Diagramy czynności i syntaktyka diagramów klas
Wykład3

Zofia Kruczkiewicz

Składnia elementów na diagramach UML

1. W prezentacji składni diagramów czynności UML (str. 7-18) oraz diagramów klas (str 40-53) o charakterze tutorialowym sposób definiowania składowych klas (aktywności, akcji odwzorowanych na elementy klas, itd oraz atrybuty, operacje, parametry operacji) jest jednym z przyjętych sposobów interpretowania specyfikacji języka UML w tutorialach – często odbiegająca od syntaktyki znanych języków obiektowych (Java, C++) i zazwyczaj uproszczona.
2. W prezentacji przykładów diagramów aktywności UML (str. 21, 26, 27, 32) oraz diagramów klas (65-66) sposób definiowania elementów odwzorowanych na składowe klas jest jednym z kolejnych przyjętych sposobów interpretowania specyfikacji języka UML w narzędziach UML. Składnia tych diagramów różni się od prezentowanych w tutorialach (p.1) i jest zbliżona do składni języka Java.

Wniosek: W wielu narzędziach UML sposób definiowania elementów diagramów oparty na tej samej specyfikacji UML różni się. W prezentowanych materiałach przedstawiono te różnice, stosując dwa różne sposoby definiowania oparte na:

1) tutorialach (p.1): **https://sparxsystems.com/resources/tutorials/uml2/index.html**
2) narzędziu z serii Visual Paradigm CE, VP CE (p. 2: **http://zofia.kruczkiewicz.staff.iiar.pwr.wroc.pl/wyklady/IO_UML/Instrukcja_1_2.pdf**.

Diagramy czynności

1. **Diagramy czynności UML**
2. **Przykład diagramów czynności UML – modelowanie przepływu czynności**

   (wg Booch G., Rumbaugh J., Jacobson I., UML przewodnik użytkownika)
3. **Przykład diagramów czynności UML – modelowanie operacji**

   [Booch G., Rumbaugh J., Jacobson I., UML przewodnik użytkownika]

Diagramy czynności

1. Diagramy czynności UML

Diagramy UML 2 – część druga

Na podstawie

UML 2.0 Tutorial

https://sparxsystems.com/resources/tutorials/uml2/activity-diagram.html

Dwa rodzaje diagramów UML 2

Diagramy UML <u>modelowania strukturalnego</u>

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

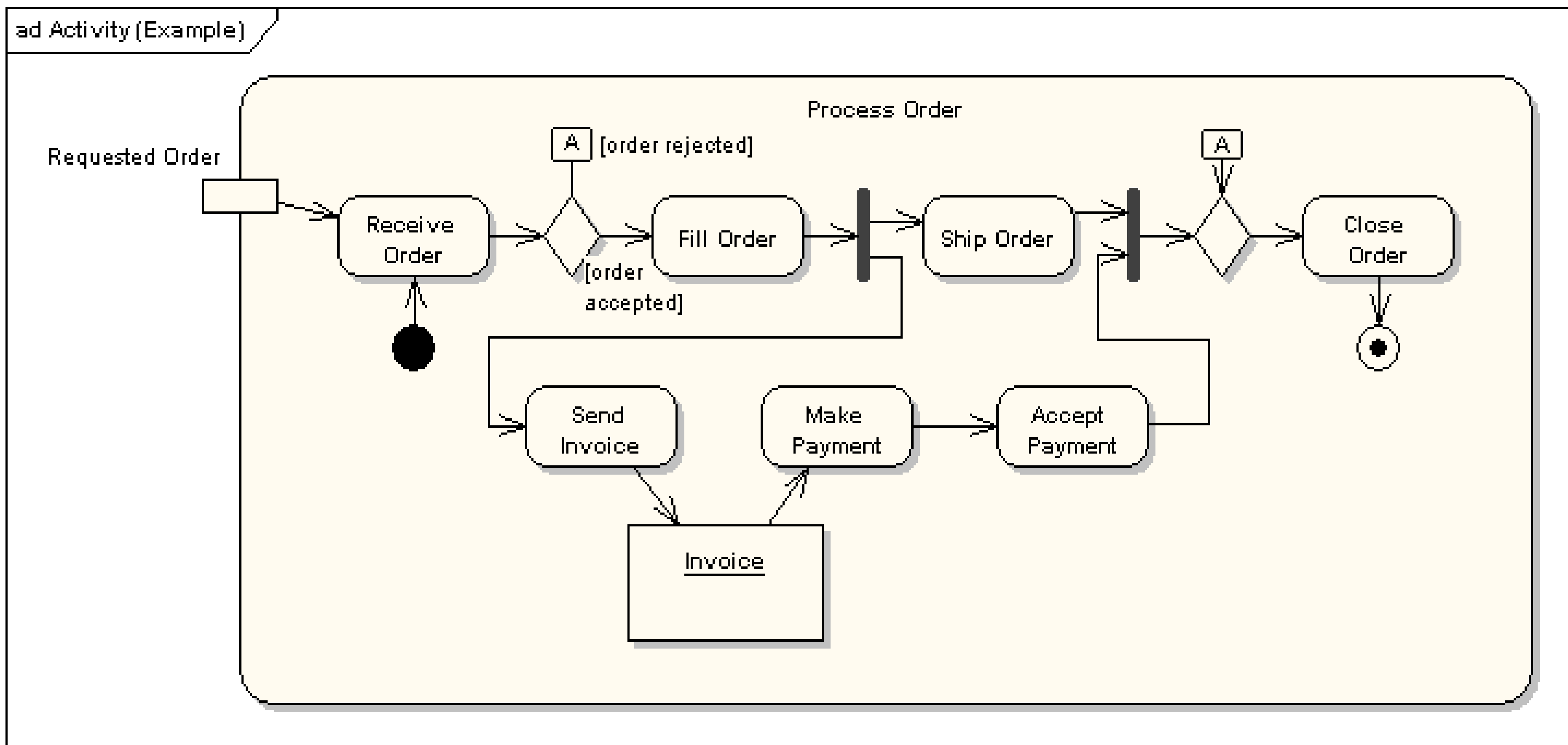
Diagramy UML <u>modelowania zachowania</u>

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

Diagramy czynności

<u>Diagram czynności</u> opisuje interakcje między obiektami:

- **jak** pobierane są operacje,
- **co** operacje wykonują (zmiana stanu obiektu),
- **kiedy** operacje są wykonywane (sekwencje czynności lub akcji)
- **gdzie** są wykonywane.



Czynność- zawiera specyfikację sparametryzowanych zachowań:

- akcje
- wierzchołki sterujące
- obiekty
- przepływ sterowania

Akcja – elementarny krok czynności

ad Conditions

«localPreCondition»
{A drink is selected that the vending machine contains}

Dispense Drink

«localPostCondition»
{The vending machine dispensed the drink selected}

Ograniczenia akcji:

Warunki przed akcją i po akcji

np.

Warunek przed: Wybór napoju w automacie, jeśli istnieje

Stan: akcja wydania napoju (*Dispense drink*)

Warunek po: Maszyna wydała wybrany napój

ad Activity Edge

Send Payment

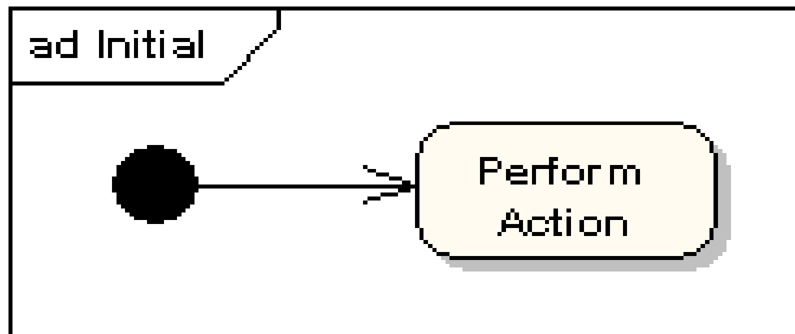
Accept Payment

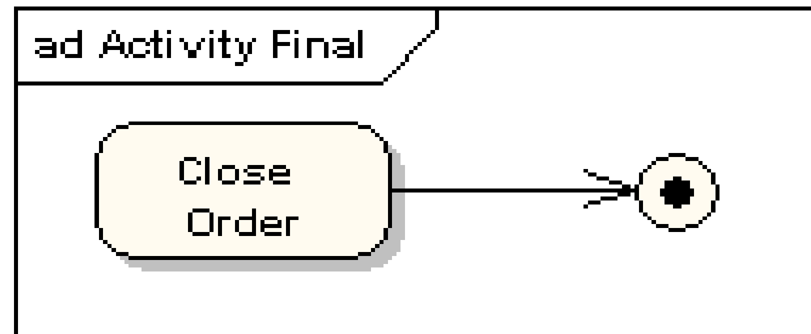
Przepływ sterowania:

Przejście z jednej akcji aktywności do akcji drugiej aktywności

np.

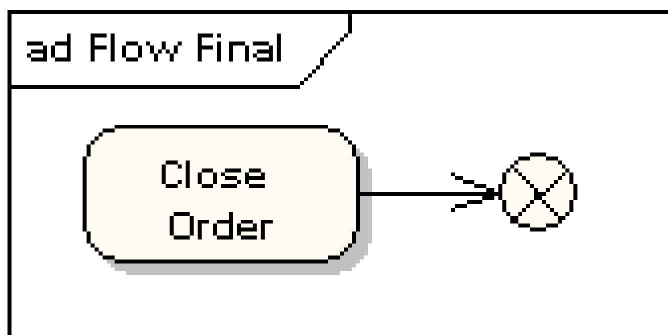
wysłanie opłaty (*Send Payment*) i akceptacja opłaty (*Accept Payment*)





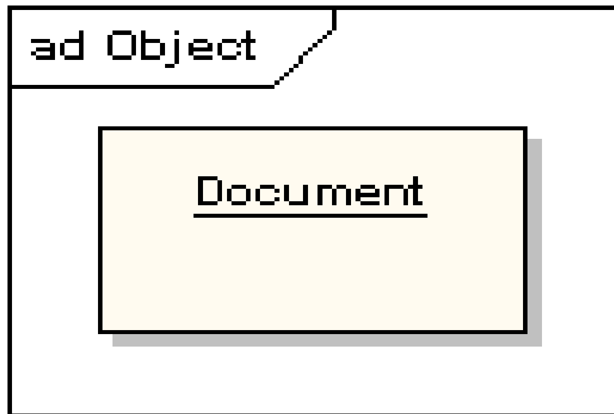
Wierzchołek początkowy (*Initial*) wielu przebiegów sterowania

Wierzchołek końcowy (*Final*) wielu przebiegów sterowania związanych z jedną czynnością

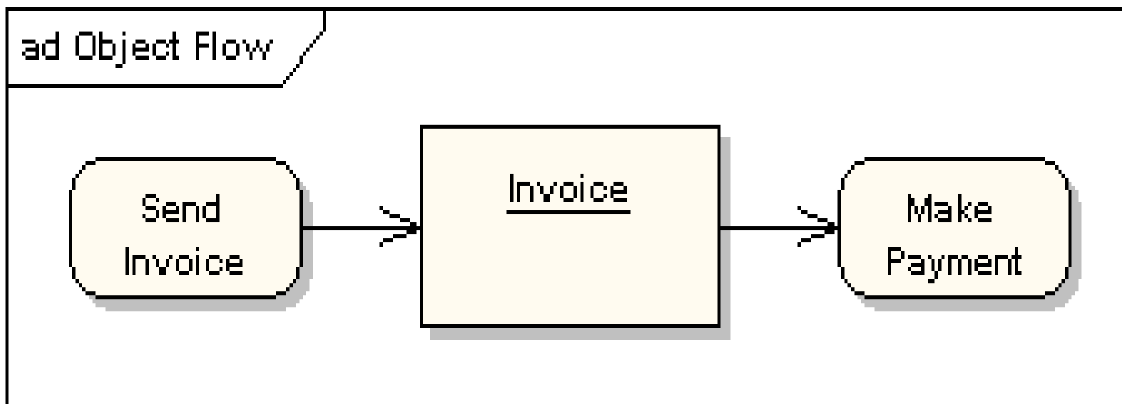


Wierzchołek końca sterowania (*Flow Final*) koniec pojedynczego przebiegu sterowania

Obiekt

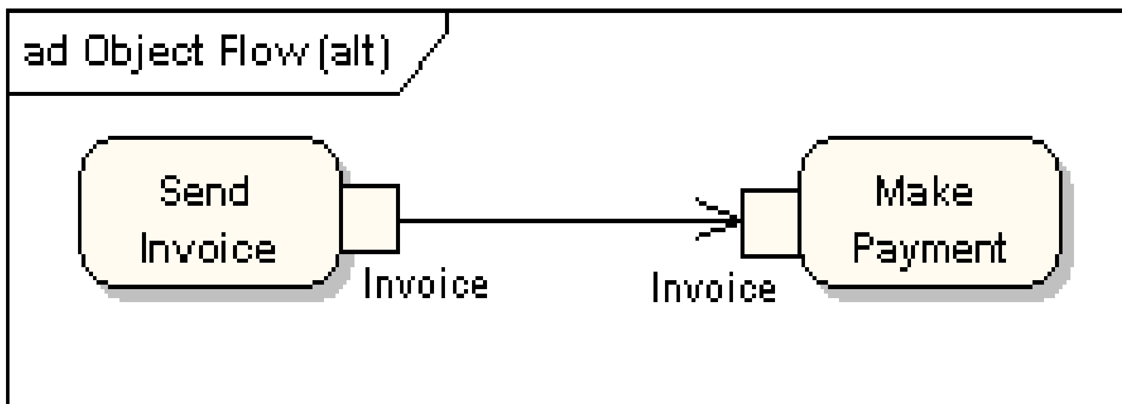


Magazyn danych



Przepływ obiektów

Wysłanie (*Send Invoice*) obiektu Faktura (*Invoice*) w celu dokonania opłaty (*Make Payment*)

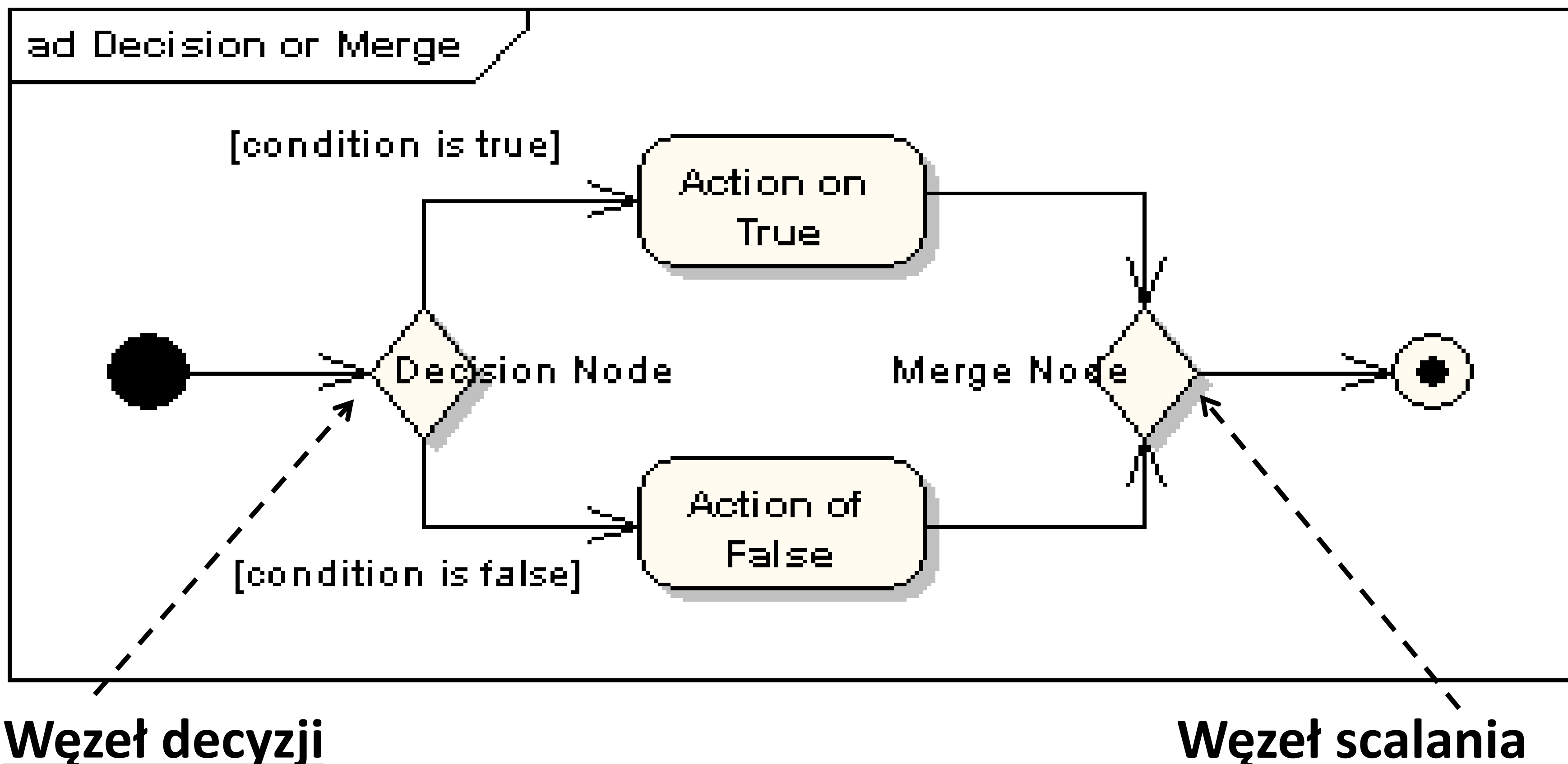


Przepływ obiektów

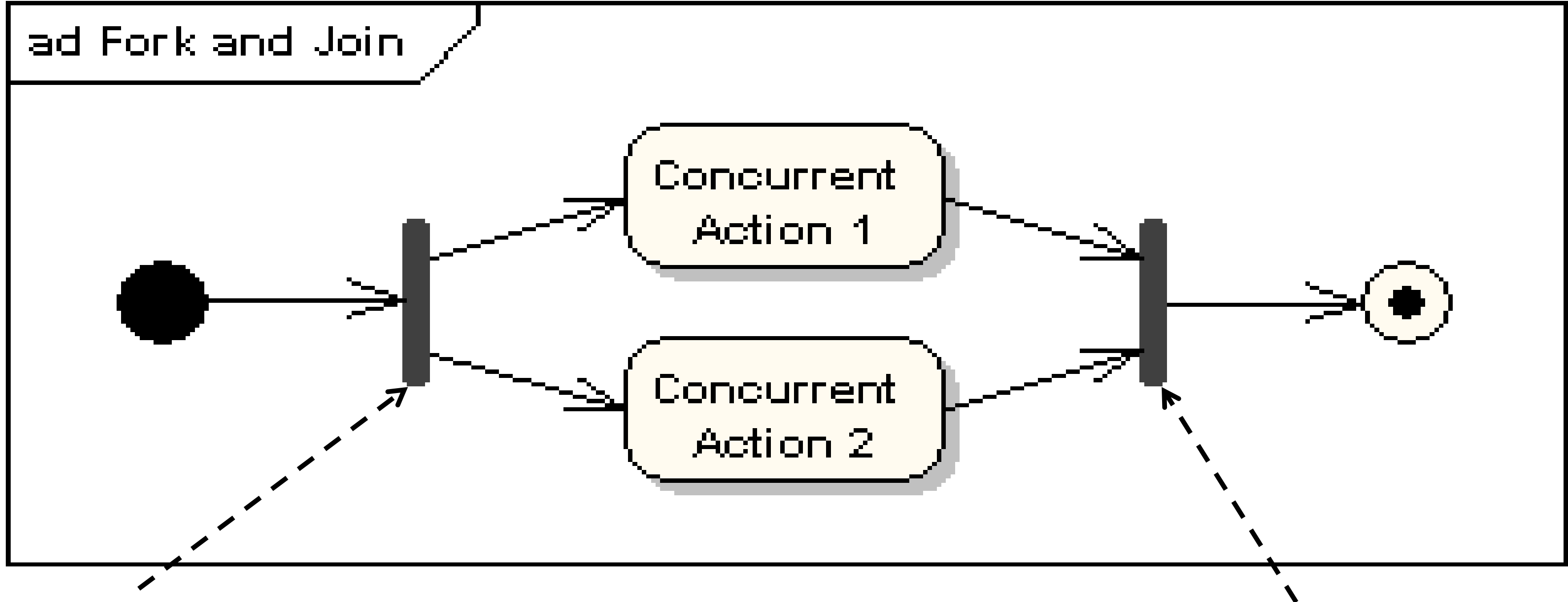
Równoważny diagram

Węzły decyzji i scalania

Wybór przepływu sterowania w węźle decyzji (*Decision Node*) po zbadaniu warunku (*condition*) i scalenie z przepływem sterowania znajdującym się za węzłem scalania (*Merge Node*)



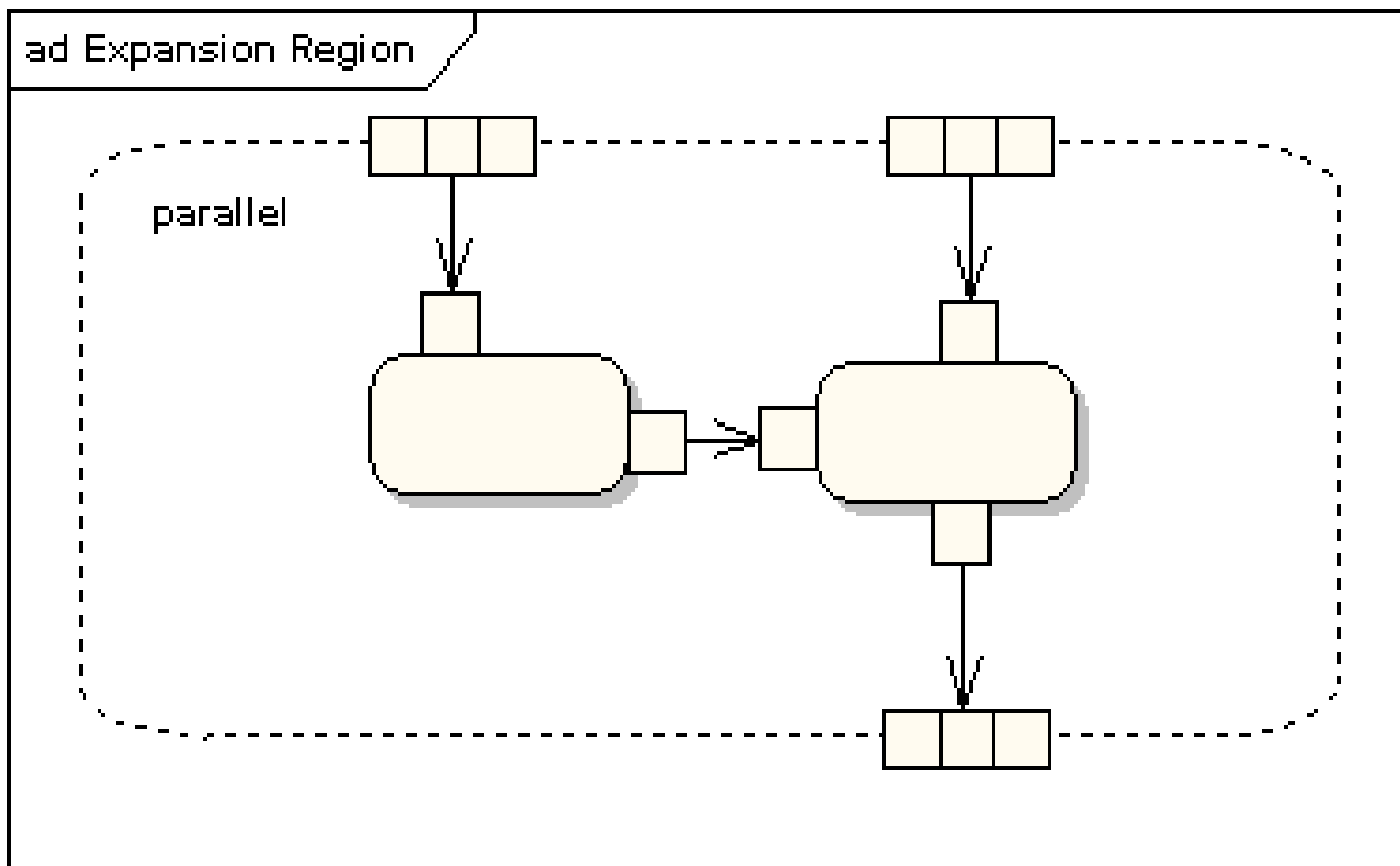
Węzły rozdzielania i łączenia



Węzeł rozdzielania (*Fork – pionowa lub pozioma linia*) **przepływu sterowania** na kilka współbieżnie działających przepływów sterowania

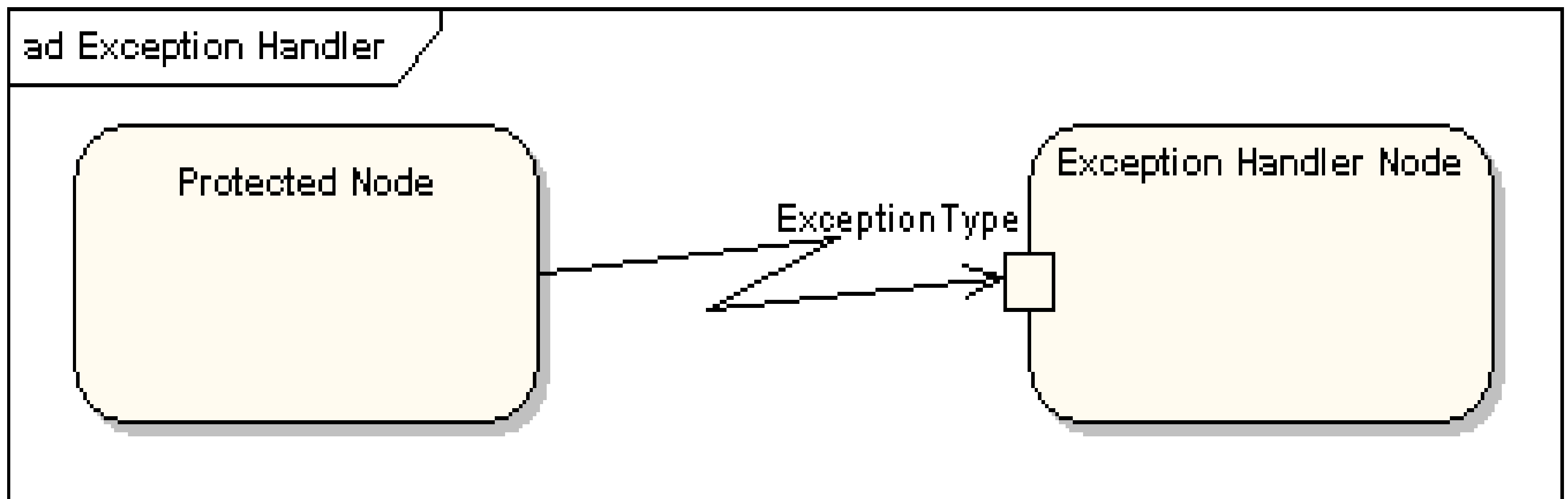
Węzeł łączenia (*Join - pionowa lub pozioma linia*) **współbieżnie działających przepływów sterowania do jednego przepływu sterowania** –po zakończeniu każdego z tych współbieżnych procesów

Region rozszerzający - powtarzanie czynności: iteracyjnie (*iterative*), równolegle (*parallel*) lub w postaci strumienia (*stream*) – nazwa sposobu wykonania regionu diagramu czynności. Węzły rozszerzenia wejścia i wyjścia są rysowane jako grupa trzech pól reprezentujących wielokrotny wybór elementów.

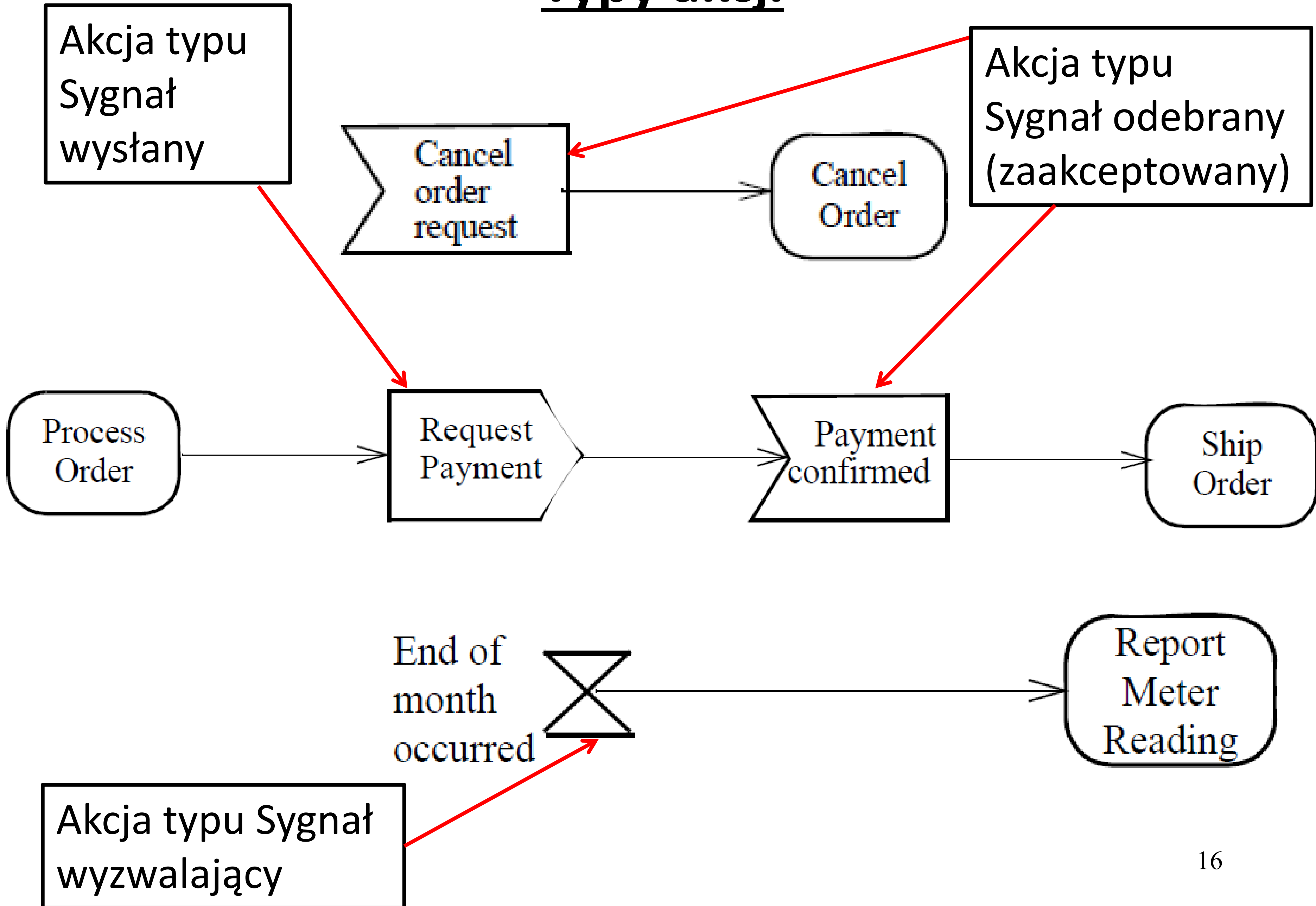


Obsługa wyjątków

Np. Reakcja na błąd podczas wykonania akcji w czynności „*Protected Node*” – nastąpi przerwanie tych akcji i przejście do wykonania akcji w czynności „*Exception Handler Node*”

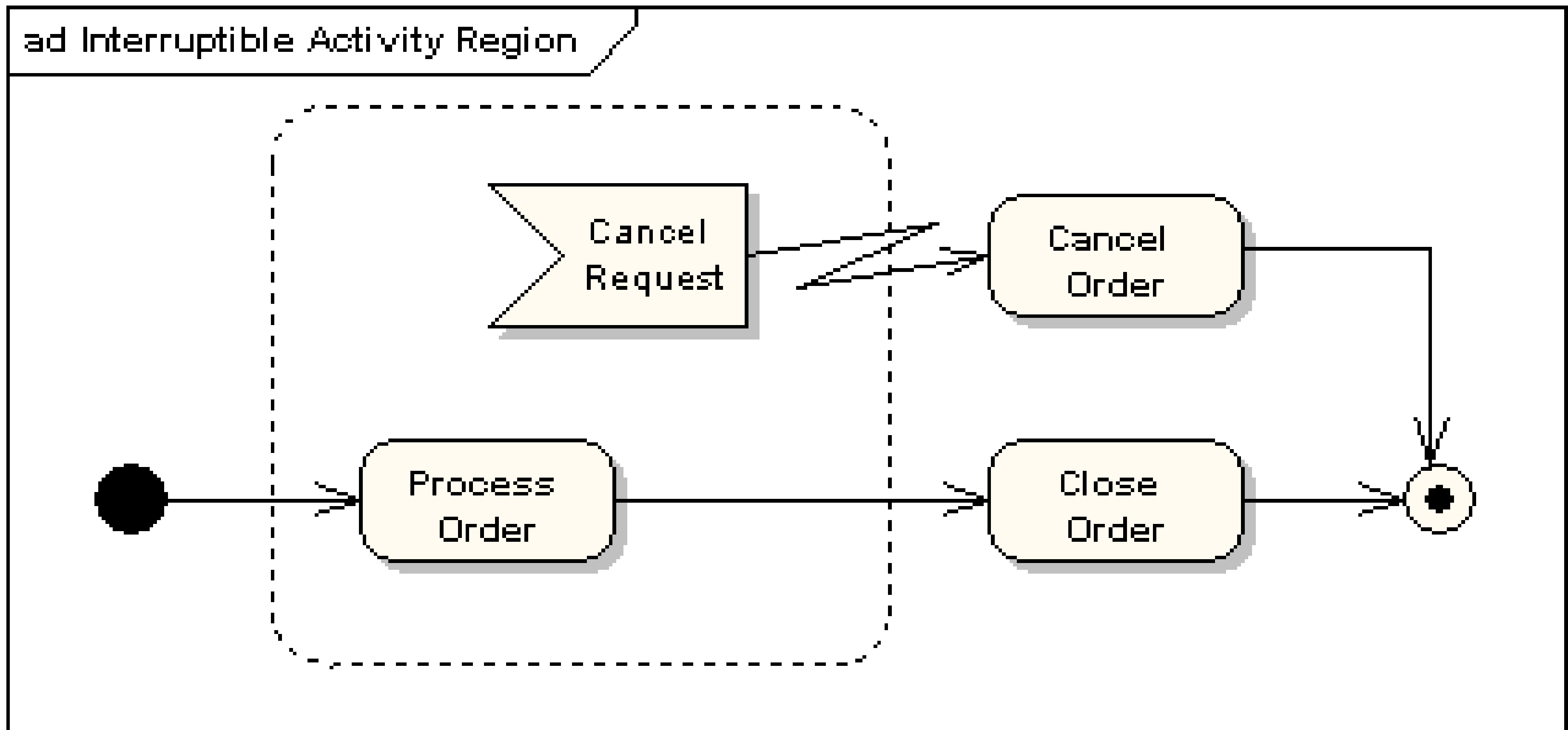


Typy akcji

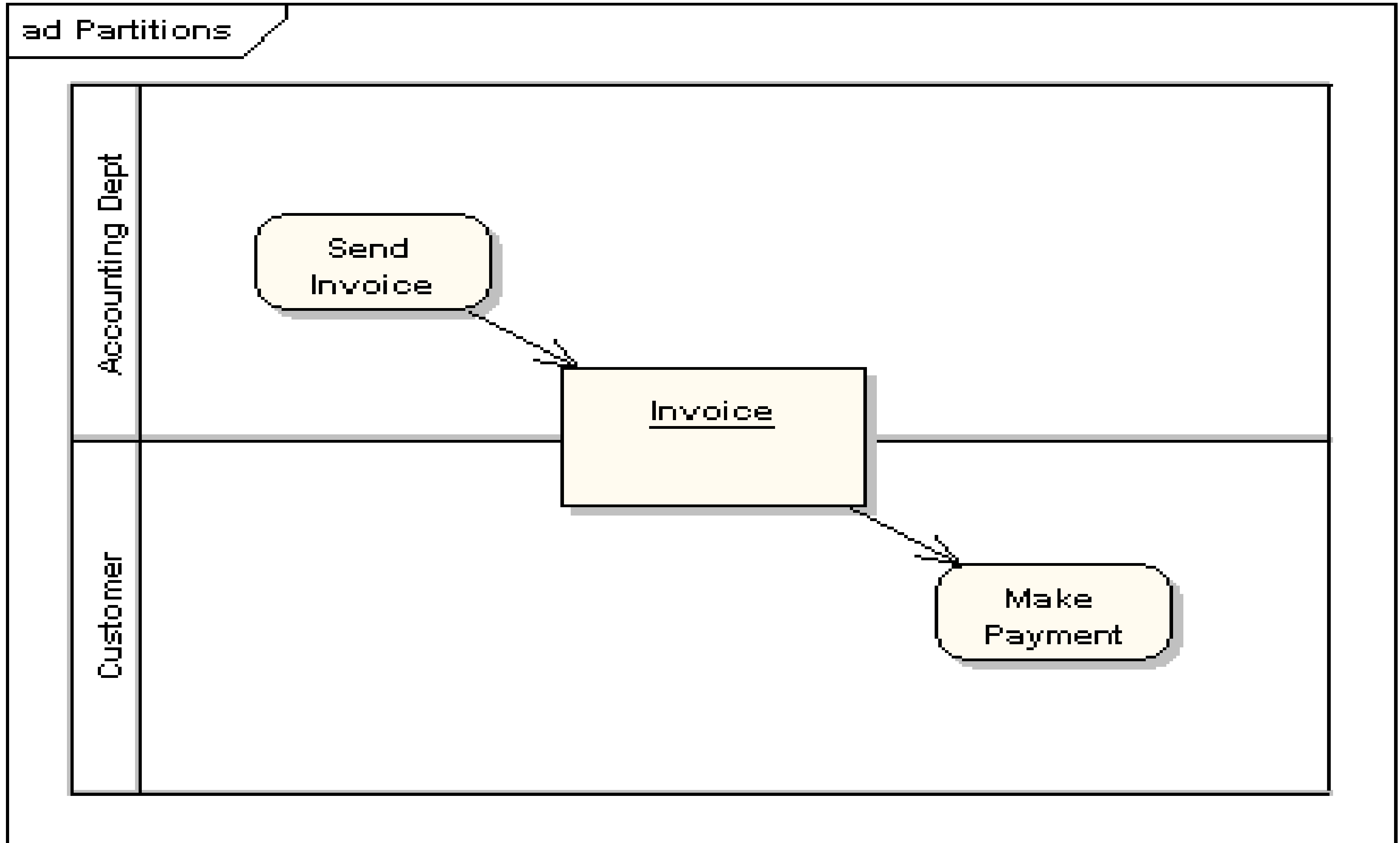


Obsługa przerwań reakcja na inną akcję niż typowa

Np. Akcja "*Process Order*" aktywności zostanie wykonana i następnie można przejść do kolejnej czynności i wykonać akcję „*Close Order*” i zakończyć proces. Jednak w przypadku, gdy podczas akcji „*Process Order*” nastąpi przerwanie „*Cancel Request*”, zostanie wykonana akcja „*Cancel Order*” w innej czynności i nastąpi zakończenie procesu.



<u>Partycje(tory)</u> – np. podział czynności wykonywanych na obiekcie *Faktura* (*Invoice*) przez dwa różne obiekty reprezentowane przez partycje: *Wydział Finansowy* (*Accounting Department*) i *Klient* (*Customer*).



Diagramy czynności

1. Diagramy czynności UML

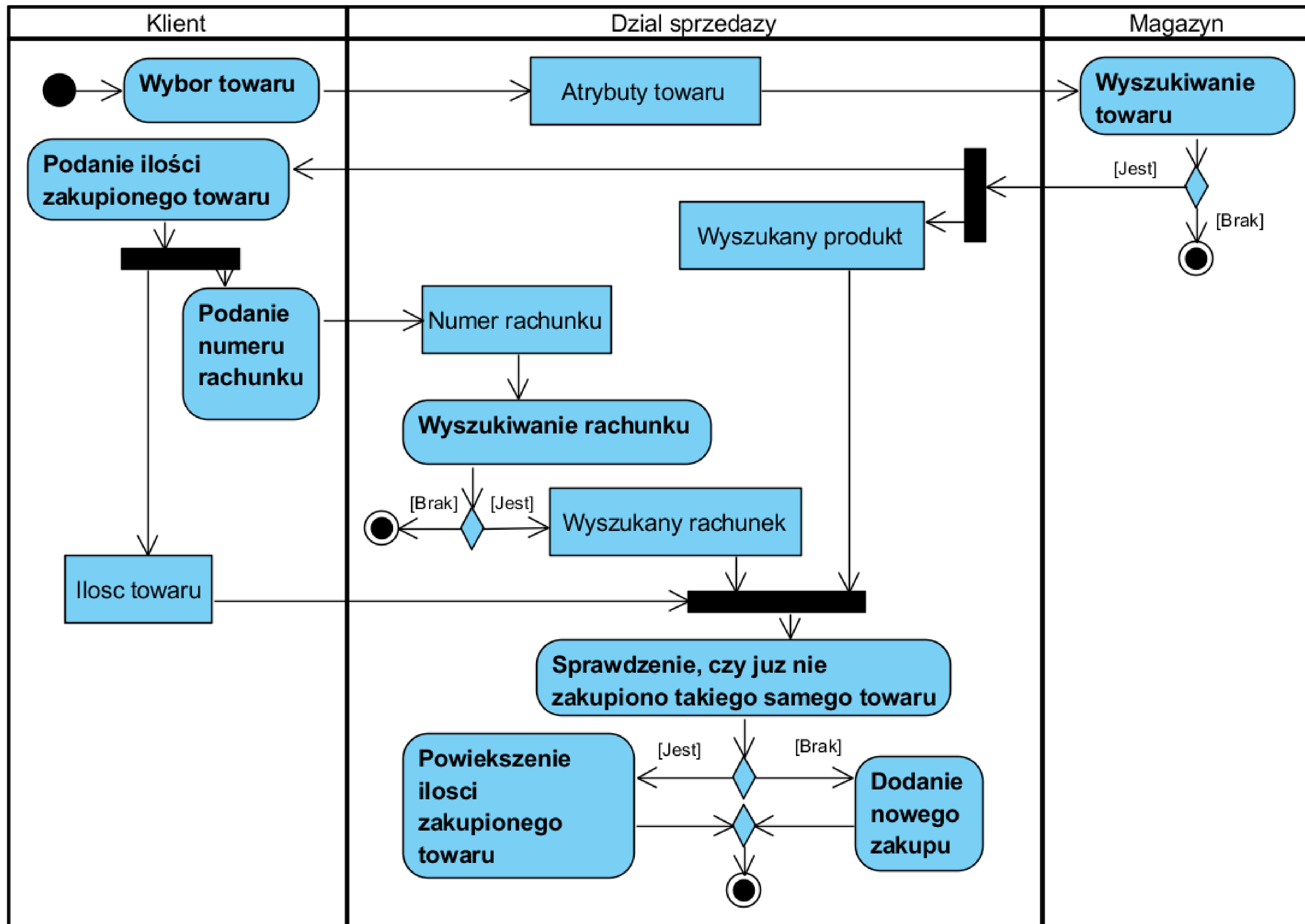
https://sparxsystems.com/resources/tutorials/uml2/activity-diagram.html

2. Przykład diagramów czynności UML – modelowanie przepływu czynności i obiektów

(wg Booch G., Rumbaugh J., Jacobson I., UML przewodnik użytkownika)

Diagramy czynności - System sporządzania rachunków
Kontynuacja (przykład 2 z wykładu 2)

(1) Diagram czynności jako **model biznesowy** „świata rzeczywistego” systemu sprzedaży towarów – proces zakupu



(1 cd) Obliczanie wartości rachunku

Sklep Spożywczo – Przemysłowy „ABC”
Jan Kowalski
ul. Leśna 1, xx-xxx Jakieś miasto
NIP xxx-xxx-xx-xx
Dn. 07r-09-24 nr wydr.8212

PARAGON FISKALNY

xxxxxxxxxxxxx
Nazwa produktu1 xxxxx *1*6.79* *A*
xxxxxxxxxxxxx
Nazwa produktu2 xxxx *4*0.59* *B*
Nazwa produktu3 xxx *0.6*4.59* *D*
xxxxxxxxxxxxx
Nazwa produktu4 xxxxx *2*2.49* *A*

To jest cena jednostkowa brutto

To jest ilość zakupioneg o towaru

To są kategorie podatków

Sp.op.A	11.77	PTU A = 22.00%	2.12
Sp.op.B	2.36	PTU B = 7.00%	0.15
Sp.op.D	2.75	PTU D = 3.00%	0.08
		Razem PTU	2.35

To jest cena brutto towarów z danej kategorii podatku

To są kwoty tara wynikające z istniejących kategorii podatków

RAZEM ZŁ 16.88

(2) Wykład 2 - Przykład 2. System sporządzania rachunków

Lista wymagań funkcjonalnych programu

1. System zawiera katalog produktów
2. Można zakupić cztery typy produktów różniące się sposobem obliczania ceny detalicznej: : bez promocji i bez podatku, z promocją i bez podatku, z podatkiem bez promocji, z podatkiem i z promocją,
3. Można wprowadzić wiele rachunków
4. Pozycje rachunku muszą zawierać produkty różne w sensie nazwy, ceny, podatku i promocji
5. Każda pozycja rachunku powinna podać swoją wartość brutto oraz dane produktu oraz ilość zakupionego produktu.
6. Na rachunku powinna znajdować się wartość łączna wszystkich zakupów oraz wartości zakupów należących do wybranych kategorii

Lista wymagań niefunkcjonalnych programu

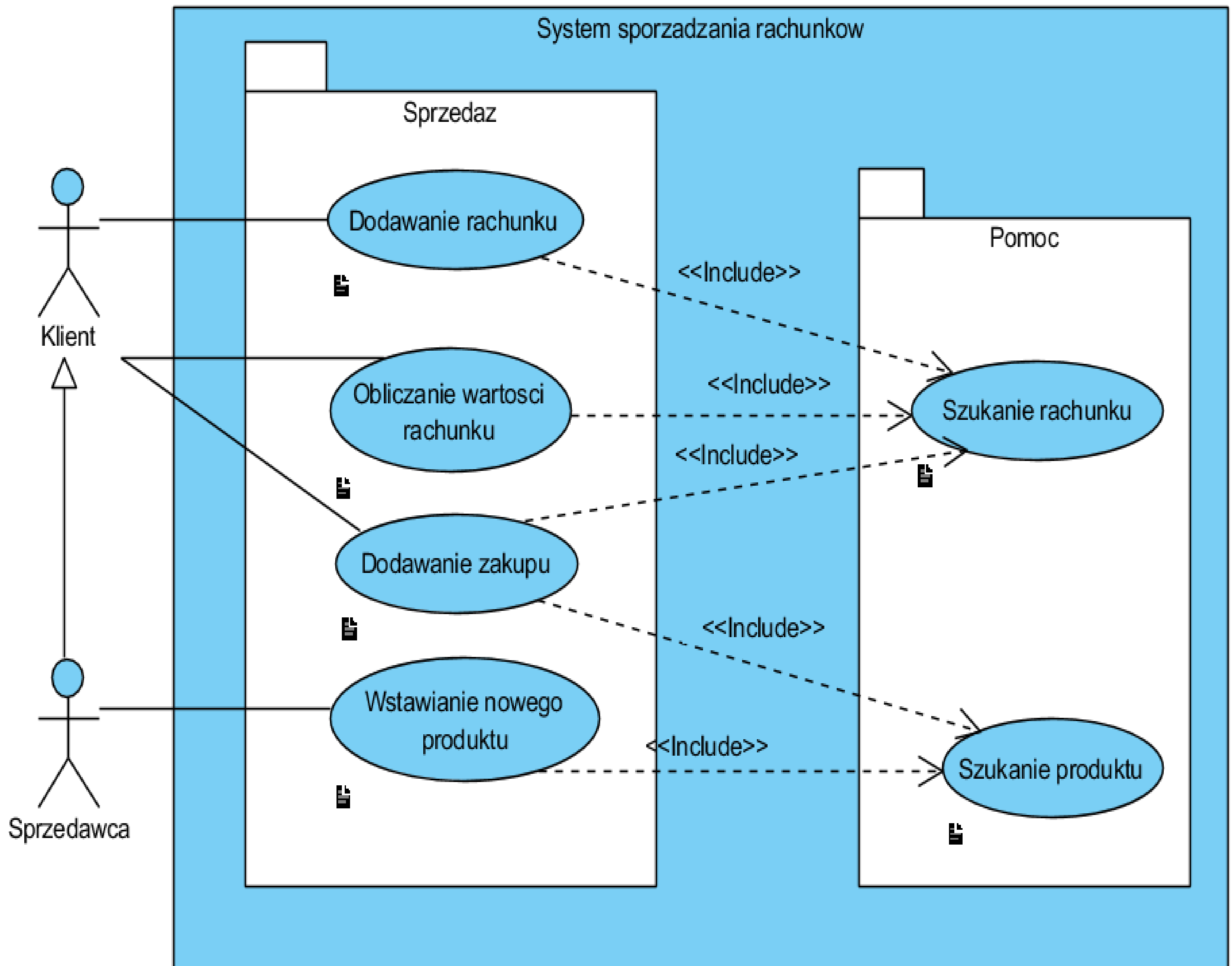
1. Wstawianie produktów może odbywać się tylko przez uprawnione osoby
2. Wstawianie nowych rachunków oraz wstawianie nowych zakupów jest dokonywane przez klientów
3. Zakupy mogą być dokonane przez Internet przez aplikację uruchamianą przez przeglądarkę lub bez jej pośrednictwa

3. Identyfikacja aktorów

AKTOR	OPIS	PRZYPADKI UŻYCIA
Klient	*Klient może dokonywać zakupów wybranych produktów przez Internet korzystając z przeglądarki lub z aplikacji*	• Wstawianie nowego rachunku powiązane przez <<include>> z PU Szukanie rachunku • Obliczanie wartosci rachunku powiązane przez <<include>> z PU Szukanie rachunku • Wstawianie nowego zakupu powiązane przez <<include>> z PU Szukanie rachunku oraz powiązane przez <<include>> z PU Szukanie produktu
Sprzedawca	*Sprzedawca może dodatkowo dodawać nowe produkty*	• Wstawianie nowego rachunku powiązane przez <<include>> z PU Szukanie rachunku • Obliczanie wartosci rachunku powiązane przez <<include>> z PU Szukanie rachunku • Wstawianie nowego zakupu powiązane przez <<include>> z PU Szukanie rachunku oraz powiązane przez <<include>> z PU Szukanie produktu • Wstawianie nowego produktu powiązane przez <<include>> z PU Szukanie produktu

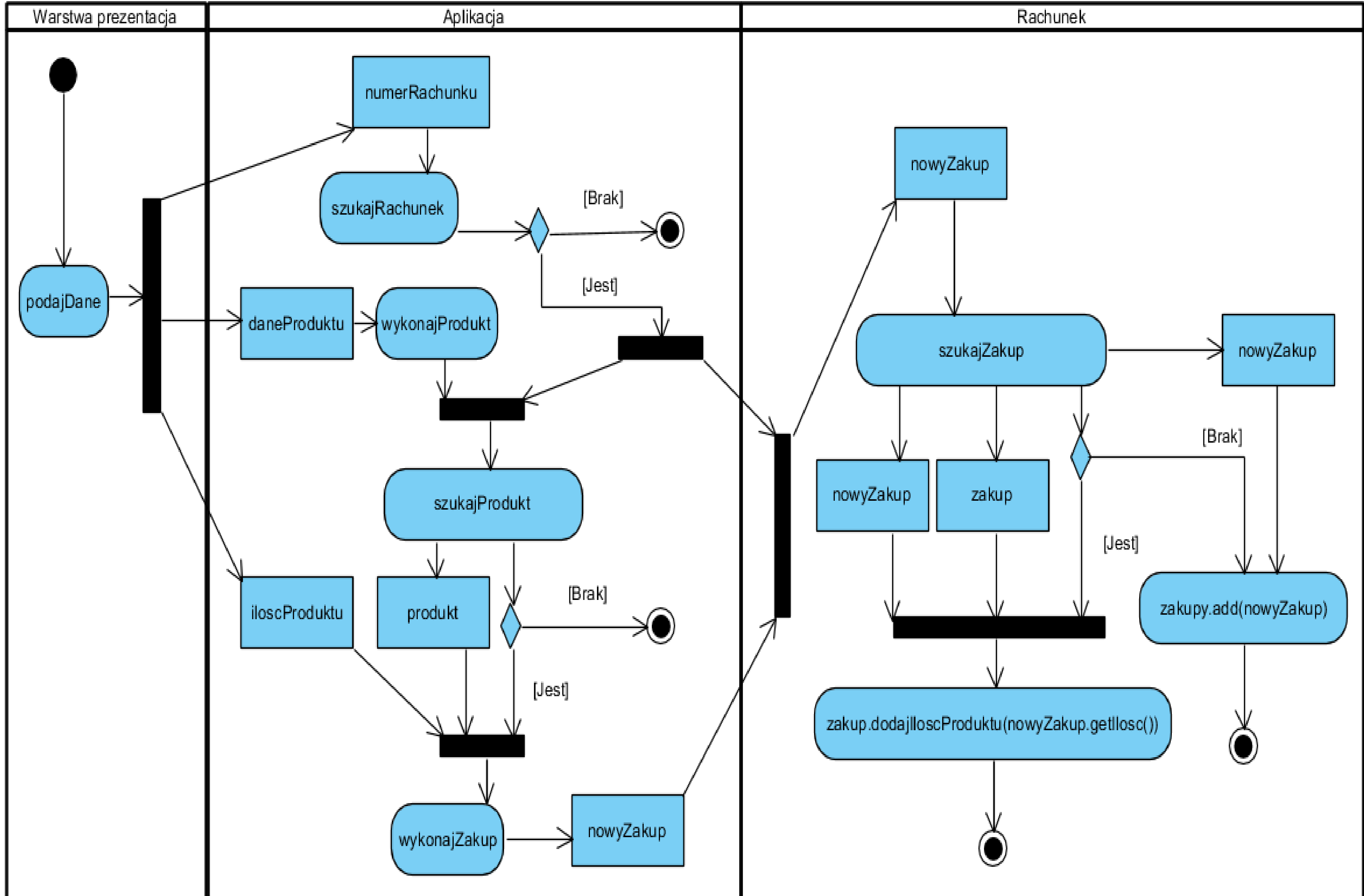
4. Specyfikacja wymagań

Funkcje_sporzadzania_rachunkow

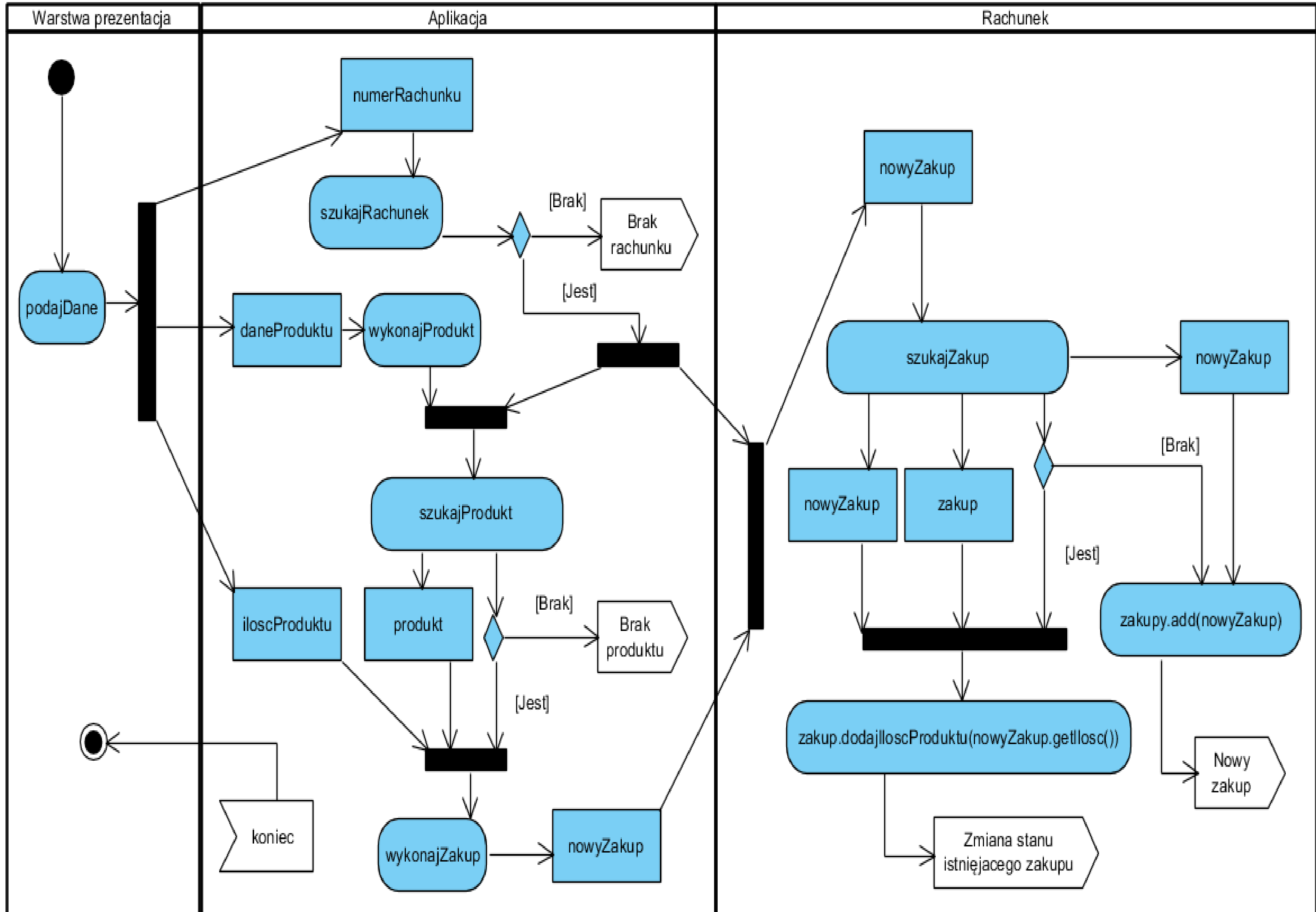


Cursor
Use Case
Association
Actor
System
Include
Extend
Dependency
Generalization
Collaboration
Note
Anchor
Constraint
Containment
Package
Diagram Overview
ETL Table

(5a) Wersja 1 - diagram czynności przypadku użycia ***Wstawianie nowego zakupu***
(**model przypadku użycia w warstwie biznesowej – partycje *Aplikacja* i *Rachunek***)



(5b) Wersja 2 -Diagram czynności przypadku użycia *Wstawianie nowego zakupu* (**model przypadku użycia w warstwie biznesowej – partycje *Aplikacja* i *Rachunek***)



(6) Kod źródłowy metody *wstawZakup* obiektu typu *Aplikacja* – pierwsza część realizacji przypadku użycia ***Wstawianie nowego zakupu***

```
//partycja Aplikacja

public   void wstawZakup (int nr, int ile, String dane[])
 {
   Rachunek rachunek;
   Fabryka fabryka = new Fabryka();
   ProduktBezPodatku produktWzor = fabryka.wykonajProdukt(dane), produkt;
    if ((rachunek=szukajRachunek(nr)) != null)
      if ((produkt=szukajProdukt(produktWzor)) != null)
        rachunek.wstawZakup(new Zakup(ile, produkt));
   }
```

(7) Kod źródłowy metody *wstawZakup* obiektu typu *Rachunek* – druga część realizacji przypadku użycia ***Wstawianie nowego zakupu***

```
//partycja Rachunek

 private ArrayList<Zakup> zakupy = new ArrayList<>();

public void wstawZakup (Zakup nowyZakup)
    {
      Zakup zakup;
      if ((zakup = szukajZakup(nowyZakup)) != null)
           zakup.dodajIloscProduktu(nowyZakup.getIlosc());
      else
           zakupy.add(nowyZakup);
  }
```

Wytyczne dla tworzenia diagramów czynności

1. Należy ustalić najważniejsze czynności - nie można przedstawić na jednym diagramie wszystkich czynności
2. Należy wybrać obiekty przedsiębiorstwa, które są zobowiązane do realizacji bardziej ogólnego przepływu. Mogą to być elementy rzeczywiste, istniejące w słownictwie systemu, lub elementy abstrakcyjne. W obu przypadkach należy utworzyć tor dla każdego wybranego obiektu.
3. Należy zidentyfikować stan początkowy i końcowy modelowanego przepływu.
4. Przechodząc od stanu początkowego do końcowego należy modelować kolejne stany czynności lub stany akcji.
5. W przypadku złożonych akcji lub często występujących zbiorów akcji należy je połączyć w stany czynności. Z każdym stanem skojarz oddzielny diagram czynności, który przedstawia zebrane nim akcje.
6. Należy zobrazować przepływy czynności między stanami akcji i stanami czynności. Pierwsze powinny być brane pod uwagę przepływy sekwencyjne, potem rozgałęzienia, na końcu rozwidlenia i scalenia.
7. Jeśli w modelowanym przepływie czynności biorą udział istotne obiekty, należy je umieścić na diagramie. Należy uwzględniać zmieniające się atrybuty i stany tych obiektów, jeśli jest to konieczne do zrozumienia przepływu tych obiektów.

Diagramy czynności

1. **Diagramy czynności UML**

   https://sparxsystems.com/resources/tutorials/uml2/activity-diagram.html

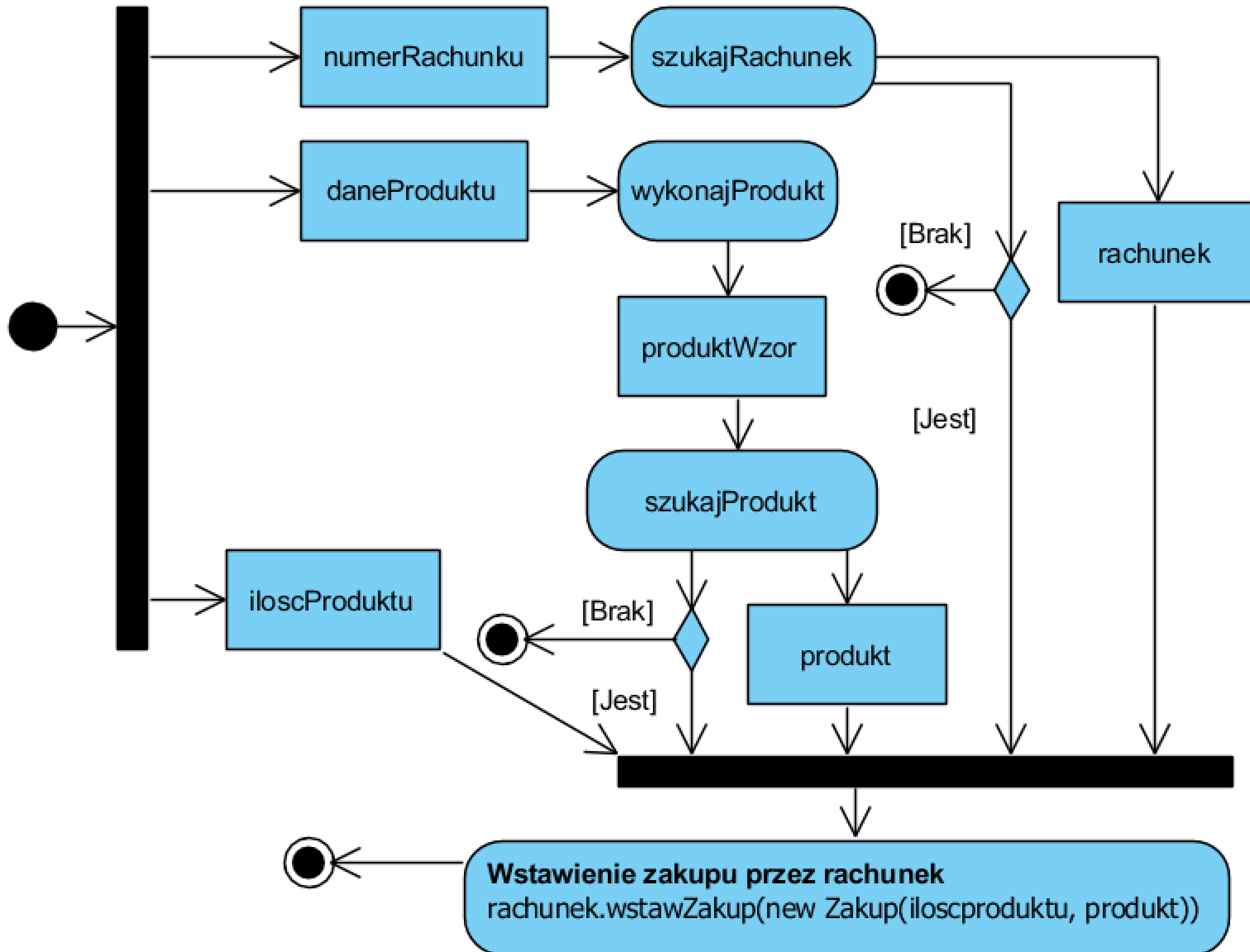
2. **Przykład diagramów czynności UML – modelowanie przepływu czynności i obiektów**

   (wg Booch G., Rumbaugh J., Jacobson I., UML przewodnik użytkownika)

3. **Przykład diagramów czynności UML – modelowanie operacji**

   (wg Booch G., Rumbaugh J., Jacobson I., UML przewodnik użytkownika)

Modelowanie operacji *wstawZakup*



Kod źródłowy projektowanej operacji za pomocą diagramu aktywności

```
//Aplikacja

public   void wstawZakup (int nr, int ile, String dane[])
 {
   Rachunek rachunek;
   Fabryka fabryka = new Fabryka();
   ProduktBezPodatku produktWzor = fabryka.wykonajProdukt(dane), produkt;
   if ((rachunek=szukajRachunek(nr)) != null)
     if ((produkt=szukajProdukt(produktWzor)) != null)
       rachunek.wstawZakup(new Zakup(ile, produkt));
  }
```

Wytyczne dla tworzenia diagramów czynności do modelowania operacji

1. Należy uwzględnić abstrakcje uczestniczące w tej operacji. Są to parametry przekazywane do metody, atrybuty klasy metody oraz klas obiektów związanych z metodą.
2. Należy zidentyfikować warunki wstępne stanu początkowego i końcowego modelowanej operacji. Należy znaleźć wszystkie wartości, które nie mogą zmienić się podczas realizacji operacji.
3. Przechodząc od stanu początkowego do końcowego należy modelować kolejne stany czynności lub stany akcji.
4. Należy w razie potrzeby korzystać z rozgałęzień do modelowania ścieżek warunkowych i iteracji.
5. Jeśli operacja należy do klasy aktywnej (i tylko wtedy), należy wtedy korzystać z rozwidleń i scaleń, żeby określić równoległe przepływy sterowania.

Syntaktyka diagramów klas

1. **Diagramy klas UML**
2. **Identyfikacja elementów diagramów klas**

(wg Booch G., Rumbaugh J., Jacobson I., UML przewodnik użytkownika)

[Shalloway A.,Trott James R.,Projektowanie zorientowane obiektowo. Wzorce projektowe. Gliwice, Helion, 2005]

Syntaktyka diagramów klas

1. **Diagramy klas UML**

Diagramy UML 2 – część trzecia

Na podstawie

UML 2.0 Tutorial

https://sparxsystems.com/resources/tutorials/uml2/class-diagram.html

Dwa rodzaje diagramów UML 2

Diagramy UML modelowania strukturalnego

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

Diagramy UML modelowania zachowania

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

Diagramy klas (Class Diagrams)

- **Diagram klas** reprezentuje statyczny model świata rzeczywistego: jego atrybuty i właściwości, odpowiedzialności oraz powiązania
- **Klasa** reprezentuje model rzeczy konceptualnej i fizycznej i jest powielana w postaci **obiektów**, czyli wystąpień klasy.
- **Atrybuty**: składowe klasy do przechowywania danych, które posiadają nazwę, typ, zakres wartości oraz określony dostęp.
- **Operacje:** składowe klasy 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.

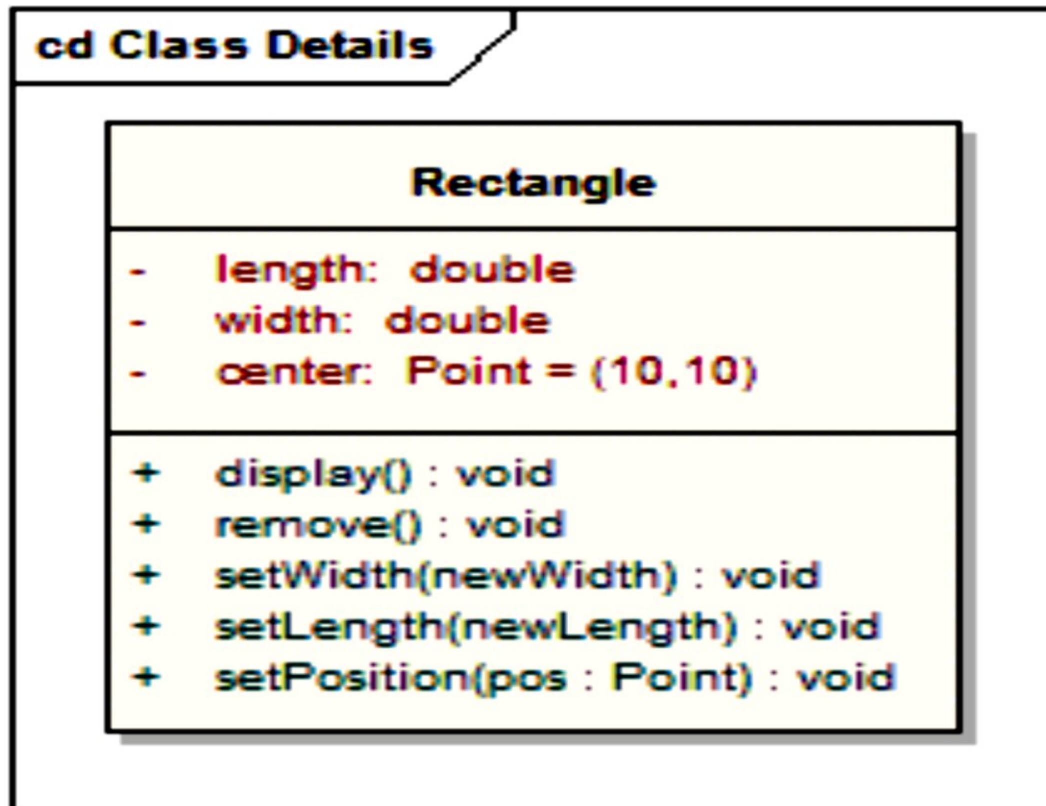
Operacje: setWidth, setLength, setPosition

+ składowa publiczna

- składowa prywatna

składowa typu protected

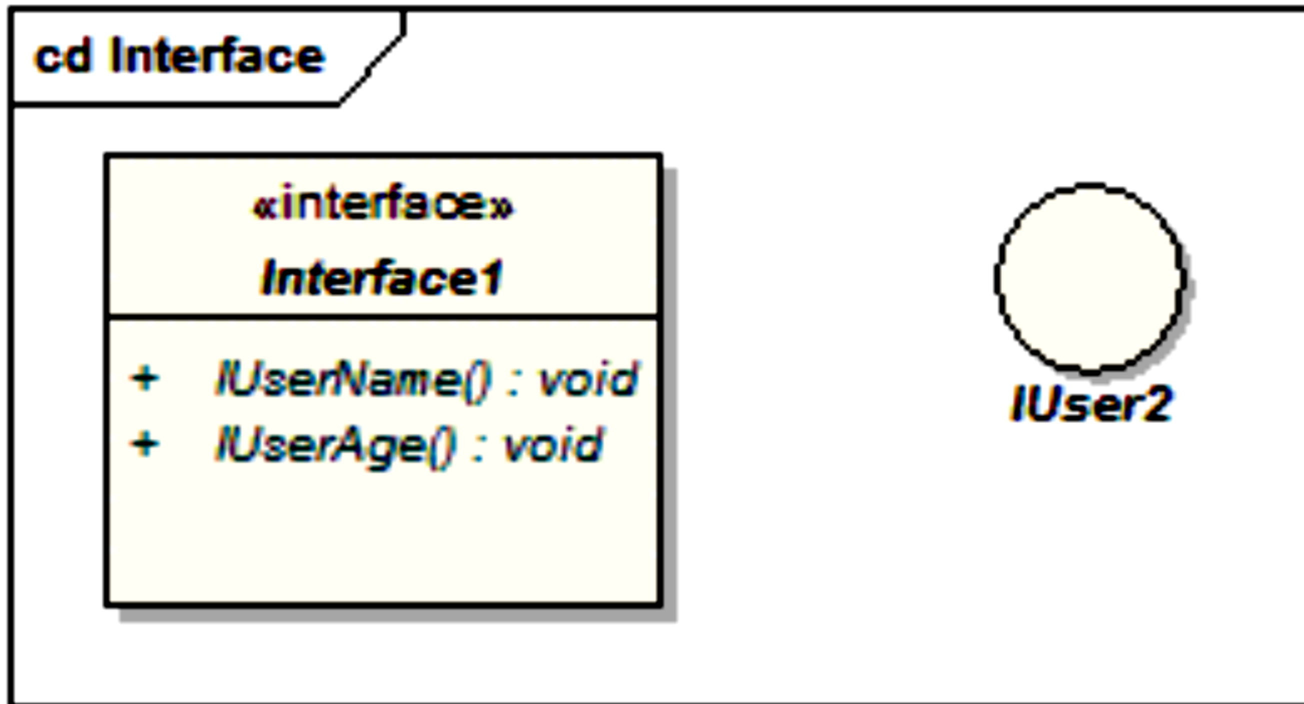
~ składowa publiczna w zasięgu pakietu

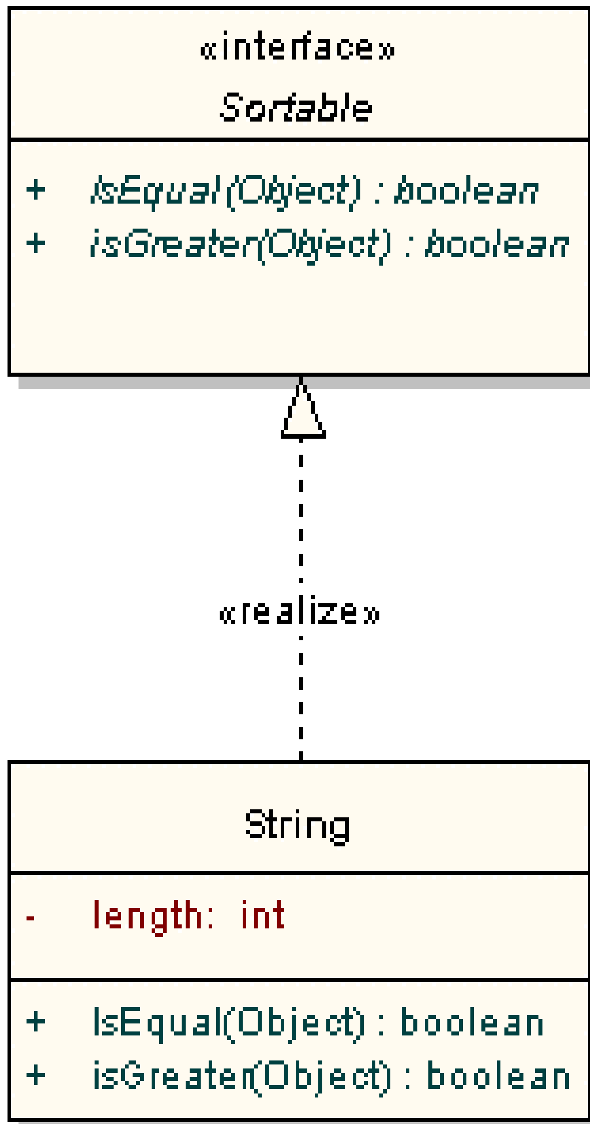


Interfejs<<interface>>

Jest przedstawiany jako:

- klasa zawierająca specyfikację właściwości (operacji czyli metod), które musi zdefiniować implementująca go klasa
- **reprezentowany jest jako koło** bez wyspecyfikowanych metod i połączenia z interfejsem przez klasę implementującą nie są oznaczane strzałką

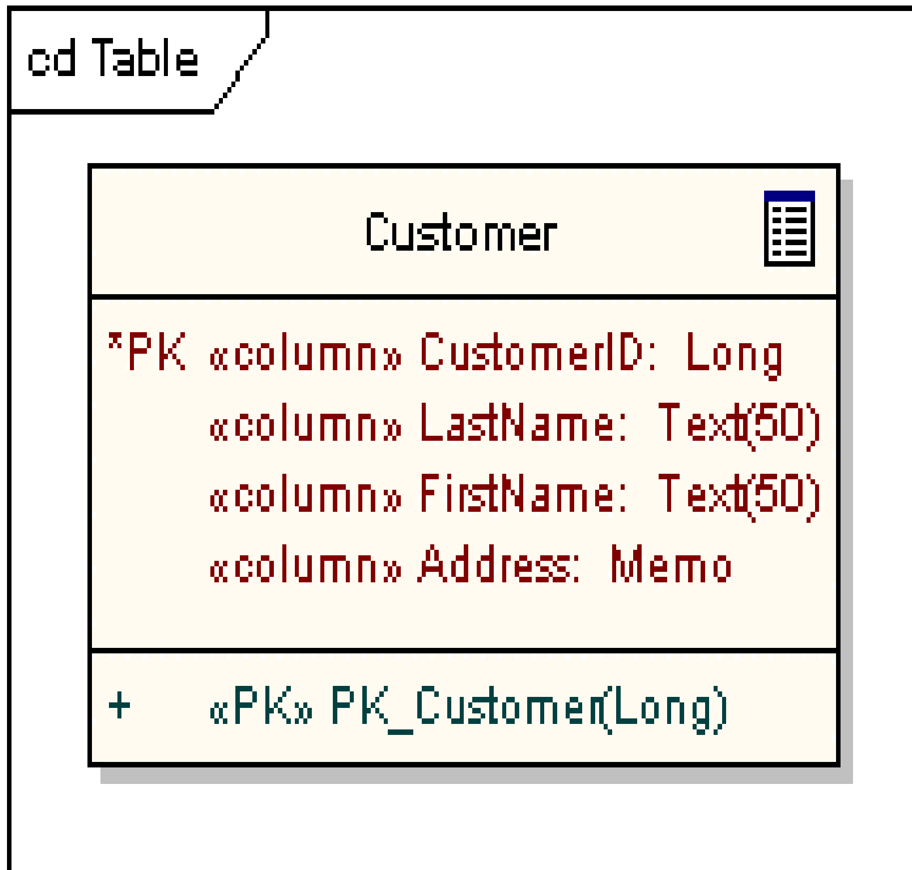




Realizacja (Realization)

- oznaczane są przerywaną strzałką ze stereotypem **<<realize>>**
- strzałka wychodzi z klasy implementującej do klasy implementowanej
- implementacja właściwości klasy typu ***interface***
- klasa implementująca jest rysowana podobnie jak klasa implemetowana

Tabele (table)



- **klasa stereotypowa**

- **atrybuty tabeli o stereotypie <<column>>**

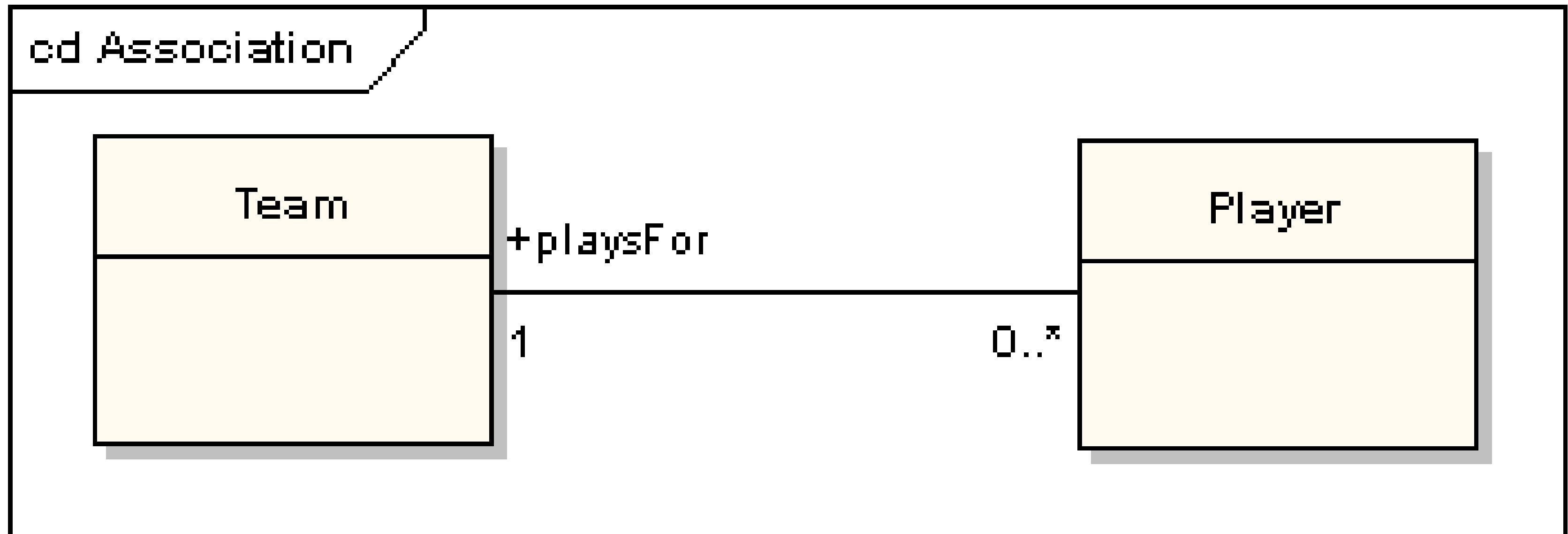
- **posiada klucz główny (<<PK>> – primary key)** obejmujący jedną lub wiele kolumn o unikatowym znaczeniu

- **może posiadać jeden lub wiele kluczy obcych (<<FK>>- foreign key)** jako kluczy głównych w powiązanych tabelach po stronie „1" powiązanych tabel.

Powiązanie (Association)

Wiąże dwa elementy modelu w związek strukturalny:

- połączenie może zawierać nazwy ról na każdym końcu, liczność wystąpień instancji tych elementów, kierunek oraz ograniczenia

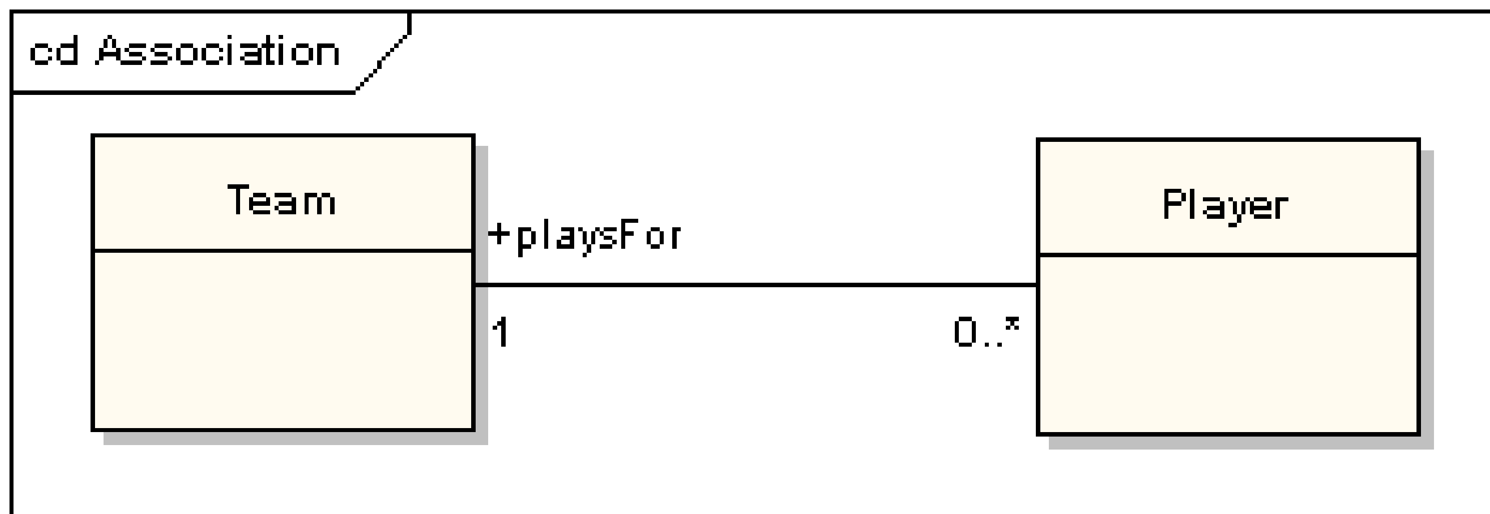


- **jest implementowana następująco:**
  1. **relacje wiele do jeden** lub **jeden do jeden: w obiekcie po stronie wiele lub jeden znajduje się referencja do obiektu z przeciwnej strony relacji (strony jeden)**
  2. **relacje jeden do wiele: kolekcja referencji instancji obiektów po stronie wiele w obiekcie po stronie jeden**

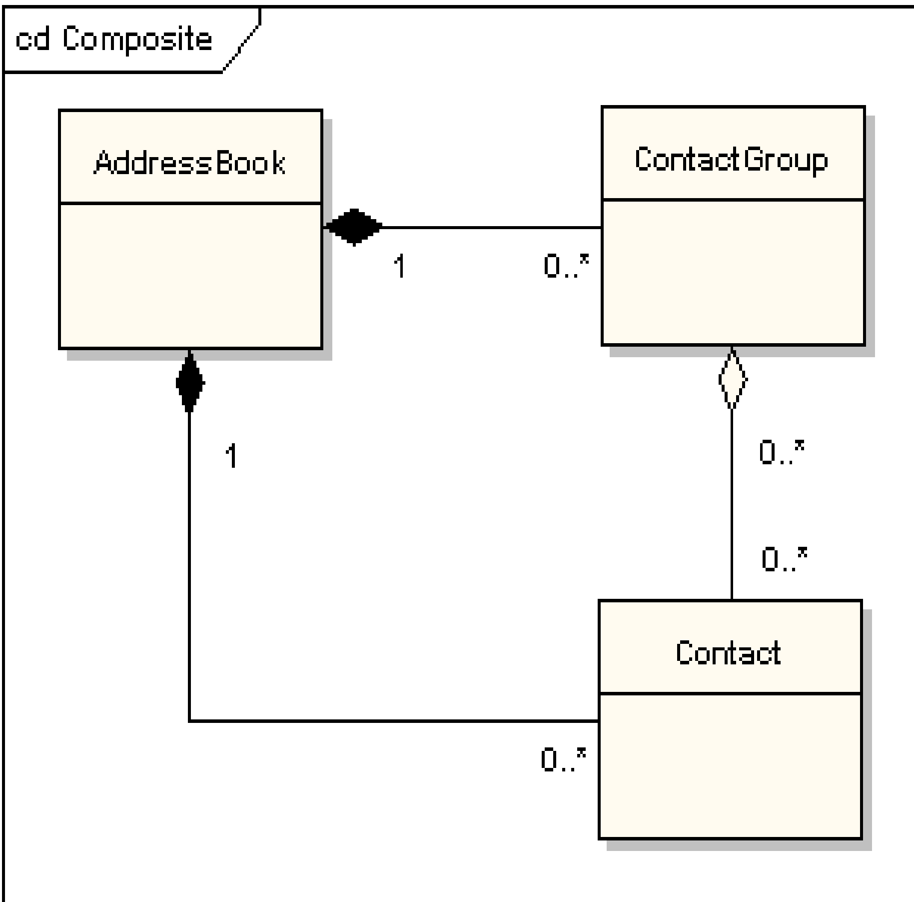
  (np. referencja do obiektu typu *Team* występuje w obiekcie typu *Player jako atrybut* oraz kolekcja referencji obiektów typu *Player* w obiekcie klasy *Team jako atrybut*)

Role

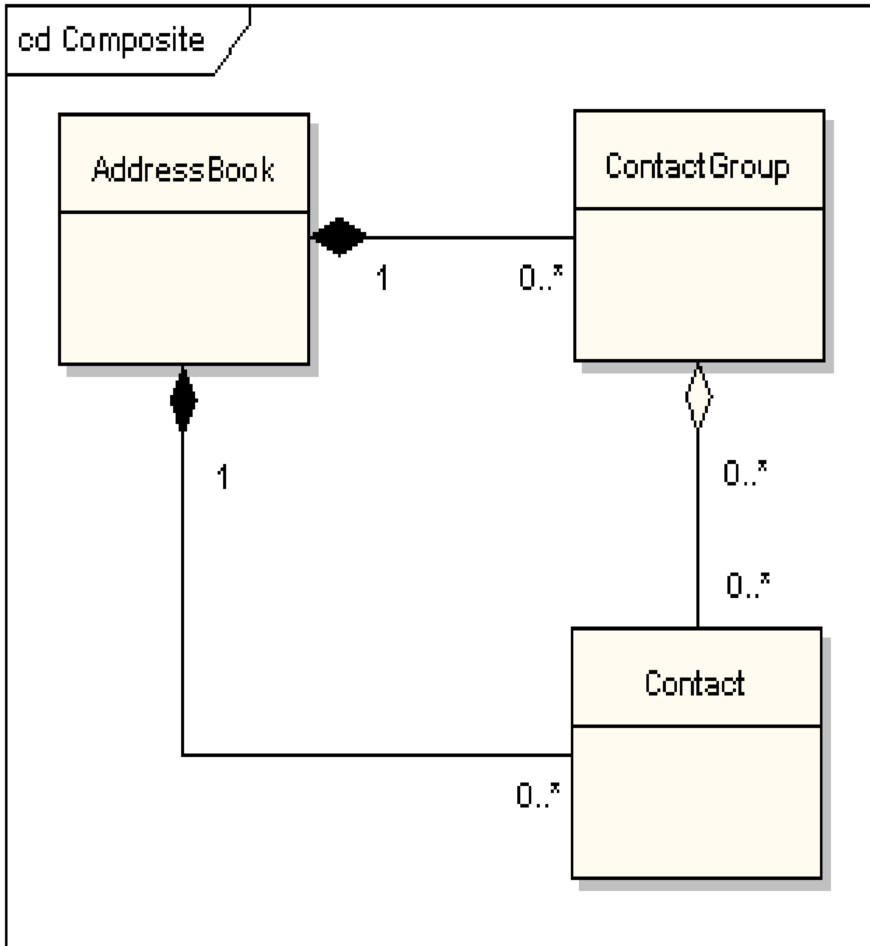
Role to oblicze, jakie prezentuje klasa przy jednym końcu drugiej klasie na drugim końcu asocjacji (instancja typu *Player* gra (*+playsFor*) dla instancji typu *Team*)



Agregacja (Aggregation)



- 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 (agregacja kompozytowa) 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

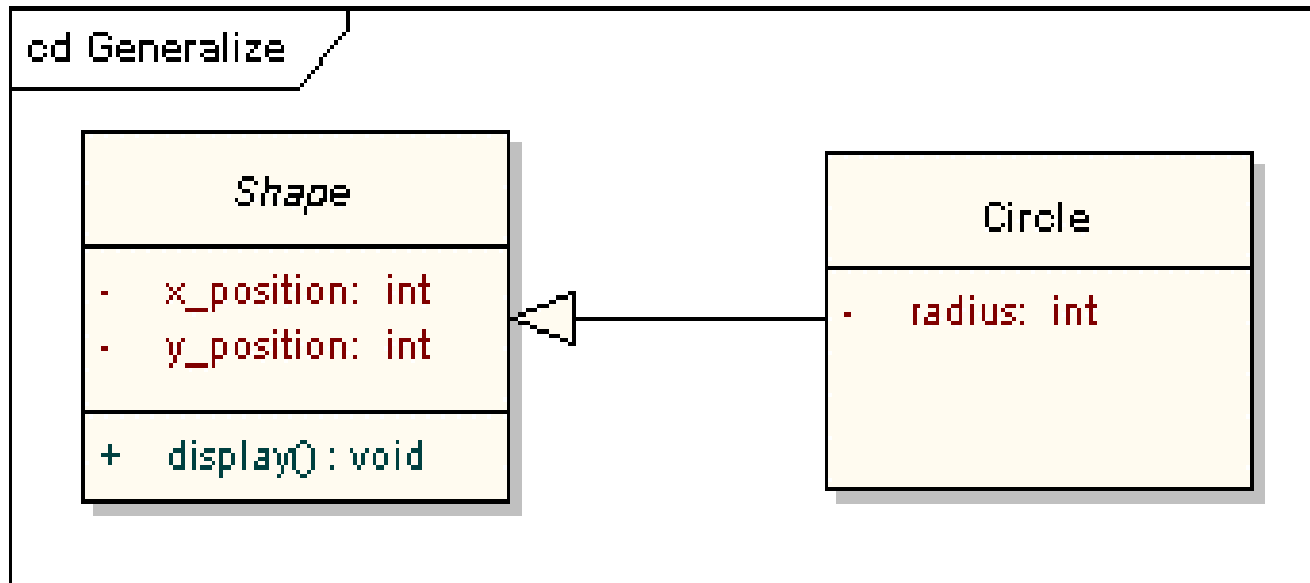


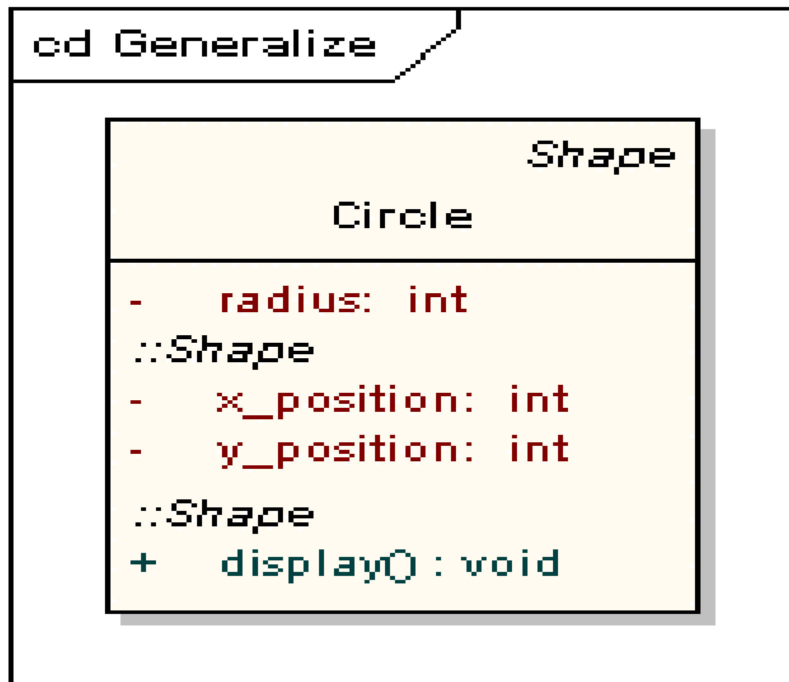
- **jest implementowana podobnie jak związek typu Association**

1. obiekt typu *Contact* zawiera atrybut typu kolekcja obiektów typu *ContactGroup* (*strona wiele do wiele*) *oraz atrybut typu* rerefencja obiektu typu *AddressBook* (*strona wiele do jeden*)
2. Obiekt typu *ContactGroup* zawiera atrybut typu kolekcja referencji do obiektów typu *Contact* (*strona wiele do wiele*) *oraz atrybut typu* rerefencja obiektu typu *AddressBook* (*strona wiele do jeden*)
3. Obiekt typu *AddressBook zawiera* dwa atrybuty: typu kolekcja referencji obiektów typu *Contact* oraz kolekcja referencji obiektów typu *ContactGroup* -(*strona jeden do wiele*)

Usuwanie obiektów: 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 *ContactGroup* nie pociąga za sobą usuwania obiektów klasy *Contact.*

Generalizacja czyli dziedziczenie (Generalization)

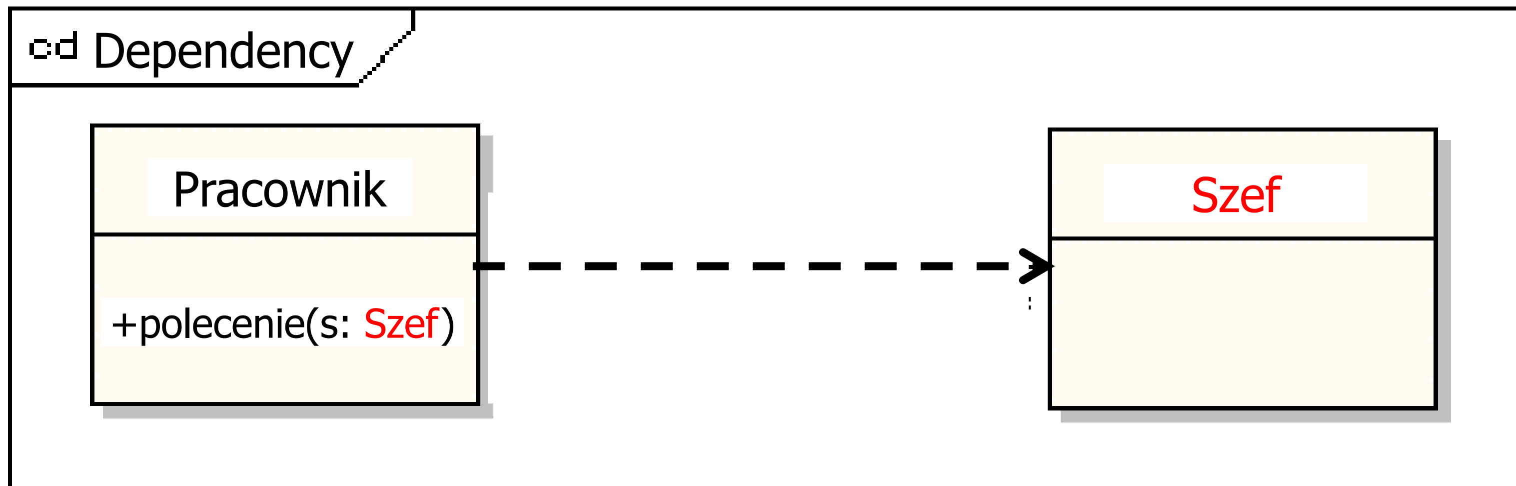




Używana do oznaczania **dziedziczenia**

- **strzałka wychodzi z klasy** dziedziczącej do klasy, po której dziedziczy
- np. klasa Circle dziedziczy atrybuty ***x_position, y_position*** **i metodę display() po klasie Shape** oraz dodaje atrybut **radius**

Zależność (Dependency)



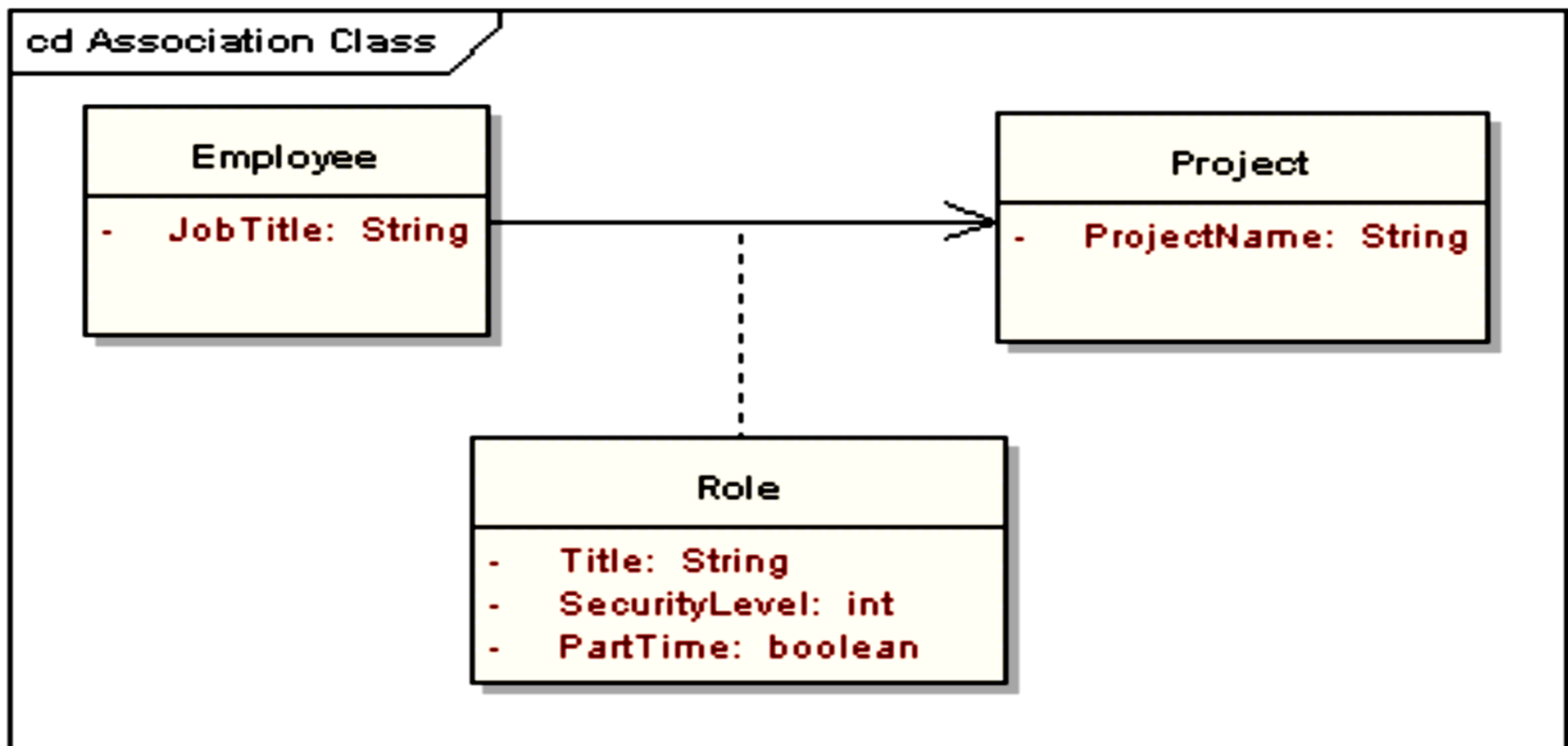
- zależności są używane do modelowania powiązań między elementami modelu we wczesnej fazie projektowania, jeśli nie można określić precyzyjnie typu powiązania. Stanowią one wtedy związek użycia **(<<usage>>)**.
- strałka przerywana wskazuje grotem na klasę, od której coś zależy.
- Później są one uzupełniane 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 (Trace)

- łączy elementy modelu o tym samym przeznaczeniu, wymaganiach lub tym samym momencie zmian
- ma znaczenie informacyjne

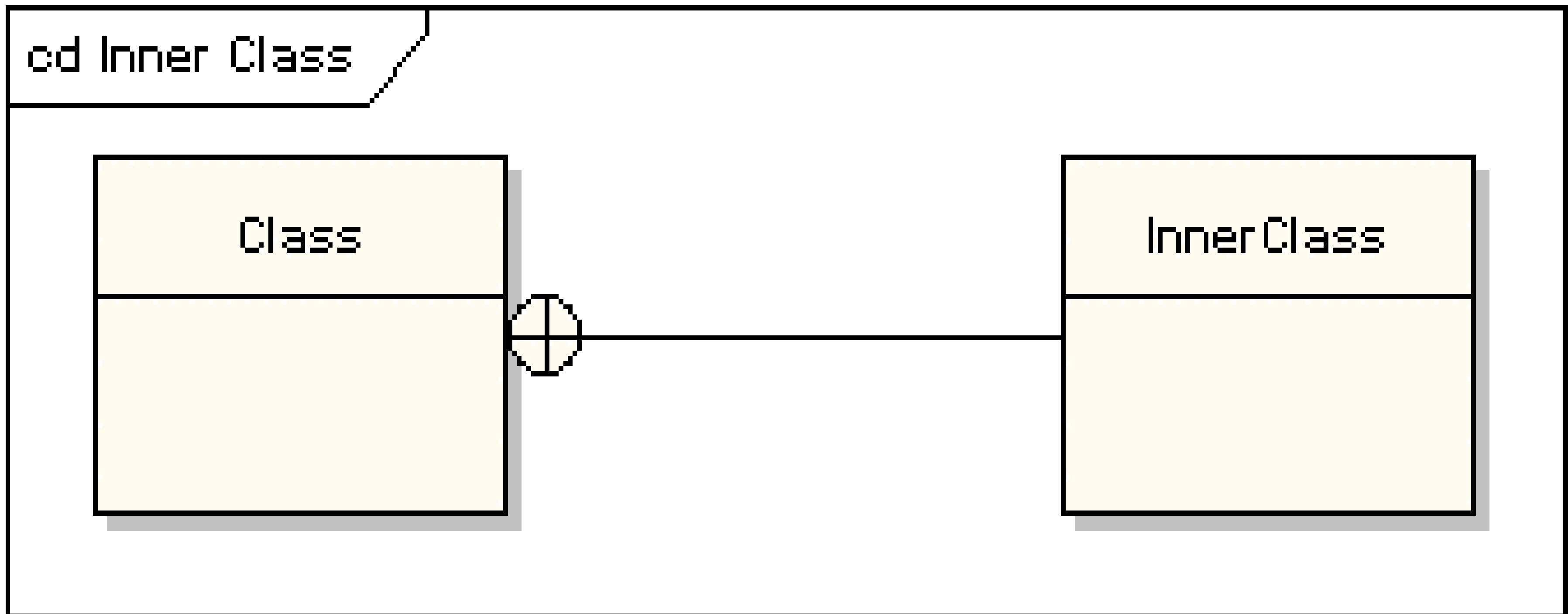
Klasa powiązań (Association Class)

- uzupełnia powiązane obiekty 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

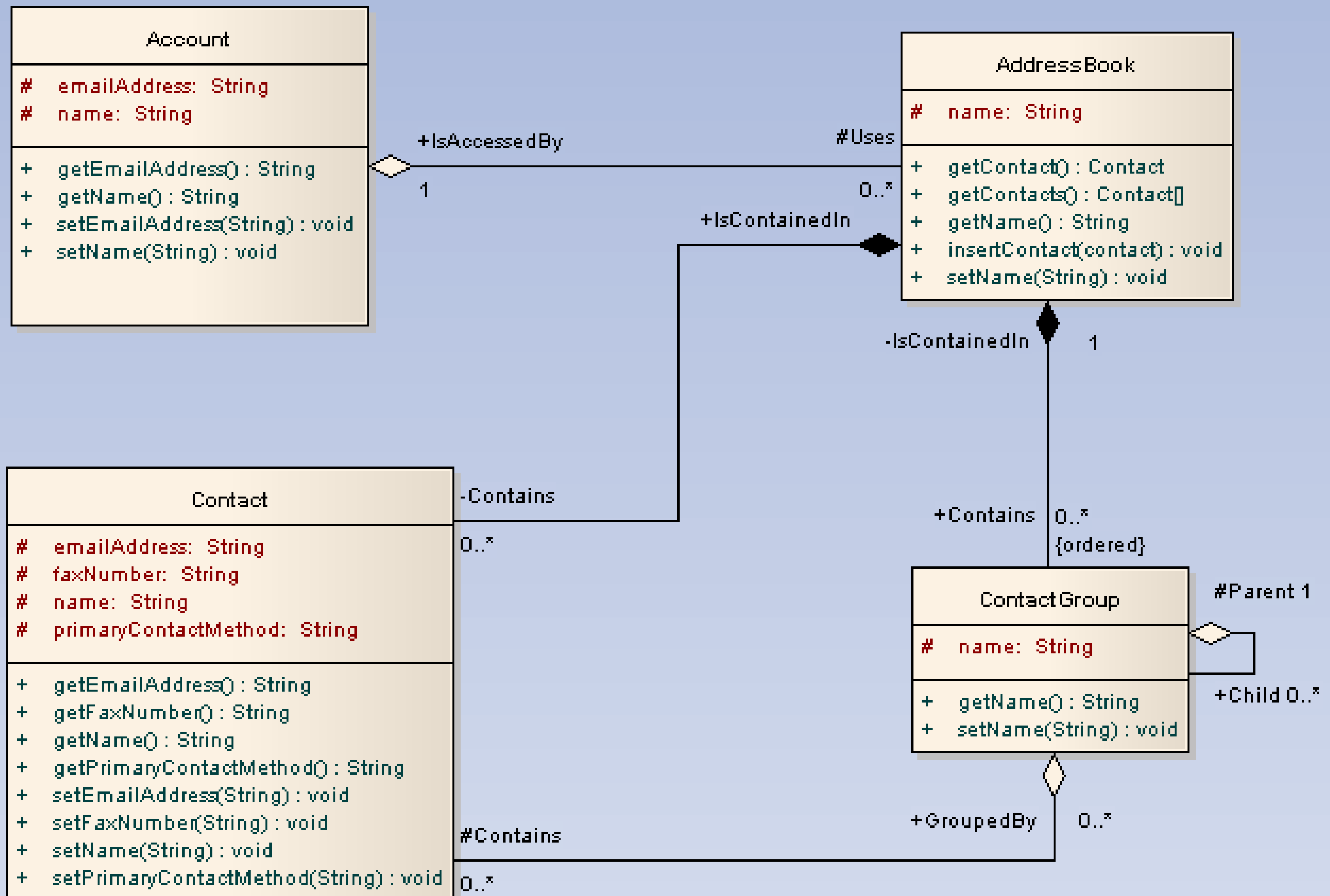


Zagnieżdżenie (Nesting)

- 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***



Podsumowanie – diagram klas



Syntaktyka diagramów klas

1. Diagramy klas UML

https://sparxsystems.com/resources/tutorials/uml2/class-diagram.html

2. Identyfikacja elementów diagramów klas

[Booch G., Rumbaugh J., Jacobson I., UML przewodnik użytkownika]

[Shalloway A.,Trott James R.,Projektowanie zorientowane obiektowo. Wzorce projektowe. Gliwice, Helion, 2005]

Identyfikacja klas

(wg Booch G., Rumbaugh J., Jacobson I., UML przewodnik użytkownika)

- Zidentyfikuj zbiór klas, które współpracują ze sobą w celu wykonania poszczególnych czynności
- Określ zbiór zobowiązań każdej klasy
- Rozważ zbiór klas jako całość: podziel na mniejsze te klasy, które mają zbyt wiele zobowiązań; scal w większe te klasy, które mają zbyt mało zobowiązań
- Rozpatrz sposoby wzajemnej kooperacji tych klas i porozdzielaj ich zobowiązania tak, aby żadna z nich była ani zbyt złożona ani zbyt prosta
- Elementy nieprogramowe (urządzenia) przedstaw w postaci klasy i odróżnij go za pomocą własnego stereotypu; jeśli ma on oprogramowanie, może być traktowany jako węzeł diagramu klas w celu rozwijania tego oprogramowania
- Zastosuj typy pierwotne (tabele, wyliczenia, typy proste np. boolean itp.)

Identyfikacja związków: zależność (Dependency)

(wg Booch G., Rumbaugh J., Jacobson I., UML przewodnik użytkownika)

Modelowanie zależności

- Utworzyć zależności między klasą z operacją, a klasą użytą jako parametr tej operacji
- Stosuj zależności tylko wtedy, gdy modelowany związek nie jest strukturalny

Identyfikacja związków: generalizacja czyli dziedziczenie (Generalization)

(wg Booch G., Rumbaugh J., Jacobson I., UML przewodnik użytkownika)

- Ustaliwszy zbiór klas poszukaj zobowiązań, atrybutów i operacji wspólnych dla co najmniej dwóch klas
- Przenieś te wspólne zobowiązania, atrybuty i operacje do klasy bardziej ogólnej; jeśli to konieczne, utwórz nową klasę, do której zostaną przypisane te właśnie byty (uważaj z wprowadzaniem zbyt wielu poziomów generalizacji)
- Zaznacz, że klasy szczegółowe dziedziczą po klasie ogólnej, to znaczy uwzględnij uogólnienia biegnące od każdego potomka do bardziej ogólnego przodka
- Stosuj uogólnienia tylko wtedy, gdy masz do czynienia ze związkiem „jest rodzajem”; dziedziczenie wielobazowe często można zastąpić agregacją
- Wystrzegaj się wprowadzania cyklicznych uogólnień – żadna klasa nie może być swoim przodkiem
- Utrzymuj uogólnienia w pewnej równowadze; krata dziedziczenia nie powinna być zbyt głęboka (pięć lub więcej poziomów już budzi wątpliwości) ani zbyt szeroka (lepiej wprowadzić pośrednie klasy abstrakcyjne)

Identyfikacja związków strukturalnych: powiązanie (Association) , agregacja (Aggregation)

(wg Booch G., Rumbaugh J., Jacobson I., UML przewodnik użytkownika)

- Rozważ, czy w wypadku każdej pary klas jest konieczne przechodzenie od obiektów jednej z nich do obiektów drugiej
- Rozważ, czy w wypadku każdej pary klas jest konieczna inna interakcja między obiektami jednej z nich a obiektami drugiej niż tylko przekazywanie ich jako parametrów; jeśli tak, uwzględnij powiązanie między tymi klasami, w przeciwnym wypadku jest to zależność użycia. Ta metoda identyfikacji powiązań jest oparta na zachowaniu
- Dla każdego powiązania określ liczebność (szczególnie wtedy, kiedy nie jest to 1 - wartość domyślna) i nazwy ról (ponieważ ułatwiają zrozumienie modelu)
- Jeśli jedna z powiązanych klas stanowi strukturalną lub organizacyjną całość w porównaniu z klasami z drugiego końca związku, które wyglądają jak części, zaznacz przy niej specjalnym symbolem, że chodzi o agregację.
- Stosuj powiązania głównie wtedy, kiedy między obiektami zachodzą związki strukturalne

Identyfikacja wzorców projektowych (wstęp do wykładu 5)

- Dobrze zbudowany system obiektowy jest pełen wzorców obiektowych
- Wzorzec to zwyczajowo przyjęte rozwiązanie typowego problemu w danym kontekście
- Strukturę wzorca przedstawia się w postaci diagramu klas
- Zachowanie się wzorca przedstawia się za pomocą diagramu sekwencji
- Wzorce projektowe: Wzorzec reprezentuje powiązanie problemu z rozwiązaniem **(wg Booch G., Rumbaugh J., Jacobson I., UML przewodnik użytkownika)**

- Każdy wzorzec składa się z trzech części, które wyrażają związek między konkretnym kontekstem, problemem i rozwiązaniem **(Christopher Aleksander)**
- Każdy wzorzec to trzyczęściowa reguła, która wyraża związek między konkretnym kontekstem, rozkładem sił powtarzającym się w tym kontekście i konfiguracją oprogramowania pozwalająca na wzajemne zrównoważenie się tych sił w celu rozwiązania zadania. **(Richar Gabriel)**
- Wzorzec to pomysł, który okazał się użyteczny w jednym rzeczywistym kontekście i prawdopodobnie będzie użyteczny w innym. **(Martin Fowler)**

Identyfikacja klas
Analiza wspólności (perspektywa koncepcji, model analizy – wykład 1)

Przykład 2 z wykładu 2 i jego kontynuacja

Wykryto **trzy główne klasy** typu **„Entity”** ze względu na odpowiedzialność:

- **Rachunek** (PU: **Wstawianie nowego rachunku, Wstawianie nowego zakupu, Obliczanie wartosci rachunku**),
- **Zakup** (PU: **Wstawianie nowego zakupu, Obliczanie wartosci rachunku**),
- **ProduktBezPodatku** (PU: **Wstawianie nowego produktu, Wstawianie nowego zakupu, Obliczanie wartosci rachunku**)

Analiza zmienności (perspektywa specyfikacji, model projektowy – wykład 1)

Przykład 2 z wykładu 2 i jego kontynuacja

Wykryto **dziedziczenie** w właściwościach produktów, które podają cenę jednostkową podawaną jako cenę netto, jeśli produkt nie posiada atrybutu podatek lub cenę brutto, jeśli posiada atrybut podatek. Zdefiniowano klasę pochodną:

- **ProduktZPodatkiem** typu **„Entity”**, która dziedziczy od klasy **ProduktBezPodatku** (PU: **Wstawianie nowego produktu, Obliczanie wartosci rachunku**)

Analiza zmienności (c.d)

Wykryto **strategię zmniejszania ceny jednostkowej wynikającej z promocji** powiązaną z produktem zarówno z podatkiem, jak i bez podatku:

- Zdefiniowano klasę **Promocja** typu **„Entity"**
- Zdefiniowano **związek typu asocjacja (lub słaba agregacja)** między klasami **ProduktBezPodatku i Promocja**, który jest dziedziczony przez pozostałe typy produktu tzn. **ProduktZPodatkiem**. Ponieważ jednak promocja nie musi dotyczyć każdego produktu, jest **w związku asocjacji (lub agregacji słabej)** **0..1 do 0..*** z bazowym (głównym) produktem typu **ProduktBezPodatku**
- Dzięki temu produkty typu **ProduktBezPodatku** i **ProduktZPodatkiem** powinny podawać uogólnioną cenę detaliczną: **bez podatku, z podatkiem oraz w razie potrzeby z uwzględnieniem scenariusza dodawania promocji do ceny detalicznej produktu dla dwóch pierwszych przypadków** (stąd cztery typy ceny detalicznej)
- Podstawą identyfikacji są PU: **Wstawianie nowego produktu, Wstawianie nowego zakupu, Obliczanie wartosci rachunku.**

Analiza zmienności (c.d)

Wykryto związki:

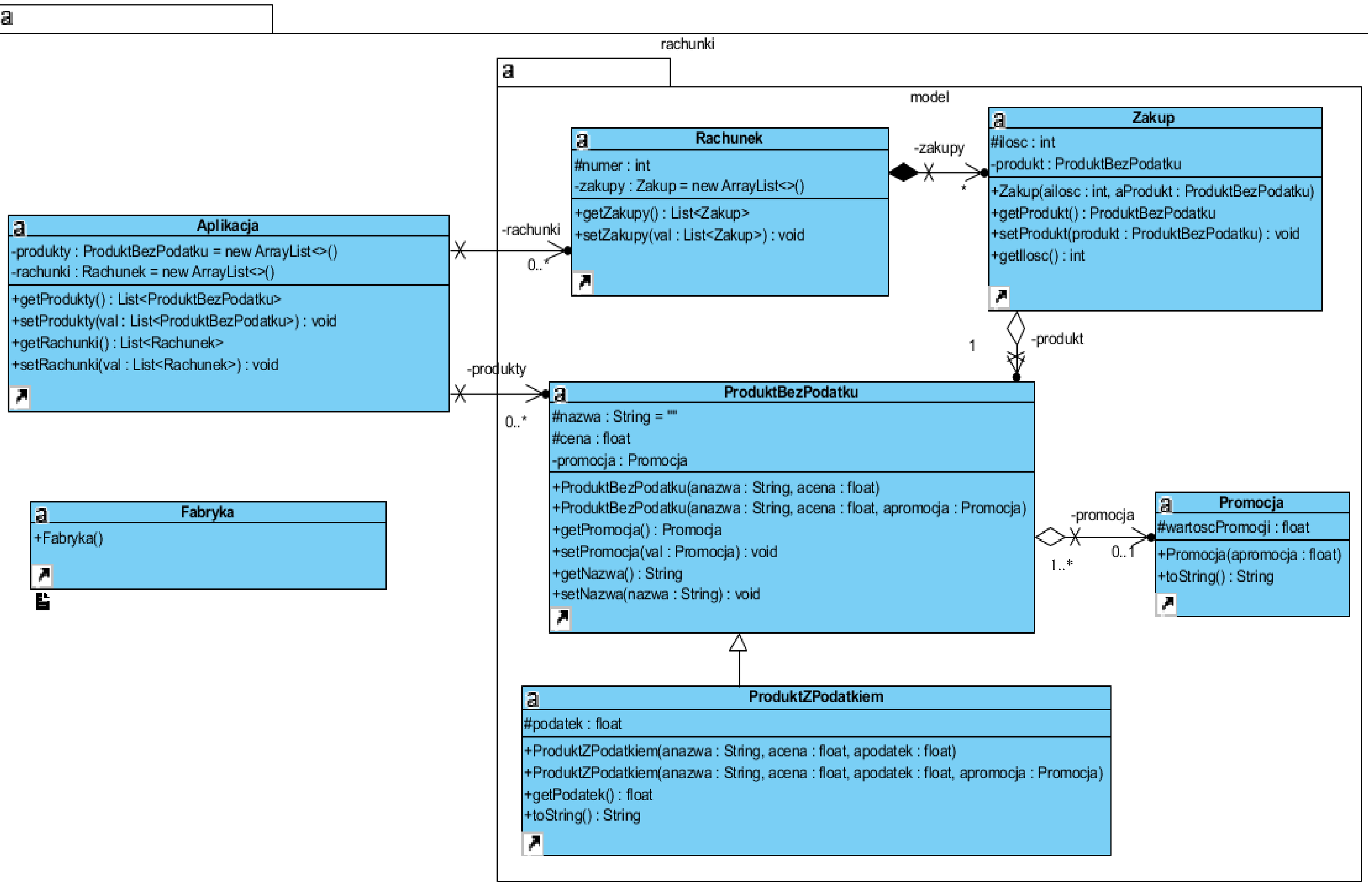
- **silnej agregacji** między obiektem typu **Rachunek** i obiektami typu **Zakup** (rachunek posiada kolekcję zakupów)
- oraz **słabej agregacji** między obiektem typu **Zakup** a obiektem typu **ProduktBezPodatku** (zakup składa się z produktu bazowego lub jego następców)
- Podstawą identyfikacji są PU: **Wstawianie nowego zakupu, Obliczanie wartosci rachunku.**

Analiza zmienności (c.d)

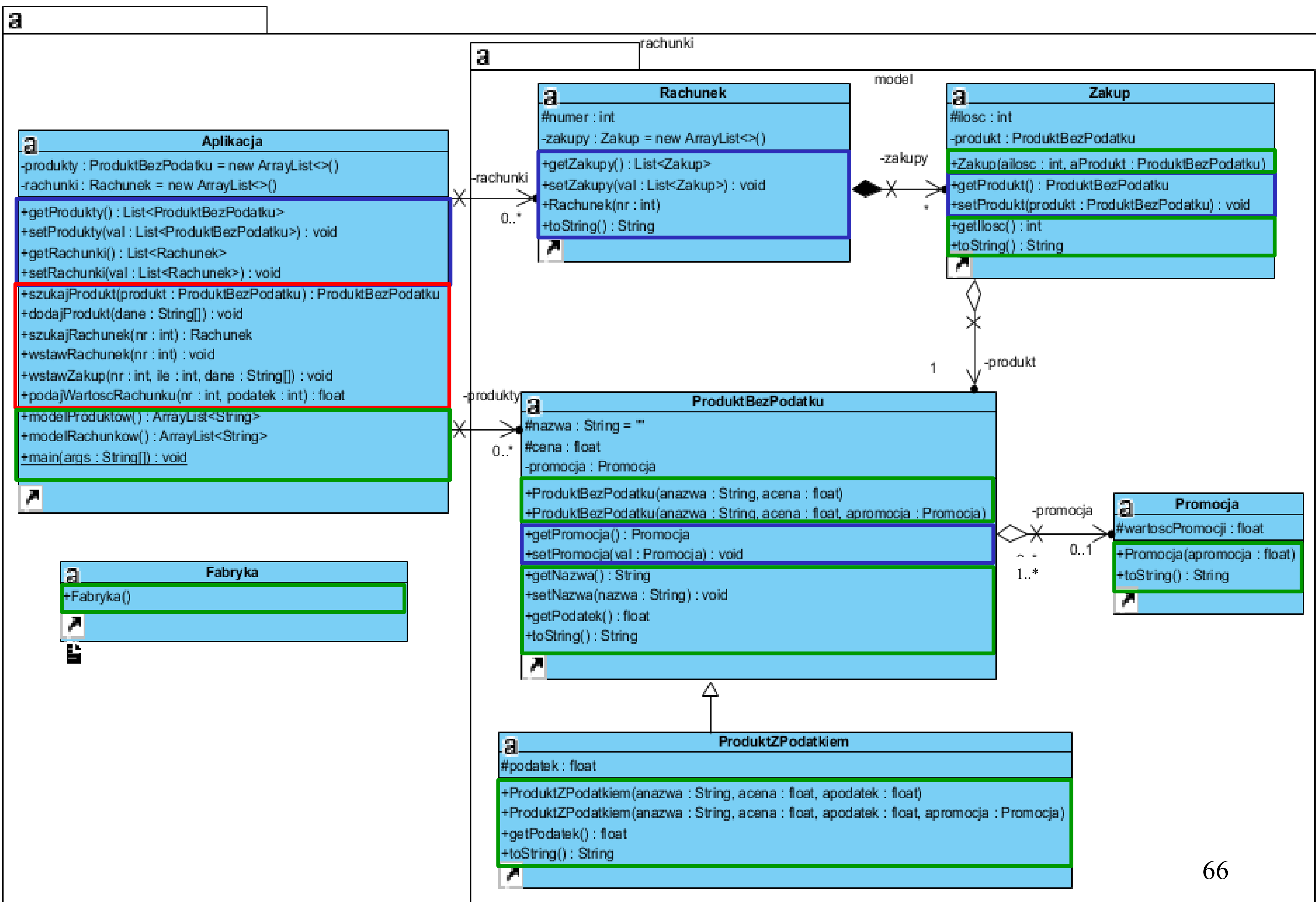
Zastosowano:

- klasę fasadową **Aplikacja** typu **„Control”** do oddzielenia przetwarzania obiektów typu „**Entity**” od pozostałej części systemu
- klasę typu **„Control”** jako fabrykę obiektów (**Fabryka**) do tworzenia różnych typów produktów – czyli obiektów typu **ProduktBezPodatku** i **ProduktZPodatkiem**.

Analiza wspólności i zmienności - rezultat



Projekt powiązań | Metody przypadków użycia | Decyzja projektowa



```java
package rachunki;
import java.util.ArrayList;
import java.util.List;
import rachunki.model.*;
public class Aplikacja
{   private List<ProduktBezPodatku> produkty = new ArrayList<>();
    private List<Rachunek> rachunki = new ArrayList<>();

    List<ProduktBezPodatku> getProdukty ()                                   { return null;  }
    void setProdukty (ArrayList<ProduktBezPodatku> val)                      {   }

    List<Rachunek> getRachunki ()                                            { return null;  }
    public void setRachunki (ArrayList<Rachunek> val)                        {   }

    public void wstawZakup (int nr, int ile, String dane[])                  {   }
    public Rachunek szukajRachunek (int nr)                                  { return null;  }
    public void wstawRachunek (int nr)                                       {   }
    public float podajWartoscRachunku (int nr, int podatek_)                 { return 0.0f;  }
    public Produkt1 szukajProdukt (ProduktBezPodatku produkt)                { return null;  }
    public void dodajProdukt (String[] dane)                                 {   }

    public ArrayList<String> modelProduktow ()                               {return null;  }
    public ArrayList<String> modelRachunkow ()                               {return null;  }
    public static void main (String[] args)                                  {   }
//metoda main służy jedynie do ręcznego testowania głównych metod logiki biznesowej
}
```

Dziękuję za uwagę