Diagramy klas, diagramy sekwencji – tworzenie modeli analizy i projektu

Zofia Kruczkiewicz

Diagramy klas, diagramy sekwencji – tworzenie modeli analizy i projektu

1. **Syntaktyka diagramów klas**

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

2. **Identyfikacja elementów diagramów klas**

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

3. **Diagramy sekwencji UML**

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

4. **Przykłady diagramów sekwencji – kontynuacja przykładu 3 z wykładów: 2, 3, 4**

Diagramy klas, diagramy sekwencji– tworzenie modeli analizy i projektu

1. Syntaktyka diagramów klas

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

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.

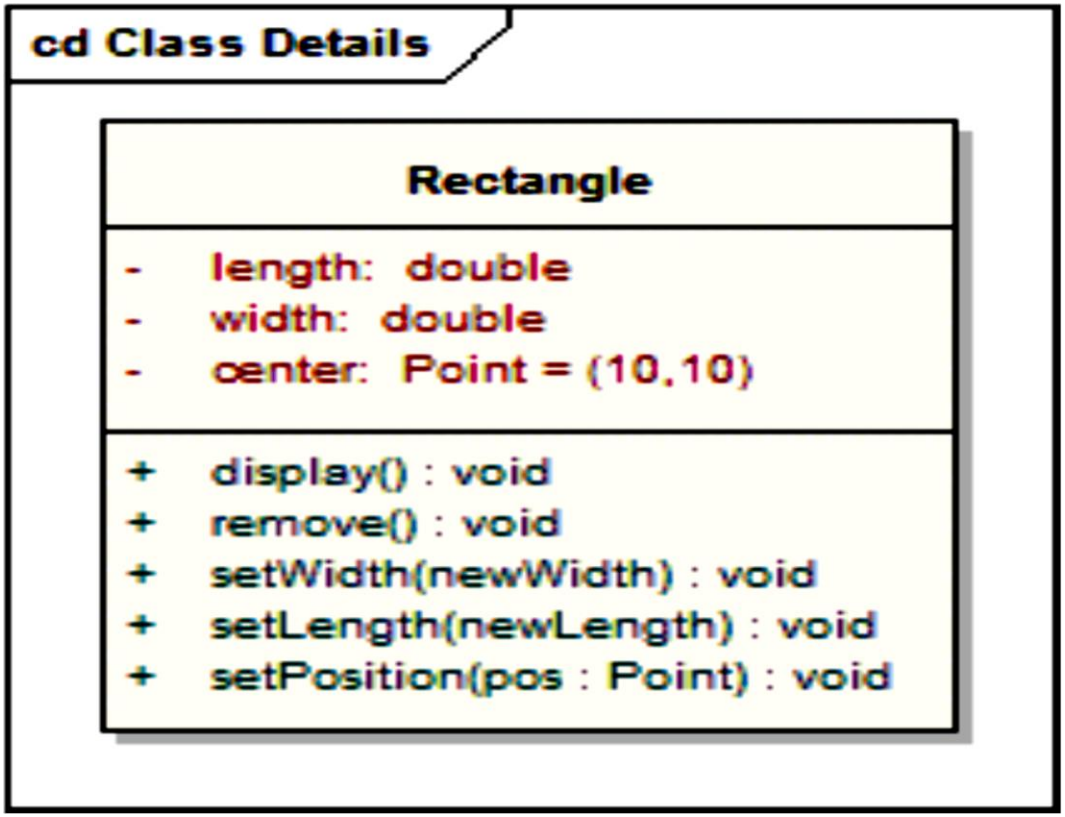
Operacje: setWidth, setLength, setPosition

+ składowa publiczna

- składowa prywatna

składowa typu protected

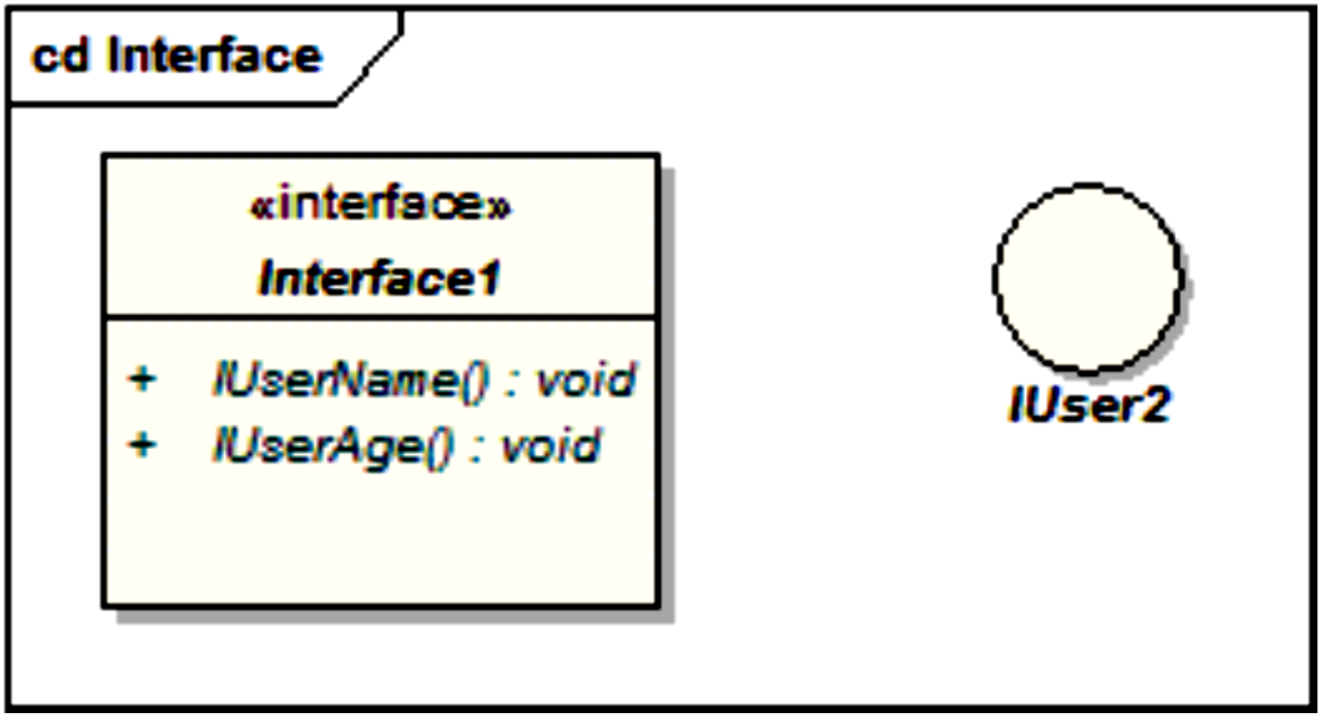
~ składowa publiczna w zasięgu pakietu

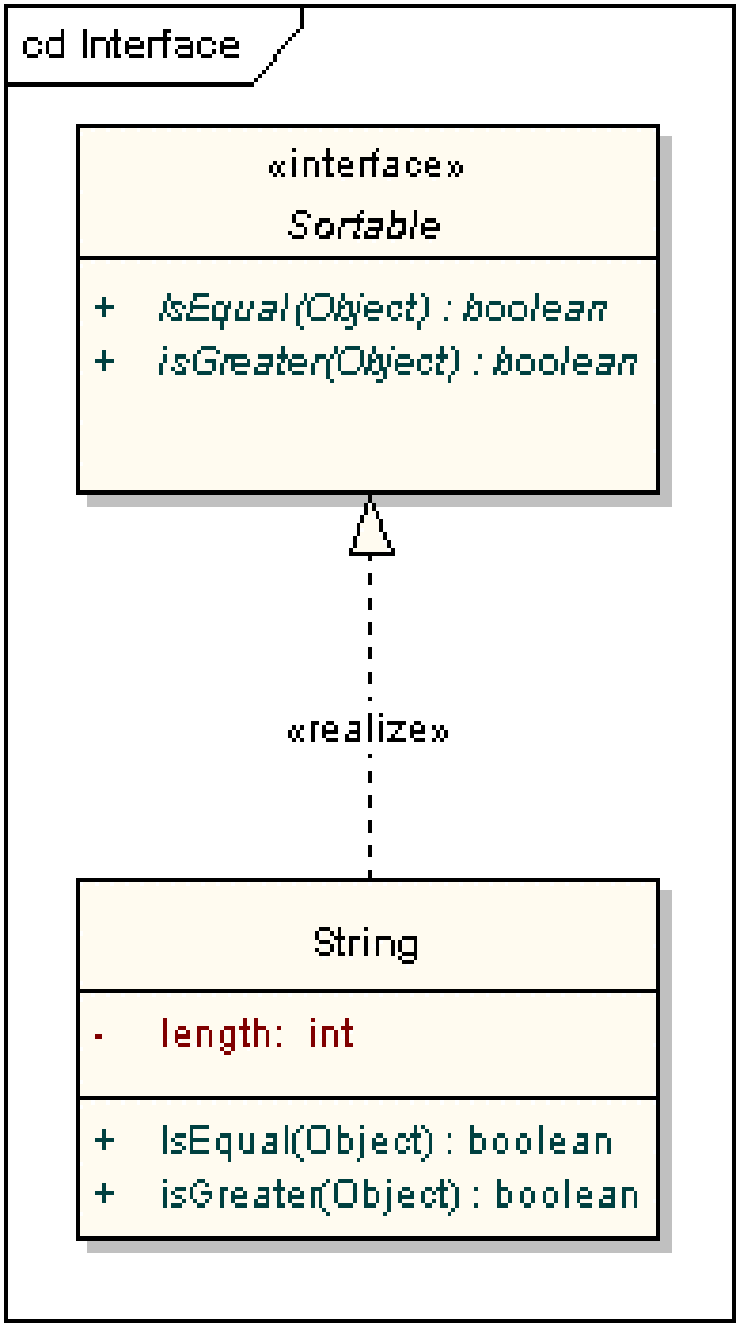


Interfejs<<interface>>

Jest przedstawiany jako:

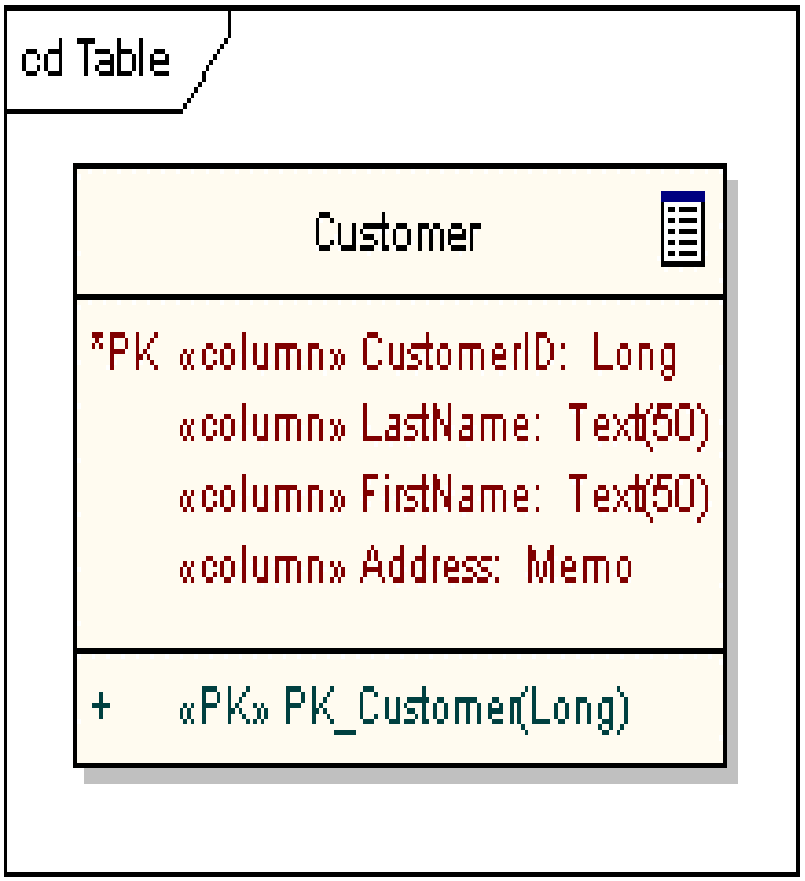
- klasa zawierająca specyfikację właściwości (operacji czyli metod), które musi zdefiniować implementująca go klasa
- **reprezentowany jest jako koło** bez wyspecyfikowanych metod i połączenia z interfejsem przez klasę implementującą nie są oznaczane strzałką





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



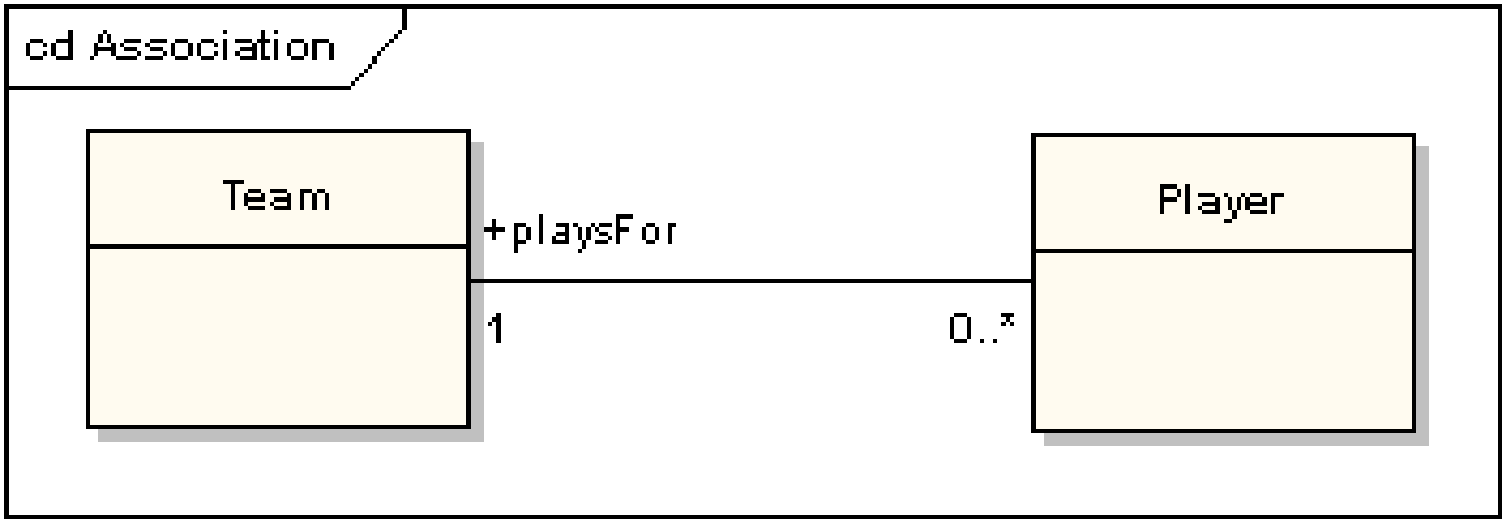
Tabele (table)

- **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
- dla większej liczby powiązanych elementów jest przedstawiana jako romb

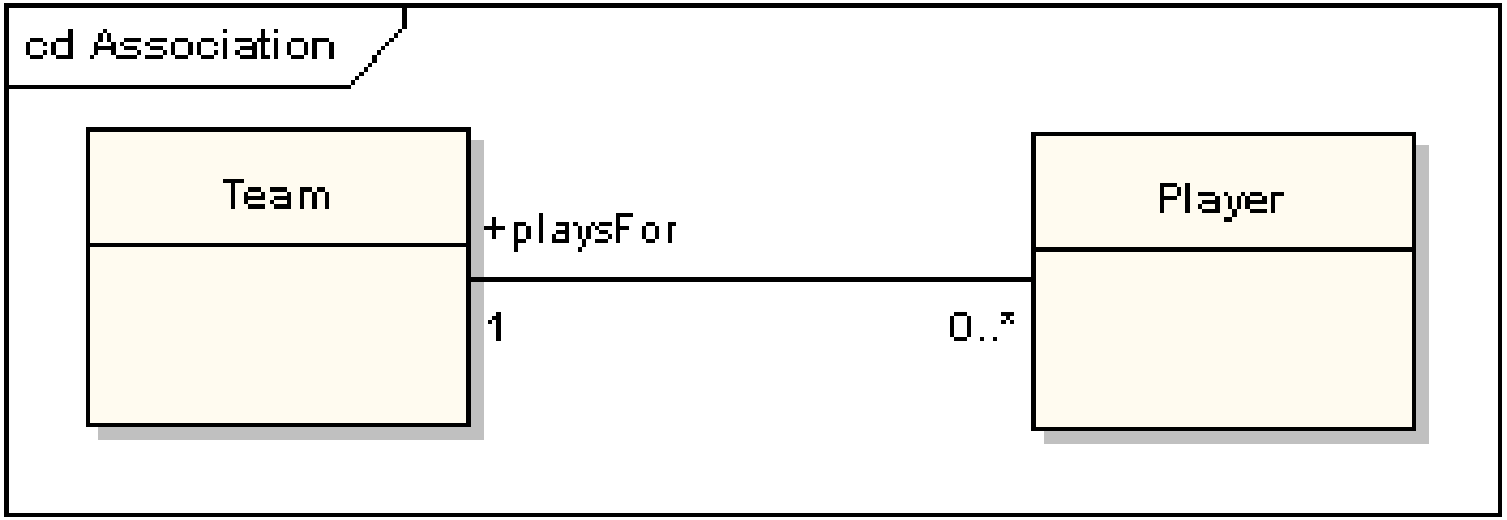


- **jest implementowana następująco:**
  1. **relacje wiele do jeden** lub **jeden do jeden: w obiekcie po stronie wiele lub jeden znajduje się referencja do obiektu z przeciwnej strony relacji (strony jeden)**
  2. **relacje jeden do wiele: kolekcja referencji instancji obiektów po stronie wiele w obiekcie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

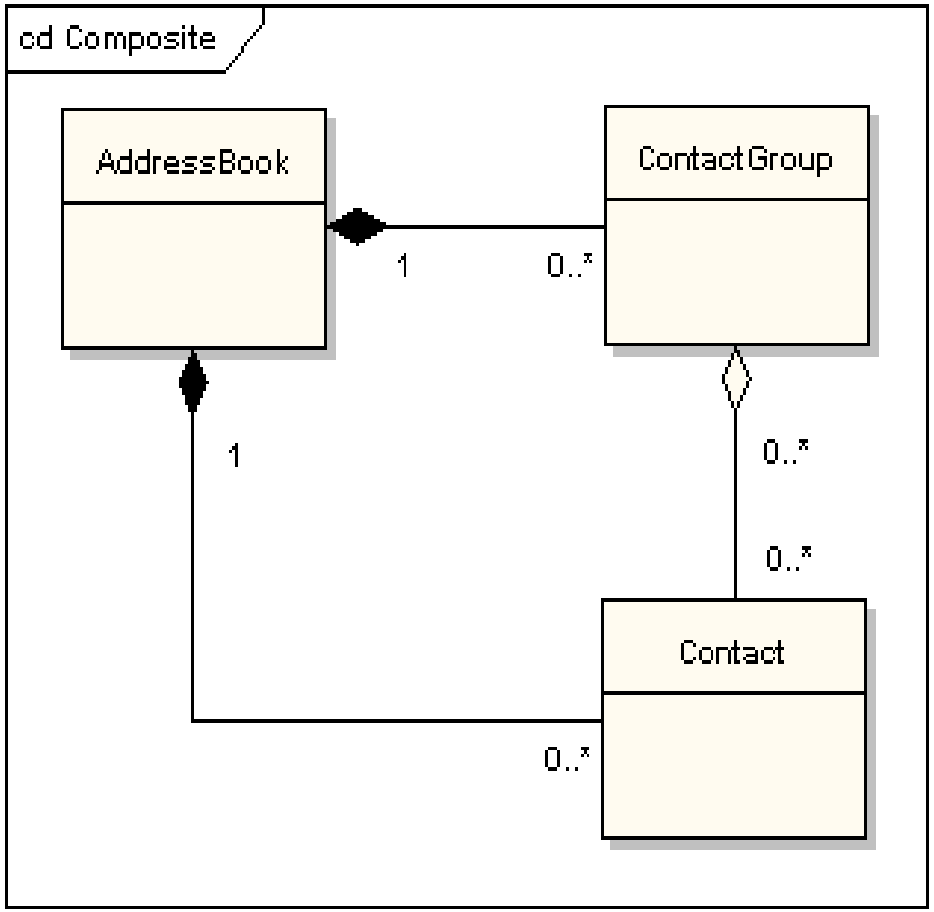
  (np. referencja do obiektu typu *Team* występuje w obiekcie typu *Player jako atrybut* oraz kolekcja referencji obiektów typu *Player* w obiekcie klasy *Team jako atrybut*)

Role

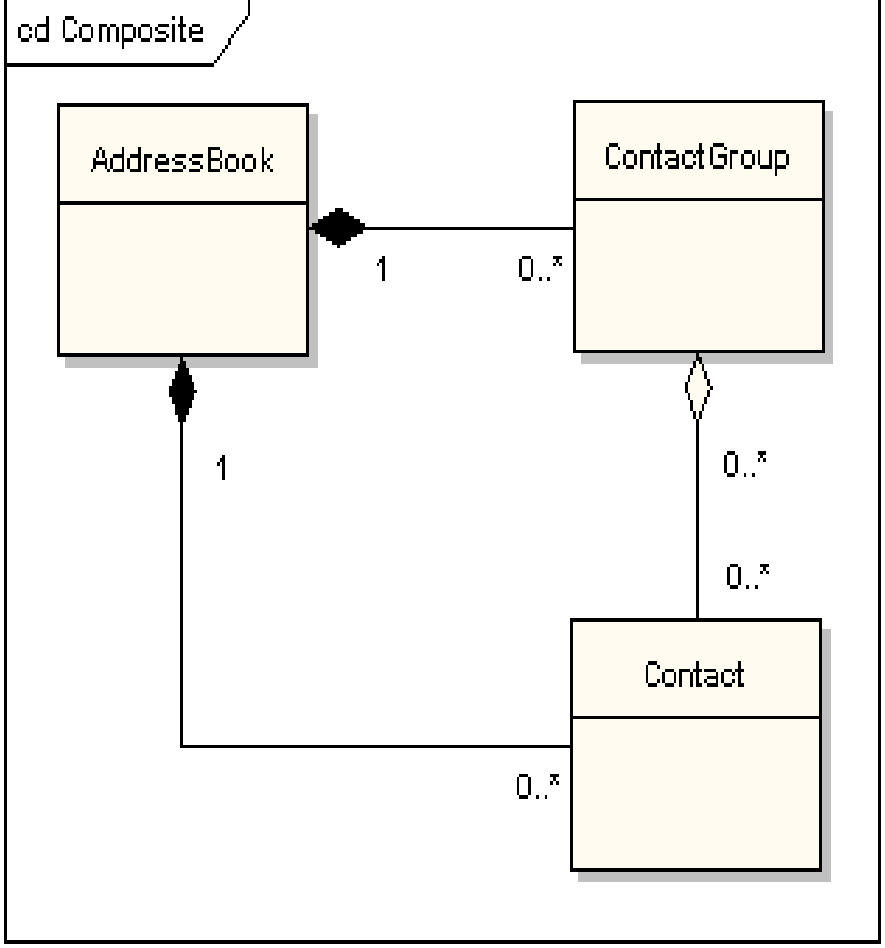
Role to oblicze, jakie prezentuje klasa przy jednym końcu drugiej klasie na drugim końcu asocjacji (instancja typu *Player* gra (*+playsFor*) dla instancji typu *Team*)



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

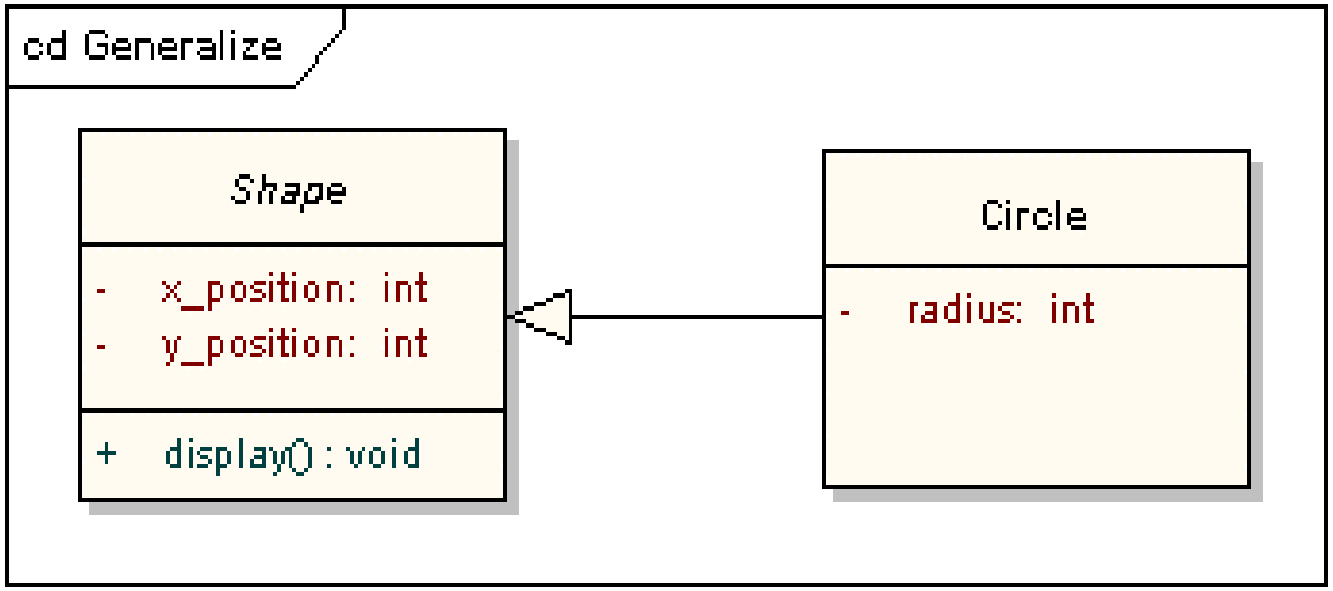


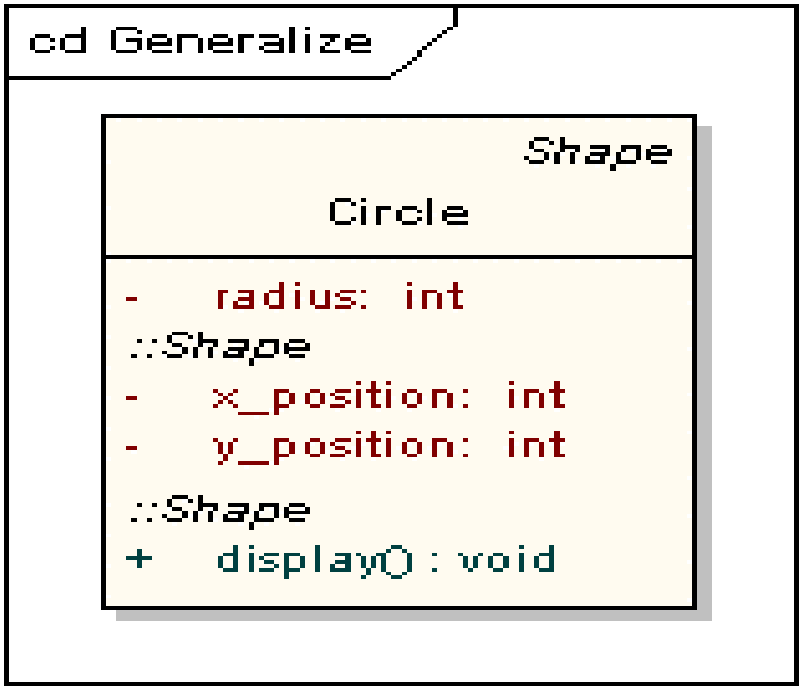
- **jest implementowana podobnie jak związek typu Association**

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 *Contact Gropup* nie pociąga za sobą usuwania obiektów klasy *Contact.*

Generalizacja czyli dziedziczenie (Generalization)

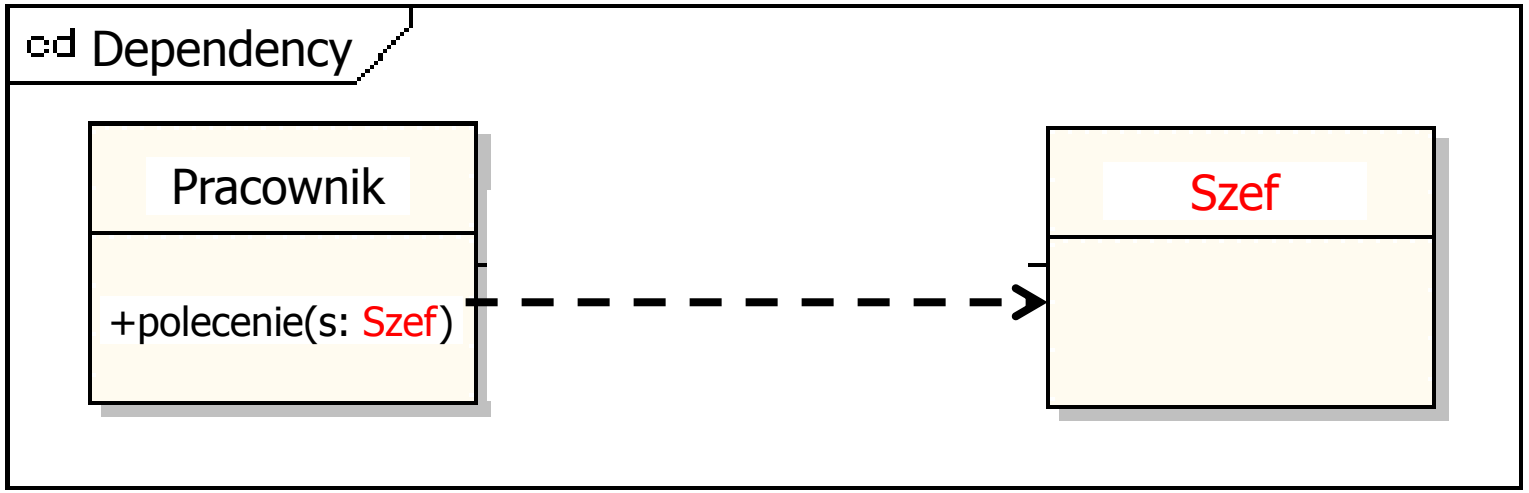




Używana do oznaczania **dziedziczenia**

- **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)



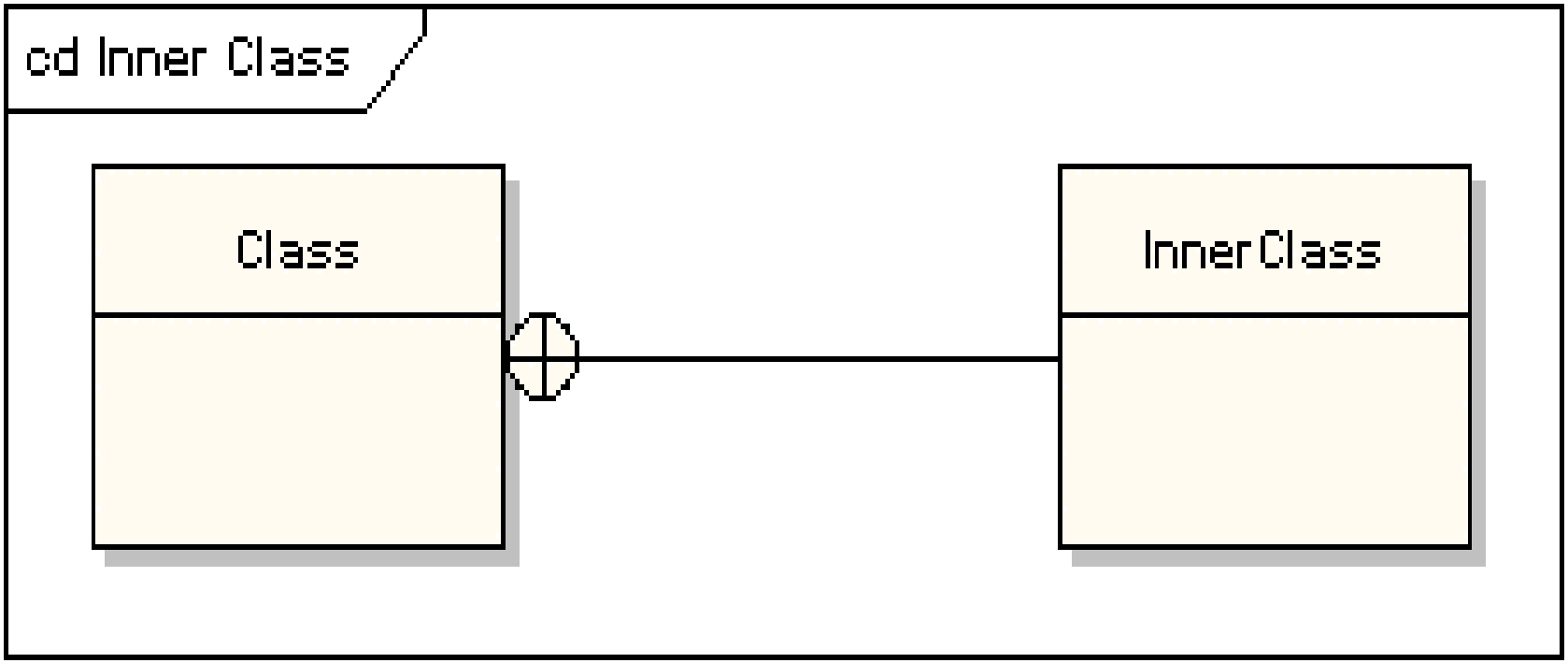
- zależności są używane do modelowania powiązań między elementami modelu we wczesnej fazie projektowania, jeśli nie można określić precyzyjnie typu powiązania. Stanowią one wtedy związek użycia **(<<usage>>)**.
- strałka przerywana wskazuje grotem na klasę, od której coś zależy.
- Później są one uzupełniane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

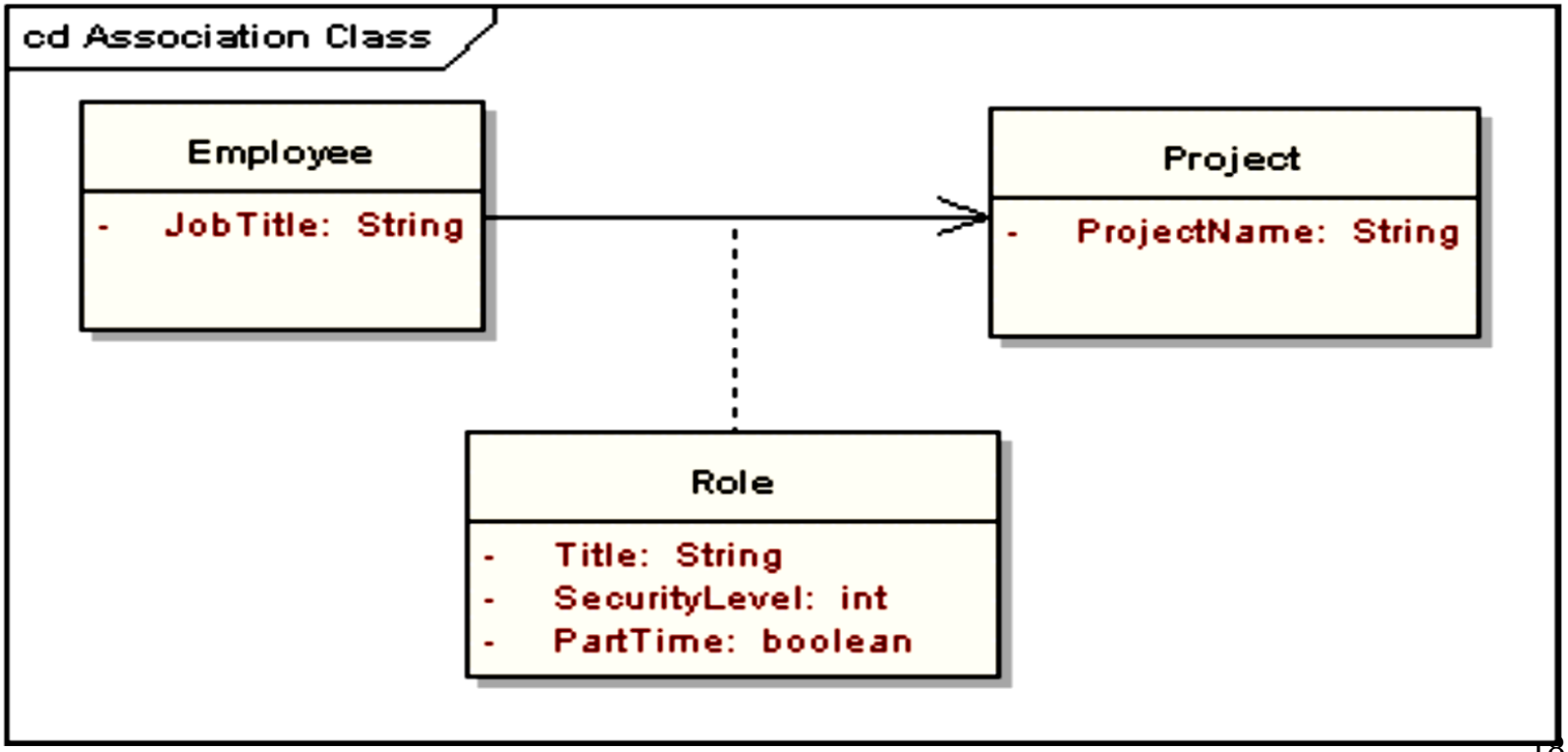
Zagnieżdżenie (Nesting)

- 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***

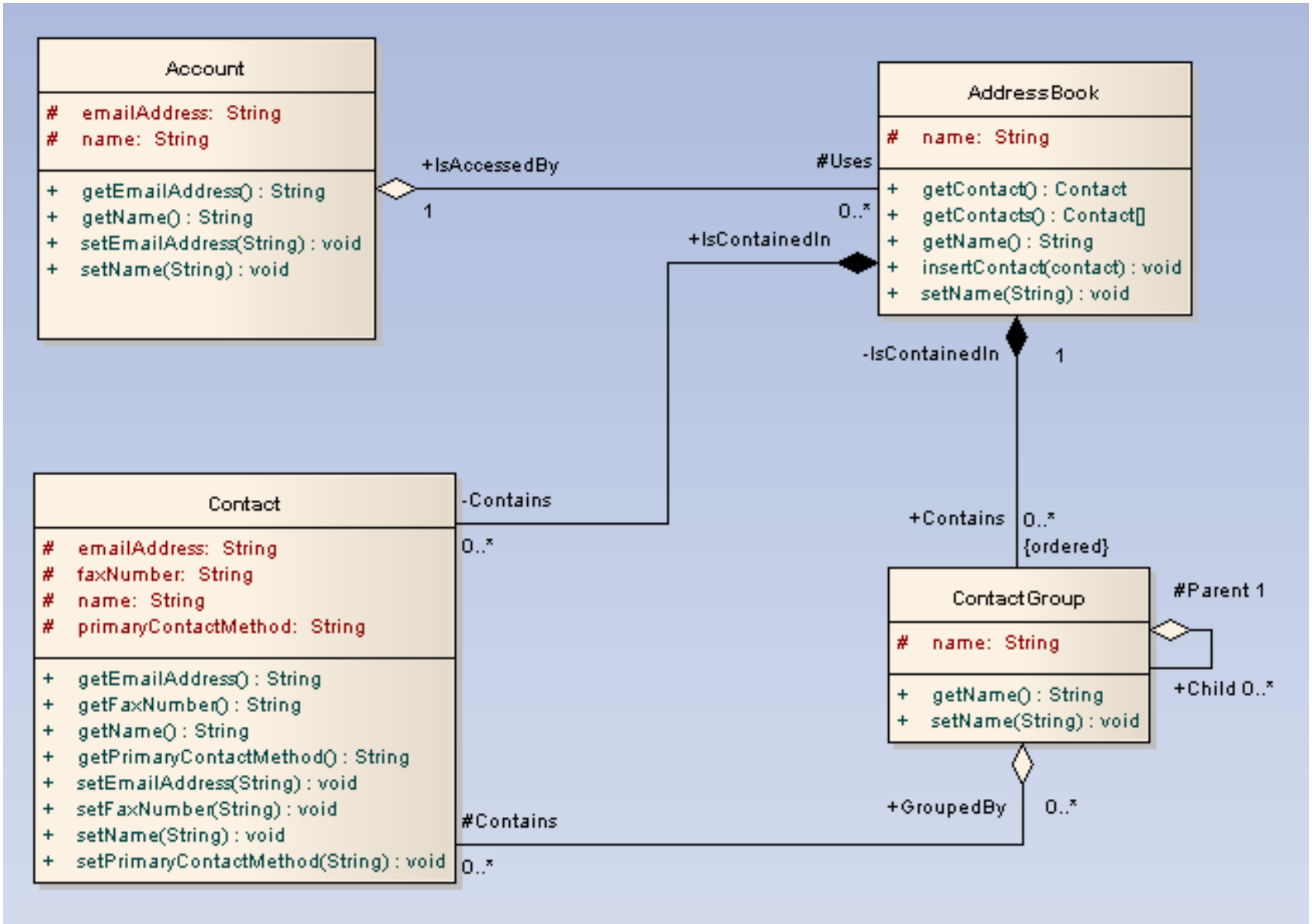


Klasa powiązań (Association Class)

- 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



Podsumowanie – diagram klas



Diagramy klas, diagramy sekwencji – tworzenie modeli analizy i projektu

1. **Syntaktyka diagramów klas**

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

2. **Identyfikacja elementów diagramów klas**

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

Modele analizy i projektu – typy klas na diagramach klas

Produkt	Opis produktu (reprezentowanego w języku UML)
a) klasy typu „Control” Warstwy: prezentacji **biznesowa,** integracji	•reprezentują koordynację (*coordination*), sekwencje (*sequencing*), transakcje (*transactions*), sterowanie (*control*) •często są używane do hermetyzacji sterowania odniesionego do przypadku użycia dla każdej warstwy tzn hermetyzują warstwę biznesową dla warstwy prezentacji oraz warstwę integracji dla warstwy biznesowej; •klasy te modelują dynamikę systemu czyli główne akcje (*actions*) i przepływ sterowania (*control flows*) i przekazują działania do klas warstwy prezentacji, biznesowej oraz integracji ;

b) klasy typu „Entity” - formalnie obiekty realizowane przez system, często przedstawiane jako logiczne struktury danych (*logical data structure*) Warstwa biznesowa	• używane do modelowania informacji o długim okresie istnienia i często niezmiennej (*persistent*); • klasy realizowana jako obiekty typu „real-life” lub zdarzenia typu „real-life”; • są wyprowadzane z modelu analizy • mogą zawierać specyfikację złożonego zachowania reprezentowanej informacji
c) klasy typu „Boundary” Warstwa klienta	• klasy te reprezentują abstrakcje: okien, formularzy, interfejsów komunikacyjnych, interfejsów drukarek, sensorów, terminali i API (również nieobiektowych); • jedna klasa odpowiada jednemu użytkownikowi typu aktor • używane do modelowania interakcji między systemem i aktorami czyli użytkownikami (*users*) lub zewnętrznymi systemami;

Identyfikacja klas - zasady

(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) - zasady

(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) - zasady

(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
- 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) - zasady

(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 – przykład cd

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

Przykład 3 z wykładu 2 i jego kontynuacja

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

- **TRachunek** (PU: **Wstawianie nowego rachunku, Wstawianie nowego zakupu, Obliczanie wartosci rachunku**),
- **TZakup** (PU: **Wstawianie nowego zakupu, Obliczanie wartosci rachunku**),
- **TProdukt1** (PU: **Wstawianie nowego produktu, Wstawianie nowego zakupu, Obliczanie wartosci rachunku**)

Identyfikacja klas – przykład cd

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

Przykład 3 z wykładu 2 i jego kontynuacja z wykładu 3

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:

- **TProdukt2** typu **„Entity”**, która dziedziczy od klasy **TProdukt1** (PU: **Wstawianie nowego produktu, Obliczanie wartosci rachunku**)

Identyfikacja klas – przykład cd

Analiza zmienności (c.d)

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

- Zdefiniowano klasę **TPromocja** typu **„Entity”**
- Zdefiniowano **związek typu asocjacja** między klasami **TProdukt1 i TPromocja**, który jest dziedziczony przez pozostałe typy produktu tzn. **TProdukt2**. Ponieważ jednak promocja nie musi dotyczyć każdego produktu, jest **w związku asocjacji 0..1 do 0..*** z bazowym (głównym) produktem typu **TProdukt1**
- Dzięki temu produkty typu **TProdukt1** i **TProdukt2** 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.**

Identyfikacja klas – przykład cd
Analiza zmienności (c.d)

Wykryto związki:

- silnej agregacji między obiektem typu **TRachunek** i obiektami typu **TZakup** (rachunek posiada kolekcję zakupów)
- oraz słabej agregacji między obiektem typu **TZakup** a obiektem typu **TProdukt1** (zakup składa się z produktu bazowego lub jego następców)
- Podstawą identyfikacji są PU: **Wstawianie nowego zakupu, Obliczanie wartosci rachunku.**

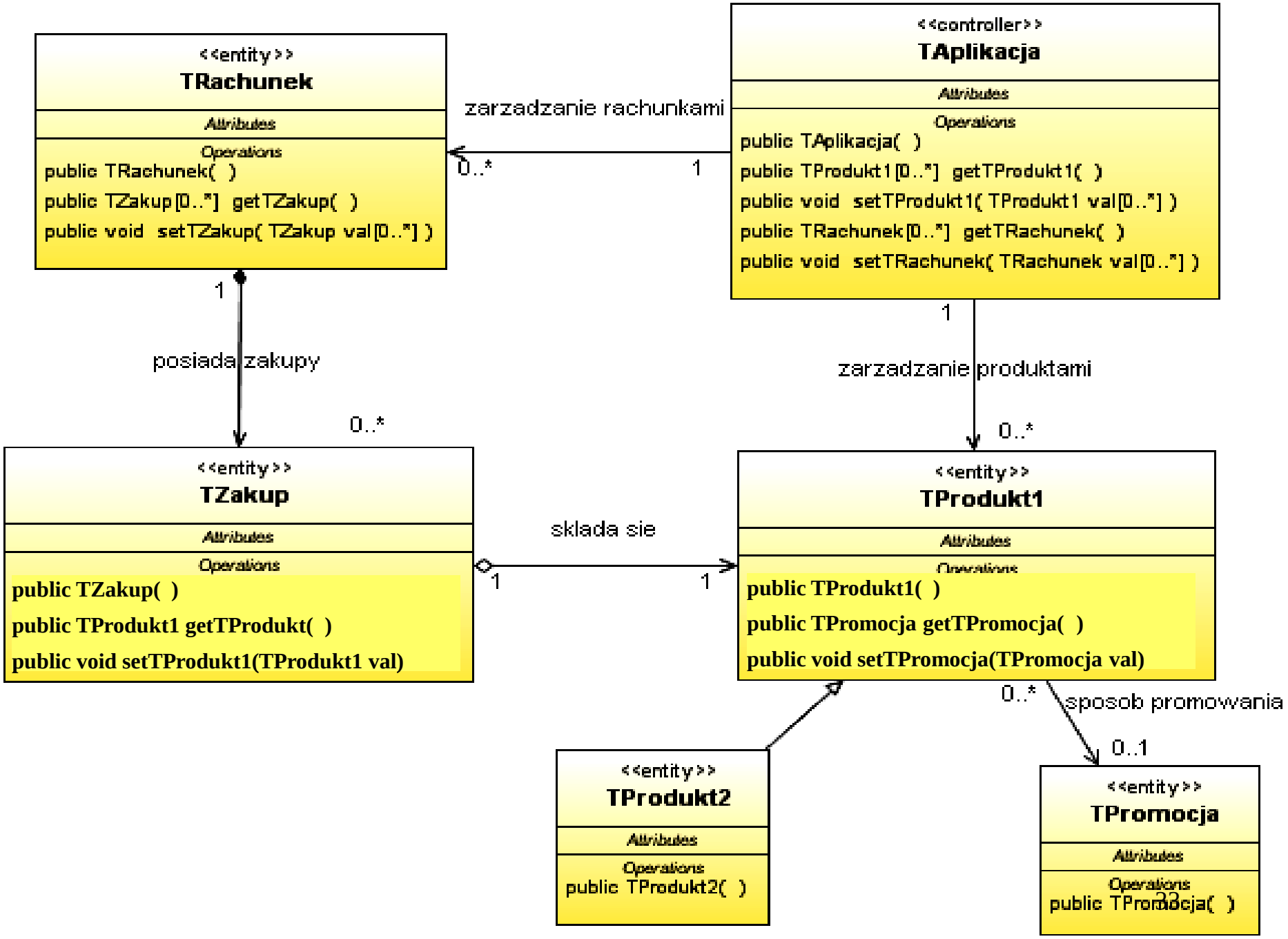
Zastosowano:

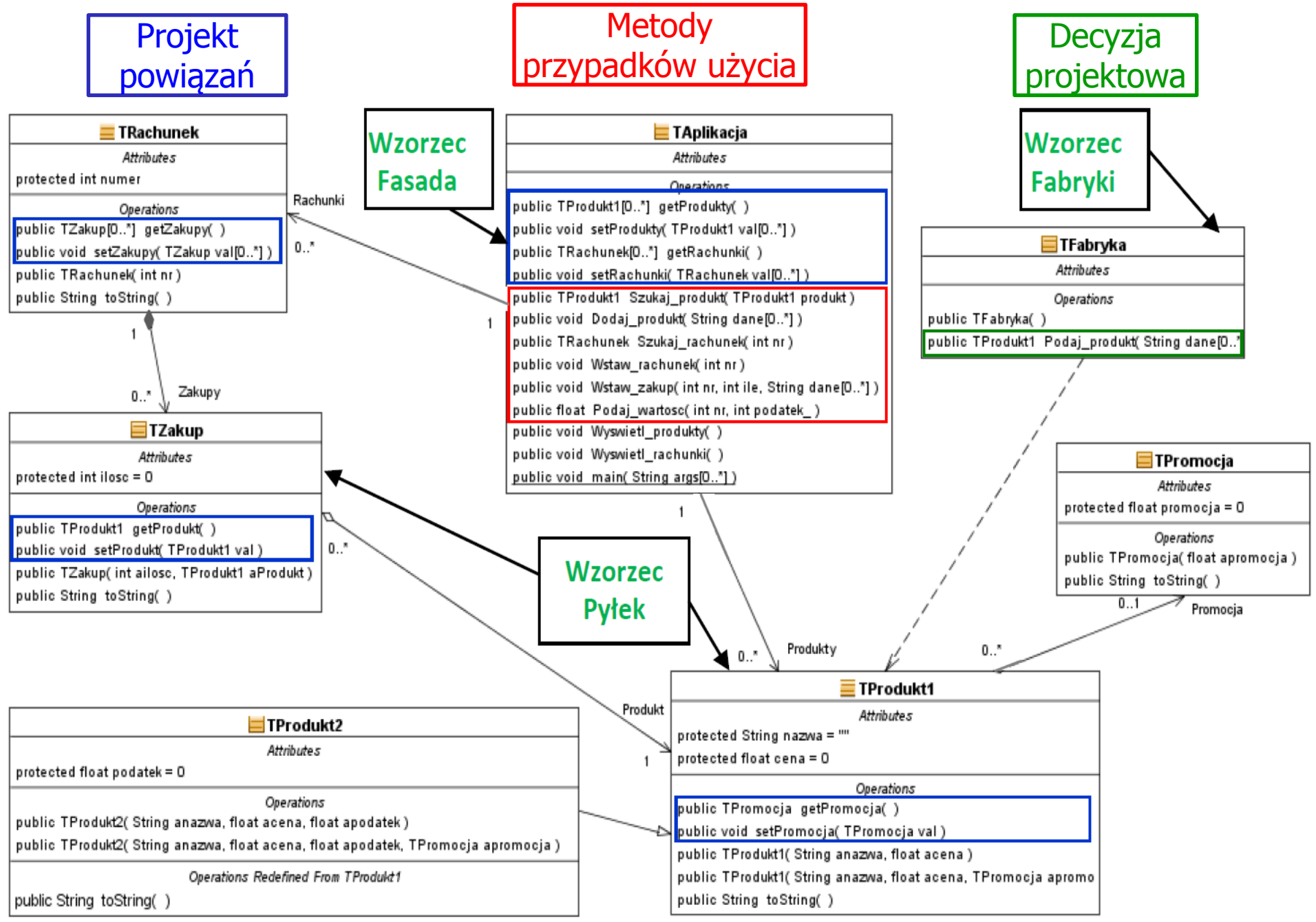
- klasę fasadową **TAplikacja** typu **„Control”** do oddzielenia przetwarzania obiektów typu „**Entity**” od pozostałej części systemu

Identyfikacja klas – przykład cd
Analiza zmienności (c.d)

Zastosowano

- klasę typu **„Control”** jako fabrykę obiektów (**TFabryka**) do tworzenia różnych typów produktów – czyli obiektów typu **TProdukt1** i **TProdukt2**.


«entity»
TRachunek
Attributes
Operations
public TRachunek( )
public TZakup[0..*] getTZakup( )
public void setTZakup(TZakup val[0..*])
«controller»
TAplikacja
Attributes
Operations
public TAplikacja( )
public TProdukt1[0..*] getTProdukt1( )
public void setTProdukt1(TProdukt1 val[0..*])
public TRachunek[0..*] getTRachunek( )
public void setTRachunek(TRachunek val[0..*])
zarzadzanie rachunkami
0..*
1
1
posiada zakupy
1
0..*
zarzadzanie produktami
1
0..*
«entity»
TZakup
Attributes
Operations
public TZakup( )
public TProdukt1 getTProdukt( )
public void setTProdukt1(TProdukt1 val)
sklada sie
1
1
«entity»
TProdukt1
Attributes
public TProdukt1( )
public TPromocja getTPromocja( )
public void setTPromocja(TPromocja val)
0..*
sposob promowania
0..1
«entity»
TProdukt2
Attributes
Operations
public TProdukt2( )
«entity»
TPromocja
Attributes
Operations
public TPromocja( )


Projekt powiązań
Metody przypadków użycia
Decyzja projektowa
Wzorzec Fasada
Wzorzec Fabryki
Wzorzec Pyłek
TRachunek
Attributes
protected int numer
Operations
public TZakup[0..*] getZakupy( )
public void setZakupy(TZakup val[0..*])
public TRachunek(int nr)
public String toString( )
Rachunki
0..*
1
0..*
Zakupy
TAplikacja
Attributes
Operations
public TProdukt1[0..*] getProdukty( )
public void setProdukty(TProdukt1 val[0..*])
public TRachunek[0..*] getRachunki( )
public void setRachunki(TRachunek val[0..*])
public TProdukt1 Szukaj_produkt(TProdukt1 produkt)
public void Dodaj_produkt(String dane[0..*])
public TRachunek Szukaj_rachunek(int nr)
public void Wstaw_rachunek(int nr)
public void Wstaw_zakup(int nr, int ile, String dane[0..*])
public float Podaj_wartosc(int nr, int podatek_)
public void Wyswietl_produkty( )
public void Wyswietl_rachunki( )
public void main(String args[0..*])
1
TFabryka
Attributes
Operations
public TFabryka( )
public TProdukt1 Podaj_produkt(String dane[0..*
TZakup
Attributes
protected int ilosc = 0
Operations
public TProdukt1 getProdukt( )
public void setProdukt(TProdukt1 val)
public TZakup(int ailosc, TProdukt1 aProdukt)
public String toString( )
0..*
TPromocja
Attributes
protected float promocja = 0
Operations
public TPromocja(float apromocja)
public String toString( )
0..1
Promocja
0..*
Produkty
0..*
TProdukt1
Attributes
protected String nazwa = ""
protected float cena = 0
Operations
public TPromocja getPromocja( )
public void setPromocja(TPromocja val)
public TProdukt1(String anazwa, float acena)
public TProdukt1(String anazwa, float acena, TPromocja apromo
public String toString( )
Produkt
1
TProdukt2
Attributes
protected float podatek = 0
Operations
public TProdukt2(String anazwa, float acena, float apodatek)
public TProdukt2(String anazwa, float acena, float apodatek, TPromocja apromocja)
Operations Redefined From TProdukt1
public String toString( )

```
package rachunek1;
import java.util.ArrayList;
import java.util.List;
public class TAplikacja
{    private List<TProdukt1> Produkty = new ArrayList<TProdukt1>();
     private List<TRachunek> Rachunki = new ArrayList<TRachunek>();
     List<TProdukt1> getProdukty ()                          { return null;  }
     void setProdukty (ArrayList<TProdukt1> val)             {    }
     List<TRachunek> getRachunki ()                          { return null;  }
     public void setRachunki (ArrayList<TRachunek> val)      {    }
     public void Wstaw_zakup (int nr, int ile, String dane[]) {    }
     public TRachunek Szukaj_rachunek (int nr)               { return null;  }
     public void Wstaw_rachunek (int nr)                     {    }
     public float Podaj_wartosc (int nr, int podatek_)       { return 0.0f;  }
     public TProdukt1 Szukaj_produkt (TProdukt1 produkt)     { return null;  }
     public void Dodaj_produkt (String[] dane)               {    }
     public void Wyswietl_produkty ()                        {    }
     public void Wyswietl_rachunki ()                        {    }
     public static void main (String[] args)                 {    }
}
```

Diagramy klas, diagramy sekwencji – tworzenie modeli analizy i projektu

1. **Syntaktyka diagramów klas**

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

2. **Identyfikacja elementów diagramów klas**

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

3. **Diagramy sekwencji UML**

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

Diagramy UML 2 – część czwarta

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**

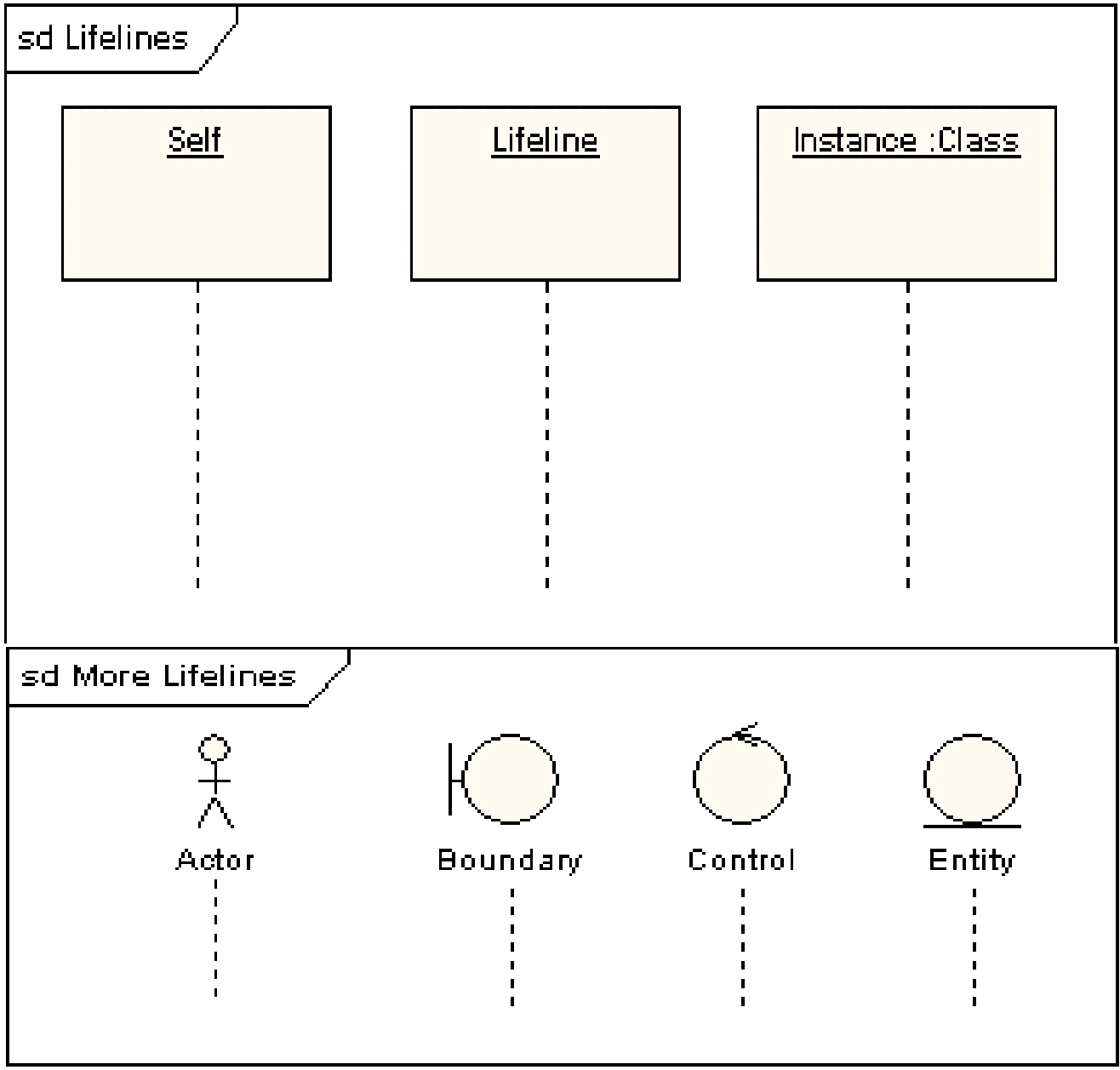
Diagramy sekwencji (Sequence Diagrams)

- 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**
- **są używane do modelowanie scenariusza przypadku użycia**

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 diagramu sekwencji jest **przypadek użycia**. Elementy oznaczające ***obiekty typu „boundary”, „control”*** **,** ***„Entity”*** mają również swoje linie życia.

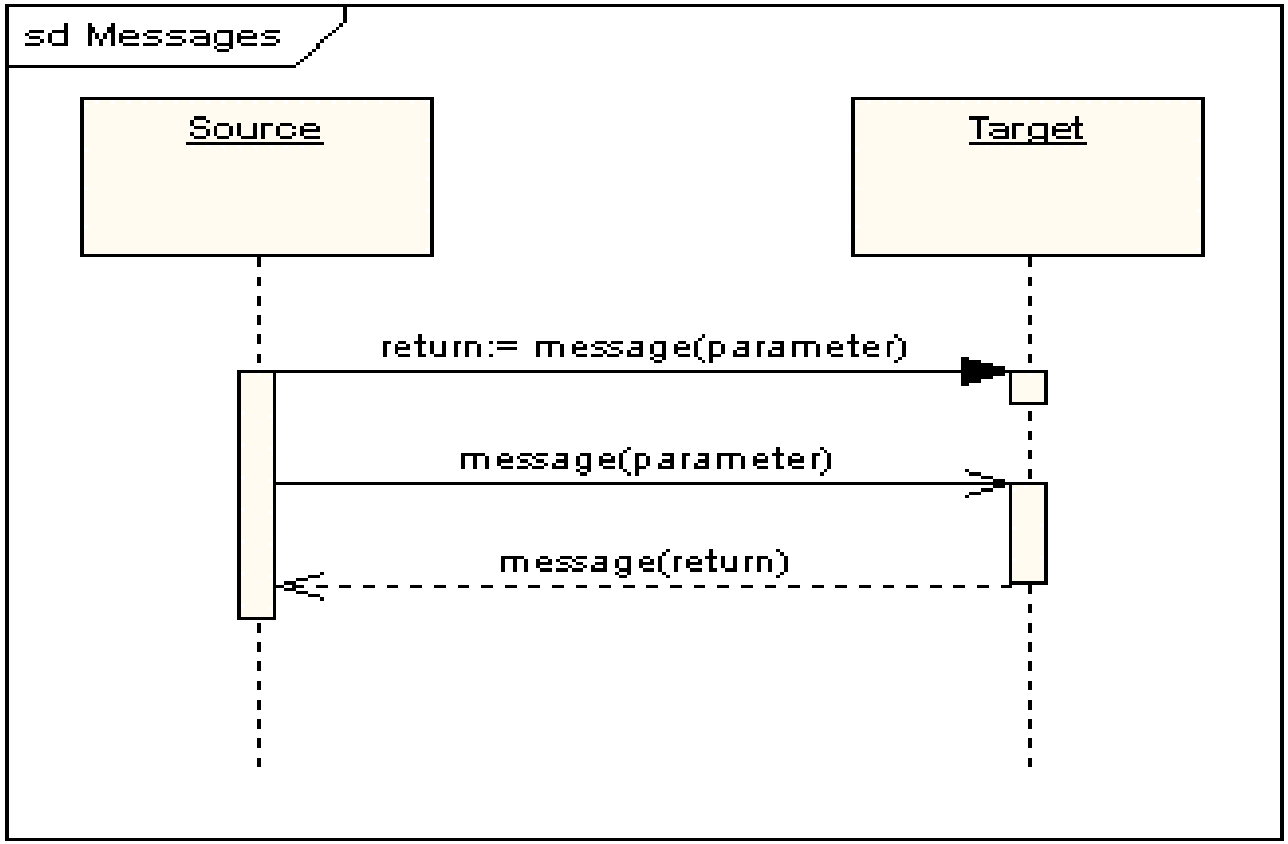


Wiadomości (Messages)

- 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**

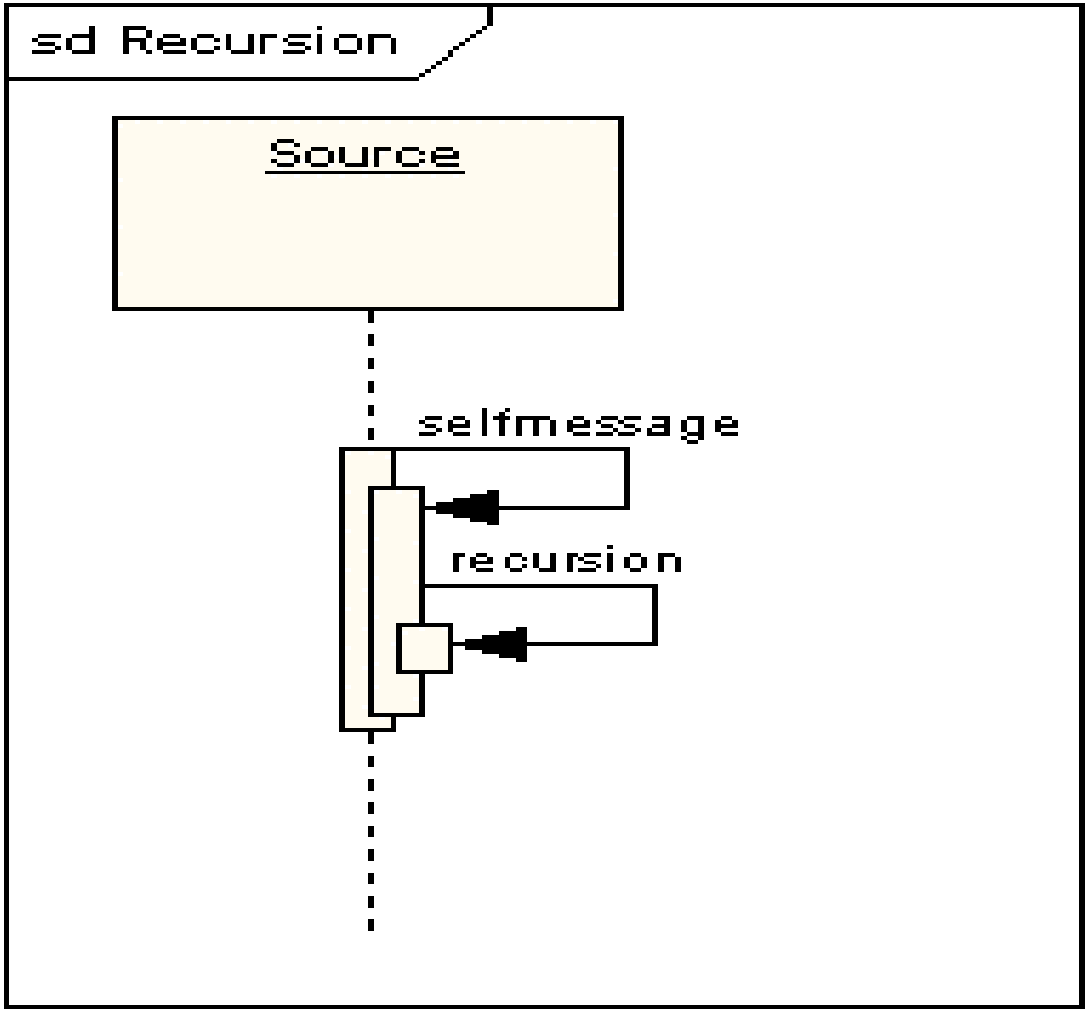
Wykonywanie interakcji (Execution Occurrence)

1. pierwsza wiadomość jest synchroniczna, kompletna i posiada return (wywołanie metody obiektu Target przez obiekt przez Source),
2. druga wiadomość jest asynchroniczna (wywołanie metody obiektu Target przez obiekt przez Source),
3. trzecia wiadomość jest asynchroniczną wiadomością typu return (przerywana linia – return metody asynchronicznej obiektu Target).



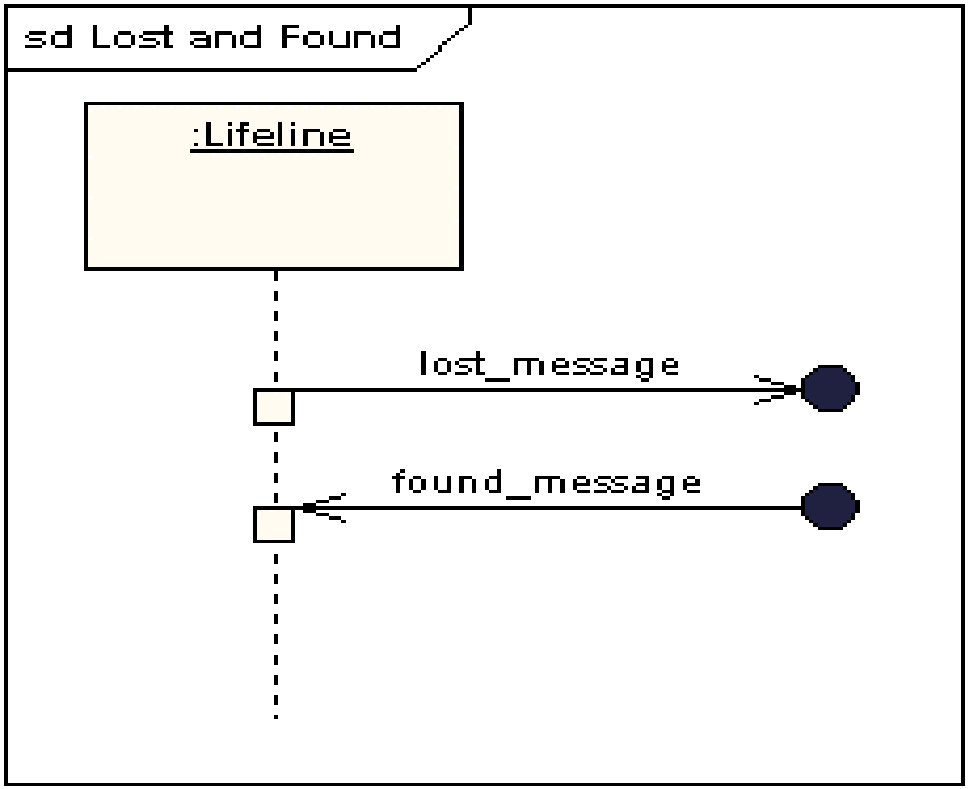
Własne wiadomości (Self Message)

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



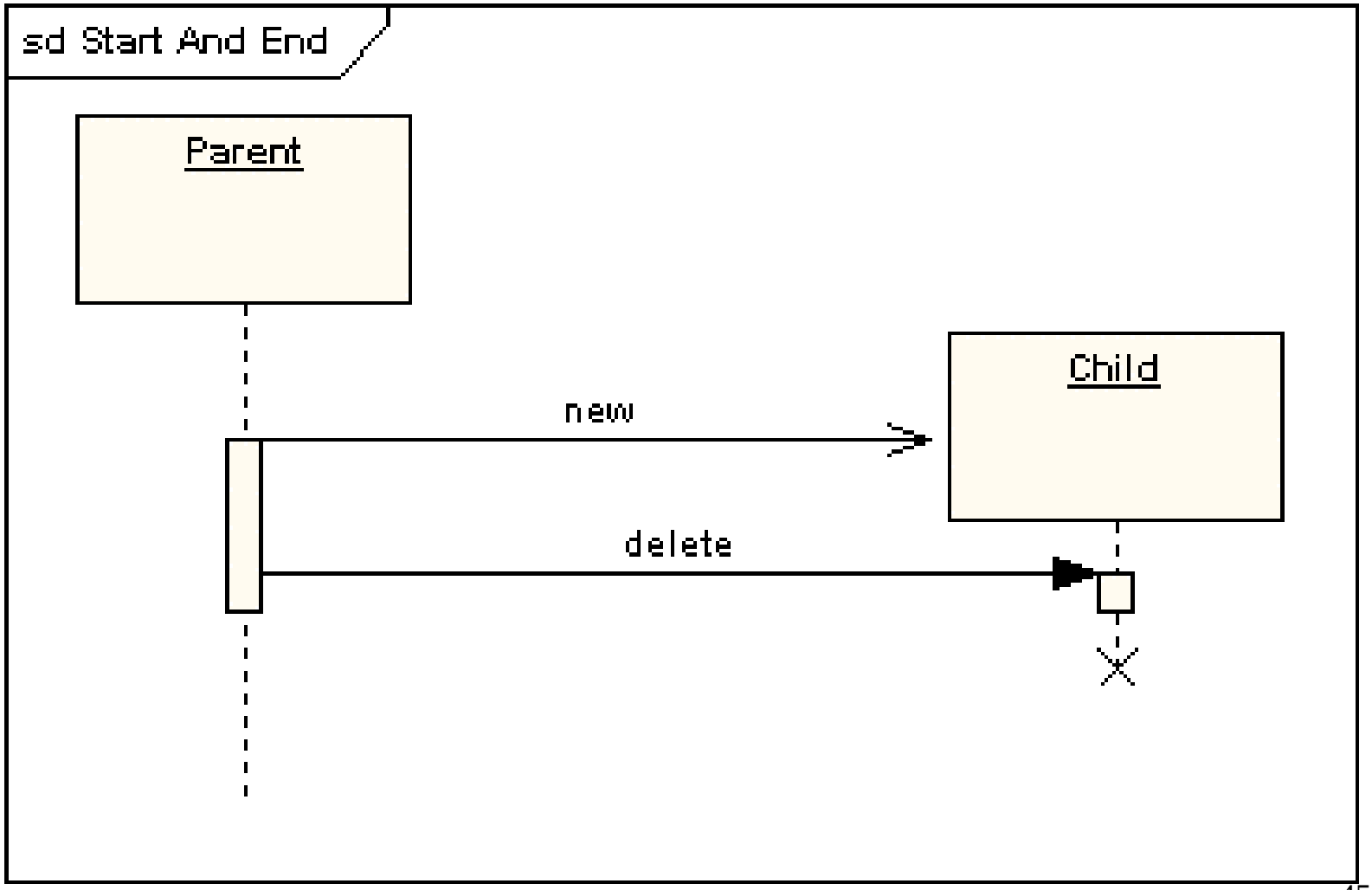
Zgubione i znalezione wiadomości (Lost and Found Messages)

- **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, który nie jest pokazany na bieżącym diagramie.



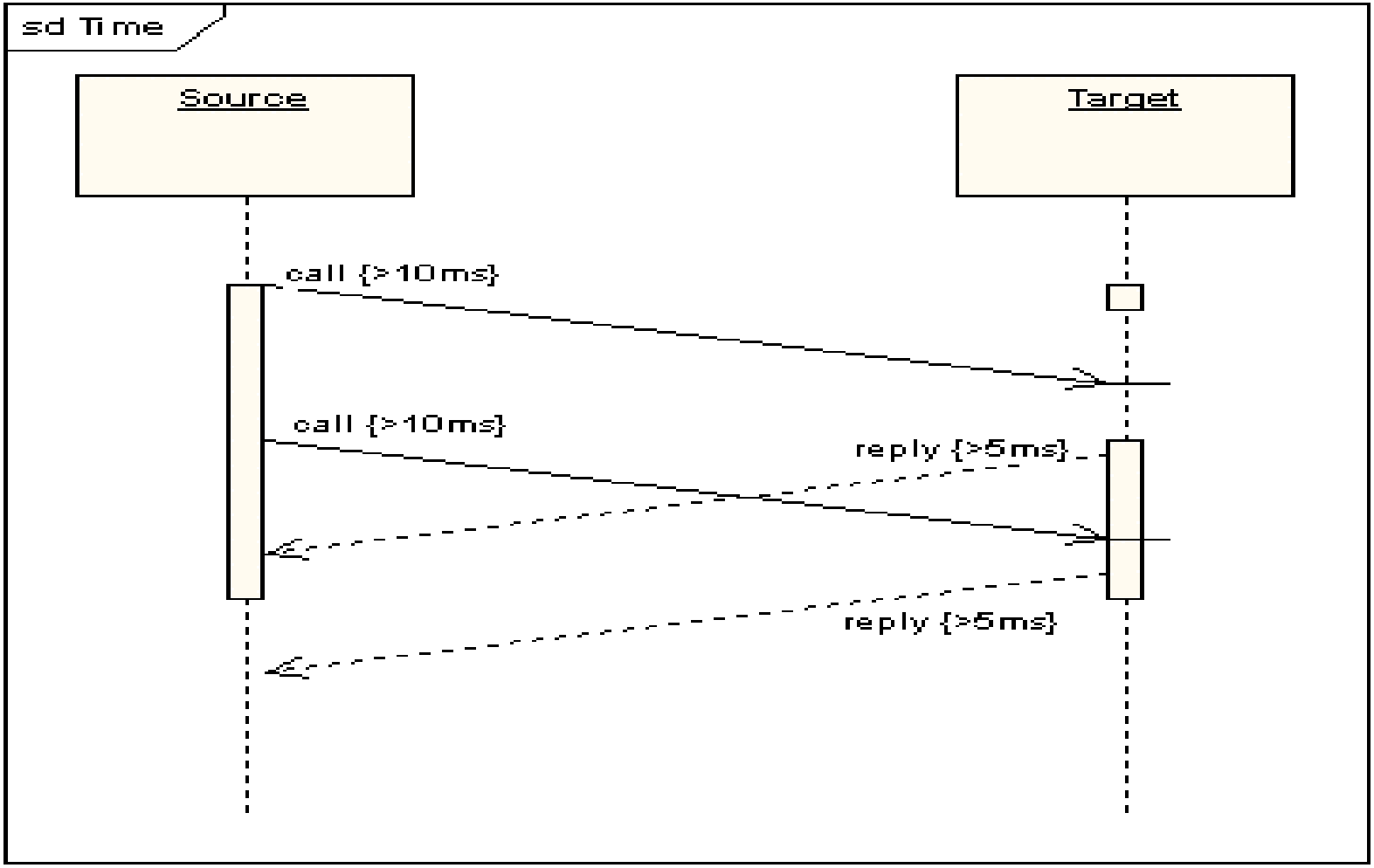
Start linii życia i jej koniec (Lifeline Start and End)

Oznacza to tworzenie (typu **Create Message**) i usuwanie obiektu (symbol **X**)



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.

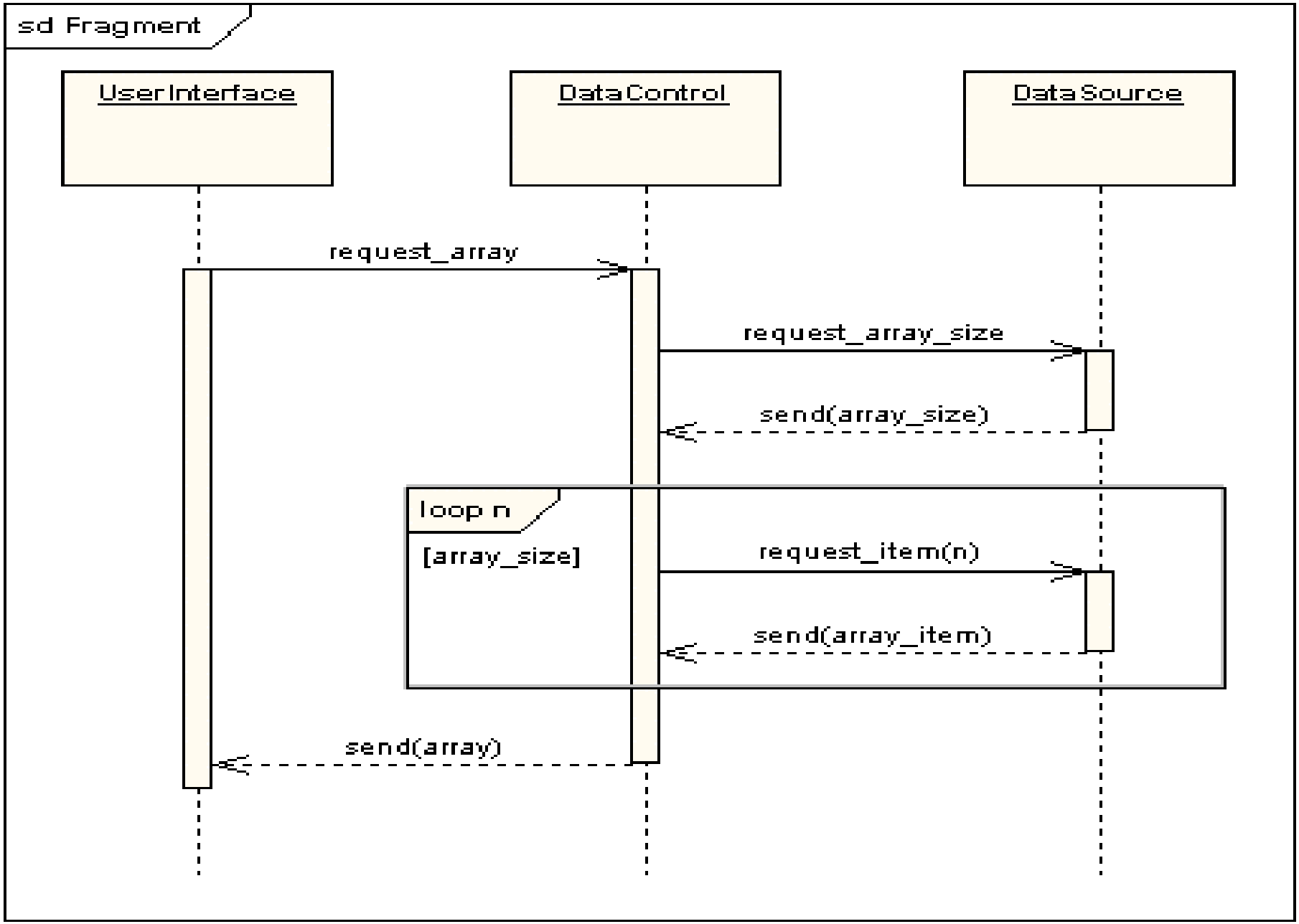


Złożone modelowanie sekwencji wiadomości

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
10. **fragment rozważany-** tylko ważne są wiadomości w tym fragmencie
11. **fragment asercji** (oznaczony "**assert**") eliminuje wszystkie sekwencje wiadomości, które są objęte danym operatorem, jeśli jego wynik jest fałszywy
12. **pętla** (oznaczony "**loop**") oznacza powtarzanie interakcji we fragmencie.

Pętla Wykonanie w pętli fragmentu diagramu sekwencji

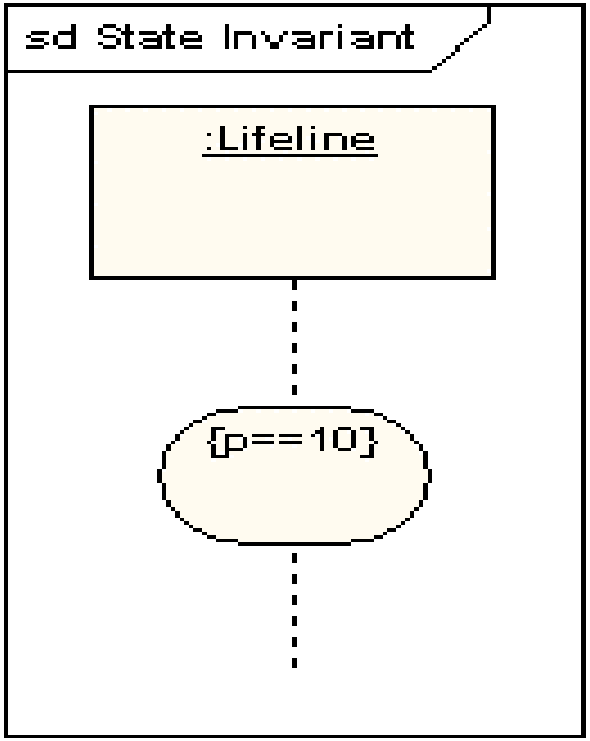


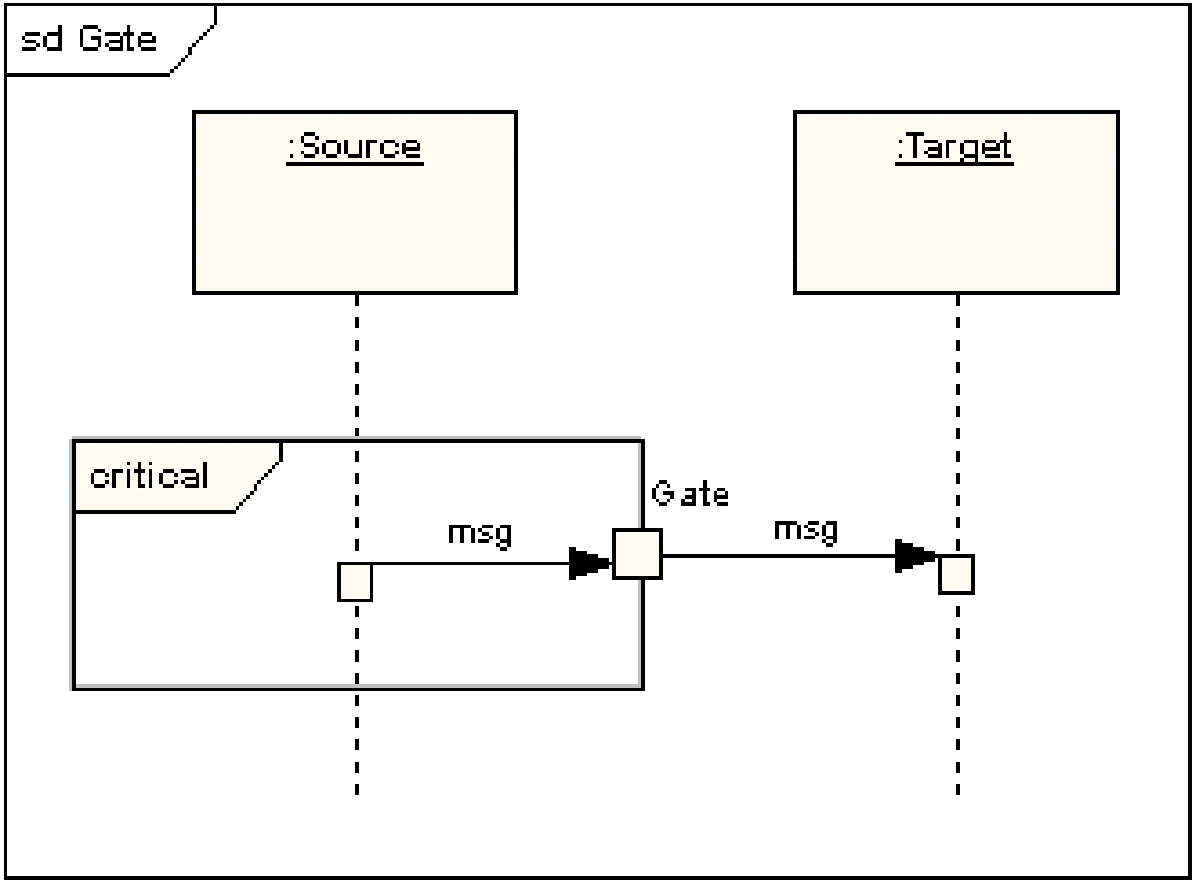
Stan niezmienny lub ciągły (State Invariant /Continuations)

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

Brama (Gate)

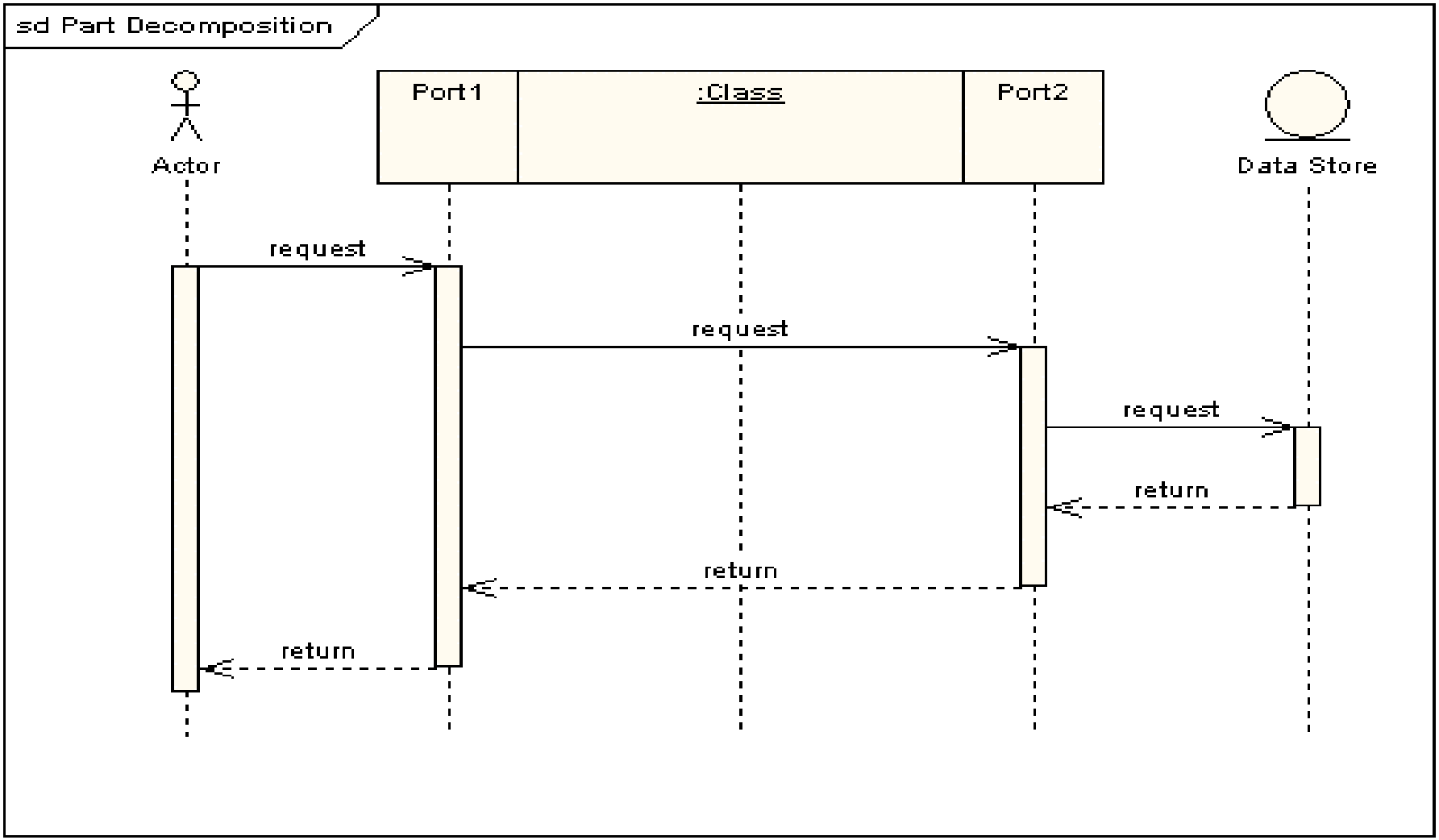
Oznacza przekazywanie wiadomości na zewnątrz między fragmentem i pozostałą częścią diagramu (linie życia, inne fragmenty)





Dekompozycja (Part Decomposition)

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



Diagramy klas, diagramy sekwencji – tworzenie modeli analizy i projektu

1. **Syntaktyka diagramów klas**

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

2. **Identyfikacja elementów diagramów klas**

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

3. **Diagramy sekwencji UML**

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

4. **Przykłady diagramów sekwencji – kontynuacja przykładu 3 z wykładów: 2**

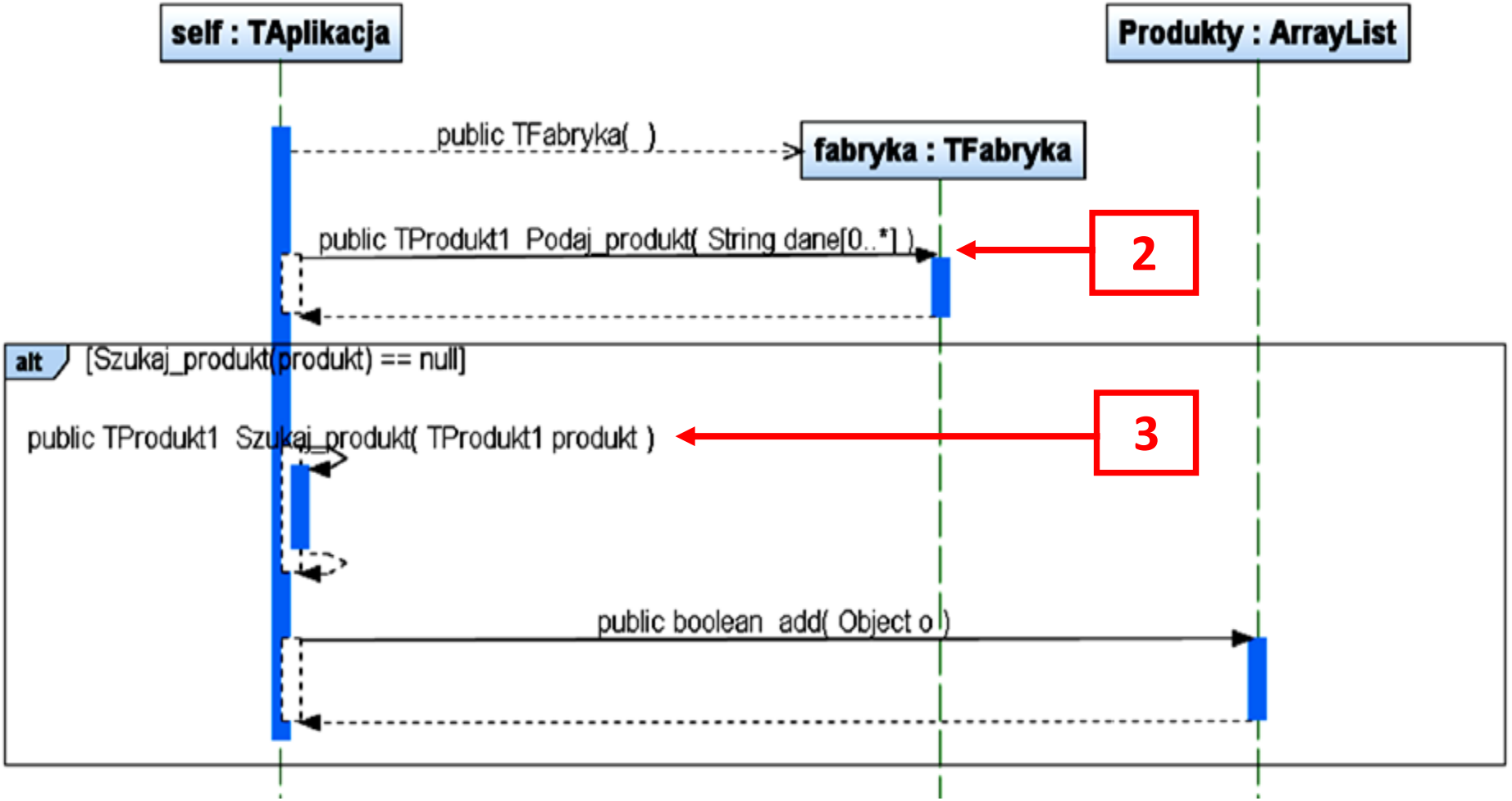
Iteracja 1

Projekt przypadku użycia

„ **Wstawianie nowego produktu**”

za pomocą diagramu sekwencji i diagramu klas. Diagram klas jest uzupełniany metodami zidentyfikowanymi podczas projektowania scenariusza przypadku użycia za pomocą diagramu sekwencji.

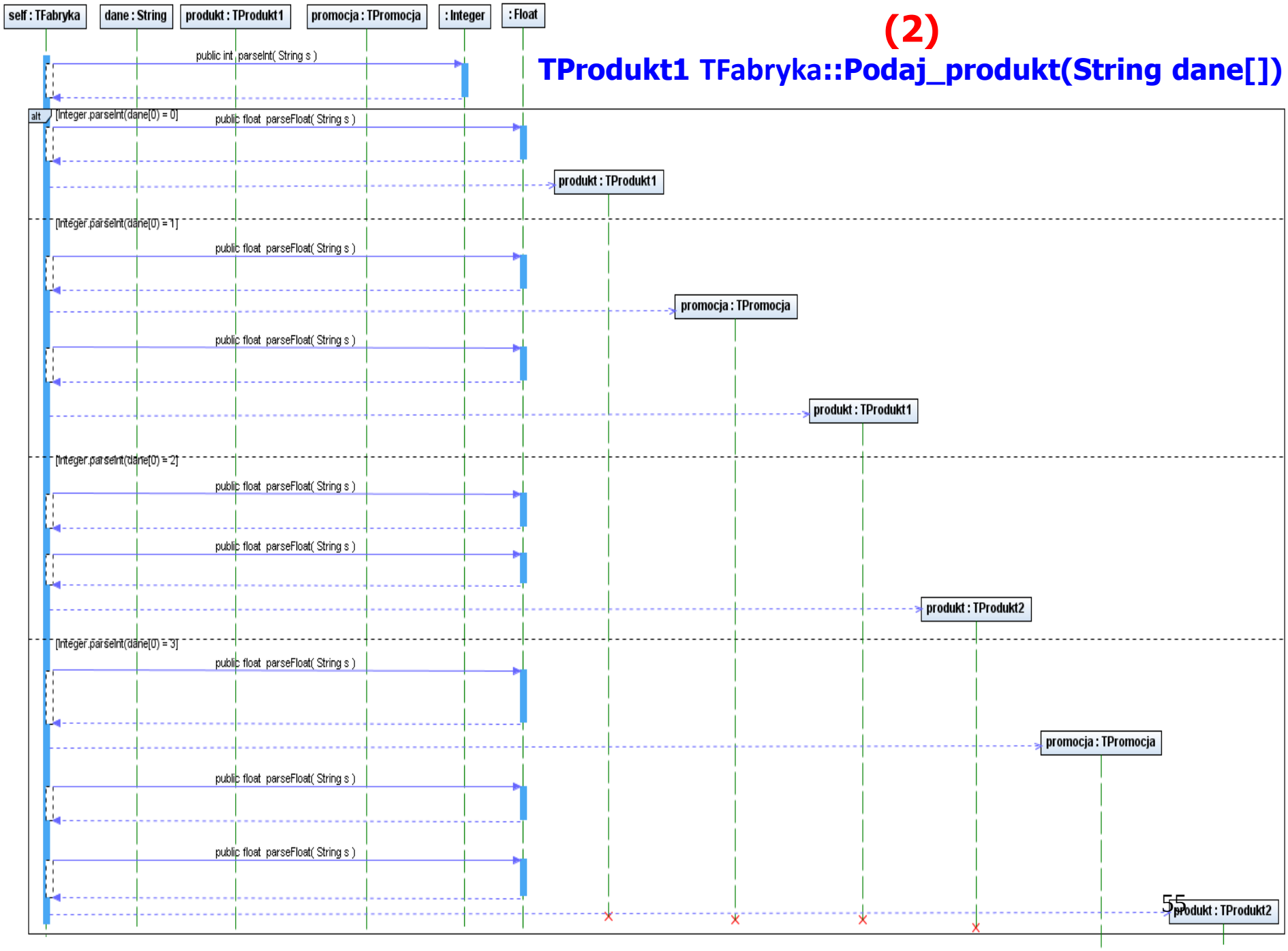
(1) Wstawianie nowego produktu
void TAplikacja::Dodaj_produkt(String [] dane)



```
//class TAplikacja

private List <TProdukt1> Produkty =
                                    new ArrayList <TProdukt1>();

public void Dodaj_produkt (String dane[])
  {
    TFabryka fabryka = new TFabryka();
    TProdukt1 produkt = fabryka.Podaj_produkt(dane);
     if (Szukaj_produkt(produkt) == null)
       Produkty.add(produkt);
  }
```

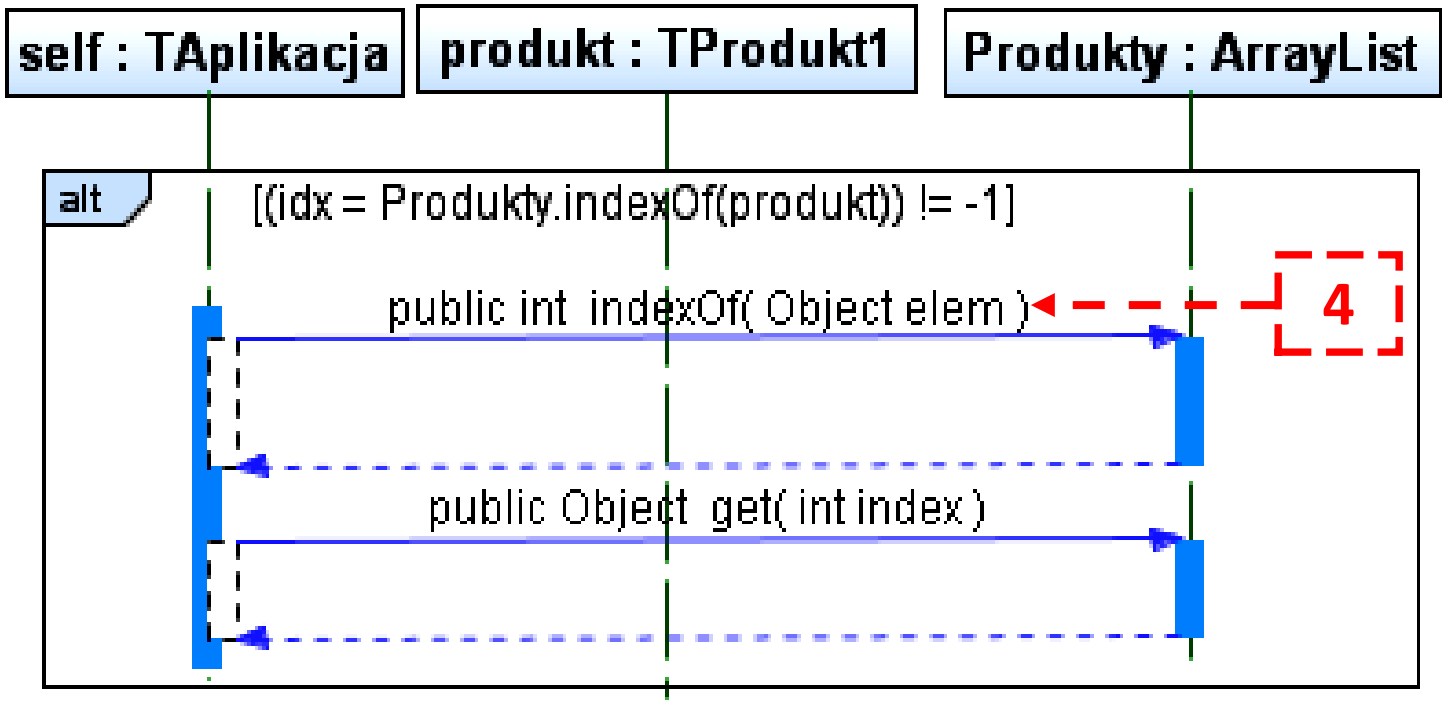
(2)

TProdukt1 TFabryka::Podaj_produkt(String dane[])



```
public class TFabryka                    //TFabryka -decyzje na poziomie tworzenia kodu
 { public TFabryka() { }
   public TProdukt1 Podaj_produkt(String dane[])
   { TProdukt1 produkt = null; TPromocja promocja;
     switch ( Integer.parseInt(dane[0]) )
     {case 0: produkt= new TProdukt1(dane[1], Float.parseFloat(dane[2]));
              break;
      case 1: promocja = new TPromocja(Float.parseFloat(dane[3]));
               produkt= new TProdukt1(dane[1],
                                      Float.parseFloat(dane[2]),promocja);
              break;
      case 2: produkt = new TProdukt2(dane[1], Float.parseFloat(dane[2]),
                                       Float.parseFloat(dane[3]));
              break;
      case 3: promocja = new TPromocja(Float.parseFloat(dane[4]));
              produkt= new TProdukt2(dane[1], Float.parseFloat(dane[2]),
                                      Float.parseFloat(dane[3]),promocja);
              break;
      }
     return produkt;
   }
}
```

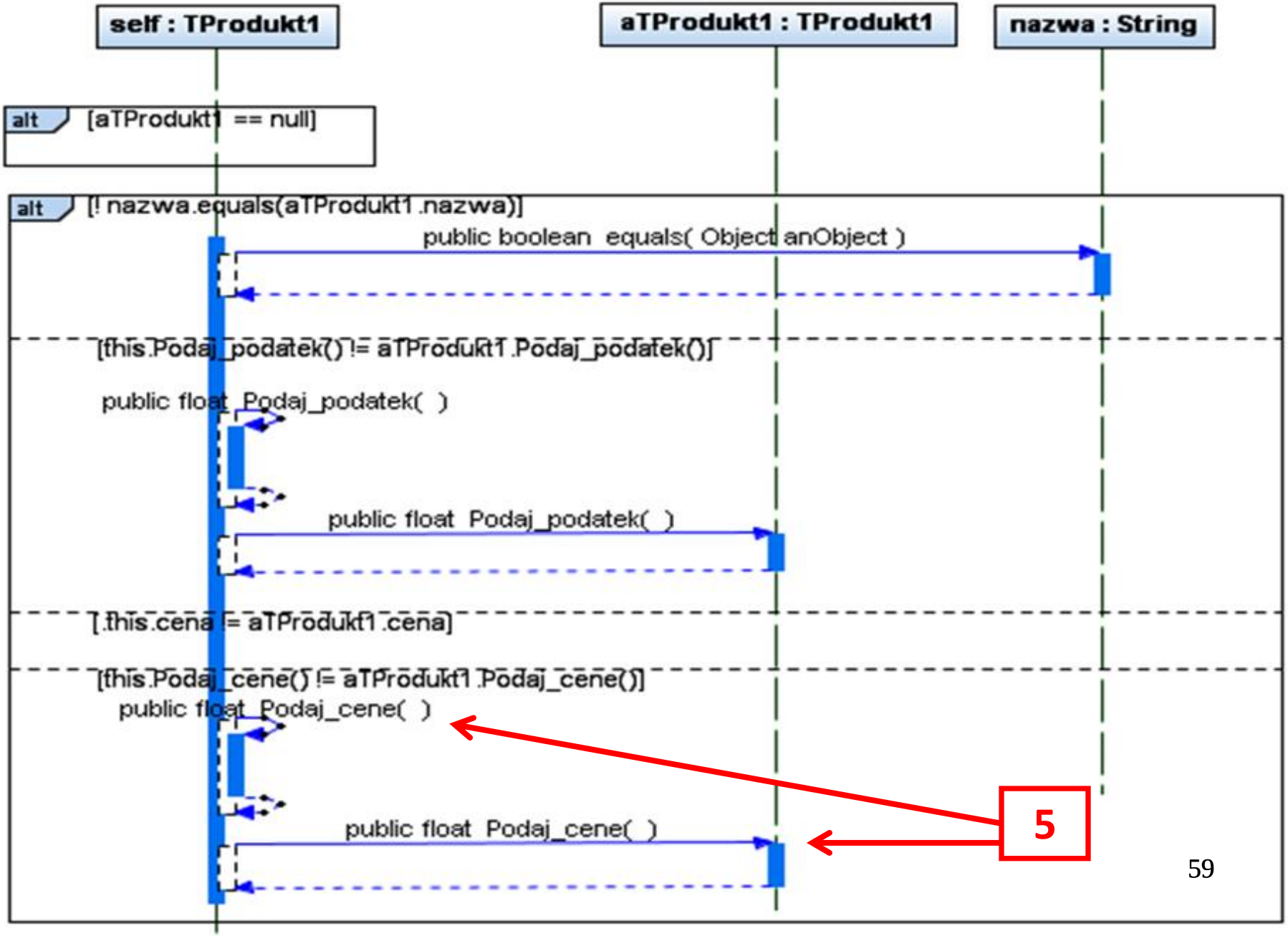
(3) TProdukt1 TAplikacja::Szukaj_produkt(TProdukt1 produkt)



```
//TAplikacja
private List <TProdukt1> Produkty =
                                    new ArrayList <TProdukt1>();

TProdukt1 Szukaj_produkt (TProdukt1 produkt)
  {
    int idx;
    if ((idx=Produkty.indexOf(produkt))!=-1 )
     {
       produkt=Produkty.get(idx);
       return produkt;
     }
    return null;
   }
```

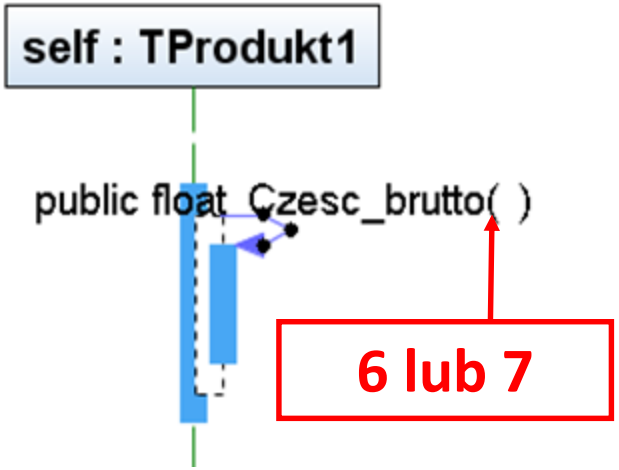
```
public int indexOf(Object o) {
    if (o == null) {
      for (int i = 0; i < size; i++)
        if (elementData[i]==null)
          return i;
    } else {
      for (int i = 0; i < size; i++)
        if (o.equals(elementData[i]))
          return i;  }
    return -1;  }
```

(4) boolean TProdukt1::equals(Object aTProdukt)

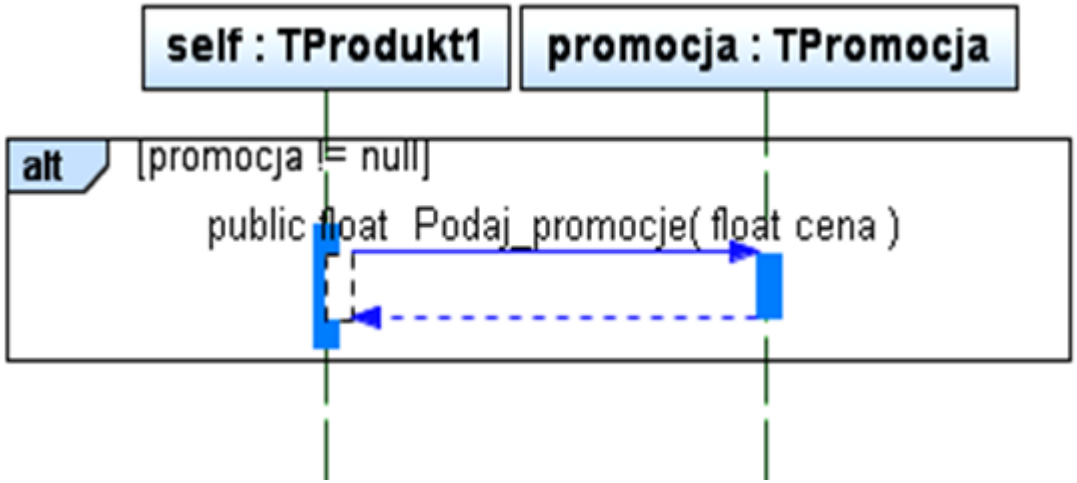


```
//class TProdukt1
public boolean equals (Object aTProdukt)
 {
    TProdukt1 aTProdukt1=(TProdukt1)aTProdukt;
    if ( aTProdukt1 == null ) return false;
    boolean bStatus = true;
    if ( !nazwa.equals(aTProdukt1.nazwa)) bStatus = false;
    else
     if (this.Podaj_podatek()!=aTProdukt1.Podaj_podatek())
         bStatus = false;
     else
       if (this.cena!=aTProdukt1.cena)
          bStatus = false;
       else
          if (this.Podaj_cene()!=aTProdukt1.Podaj_cene())
            bStatus = false;
     return bStatus;
 }
```

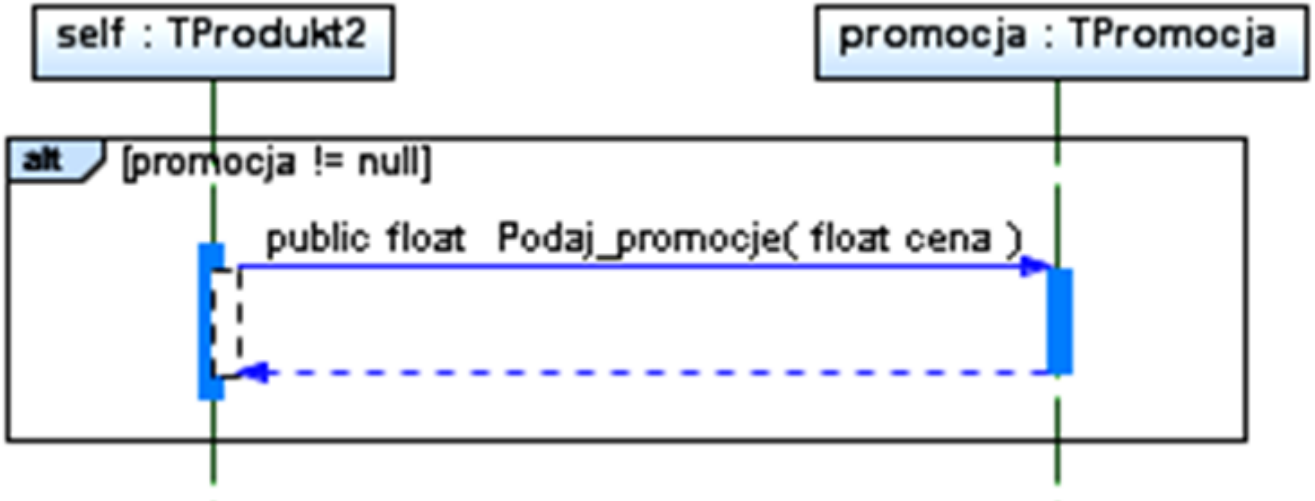
(5)
float TProdukt1::Podaj_cene()



(6)
float TProdukt1::Czesc_brutto()



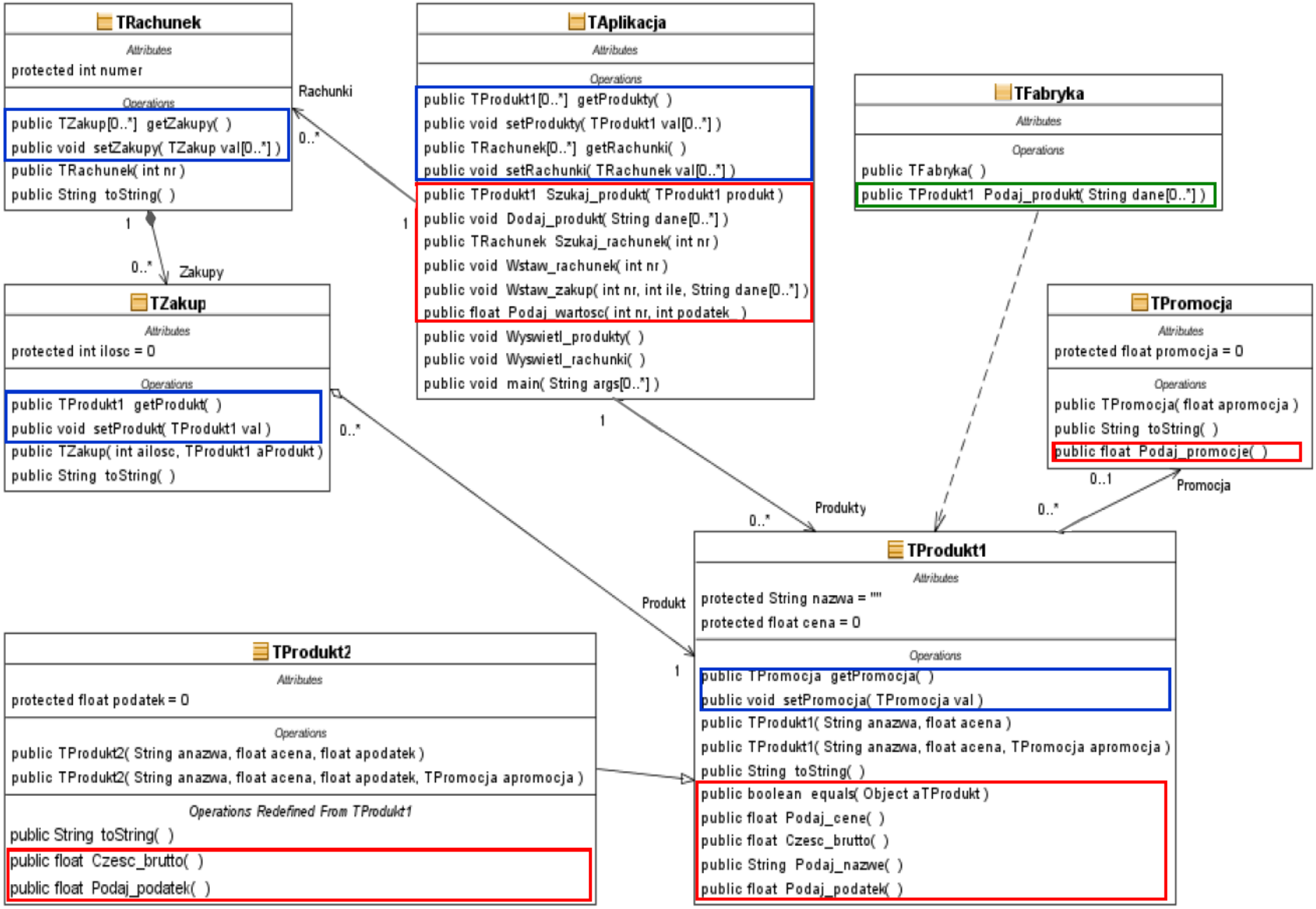
(7)
float TProdukt2::Czesc_brutto()



```
//class TProdukt1
public float Podaj_cene ()
  {
      return cena + Czesc_brutto();
  }

 public float Podaj_podatek ()
   {
     return -1;
   }

 public float Czesc_brutto ()
 {
    if (promocja != null)
        return cena * (-promocja.Podaj_promocje()/100);
    return 0F;
  }
```

```
public  float Czesc_brutto ()            //class TProdukt2
  {
    float dodatek = 0;
    if (promocja  != null)
      dodatek= cena*(-promocja.Podaj_promocje()/100);
     return  cena*podatek/100 + dodatek;
  }
public  float Podaj_podatek ()
  {
     return podatek;
  }
```

```
                                  //class TPromocja lub dowolny jej następca
public  float Podaj_promocje ()
  {  if (promocja<50)     //jakiś algorytm obliczania promocji
        return promocja;
     return promocja *1.1F;
  }
```

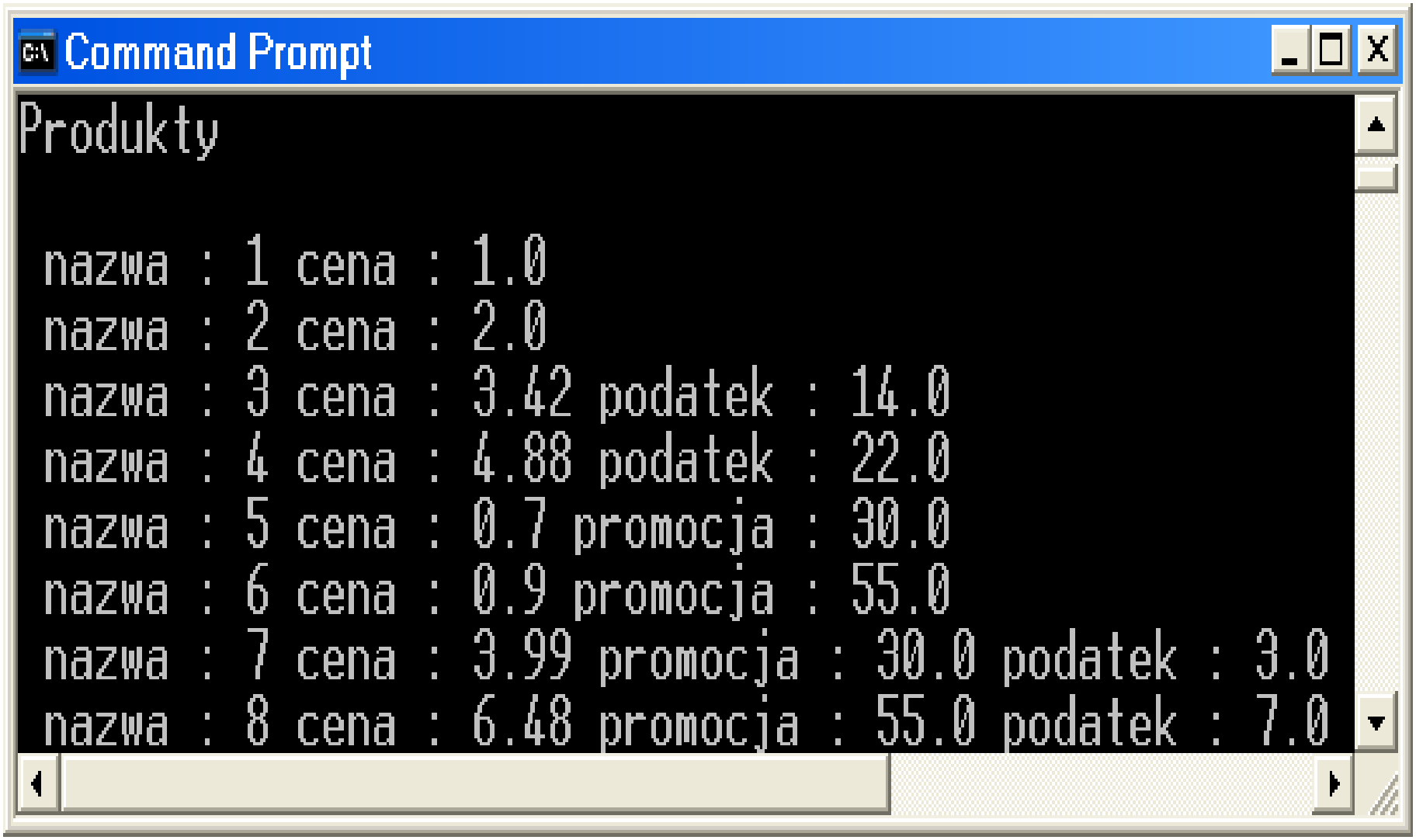

TRachunek
Attributes
protected int numer
Operations
public TZakup[0..*] getZakupy( )
public void setZakupy(TZakup val[0..*])
public TRachunek(int nr)
public String toString( )
Rachunki
0..*
1
1
0..*
Zakupy
TAplikacja
Attributes
Operations
public TProdukt1[0..*] getProdukty( )
public void setProdukty(TProdukt1 val[0..*])
public TRachunek[0..*] getRachunki( )
public void setRachunki(TRachunek val[0..*])
public TProdukt1 Szukaj_produkt(TProdukt1 produkt)
public void Dodaj_produkt(String dane[0..*])
public TRachunek Szukaj_rachunek(int nr)
public void Wstaw_rachunek(int nr)
public void Wstaw_zakup(int nr, int ile, String dane[0..*])
public float Podaj_wartosc(int nr, int podatek)
public void Wyswietl_produkty( )
public void Wyswietl_rachunki( )
public void main(String args[0..*])
TFabryka
Attributes
Operations
public TFabryka( )
public TProdukt1 Podaj_produkt(String dane[0..*])
TZakup
Attributes
protected int ilosc = 0
Operations
public TProdukt1 getProdukt( )
public void setProdukt(TProdukt1 val)
public TZakup(int ailosc, TProdukt1 aProdukt)
public String toString( )
0..*
1
0..*
Produkty
TPromocja
Attributes
protected float promocja = 0
Operations
public TPromocja(float apromocja)
public String toString( )
public float Podaj_promocje( )
0..1
Promocja
0..*
TProdukt1
Attributes
protected String nazwa = ""
protected float cena = 0
Operations
public TPromocja getPromocja( )
public void setPromocja(TPromocja val)
public TProdukt1(String anazwa, float acena)
public TProdukt1(String anazwa, float acena, TPromocja apromocja)
public String toString( )
public boolean equals(Object aTProdukt)
public float Podaj_cene( )
public float Czesc_brutto( )
public String Podaj_nazwe( )
public float Podaj_podatek( )
Produkt
1
TProdukt2
Attributes
protected float podatek = 0
Operations
public TProdukt2(String anazwa, float acena, float apodatek)
public TProdukt2(String anazwa, float acena, float apodatek, TPromocja apromocja)
Operations Redefined From TProdukt1
public String toString( )
public float Czesc_brutto( )
public float Podaj_podatek( )

```
//TAplikacja
public void Wyswietl_produkty()  {
      for (TProdukt1 produkt: Produkty)
           System.out.println(produkt.toString());
 }
```

```
//TProdukt1
public String toString()
 {  StringBuilder sb = new StringBuilder ();
    sb.append(" nazwa : ");
    sb.append(nazwa);
    sb.append(" cena : ");
    sb.append(Podaj_cene());
    if (promocja != null) {
     sb.append(promocja.toString());
   }
   return sb.toString();
 }
```

```
//TProdukt2
public String toString()
  {
   StringBuilder sb =
            new StringBuilder ();
   sb.append(super.toString());
   sb.append (" podatek : " );
   sb.append ( podatek );
   return sb.toString ();
  }
```

```
public static void main(String args[])                                    //TAplikacja
 { TAplikacja app=new TAplikacja();
   String dane1[]={"0","1","1"};              String dane2[]={"0","2","2"};
   app.Dodaj_produkt(dane1);
   app.Dodaj_produkt(dane2);
   app.Dodaj_produkt(dane1);
   String dane3[]={"2","3","3","14"};         String dane4[]={"2","4","4","22"};
   app.Dodaj_produkt(dane3);
   app.Dodaj_produkt(dane4);
   app.Dodaj_produkt(dane3);
   String dane5[]={"1","5","1","30"};         String dane6[]={"1","6","2","50"};
   String dane7[]={"3","7","5.47","3","30"};
   String dane8[]={"3","8", "13.93","7","50"};
   app.Dodaj_produkt(dane5);
   app.Dodaj_produkt(dane6);
   app.Dodaj_produkt(dane5);
   app.Dodaj_produkt(dane7);
   app.Dodaj_produkt(dane8);
   app.Dodaj_produkt(dane7);
   System.out.println("\nProdukty\n");
   app.Wyswietl_produkty();}
```


Command Prompt
Produkty
nazwa : 1 cena : 1.0
nazwa : 2 cena : 2.0
nazwa : 3 cena : 3.42 podatek : 14.0
nazwa : 4 cena : 4.88 podatek : 22.0
nazwa : 5 cena : 0.7 promocja : 30.0
nazwa : 6 cena : 0.9 promocja : 55.0
nazwa : 7 cena : 3.99 promocja : 30.0 podatek : 3.0
nazwa : 8 cena : 6.48 promocja : 55.0 podatek : 7.0

Dodatek – kolejne iteracje rozwoju tworzenia oprogramowania
(kontynuacja przykładu 3 z wykładu 2)

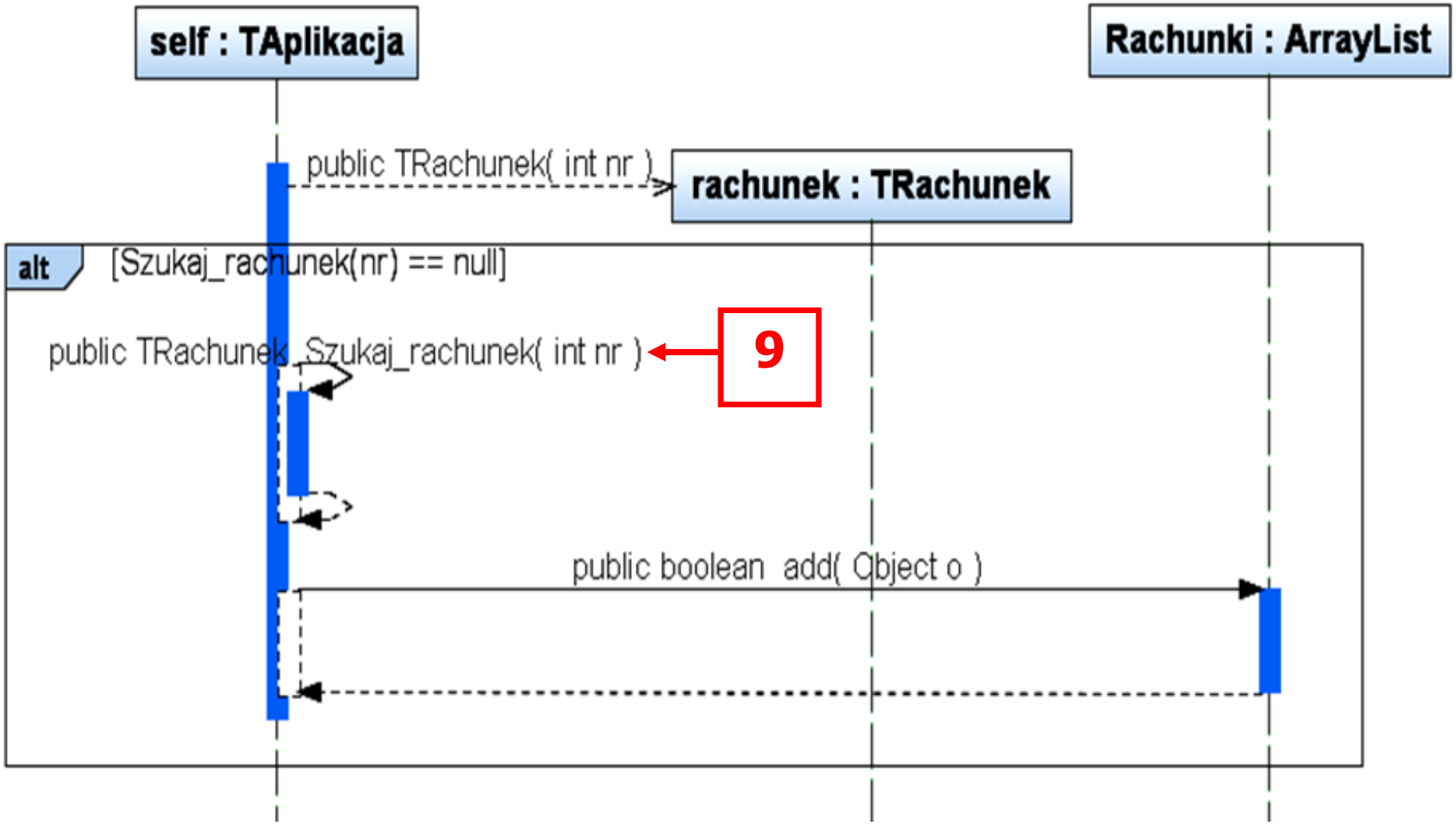
Iteracja 2

Projekt przypadku użycia

„ **Wstawianie nowego rachunku**”

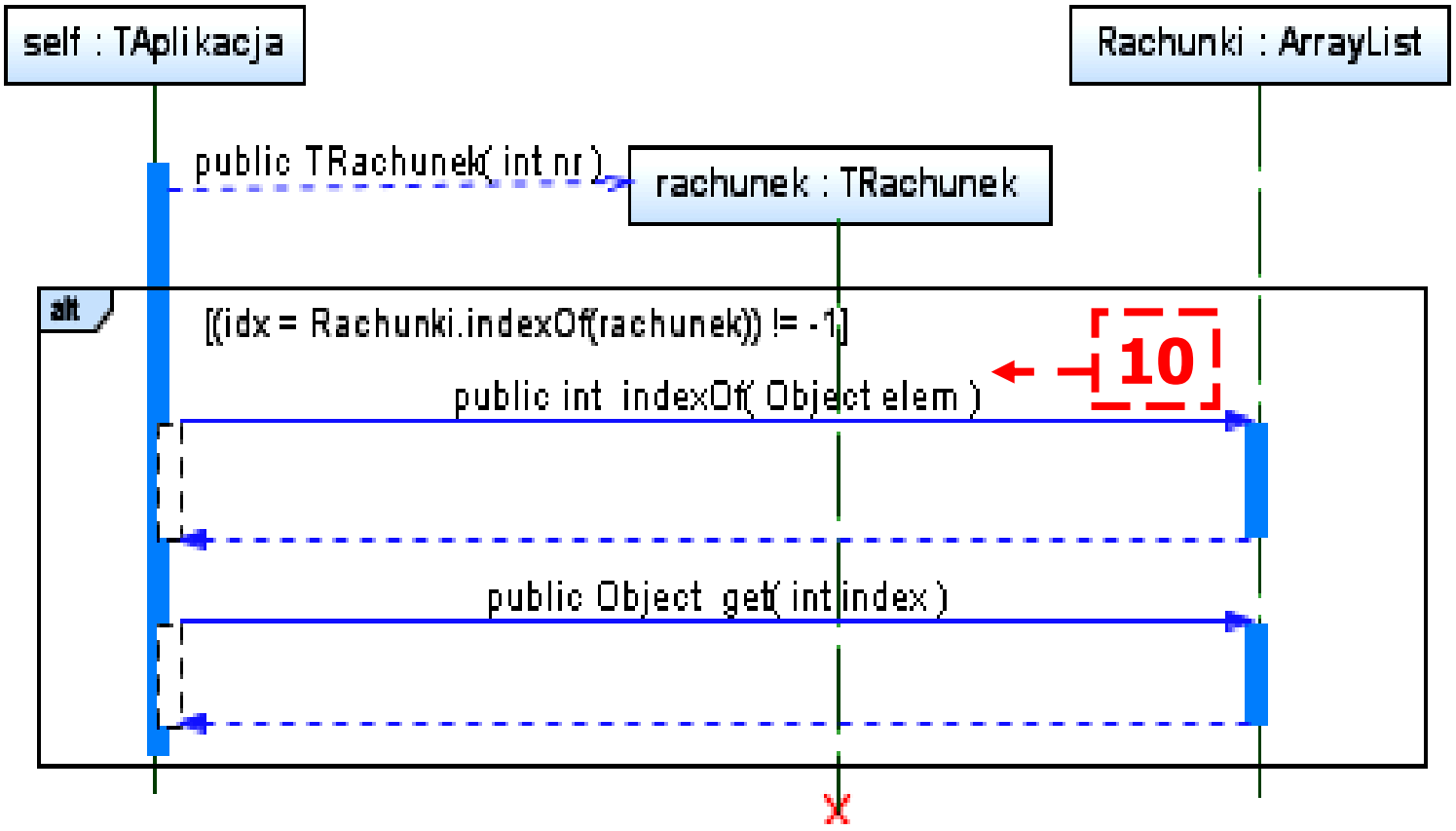
za pomocą diagramu sekwencji i diagramu klas. Diagram klas jest uzupełniany metodami zidentyfikowanymi podczas projektowania scenariusza przypadku użycia za pomocą diagramu sekwencji.

(8) Wstawianie nowego rachunku
void TAplikacja::Wstaw_rachunek(int nr)



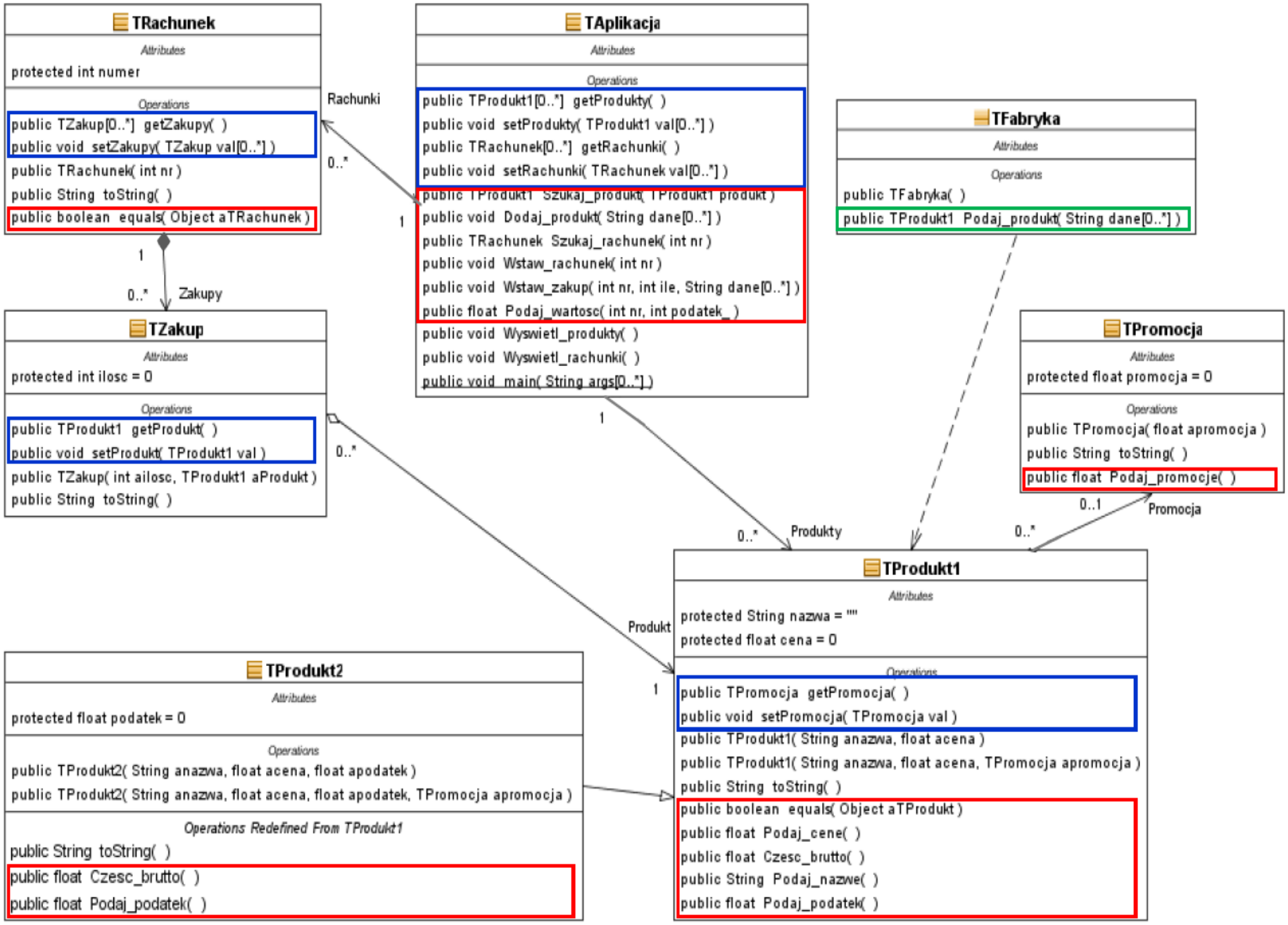
```
                                                    //TAplikacja

private List <TRachunek> Rachunki  =
                              new ArrayList <TRachunek>();

 public void Wstaw_rachunek (int nr)
  {
     TRachunek rachunek=new TRachunek(nr);
      if (Szukaj_rachunek(nr) == null)
           Rachunki.add(rachunek);
  }
```

(9) Szukanie rachunku
TRachunek TAplikacja::Szukaj_rachunek(int nr)



```
//TAplikacja
private List <TRachunek> Rachunki  =
                        new ArrayList <TRachunek>();

 public  TRachunek Szukaj_rachunek (int nr)
  {
    TRachunek rachunek = new TRachunek(nr);
    int idx;
     if ((idx=Rachunki.indexOf(rachunek)) != -1)
      {
        rachunek=Rachunki.get(idx);
        return rachunek;
      }
     return null;
  }
```

```
                                                    //TRachunek

public boolean equals (Object aTRachunek)
  {
    TRachunek rachunek= (TRachunek)aTRachunek;
    boolean bStatus = true;
     if ( numer!= rachunek.numer )
        bStatus = false;
    return bStatus;
   }
```

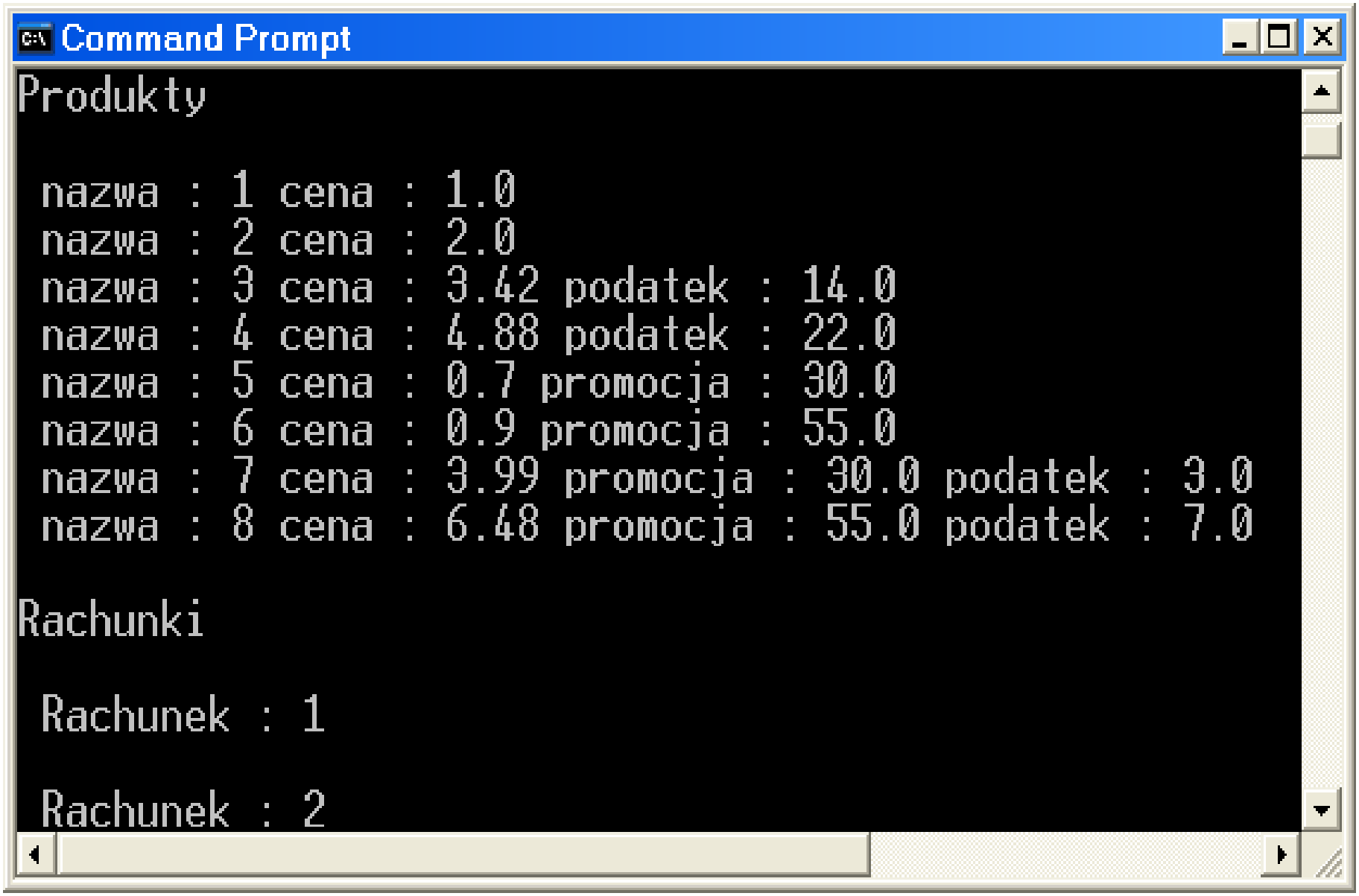

TRachunek
Attributes
protected int numer
Operations
public TZakup[0..*] getZakupy( )
public void setZakupy(TZakup val[0..*])
public TRachunek(int nr)
public String toString( )
public boolean equals(Object aTRachunek)
TAplikacja
Attributes
Operations
public TProdukt1[0..*] getProdukty( )
public void setProdukty(TProdukt1 val[0..*])
public TRachunek[0..*] getRachunki( )
public void setRachunki(TRachunek val[0..*])
public TProdukt1 Szukaj_produkt(TProdukt1 produkt)
public void Dodaj_produkt(String dane[0..*])
public TRachunek Szukaj_rachunek(int nr)
public void Wstaw_rachunek(int nr)
public void Wstaw_zakup(int nr, int ile, String dane[0..*])
public float Podaj_wartosc(int nr, int podatek_)
public void Wyswietl_produkty( )
public void Wyswietl_rachunki( )
public void main(String args[0..*])
TFabryka
Attributes
Operations
public TFabryka( )
public TProdukt1 Podaj_produkt(String dane[0..*])
Rachunki
0..*
1
1
0..*
Zakupy
TZakup
Attributes
protected int ilosc = 0
Operations
public TProdukt1 getProdukt( )
public void setProdukt(TProdukt1 val)
public TZakup(int ailosc, TProdukt1 aProdukt)
public String toString( )
0..*
1
TPromocja
Attributes
protected float promocja = 0
Operations
public TPromocja(float apromocja)
public String toString( )
public float Podaj_promocje( )
0..1
Promocja
0..*
Produkty
0..*
TProdukt1
Attributes
protected String nazwa = ""
protected float cena = 0
Operations
public TPromocja getPromocja( )
public void setPromocja(TPromocja val)
public TProdukt1(String anazwa, float acena)
public TProdukt1(String anazwa, float acena, TPromocja apromocja)
public String toString( )
public boolean equals(Object aTProdukt)
public float Podaj_cene( )
public float Czesc_brutto( )
public String Podaj_nazwe( )
public float Podaj_podatek( )
Produkt
1
TProdukt2
Attributes
protected float podatek = 0
Operations
public TProdukt2(String anazwa, float acena, float apodatek)
public TProdukt2(String anazwa, float acena, float apodatek, TPromocja apromocja)
Operations Redefined From TProdukt1
public String toString( )
public float Czesc_brutto( )
public float Podaj_podatek( )

```
//Decyzje na poziomie tworzenia kodu
                                                    //TAplikacja
public void Wyswietl_rachunki() {
      for(TRachunek rachunek: Rachunki )
         System.out.println(rachunek.toString());
   }
```

```
                                                    //TRachunek
public String toString() {
 TZakup z;
 StringBuilder sb = new StringBuilder();
 sb.append(" Rachunek : ");
 sb.append(numer).append("\n");
 for (TZakup zakup:Zakupy)
   sb.append(zakup.toString()).append("\n");
 return sb.toString();
}
```

```
//TZakup
public String toString() {
  StringBuilder sb =
       new StringBuilder();
   sb.append(" ilosc : ");
   sb.append(ilosc);
   sb.append(" Produkt : ");
   sb.append(Produkt.toString());
   return sb.toString();
 }
```

```
//c.d. kodu metody main po implementacji przypadków użycia:
// Szukanie rachunku i Wstawianie nowego rachunku

        app.Wstaw_rachunek(1);
        app.Wstaw_rachunek(1);
        app.Wstaw_rachunek(2);
        System.out.println("\nRachunki\n");
        TRachunek  rachunek;
        if ((rachunek = app.Szukaj_rachunek(1)) != null) {
           System.out.println(rachunek.toString());
        }
        if ((rachunek = app.Szukaj_rachunek(2)) != null) {
           System.out.println(rachunek.toString());
        }
    }
}
```


Command Prompt
Produkty
nazwa : 1 cena : 1.0
nazwa : 2 cena : 2.0
nazwa : 3 cena : 3.42 podatek : 14.0
nazwa : 4 cena : 4.88 podatek : 22.0
nazwa : 5 cena : 0.7 promocja : 30.0
nazwa : 6 cena : 0.9 promocja : 55.0
nazwa : 7 cena : 3.99 promocja : 30.0 podatek : 3.0
nazwa : 8 cena : 6.48 promocja : 55.0 podatek : 7.0
Rachunki
Rachunek : 1
Rachunek : 2

Iteracja 3

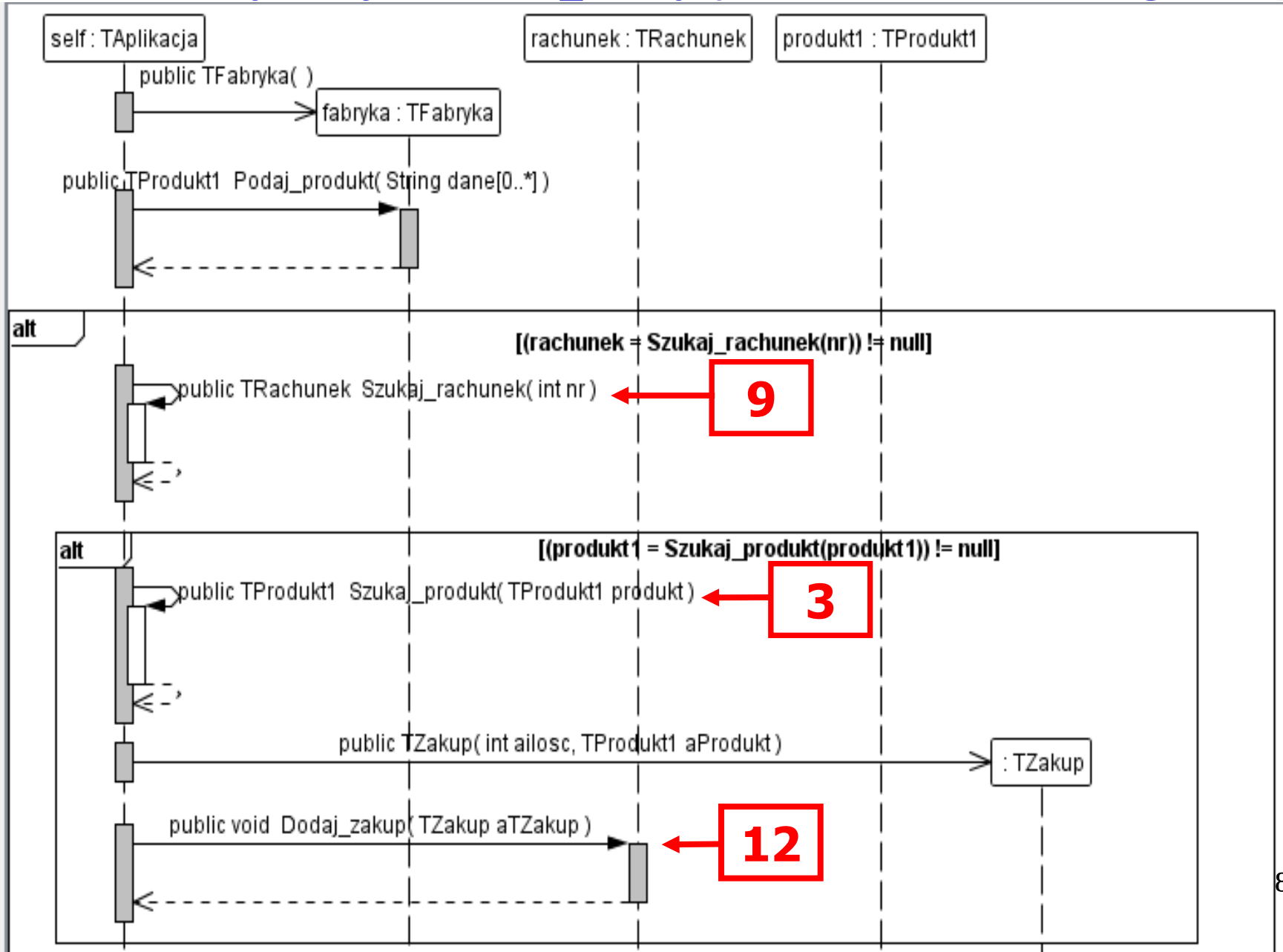
Projekt przypadku użycia

„**Wstawianie nowego zakupu**”

za pomocą diagramu sekwencji i diagramu klas. Diagram klas jest uzupełniany metodami zidentyfikowanymi podczas projektowania scenariusza przypadku użycia za pomocą diagramu sekwencji.

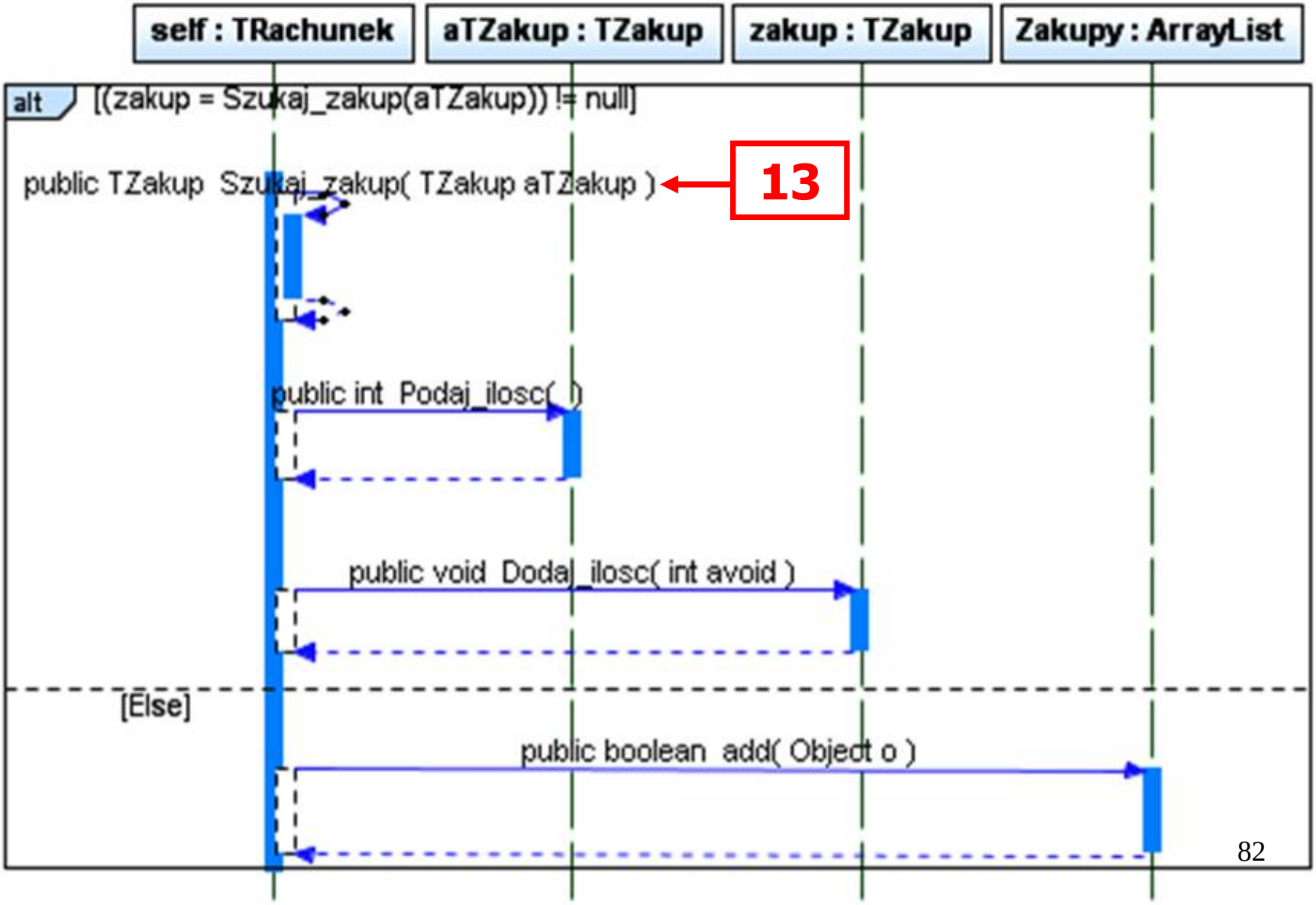
(11) Wstawianie nowego zakupu

void TAplikacja::Wstaw_zakup (int nr, int ailosc, String dane[])



```
                                                    //TAplikacja

public   void Wstaw_zakup (int nr, int ile, String dane[])
{
  TRachunek rachunek;
   TFabryka fabryka = new TFabryka();
   TProdukt1 produkt1 = fabryka.Podaj_produkt(dane);
   if ((rachunek=Szukaj_rachunek(nr)) != null)
     if ((produkt1=Szukaj_produkt(produkt1)) != null)
       rachunek.Dodaj_zakup(new TZakup(ile, produkt1));
 }
```

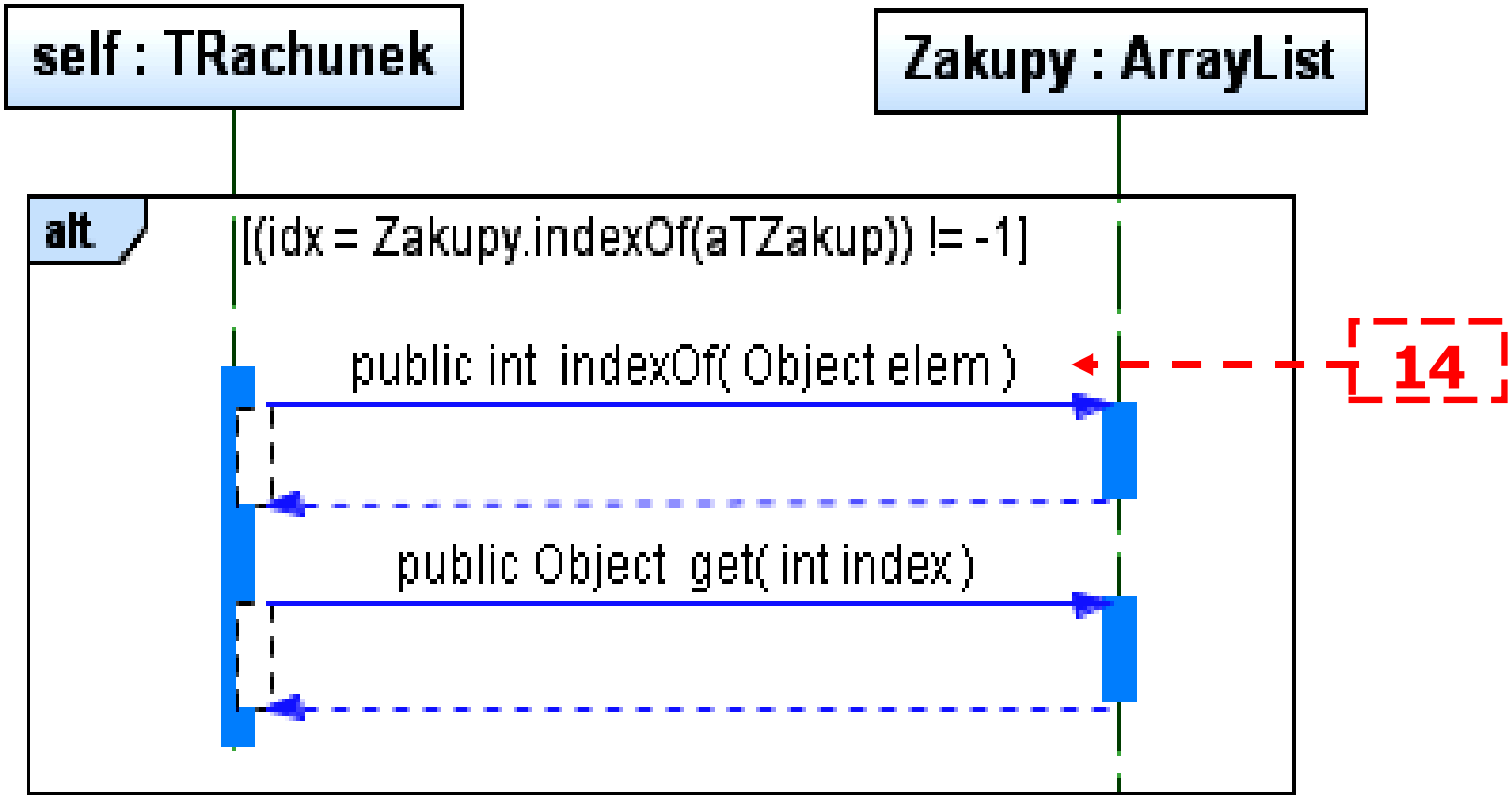
(12) void TRachunek::Dodaj zakup(TZakup aTZakup)



```
//TRachunek

private List<TZakup> Zakupy = new ArrayList<TZakup>();

public void Dodaj_zakup (TZakup aTZakup)
    {
      TZakup zakup;
      if ((zakup = Szukaj_zakup(aTZakup)) != null)
           zakup.Dodaj_ilosc(aTZakup.Podaj_ilosc());
      else
           Zakupy.add(aTZakup);
  }
```

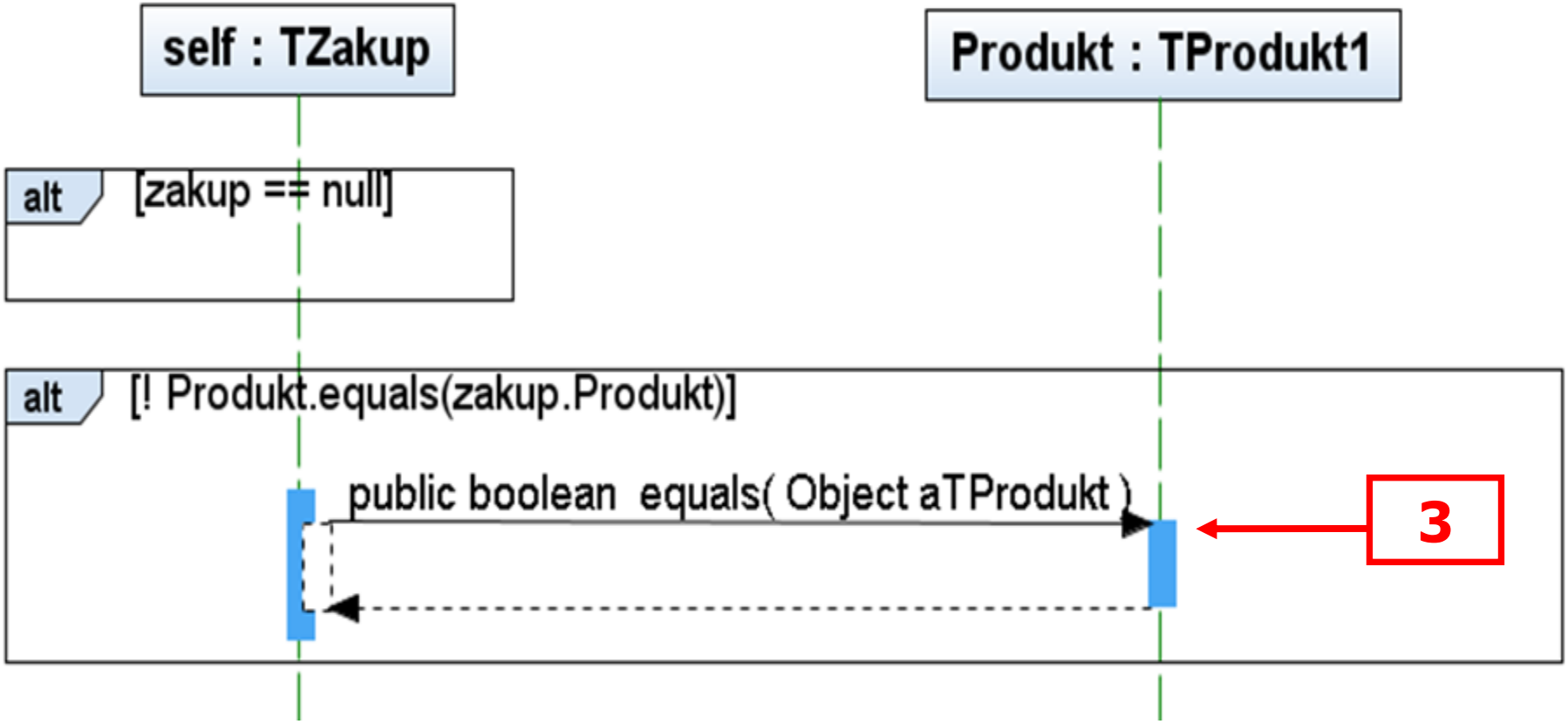
```
                                                    //TZakup


public  void Dodaj_ilosc ( int avoid)
  {
     ilosc+=avoid;
  }

public  int Podaj_ilosc ()
  {
     return ilosc;
 }
```

(13) TZakup TRachunek::Szukaj_zakup(TZakup aTZakup)

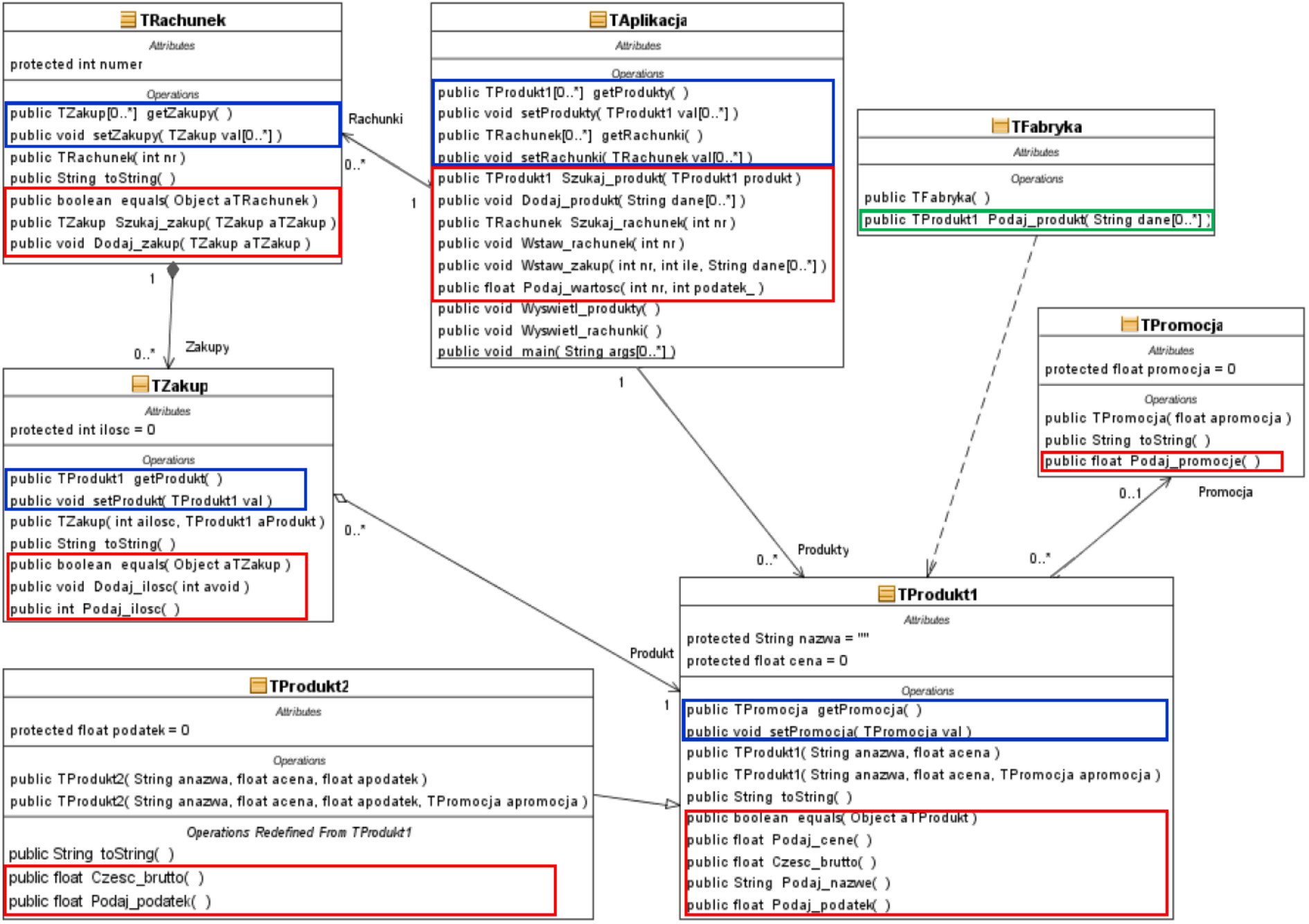


```
//TRachunek

private List<TZakup> Zakupy = new ArrayList<TZakup>();

public TZakup Szukaj_zakup (TZakup aTZakup)
 {
   int idx;
   if ((idx=Zakupy.indexOf(aTZakup))!=-1)
      {
       aTZakup=Zakupy.get(idx);
       return aTZakup;
      }
    return null;
 }
```

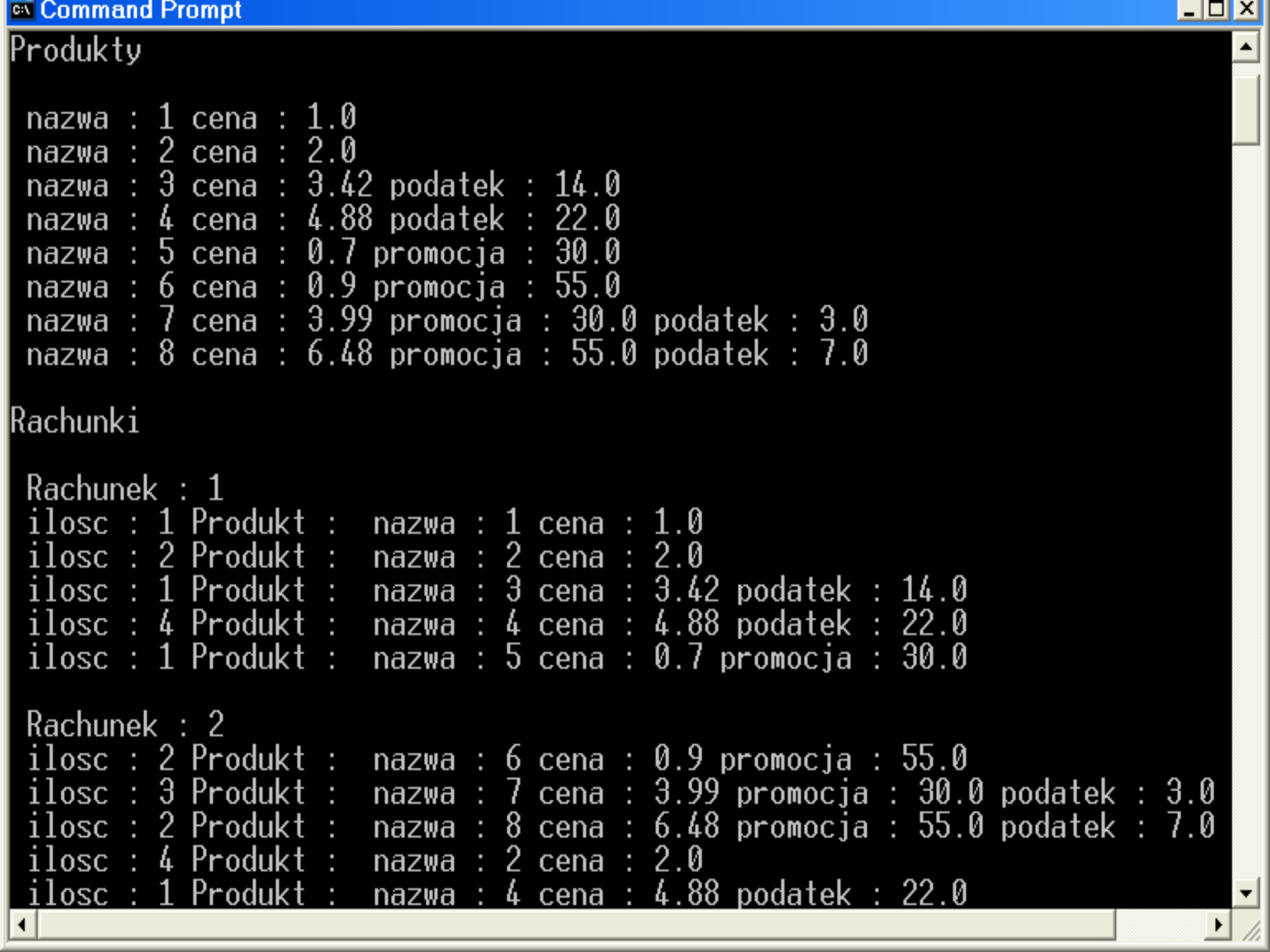
(14) boolean TZakup::equals(Object zakup)



```
                                                    //TZakup

private   TProdukt1 Produkt = null;

 public boolean equals ( Object aTZakup )
  {
   TZakup zakup=(TZakup)aTZakup;
    if ( zakup == null )
          return false;
    boolean bStatus = true;
    if ( !Produkt.equals(zakup.Produkt) )
       bStatus = false;
    return bStatus;
   }
```


TRachunek
Attributes
protected int numer
Operations
public TZakup[0..*] getZakupy( )
public void setZakupy(TZakup val[0..*])
public TRachunek(int nr)
public String toString( )
public boolean equals(Object aTRachunek)
public TZakup Szukaj_zakup(TZakup aTZakup)
public void Dodaj_zakup(TZakup aTZakup)
Rachunki
0..*
1
1
0..*
Zakupy
TAplikacja
Attributes
Operations
public TProdukt1[0..*] getProdukty( )
public void setProdukty(TProdukt1 val[0..*])
public TRachunek[0..*] getRachunki( )
public void setRachunki(TRachunek val[0..*])
public TProdukt1 Szukaj_produkt(TProdukt1 produkt)
public void Dodaj_produkt(String dane[0..*])
public TRachunek Szukaj_rachunek(int nr)
public void Wstaw_rachunek(int nr)
public void Wstaw_zakup(int nr, int ile, String dane[0..*])
public float Podaj_wartosc(int nr, int podatek_)
public void Wyswietl_produkty( )
public void Wyswietl_rachunki( )
public void main(String args[0..*])
1
TFabryka
Attributes
Operations
public TFabryka( )
public TProdukt1 Podaj_produkt(String dane[0..*])
TPromocja
Attributes
protected float promocja = 0
Operations
public TPromocja(float apromocja)
public String toString( )
public float Podaj_promocje( )
0..1
Promocja
TZakup
Attributes
protected int ilosc = 0
Operations
public TProdukt1 getProdukt( )
public void setProdukt(TProdukt1 val)
public TZakup(int ailosc, TProdukt1 aProdukt)
public String toString( )
public boolean equals(Object aTZakup)
public void Dodaj_ilosc(int avoid)
public int Podaj_ilosc( )
0..*
0..*
Produkty
0..*
TProdukt1
Attributes
protected String nazwa = ""
protected float cena = 0
Operations
public TPromocja getPromocja( )
public void setPromocja(TPromocja val)
public TProdukt1(String anazwa, float acena)
public TProdukt1(String anazwa, float acena, TPromocja apromocja)
public String toString( )
public boolean equals(Object aTProdukt)
public float Podaj_cene( )
public float Czesc_brutto( )
public String Podaj_nazwe( )
public float Podaj_podatek( )
Produkt
1
TProdukt2
Attributes
protected float podatek = 0
Operations
public TProdukt2(String anazwa, float acena, float apodatek)
public TProdukt2(String anazwa, float acena, float apodatek, TPromocja apromocja)
Operations Redefined From TProdukt1
public String toString( )
public float Czesc_brutto( )
public float Podaj_podatek( )

```
//c.d. kodu metody main po implementacji przypadków użycia:
// Wstawianie nowego zakupu
      app.Wstaw_zakup(1, 1, dane1);
       app.Wstaw_zakup(1, 2, dane2);
       app.Wstaw_zakup(1, 1, dane3);
       app.Wstaw_zakup(1, 4, dane4);
       app.Wstaw_zakup(1, 1, dane5);
       app.Wstaw_zakup(2, 1, dane6);
       app.Wstaw_zakup(2, 3, dane7);
       app.Wstaw_zakup(2, 1, dane8);
       app.Wstaw_zakup(2, 4, dane2);
       app.Wstaw_zakup(2, 1, dane4);
       app.Wstaw_zakup(2, 1, dane6);
       app.Wstaw_zakup(2, 1, dane8);
      System.out.println("\nRachunki\n");
      TRachunek rachunek;
      if ((rachunek = app.Szukaj_rachunek(1)) != null) {
         System.out.println(rachunek.toString()); }
      if ((rachunek = app.Szukaj_rachunek(2)) != null) {
         System.out.println(rachunek.toString()); }
   }
}
```


Command Prompt
Produkty
nazwa : 1 cena : 1.0
nazwa : 2 cena : 2.0
nazwa : 3 cena : 3.42 podatek : 14.0
nazwa : 4 cena : 4.88 podatek : 22.0
nazwa : 5 cena : 0.7 promocja : 30.0
nazwa : 6 cena : 0.9 promocja : 55.0
nazwa : 7 cena : 3.99 promocja : 30.0 podatek : 3.0
nazwa : 8 cena : 6.48 promocja : 55.0 podatek : 7.0
Rachunki
Rachunek : 1
ilosc : 1 Produkt : nazwa : 1 cena : 1.0
ilosc : 2 Produkt : nazwa : 2 cena : 2.0
ilosc : 1 Produkt : nazwa : 3 cena : 3.42 podatek : 14.0
ilosc : 4 Produkt : nazwa : 4 cena : 4.88 podatek : 22.0
ilosc : 1 Produkt : nazwa : 5 cena : 0.7 promocja : 30.0
Rachunek : 2
ilosc : 2 Produkt : nazwa : 6 cena : 0.9 promocja : 55.0
ilosc : 3 Produkt : nazwa : 7 cena : 3.99 promocja : 30.0 podatek : 3.0
ilosc : 2 Produkt : nazwa : 8 cena : 6.48 promocja : 55.0 podatek : 7.0
ilosc : 4 Produkt : nazwa : 2 cena : 2.0
ilosc : 1 Produkt : nazwa : 4 cena : 4.88 podatek : 22.0

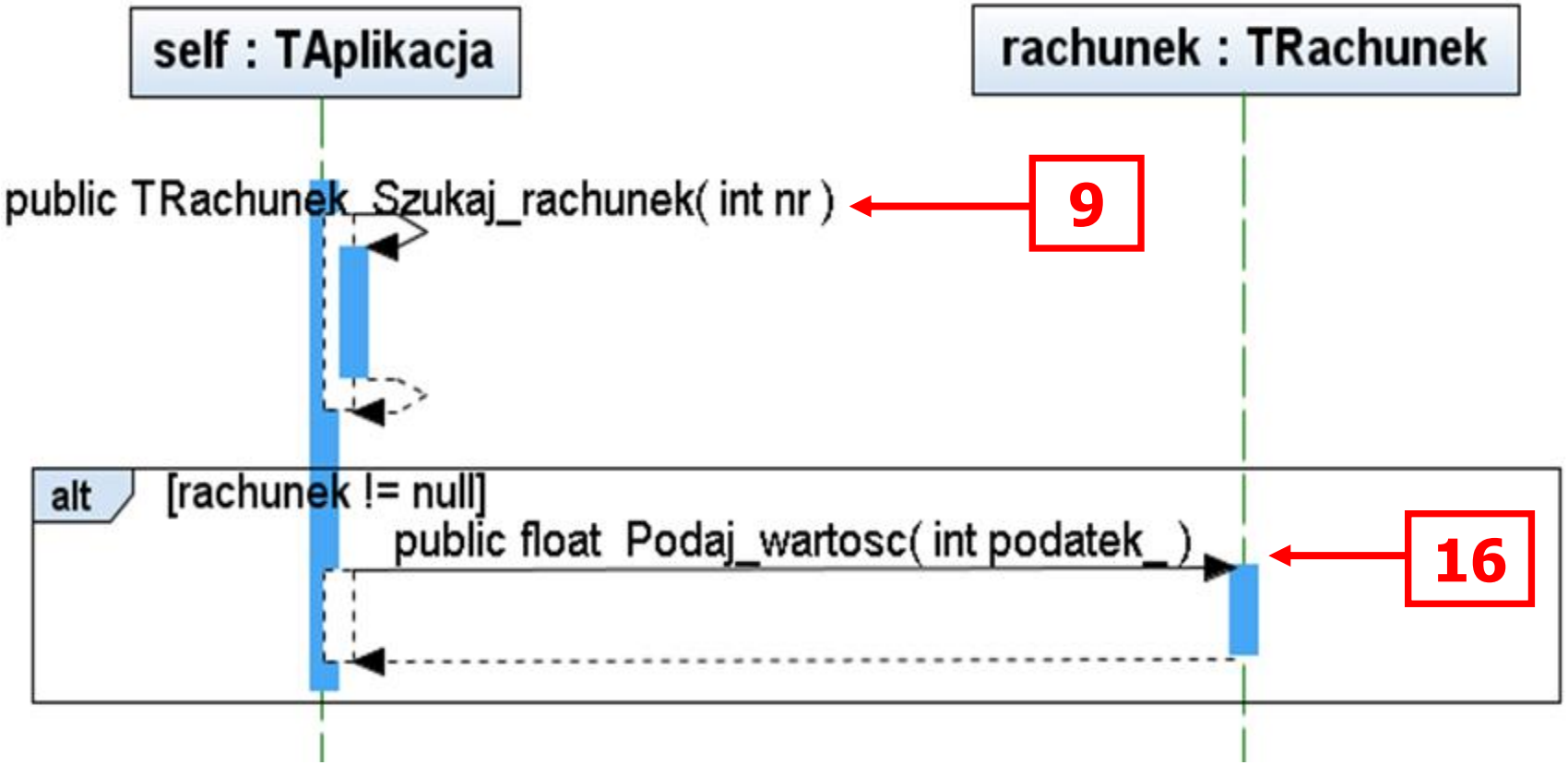
Iteracja 4

Projekt przypadku użycia

„**Obliczanie wartości rachunku**”

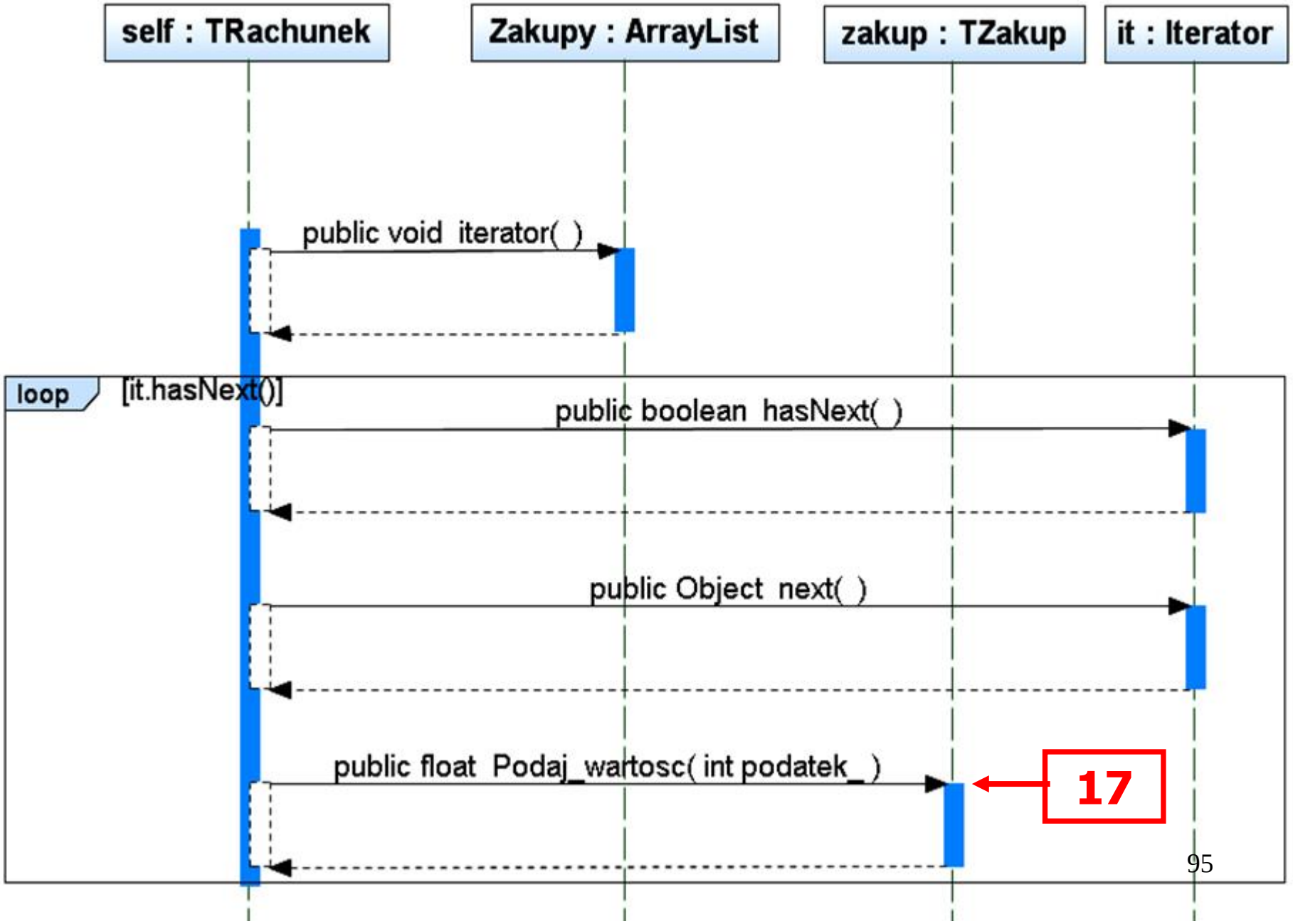
za pomocą diagramu sekwencji i diagramu klas. Diagram klas jest uzupełniany metodami zidentyfikowanymi podczas projektowania scenariusza przypadku użycia za pomocą diagramu sekwencji.

(15) Obliczanie wartosci rachunku
float TAplikacja::Podaj_wartosc(int nr, int podatek_)



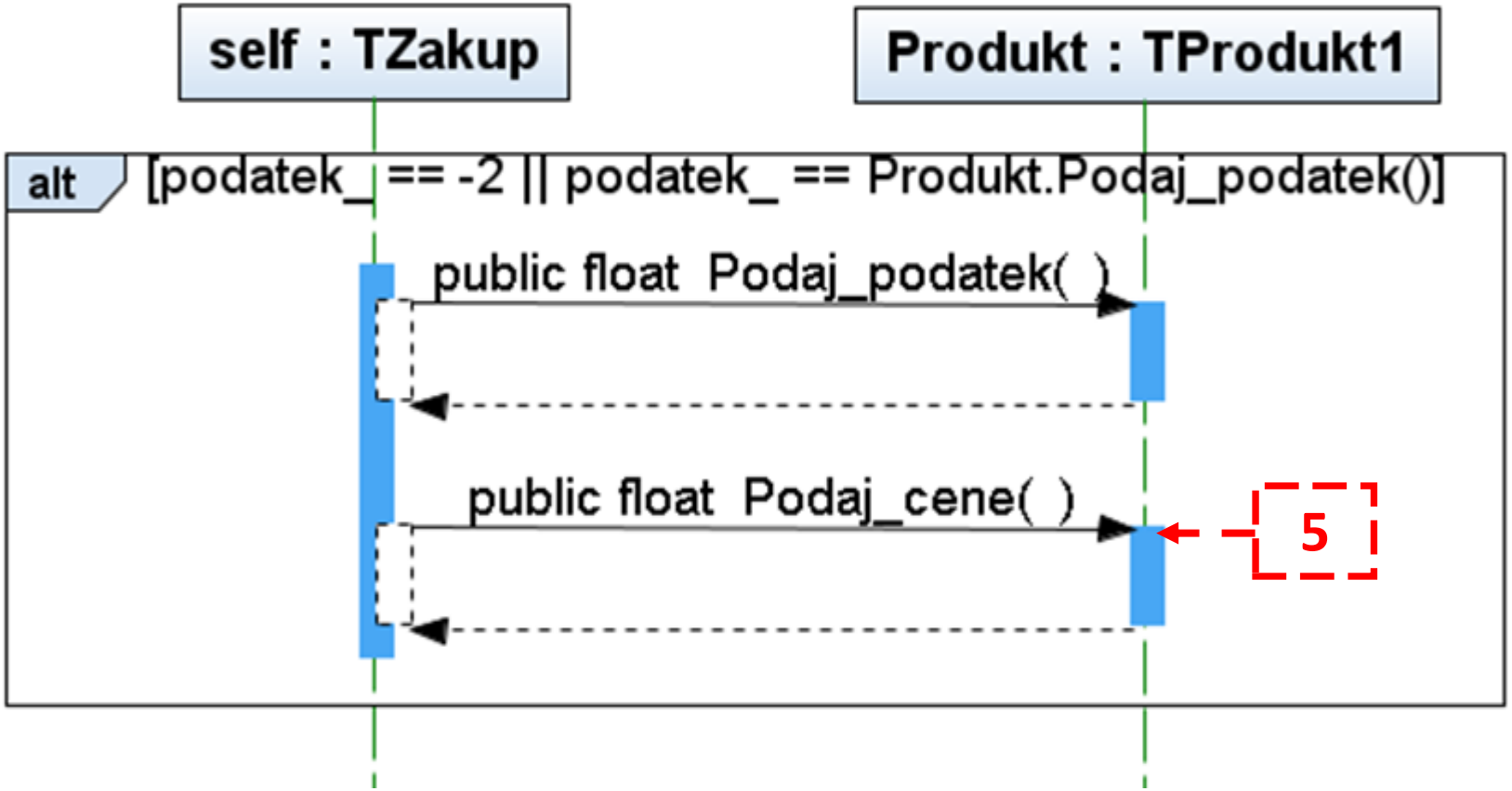
```
                                                    //TAplikacja

public  float Podaj_wartosc (int nr, int podatek_)
 {
  TRachunek rachunek;
   rachunek = Szukaj_rachunek(nr);
   if (rachunek != null)
       return rachunek.Podaj_wartosc(podatek_);
   return 0F;
 }
```

(16) float TRachunek::Podaj_wartosc(int podatek_)

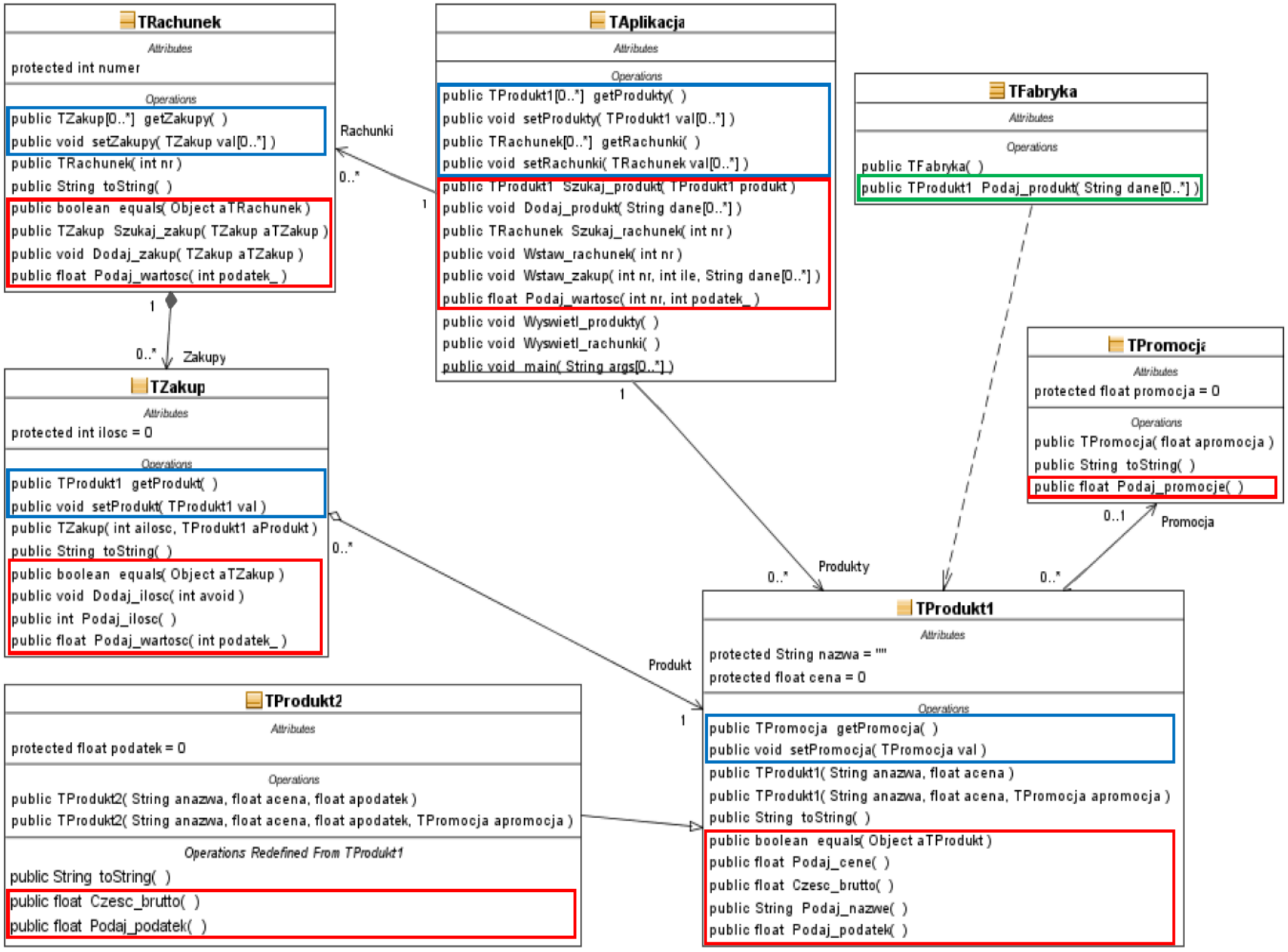


```
//TRachunek

private List<TZakup> Zakupy = new ArrayList<TZakup>();

public float Podaj_wartosc (int podatek_)
 {
   float suma=0;
   TZakup zakup;
   Iterator <TZakup> it=Zakupy.iterator();
   while (it.hasNext())
    {  zakup = it.next();
       suma += zakup.Podaj_wartosc(podatek_);
    }
   return suma;
  }
```

(17) float TZakup::Podaj_wartosc(int podatek_)



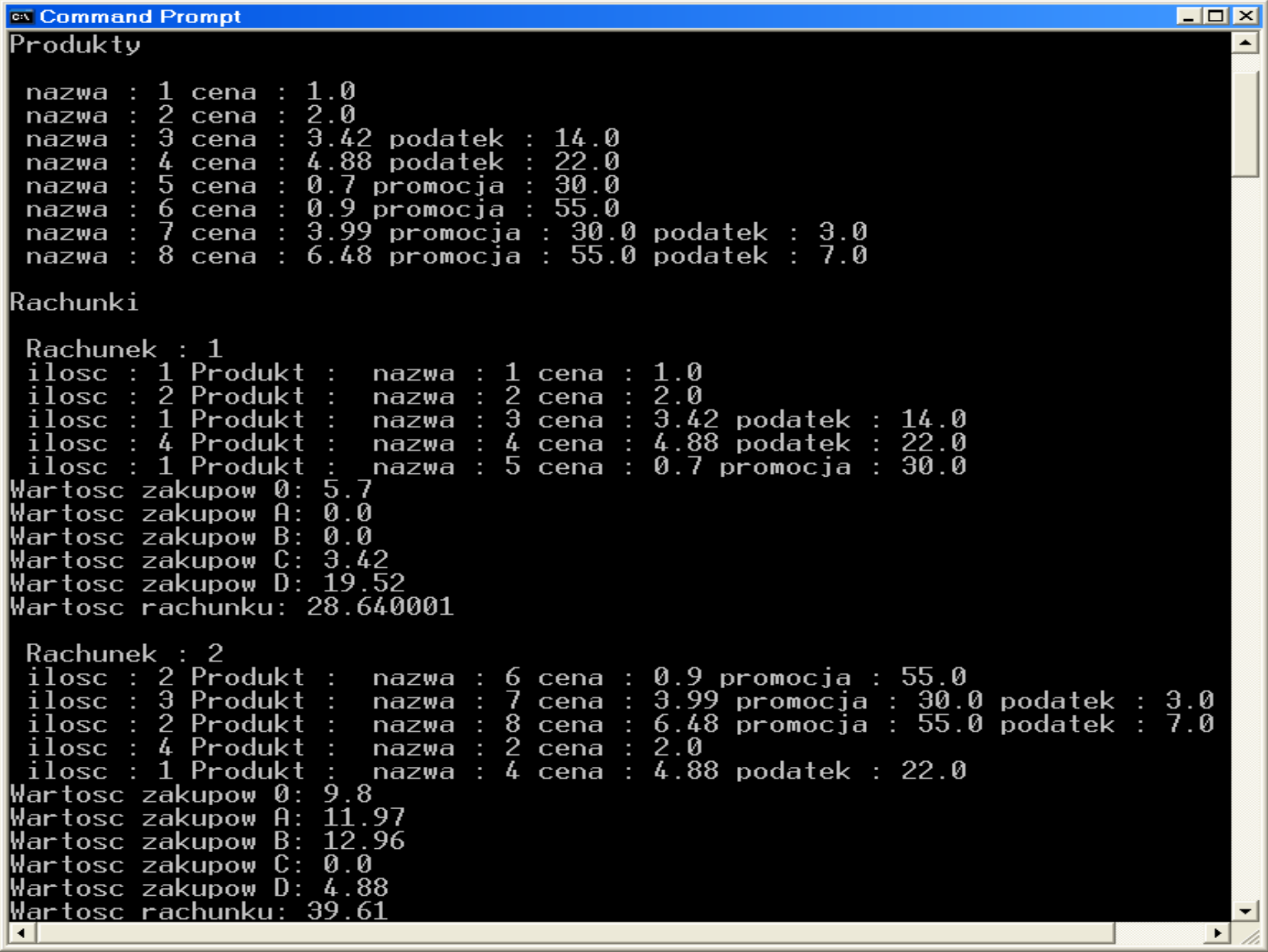
```
                                                    //TZakup


private   TProdukt1 Produkt = null;

public  float Podaj_wartosc (int podatek_)
  {
   if (podatek_==-2 ||podatek_==Produkt.Podaj_podatek())
      return ilosc*Produkt.Podaj_cene();
   return 0F;
  }
```


TRachunek
Attributes
protected int numer
Operations
public TZakup[0..*] getZakupy( )
public void setZakupy(TZakup val[0..*])
public TRachunek(int nr)
public String toString( )
public boolean equals(Object aTRachunek)
public TZakup Szukaj_zakup(TZakup aTZakup)
public void Dodaj_zakup(TZakup aTZakup)
public float Podaj_wartosc(int podatek_)
TAplikacja
Attributes
Operations
public TProdukt1[0..*] getProdukty( )
public void setProdukty(TProdukt1 val[0..*])
public TRachunek[0..*] getRachunki( )
public void setRachunki(TRachunek val[0..*])
public TProdukt1 Szukaj_produkt(TProdukt1 produkt)
public void Dodaj_produkt(String dane[0..*])
public TRachunek Szukaj_rachunek(int nr)
public void Wstaw_rachunek(int nr)
public void Wstaw_zakup(int nr, int ile, String dane[0..*])
public float Podaj_wartosc(int nr, int podatek_)
public void Wyswietl_produkty( )
public void Wyswietl_rachunki( )
public void main(String args[0..*])
TFabryka
Attributes
Operations
public TFabryka( )
public TProdukt1 Podaj_produkt(String dane[0..*])
TPromocja
Attributes
protected float promocja = 0
Operations
public TPromocja(float apromocja)
public String toString( )
public float Podaj_promocje( )
TZakup
Attributes
protected int ilosc = 0
Operations
public TProdukt1 getProdukt( )
public void setProdukt(TProdukt1 val)
public TZakup(int ailosc, TProdukt1 aProdukt)
public String toString( )
public boolean equals(Object aTZakup)
public void Dodaj_ilosc(int avoid)
public int Podaj_ilosc( )
public float Podaj_wartosc(int podatek_)
TProdukt1
Attributes
protected String nazwa = ""
protected float cena = 0
Operations
public TPromocja getPromocja( )
public void setPromocja(TPromocja val)
public TProdukt1(String anazwa, float acena)
public TProdukt1(String anazwa, float acena, TPromocja apromocja)
public String toString( )
public boolean equals(Object aTProdukt)
public float Podaj_cene( )
public float Czesc_brutto( )
public String Podaj_nazwe( )
public float Podaj_podatek( )
TProdukt2
Attributes
protected float podatek = 0
Operations
public TProdukt2(String anazwa, float acena, float apodatek)
public TProdukt2(String anazwa, float acena, float apodatek, TPromocja apromocja)
Operations Redefined From TProdukt1
public String toString( )
public float Czesc_brutto( )
public float Podaj_podatek( )
Rachunki
0..*
1
1
0..*
Zakupy
1
0..*
Produkty
0..*
0..*
Produkt
1
0..1
Promocja
0..*

```
// TRachunek – zmiana kodu metody toString(),
// drukująca wartości rachunku w rożnych kategoriach

public String toString()
{
  StringBuilder sb = new StringBuilder();
  sb.append(" Rachunek : ");
  sb.append(numer).append("\n");
   for (TZakup zakup:Zakupy)
          sb.append(zakup.toString()).append("\n");
  sb.append("Wartosc zakupow 0: ").append(Podaj_wartosc(-1)).append("\n");
  sb.append("Wartosc zakupow A: ").append(Podaj_wartosc(3)).append("\n");
  sb.append("Wartosc zakupow B: ").append(Podaj_wartosc(7)).append("\n");
  sb.append("Wartosc zakupow C: ").append(Podaj_wartosc(14)).append("\n");
  sb.append("Wartosc zakupow D: ").append(Podaj_wartosc(22)).append("\n");
  sb.append("Wartosc rachunku: ").append(Podaj_wartosc(-2)).append("\n");
  return sb.toString();
}
```

```
public static void main(String args[])          //kod metody main po
 { TAplikacja app=new TAplikacja();             //implementacji
   String dane1[]={"0","1","1"};                // 6-u przypadków użycia
   String dane2[]={"0","2","2"};                 // identyczny jak po impelemntacji
   app.Dodaj_produkt(dane1);                    // 5-go przypadku użycia
   app.Dodaj_produkt(dane2);
   app.Dodaj_produkt(dane1);
   String dane3[]={"2","3","3","14"};           String dane4[]={"2","4","4","22"};
   app.Dodaj_produkt(dane3);
   app.Dodaj_produkt(dane4);
   app.Dodaj_produkt(dane3);
   String dane5[]={"1","5","1","30"};           String dane6[]={"1","6","2","50"};
   String dane7[]={"3","7","5.47","3","30"};
   String dane8[]={"3","8","13.93","7","50"};
   app.Dodaj_produkt(dane5);
   app.Dodaj_produkt(dane6);
   app.Dodaj_produkt(dane5);
   app.Dodaj_produkt(dane7);
   app.Dodaj_produkt(dane8);
   app.Dodaj_produkt(dane7);
   System.out.println("\nProdukty\n");
   app.Wyswietl_produkty();
```

```
//c.d. kodu metody main po implementacji przypadków użycia:
// Wstawianie nowego zakupu
      app.Wstaw_zakup(1, 1, dane1);
       app.Wstaw_zakup(1, 2, dane2);
       app.Wstaw_zakup(1, 1, dane3);
       app.Wstaw_zakup(1, 4, dane4);
       app.Wstaw_zakup(1, 1, dane5);
       app.Wstaw_zakup(2, 1, dane6);
       app.Wstaw_zakup(2, 3, dane7);
       app.Wstaw_zakup(2, 1, dane8);
       app.Wstaw_zakup(2, 4, dane2);
       app.Wstaw_zakup(2, 1, dane4);
       app.Wstaw_zakup(2, 1, dane6);
       app.Wstaw_zakup(2, 1, dane8);
      System.out.println("\nRachunki\n");
      TRachunek rachunek;
      if ((rachunek = app.Szukaj_rachunek(1)) != null) {
         System.out.println(rachunektoString()); }
      if ((rachunek = app.Szukaj_rachunek(2)) != null) {
         System.out.println(rachunek.toString()); }
   }
}
```


Command Prompt
Produkty
nazwa : 1 cena : 1.0
nazwa : 2 cena : 2.0
nazwa : 3 cena : 3.42 podatek : 14.0
nazwa : 4 cena : 4.88 podatek : 22.0
nazwa : 5 cena : 0.7 promocja : 30.0
nazwa : 6 cena : 0.9 promocja : 55.0
nazwa : 7 cena : 3.99 promocja : 30.0 podatek : 3.0
nazwa : 8 cena : 6.48 promocja : 55.0 podatek : 7.0
Rachunki
Rachunek : 1
ilosc : 1 Produkt : nazwa : 1 cena : 1.0
ilosc : 2 Produkt : nazwa : 2 cena : 2.0
ilosc : 1 Produkt : nazwa : 3 cena : 3.42 podatek : 14.0
ilosc : 4 Produkt : nazwa : 4 cena : 4.88 podatek : 22.0
ilosc : 1 Produkt : nazwa : 5 cena : 0.7 promocja : 30.0
Wartosc zakupow 0: 5.7
Wartosc zakupow A: 0.0
Wartosc zakupow B: 0.0
Wartosc zakupow C: 3.42
Wartosc zakupow D: 19.52
Wartosc rachunku: 28.640001
Rachunek : 2
ilosc : 2 Produkt : nazwa : 6 cena : 0.9 promocja : 55.0
ilosc : 3 Produkt : nazwa : 7 cena : 3.99 promocja : 30.0 podatek : 3.0
ilosc : 2 Produkt : nazwa : 8 cena : 6.48 promocja : 55.0 podatek : 7.0
ilosc : 4 Produkt : nazwa : 2 cena : 2.0
ilosc : 1 Produkt : nazwa : 4 cena : 4.88 podatek : 22.0
Wartosc zakupow 0: 9.8
Wartosc zakupow A: 11.97
Wartosc zakupow B: 12.96
Wartosc zakupow C: 0.0
Wartosc zakupow D: 4.88
Wartosc rachunku: 39.61