

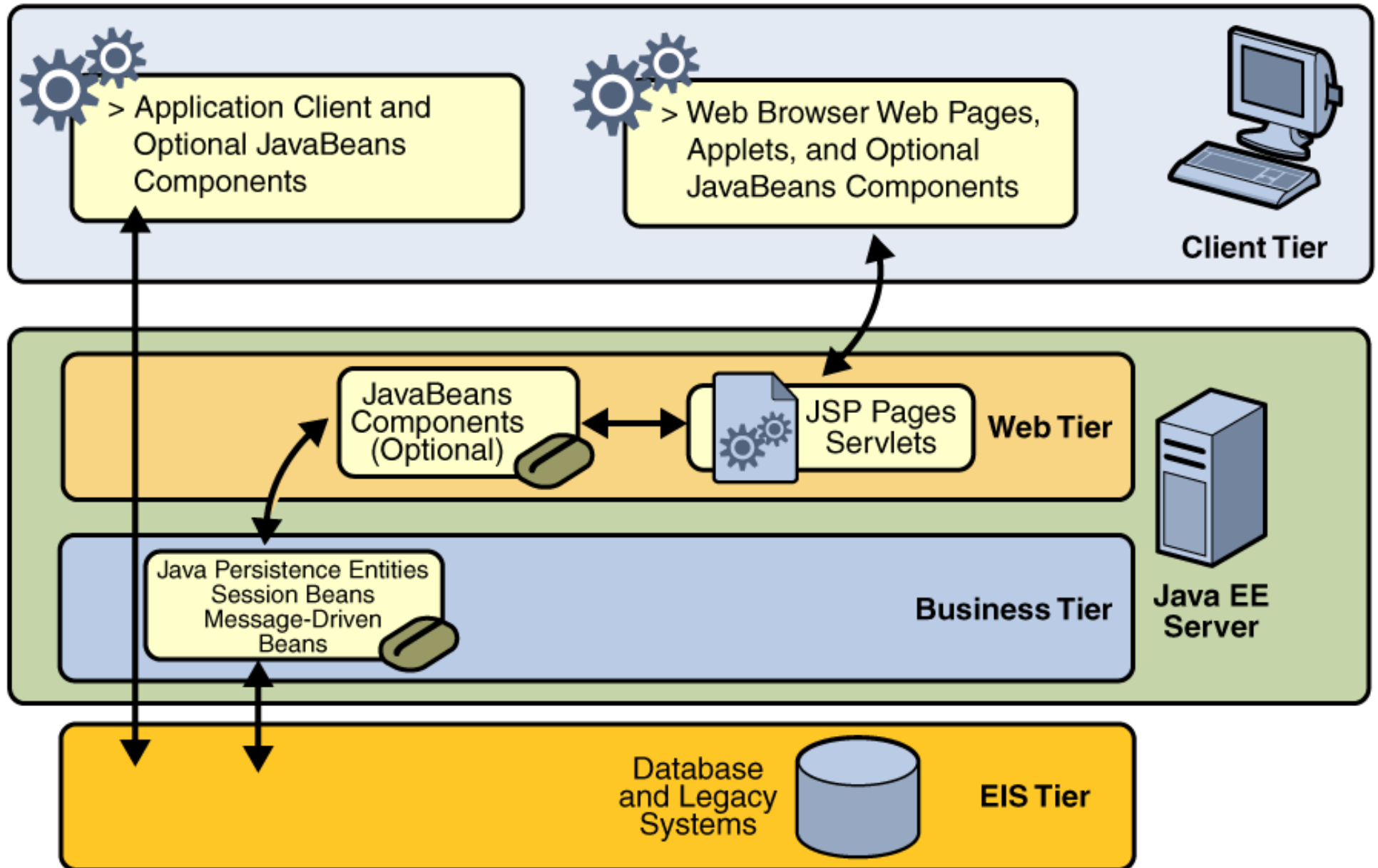
# **Wzorce oprogramowania zastosowane w modelu obiektowym**

(wg Alan Shalloway, James R.Trott)

- 1. Wstęp**
- 2. Refaktoryzacja architektury  
wielowarstwowej**
- 3. Wzorce projektowe**
- 4. Przykład warstwy biznesowej  
stosującej wzorce obiektowe**

# 1. Wstep

# Warstwy aplikacji (Java EE)



# Pięciowarstwowy model logicznego rozdzielania zadań (wg. D.Alur, J.Crupi, D. Malks, Core J2EE. Wzorce projektowe.)

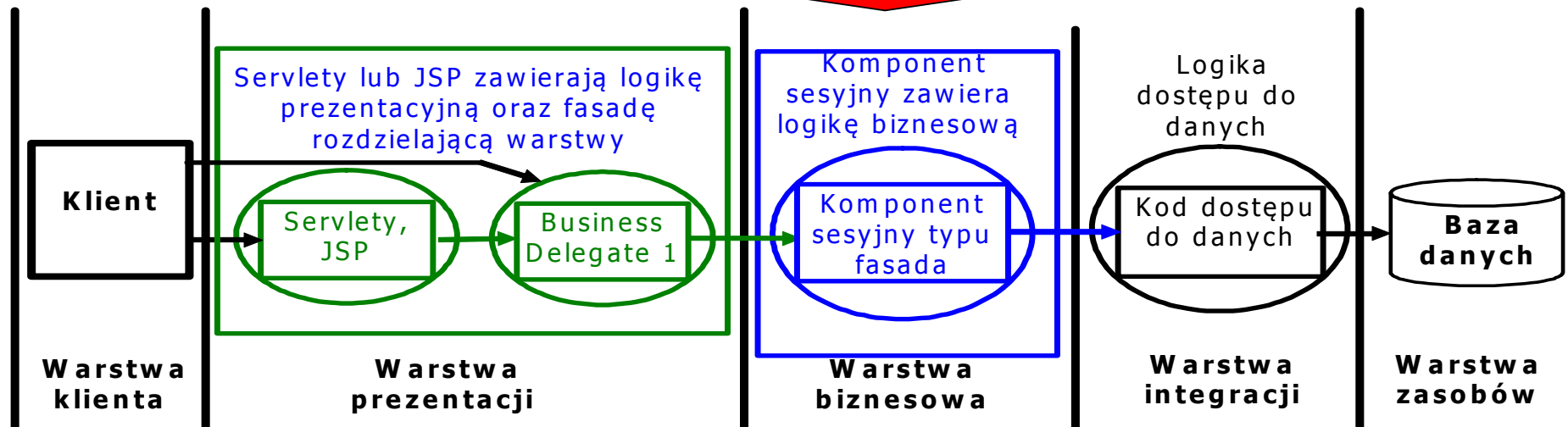
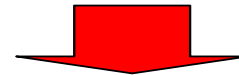


## **2. Refaktoryzacja architektury wielowarstwowej**

## Refaktoryzacja architektury wielowarstwowej 1

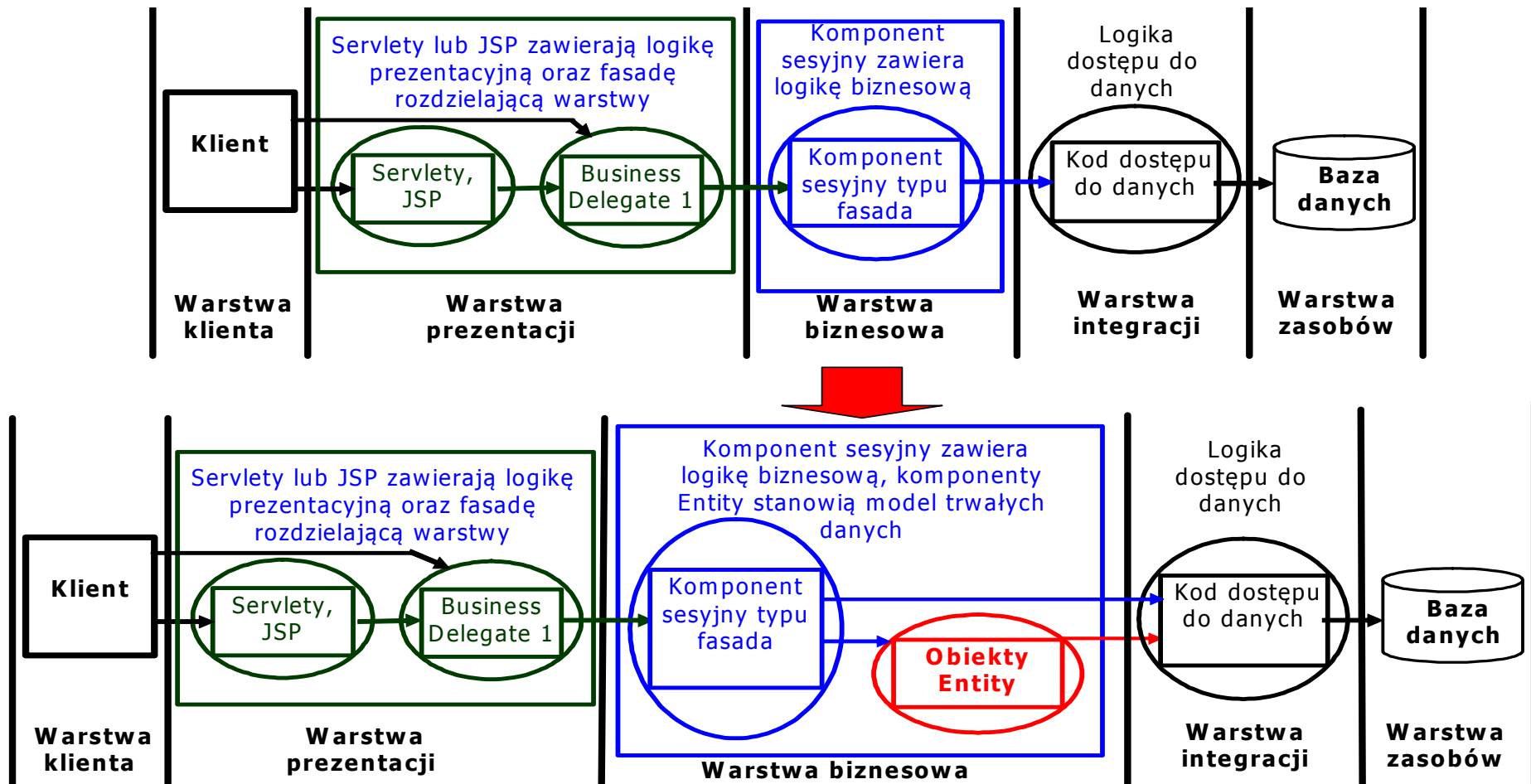
Należy przenieść kod dostępu do danych logicznie lub fizycznie bliżej rzeczywistego źródła danych, a logikę przetwarzania z klienta i warstwy prezentacji do warstwy biznesowej zawierającej **fasadowe komponenty sesyjne typu „Control”**.

**Komponenty Business Delegate typu „Control”** hermetyzują dostęp do warstwy biznesowej z warstwy prezentacji – stanowią przedłużenie warstwy biznesowej.



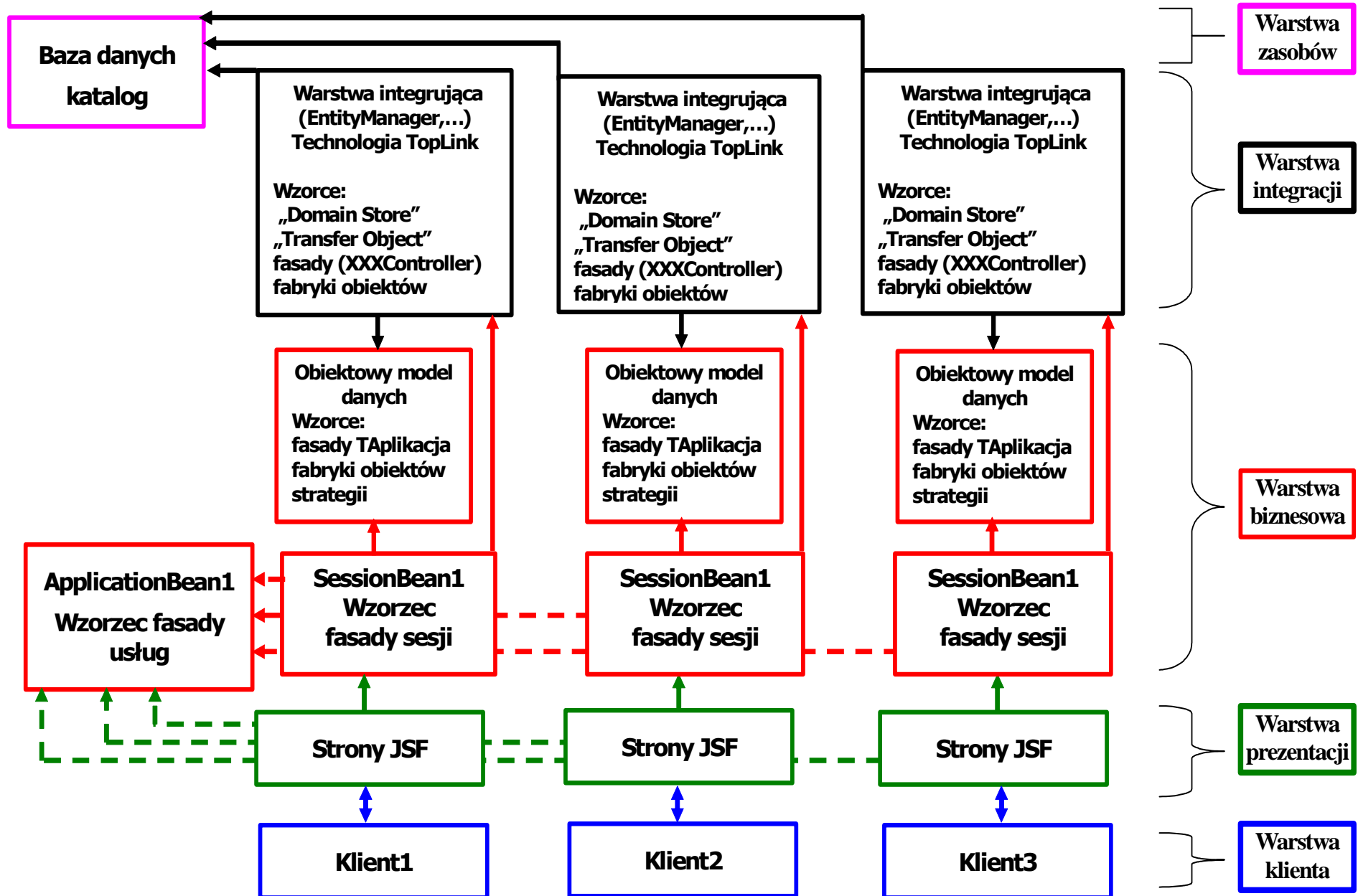
## Refaktoryzacja architektury wielowarstwowej 2

Należy przenieść kod dostępu do danych logicznie lub fizycznie bliżej rzeczywistego źródła danych, a złożoną logikę przetwarzania z klienta i warstwy prezentacji typu do warstwy biznesowej zawierającą **obiekty danych typu „Entity”** i **hermetyzujące dostęp do tych komponentów fasadowe komponenty sesyjne typu „Control”**. **Komponenty Business Delegate typu „Control”** hermetyzują dostęp do warstwy biznesowej z warstwy prezentacji.



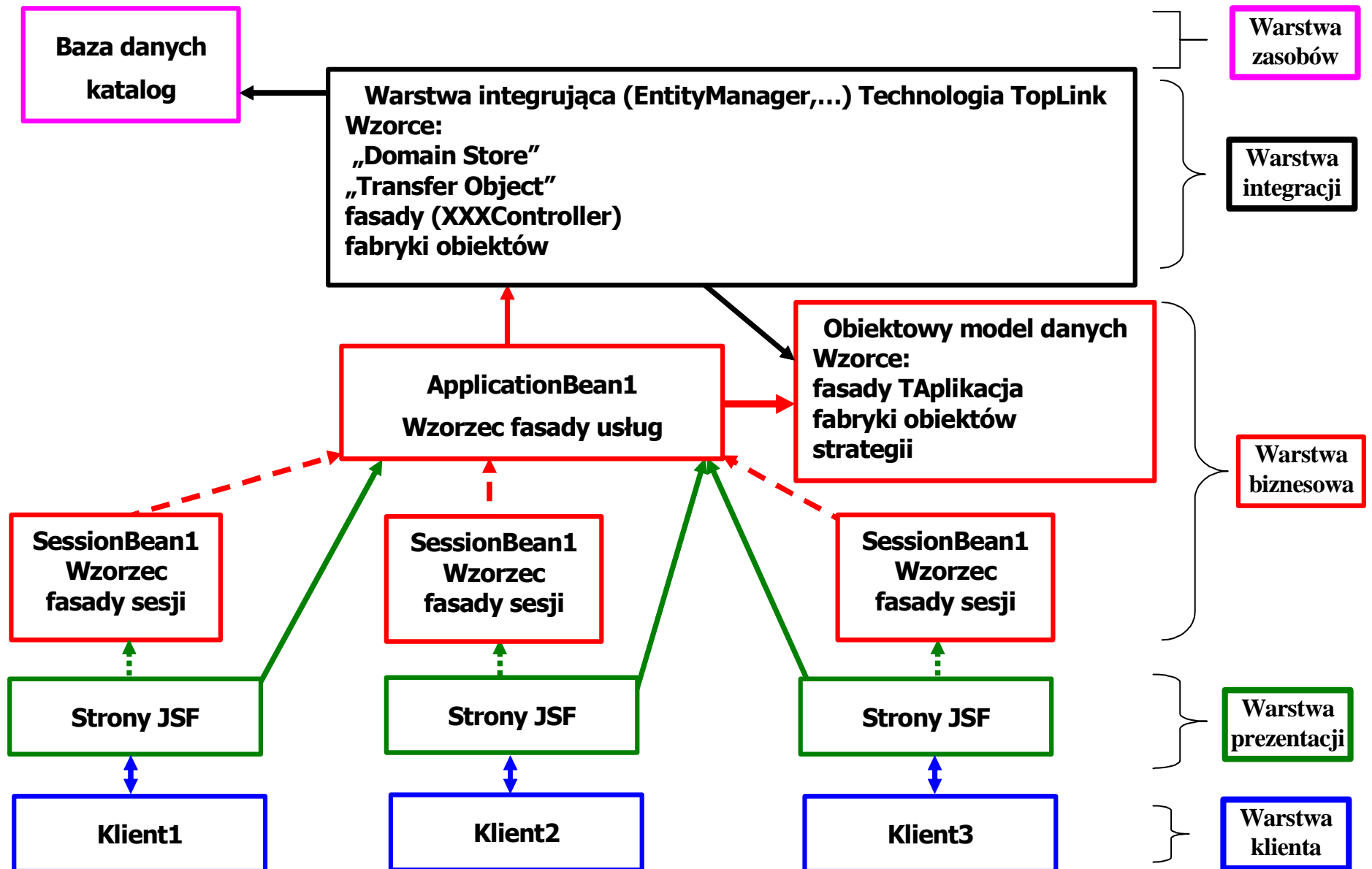
# Architektura aplikacji pięciowarstwowej – Java EE 5.0 Visual Web Java Server Faces

(linie przerywane oznaczają powiązania nie wykorzystane w aplikacji)





# Architektura aplikacji pięciowarstwowej Java EE 5.0 Visual Web Java Server Faces - linie przerywane oznaczają powiązania nie wykorzystane w aplikacji



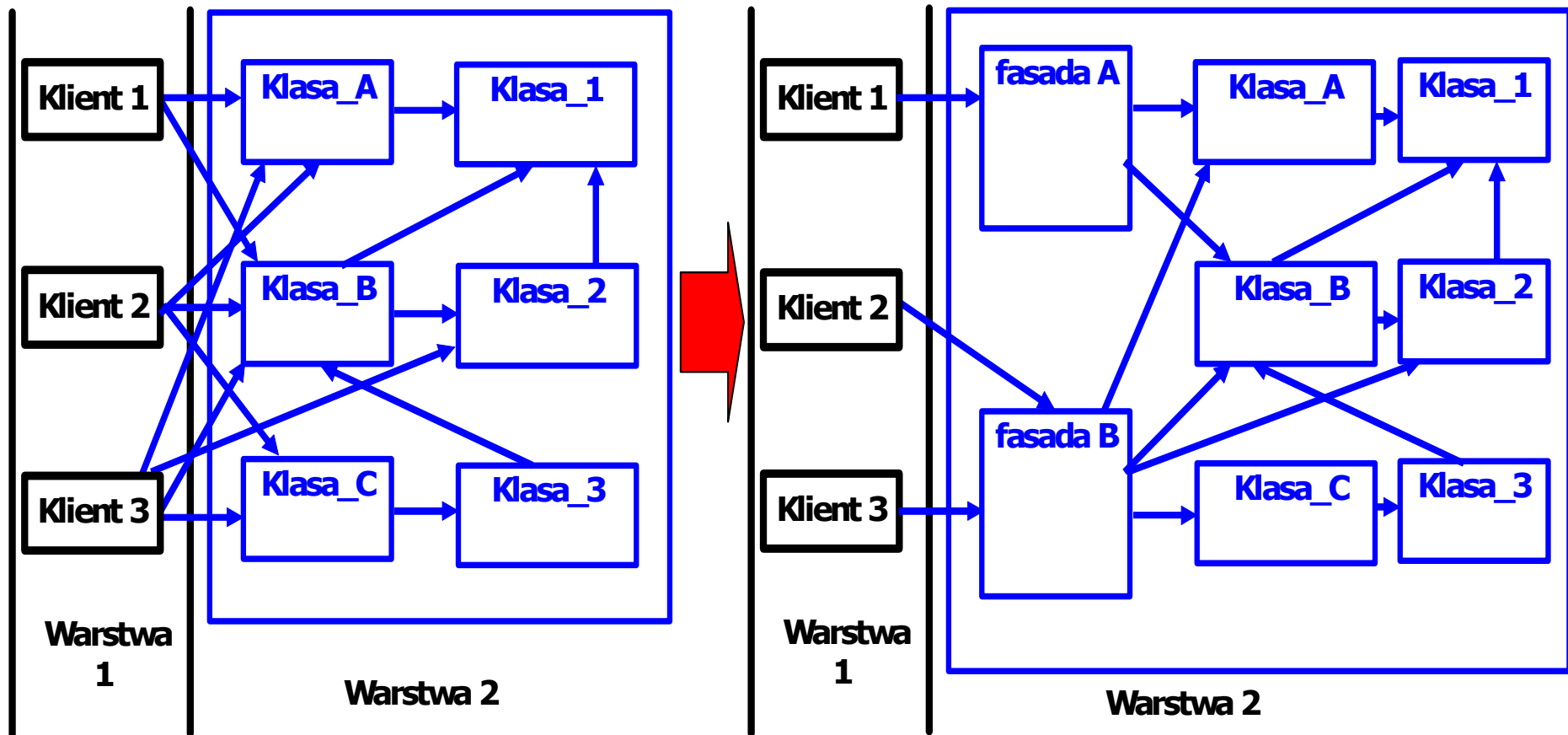
# **3. Wzorce projektowe**

# Wzorce projektowe

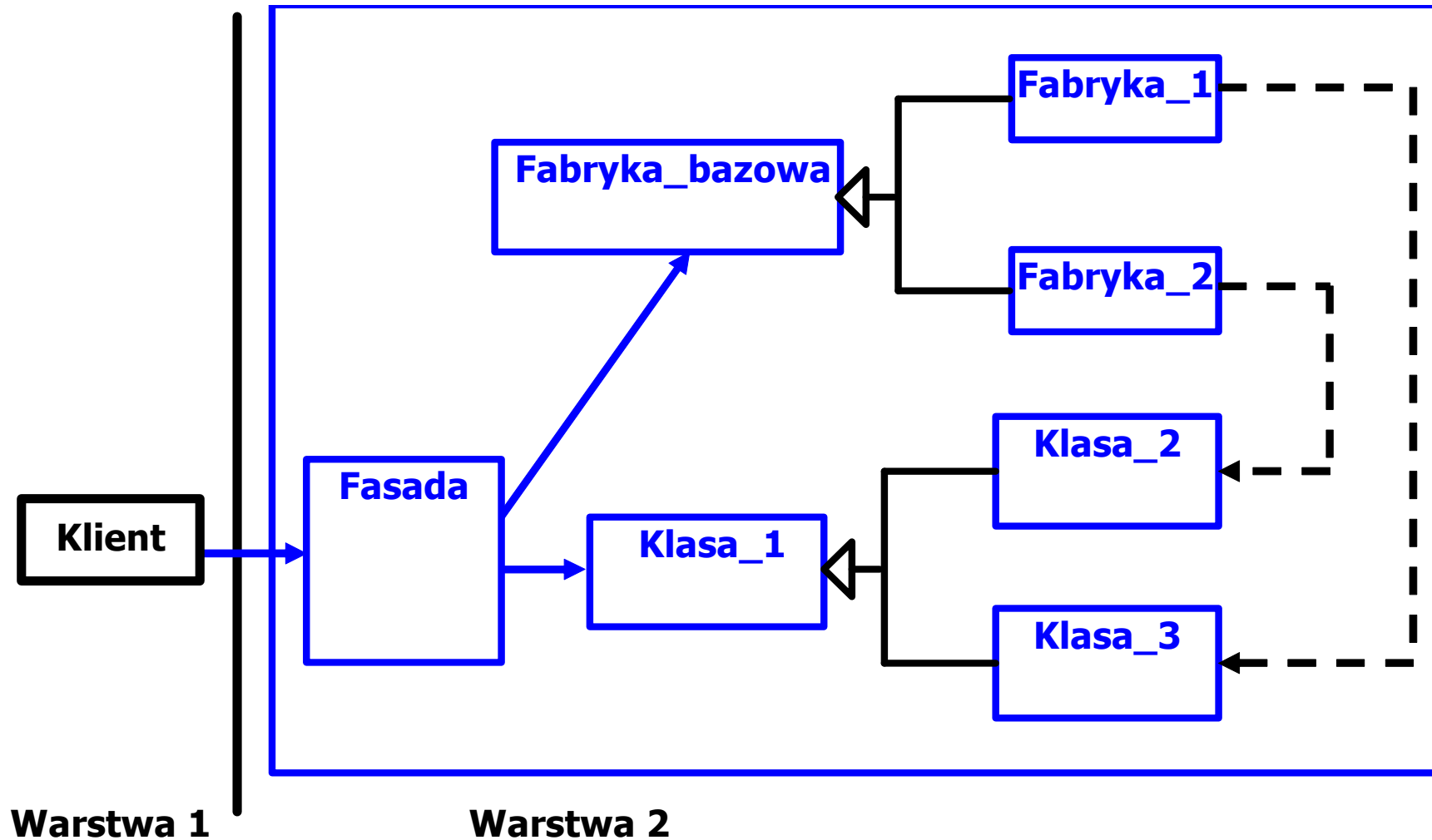
- Dobrze zbudowany system obiektowy jest pełen wzorców obiektowych
- Wzorzec to zwyczajowo przyjęte rozwiązanie typowego problemu w danym kontekście
- Strukturę wzorca przedstawia się w postaci diagramu klas
- Zachowanie się wzorca przedstawia się za pomocą diagramu sekwencji
- Wzorce projektowe: Wzorzec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

- Wzorzec to pomysł, który okazał się użyteczny w jednym rzeczywistym kontekście i prawdopodobnie będzie użyteczny w innym. **(Martin Fowler)**

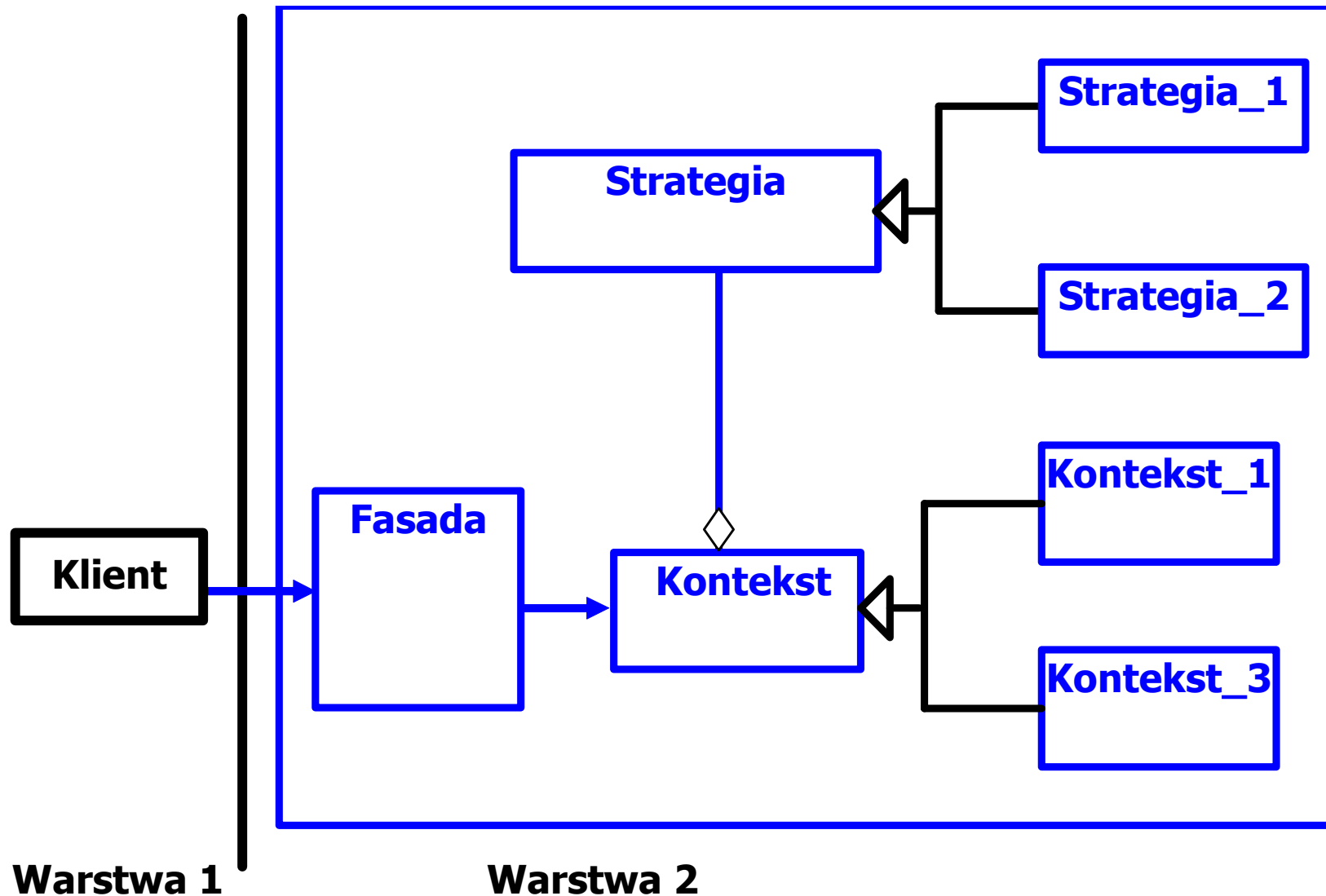
# 3.1. Wzorzec fasady



# 3.2. Wzorzec fabryki obiektów



# 3.3. Wzorzec strategii



# **4. Przykład warstwy biznesowej stosującej wzorce obiektowe**

# System sporządzania rachunków

- 4.1. Sformułowanie wymagań funkcjonalnych i niefunkcjonalnych systemu
- 4.2. **Model analizy całego systemu** oparty na diagramie przypadków użycia
- 4.3. **Model projektowy warstwy biznesowej** oparty na diagramie klas i diagramie sekwencji tworzony metodą iteracyjno-rozwojową sterowany realizacją przypadków użycia
- 4.4. **Implementacja warstwy biznesowej** tworzona w cyklu iteracyjno-rozwojowym sterowana rozwojem modelu projektowego



# **4.1. Sformułowanie wymagań funkcjonalnych i niefunkcjonalnych systemu**

# System sporządzania rachunków

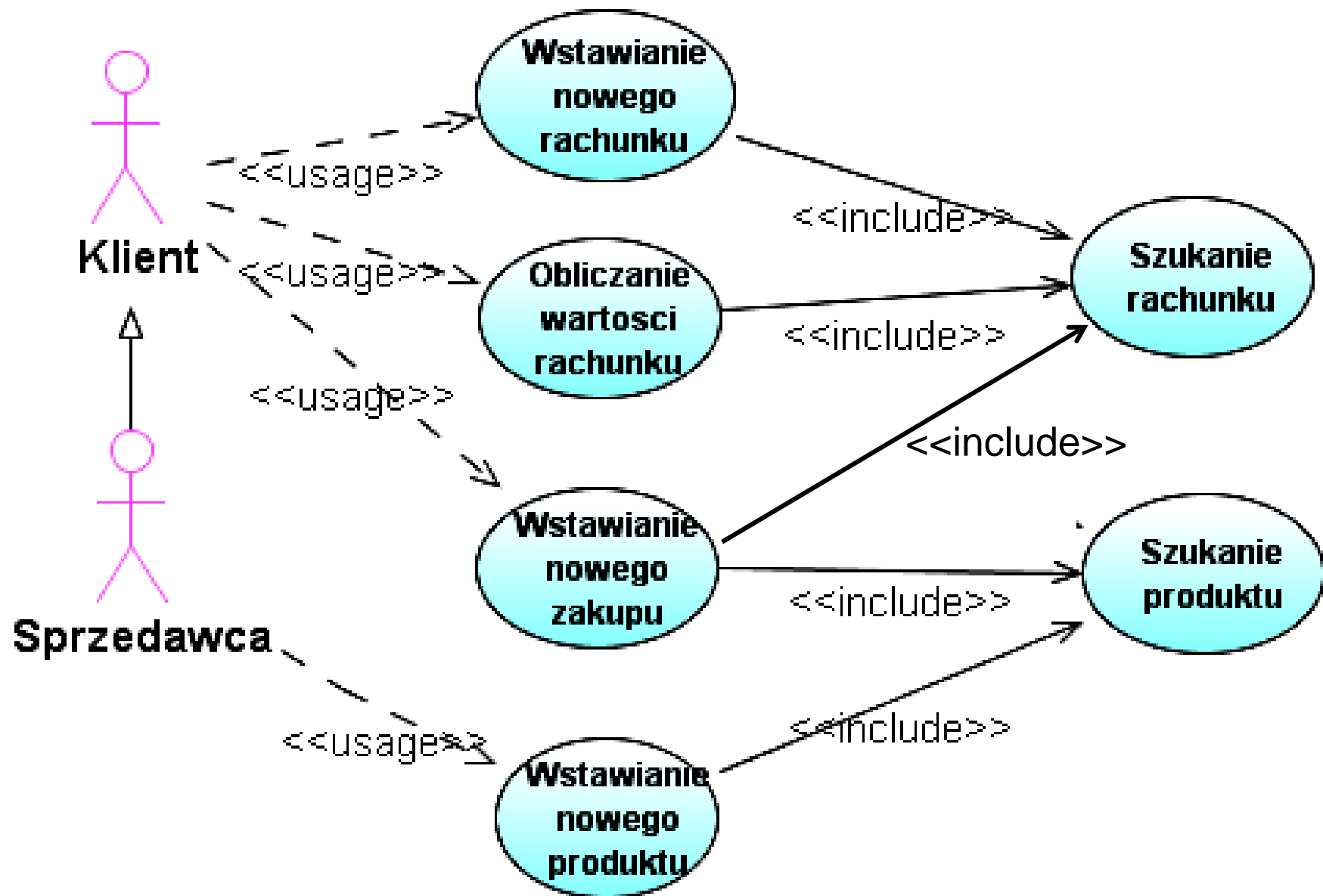
## Lista wymagań funkcjonalnych

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

## **4.2. Model analizy całego systemu oparty na diagramie przypadków użycia**



AKTOR	OPIS	PRZYPADKI UŻYCIA
Klient	<i>Klient może dokonywać zakupów wybranych produktów przez Internet korzystając z przeglądarki lub z aplikacji</i>	<ul style="list-style-type: none"> <li>• Wstawianie nowego rachunku powiązane przez &lt;&lt;include&gt;&gt; z PU Szukanie rachunku</li> <li>• Obliczanie wartości rachunku powiązane przez &lt;&lt;include&gt;&gt; z PU Szukanie rachunku</li> <li>• Wstawianie nowego zakupu powiązane przez &lt;&lt;include&gt;&gt; z PU Szukanie rachunku oraz powiązane przez &lt;&lt;include&gt;&gt; z PU Szukanie produktu</li> </ul>
Sprzedawca	<i>Sprzedawca może dodatkowo dodawać nowe produkty</i>	<ul style="list-style-type: none"> <li>• Wstawianie nowego rachunku powiązane przez &lt;&lt;include&gt;&gt; z PU Szukanie rachunku</li> <li>• Obliczanie wartości rachunku powiązane przez &lt;&lt;include&gt;&gt; z PU Szukanie rachunku</li> <li>• Wstawianie nowego zakupu powiązane przez &lt;&lt;include&gt;&gt; z PU Szukanie rachunku oraz powiązane przez &lt;&lt;include&gt;&gt; z PU Szukanie produktu</li> <li>• Wstawianie nowego produktu powiązane przez &lt;&lt;include&gt;&gt; z PU Szukanie produktu<sup>21</sup></li> </ul>

## **PU Szukanie produktu**

### **OPIS**

#### **CEL: Poszukiwanie produktu**

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

**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 według atrybutów: nazwy i ceny (obowiązkowo) oraz podatku i promocji (jeśli jest to wymagane) zgodnie z danymi podanymi do przypadku użycia
2. Jeśli istnieje produkt o podanych atrybutach, zwracany jest produkt, w przeciwnym wypadku zwracana jest informacja o braku produktu.

## **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. Należy podać atrybuty produktu: nazwę, cenę jako obowiązkowe dane oraz podatek i cenę detaliczną, jeśli jest to wymagane
2. Należy wywołać **PU Szukanie produktu**. Należy sprawdzić, czy produkt o podanych atrybutach już istnieje. Jeśli tak, należy zakończyć PU, w przeciwnym wypadku należy wstawić nowy produkt.

## **PU Szukanie rachunku**

### **OPIS**

#### **CEL: Poszukiwanie rachunku**

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

**WK (warunki końcowe):** podanie rachunku o podanym numerze lub komunikat o braku rachunku

### **PRZEBIEG:**

1. Szukanie rachunku przebiega według numeru podanego do przypadku użycia
2. Jeśli 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. Należy podać numer rachunku, który powinien być niepowtarzalny, ponieważ służy do identyfikacji rachunku
2. Należy wywołać **PU Szukanie rachunku** w celu sprawdzenia, czy numer rachunku się powtarza.
3. Jeśli zwrócony wynik oznacza brak rachunku o podanym numerze, można wstawić nowy rachunek i zakończyć PU, w przeciwnym wypadku należy zakończyć 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 zwiększenie ilości zakupionego produktu, jeśli już taki produkt zakupiono lub komunikat o braku rachunku

#### **PRZEBIEG:**

1. Należy podać numer rachunku, który powinien być niepowtarzalny, ponieważ służy do identyfikacji rachunku
2. Należy wywołać **PU Szukanie rachunku** w celu sprawdzenia, czy istnieje rachunek o podanym numerze.
3. Jeśli zwrócony wynik oznacza brak rachunku o podanym numerze, nie można wstawić nowego zakupu do rachunku i należy zakończyć PU, w przeciwnym wypadku należy wstawić nowy zakup
4. Należy wybrać produkt oraz ilość zakupionego produktu.
5. Należy wywołać **PU Szukanie produktu**. Jeśli wybrany produkt nie istnieje, należy zakończyć PU. W przeciwnym przypadku należy wstawić nowy zakup do rachunku, przeglądając, czy istnieje już zakup z takim samym produktem. Jeśli istnieje, nie tworzy się nowego zakupu, tylko powiększa się ilość zakupu istniejącego o ilość nowego zakupu, w przeciwnym przypadku wstawia się 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

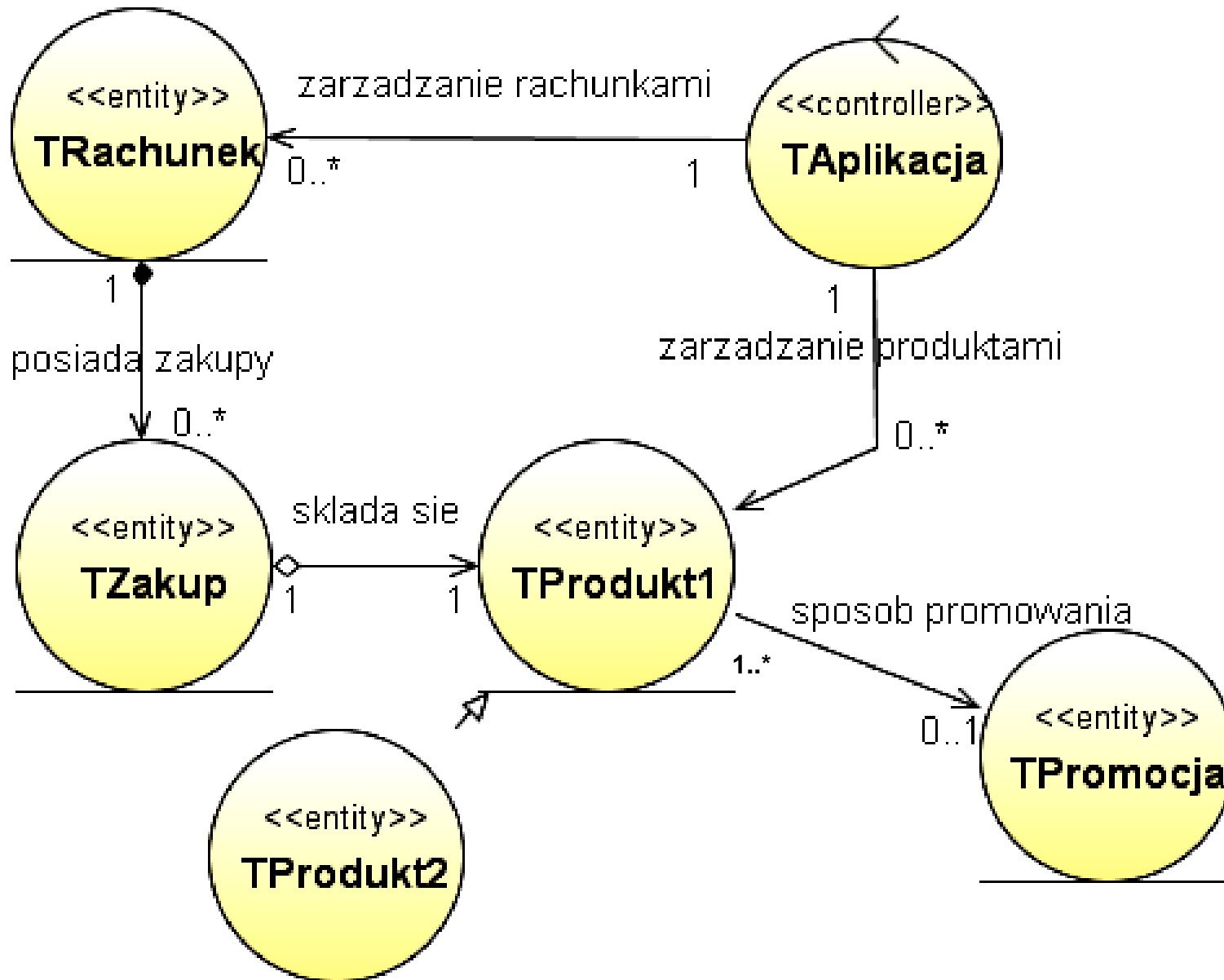
### **PRZEBIEG:**

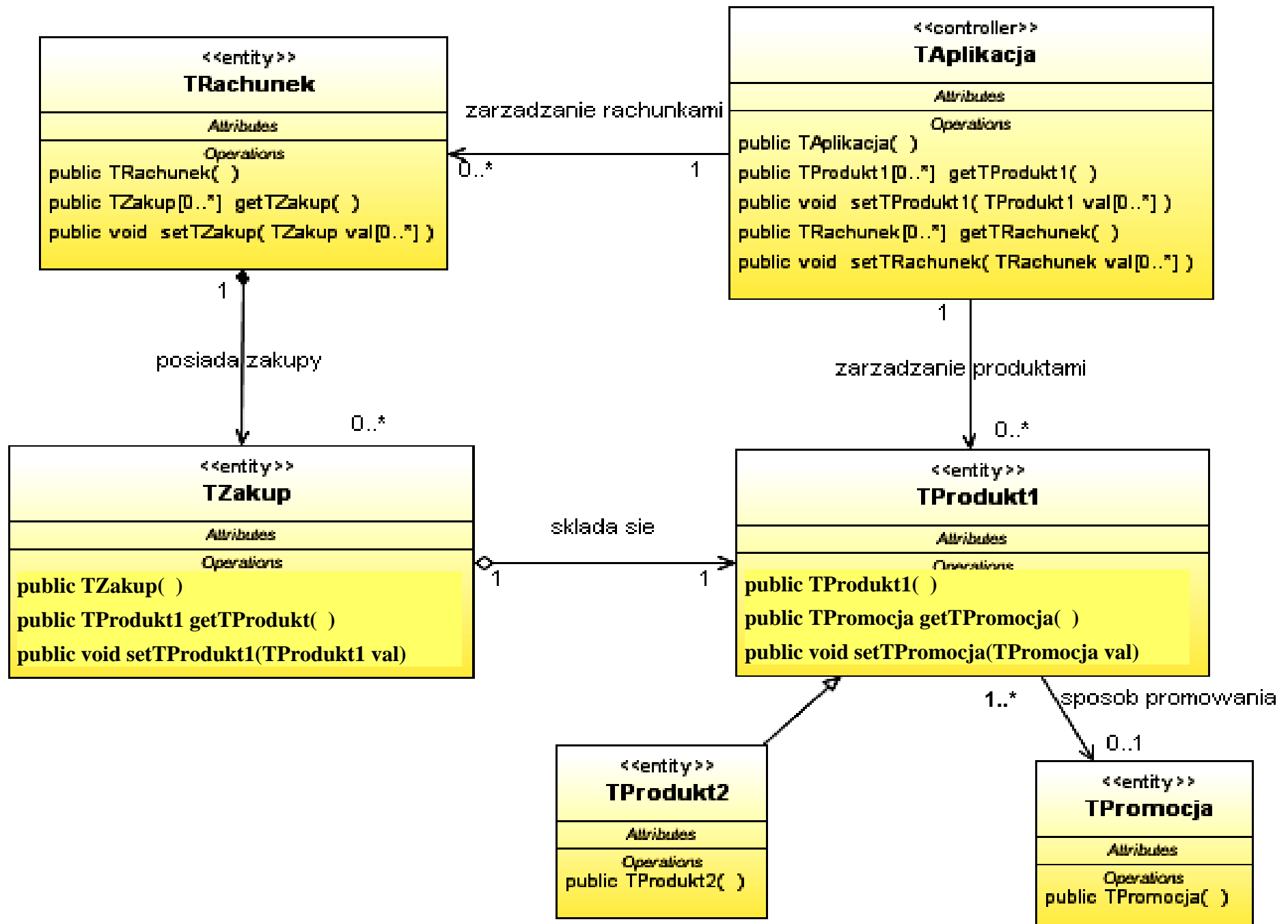
1. Należy podać numer rachunku, który powinien być niepowtarzalny, ponieważ służy do identyfikacji rachunku oraz wartość podatku lub wartość -2
2. Należy wywołać **PU Szukanie rachunku** w celu sprawdzenia, czy rachunek o podanym numerze istnieje.
3. Jeśli zwrócony wynik oznacza brak rachunku o podanym numerze, nie można obliczyć wartości wybranego rachunku i należy zakończyć PU, w przeciwnym wypadku należy obliczyć wartość rachunku
4. Należy uruchomić petle, w której sumowane są wartości zakupu obliczane jako iloczyn ceny jednostkowej zakupionego produktu i ilości zakupu. Jeśli zachodzi potrzeba sumowania wartości zakupu zależna od wysokości podatku, należy podać wartość podatku i sumować jedynie zakupy o podanym podatku, w przeciwnym wypadku sumowane są wszystkie zakupy (gdy zamiast podatku zostanie przekazana wartość -2).

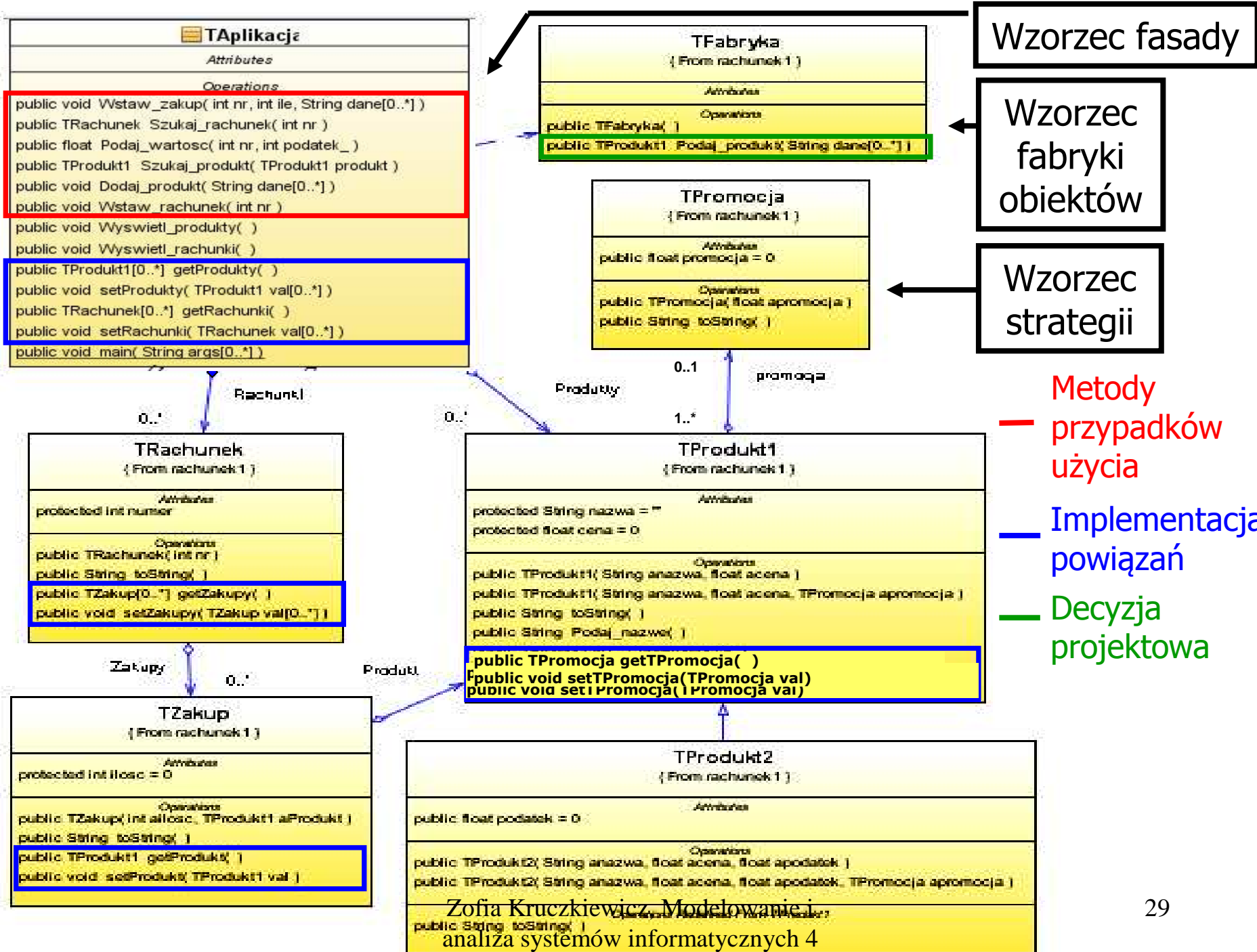
# Analiza wspólności i zmienności

- 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. Stąd 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 klasę **TAplikacja 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”







# **Wygenerowany kod klas na podstawie diagramu klas opracowanego w fazy analizy**

```

package rachunek1;
import java.util.ArrayList;
public class TAplikacja
{
    private ArrayList<TProdukt1> Produkty = new ArrayList<TProdukt1>();
    private ArrayList<TRachunek> Rachunki = new ArrayList<TRachunek>();
    public ArrayList<TProdukt1> getProdukty ()                { return null; }
    public void setProdukty (ArrayList<TProdukt1> val)      { }
    public ArrayList<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_produk (TProdukt1 produkt)    { return null; }
    public void Dodaj_produk (String[] dane)               { }
    public static void main (String[] args)                { }
    public void Wyswietl_produkty ()                       { }
    public void Wyswietl_rachunki ()                       { }
}

```

```

package rachunek1;
import java.util.ArrayList;

class TRachunek
{
    protected int numer;
    private ArrayList<TZakup> Zakupy = new ArrayList<TZakup>();
    public ArrayList<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; }
```

```
}
```

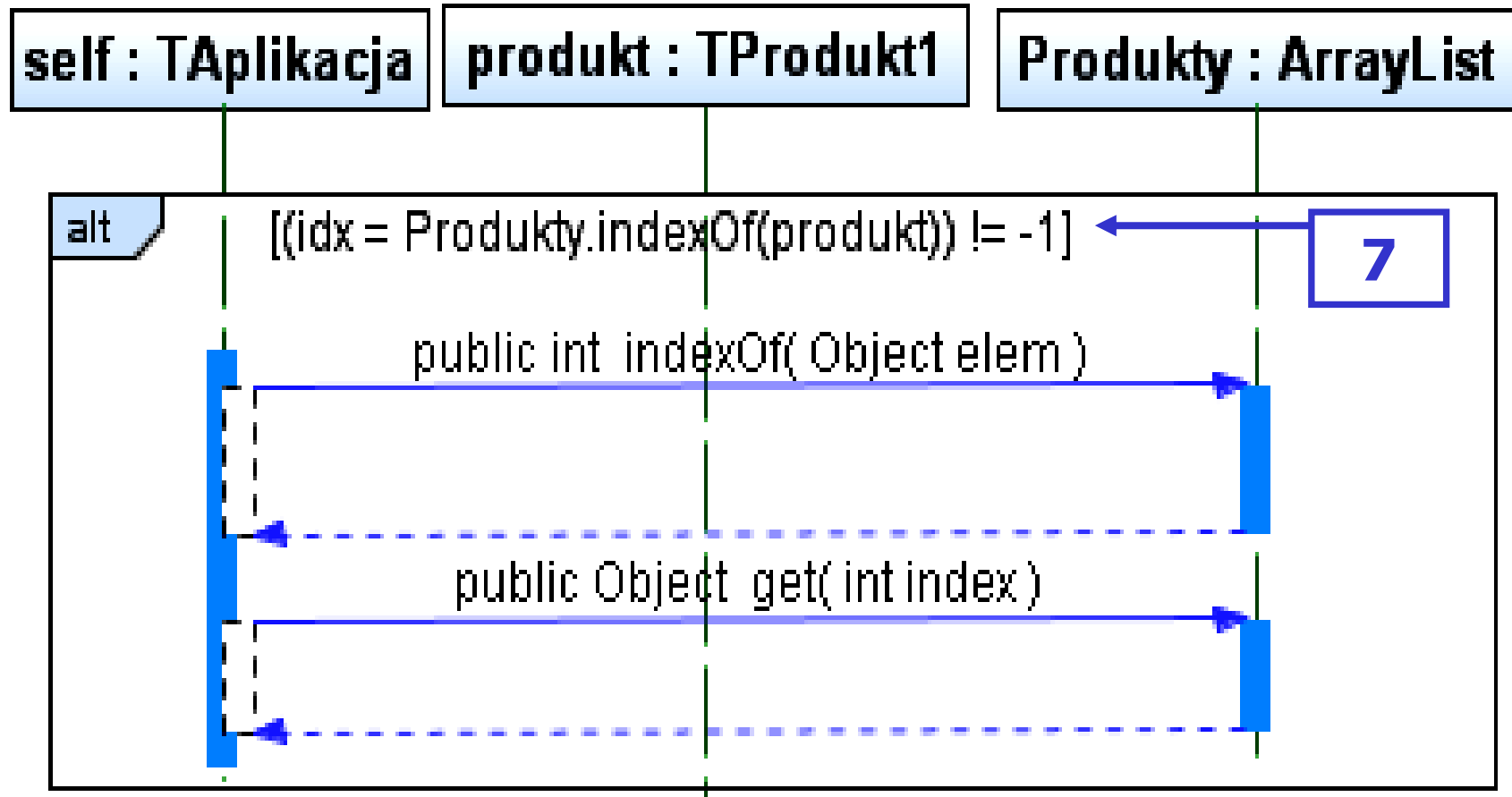
**4.3. Model projektowy warstwy biznesowej** oparty na diagramie klas i diagramie sekwencji tworzony metodą iteracyjno-rozwojową sterowany realizacją przypadków użycia

**4.4. 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))

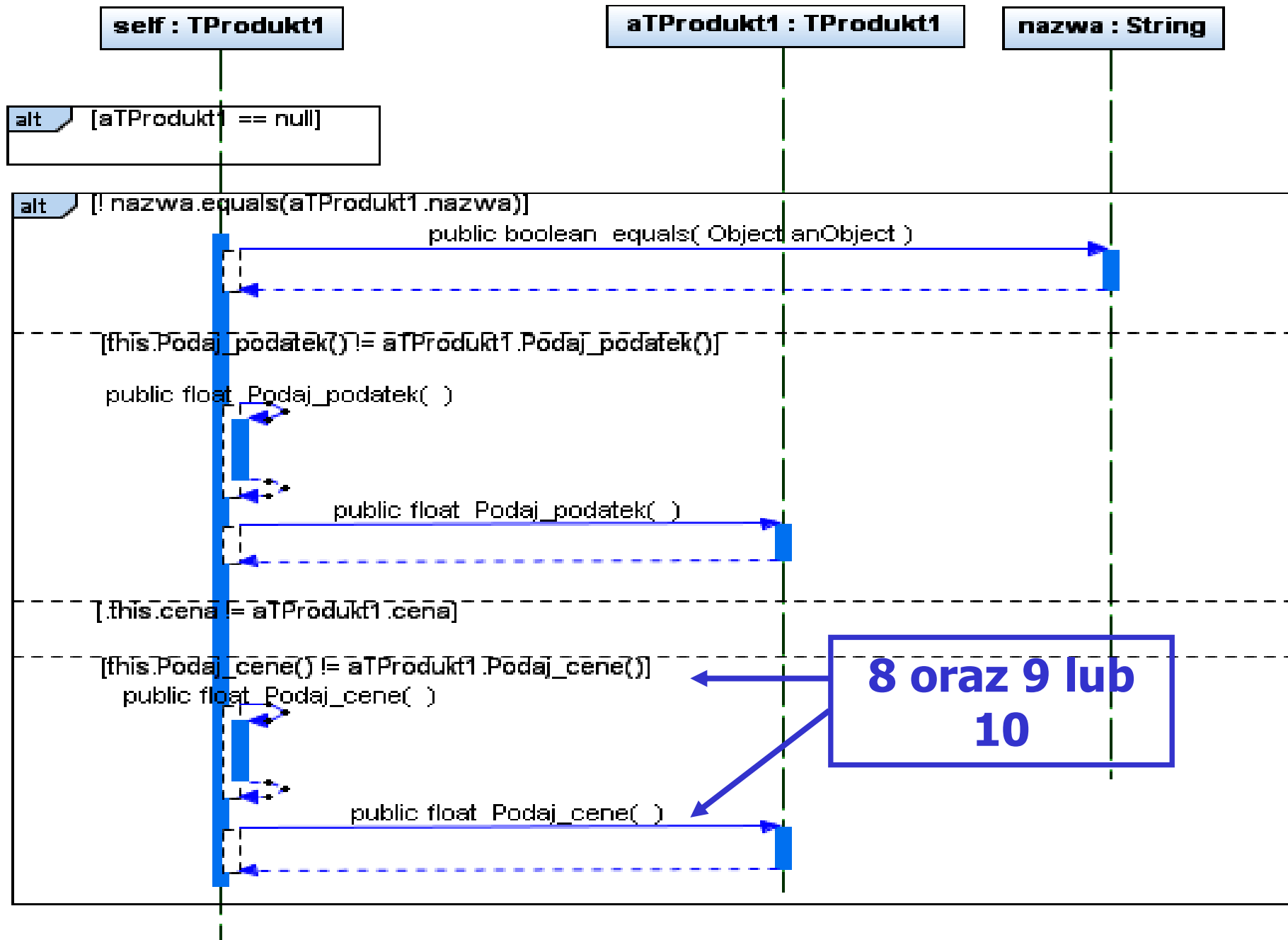


```
//TAplikacja
```

```
static private ArrayList <TProdukt1> Produkty =  
                new ArrayList <TProdukt1>();
```

```
public 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)

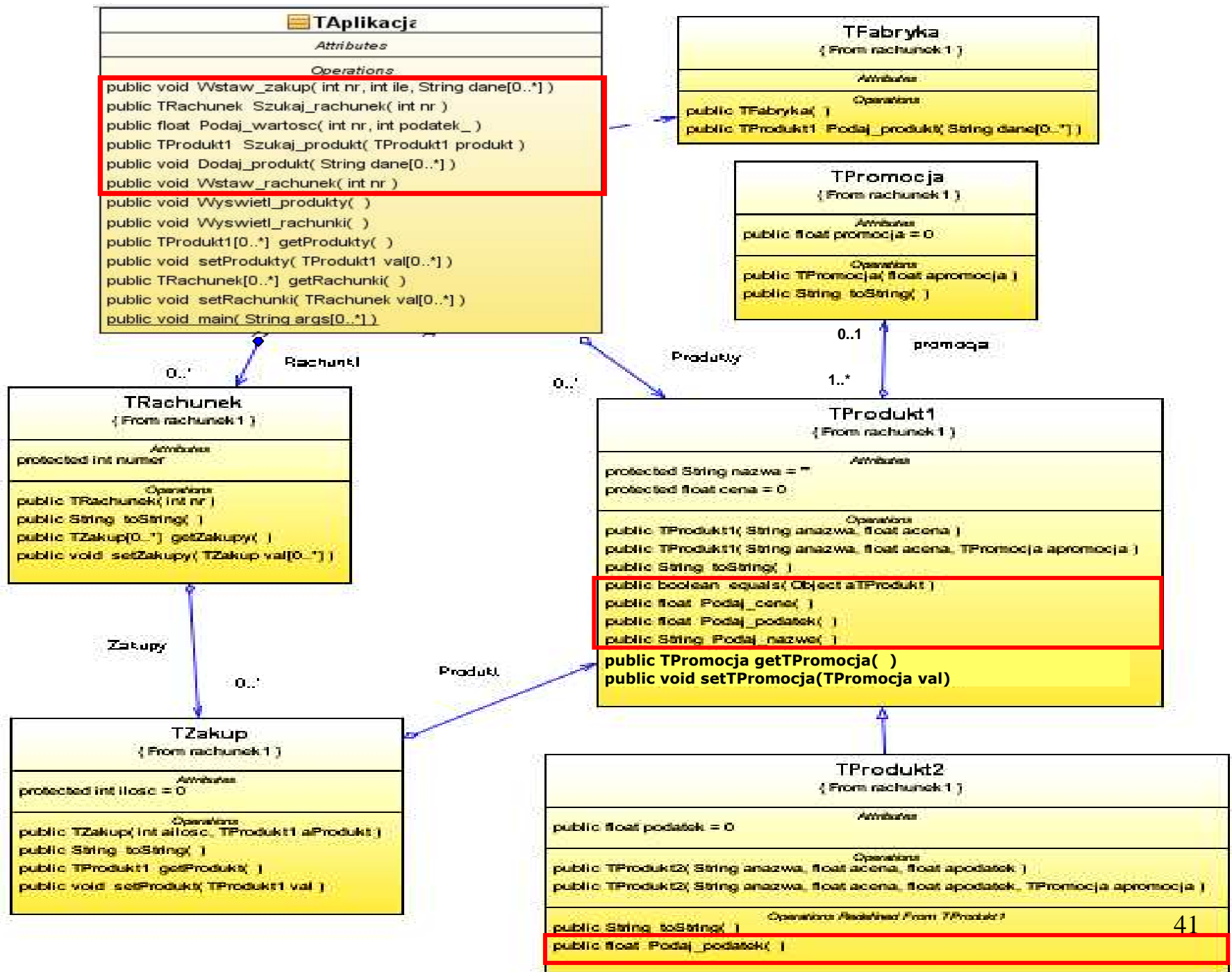


```

//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;
}

```





(8)

float TProdukt1::Podaj\_cene()

self : TProdukt1

public float Czesc\_brutto( )

9 lub 10

(9)

float TProdukt1::Czesc\_brutto()

self : TProdukt1

promocja : TPromocja

alt [promocja != null]

public float Podaj\_promocje( float cena )

(10)

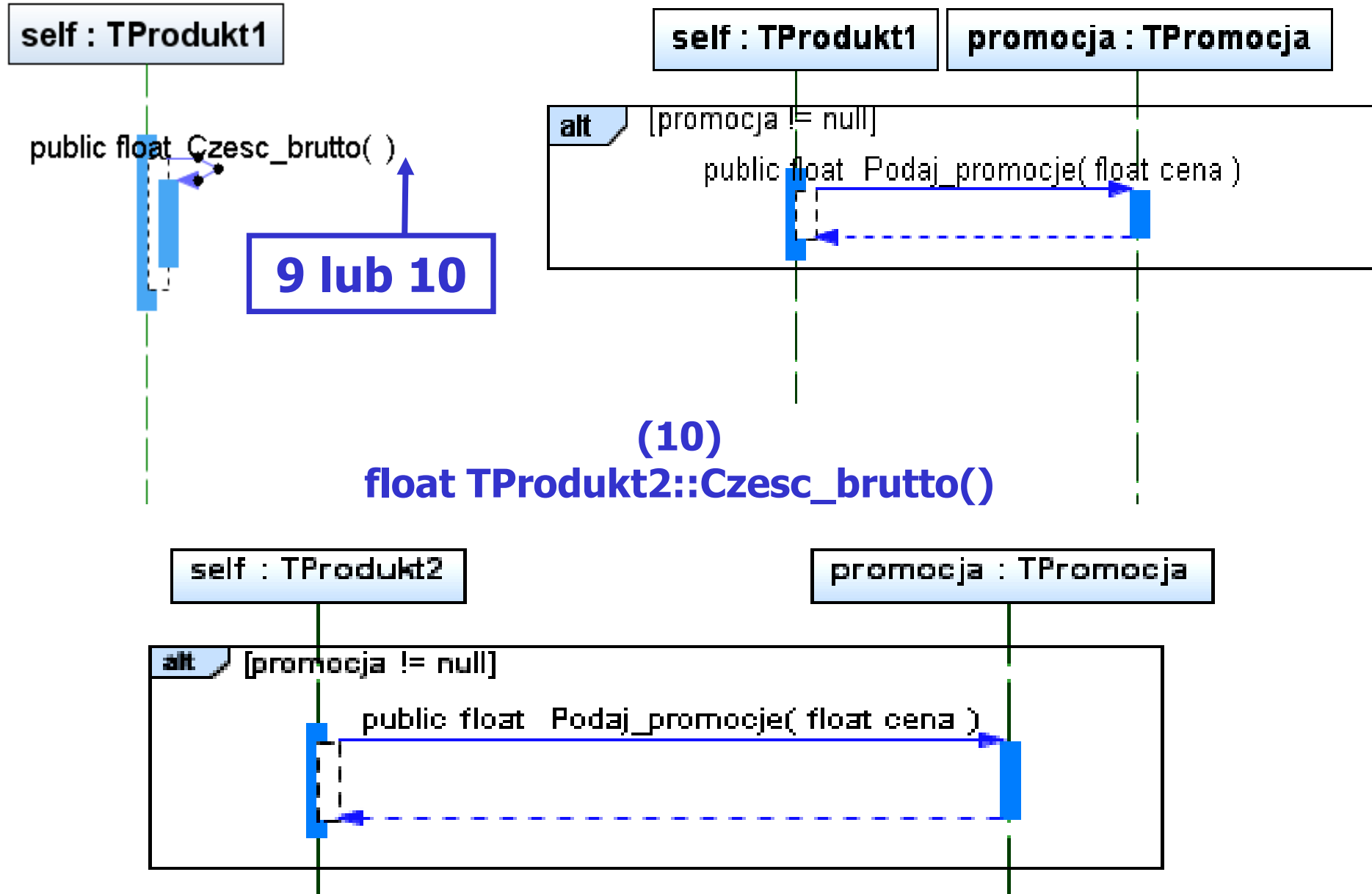
float TProdukt2::Czesc\_brutto()

self : TProdukt2

promocja : TPromocja

alt [promocja != null]

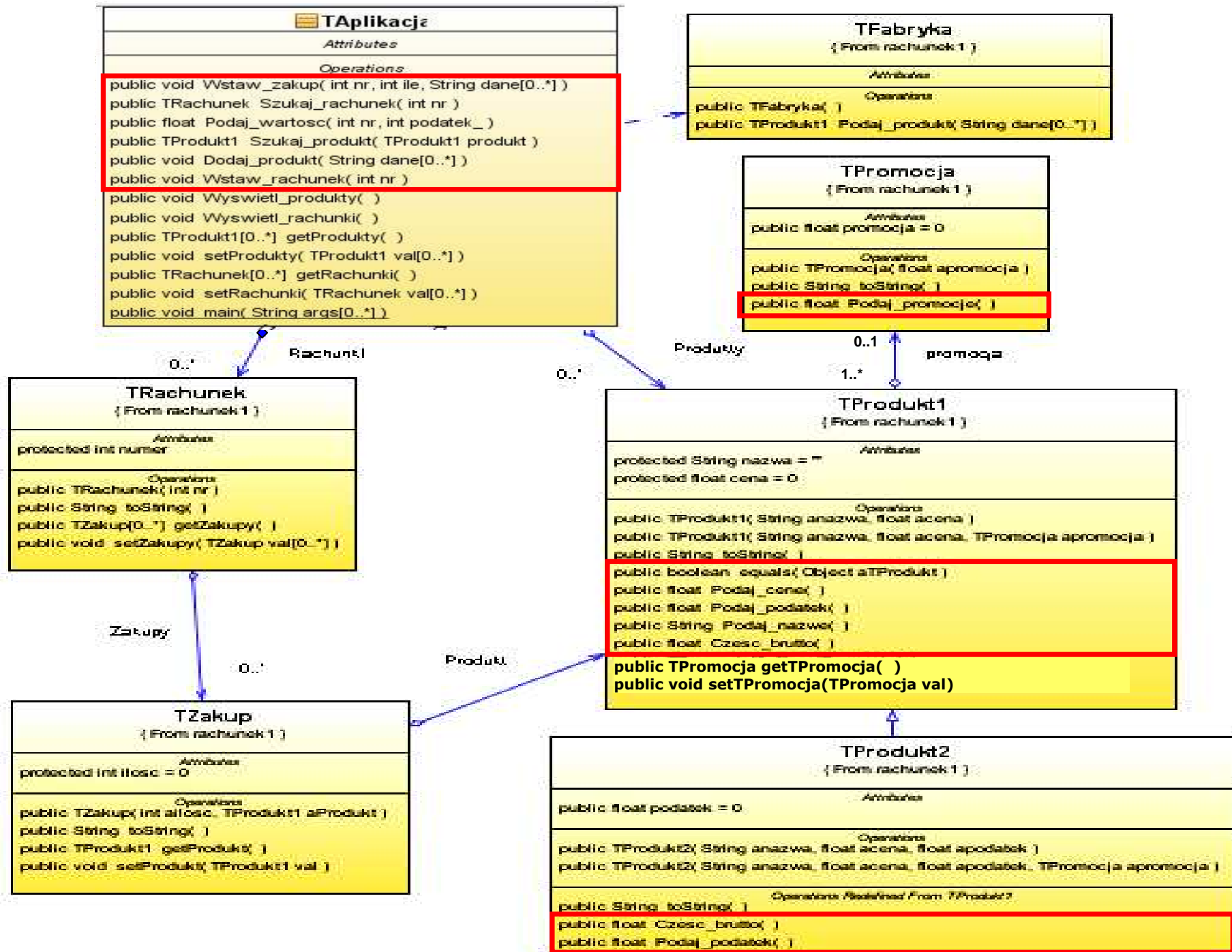
public float Podaj\_promocje( float cena )



```
//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 następc
public float Podaj_promocje ()
{ if (promocja<50)    //jakiś algorytm obliczania promocji
    return promocja;
    return promocja *1.1F;
}
```



Projekt przypadku użycia

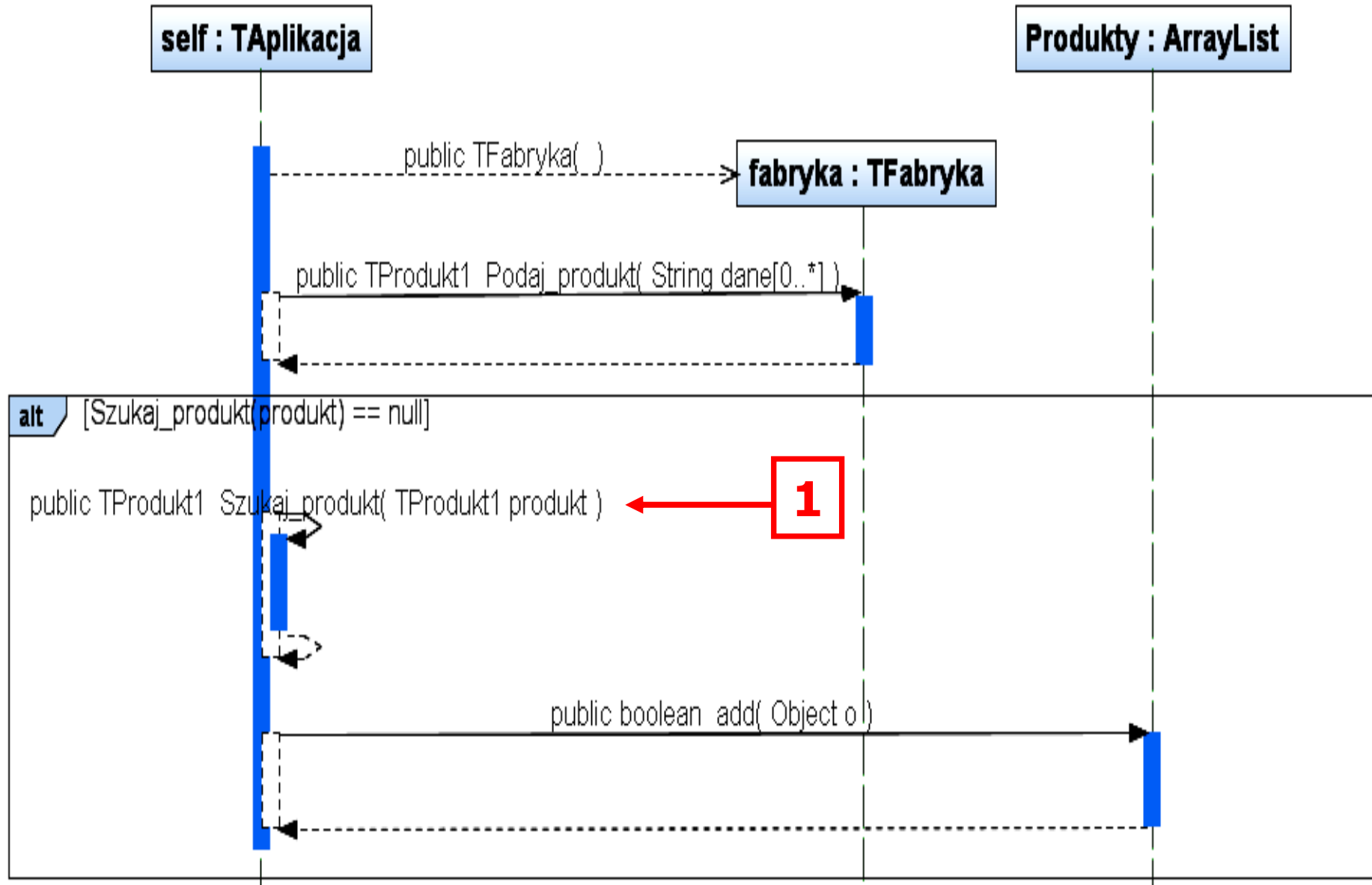
„ **Wstawianie nowego produktu** ”

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

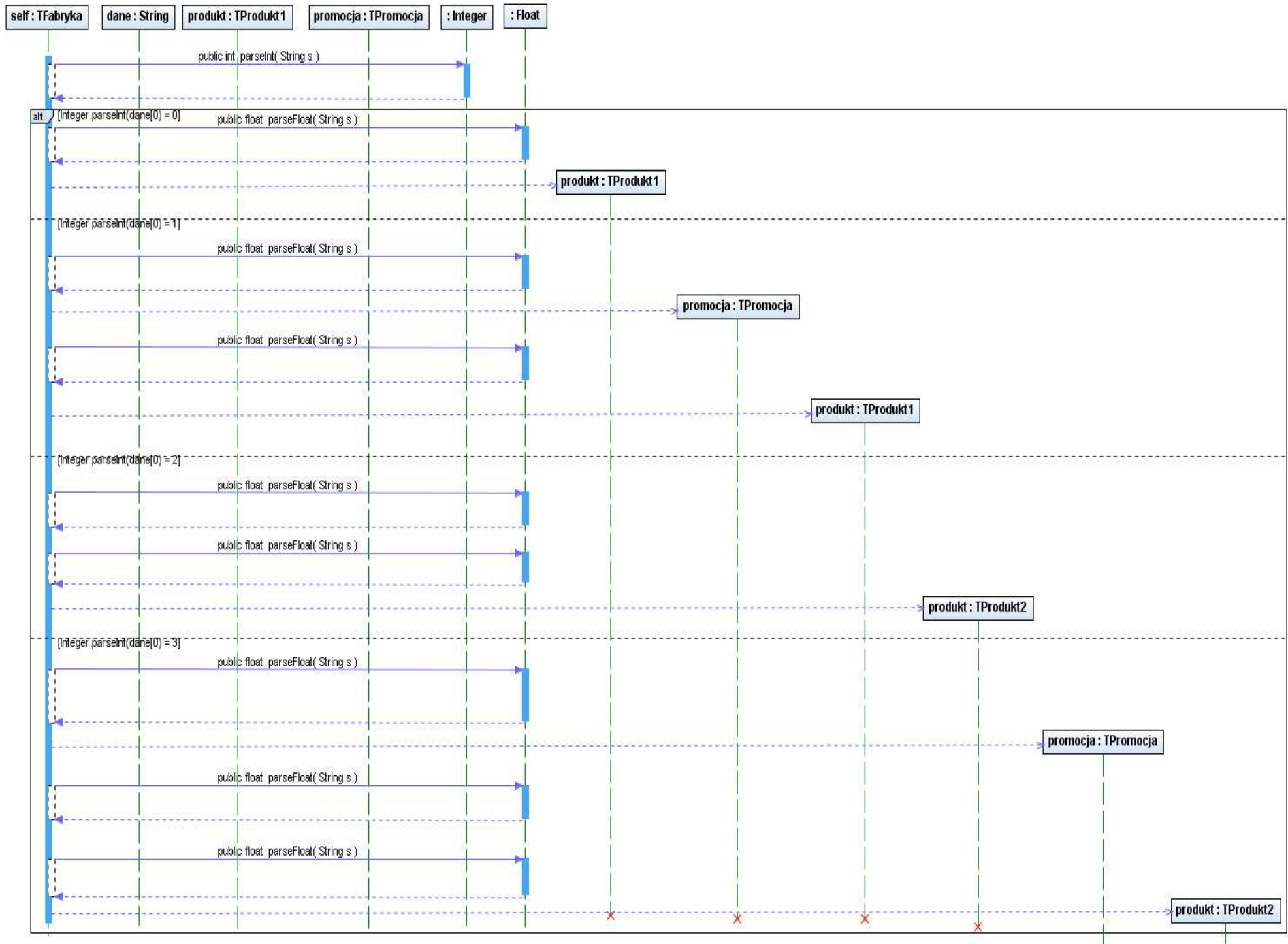
## (2) Wstawianie nowego produktu

(void TAplikacja::Dodaj\_produkt(String [] dane))



```
//class TAplikacja  
  
static private ArrayList <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 //Decyzje na poziomie tworzenia kodu
{
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;

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;
}
}

```

**promocja** – obiekt z rodziny typu **Strategia**  
**produkt** – obiekt z rodziny **Kontekst**

```
//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()  
{ StringBuffer sb = new StringBuffer();  
  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()  
{  
  StringBuffer sb =  
    new StringBuffer ();  
  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"};
    app.Dodaj_produkt(dane1);
    String dane2[]={"0","2","2"};
    app.Dodaj_produkt(dane2);
    app.Dodaj_produkt(dane1);
    String dane3[]={"2","3","3","14"};
    app.Dodaj_produkt(dane3);
    String dane4[]={"2","4","4","22"};
    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
```

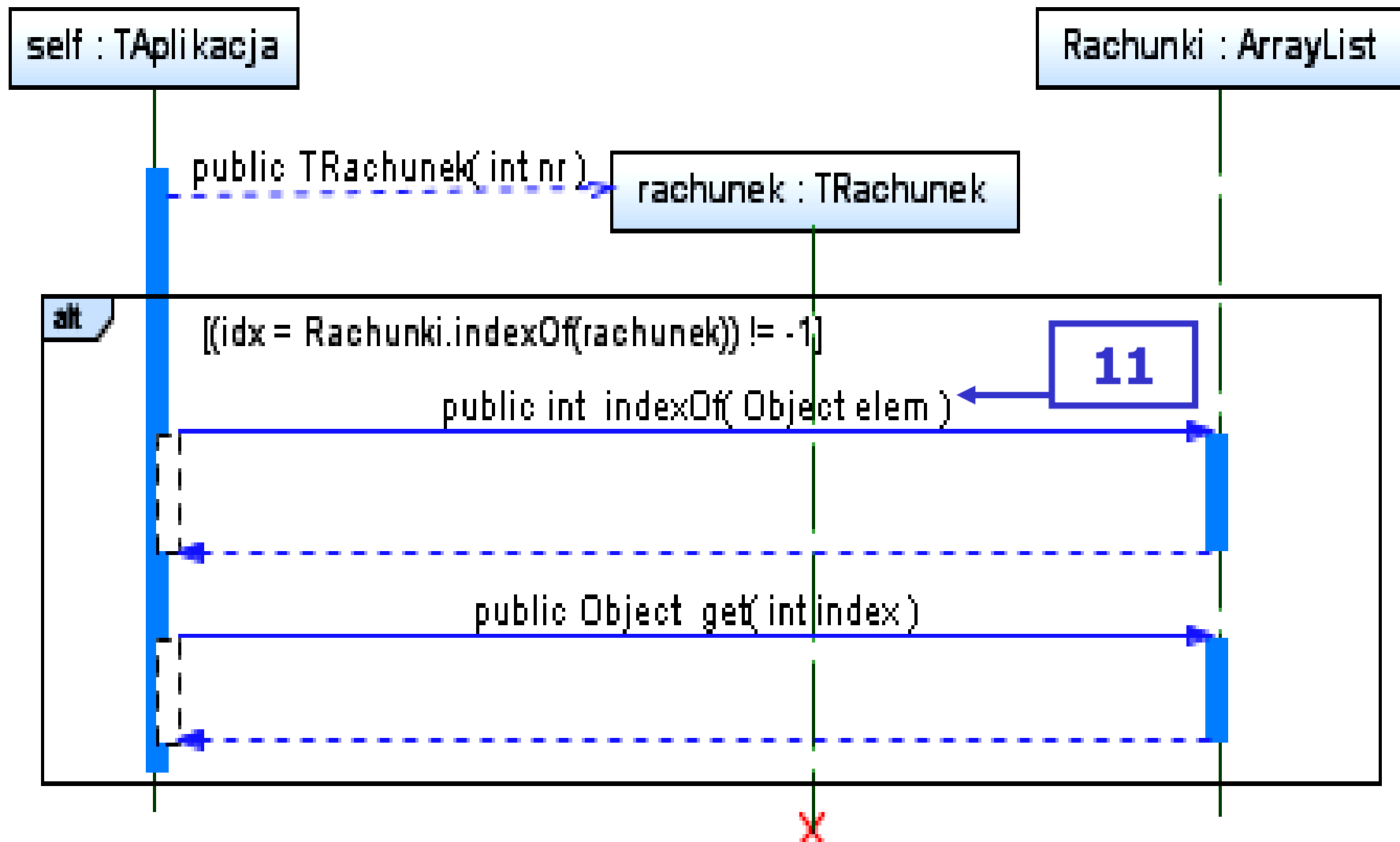
Projekt przypadku użycia  
„ **Szukanie rachunku** ”

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))



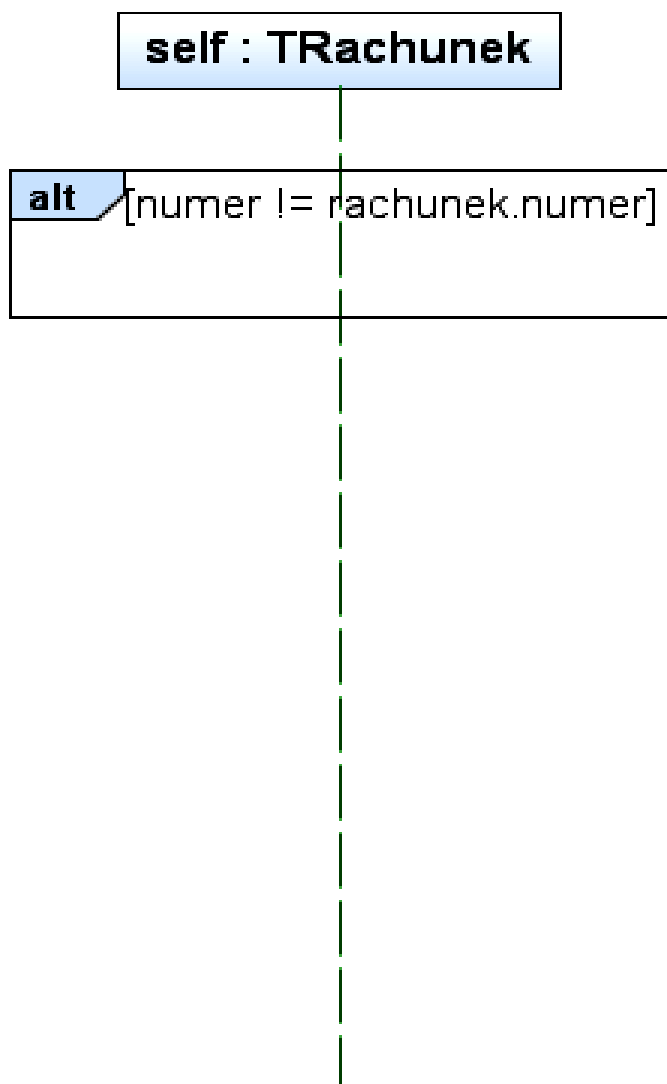
```
//TAplikacja
```

```
static private ArrayList <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;  
}
```

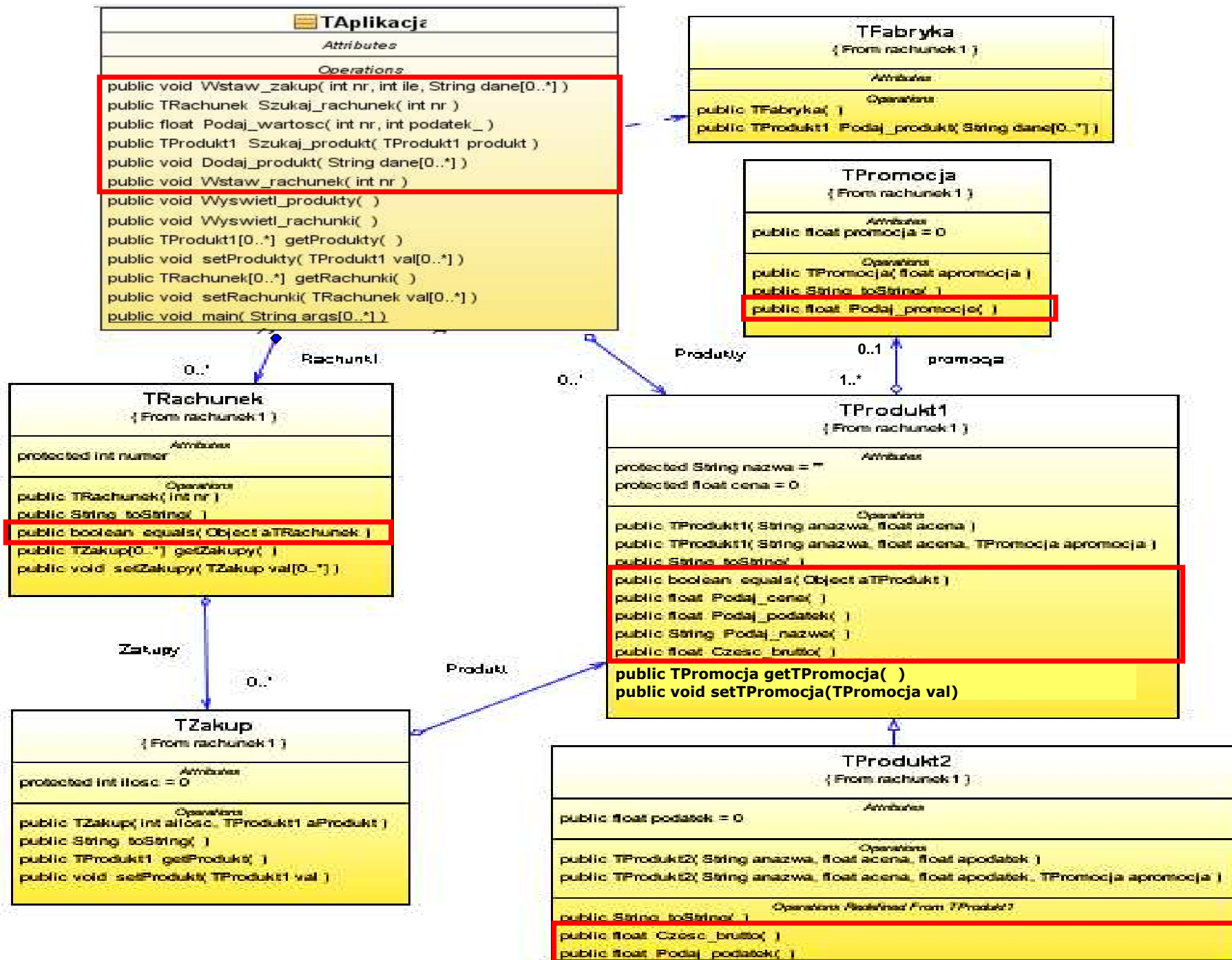


## (11) boolean TRachunek::equals(Object rachunek)



```
//TRachunek
```

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



Projekt przypadku użycia

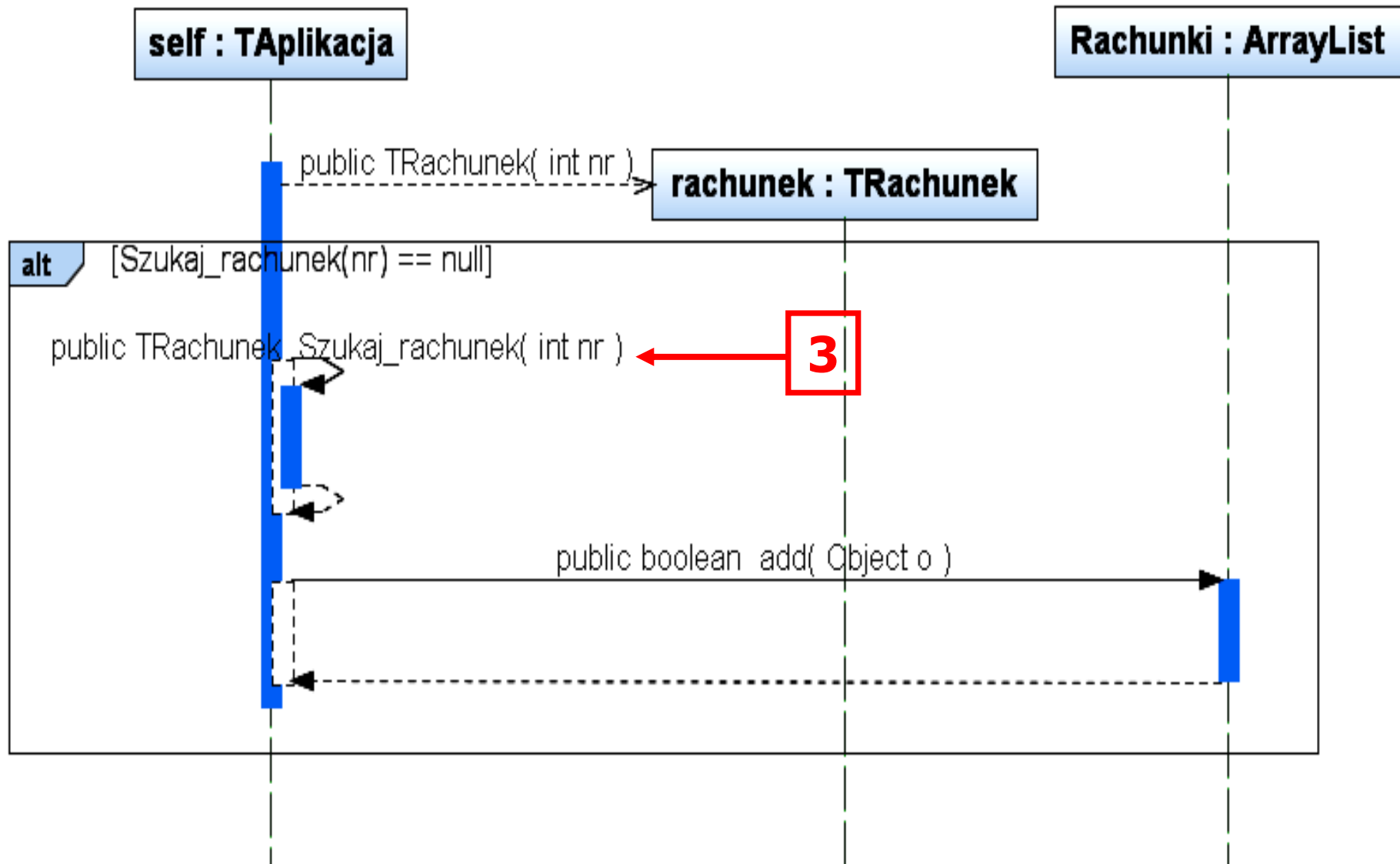
„ **Wstawianie nowego rachunku** ”

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
```

```
static private ArrayList <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;  
    StringBuffer sb = new StringBuffer();  
    sb.append(" Rachunek : ");  
    sb.append(numer + "\n");  
    Iterator<TZakup> it = Zakupy.iterator();  
    while (it.hasNext()) {  
        z = it.next();  
        sb.append(z.toString() + "\n"); }  
    return sb.toString(); }
```

```
//TZakup
```

```
public String toString() {  
    StringBuffer sb =  
        new StringBuffer();  
    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());  
    }  
}  
}
```



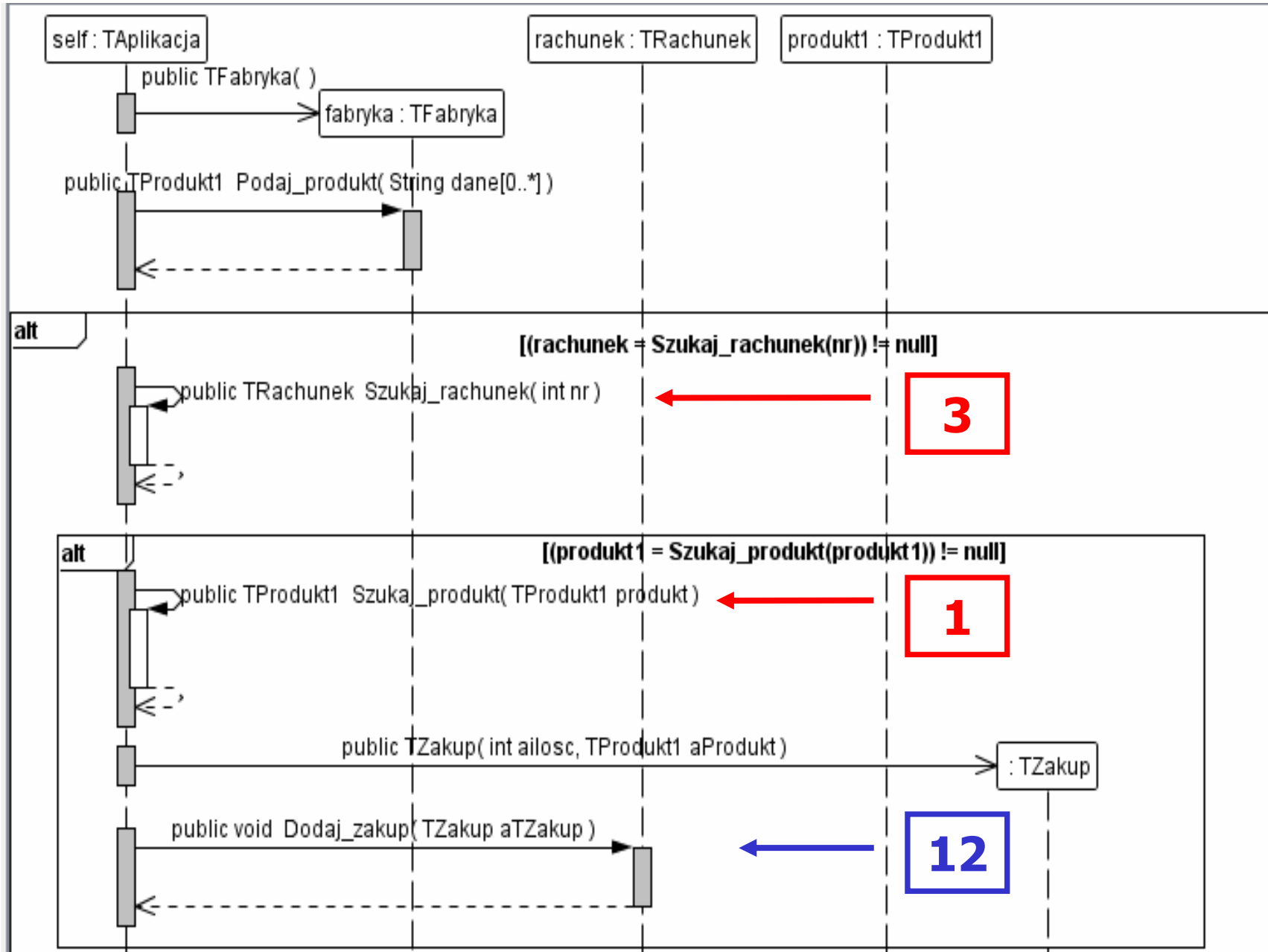
```
Command Prompt
Produkty
nazwa : 1 cena : 1.0
nazwa : 2 cena : 2.0
nazwa : 3 cena : 3.42 podatek : 14.0
nazwa : 4 cena : 4.88 podatek : 22.0
nazwa : 5 cena : 0.7 promocja : 30.0
nazwa : 6 cena : 0.9 promocja : 55.0
nazwa : 7 cena : 3.99 promocja : 30.0 podatek : 3.0
nazwa : 8 cena : 6.48 promocja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

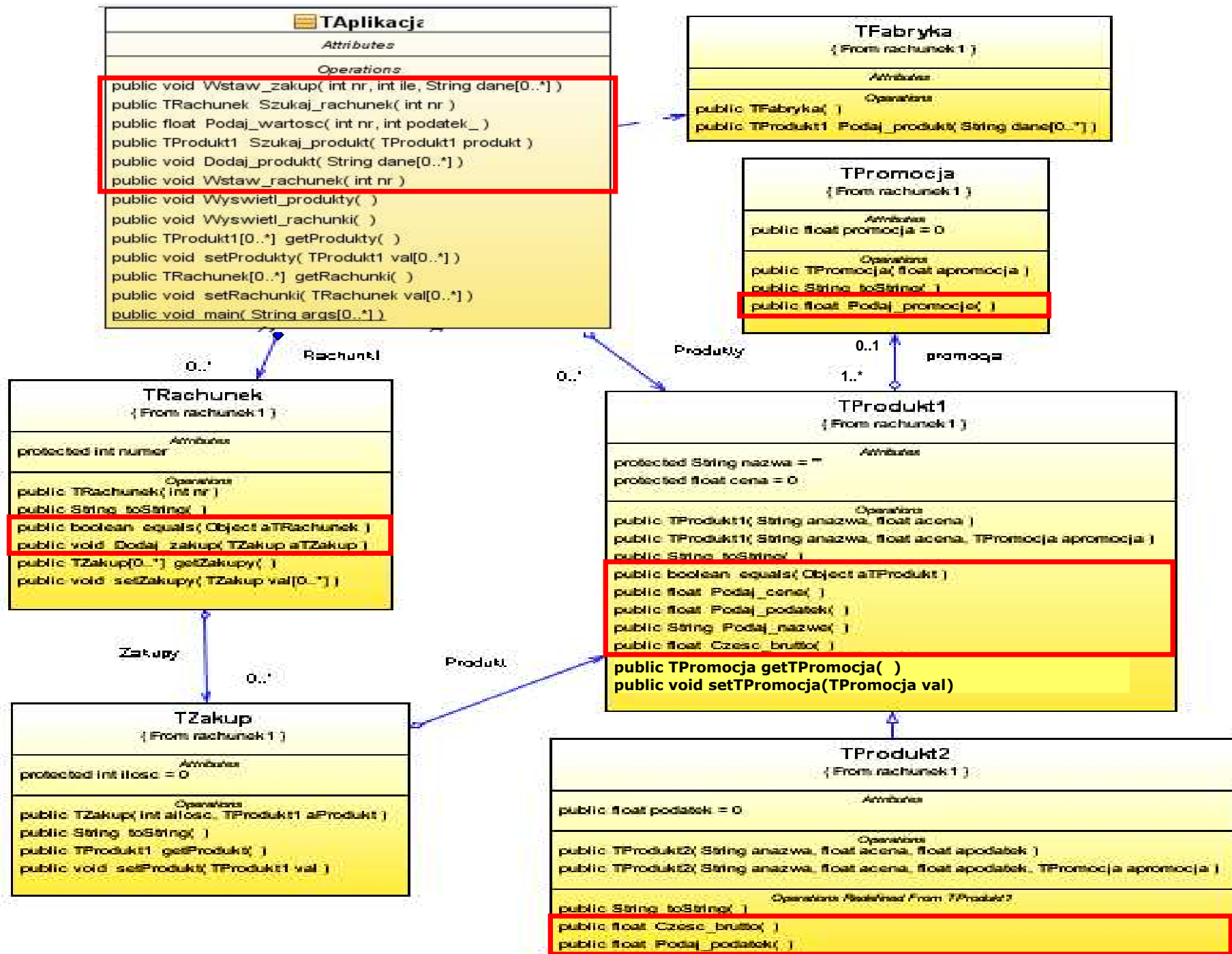
# (5) Wstawianie nowego zakupu

(void TApplikacja::Wstaw\_zakup (int nr, int ailosc, String dane[]))

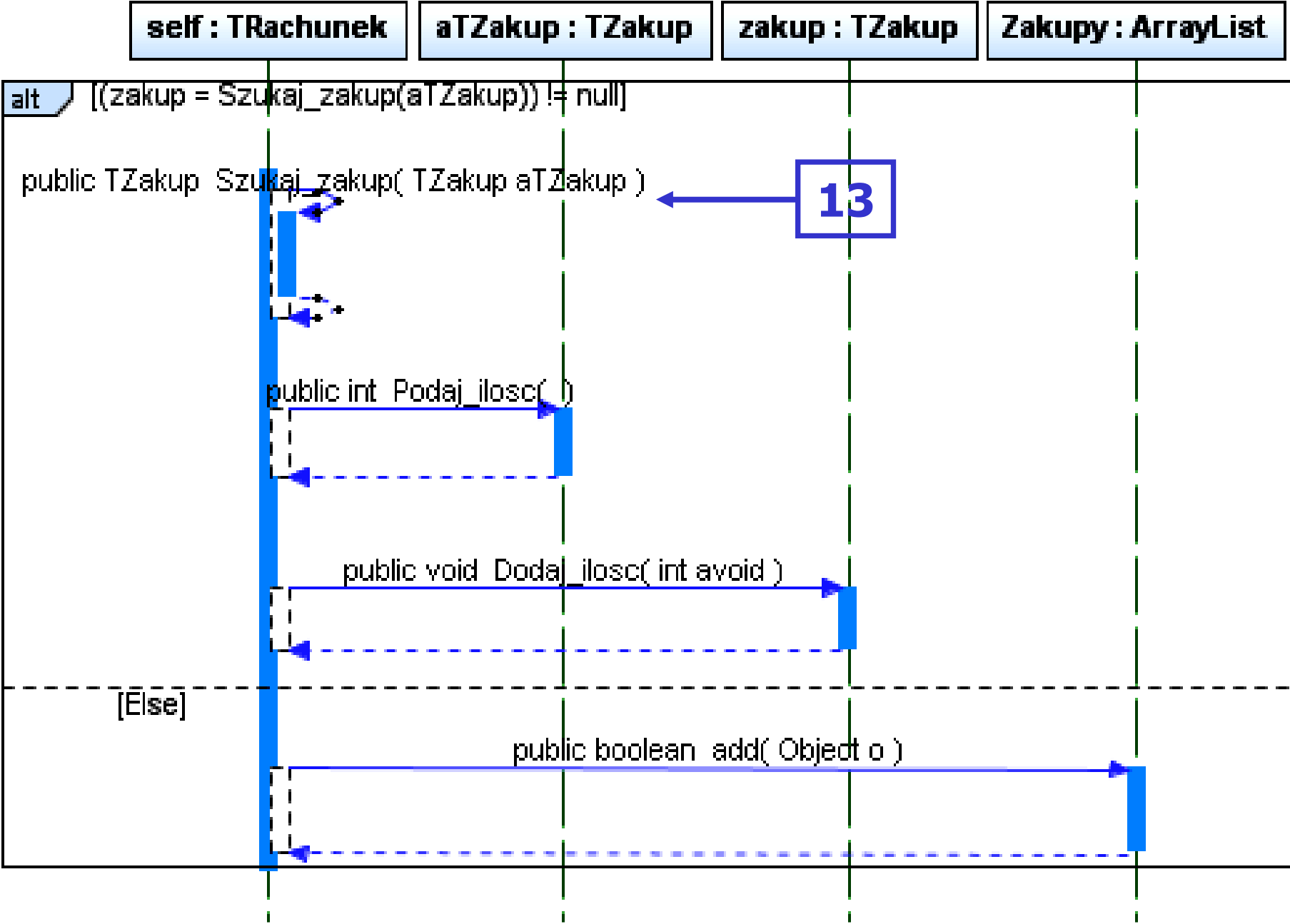


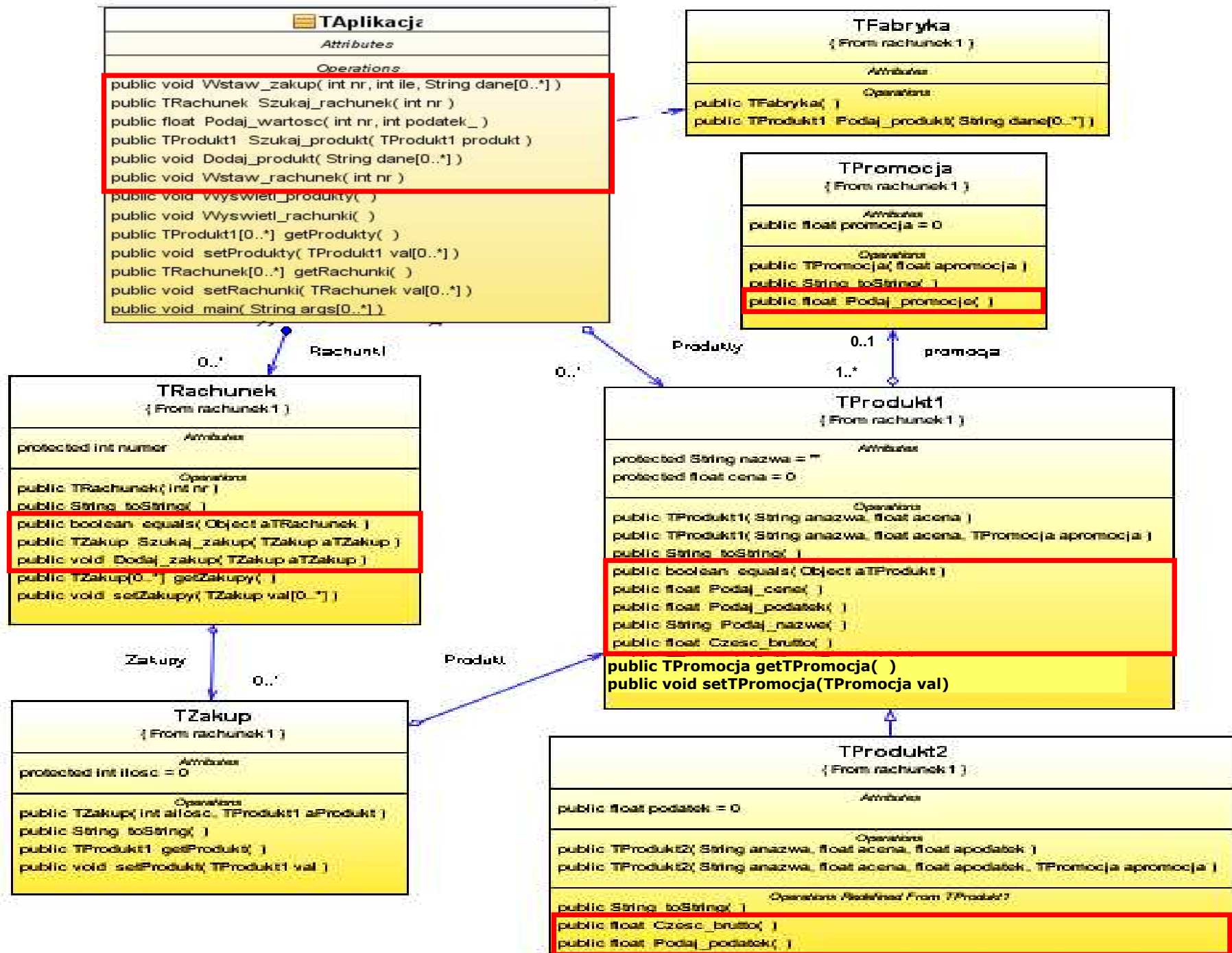
## //TAplikacja

```
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)





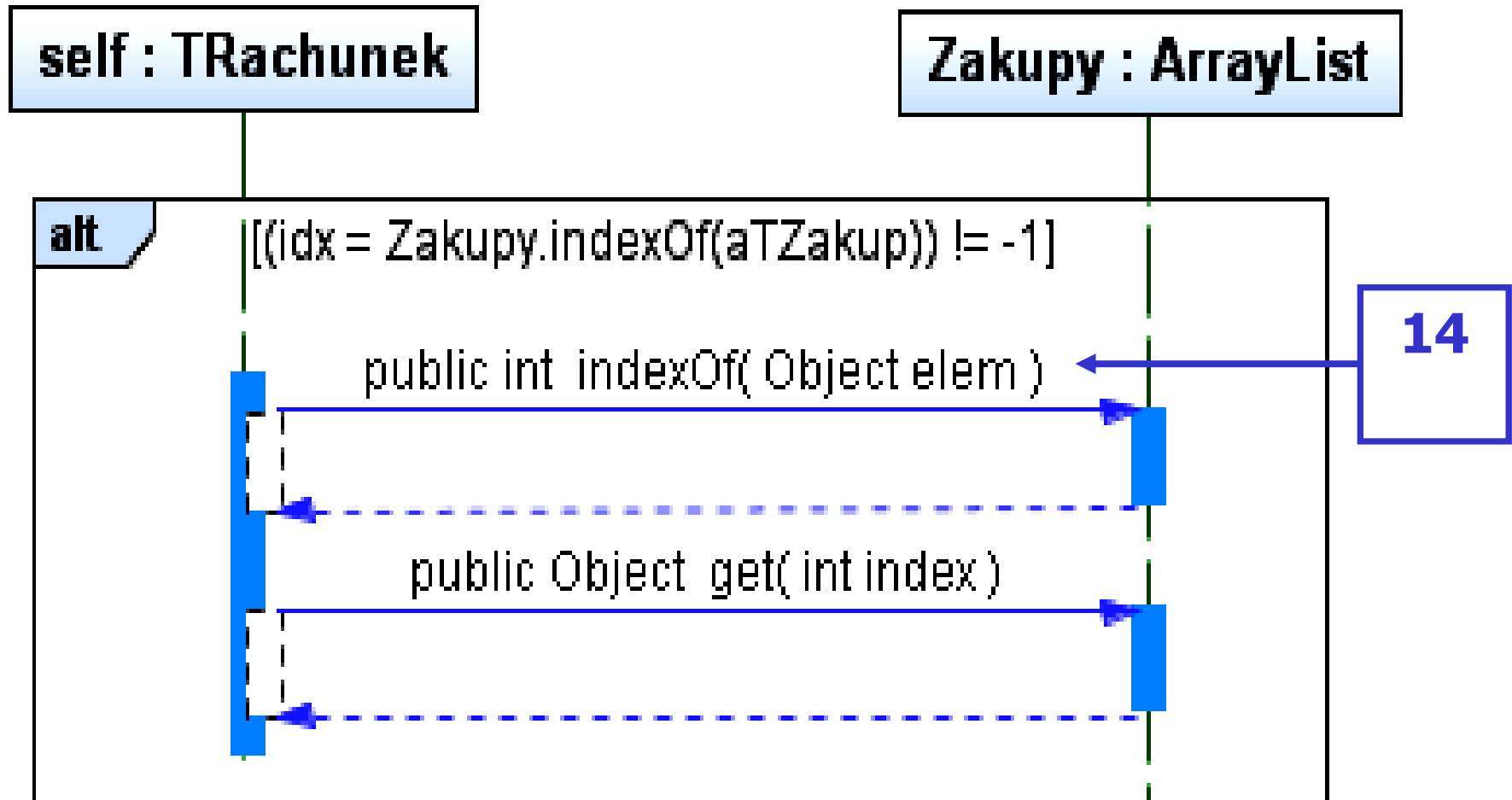
```
//TRachunek
```

```
private ArrayList<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)

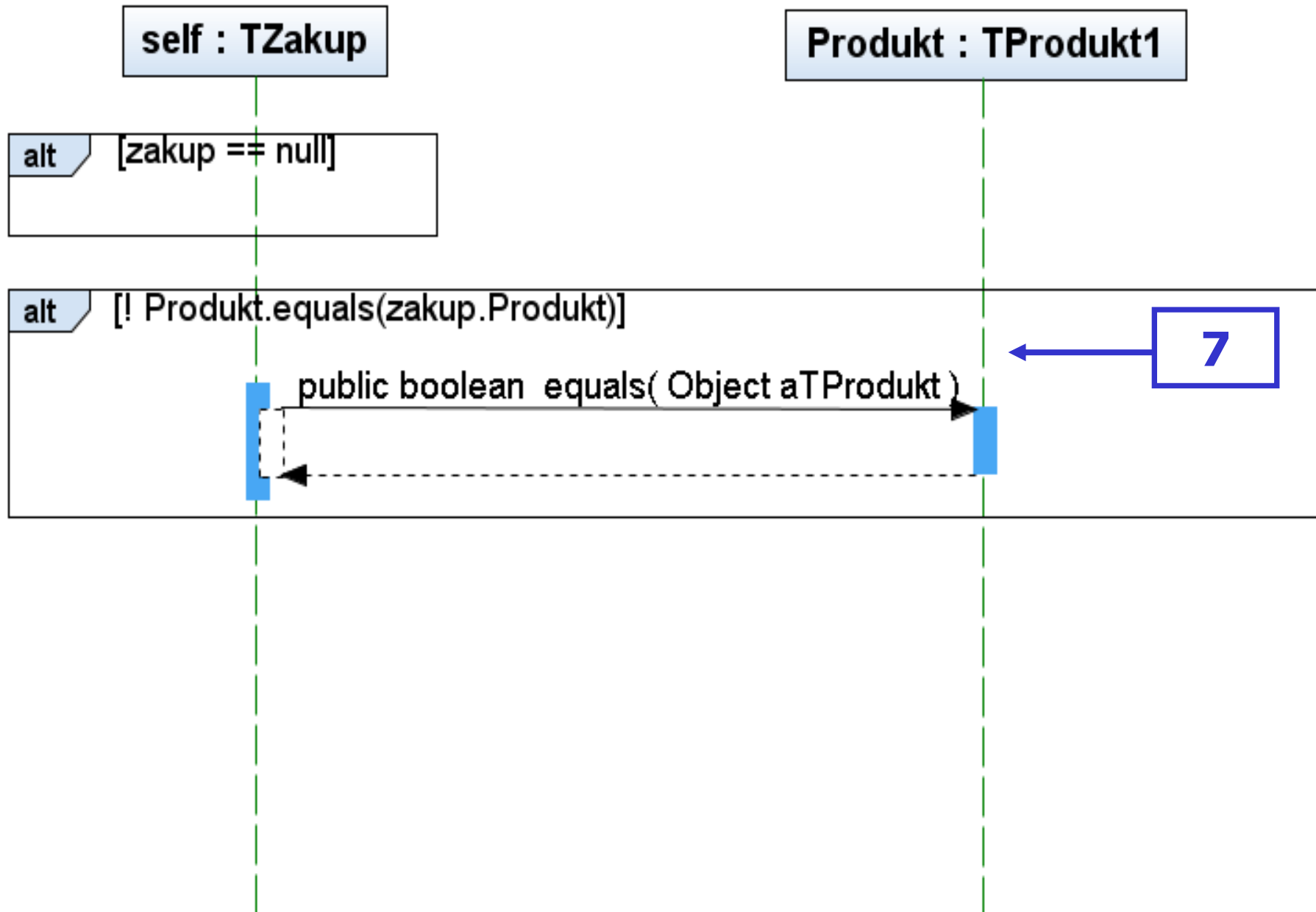


```
//TRachunek
```

```
private ArrayList<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;
```

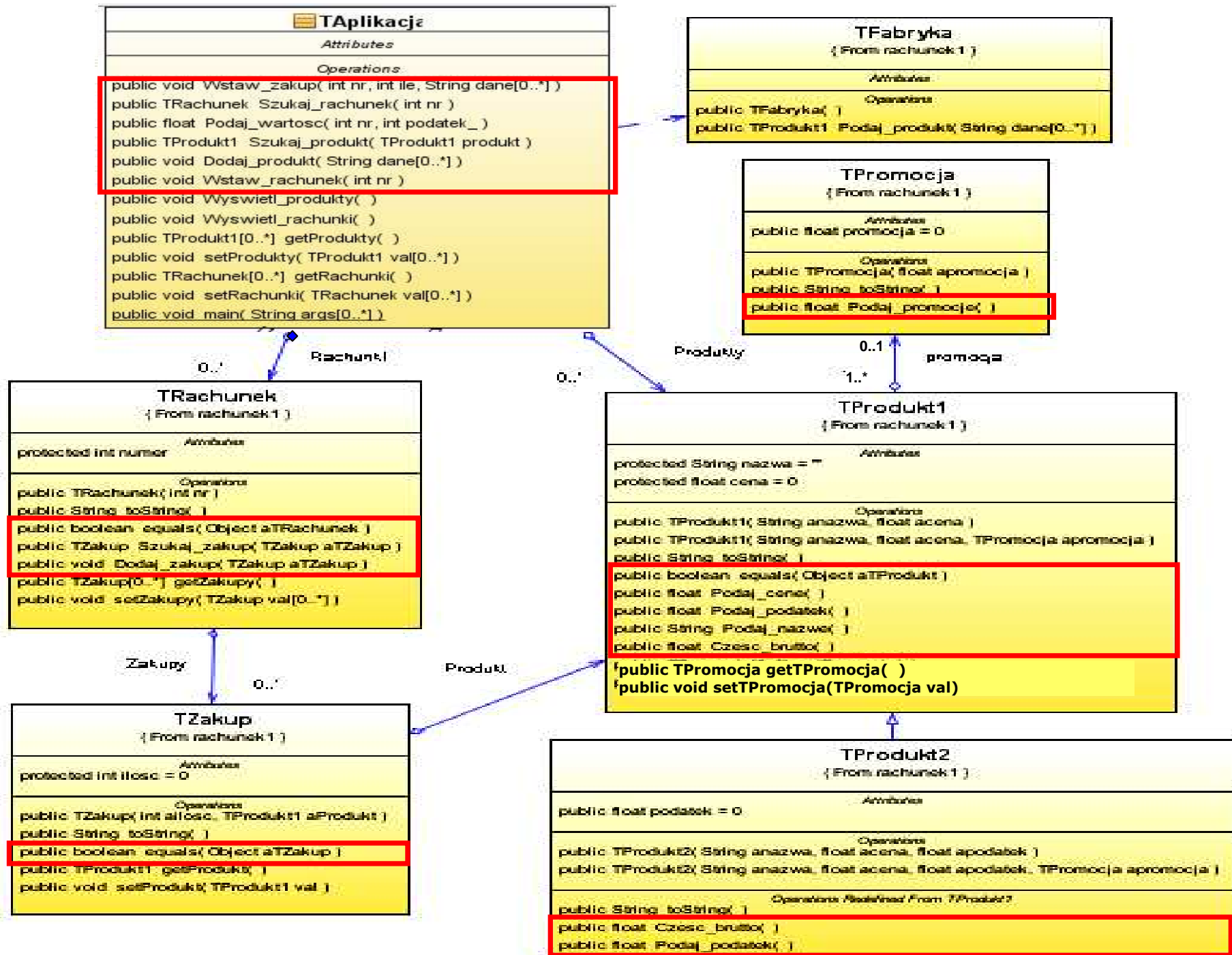
```
    boolean bStatus = true;
```

```
    if ( !Produkt.equals(zakup.Produkt) )
```

```
        bStatus = false;
```

```
    return bStatus;
```

```
}
```



```
//TZakup
```

```
public void Dodaj_ilosc ( int avoid)
```

```
{
```

```
    ilosc+=avoid;
```

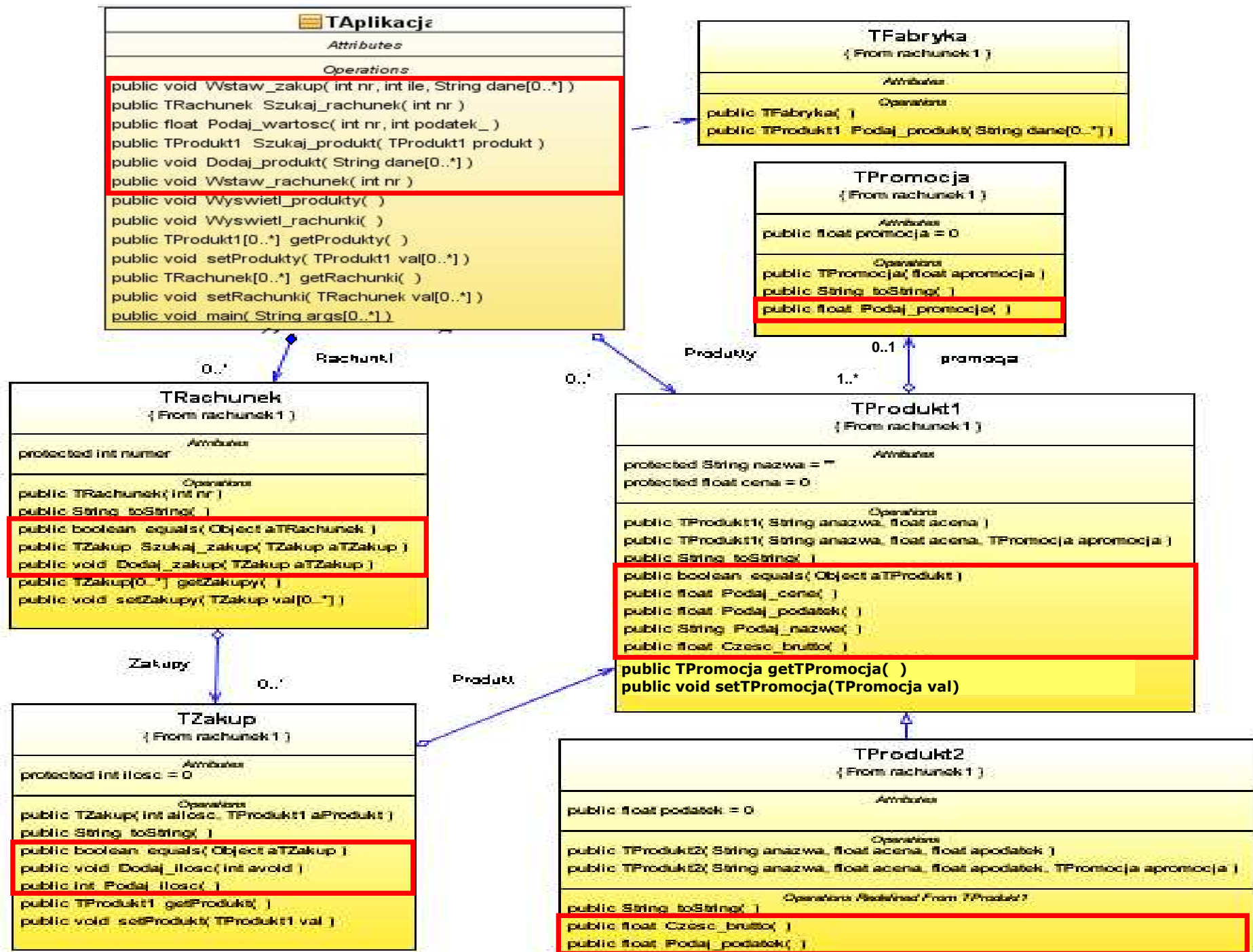
```
}
```

```
public int Podaj_ilosc ()
```

```
{
```

```
    return ilosc;
```

```
}
```



//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()); }
}
}
```



## 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
```

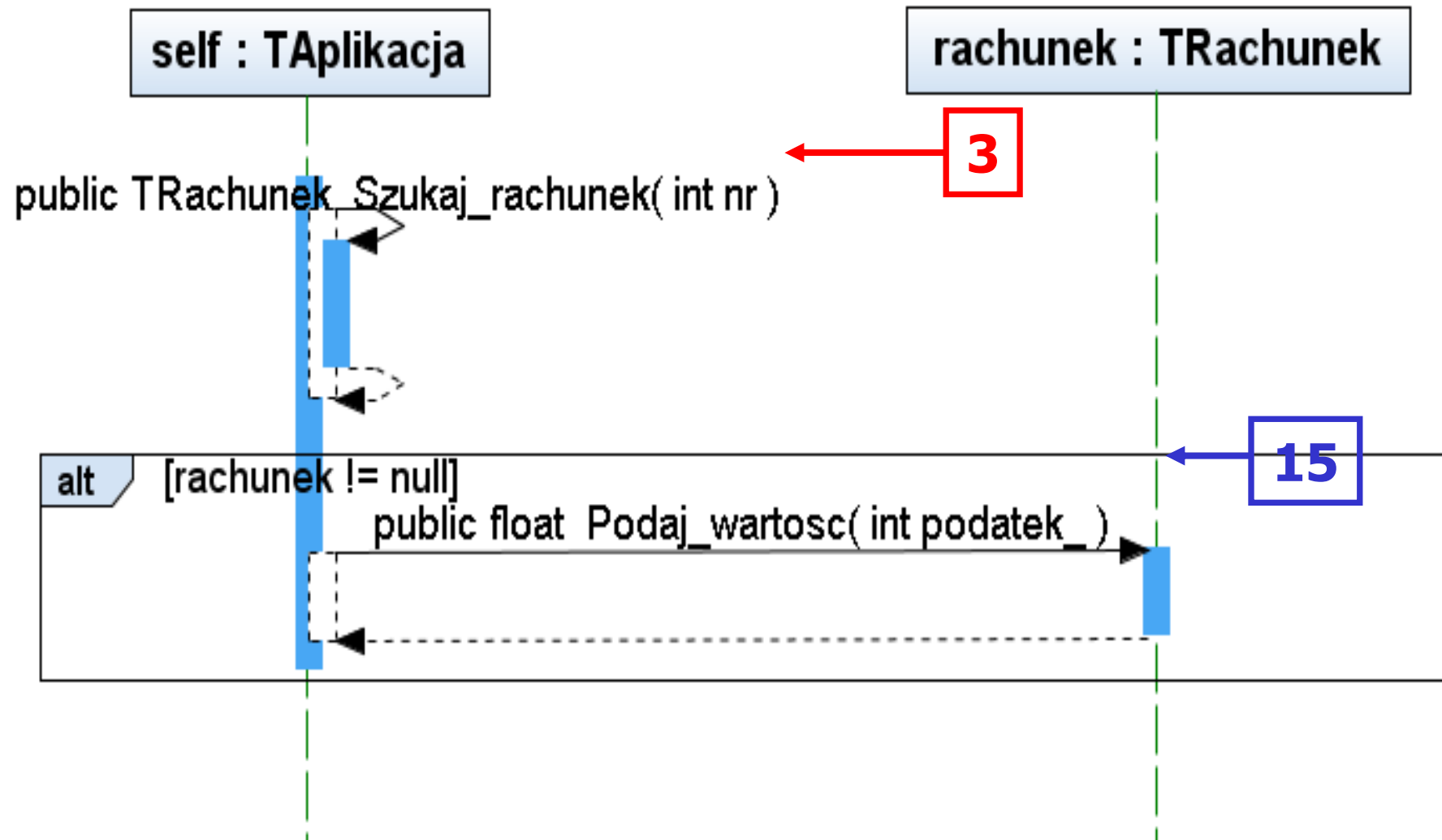
Projekt przypadku użycia  
**„Obliczanie wartości rachunku”**

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

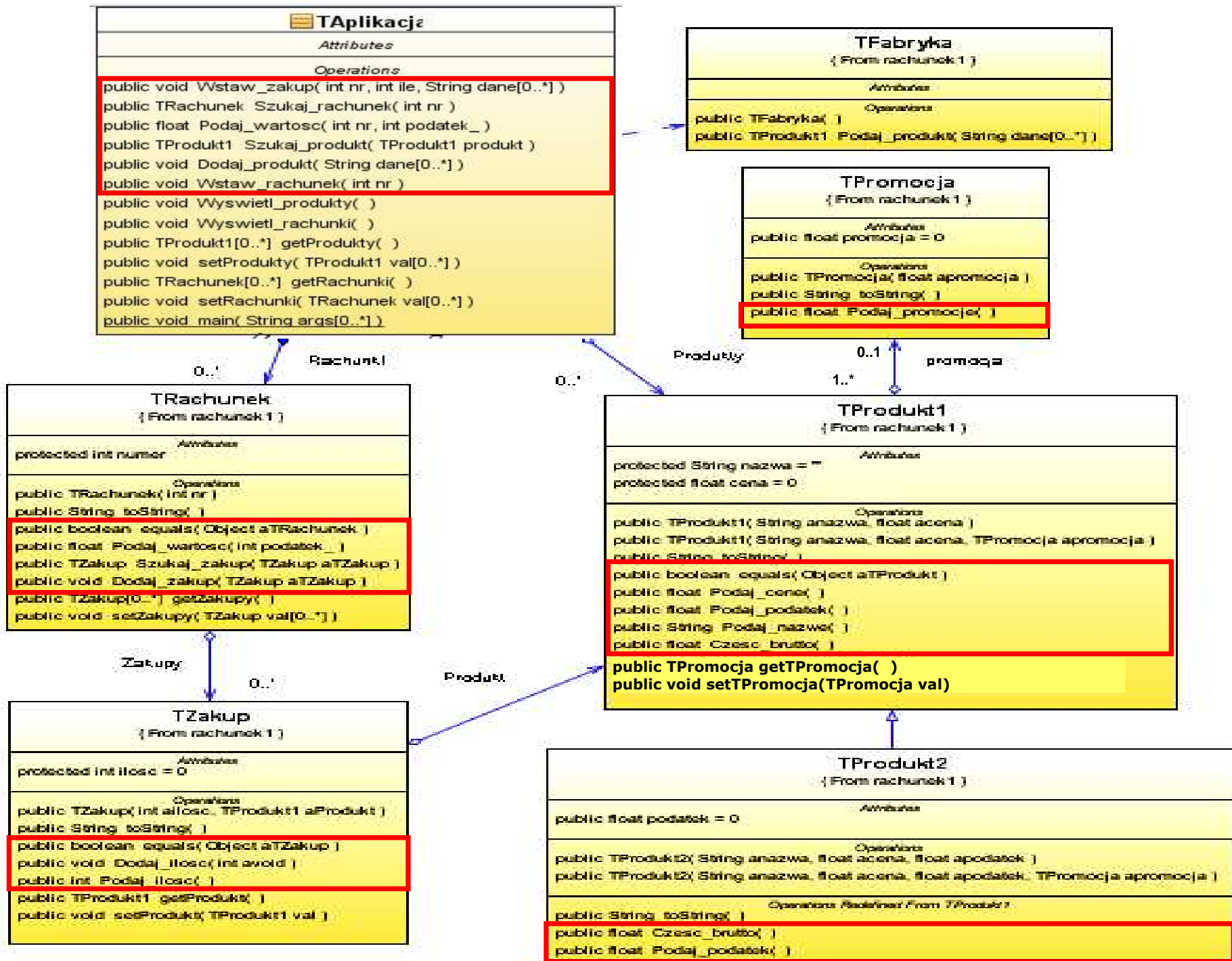
## (6) Obliczanie wartosci rachunku

(float TAplikacja::Podaj\_wartosc(int nr, int podatek\_))

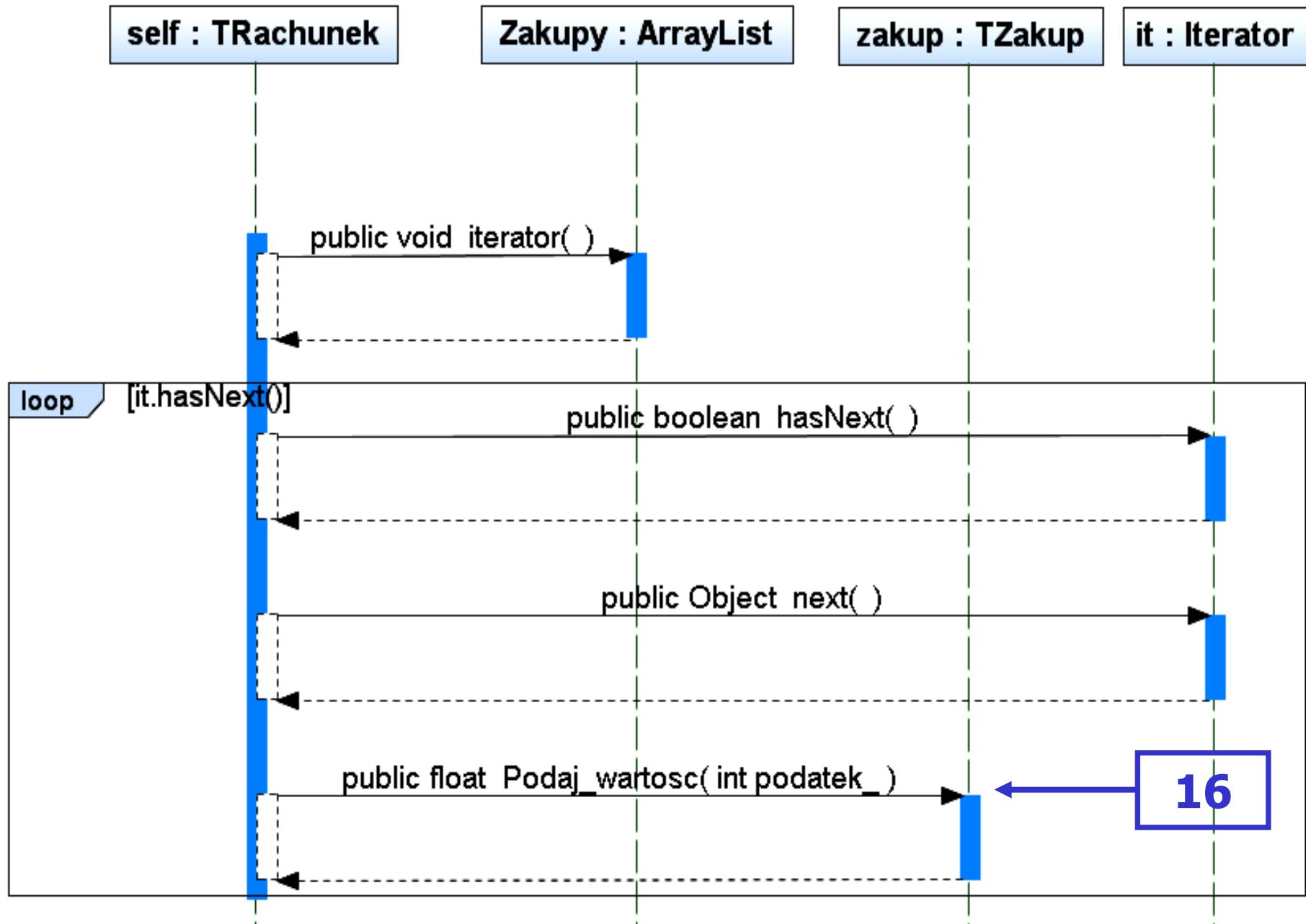


```
//TAplikacja
```

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



## (15) float TRachunek::Podaj\_wartosc(int podatek\_)

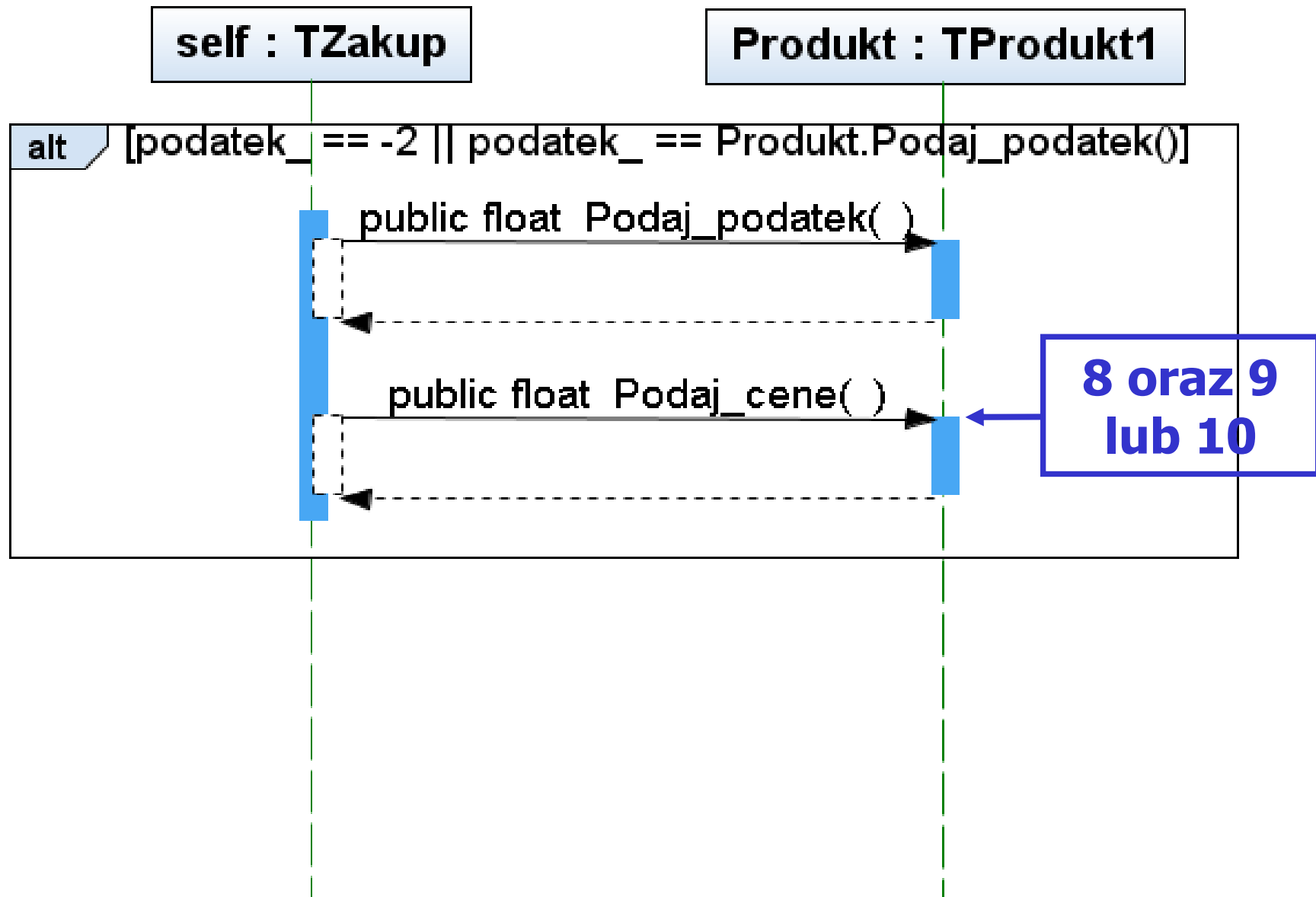


```
//TRachunek
```

```
private ArrayList<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_)
```

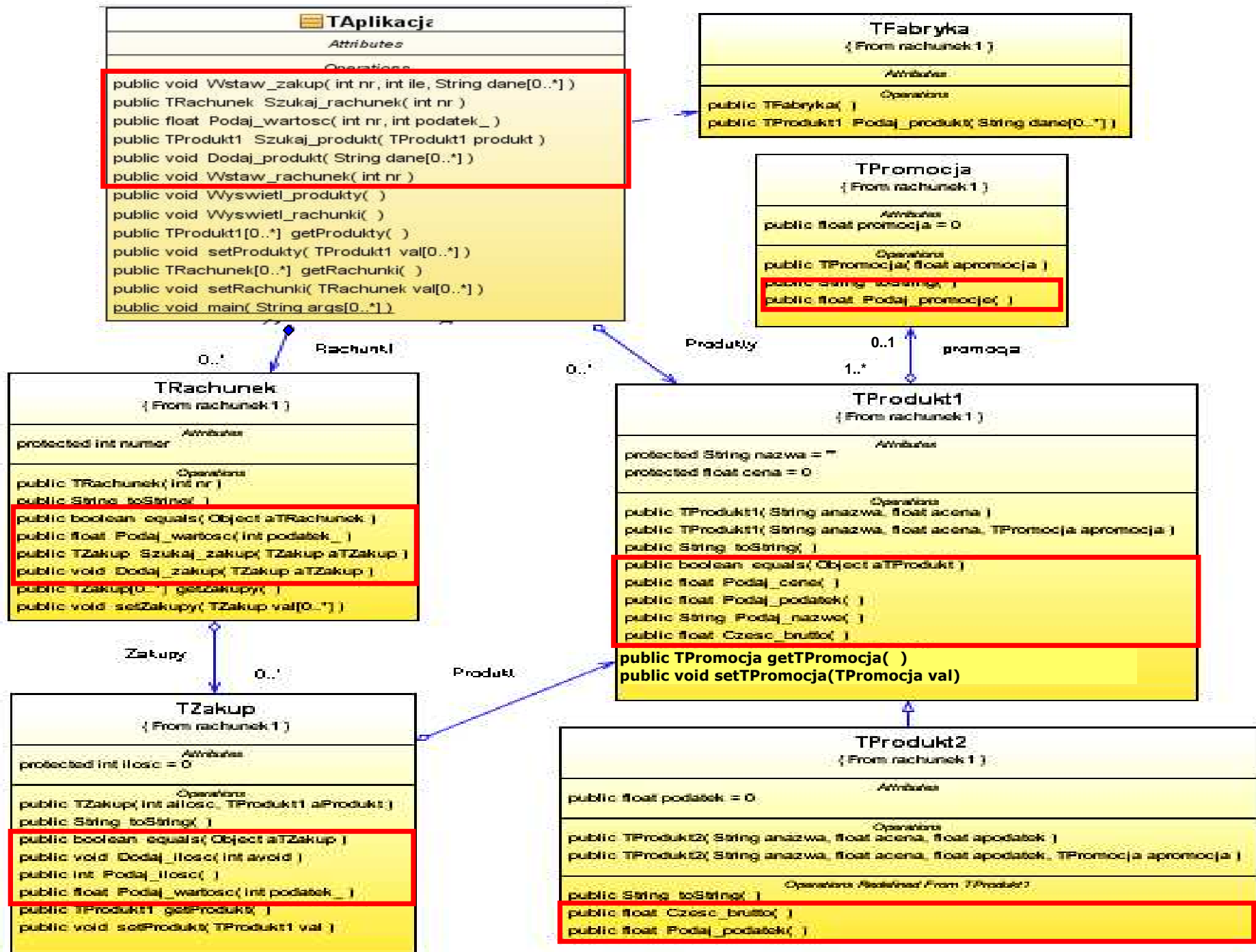
```
{
```

```
    if (podatek_ == -2 || podatek_ == Produkt.Podaj_podatek())
```

```
        return ilosc*Produkt.Podaj_cene();
```

```
    return 0F;
```

```
}
```



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

```
public String toString()  
{  
    TZakup z;  
    StringBuffer sb = new StringBuffer();  
    sb.append(" Rachunek : ");  
    sb.append(numer + "\n");  
    Iterator<TZakup> it = Zakupy.iterator();  
    while (it.hasNext()) {  
        z = it.next();  
        sb.append(z.toString() + "\n");  
    }  
    sb.append("Wartosc zakupow 0: " + Podaj_wartosc(-1) + "\n");  
    sb.append("Wartosc zakupow A: " + Podaj_wartosc(3) + "\n");  
    sb.append("Wartosc zakupow B: " + Podaj_wartosc(7) + "\n");  
    sb.append("Wartosc zakupow C: " + Podaj_wartosc(14) + "\n");  
    sb.append("Wartosc zakupow D: " + Podaj_wartosc(22) + "\n");  
    sb.append("Wartosc rachunku: " + Podaj_wartosc(-2) + "\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 implemmentacji
  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()); }
}
}
```

C:\> 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
    
```