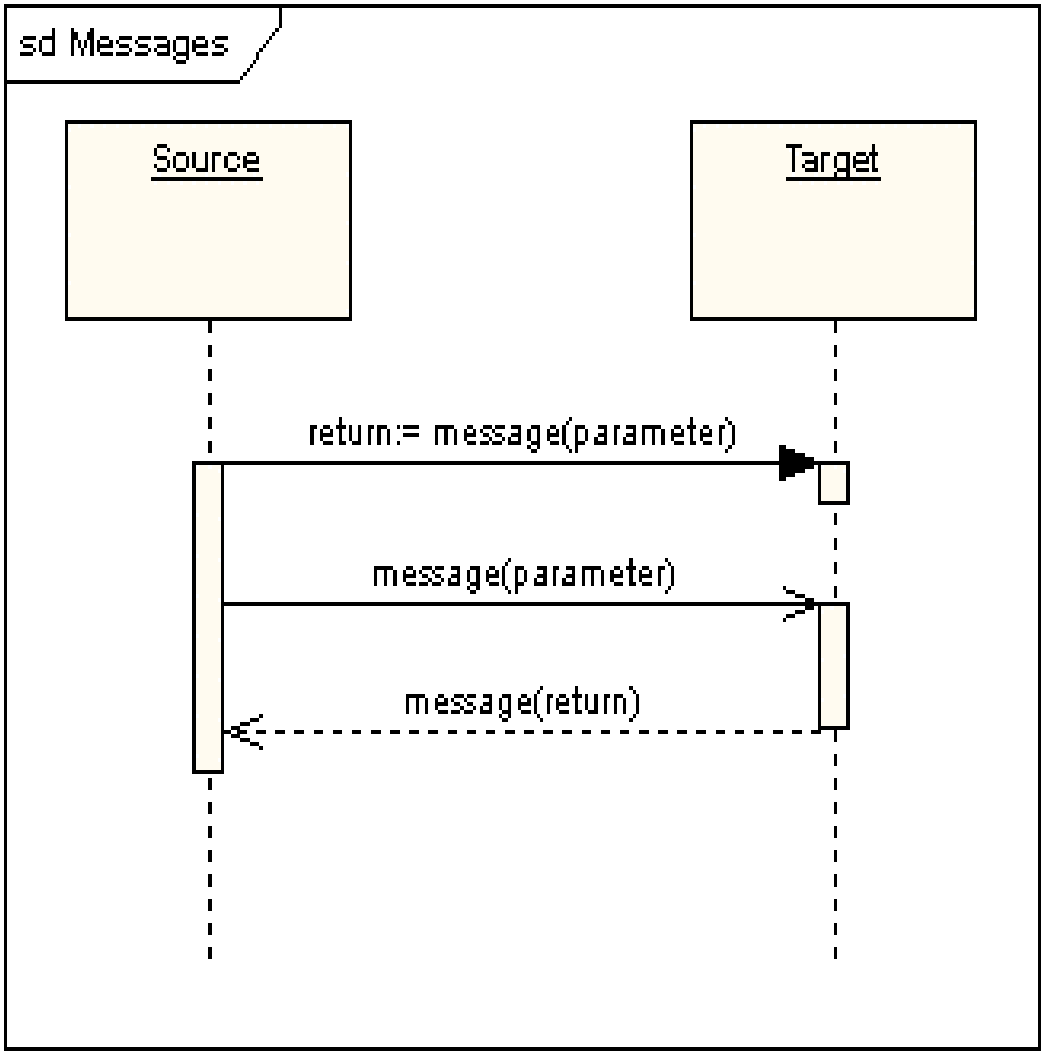
Czasem diagram sekwencji zawiera **linię życia aktora**. Oznacza to, że właścicielem przypadku użycia jest przypadek użycia. Elementy oznaczające ***granice, sterowanie i encje*** mają również swoje linie życia.



Wiadomości (Messages)

- są wyświetlane jako strzałki.
- mogą być *kompletne, zgubione i znalezione;*
- mogą być *synchroniczne i asychroniczne*
- Mogą być typu wywołanie operacji (*call)* lub sygnał (*signal*)
- dla wywołań operacji (call) wyjście strzałki z linii życia oznacza, że obiekt ten wywołuje metodę obiektu, do którego strzałka dochodzi

np.na diagramie pierwsza wiadomość jest synchroniczna, kompletna i posiada return, druga wiadomość jest asynchroniczna), trzecia wiadomość jest asynchroniczna wiadomością typu return (przerywana linia).

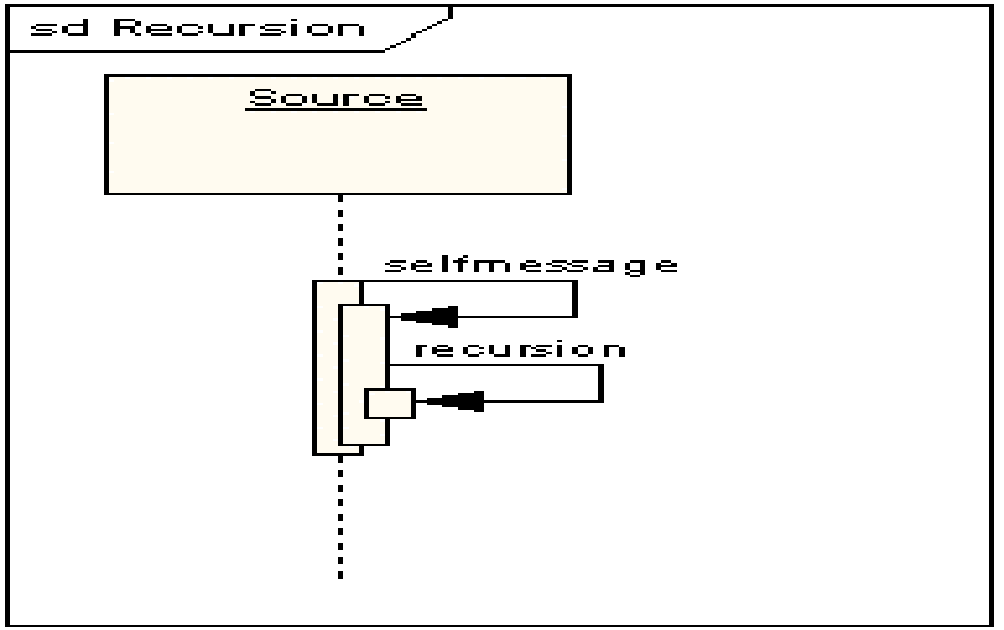


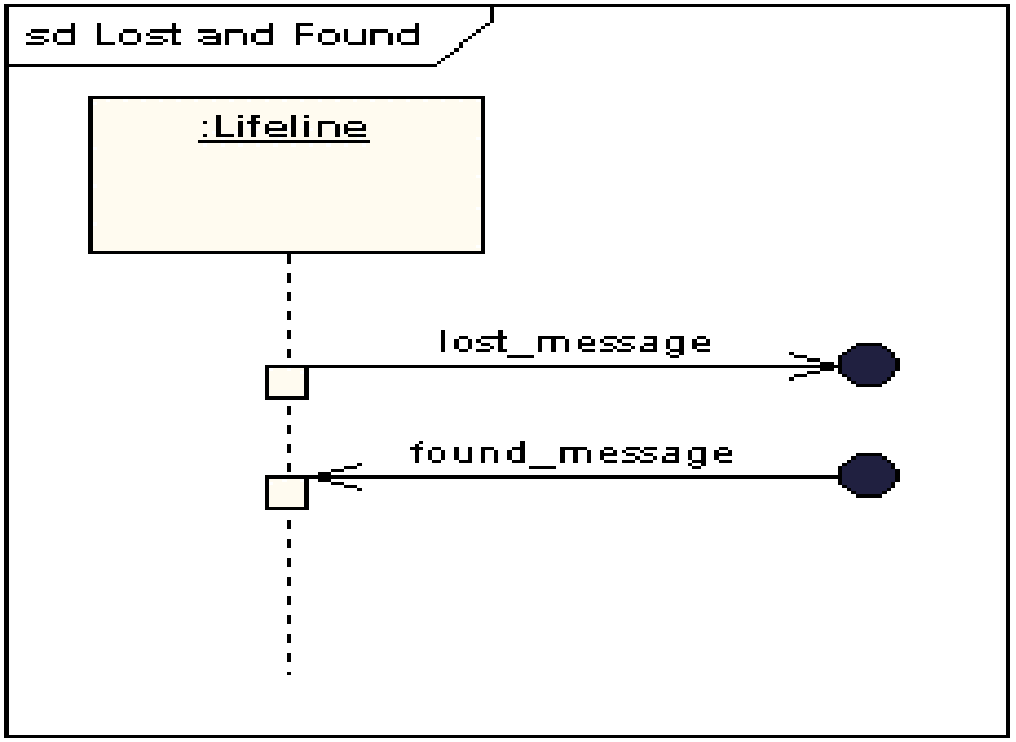
Wykonywanie interakcji (Execution Occurrence)

Cienki prostokąt oznacza aktywność linii życia podczas wykonywania interakcji.
np. Pierwsza linia życia wysyła dwie wiadomości i otrzymuje dwie odpowiedzi: druga linia życia otrzymuje pierwszą wiadomość synchroniczną i wysyła drugą wiadomość jako odpowiedź, potem otrzymuje trzecią wiadomość asynchroniczną, po otrzymaniu której wysyła odpowiedź.

Własne wiadomości (Self Message)
Własne wiadomości reprezentują rekursywne wywoływanie operacji albo jedna operacja wywołuje inną operację należącą do tego samego obiektu.

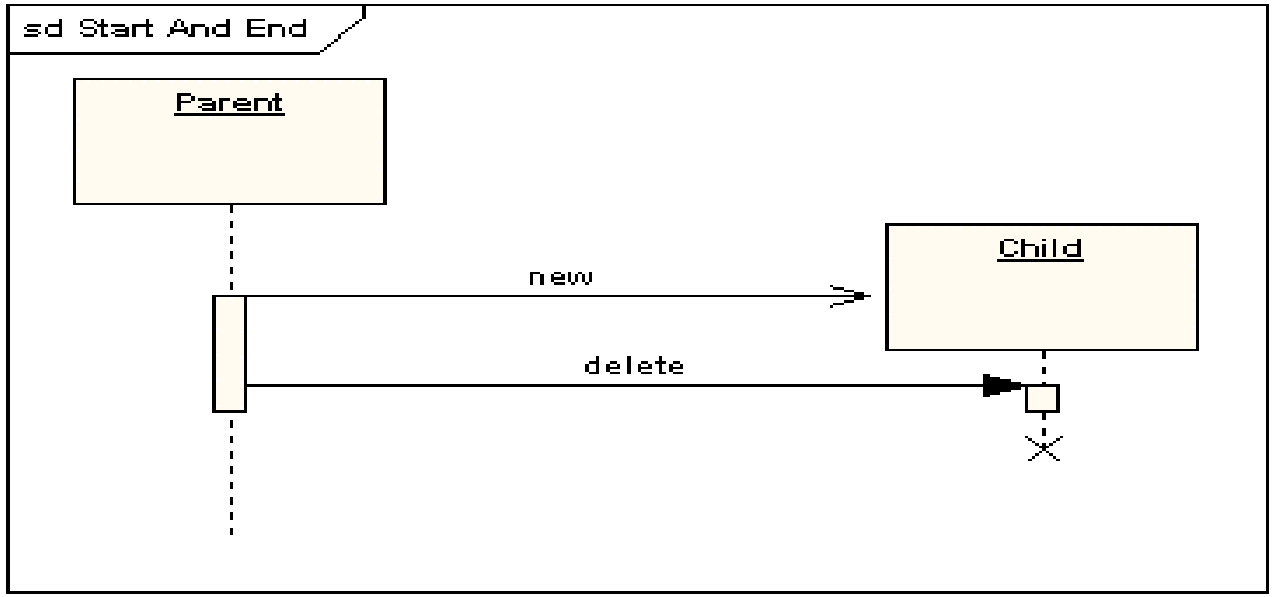
Zgubione i znalezione wiadomości (Lost and Found Messages)
Zgubione wiadomości są wysłane i nie docierają do obiektu docelowego lub nie są pokazane na bieżącym diagramie.
Znalezione wiadomości docierają od nieznanego nadawcy albo od nadawcy nie pokazanego na bieżącym diagramie.





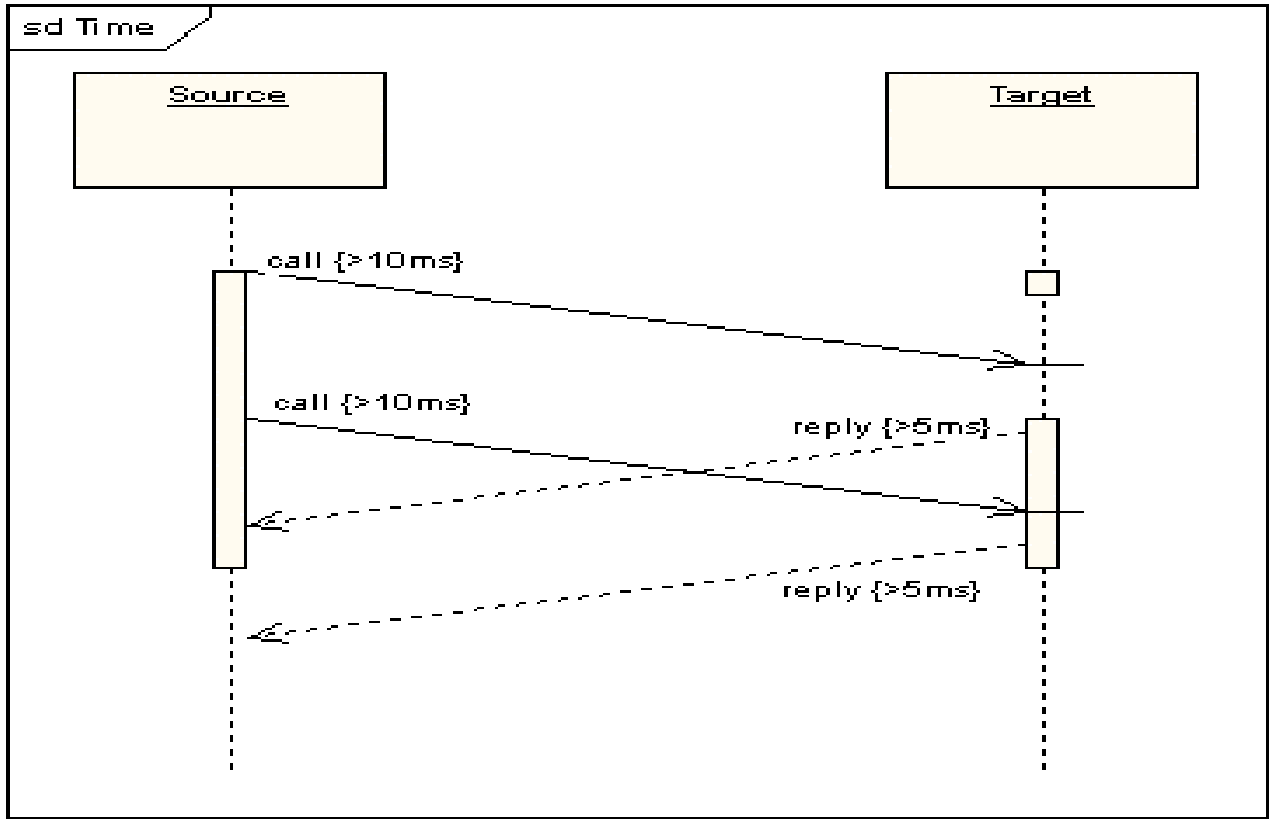
Start linii życia i jej koniec (Lifeline Start and End)

Oznacza to tworzenie i usuwanie obiektu



Ograniczenia czasowe (Duration and Time Constraints)

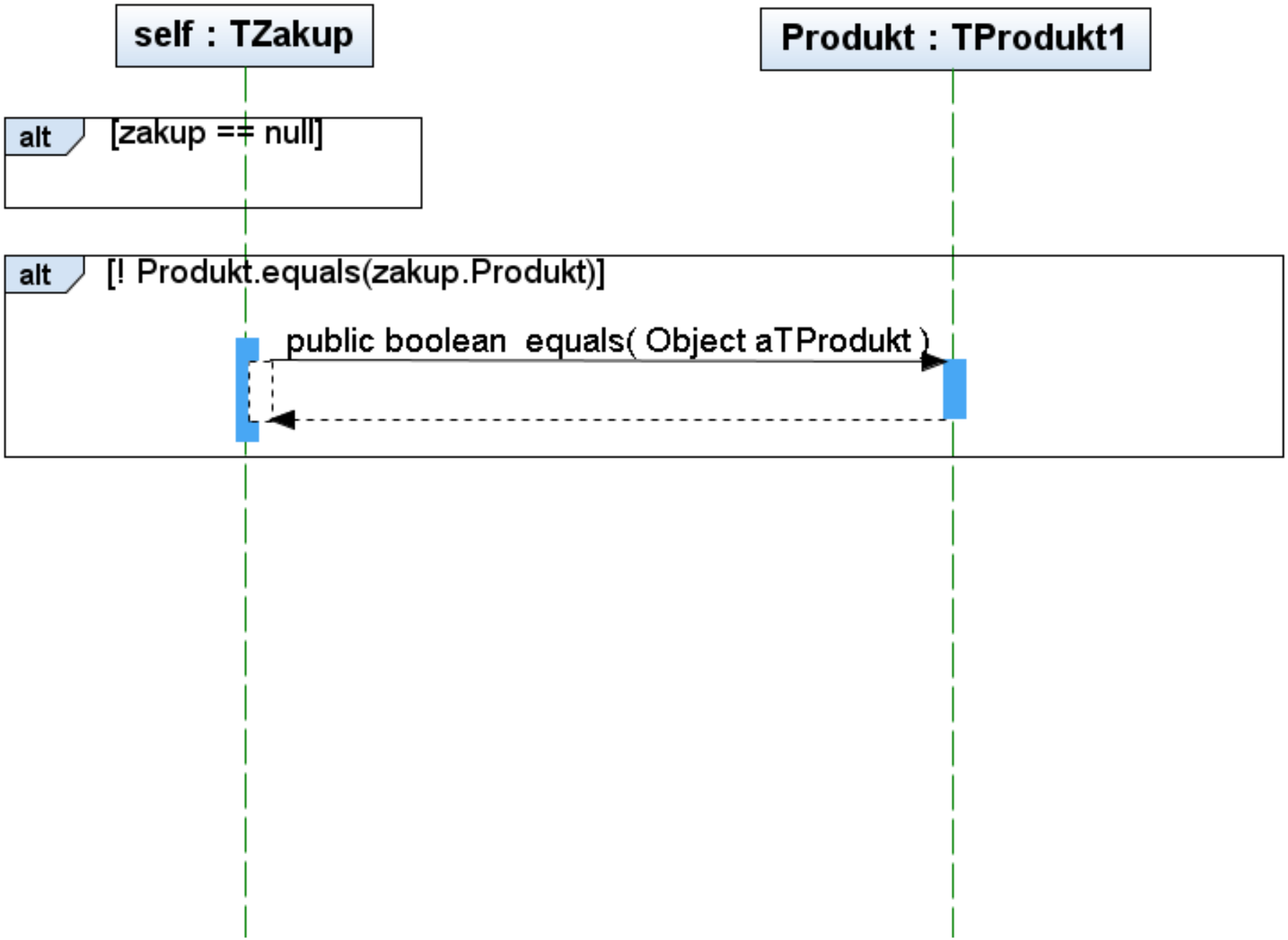
Domyślnie, wiadomość jest poziomą linią. W przypadku, gdy należy ukazać opóźnienia czasu wynikające z czasu podjętych akcji przez obiekt po otrzymaniu wiadomości, wprowadza się ukośne linie wiadomości.

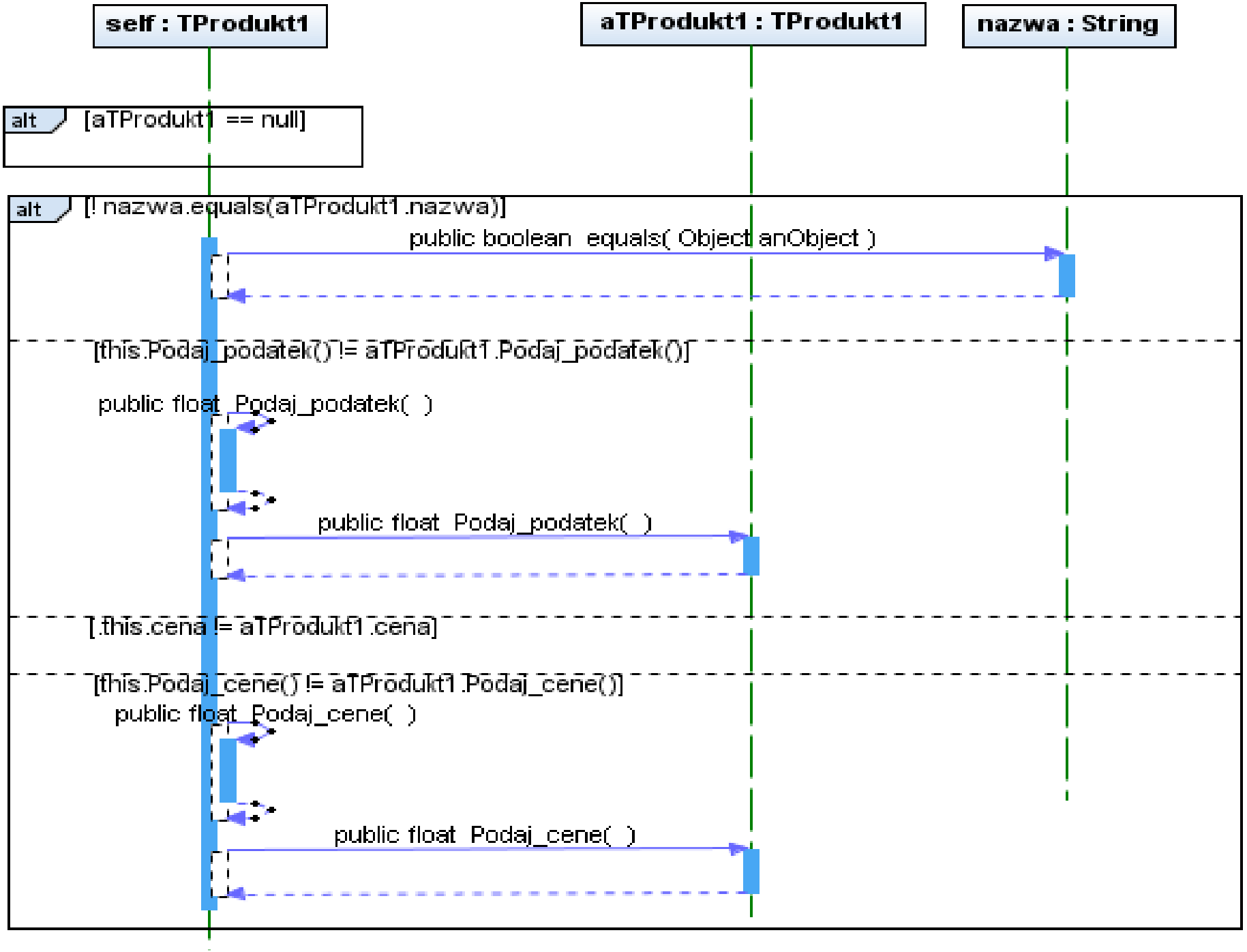


Złożone modelowanie sekwencji wiadomości

Fragmenty ujęte w ramki umożliwiają:

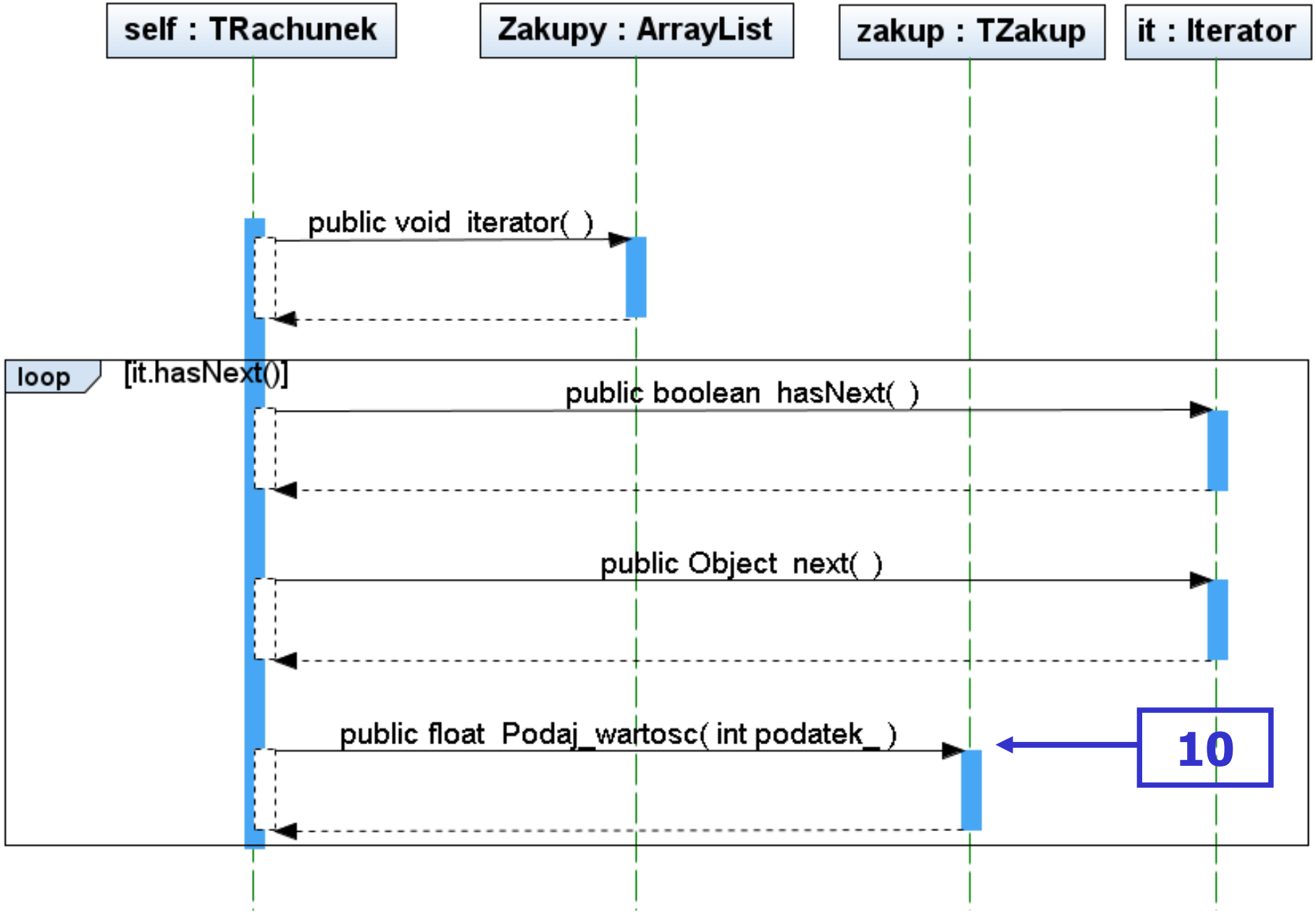
1. **fragmenty alternatywne** (oznaczone "alt") modelują konstrukcje **if…then…else**
2. **fragmenty opcjonalne** (oznaczone "opt") modelują konstrukcje **switch**.
3. **fragment Break** modeluje alternatywną sekwencję zdarzeń dla pozostałej części diagramu.
4. **fragment równoległy** (oznaczony "par") modeluje proces równoległy.
5. **słaba sekwencja** (oznaczona "seq") zamyka pewna liczbę sekwencji, w której wszystkie wiadomości muszą być wykonane przed rozpoczęciem innych wiadomości z innych fragmentów, z wyjątkiem tych wiadomości, które nie dzielą linii życia oznaczonego fragmentu.
6. **dokładna sekwencja** (oznaczona jako "strict") zamyka wiadomości, które muszą być wykonane w określonej kolejności
7. **fragment negatywny** (oznaczony "neg") zamyka pewną liczbę niewłaściwych wiadomości
8. **fragment krytyczny** (oznaczony jako „"critical") zamyka sekcję krytyczną.
9. **fragment ignorowany** (oznaczony jako "ignored") deklaruje wiadomość/ci nieistotne; określa **fragment**, w którym wszystkie wiadomości powinny być ignorowane.
10. **fragment asercji** (oznaczony "assert") eliminuje wszystkie sekwencje wiadomości, które są objęte danym operatorem, jeśli jego wynik jest fałszywy
11. **pętla** (oznaczony "loop") oznacza powtarzanie interakcji w wybranym fragmencie.


self : TZakup
Produkt : TProdukt1
alt
[zakup == null]
alt
[! Produkt.equals(zakup.Produkt)]
public boolean equals(Object aTProdukt)


self : TProdukt1
aTProdukt1 : TProdukt1
nazwa : String
alt
[aTProdukt1 == null]
alt
[! nazwa.equals(aTProdukt1.nazwa)]
public boolean equals(Object anObject)
[this.Podaj_podatek() != aTProdukt1.Podaj_podatek()]
public float Podaj_podatek( )
public float Podaj_podatek( )
[.this.cena != aTProdukt1.cena]
[this.Podaj_cene() != aTProdukt1.Podaj_cene()]
public float Podaj_cene( )
public float Podaj_cene( )

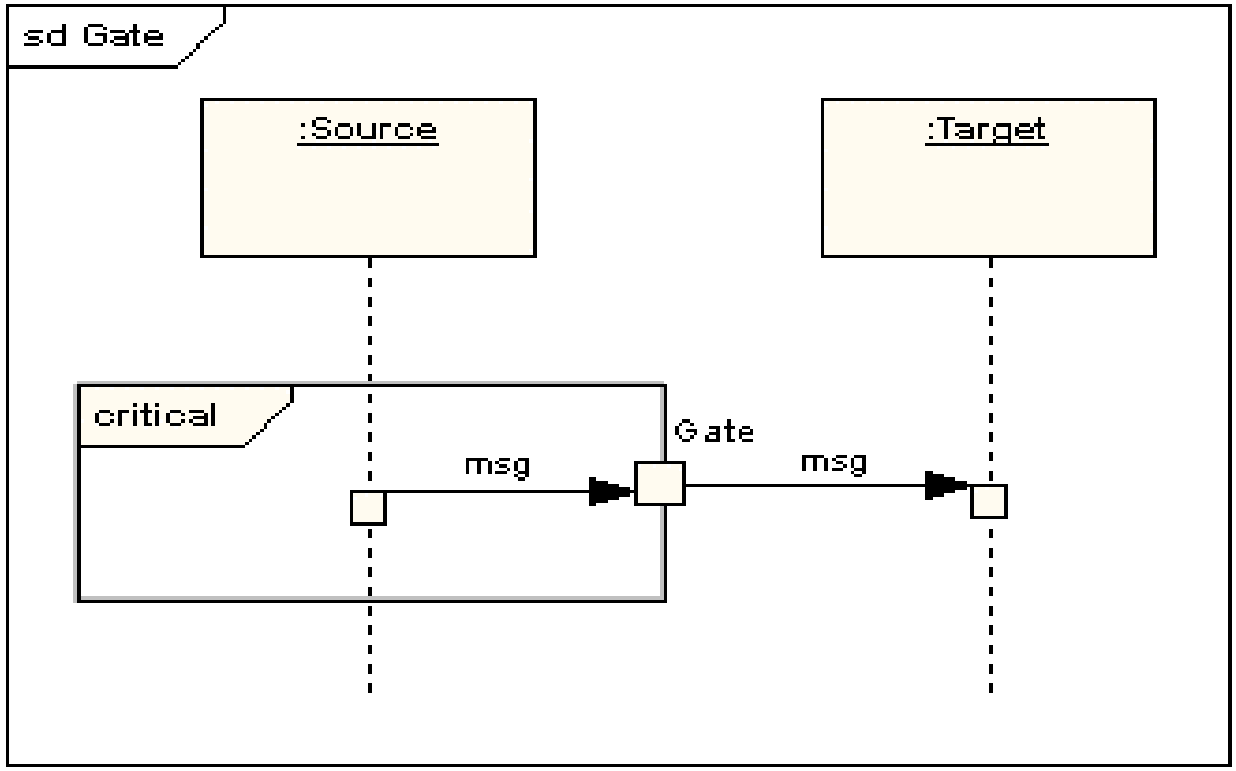
Pętla

Wykonanie w pętli fragmentu diagramu sekwencji



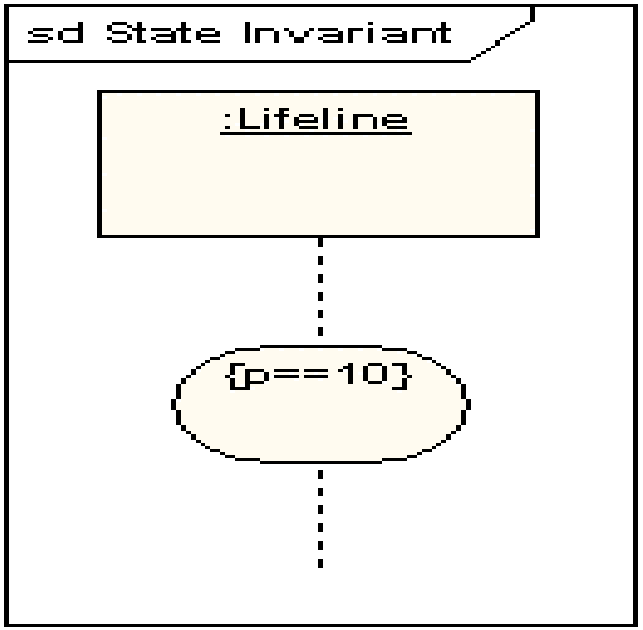
Brama (Gate)

Oznacza przekazywanie wiadomości na zewnątrz między fragmentem i pozostałą częścią diagramu (linie życia, inne fragmenty)



Stan niezmienny lub ciągły (State Invariant / Continuations)

Stan niezmienny jest oznaczany symbolem prostokąta z zaokrąglonymi wierzchołkami.
Stany ciągłe są oznaczone takim samym symbolem, obejmującym kilka linii życia



Dekompozycja (Part Decomposition)

Obiekt ma więcej niż jedną linę życia (obiekt typu :Class). Pozwala to pokazać zagnieżdżone protokoły przekazywanych wiadomości np. wewnątrz obiektu i na zewnątrz