Wzorce oprogramowania zastosowane w modelu obiektowym

(wg Alan Shalloway, James R.Trott)

1. **Wzorce**
   - **1.1. Wzorzec fasady**
   - **1.2. Wzorzec fabryki obiektów**
   - **1.3. Wzorzec strategii**
2. **Przykład warstwy biznesowej stosującej wzorce obiektowe**

1. Wzorce projektowe - wprowadzenie

Podstawowe pojęcia dotyczące wzorców projektowych

- 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)**

1.1. Wzorce strukturalne – tworzenie złożonych obiektowych struktur danych

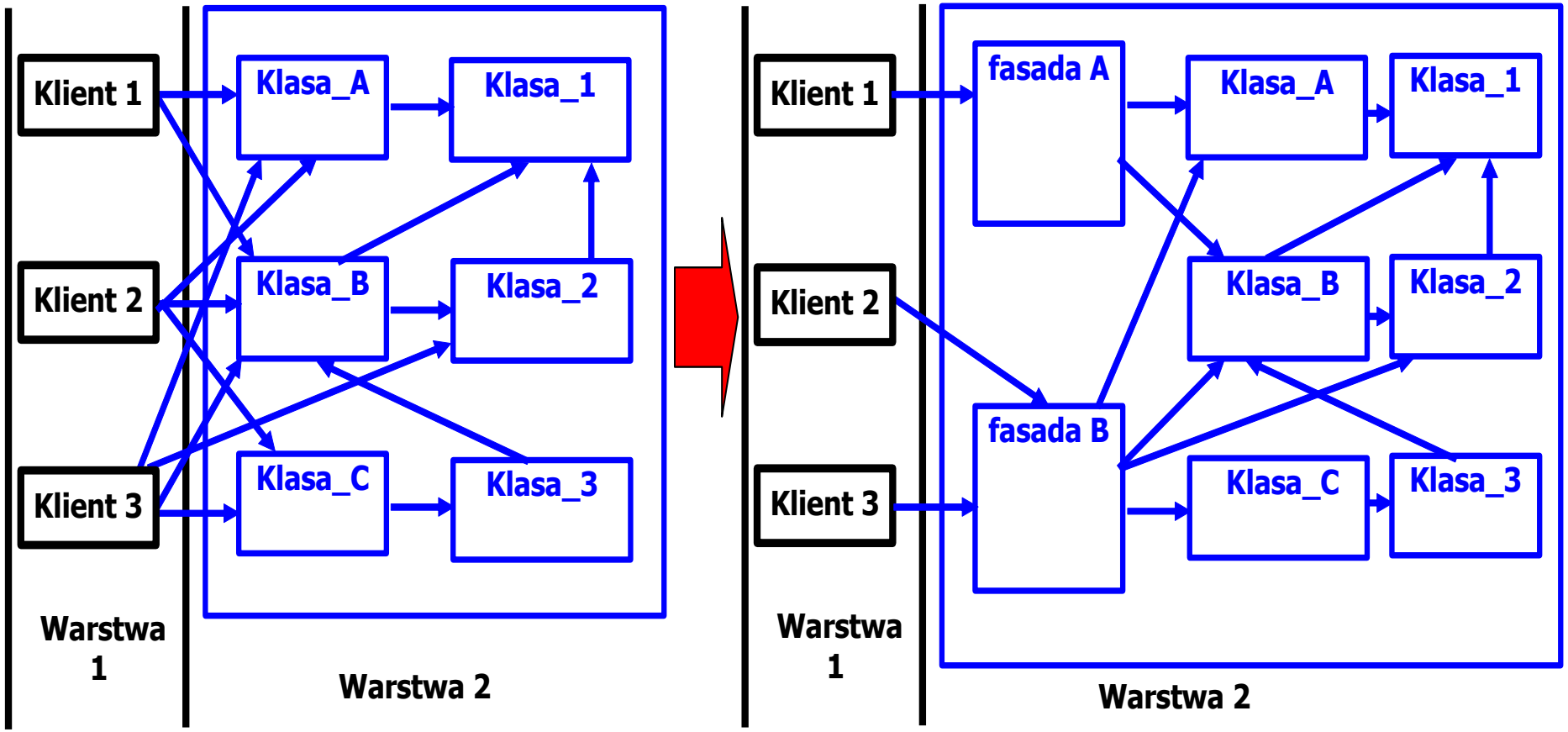
Składanie klas i obiektów w większe struktury

Wzorce klasowe: **zastosowanie dziedziczenia i polimorfizmu do składania struktur interfejsów lub ich implementacji**

Wzorce obiektowe: **opisują sposoby składania obiektów w celu uzyskania nowej funkcjonalności. Istnieje możliwość składania obiektów podczas działania programu**

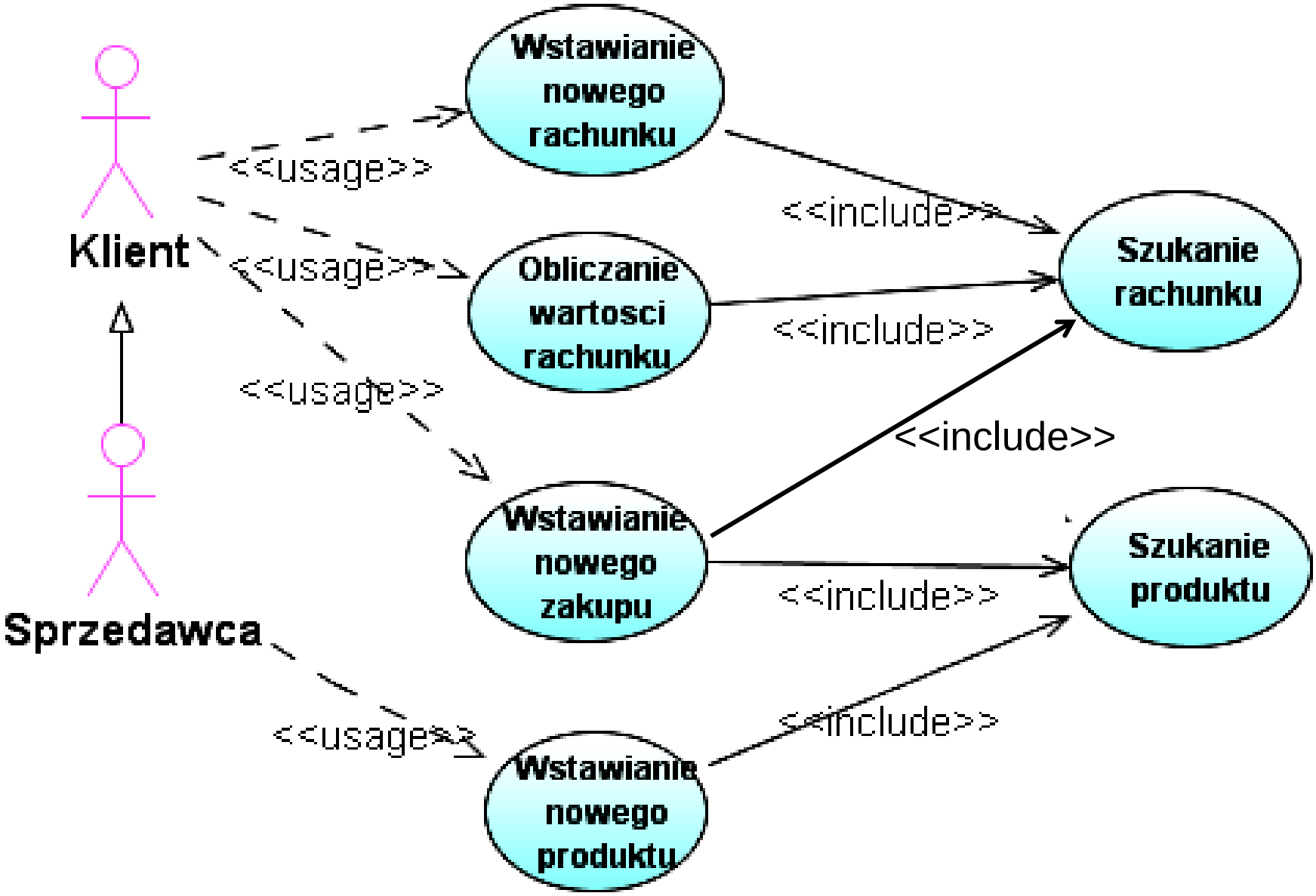
- **Adapter –** ***wzorzec klasowy i obiektowy***
- **Dekorator -** ***Decorator – wzorzec obiektowy***
- **Fasada -** ***Facade - wzorzec obiektowy***
- **Kompozyt –** ***Composite - wzorzec obiektowy***
- **Most –** ***Bridge - wzorzec obiektowy***
- **Pełnomocnik –** ***Proxy - wzorzec obiektowy***
- **Pyłek –** ***Flyweight - wzorzec obiektowy***

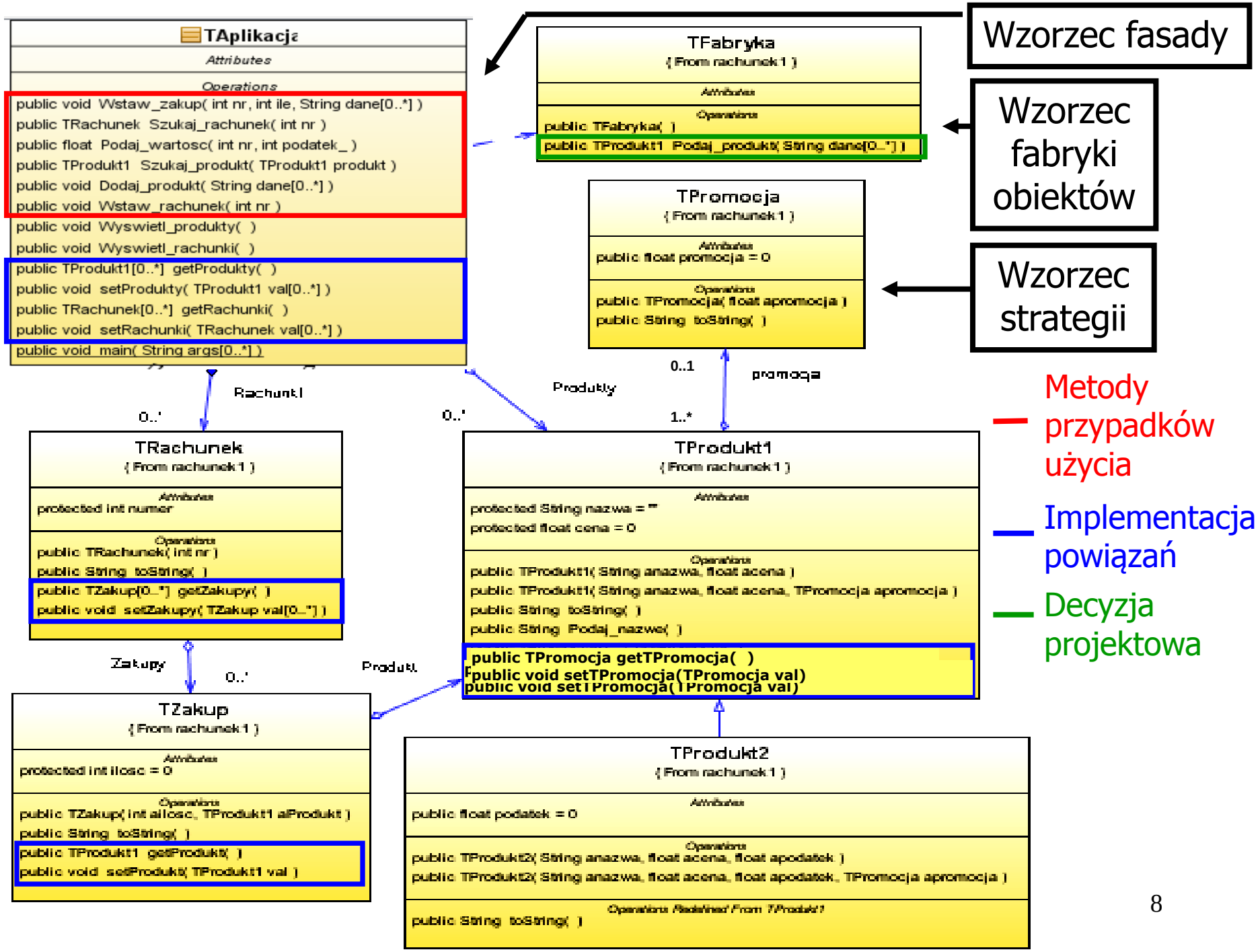
Wzorzec fasady



Charakterystyka wzorca fasady

- **Problem**: udostępnienie tylko wybranych funkcji warstwy systemu
- **Rozwiązanie**: Stanowi interfejs lub interfejsy warstwy systemu- kilka fasad grupuje metody dla wybranych podsystemów
- **Klient wzorca**: otrzymuje jedynie potrzebne metody
- **Rezultat**:
  - Udostępnienie istotnych metod warstwy systemu np. reprezentujących przypadki użycia – hermetyzacja klas warstwy systemu
  - Fasada uniemożliwia dostęp do wszystkich metod hermetyzowanych klas
- **Implementacja**: nowa klasa typu „Control”


Wstawianie nowego rachunku
Klient
<<usage>>
<<usage>>
Obliczanie wartosci rachunku
<<include>>
Szukanie rachunku
<<include>>
<<usage>>
<<include>>
Wstawianie nowego zakupu
Szukanie produktu
<<include>>
Sprzedawca
<<usage>>
<<include>>
Wstawianie nowego produktu


TAplikacja
Attributes
Operations
public void Wstaw_zakup(int nr, int ile, String dane[0..*])
public TRachunek Szukaj_rachunek(int nr)
public float Podaj_wartosc(int nr, int podatek_)
public TProdukt1 Szukaj_produkt(TProdukt1 produkt)
public void Dodaj_produkt(String dane[0..*])
public void Wstaw_rachunek(int nr)
public void Wyswietl_produkty( )
public void Wyswietl_rachunki( )
public TProdukt1[0..*] getProdukty( )
public void setProdukty(TProdukt1 val[0..*])
public TRachunek[0..*] getRachunki( )
public void setRachunki(TRachunek val[0..*])
public void main(String args[0..*])
TFabryka
{ From rachunek1 }
Attributes
Operations
public TFabryka( )
public TProdukt1 Podaj_produkt(String dane[0..*])
TPromocja
{ From rachunek1 }
Attributes
public float promocja = 0
Operations
public TPromocja(float apromocja)
public String toString( )
Wzorzec fasady
Wzorzec fabryki obiektów
Wzorzec strategii
Rachunki
0..*
Produkty
0..*
0..1
promocja
1..*
TRachunek
{ From rachunek1 }
Attributes
protected int numer
Operations
public TRachunek(int nr)
public String toString( )
public TZakup[0..*] getZakupy( )
public void setZakupy(TZakup val[0..*])
TProdukt1
{ From rachunek1 }
Attributes
protected String nazwa = ""
protected float cena = 0
Operations
public TProdukt1(String anazwa, float acena)
public TProdukt1(String anazwa, float acena, TPromocja apromocja)
public String toString( )
public String Podaj_nazwe( )
public TPromocja getTPromocja( )
public void setTPromocja(TPromocja val)
Zakupy
0..*
Produkt
TZakup
{ From rachunek1 }
Attributes
protected int ilosc = 0
Operations
public TZakup(int ailosc, TProdukt1 aProdukt)
public String toString( )
public TProdukt1 getProdukt( )
public void setProdukt(TProdukt1 val)
TProdukt2
{ From rachunek1 }
Attributes
public 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( )
Metody przypadków użycia
Implementacja powiązań
Decyzja projektowa

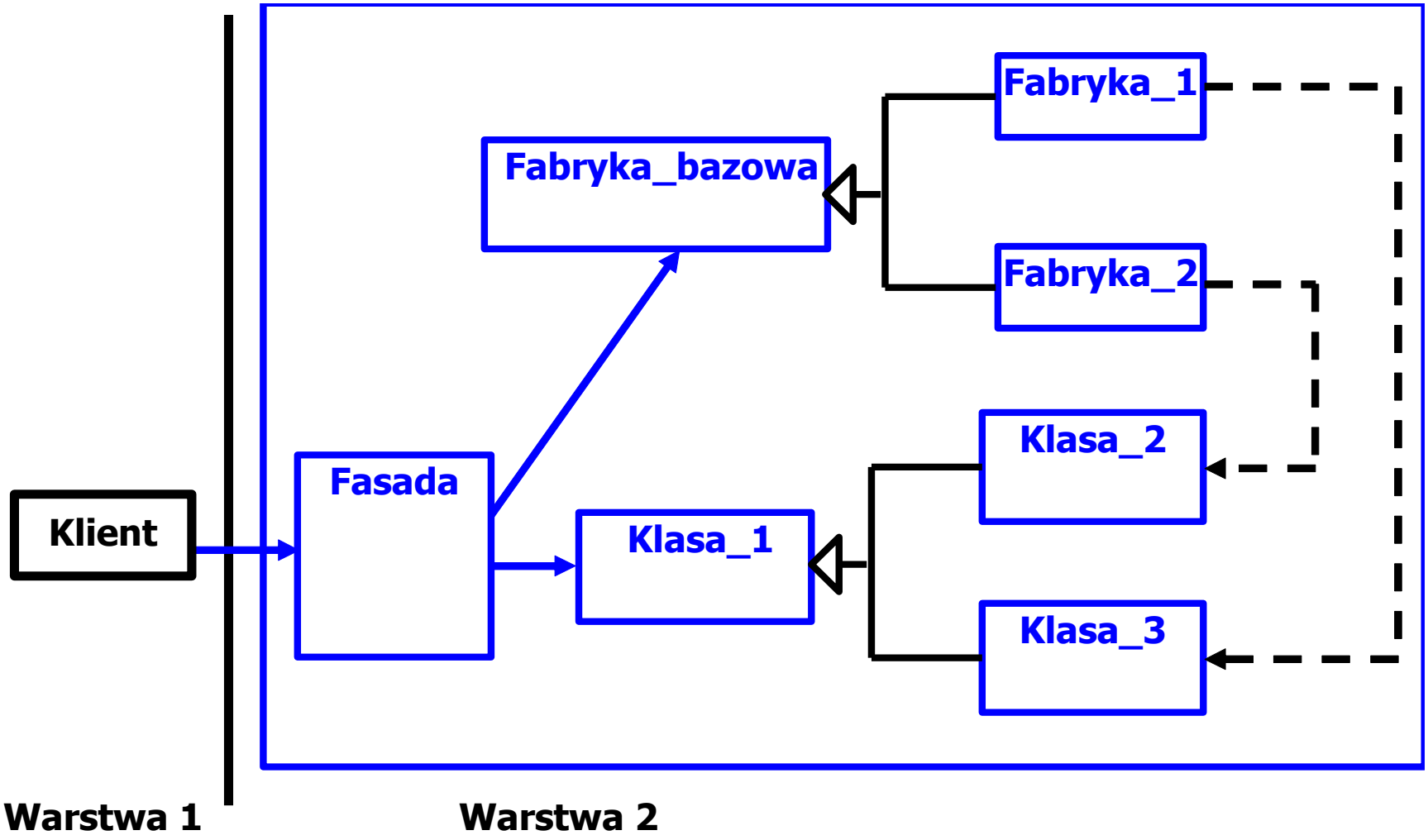
```
package rachunek1;
import java.util.ArrayList;
public class TAplikacja
{    private List<TProdukt1> Produkty = new ArrayList<TProdukt1>();
     private List<TRachunek> Rachunki = new ArrayList<TRachunek>();

     List<TProdukt1> getProdukty ()                          { return null;  }
     void setProdukty (ArrayList<TProdukt1> val)             {    }
     List<TRachunek> getRachunki ()                          { return null;  }
     void setRachunki (ArrayList<TRachunek> val)             {    }

     public void Wstaw_zakup (int nr, int ile, String dane[]) {    }
     TRachunek Szukaj_rachunek (int nr)                      { return null;  }
     public void Wstaw_rachunek (int nr)                     {    }
     public float Podaj_wartosc (int nr, int podatek_)       { return 0.0f;  }
     TProdukt1 Szukaj_produkt (TProdukt1 produkt)            { return null;  }
     public void Dodaj_produkt (String[] dane)               {    }

     public static void main (String[] args)                 {    }
     public void Wyswietl_produkty ()                        {    }
     public void Wyswietl_rachunki ()                        {    }
}
```

1.2. Wzorce kreacyjne

Izolacja reguł tworzenia obiektów od reguł określających sposób używania obiektów (oddzielenie w kodzie programu kodu tworzącego obiekty od kodu, który używa obiekty) – klasy typu „Control”

1) Budowniczy - *Builder*
2) **Fabryka abstrakcyjna – *Abstract Factory***
3) Metoda wytwórcza – *Factory Method*
4) Prototyp - *Prototype*
5) Singleton

Wzorzec fabryki obiektów

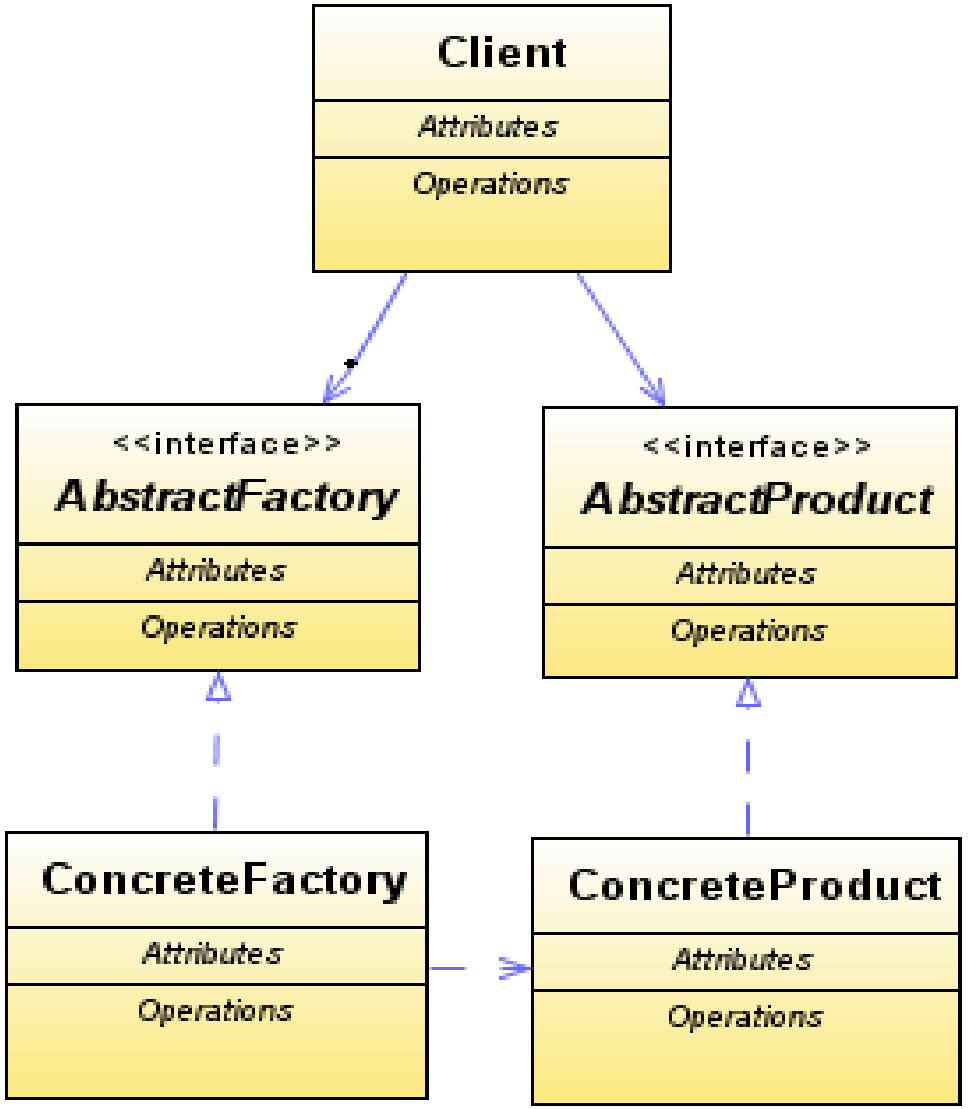


Fabryka abstrakcyjna – *Abstract Factory*

Selects the participants for the roles in the pattern.

Role	Participant
AbstractFactory	AbstractFactory
ConcreteFactory	
ConcreteFactory	ConcreteFactory
AbstractProduct	
AbstractProduct	AbstractProduct
ConcreteProduct	
ConcreteProduct	ConcreteProduct
Client	Client

< Back | Next > | Finish | Cancel



Charakterystyka wzorca fabryki obiektów

- **Problem**: Tworzenie właściwych rodzin obiektów w określonym przypadku. Przyczyny istnienia różnych rodzin obiektów:
  - różne systemy operacyjne,
  - różne wymagania dotyczące efektywności, wydajności itp.
  - różne wersje aplikacji
  - różne typy aplikacji współpracujących (np. różne typy baz danych)
  - różne funkcjonalności dla różnych użytkowników
  - różne grupy elementów zależnych od ustawień związanych z lokalizacją (okna dialogowe, format zapisu danych)
- **Rozwiązanie**: Określenie reguł tworzenia obiektów, które mogą najlepiej realizować cele aplikacji
- **Rezultat**:
  - Izolacja reguł tworzenia obiektów od reguł określających sposób używania obiektów
  - Określenie reguł tworzenia obiektów, które mogą najlepiej realizować cele aplikacji
  - Konfiguracja aplikacji za pomocą powiązanych rodzin obiektów np. ***TProdukt*** i ***TPromocja***
  - System używa tworzone obiekty znając klasy bazowe tych obiektów i klasy bazowe fabryk
- **Klient wzorca**: obiekt klasy **zarządzający** obiektami tworzonymi przez fabrykę obiektów, **który powinien być niezależny** od reguł tworzenia tych obiektów
- **Implementacja**: Zdefiniowanie klasy bazowej typu „Control ”dla klas tworzonej rodziny obiektów. Do wyboru obiektów można stosować pliki konfiguracyjne, tabele baz danych itp.

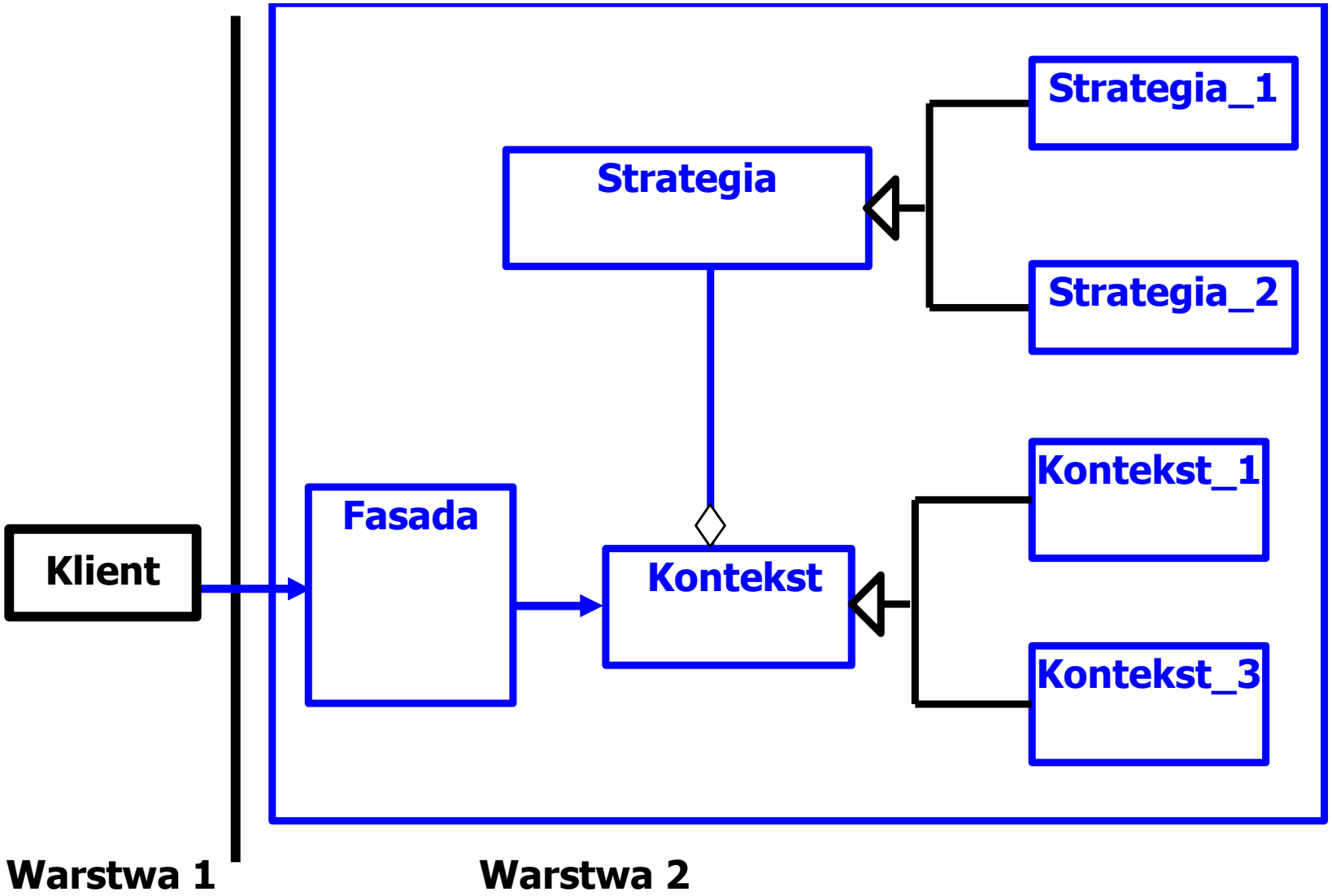
```
public class TFabryka
{
   public TFabryka() { }
   public TProdukt1 Podaj_produkt(String dane[])
   {
      TProdukt1 produkt = null;
      TPromocja promocja = null;
      switch ( Integer.parseInt(dane[0]) )
        {
          case 0: produkt= new TProdukt1(dane[1],
                                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;
           //.....
```

1.3. Wzorce czynnościowe

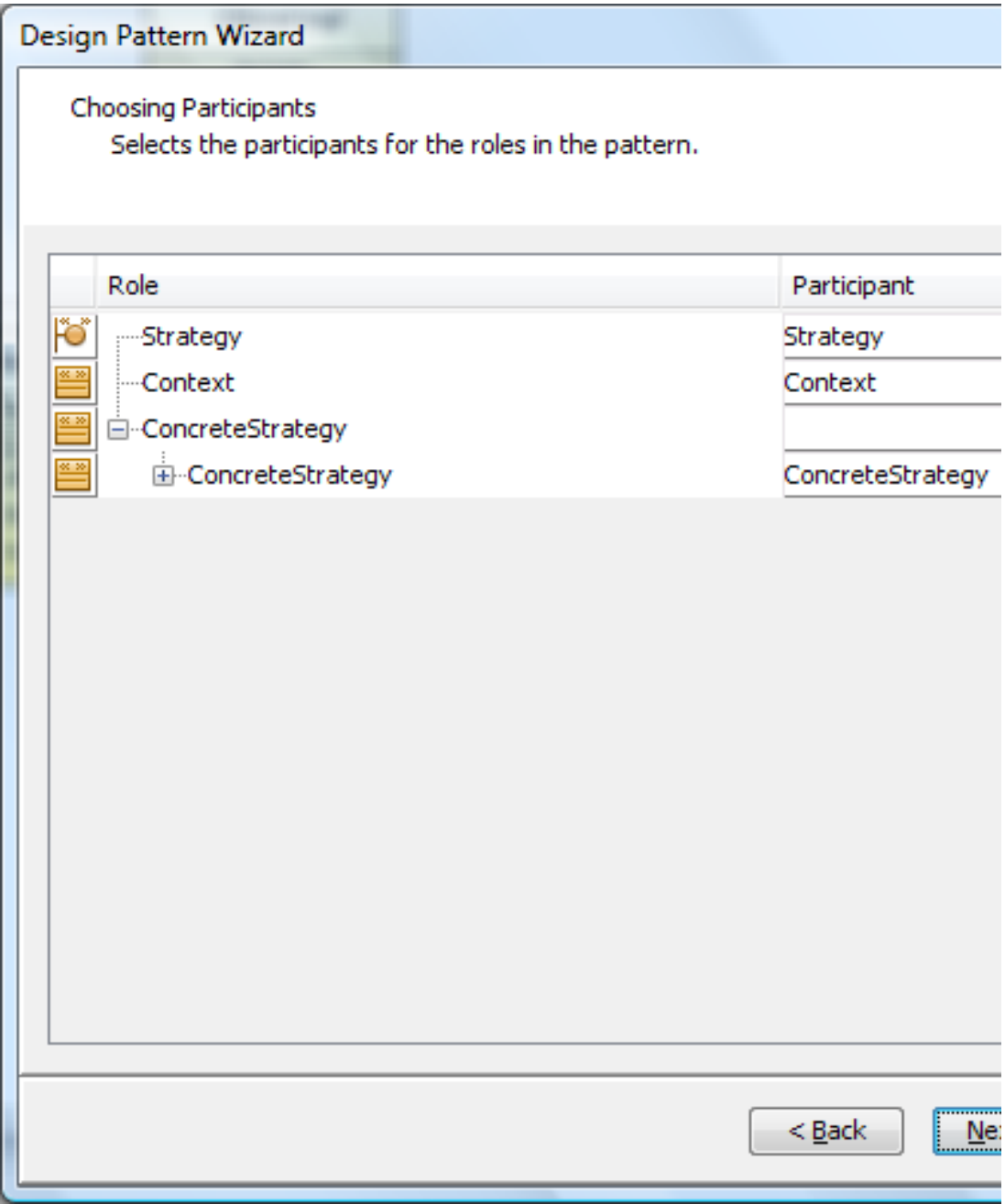
Przydzielają algorytmy i zobowiązania obiektom, obejmują wzorce obiektów, klas oraz komunikacji między obiektami

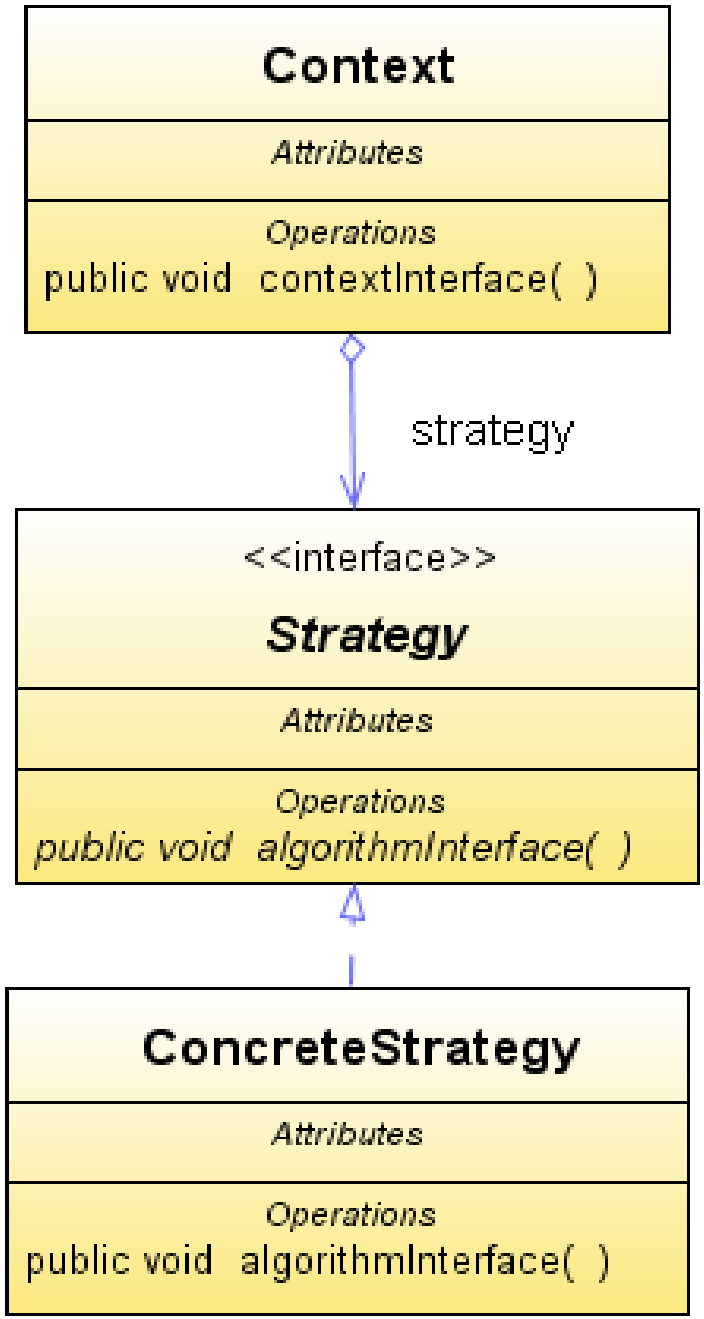
1) Interpreter
2) Iterator
3) Łańcuch zobowiązań – *Chain of Responsibility*
4) Mediator
5) Metoda szablonowa – *Template method*
6) Obserwator - *Observer*
7) Odwiedzający - Visitor
8) Pamiątka - *Memento*
9) Polecenie - *Command*
10) Stan - *State*
11) **Strategia - *Strategy***

Wzorzec strategii



Strategia - *Strategy*





Charakterystyka wzorca strategii

- **Problem:** Wybór różnych reguł biznesowych lub różnych wersji algorytmów w zależności od kontekstu
- **Rozwiązanie**: Separuje wybór wersji algorytmu od jego implementacji
- **Rezultat**:
  - Zdefiniowanie rodziny algorytmów
  - Zdefiniowanie interfejsu klasy typu **Strategia** zawierającej metodę dostarczającą algorytm i metody w klasie typu **Kontekst**, która korzysta z algorytmu
  - Eliminacja instrukcji wyboru lub warunkowej do wyboru algorytmu strategii- wprowadzenie mechanizmu polimorfizmu, szczególnie, gdy wybór algorytmu nie ma charakteru przejściowego
- **Klient**: Decyzję o **implementacji** obiektu strategii i kontekstu podejmuje właściciel tych obiektów, który dostarcza informacji o utworzeniu właściwego obiektu strategii oraz obiektu kontekstu np. za pomocą fabryki obiektów. Jest nim obiekt typu **klient wzorca** np. dowolny obiekt z warstwy prezentacji.
- **Implementacja**:
  - Obiekt kontekstowy klasy bazowej typu **Kontekst („Entity")** i jej pochodnych, który używa algorytmu, posiada obiekt klasy bazowej typu **Strategia („Entity")** lub jej pochodnych**,** dostarczający algorytm**.** Obiekt kontekstu posiada metodę wirtualną wywołującą wirtualną metodę algorytmu obiektu strategii. Każda klasa dziedzicząca od klasy typu **Strategia** implementuje taką metodę algorytmu w inny sposób.
  - Decyzja o sposobie użycia strategii, czyli użycia właściwego obiektu typu **Strategia** zależy od klasy typu **Kontekst**

klient (obiekt wywołujący metodę *Dodaj_produkt*) – obiekt dokonujący wyboru implementacji **kontekstu** i przekazujący mu implementację **strategii**

```
//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);
  }
```

```
public class TFabryka
{ 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: promocja = new TPromocja1(Float.parseFloat(dane[4]));
            produkt = new TProdukt2(dane[1],Float.parseFloat(dane[2]),
                          Float.parseFloat(dane[3]),promocja);     break;
   case 3: promocja = new TPromocja2(Float.parseFloat(dane[4]));
            produkt= new TProdukt2(dane[1], Float.parseFloat(dane[2]),
                          Float.parseFloat(dane[3]),promocja);     break;
    }
    return produkt;
  }
```

promocja – obiekt z rodziny typu **Strategia**

produkt – obiekt z rodziny **Kontekst**

```
//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;
  }
```

2. Przykład warstwy biznesowej stosującej wzorce obiektowe

System sporządzania rachunków

2.1. Sformułowanie wymagań funkcjonalnych i niefunkcjonalnych systemu

2.2. Model analizy całego systemu oparty na diagramie przypadków użycia

2.3. Model projektowy warstwy biznesowej oparty na diagramie klas i diagramie sekwencji tworzony metodą iteracyjno-rozwojową sterowany realizacją przypadków użycia

2.4. Implementacja warstwy biznesowej tworzona w cyklu iteracyjno-rozwojowym sterowana rozwojem modelu projektowego

2.1. Sformułowanie wymagań funkcjonalnych i niefunkcjonalnych systemu

System sporządzania rachunków

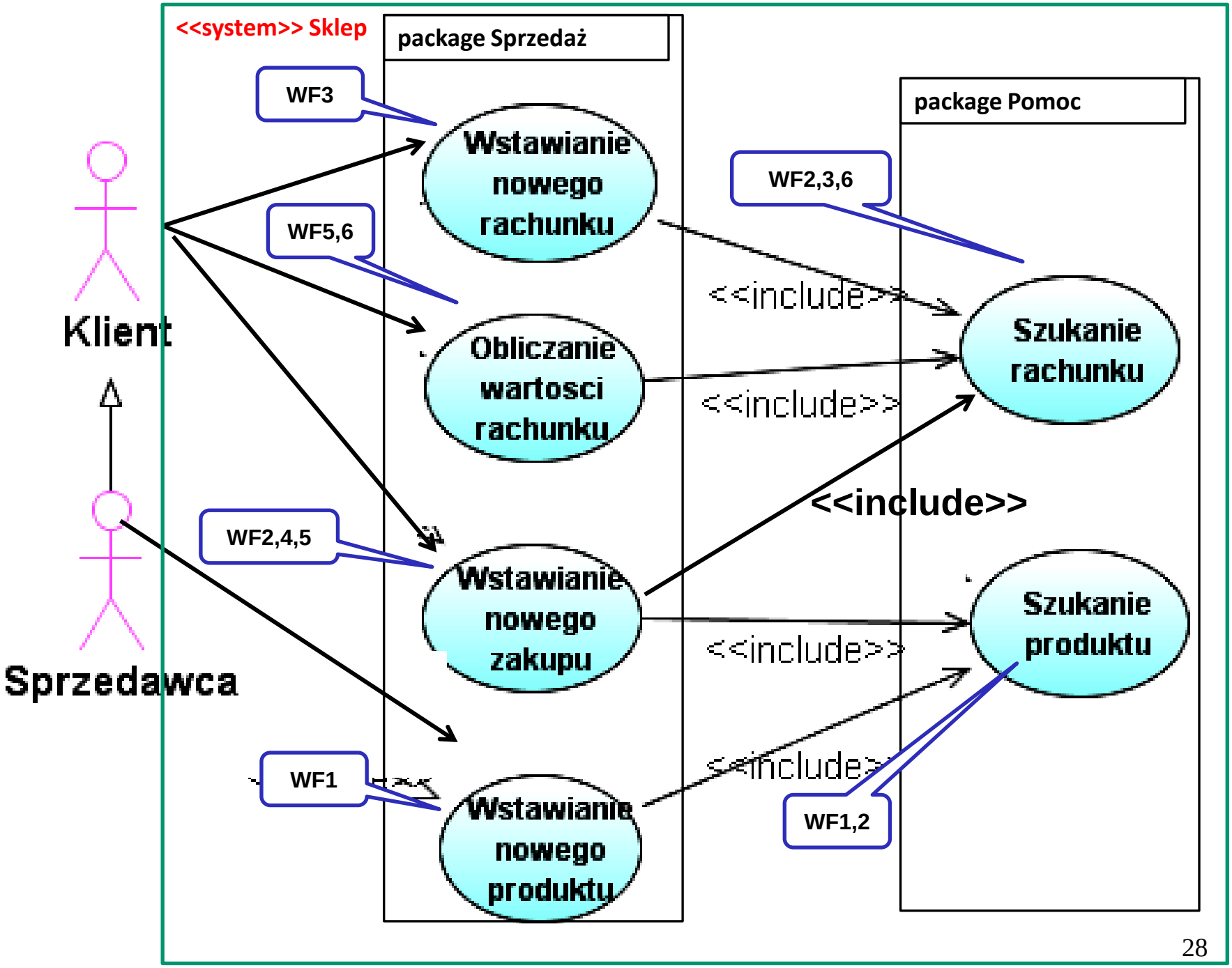
Lista wymagań funkcjonalnych

1. System zawiera katalog produktów
2. Można zakupić trzy typy produktów różniące się sposobem obliczania ceny detalicznej: netto, z podatkiem, 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

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

2.2. **Model analizy całego systemu** oparty na diagramie przypadków użycia


<<system>> Sklep
package Sprzedaż
package Pomoc
WF3
WF5,6
WF2,4,5
WF1
WF2,3,6
WF1,2
Wstawianie nowego rachunku
Obliczanie wartosci rachunku
Wstawianie nowego zakupu
Wstawianie nowego produktu
Szukanie rachunku
Szukanie produktu
<<include>>
<<include>>
<<include>>
<<include>>
<<include>>
Klient
Sprzedawca

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

1. Opisy przypadków użycia do przykładu 3

PU Szukanie produktu
OPIS
CEL: Poszukiwanie produktu
WS (warunki wstępne): może być wywołany z PU Wstawianie nowego produktu, PU Wstawianie nowego zakupu
WK (warunki końcowe): podanie produktu o podanych atrybutach obowiązkowych: nazwa i cena oraz jeśli jest to wymagane: z podatkiem i promocją lub komunikat o braku produktu
PRZEBIEG:
1. Szukanie produktu przebiega wedlug atrybutow: nazwy i ceny (obowiazkowo) oraz podatku i promocji **(jeśli jest to wymagane) zgodnie z danymi podanymi do przypadku uzycia**
2. Jesli istnieje produkt o podanych atrybutach, zwracany jest produkt, w przeciwnym wypadku

PU Wstawianie nowego produktu
OPIS
CEL: Wstawienie nowego produktu
WS (warunki wstępne): inicjalizacja przez uruchomienie programu (np. otwarcie strony WWW, start aplikacji)
WK (warunki końcowe): dodanie produktu o podanych atrybutach obowiązkowych: nazwa i cena oraz jeśli jest to wymagane: z podatkiem i promocją, jeśli nie było takiego produktu
PRZEBIEG:
1. Nalezy podac atrybuty produktu: nazwe, cene jako obowiazkowe dane oraz podatek i cene detaliczna, jeśli jest to wymagane
2. 2. Należy wywolac **PU Szukanie produktu**. Nalezy sprawdzic, czy produkt o podanych atrybutach juz istnieje. Jesli tak, nalezy zakonczyc PU, w przeciwnym wypadku nalezy wstawic nowy produkt.

PU Szukanie rachunku
OPIS
CEL: Poszukiwanie rachunku
WS (warunki wstępne): może być wywołany z PU Wstawianie nowego rachunku, PU Wstawianie nowego zakupu, PU Obliczanie wartosci rachunku
WK (warunki końcowe): podanie rachunku o podanym numerze lub komunikat o braku rachunku
PRZEBIEG:
1. Szukanie rachunku przebiega wedlug numeru podanego do przypadku uzycia
2. Jesli istnieje rachunek o podanym numerze, zwracany jest rachunek, w przeciwnym wypadku zwracana jest informacja o braku rachunku.

PU Wstawianie nowego rachunku
OPIS
CEL: Wstawienie nowego rachunku
WS (warunki wstępne): inicjalizacja przez uruchomienie programu (np. otwarcie strony WWW, start aplikacji)
WK (warunki końcowe): dodanie rachunku o podanym numerze, jeśli jest to unikatowy numer
PRZEBIEG:
1. Nalezy podac numer rachunku, ktory powinien byc niepowtarzalny, poniewaz sluzy do identyfikacji rachunku
2. Nalezy wywolac **PU Szukanie rachunku** w celu sprawdzenia, czy numer rachunku sie powtarza.
3. Jesli zwrocony wynik oznacza brak rachunku o podanym numerze, mozna wstawic nowy rachunek i zakonczyc PU, w przeciwnym wypadku nalezy zakonczyc PU bez wstawiania nowego rachunku.

PU Wstawianie nowego zakupu
OPIS
CEL: Wstawianie nowego zakupu
WS (warunki wstępne): inicjalizacja przez uruchomienie programu (np. otwarcie strony WWW, start aplikacji)
WK (warunki końcowe): dodanie nowego zakupu o podanych atrybutach lub zwiekszenie ilosci zakupionego produktu, jeśli już taki produkt zakupiono lub komunikat o braku rachunku
PRZEBIEG:
1. Nalezy podac numer rachunku, ktory powinien byc niepowtarzalny, poniewaz sluzy do identyfikacji rachunku
2. Nalezy wywolac **PU Szukanie rachunku** w celu sprawdzenia, czy istnieje rachunek o podanym numerze.
3. Jesli zwrocony wynik oznacza brak rachunku o podanym numerze, nie mozna wstawic nowego zakupu do rachunku i nalezy zakonczyc PU, w przeciwnym wypadku nalezy wstawic nowy zakup
4. Nalezy wybrac produkt oraz ilosc zakupionego produktu.
5. Należy wywolac **PU Szukanie produktu**. Jeśli wybrany produkt nie istnieje, należy zakonczyc PU. W przeciwnym przypadku nalezy wstawic nowy zakup do rachunku, przegladajac, czy istnieje juz zakup z takim samym produktem. Jesli istnieje, nie tworzy sie nowego zakupu, tylko powieksza się ilosc zakupu istniejacego o ilosc nowego zakupu, w przeciwnym przypadku wstawia sie nowy zakup.

PU Obliczanie wartosci rachunku

OPIS

CEL: Obliczanie wartosci rachunku wg podanego podatku

WS (warunki wstępne): inicjalizacja przez uruchomienie programu (np. otwarcie strony WWW, start aplikacji)

WK (warunki końcowe): podanie wartości całego rachunku o podanym numerze i parametrze wejściowym równym -2 lub wartości zakupionych towarów wg podanej kategorii podatku lub komunikat o braku rachunku

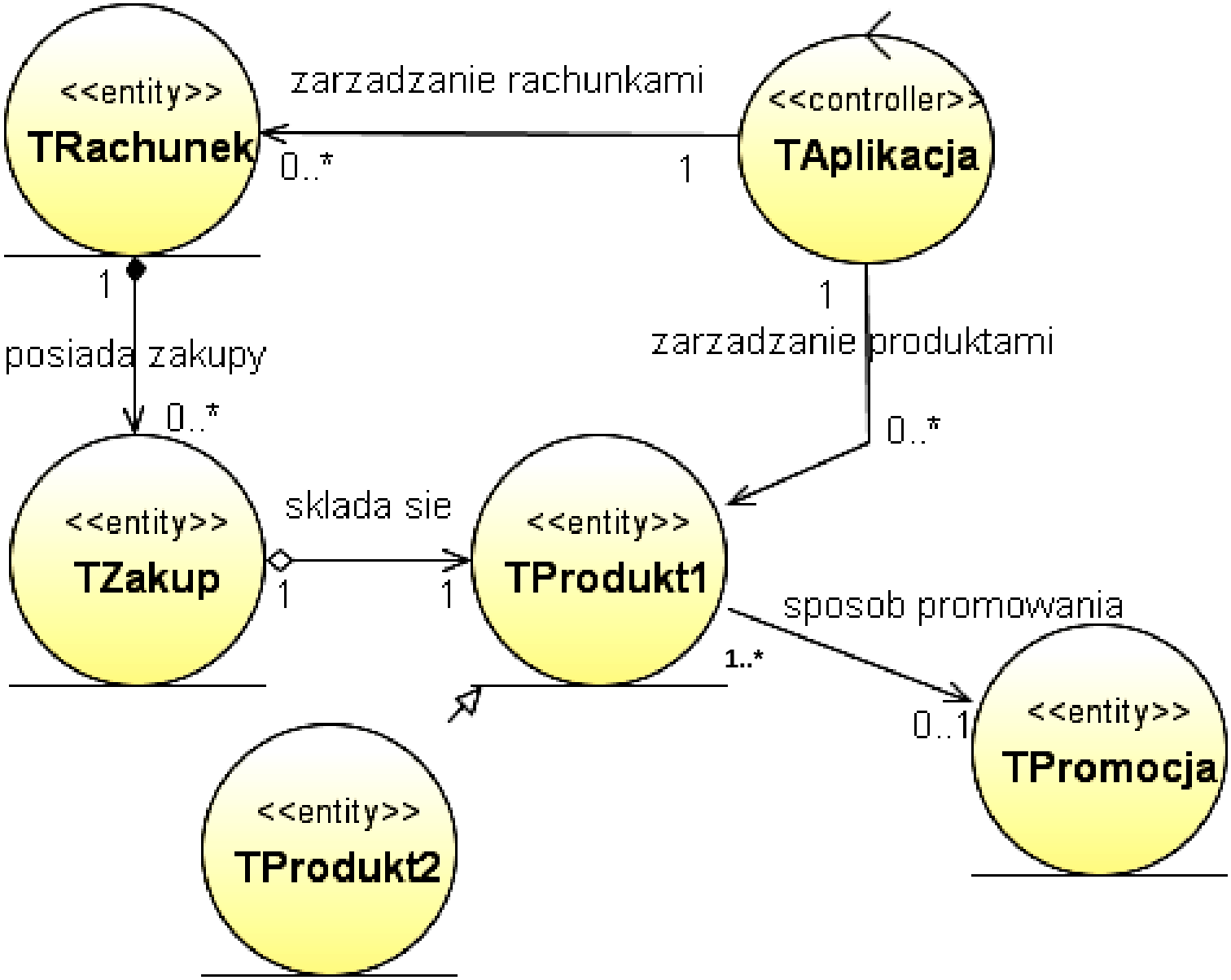
PRZEBIEG:

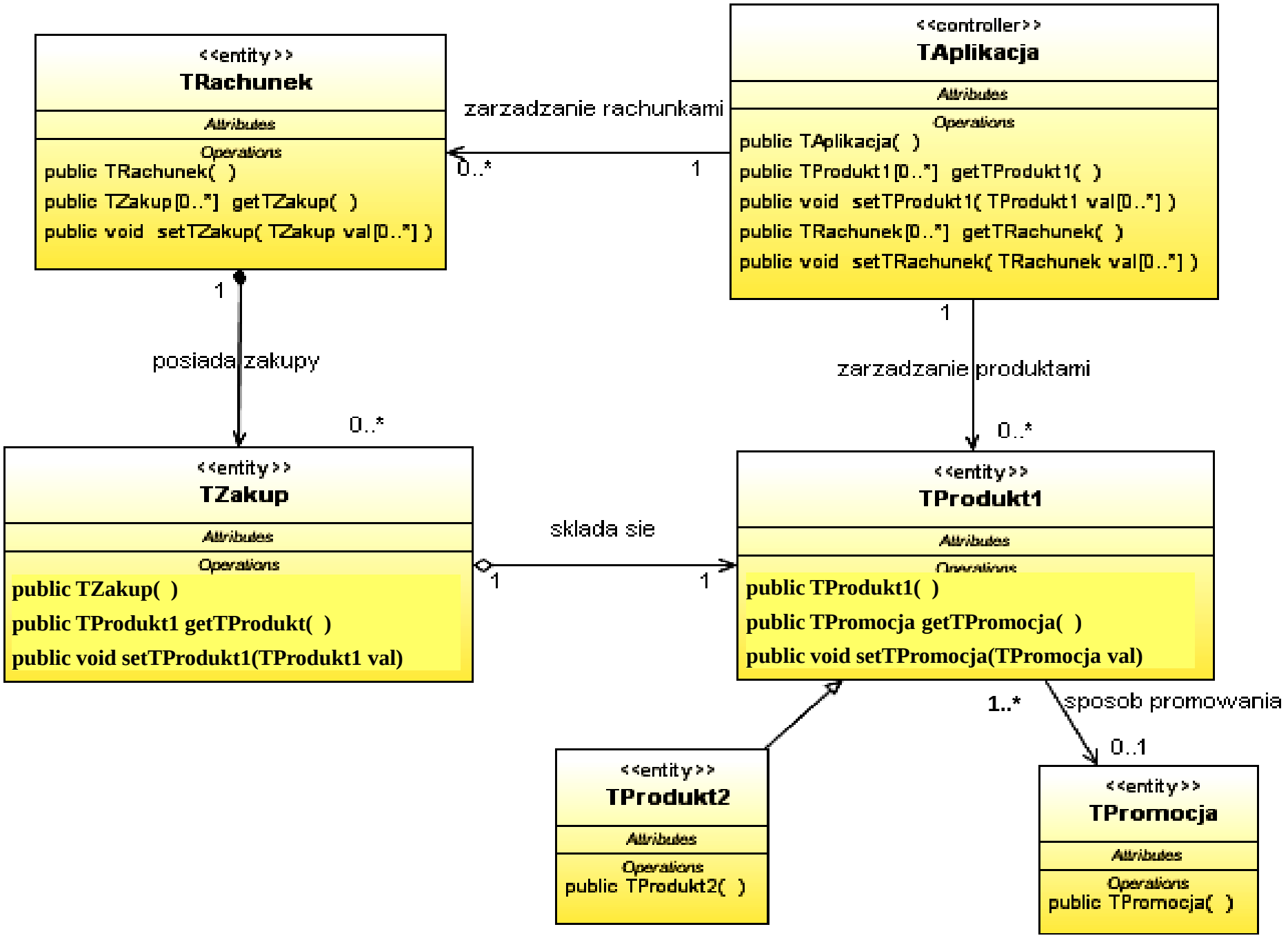
1. Nalezy podac numer rachunku, ktory powinien byc niepowtarzalny, poniewaz sluzy do identyfikacji rachunku oraz wartość podatku lub wartosc -2
2. Nalezy wywolac **PU Szukanie rachunku** w celu sprawdzenia, czy rachunek o podanym numerze istnieje.
3. Jesli zwrocony wynik oznacza brak rachunku o podanym numerze, nie mozna obliczyc wartosci wybranego rachunku i nalezy zakonczyc PU, w przeciwnym wypadku nalezy obliczyc wartosc rachunku
4. Nalezy uruchomic petle, w ktorej sumowane sa wartosci zakupu obliczane jako iloczyn ceny jednostkowej zakupionego produktu i ilosci zakupu. Jesli zachodzi potrzeba sumowania wartosci zakupu zalezna od wysokosci podatku, nalezy podac wartosc podatku i sumowac jedynie zakupy o podanym podatku, w przeciwnym wypadku sumowane sa wszystkie zakupy (gdy zamiast podatku zostanie przekazana wartosc -2).

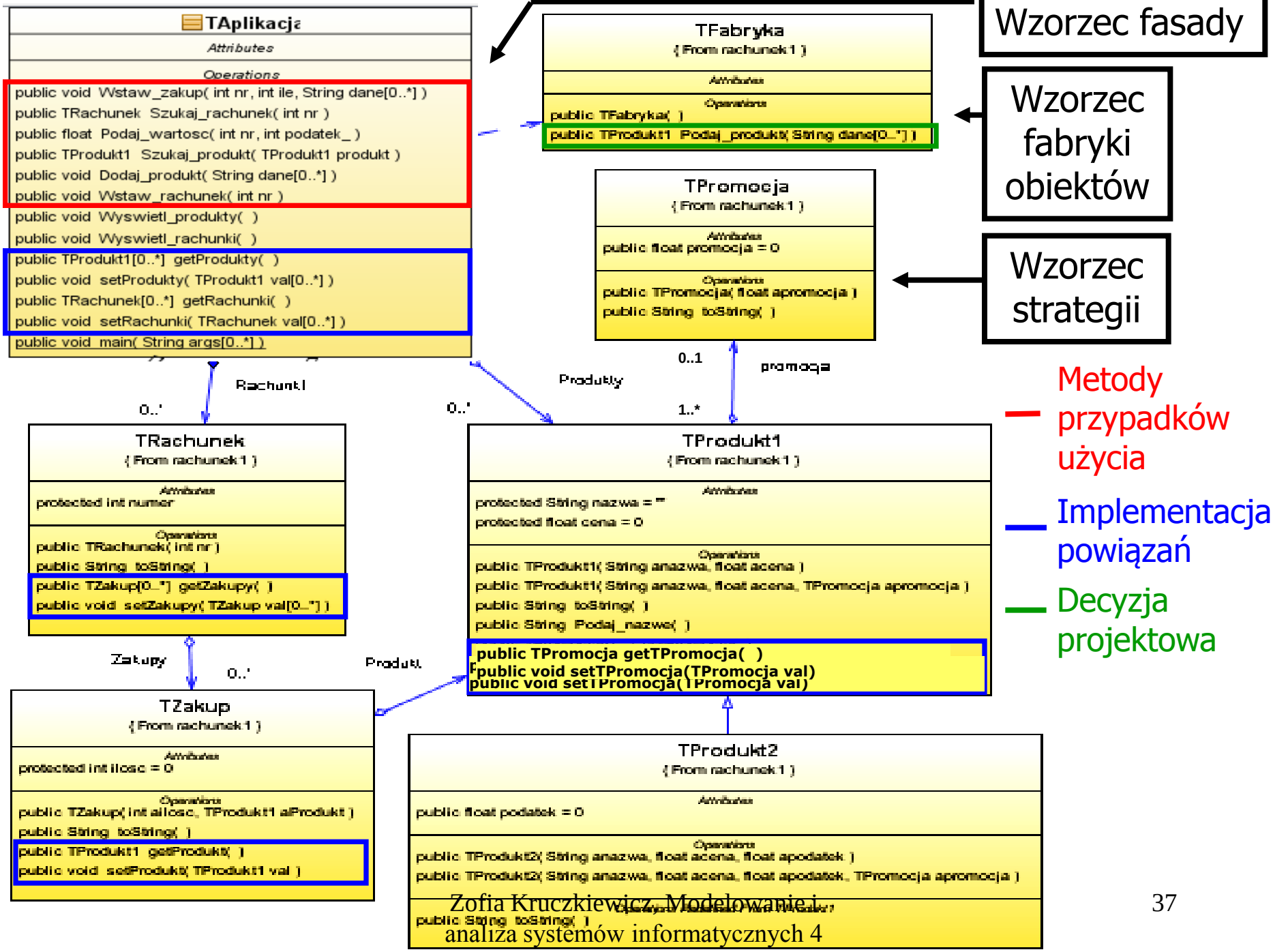
Analiza wspólności i zmienności

- Wykryto trzy główne klasy typu „Entity” ze względu na odpowiedzialność: TRachunek (wstawia zakupy, oblicza wartość), TZakup (oblicza wartość zakupu) oraz TProdukt1 (posiada nazwę oraz oblicza cenę detaliczną)
- 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 (klasa TProdukt2 typu „Entity”, która dziedziczy od klasy TProdukt1) oraz strategię zmniejszania ceny jednostkowej wynikającej z promocji powiązaną z produktem zarówno z podatkiem, jak bez podatku. Ponieważ jednak promocja nie musi dotyczyć każdego produktu, jest w związku powiązania z bazowym (głównym) produktem typu 0..* do 1. Klasa TPromocja typu „Entity” jest dziedziczona przez pozostałe typy produktu. Stad produkt powinien 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 (cztery typy ceny detalicznej).
- Wykryto związki silnej agregacji między rachunkiem i zakupami (rachunek posiada kolekcję zakupów) oraz słabej agregacji między zakupem a produktem (zakup składa się z produktu bazowego lub jego następców), oraz związek typu powiązanie lub agregacja między promocją a produktem bazowym dziedziczony przez produkty potomne. Wybrano **wzorzec strategii** do implementacji promocji
- Zastosowano klasęTAplikacja typu „Control” jako **wzorzec fasady** do oddzielenia obiektów typu „Entity” od pozostałej części systemu oraz klasę typu „Control” jako **wzorzec fabryki obiektów** (TFabryka) do tworzenia różnych typów produktów

Diagram klas – koncepcja klas typu „Entity” oraz „Controller”




«entity»
TRachunek
Attributes
Operations
public TRachunek( )
public TZakup[0..*] getTZakup( )
public void setTZakup(TZakup val[0..*])
zarzadzanie rachunkami
0..*
1
«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..*])
1
posiada zakupy
0..*
1
zarzadzanie produktami
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)
1..*
sposob promowania
0..1
«entity»
TProdukt2
Attributes
Operations
public TProdukt2( )
«entity»
TPromocja
Attributes
Operations
public TPromocja( )


TAplikacja
Attributes
Operations
public void Wstaw_zakup(int nr, int ile, String dane[0..*])
public TRachunek Szukaj_rachunek(int nr)
public float Podaj_wartosc(int nr, int podatek_)
public TProdukt1 Szukaj_produkt(TProdukt1 produkt)
public void Dodaj_produkt(String dane[0..*])
public void Wstaw_rachunek(int nr)
public void Wyswietl_produkty( )
public void Wyswietl_rachunki( )
public TProdukt1[0..*] getProdukty( )
public void setProdukty(TProdukt1 val[0..*])
public TRachunek[0..*] getRachunki( )
public void setRachunki(TRachunek val[0..*])
public void main(String args[0..*])
TFabryka
{ From rachunek1 }
Attributes
Operations
public TFabryka( )
public TProdukt1 Podaj_produkt(String dane[0..*])
TPromocja
{ From rachunek1 }
Attributes
public float promocja = 0
Operations
public TPromocja(float apromocja)
public String toString( )
0..1
promocja
1..*
Produkty
0..*
Rachunki
0..*
TRachunek
{ From rachunek1 }
Attributes
protected int numer
Operations
public TRachunek(int nr)
public String toString( )
public TZakup[0..*] getZakupy( )
public void setZakupy(TZakup val[0..*])
TProdukt1
{ From rachunek1 }
Attributes
protected String nazwa = ""
protected float cena = 0
Operations
public TProdukt1(String anazwa, float acena)
public TProdukt1(String anazwa, float acena, TPromocja apromocja)
public String toString( )
public String Podaj_nazwe( )
public TPromocja getTPromocja( )
public void setTPromocja(TPromocja val)
Zakupy
0..*
Produkt
TZakup
{ From rachunek1 }
Attributes
protected int ilosc = 0
Operations
public TZakup(int ailosc, TProdukt1 aProdukt)
public String toString( )
public TProdukt1 getProdukt( )
public void setProdukt(TProdukt1 val)
TProdukt2
{ From rachunek1 }
Attributes
public float podatek = 0
Operations
public TProdukt2(String anazwa, float acena, float apodatek)
public TProdukt2(String anazwa, float acena, float apodatek, TPromocja apromocja)
public String toString( )
Wzorzec fasady
Wzorzec fabryki obiektów
Wzorzec strategii
Metody przypadków użycia
Implementacja powiązań
Decyzja projektowa

Wygenerowany kod klas na podstawie diagramu klas opracowanego w fazy analizy

```
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[]) {    }
     TRachunek Szukaj_rachunek (int nr)                      { return null;  }
     public void Wstaw_rachunek (int nr)                     {    }
    public float Podaj_wartosc (int nr, int podatek_)        {  return 0.0f;  }
    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)                 {    }
}
```

```
package rachunek1;
import java.util.ArrayList;
import java.util.List;

class TRachunek
{
    protected int numer;
    private List<TZakup> Zakupy = new ArrayList<TZakup>();
    public List<TZakup> getZakupy ()            { return null;   }
    public void setZakupy (ArrayList<TZakup> val)  {   }
   public TRachunek (int nr)                    {   }
    public String toString ()                    { return null;   }
}
```

```
package rachunek1;

class TZakup
{
  protected int ilosc = 0;
  private TProdukt1 Produkt = null;
  public TProdukt1 getProdukt ()                {   return null;   }
  public void setProdukt (TProdukt1 val)        {   }
  public TZakup (int ailosc, TProdukt1 aProdukt)  {   }
  public String toString ()                     {  return null;   }
}
```

```
package rachunek1;

class TProdukt1
 {
    protected String nazwa = "";
    protected float cena = 0;
    protected TPromocja promocja = null;
    public TPromocja getPromocja ()                                  { return null; }
    public void setPromocja (TPromocja val)                                 {    }
     public TProdukt1 (String anazwa, float acena)                          {    }
    public TProdukt1 (String anazwa, float acena, TPromocja apromocja) {    }
    public String toString ()                                        { return null; }
}
```

```
package rachunek1;

class TProdukt2 extends TProdukt1
{
  public float podatek = 0;
  public TProdukt2 (String anazwa, float acena, float apodatek) {          }
  public TProdukt2 (String anazwa, float acena, float apodatek,TPromocja promocja)
                                                                   {          }
  public String toString ()                                        { return null; }
}
```

```
package rachunek1;

class TPromocja
{
    public float promocja = 0;
    public TPromocja (float apromocja) {    }
    public String toString ()                  {  return null;    }
}
```

```
package rachunek1;

public class TFabryka
 {
   public TFabryka ()                                          {    }
   public TProdukt1 Podaj_produkt (String[] dane)   { return null;    }

}
```

2.3. Model projektowy warstwy biznesowej oparty na diagramie klas i diagramie sekwencji tworzony metodą iteracyjno-rozwojową sterowany realizacją przypadków użycia

2.4. Implementacja warstwy biznesowej tworzona w cyklu iteracyjno-rozwojowym sterowana rozwojem modelu projektowego

Projekt przypadku użycia

„ **Szukanie produktu**”

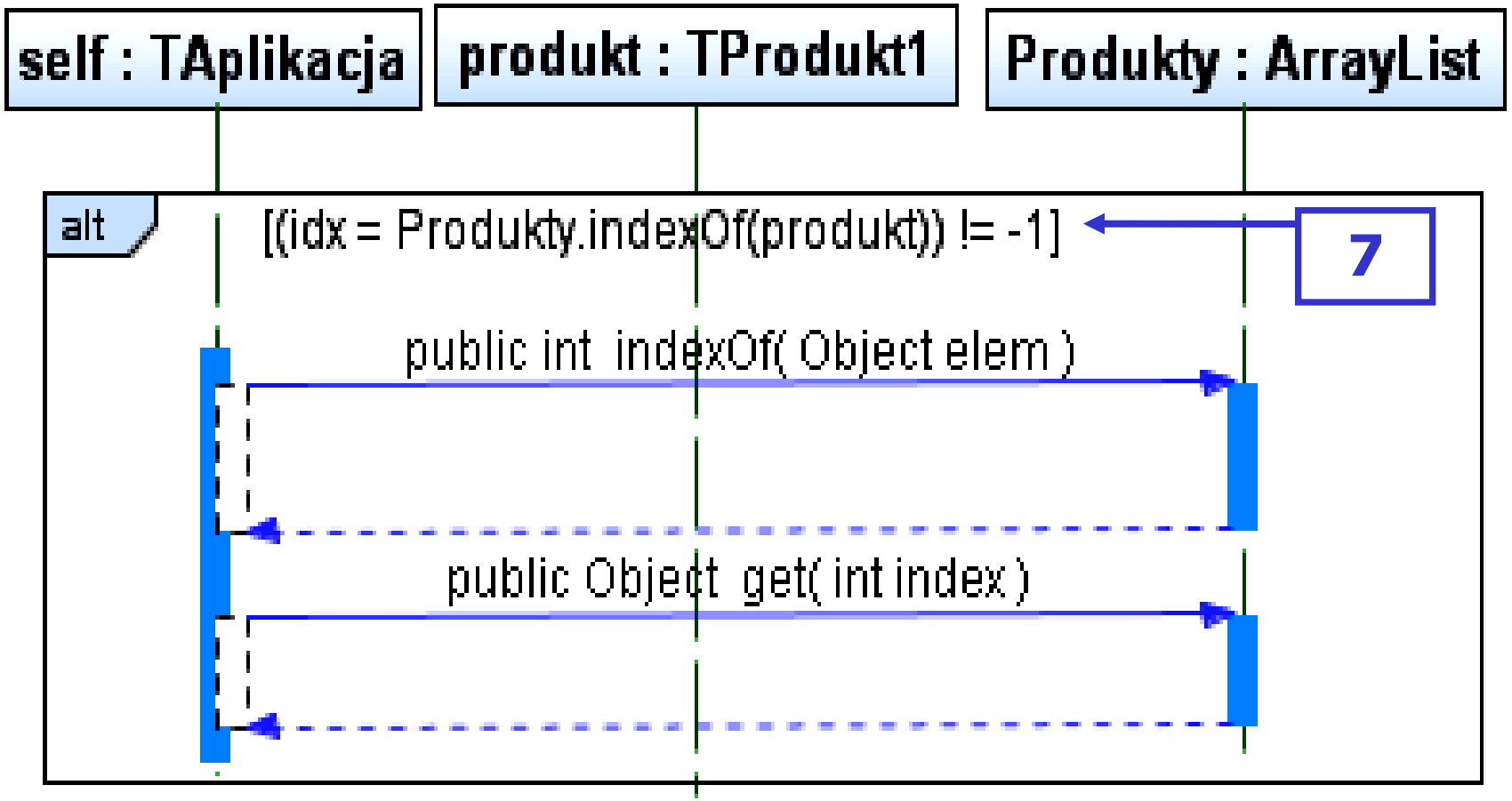
za pomocą diagramu sekwencji i diagramu klas. Diagram klas jest uzupełniany metodami zidentyfikowanymi podczas projektowania scenariusza przypadku użycia za pomocą diagramu sekwencji.

Definiowanie kodu metod realizujących przypadek użycia

na podstawie diagramów sekwencji

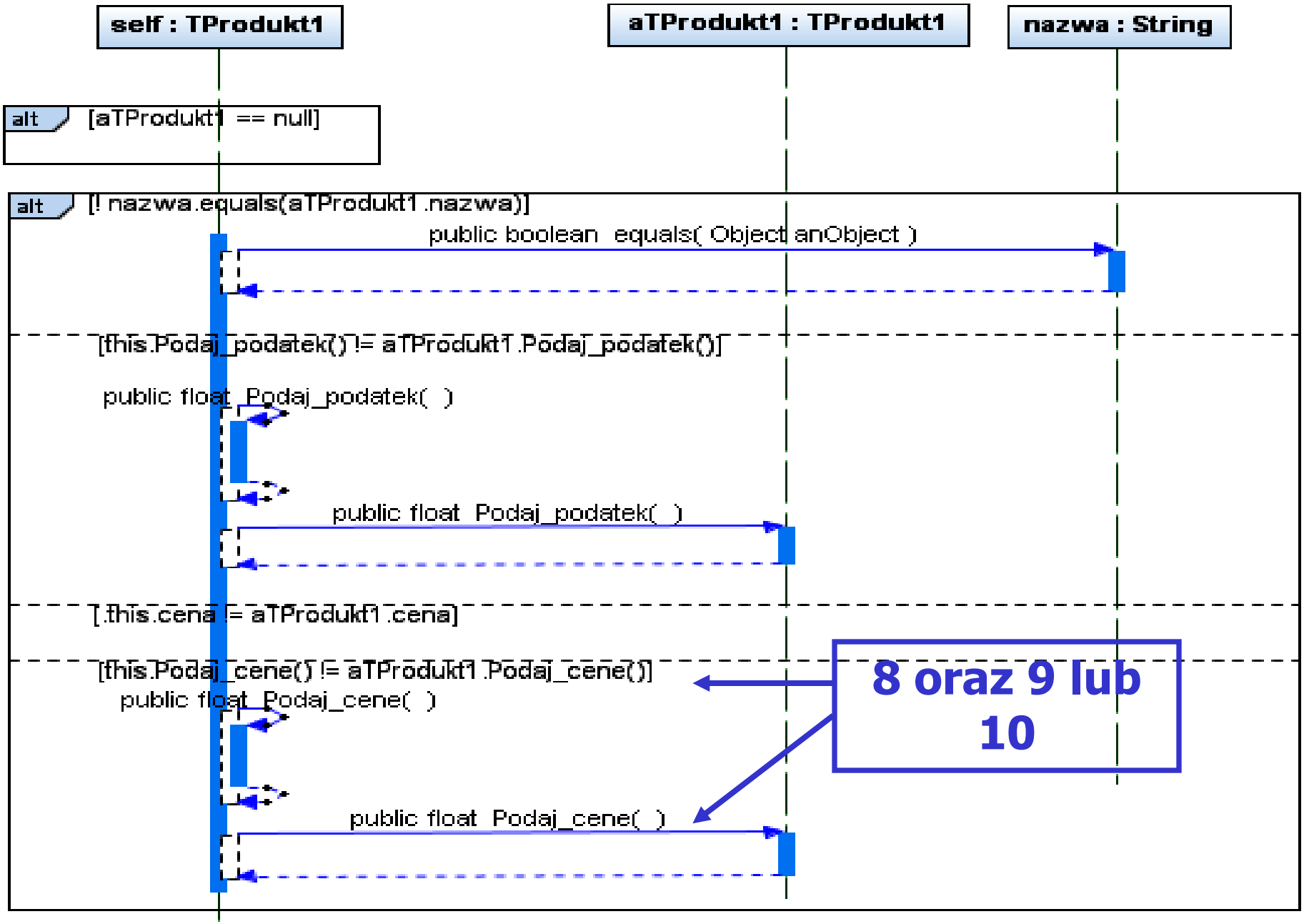
(1) Szukanie produktu

(TProdukt1 TAplikacja::Szukaj_produkt(TProdukt1 produkt))

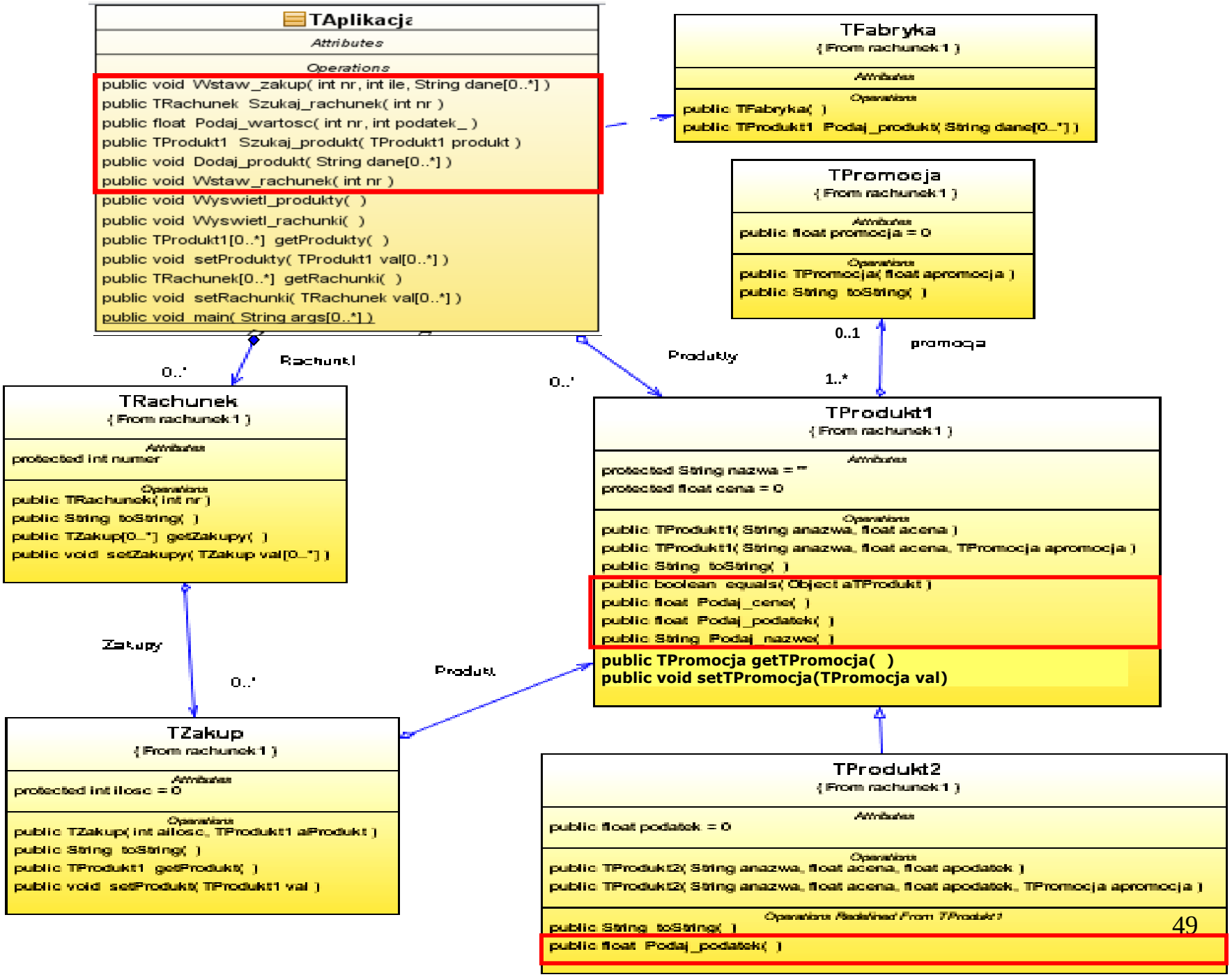


```
//TAplikacja
 private List <TProdukt1> Produkty =
                                        new ArrayList <TProdukt1>();

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

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

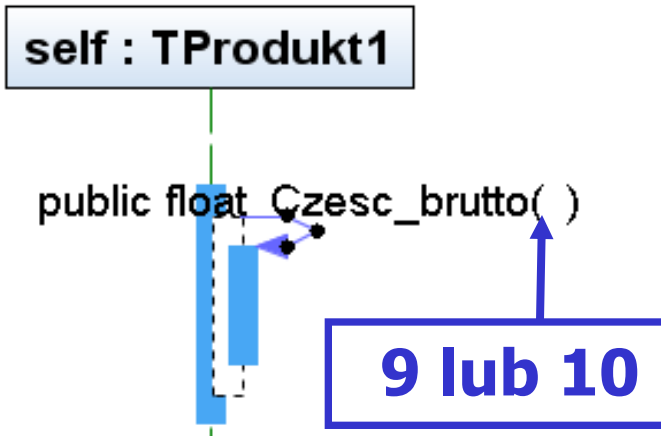


```
//class TProdukt1
public boolean equals (Object aTProdukt)
 {
    TProdukt1 aTProdukt1=(TProdukt1)aTProdukt;
    if ( aTProdukt1 == null ) return false;
    boolean bStatus = true;
    if ( !nazwa.equals(aTProdukt1.nazwa)) bStatus = false;
    else
     if (this.Podaj_podatek()!=aTProdukt1.Podaj_podatek())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


TAplikacja
Attributes
Operations
public void Wstaw_zakup(int nr, int ile, String dane[0..*])
public TRachunek Szukaj_rachunek(int nr)
public float Podaj_wartosc(int nr, int podatek_)
public TProdukt1 Szukaj_produkt(TProdukt1 produkt)
public void Dodaj_produkt(String dane[0..*])
public void Wstaw_rachunek(int nr)
public void Wyswietl_produkty( )
public void Wyswietl_rachunki( )
public TProdukt1[0..*] getProdukty( )
public void setProdukty(TProdukt1 val[0..*])
public TRachunek[0..*] getRachunki( )
public void setRachunki(TRachunek val[0..*])
public void main(String args[0..*])
TFabryka
{ From rachunek1 }
Attributes
Operations
public TFabryka( )
public TProdukt1 Podaj_produkt(String dane[0..*])
TPromocja
{ From rachunek1 }
Attributes
public float promocja = 0
Operations
public TPromocja(float apromocja)
public String toString( )
0..1
promocja
1..*
Rachunki
0..*
Produkty
0..*
TRachunek
{ From rachunek1 }
Attributes
protected int numer
Operations
public TRachunek(int nr)
public String toString( )
public TZakup[0..*] getZakupy( )
public void setZakupy(TZakup val[0..*])
TProdukt1
{ From rachunek1 }
Attributes
protected String nazwa = ""
protected float cena = 0
Operations
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 Podaj_podatek( )
public String Podaj_nazwe( )
public TPromocja getTPromocja( )
public void setTPromocja(TPromocja val)
Zakupy
0..*
Produkt
TZakup
{ From rachunek1 }
Attributes
protected int ilosc = 0
Operations
public TZakup(int ailosc, TProdukt1 aProdukt)
public String toString( )
public TProdukt1 getProdukt( )
public void setProdukt(TProdukt1 val)
TProdukt2
{ From rachunek1 }
Attributes
public 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 Podaj_podatek( )

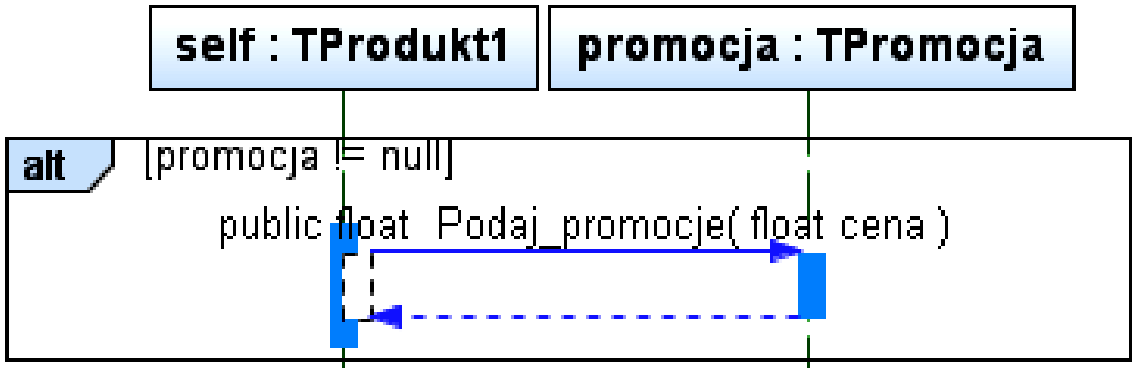
(8)

float TProdukt1::Podaj_cene()



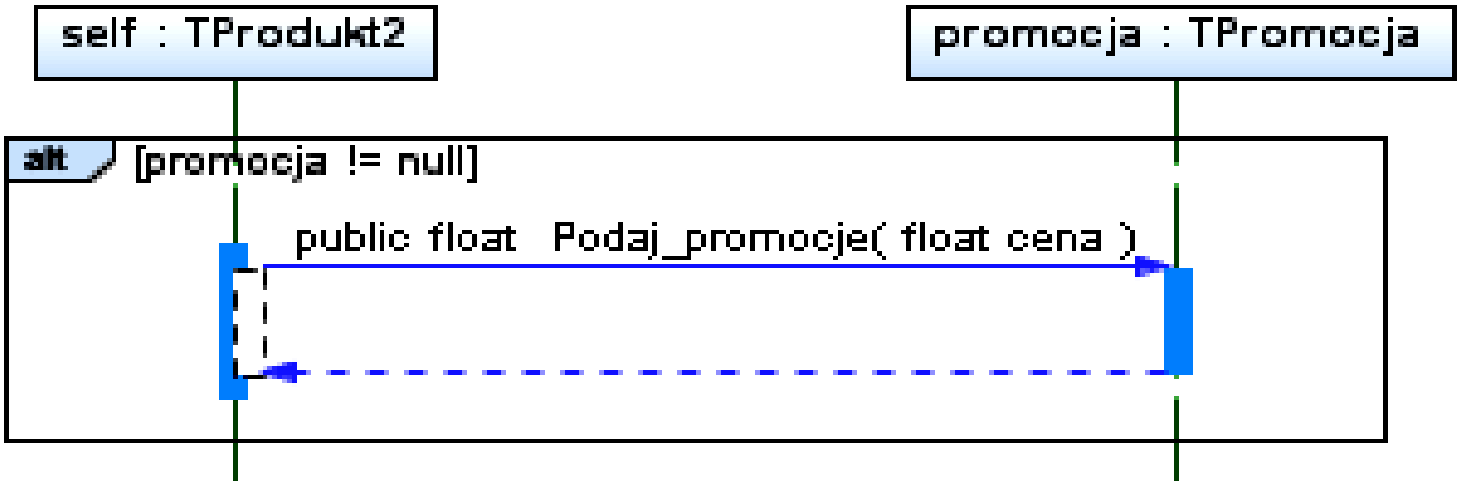
(9)

float TProdukt1::Czesc_brutto()



(10)

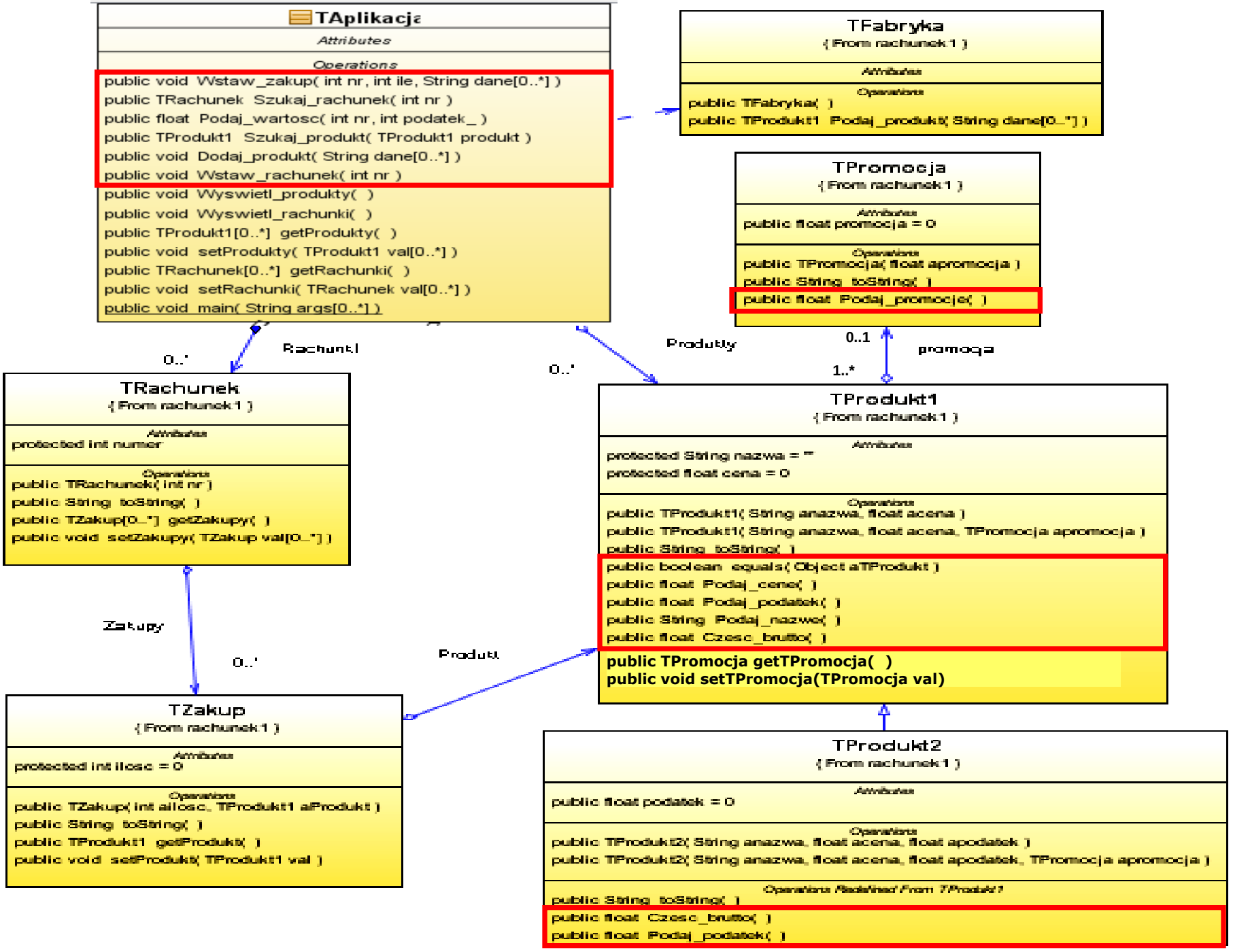
float TProdukt2::Czesc_brutto()



```
//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;
  }
```


TAplikacja
Attributes
Operations
public void Wstaw_zakup(int nr, int ile, String dane[0..*])
public TRachunek Szukaj_rachunek(int nr)
public float Podaj_wartosc(int nr, int podatek_)
public TProdukt1 Szukaj_produkt(TProdukt1 produkt)
public void Dodaj_produkt(String dane[0..*])
public void Wstaw_rachunek(int nr)
public void Wyswietl_produkty( )
public void Wyswietl_rachunki( )
public TProdukt1[0..*] getProdukty( )
public void setProdukty(TProdukt1 val[0..*])
public TRachunek[0..*] getRachunki( )
public void setRachunki(TRachunek val[0..*])
public void main(String args[0..*])
TFabryka
{ From rachunek1 }
Attributes
Operations
public TFabryka( )
public TProdukt1 Podaj_produkt(String dane[0..*])
TPromocja
{ From rachunek1 }
Attributes
public float promocja = 0
Operations
public TPromocja(float apromocja)
public String toString( )
public float Podaj_promocje( )
Rachunki
0..*
Produkty
0..*
0..1
promocja
1..*
TRachunek
{ From rachunek1 }
Attributes
protected int numer
Operations
public TRachunek(int nr)
public String toString( )
public TZakup[0..*] getZakupy( )
public void setZakupy(TZakup val[0..*])
TProdukt1
{ From rachunek1 }
Attributes
protected String nazwa = ""
protected float cena = 0
Operations
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 Podaj_podatek( )
public String Podaj_nazwe( )
public float Czesc_brutto( )
public TPromocja getTPromocja( )
public void setTPromocja(TPromocja val)
Zakupy
0..*
Produkt
TZakup
{ From rachunek1 }
Attributes
protected int ilosc = 0
Operations
public TZakup(int ailosc, TProdukt1 aProdukt)
public String toString( )
public TProdukt1 getProdukt( )
public void setProdukt(TProdukt1 val)
TProdukt2
{ From rachunek1 }
Attributes
public 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( )

Projekt przypadku użycia

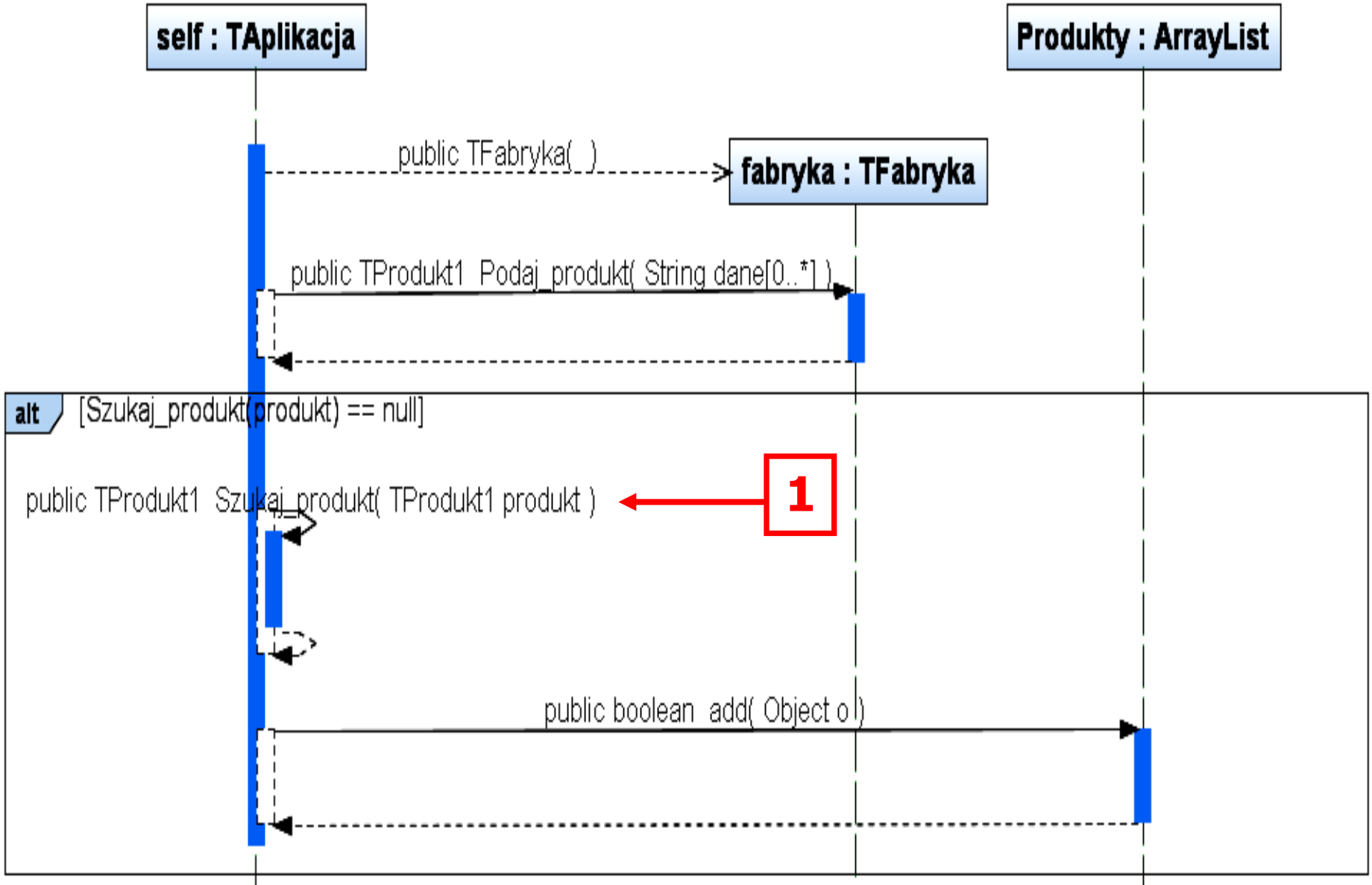
„ **Wstawianie nowego produktu**”

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

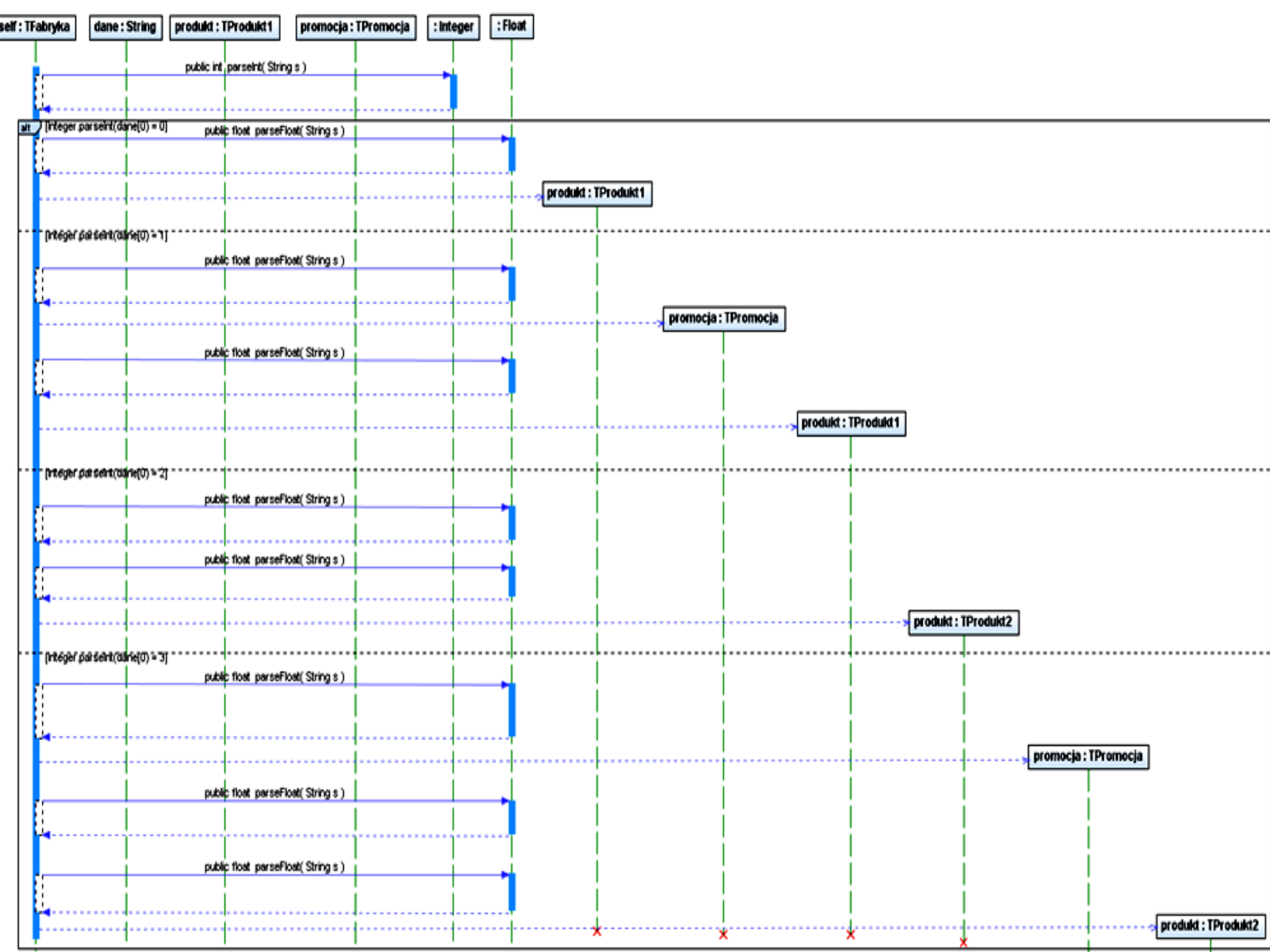
Definiowanie kodu metod realizujących przypadek użycia
na podstawie diagramów sekwencji

(2) 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);
  }
```


self : TFabryka
dane : String
produkt : TProdukt1
promocja : TPromocja
: Integer
: Float
public int parseInt(String s)
alt
[Integer.parseInt(dane[0]) = 0]
public float parseFloat(String s)
produkt : TProdukt1
[Integer.parseInt(dane[0]) = 1]
public float parseFloat(String s)
promocja : TPromocja
public float parseFloat(String s)
produkt : TProdukt1
[Integer.parseInt(dane[0]) = 2]
public float parseFloat(String s)
public float parseFloat(String s)
produkt : TProdukt2
[Integer.parseInt(dane[0]) = 3]
public float parseFloat(String s)
promocja : TPromocja
public float parseFloat(String s)
public float parseFloat(String s)
produkt : TProdukt2

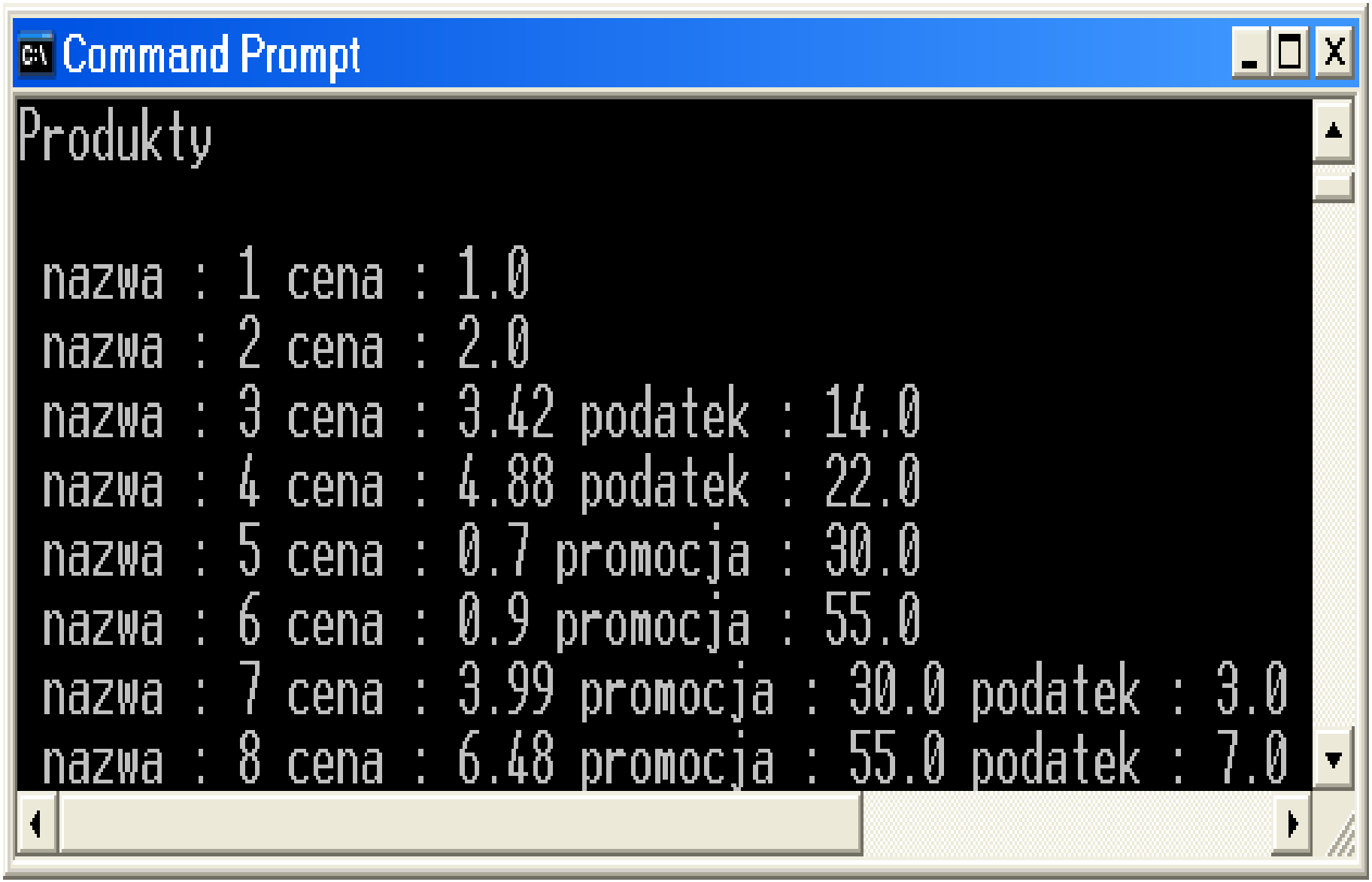
```
public class TFabryka   //Decyzje na poziomie tworzenia kodu
 { public TFabryka() { }
   public TProdukt1 Podaj_produkt(String dane[])
   { TProdukt1 produkt = null; TPromocja promocja;
     switch ( Integer.parseInt(dane[0]) )
     {case 0: produkt= new TProdukt1(dane[1], Float.parseFloat(dane[2]));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

```
//TAplikacja
public void Wyswietl_produkty()  {
        TProdukt1 produkt;
        Iterator <TProdukt1> it = Produkty.iterator();
       while (it.hasNext())  {
            produkt = it.next();
            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

Projekt przypadku użycia

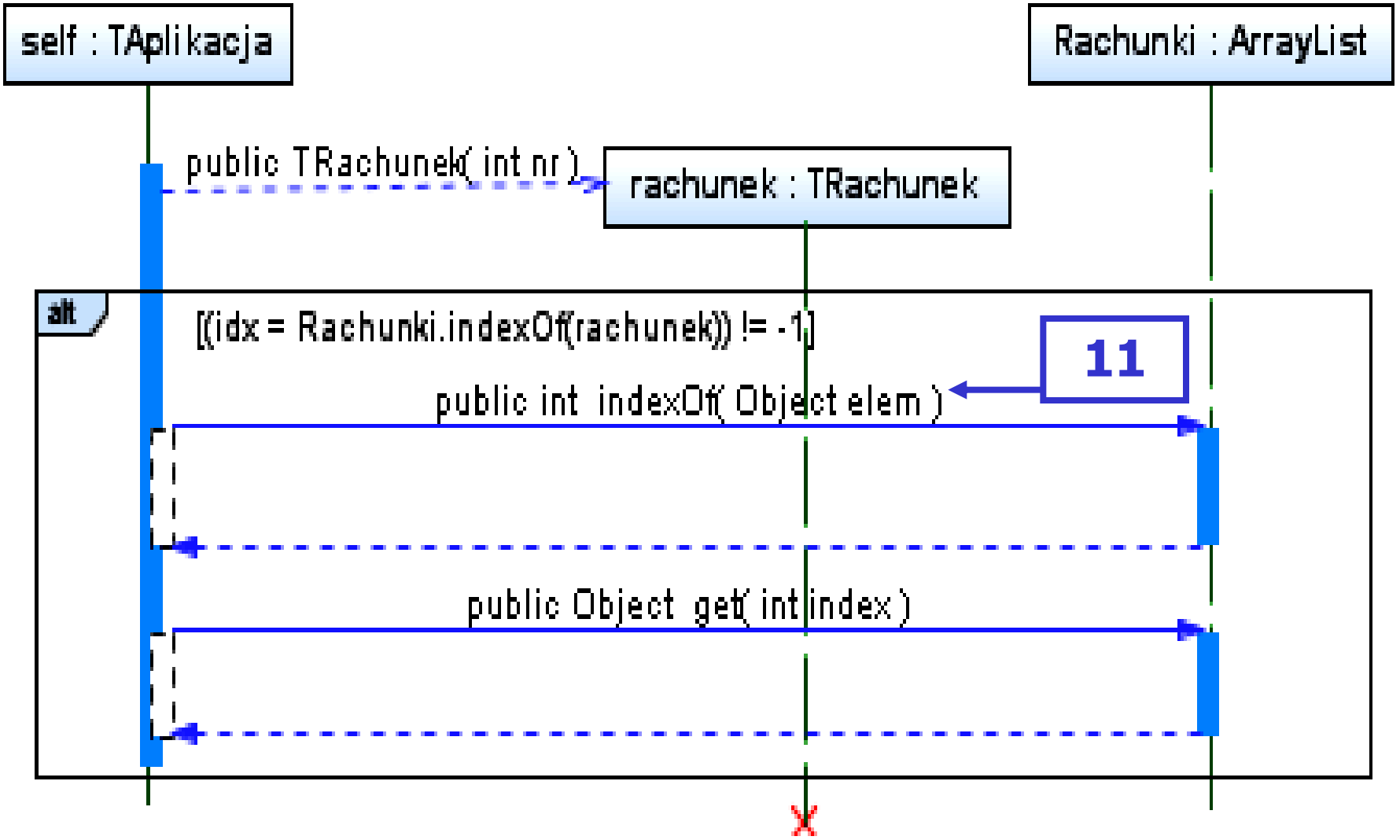
„ Szukanie rachunku ”

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

Definiowanie kodu metod realizujących przypadek użycia
na podstawie diagramów sekwencji

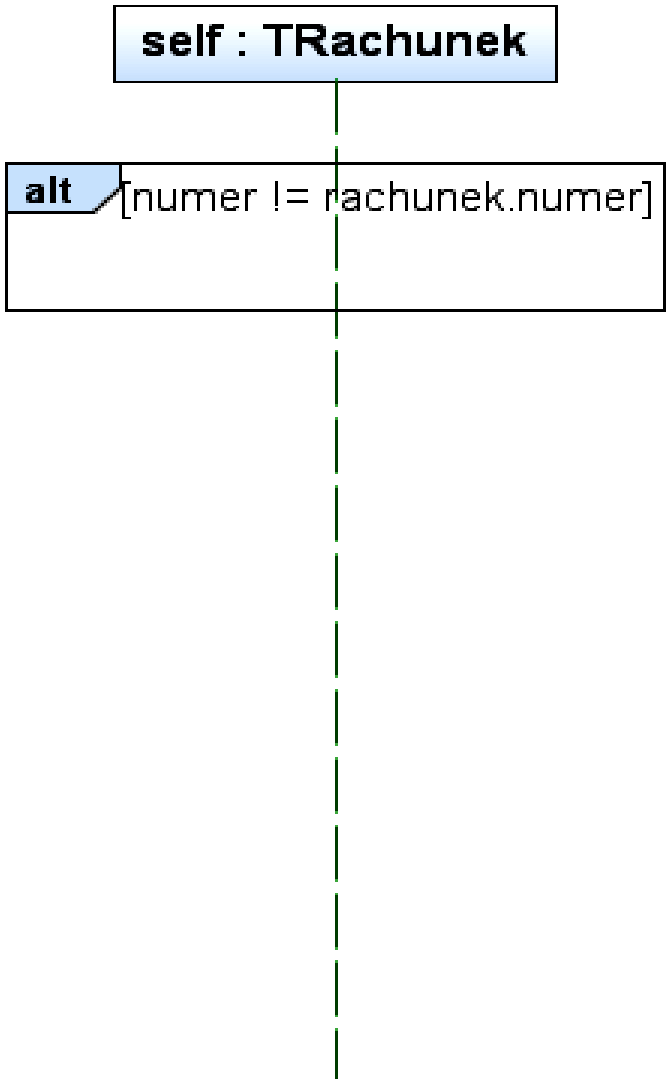
(3) Szukanie rachunku

(TRachunek TAplikacja::Szukaj_rachunek(int nr))

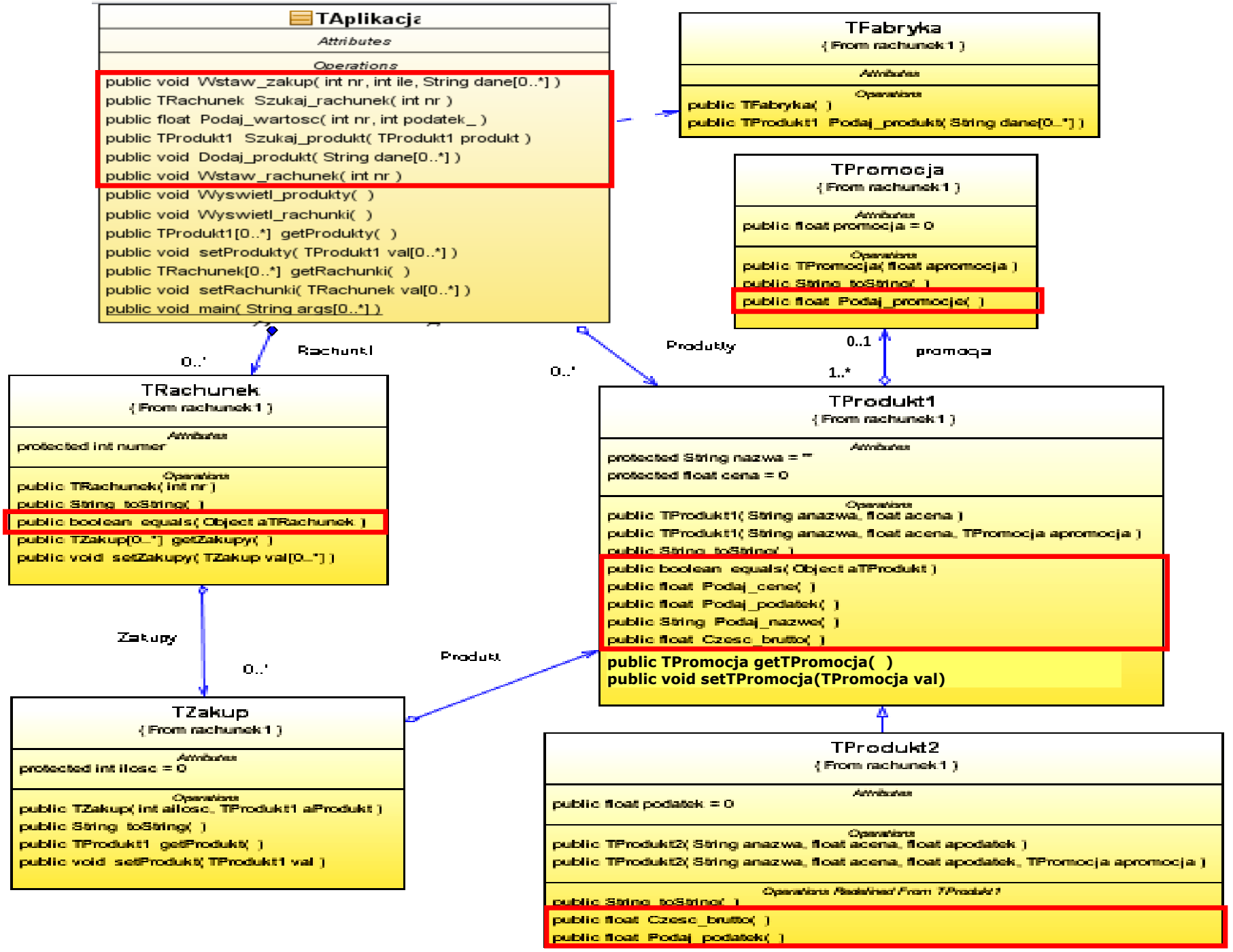


```
//TAplikacja
private List <TRachunek> Rachunki  =
                              new ArrayList <TRachunek>();

 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;
  }
```

(11) boolean TRachunek::equals(Object rachunek)



```
//TRachunek

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


TAplikacja
Attributes
Operations
public void Wstaw_zakup(int nr, int ile, String dane[0..*])
public TRachunek Szukaj_rachunek(int nr)
public float Podaj_wartosc(int nr, int podatek_)
public TProdukt1 Szukaj_produkt(TProdukt1 produkt)
public void Dodaj_produkt(String dane[0..*])
public void Wstaw_rachunek(int nr)
public void Wyswietl_produkty( )
public void Wyswietl_rachunki( )
public TProdukt1[0..*] getProdukty( )
public void setProdukty(TProdukt1 val[0..*])
public TRachunek[0..*] getRachunki( )
public void setRachunki(TRachunek val[0..*])
public void main(String args[0..*])
TFabryka
{ From rachunek1 }
Attributes
Operations
public TFabryka( )
public TProdukt1 Podaj_produkt(String dane[0..*])
TPromocja
{ From rachunek1 }
Attributes
public float promocja = 0
Operations
public TPromocja(float apromocja)
public String toString( )
public float Podaj_promocje( )
Rachunki
0..*
Produkty
0..*
0..1
promocja
1..*
TRachunek
{ From rachunek1 }
Attributes
protected int numer
Operations
public TRachunek(int nr)
public String toString( )
public boolean equals(Object aTRachunek)
public TZakup[0..*] getZakupy( )
public void setZakupy(TZakup val[0..*])
TProdukt1
{ From rachunek1 }
Attributes
protected String nazwa = ""
protected float cena = 0
Operations
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 Podaj_podatek( )
public String Podaj_nazwe( )
public float Czesc_brutto( )
public TPromocja getTPromocja( )
public void setTPromocja(TPromocja val)
Zakupy
0..*
Produkt
TZakup
{ From rachunek1 }
Attributes
protected int ilosc = 0
Operations
public TZakup(int ailosc, TProdukt1 aProdukt)
public String toString( )
public TProdukt1 getProdukt( )
public void setProdukt(TProdukt1 val)
TProdukt2
{ From rachunek1 }
Attributes
public 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( )

Projekt przypadku użycia

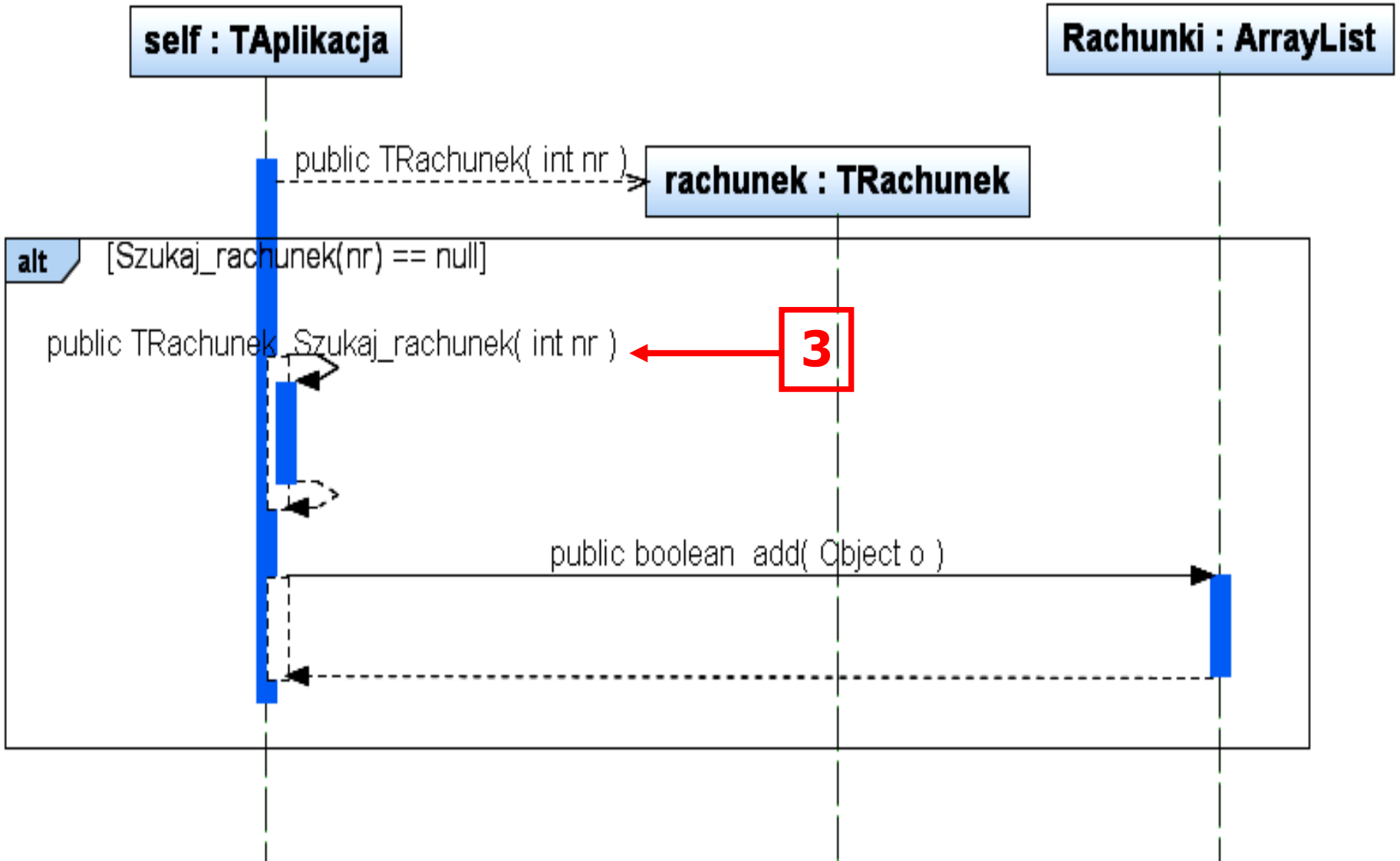
„ Wstawianie nowego rachunku”

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

Definiowanie kodu metod realizujących przypadek użycia
na podstawie diagramów sekwencji

(4) Wstawianie nowego rachunku

(void TAplikacja::Wstaw_rachunek(int nr))



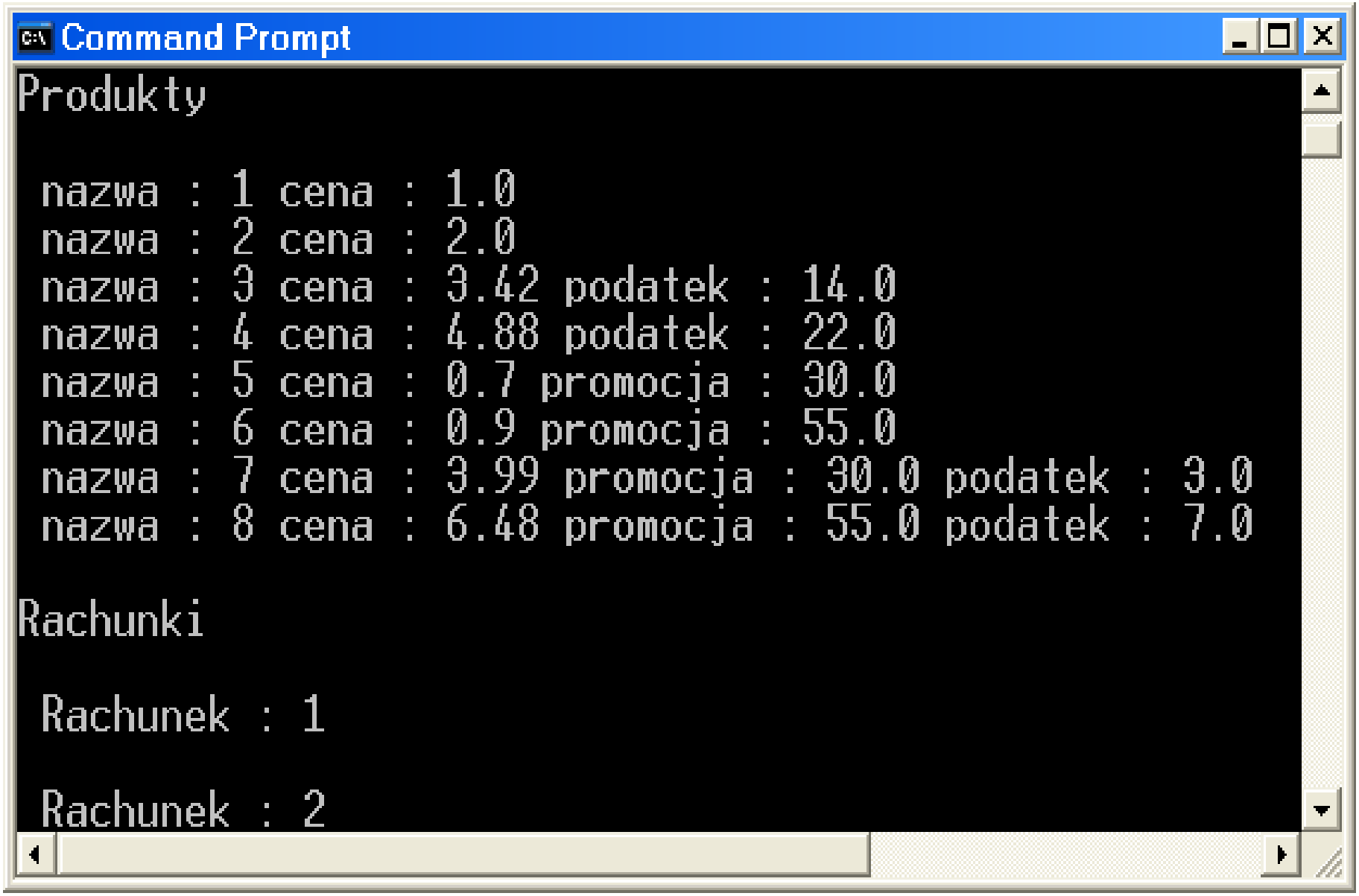
```
//TAplikacja

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);
  }
```

```
//Decyzje na poziomie tworzenia kodu
//TAplikacja
public void Wyswietl_rachunki() {
        TRachunek rachunek;
        Iterator <TRachunek> it = Rachunki.iterator();
        while (it.hasNext()) {
            rachunek = it.next();
            System.out.println(rachunek.toString()); }
    }
```

```
//TRachunek
public String toString() {
 TZakup z;
 StringBuilder sb = new StringBuilder();
 sb.append(" Rachunek : ");
 sb.append(numer).append("\n");
 Iterator<TZakup> it = Zakupy.iterator();
 while (it.hasNext()) {
   z = it.next();
   sb.append(z.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 pom;
        if ((pom = app.Szukaj_rachunek(1)) != null) {
            System.out.println(pom.toString());
        }
        if ((pom = app.Szukaj_rachunek(2)) != null) {
            System.out.println(pom.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

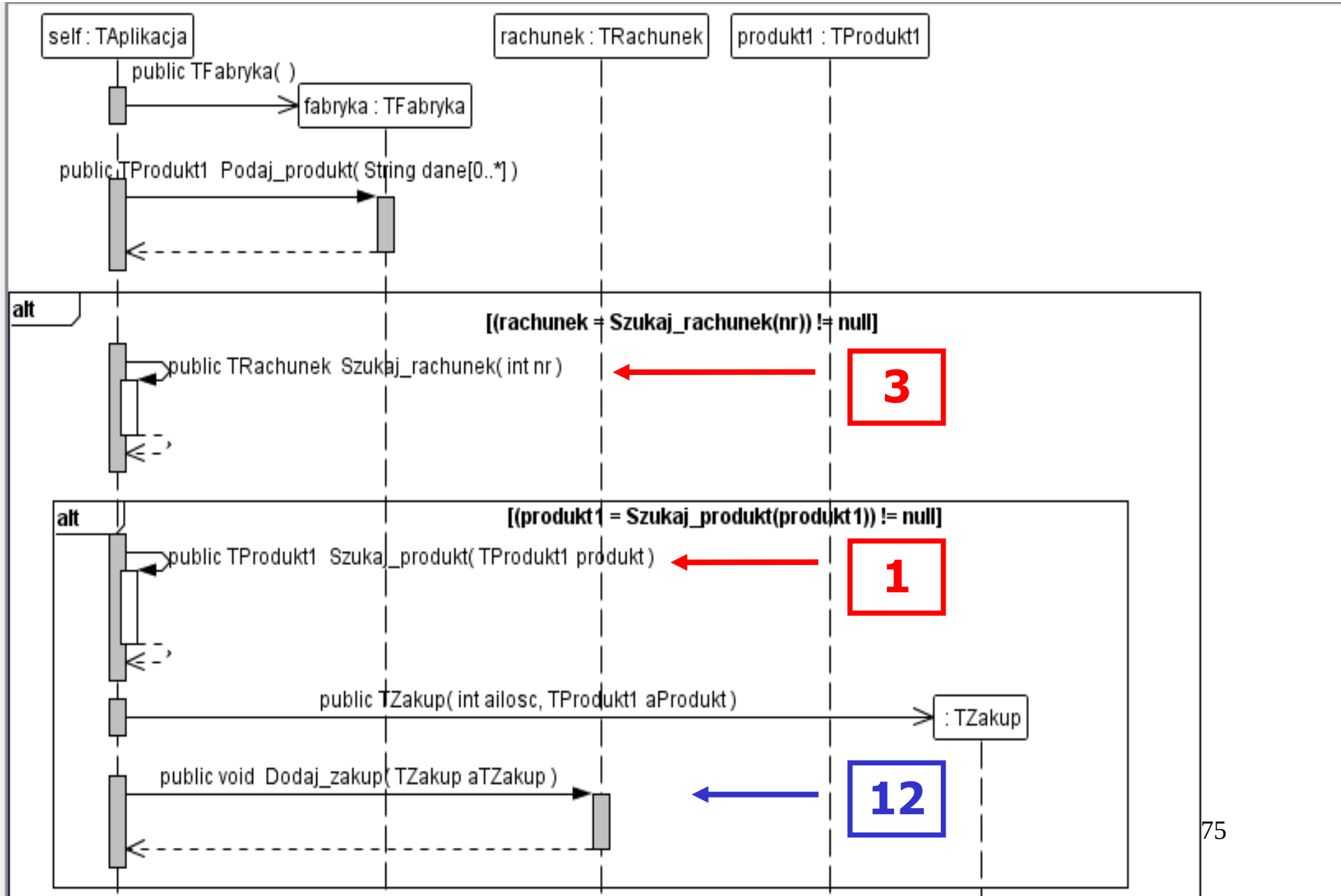
Projekt przypadku użycia

„Wstawianie nowego zakupu”

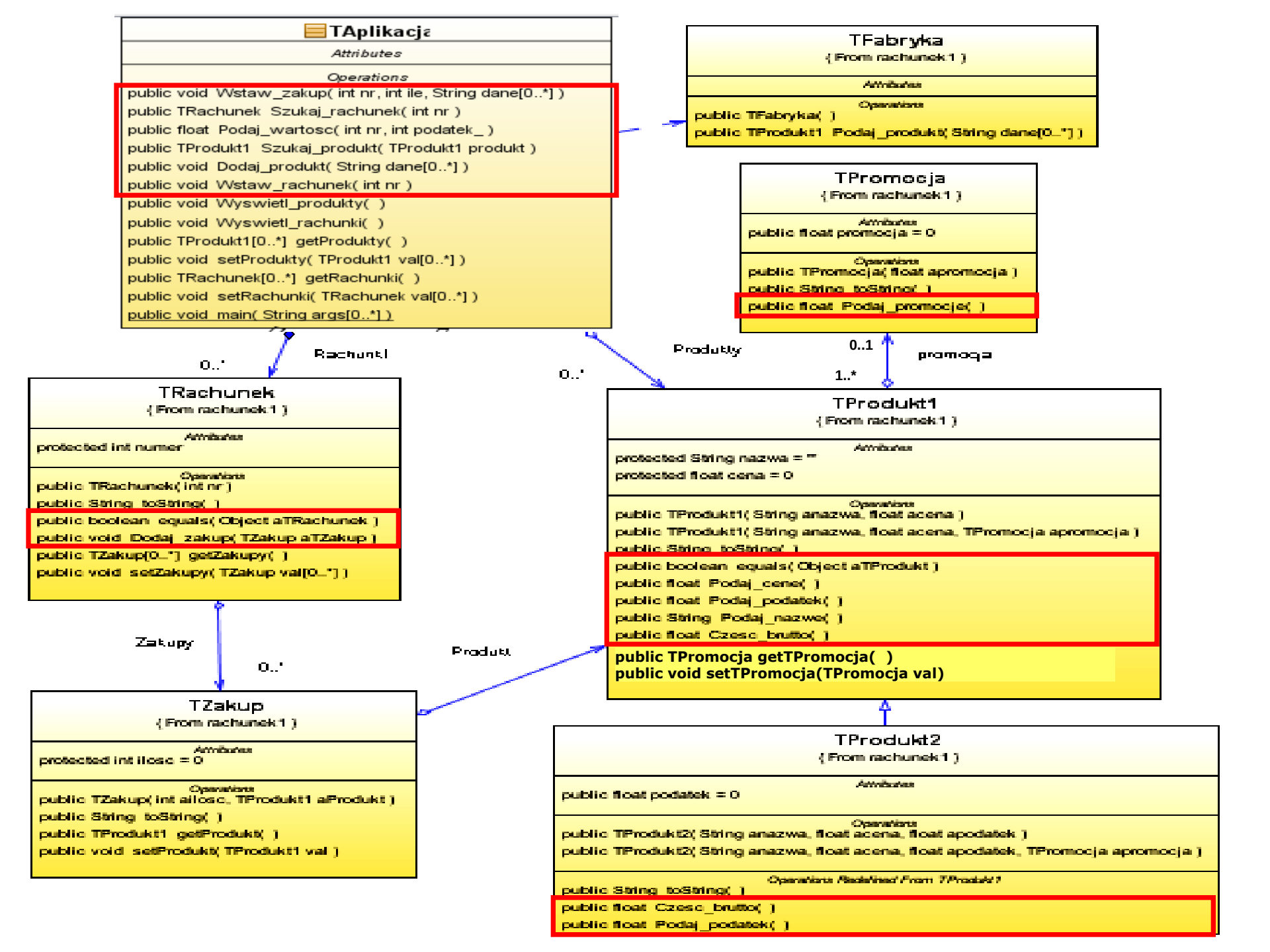
za pomocą diagramu sekwencji i diagramu klas. Diagram klas jest uzupełniany metodami zidentyfikowanymi podczas projektowania scenariusza przypadku użycia za pomocą diagramu sekwencji.

Definiowanie kodu metod realizujących przypadek użycia na podstawie diagramów sekwencji

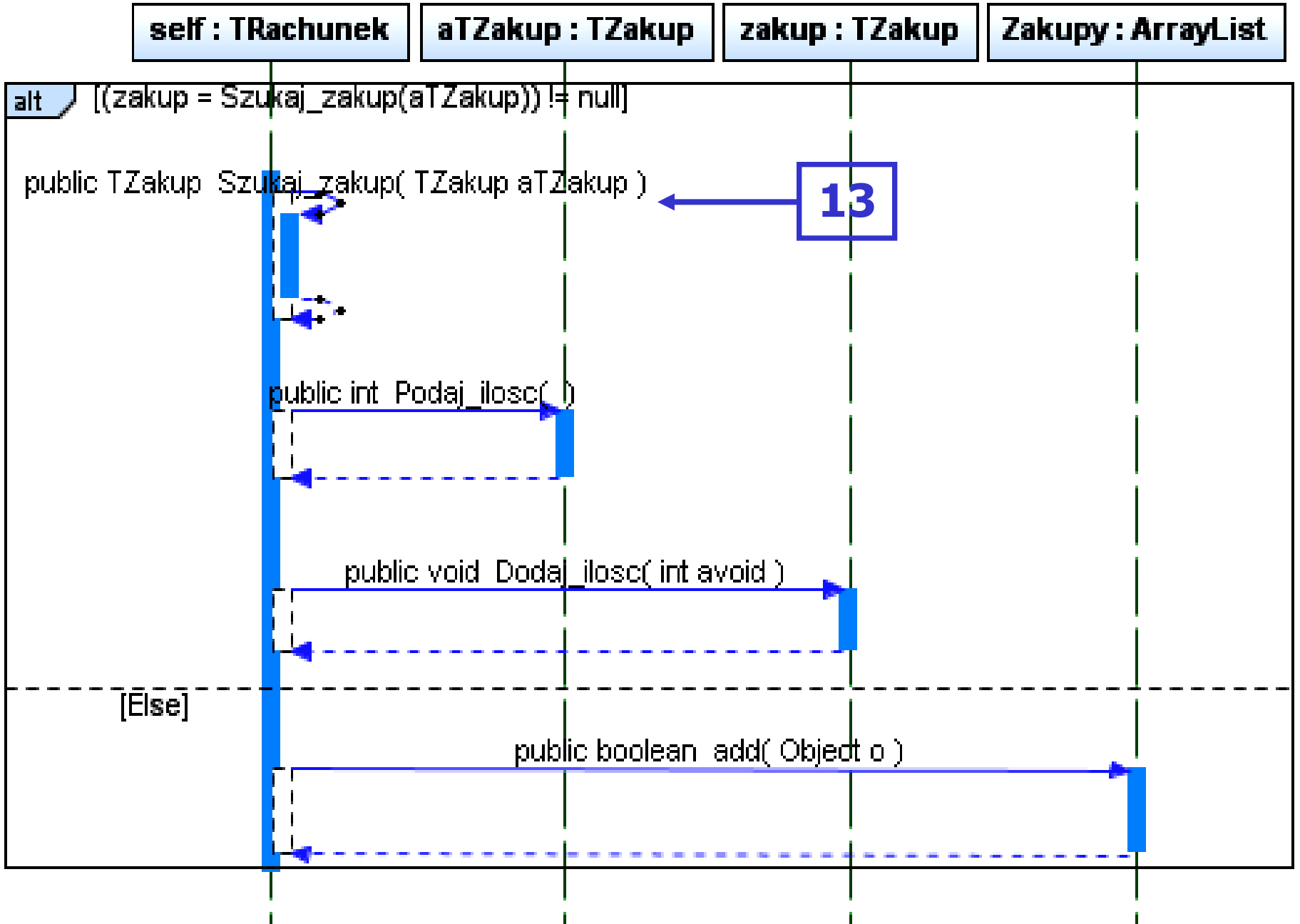
(5) Wstawianie nowego zakupu
(void TAplikacja::Wstaw_zakup (int nr, int ailosc, String dane[]))

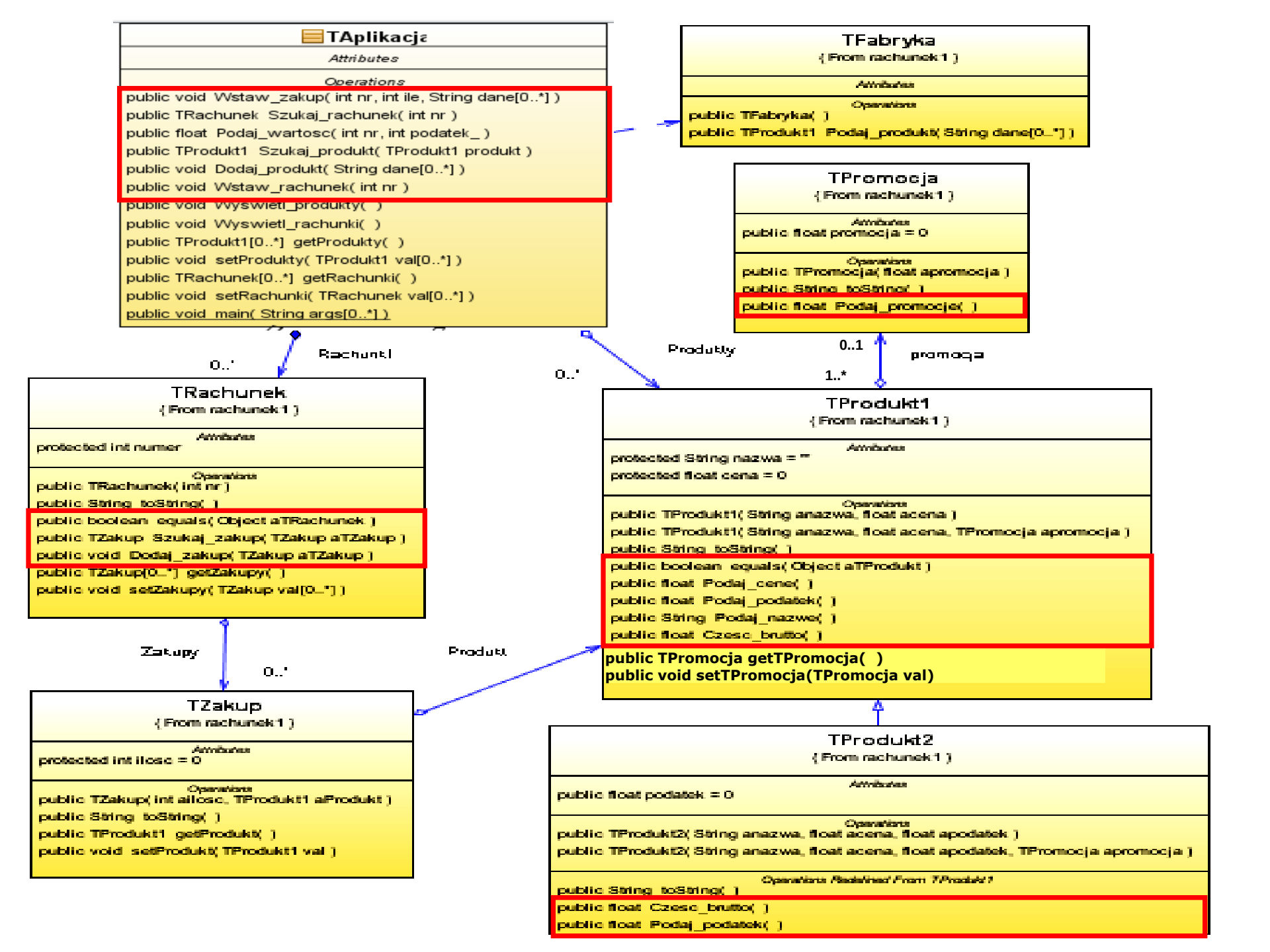


```
//TAplikacja

 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));
  }
```


TAplikacja
Attributes
Operations
public void Wstaw_zakup(int nr, int ile, String dane[0..*])
public TRachunek Szukaj_rachunek(int nr)
public float Podaj_wartosc(int nr, int podatek_)
public TProdukt1 Szukaj_produkt(TProdukt1 produkt)
public void Dodaj_produkt(String dane[0..*])
public void Wstaw_rachunek(int nr)
public void Wyswietl_produkty( )
public void Wyswietl_rachunki( )
public TProdukt1[0..*] getProdukty( )
public void setProdukty(TProdukt1 val[0..*])
public TRachunek[0..*] getRachunki( )
public void setRachunki(TRachunek val[0..*])
public void main(String args[0..*])
TFabryka
{ From rachunek1 }
Attributes
Operations
public TFabryka( )
public TProdukt1 Podaj_produkt(String dane[0..*])
TPromocja
{ From rachunek1 }
Attributes
public float promocja = 0
Operations
public TPromocja(float apromocja)
public String toString( )
public float Podaj_promocje( )
Rachunki
0..*
Produkty
0..*
0..1
promocja
1..*
TRachunek
{ From rachunek1 }
Attributes
protected int numer
Operations
public TRachunek(int nr)
public String toString( )
public boolean equals(Object aTRachunek)
public void Dodaj_zakup(TZakup aTZakup)
public TZakup[0..*] getZakupy( )
public void setZakupy(TZakup val[0..*])
TProdukt1
{ From rachunek1 }
Attributes
protected String nazwa = ""
protected float cena = 0
Operations
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 Podaj_podatek( )
public String Podaj_nazwe( )
public float Czesc_brutto( )
public TPromocja getTPromocja( )
public void setTPromocja(TPromocja val)
Zakupy
0..*
Produkt
TZakup
{ From rachunek1 }
Attributes
protected int ilosc = 0
Operations
public TZakup(int ailosc, TProdukt1 aProdukt)
public String toString( )
public TProdukt1 getProdukt( )
public void setProdukt(TProdukt1 val)
TProdukt2
{ From rachunek1 }
Attributes
public 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( )

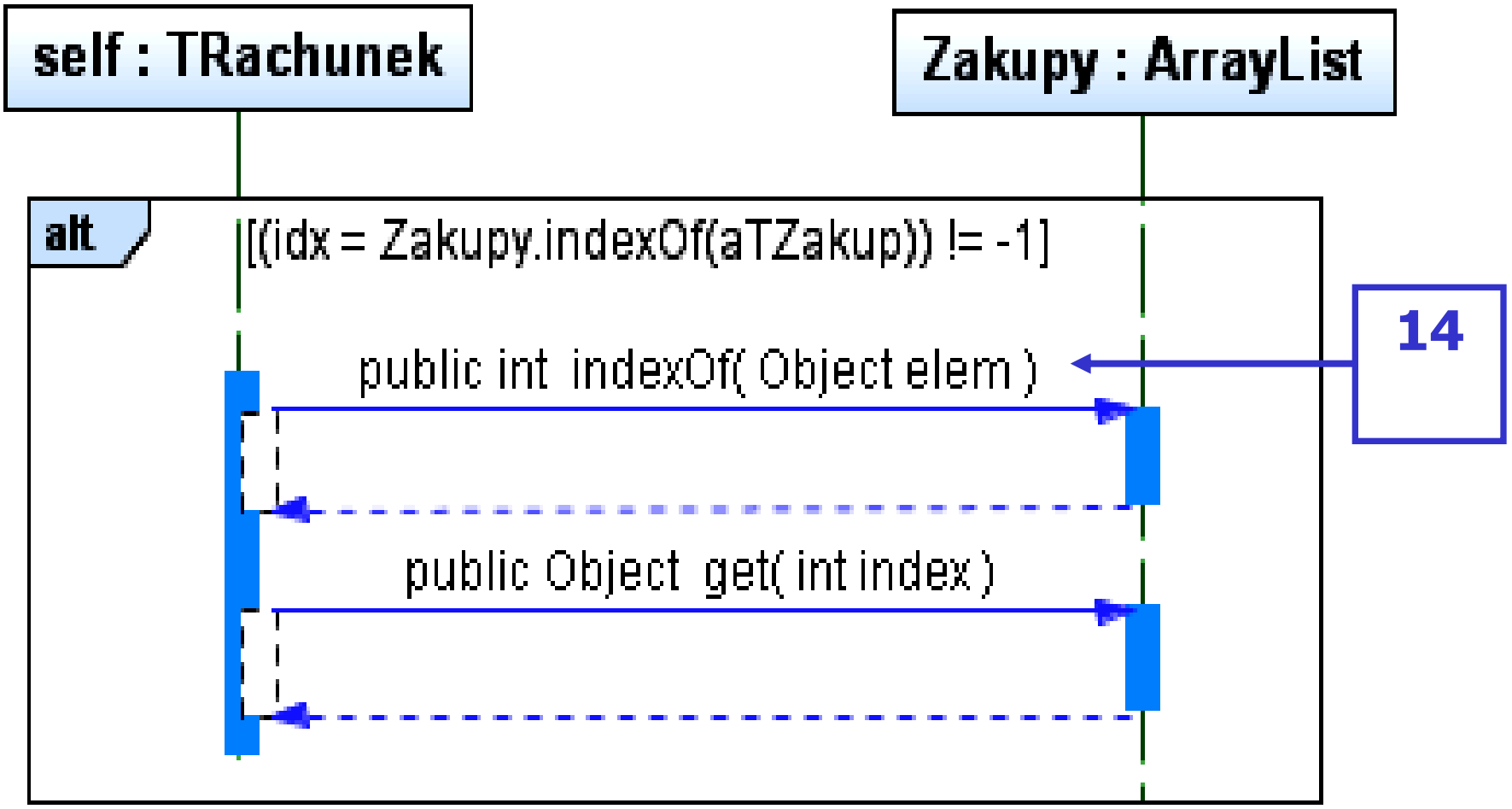
(12) void TRachunek::Dodaj_zakup(TZakup aTZakup)




TAplikacja
Attributes
Operations
public void Wstaw_zakup(int nr, int ile, String dane[0..*])
public TRachunek Szukaj_rachunek(int nr)
public float Podaj_wartosc(int nr, int podatek_)
public TProdukt1 Szukaj_produkt(TProdukt1 produkt)
public void Dodaj_produkt(String dane[0..*])
public void Wstaw_rachunek(int nr)
public void Wyswietl_produkty( )
public void Wyswietl_rachunki( )
public TProdukt1[0..*] getProdukty( )
public void setProdukty(TProdukt1 val[0..*])
public TRachunek[0..*] getRachunki( )
public void setRachunki(TRachunek val[0..*])
public void main(String args[0..*])
TFabryka
{ From rachunek1 }
Attributes
Operations
public TFabryka( )
public TProdukt1 Podaj_produkt(String dane[0..*])
TPromocja
{ From rachunek1 }
Attributes
public float promocja = 0
Operations
public TPromocja(float apromocja)
public String toString( )
public float Podaj_promocje( )
Rachunki
0..*
Produkty
0..*
0..1
promocja
1..*
TRachunek
{ From rachunek1 }
Attributes
protected int numer
Operations
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 TZakup[0..*] getZakupy( )
public void setZakupy(TZakup val[0..*])
TProdukt1
{ From rachunek1 }
Attributes
protected String nazwa = ""
protected float cena = 0
Operations
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 Podaj_podatek( )
public String Podaj_nazwe( )
public float Czesc_brutto( )
public TPromocja getTPromocja( )
public void setTPromocja(TPromocja val)
Zakupy
0..*
Produkt
TZakup
{ From rachunek1 }
Attributes
protected int ilosc = 0
Operations
public TZakup(int ailosc_, TProdukt1 aProdukt)
public String toString( )
public TProdukt1 getProdukt( )
public void setProdukt(TProdukt1 val)
TProdukt2
{ From rachunek1 }
Attributes
public 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( )

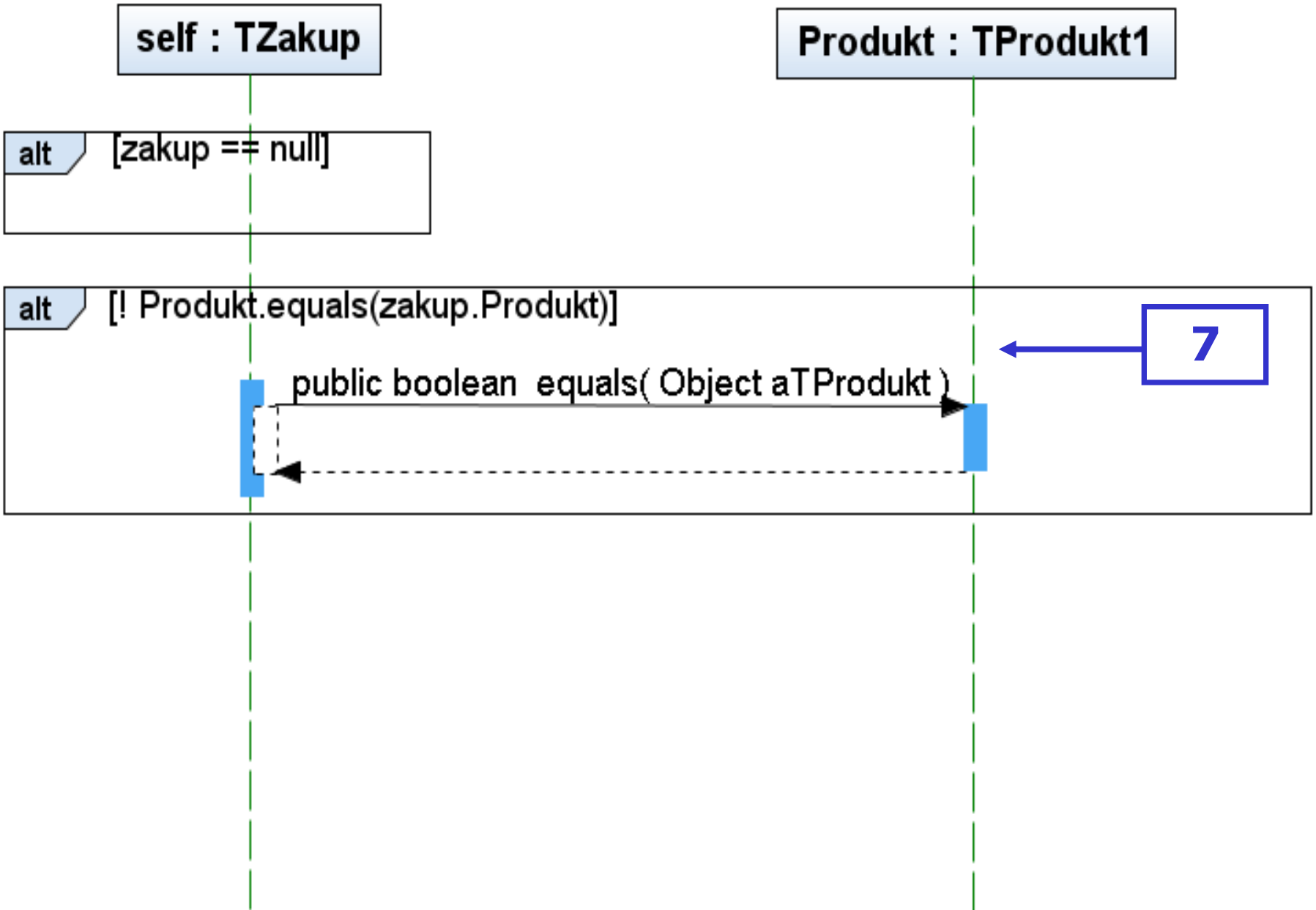
```
//TRachunek

 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);
  }
```

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

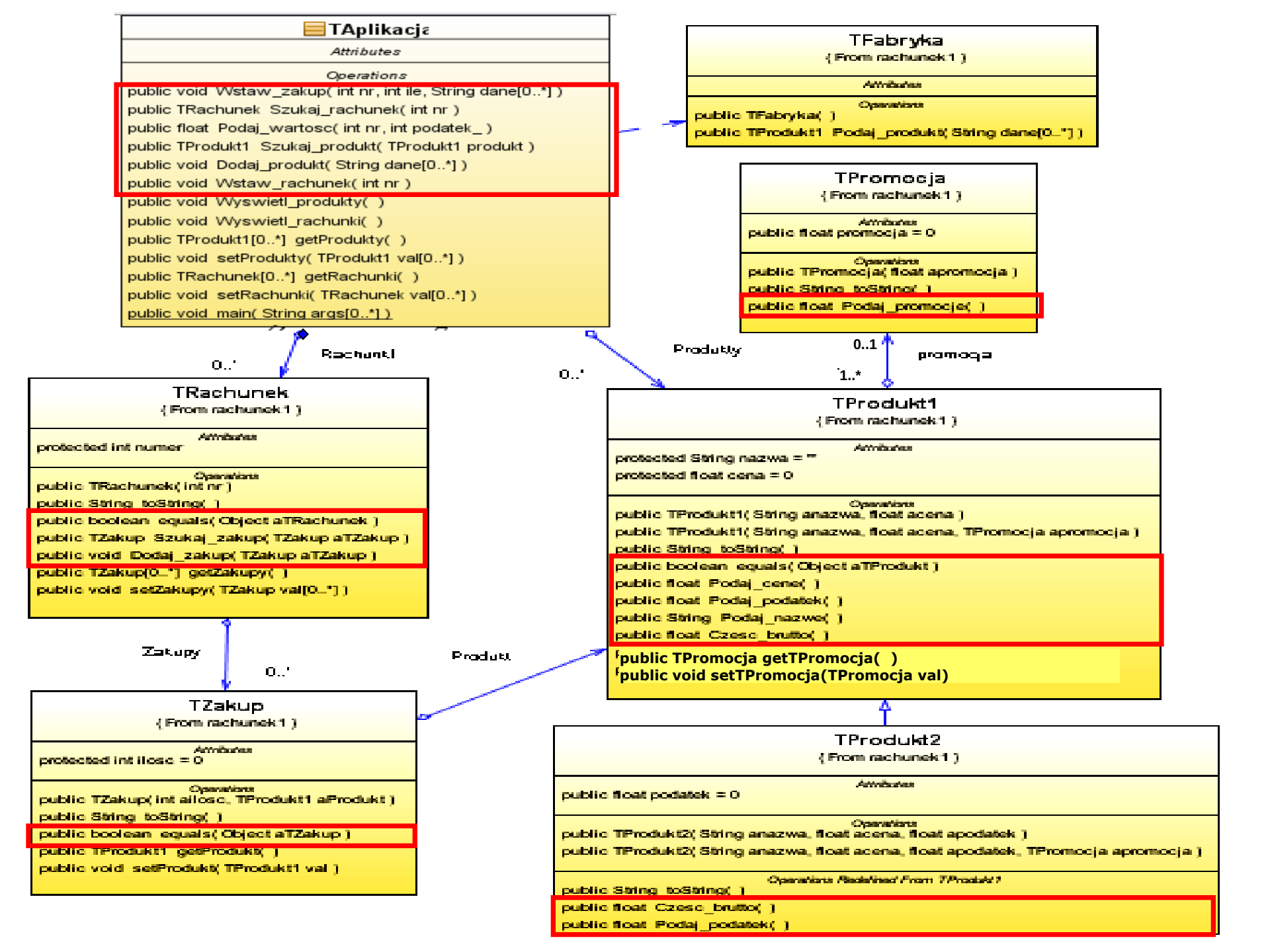


```
//TRachunek

 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;
   }
```

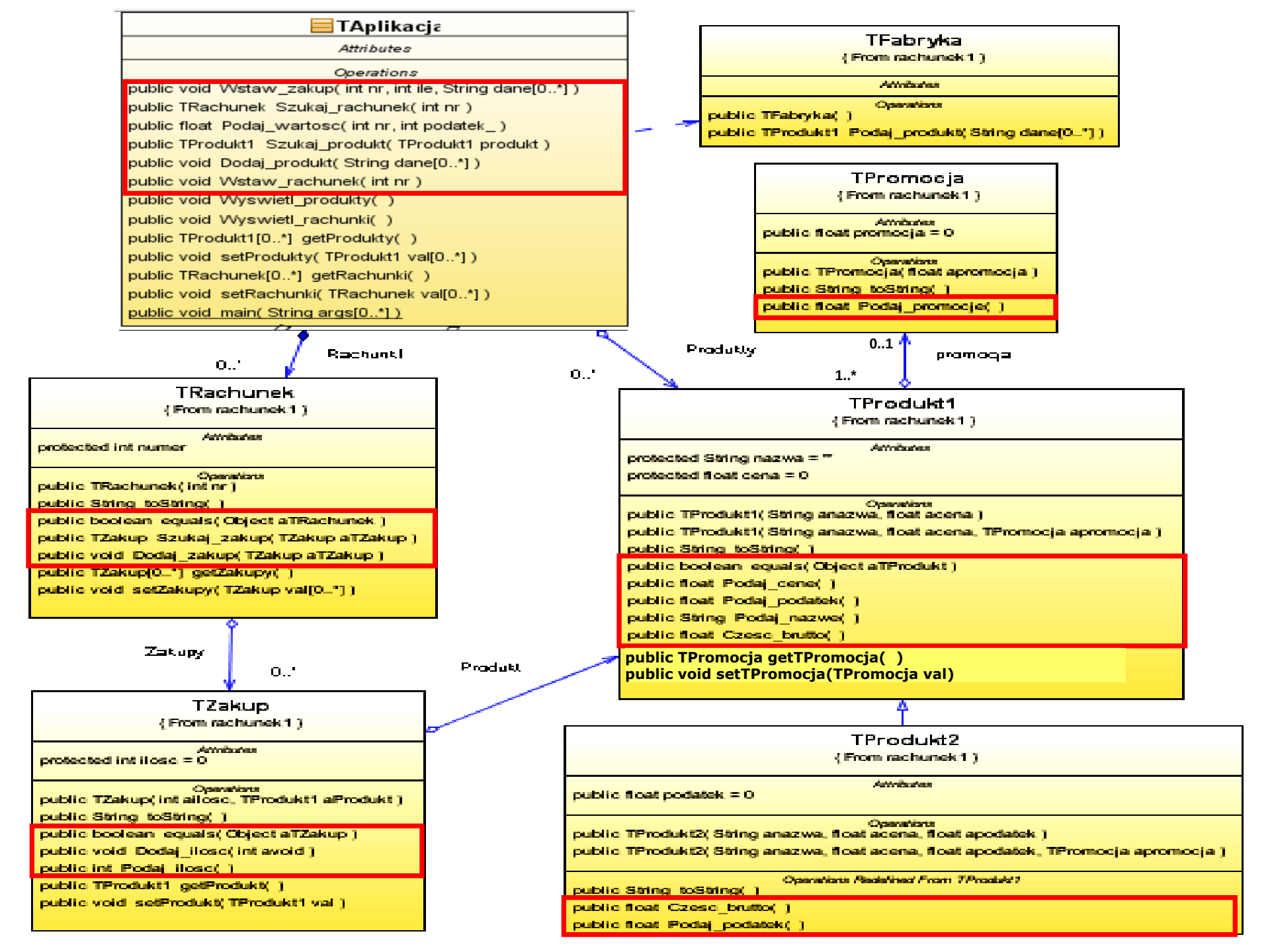
(14) boolean TZakup::equals(Object zakup)



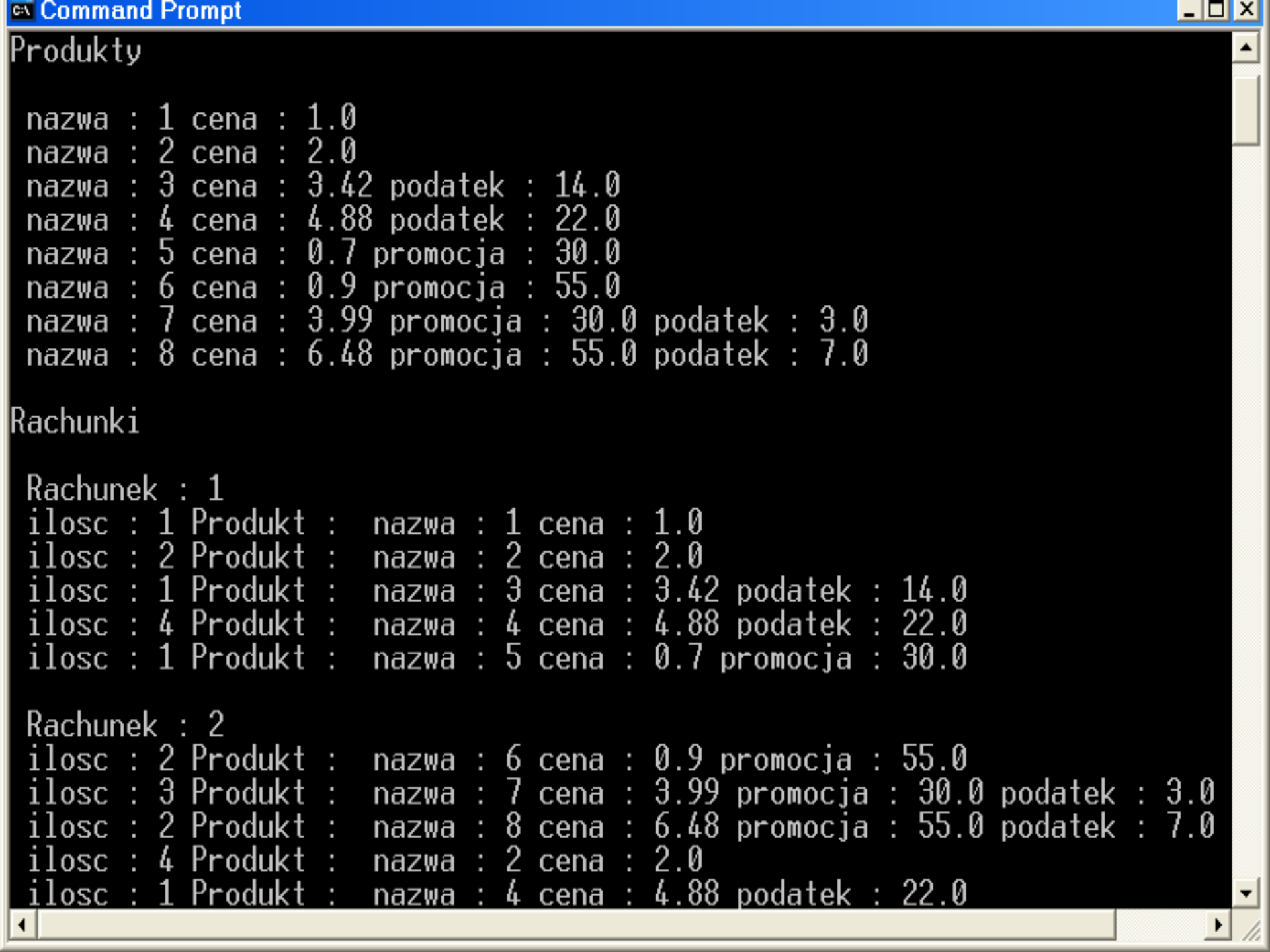
```
//TZakup

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;
     }
```


TAplikacja
Attributes
Operations
public void Wstaw_zakup(int nr, int ile, String dane[0..*])
public TRachunek Szukaj_rachunek(int nr)
public float Podaj_wartosc(int nr, int podatek_)
public TProdukt1 Szukaj_produkt(TProdukt1 produkt)
public void Dodaj_produkt(String dane[0..*])
public void Wstaw_rachunek(int nr)
public void Wyswietl_produkty( )
public void Wyswietl_rachunki( )
public TProdukt1[0..*] getProdukty( )
public void setProdukty(TProdukt1 val[0..*])
public TRachunek[0..*] getRachunki( )
public void setRachunki(TRachunek val[0..*])
public void main(String args[0..*])
TFabryka
{ From rachunek1 }
Attributes
Operations
public TFabryka( )
public TProdukt1 Podaj_produkt(String dane[0..*])
TPromocja
{ From rachunek1 }
Attributes
public float promocja = 0
Operations
public TPromocja(float apromocja)
public String toString( )
public float Podaj_promocje( )
Rachunki
0..*
Produkty
0..*
0..1
promocja
1..*
TRachunek
{ From rachunek1 }
Attributes
protected int numer
Operations
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 TZakup[0..*] getZakupy( )
public void setZakupy(TZakup val[0..*])
TProdukt1
{ From rachunek1 }
Attributes
protected String nazwa = ""
protected float cena = 0
Operations
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 Podaj_podatek( )
public String Podaj_nazwe( )
public float Czesc_brutto( )
public TPromocja getTPromocja( )
public void setTPromocja(TPromocja val)
Zakupy
0..*
Produkt
TZakup
{ From rachunek1 }
Attributes
protected int ilosc = 0
Operations
public TZakup(int ailosc, TProdukt1 aProdukt)
public String toString( )
public boolean equals(Object aTZakup)
public TProdukt1 getProdukt( )
public void setProdukt(TProdukt1 val)
TProdukt2
{ From rachunek1 }
Attributes
public 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( )

```
 //TZakup

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

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


TAplikacja
Attributes
Operations
public void Wstaw_zakup(int nr, int ile, String dane[0..*])
public TRachunek Szukaj_rachunek(int nr)
public float Podaj_wartosc(int nr, int podatek_)
public TProdukt1 Szukaj_produkt(TProdukt1 produkt)
public void Dodaj_produkt(String dane[0..*])
public void Wstaw_rachunek(int nr)
public void Wyswietl_produkty( )
public void Wyswietl_rachunki( )
public TProdukt1[0..*] getProdukty( )
public void setProdukty(TProdukt1 val[0..*])
public TRachunek[0..*] getRachunki( )
public void setRachunki(TRachunek val[0..*])
public void main(String args[0..*])
TFabryka
{ From rachunek1 }
Attributes
Operations
public TFabryka( )
public TProdukt1 Podaj_produkt(String dane[0..*])
TPromocja
{ From rachunek1 }
Attributes
public float promocja = 0
Operations
public TPromocja(float apromocja)
public String toString( )
public float Podaj_promocje( )
Rachunki
0..*
0..*
Produkty
0..1
promocja
1..*
TRachunek
{ From rachunek1 }
Attributes
protected int numer
Operations
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 TZakup[0..*] getZakupy( )
public void setZakupy(TZakup val[0..*])
TProdukt1
{ From rachunek1 }
Attributes
protected String nazwa = ""
protected float cena = 0
Operations
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 Podaj_podatek( )
public String Podaj_nazwe( )
public float Czesc_brutto( )
public TPromocja getTPromocja( )
public void setTPromocja(TPromocja val)
Zakupy
0..*
Produkt
TZakup
{ From rachunek1 }
Attributes
protected int ilosc = 0
Operations
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 TProdukt1 getProdukt( )
public void setProdukt(TProdukt1 val)
TProdukt2
{ From rachunek1 }
Attributes
public 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 pom;
        if ((pom = app.Szukaj_rachunek(1)) != null) {
            System.out.println(pom.toString()); }
        if ((pom = app.Szukaj_rachunek(2)) != null) {
            System.out.println(pom.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

Projekt przypadku użycia

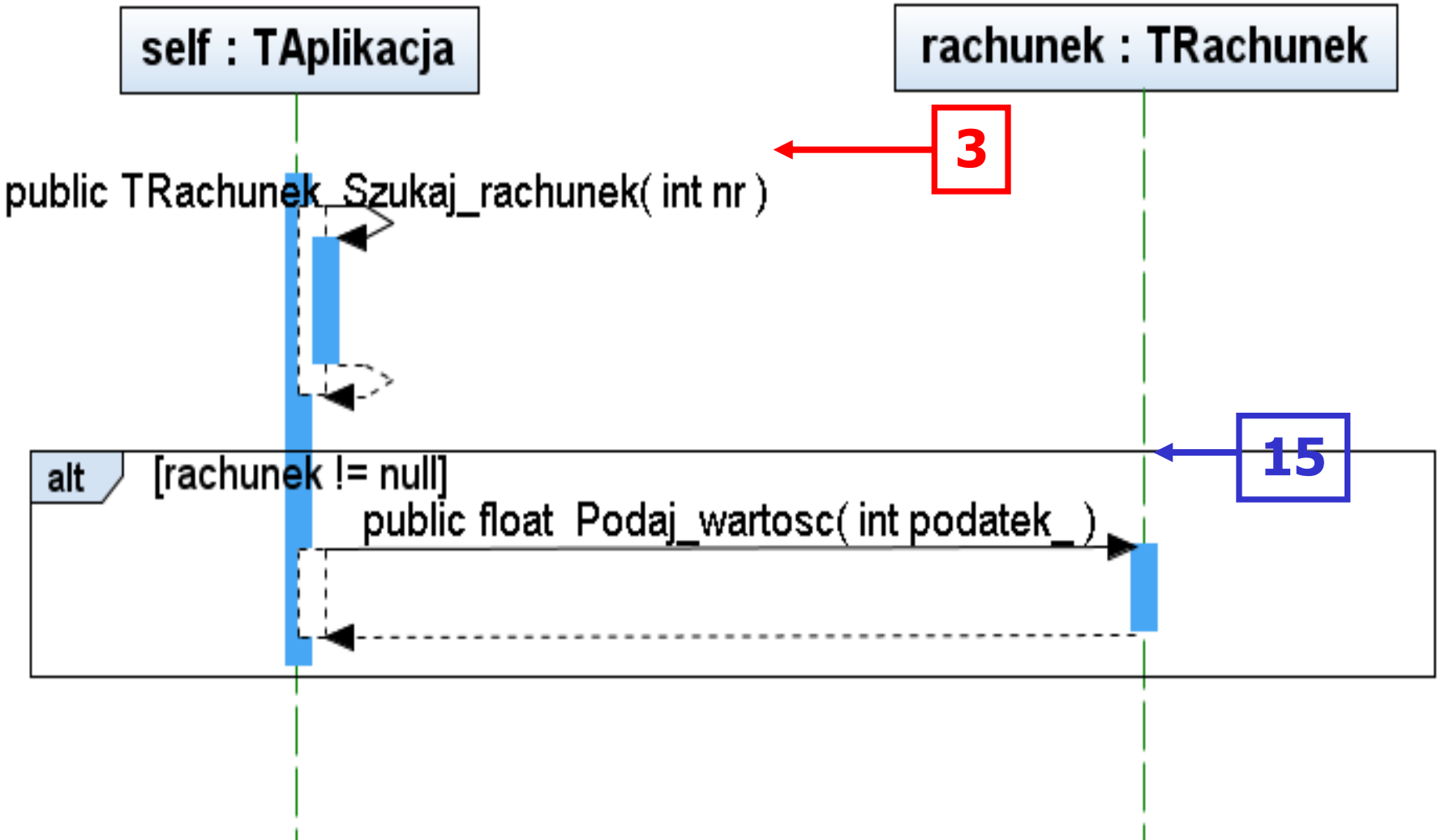
„Obliczanie wartości rachunku”

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

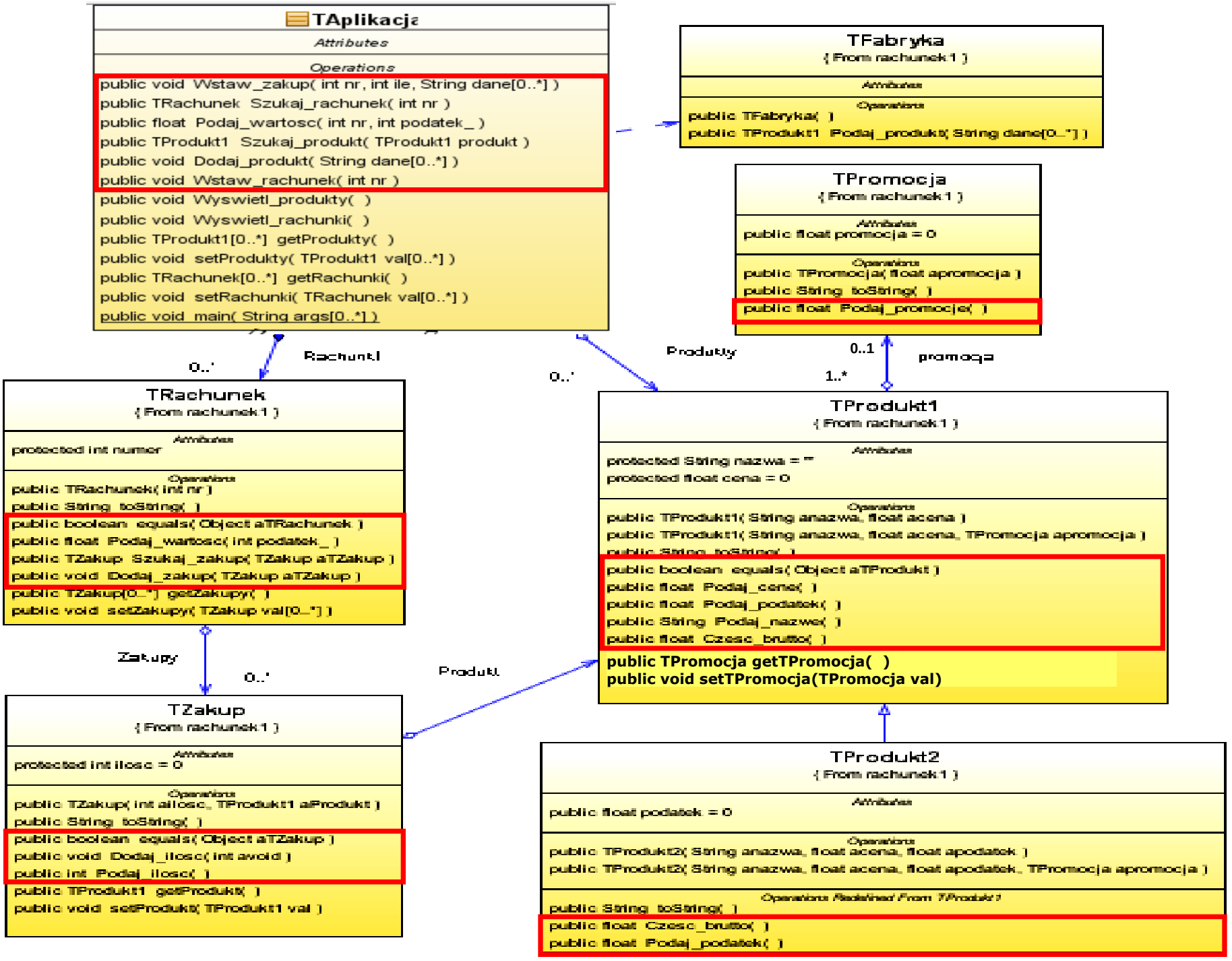
Definiowanie kodu metod realizujących przypadek użycia
na podstawie diagramów sekwencji

(6) Obliczanie wartosci rachunku

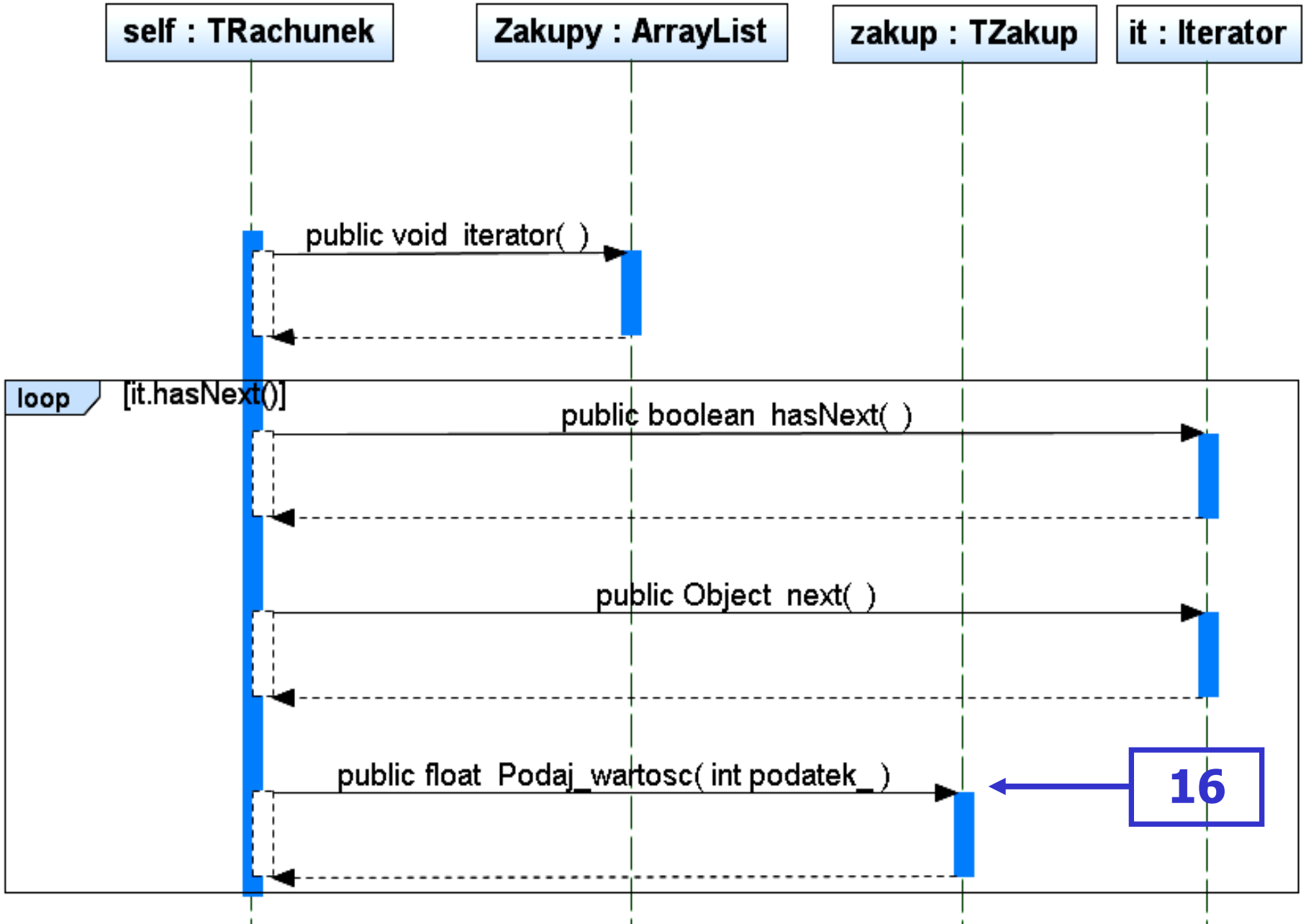
(float TAplikacja::Podaj_wartosc(int nr, int podatek_))



```
//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;
 }
```

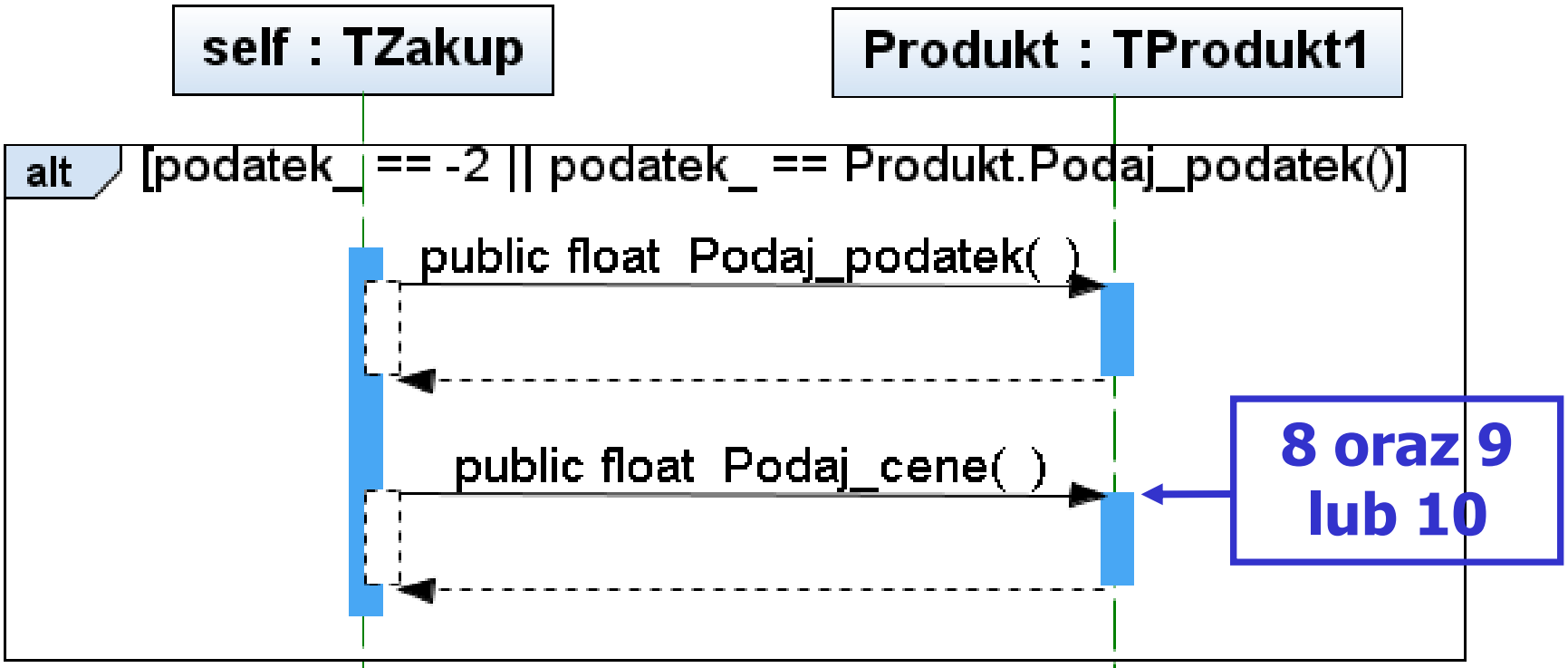

TAplikacja
Attributes
Operations
public void Wstaw_zakup(int nr, int ile, String dane[0..*])
public TRachunek Szukaj_rachunek(int nr)
public float Podaj_wartosc(int nr, int podatek_)
public TProdukt1 Szukaj_produkt(TProdukt1 produkt)
public void Dodaj_produkt(String dane[0..*])
public void Wstaw_rachunek(int nr)
public void Wyswietl_produkty( )
public void Wyswietl_rachunki( )
public TProdukt1[0..*] getProdukty( )
public void setProdukty(TProdukt1 val[0..*])
public TRachunek[0..*] getRachunki( )
public void setRachunki(TRachunek val[0..*])
public void main(String args[0..*])
TFabryka
{ From rachunek1 }
Attributes
Operations
public TFabryka( )
public TProdukt1 Podaj_produkt(String dane[0..*])
TPromocja
{ From rachunek1 }
Attributes
public float promocja = 0
Operations
public TPromocja(float apromocja)
public String toString( )
public float Podaj_promocje( )
0..'
Rachunki
0..'
Produkty
0..1
promocja
1..*
TRachunek
{ From rachunek1 }
Attributes
protected int numer
Operations
public TRachunek(int nr)
public String toString( )
public boolean equals(Object aTRachunek)
public float Podaj_wartosc(int podatek_)
public TZakup Szukaj_zakup(TZakup aTZakup)
public void Dodaj_zakup(TZakup aTZakup)
public TZakup[0..*] getZakupy( )
public void setZakupy(TZakup val[0..*])
TProdukt1
{ From rachunek1 }
Attributes
protected String nazwa = ""
protected float cena = 0
Operations
public TProdukt1(String anazwa, float acena)
public TProdukt1(String anazwa, float acena, TPromocja apromocja)
public boolean equals(Object aTProdukt)
public float Podaj_cene( )
public float Podaj_podatek( )
public String Podaj_nazwe( )
public float Czesc_brutto( )
public TPromocja getTPromocja( )
public void setTPromocja(TPromocja val)
Zakupy
0..'
Produkt
TZakup
{ From rachunek1 }
Attributes
protected int ilosc = 0
Operations
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 TProdukt1 getProdukt( )
public void setProdukt(TProdukt1 val)
TProdukt2
{ From rachunek1 }
Attributes
public 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( )

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

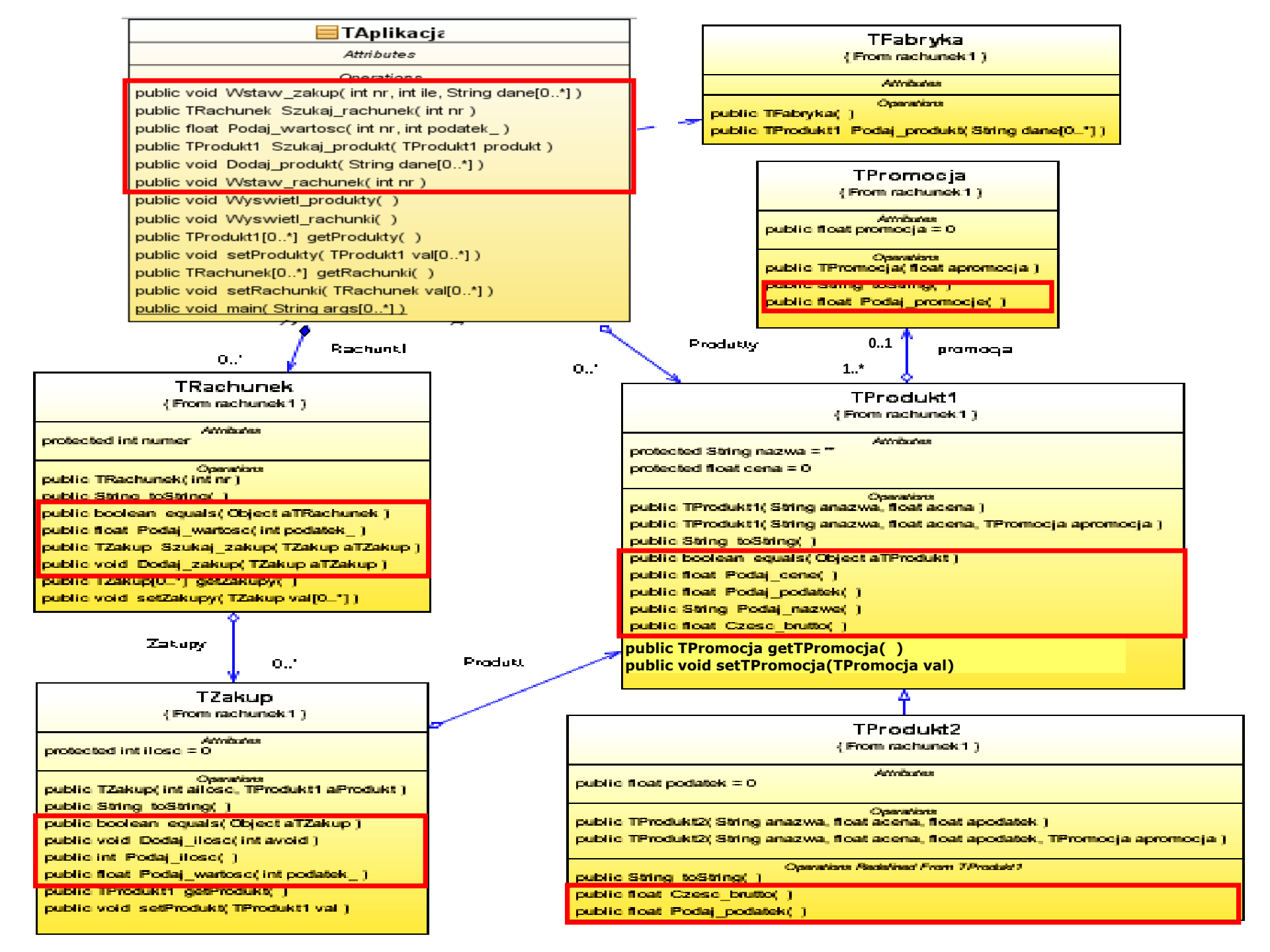


```
//TRachunek

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;
   }
```

(16) 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;
   }
```

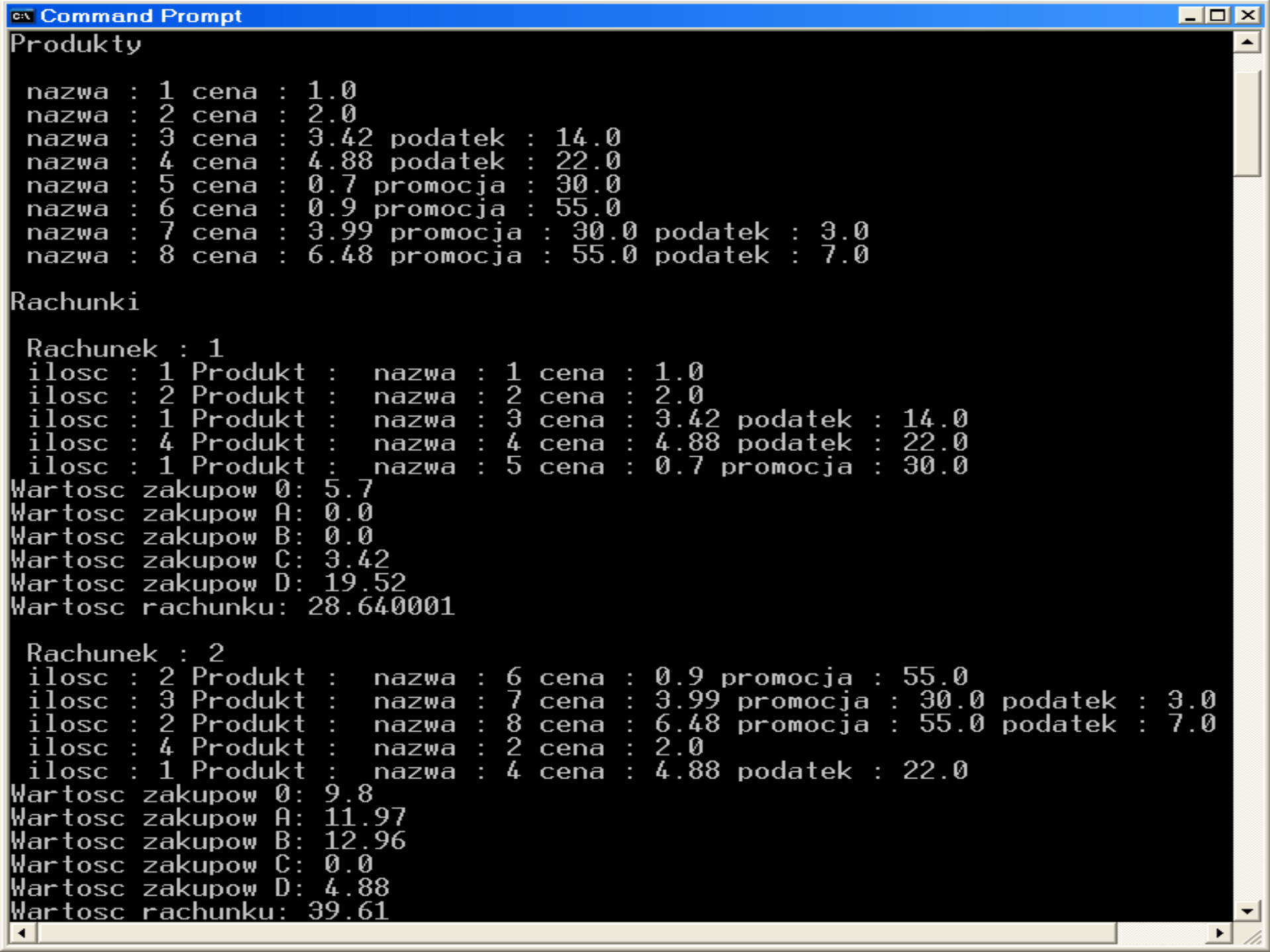

TAplikacja
Attributes
Operations
public void Wstaw_zakup(int nr, int ile, String dane[0..*])
public TRachunek Szukaj_rachunek(int nr)
public float Podaj_wartosc(int nr, int podatek_)
public TProdukt1 Szukaj_produkt(TProdukt1 produkt)
public void Dodaj_produkt(String dane[0..*])
public void Wstaw_rachunek(int nr)
public void Wyswietl_produkty( )
public void Wyswietl_rachunki( )
public TProdukt1[0..*] getProdukty( )
public void setProdukty(TProdukt1 val[0..*])
public TRachunek[0..*] getRachunki( )
public void setRachunki(TRachunek val[0..*])
public void main(String args[0..*])
TFabryka
{ From rachunek1 }
Attributes
Operations
public TFabryka( )
public TProdukt1 Podaj_produkt(String dane[0..*])
TPromocja
{ From rachunek1 }
Attributes
public float promocja = 0
Operations
public TPromocja(float apromocja)
public float Podaj_promocje( )
Rachunki
0..*
Produkty
0..*
0..1
promocja
1..*
TRachunek
{ From rachunek1 }
Attributes
protected int numer
Operations
public TRachunek(int nr)
public String toString( )
public boolean equals(Object aTRachunek)
public float Podaj_wartosc(int podatek_)
public TZakup Szukaj_zakup(TZakup aTZakup)
public void Dodaj_zakup(TZakup aTZakup)
public TZakup[0..*] getZakupy( )
public void setZakupy(TZakup val[0..*])
TProdukt1
{ From rachunek1 }
Attributes
protected String nazwa = ""
protected float cena = 0
Operations
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 Podaj_podatek( )
public String Podaj_nazwe( )
public float Czesc_brutto( )
public TPromocja getTPromocja( )
public void setTPromocja(TPromocja val)
Zakupy
0..*
Produkt
TZakup
{ From rachunek1 }
Attributes
protected int ilosc = 0
Operations
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_)
public TProdukt1 getProdukt( )
public void setProdukt(TProdukt1 val)
TProdukt2
{ From rachunek1 }
Attributes
public 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( )

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

```
public String toString()
{ TZakup z;
  StringBuilder sb = new StringBuilder();
  sb.append(" Rachunek : ");
  sb.append(numer).append("\n");
  Iterator<TZakup> it = Zakupy.iterator();
  while (it.hasNext()) {
          z = it.next();
          sb.append(z.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 pom;
        if ((pom = app.Szukaj_rachunek(1)) != null) {
            System.out.println(pom.toString()); }
        if ((pom = app.Szukaj_rachunek(2)) != null) {
            System.out.println(pom.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