

## **Instrukcja 6**

### **Laboratorium 8**

**Opracowanie diagramów sekwencji dla wybranych przypadków użycia reprezentujących usługi oprogramowania wynikających również z wykonanych diagramów czynności; definicja operacji klas na podstawie diagramów sekwencji w języku Java. Zastosowanie projektowych wzorców zachowania.**

**Cel laboratorium:**

Definiowanie w sposób iteracyjno - rozwojowy modelu projektowego programowania ([wykład 1](#)) opartego na:

- Modelowaniu logiki biznesowej reprezentowanej przez **wybrany bazowy przypadek użycia** za pomocą diagramów sekwencji, gdzie diagram klas pełni rolę struktury komunikacji wykorzystanej podczas tworzenia diagramów sekwencji. Ten model i implementacja przypadku użycia powinien stanowić bazę operacji stosowanych w kolejnych iteracjach. Należy definiować operacje i atrybuty kolejnej klasy (dziedziczenie, powiązania i agregacje) na diagramie klas zidentyfikowanej w wyniku modelowania kolejnego przypadku użycia i wykonanie scenariusza tego przypadku użycia za pomocą diagramu sekwencji.
  - Implementacja modelu projektowego wybranego przypadku użycia za pomocą języka Java SE.
1. Zdefiniować diagramy sekwencji operacji reprezentujących scenariusze poszczególnych przypadków użycia umieszczając je w projekcie UML założonym podczas realizacji instrukcji 2 i uzupełnianym podczas realizacji instrukcji 3-5.
  2. W projekcie UML należy automatycznie uzupełniać definicję klas na diagramie klas podczas modelowania kolejnych operacji za pomocą diagramów sekwencji. Należy rozwijać diagram klas utworzony podczas realizacji instrukcji 5.
  3. Podzielić ten proces modelowania na kilka iteracji. Należy wykryć pierwszy przypadek użycia, którego wynik wspiera działanie kolejnego modelowanego przypadku użycia w kolejnej iteracji ([wykład4](#), **Dodatek 1 instrukcji**). Ten wykryty przypadek użycia należy modelować w 1-ej iteracji procesu projektowania. Podobnie należy wybierać kolejne przypadki użycia do kolejnych iteracji.
  4. Należy systematycznie uzupełniać kod programu typu **Java Class Library** w projekcie założonym podczas realizacji instrukcji 5.
  5. Informacje niezbędne do modelowania oprogramowania za pomocą klas i sekwencji (tworzenia modelu projektowego) z wykorzystaniem wzorców projektowych podane zostały w wykładach: [wykład 3](#), [wykład4](#), [wykład 5-część 1](#), [wykład5-część2](#).

**Uwaga:**

Notacje stosowane na diagramie klas i sekwencji w Dodatku 1, odpowiadające składni składowych klas w języku Java, różnią się od notacji przedstawionych w instrukcji 1, prezentujących diagramy wykonane w środowisku VP CE. Oczywiście, w realizowanym projekcie w ramach laboratoriów należy zastosować odpowiadające notacje proponowane w środowisku VP CE.

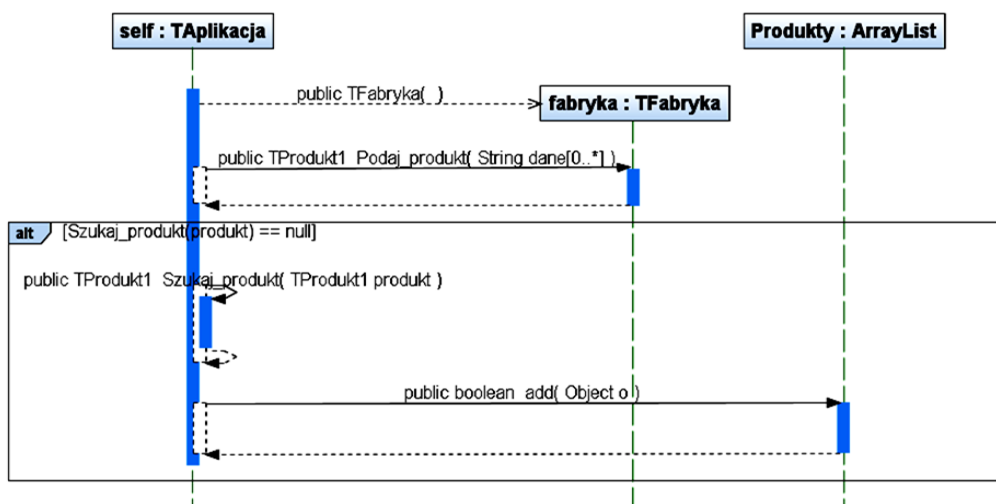
## Dodatek 1

Przykład modelowania i implementacji przypadków użycia za pomocą diagramów sekwencji oraz diagramów klas i pakietów. Zastosowanie projektowych wzorców strukturalnych, wytwórczych i czynnościowych (cd. z instrukcji 2 - 5). Prezentowany dalej kod jest uzyskany na drodze „inżynierii wprost” oraz proponowane uzupełnienia kodu w p. 6 są dodawane do kodu programu wykonanego w p.1.4. Dodatku 1 do Instrukcji 5. Składnia

### 1-a iteracja: modelowanie przypadku użycia **PU Wstawianie nowego produktu**

1. Modelowanie i implementacja operacji `void Dodaj_produkt(String [] dane)` w klasie **TAplikacja**.

- 1.1. Diagram sekwencji operacji:



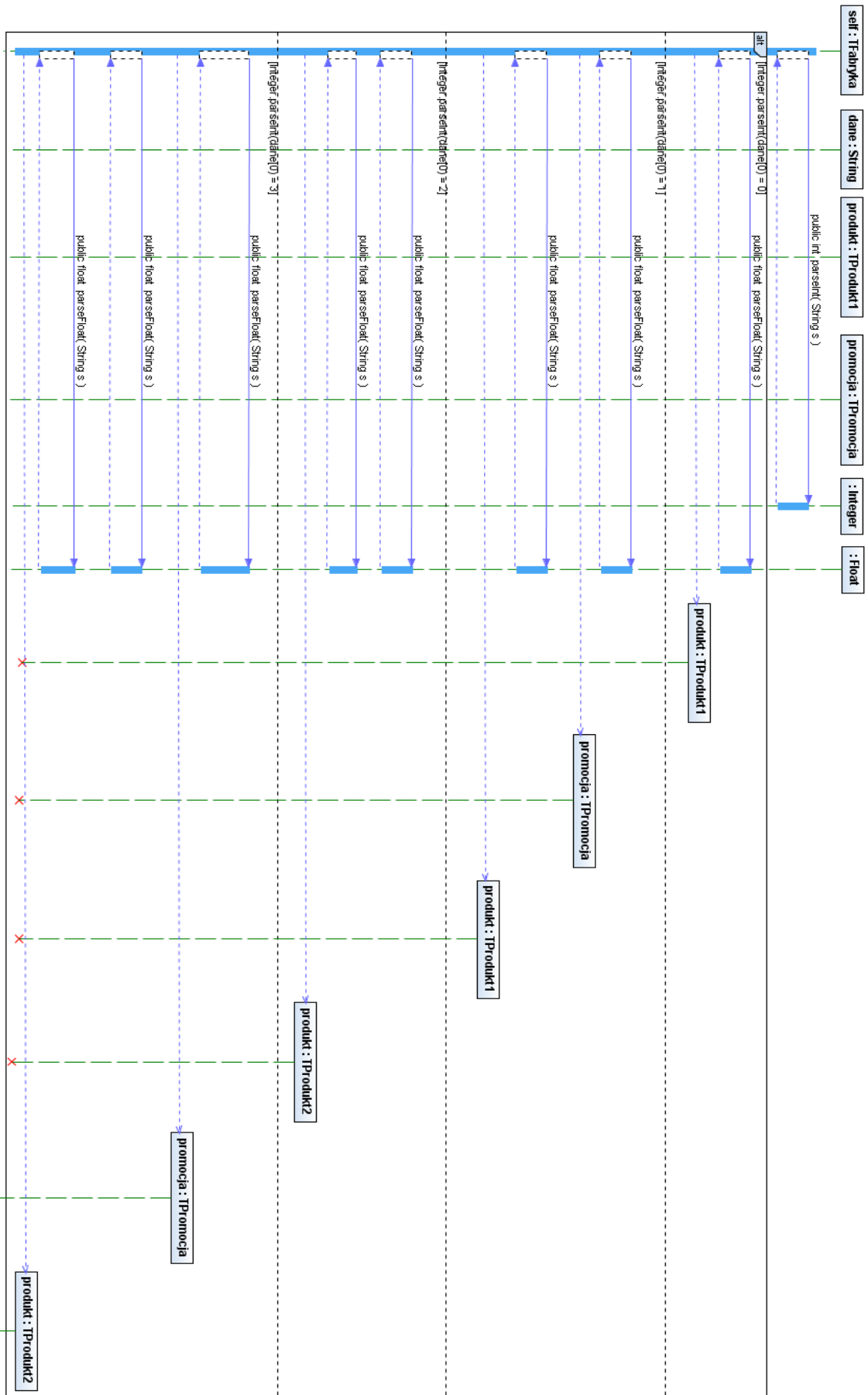
- 1.2. Kod operacji:

```

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

2. Modelowanie i implementacja operacji `TProdukt1 Podaj_produkt(String dane[])` klasy **TFabryka** tworząca 4 różne warianty obiektów: typu **TProdukt1** bez obiektu typu **TPromocja**, typu **TProdukt1** z obiektem typu **TPromocja**, typu **TProdukt2** bez obiektu typu **TPromocja**, typu **TProdukt2** z obiektem typu **TPromocja**.

- 2.1. Diagram sekwencji operacji:



## 2.2. Kod operacji:

```

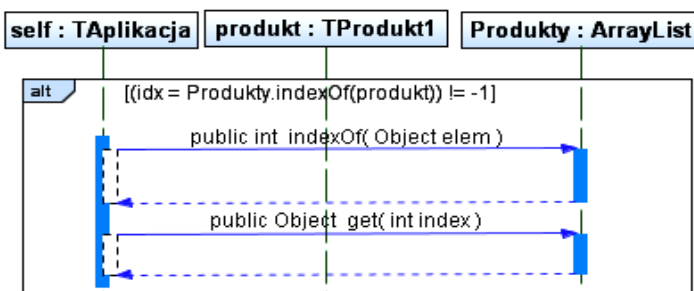
package rachunek1;
class TFabryka {
    public TFabryka() { }

    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:
                produkt = new TProdukt2(dane[1], Float.parseFloat(dane[2]), Float.parseFloat(dane[3]));
                break;
            case 3:
                promocja = new TPromocja(Float.parseFloat(dane[4]));
                produkt = new TProdukt2(dane[1], Float.parseFloat(dane[2]), Float.parseFloat(dane[3]), promocja);
                break;
        }
        return produkt;
    }
}

```

3. Modelowanie i implementacja operacji **TProdukt1 Szukaj\_produkt (TProdukt1 produkt)** w klasie **TAplikacja** – modelowanie i implementacja **PU Szukanie produktu**.

## 3.1. Diagram sekwencji operacji:



## 3.2. Kod operacji:

```

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

```

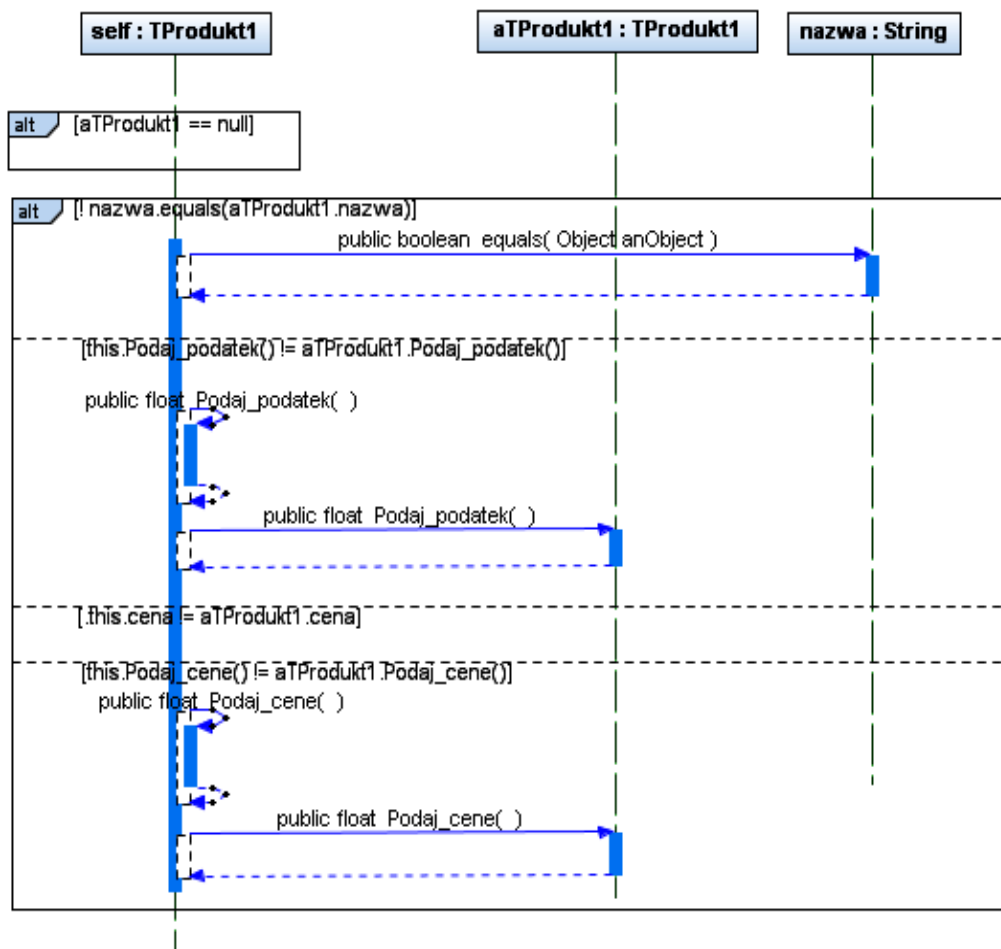
4. Modelowanie i implementacja operacji **boolean equals(Object o)** wywoływanej od obiektów typu **TProdukt1** lub **TProdukt2** (linia życia typu **TProdukt1** może reprezentować oba typy instancji obiektów powiązanych dziedziczeniem - **TProdukt1** i **TProdukt2**), gdzie za pomocą tej operacji mogą porównywać się 4 różne pary obiektów typu:

- **TProdukt1** z **TProdukt1**, **TProdukt1** z **TProdukt2**,
- **TProdukt2** z **TProdukt2**, **TProdukt2** z **TProdukt1**.

4.1. Znana metoda **int indexOf(Object o)** z klasy typu **ArrayList**, jest wywołana w metodzie **Szukaj\_produkt** (p.3). Wymaga ona zaprojektowania metody **equals** w klasie **TProdukt1**, dziedziczonej przez klasę **TProdukt2**:

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

4.2. Diagram sekwencji operacji **boolean equals(Object o)** w klasie **TProdukt1**:



## 4.3. Kod operacji w klasie TProdukt1:

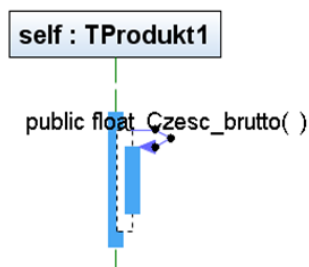
```

@Override
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()) //p.4.4.1
                    bStatus = false;
    return bStatus;
}

```

4.4. Diagramy sekwencji oraz kod wirtualnych operacji wywoływanych na diagramie sekwencji oraz w kodzie operacji **equals**, np. operacja **Czesc\_brutto()** może korzystać z różnych algorytmów przeliczania promocji pobranej od obiektu typu **TPromocja** w klasie **TProdukt1** i **TProdukt2**, które są niewidoczne na diagramie sekwencji. Klasa **TPromocja** pełni rolę **wzorca czynnościowego Strategia**.

4.4.1. Diagram sekwencji operacji **float Podaj\_cene()** w klasie **TProdukt1**, która wywołuje metodę wirtualną **Czesc\_brutto()**, zdefiniowaną w klasie **TProdukt2**:



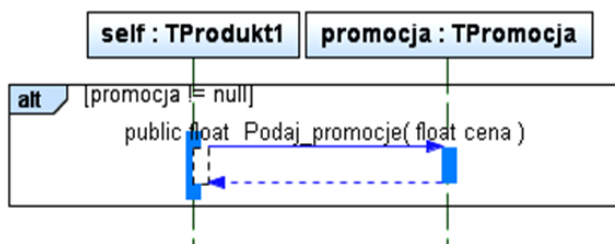
4.4.2. Kod metody **float Podaj\_cene()** w klasie **TProdukt1**:

```

public float Podaj_cene ()
{
    return cena + Czesc_brutto();
}

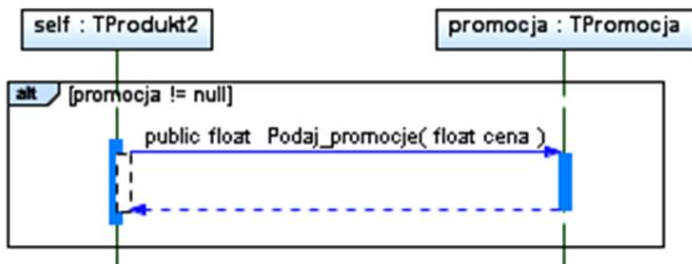
```

4.4.3. Diagram sekwencji metody **float Czesc\_brutto()** w klasie **TProdukt1**:



4.4.4. Kod metody **float Czesc\_brutto()** w klasie **TProdukt1**:

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

4.4.5. Diagram sekwencji metody **float Czesc\_brutto()** w klasie **TProdukt2**:4.4.6. Kod metody **float Czesc\_brutto()** w klasie **TProdukt2**:

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

4.4.7. Kod wirtualnych metod **float Podaj\_podatek** w klasach **TProdukt1** i **TProdukt2**, używanych w metodzie **equals** klasy **TProdukt1**, dziedziczonej przez klasę **TProdukt2**:

```
//class TProdukt1
```

```
public float Podaj_podatek ()
{ return -1; }
```

```
//class TProdukt2
```

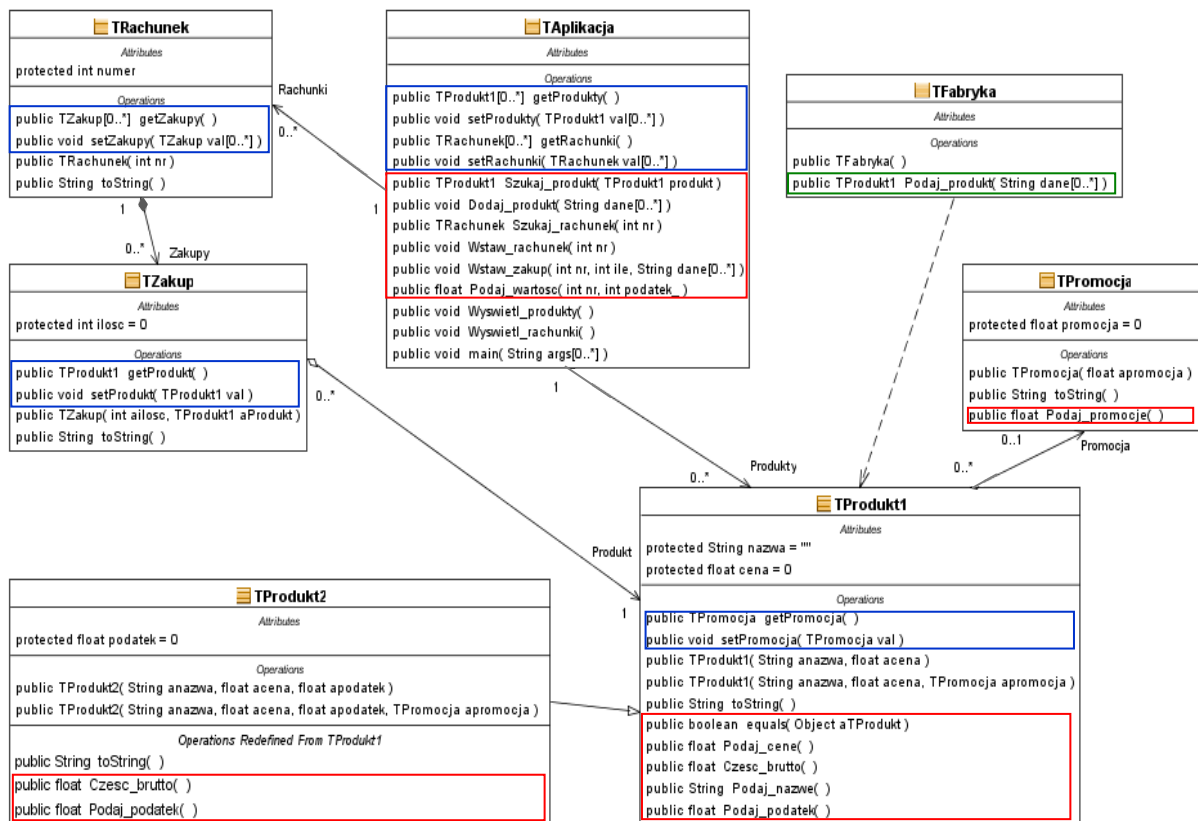
```
public float Podaj_podatek ()
{ return podatek; }
```

4.4.8. Kod metody **float Podaj\_promocję()** w klasie **TPromocja** lub klasy pochodnej wykonującej **jakiś algorytm obliczania promocji** (koncepcja **wzorca czynnościowego Strategia**) np.

```
public float Podaj_promocje ()
{ if (promocja<50)
    return promocja;
  return promocja *1.1F;
}
```



### 5. Diagram klas zawierający elementy wynikające z wykonanych diagramów sekwencji w 1-jej iteracji.



### 6. Rozszerzenie kodu źródłowego klas, dodanego do kodu wykonanego na podstawie wykonanego diagramu klas i diagramów sekwencji („inżynieria wprost”) – czyli dodanie pomocniczych metod do prezentacji wyników metod logiki biznesowej, modelowanych za pomocą diagramów sekwencji.

#### Klasa TProdukt1

```

public TProdukt1(String anazwa, float acena) {
    nazwa = anazwa;
    cena = acena;
}
public TProdukt1(String anazwa, float acena, TPromocja apromocja) {
    nazwa = anazwa;
    cena = acena;
    Promocja = apromocja;
}
@Override
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();
}
public String Podaj_nazwe() {
    return nazwa;
}
  
```

**Klasa TProdukt2**

```

public TProdukt2(String anazwa, float acena, float apodatek) {
    super(anazwa, acena);
    podatek = apodatek;
}
public TProdukt2(String anazwa, float acena, float apodatek, TPromocja apromocja) {
    super(anazwa, acena, apromocja);
    podatek = apodatek;
}
@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append(super.toString());
    sb.append(" podatek : ");
    sb.append(podatek);
    return sb.toString();
}

```

**Klasa TPromocja**

```

public TPromocja(float apromocja) {
    promocja = apromocja;
}
@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append(" promocja : ");
    sb.append(Podaj_promocje());
    return sb.toString();
}

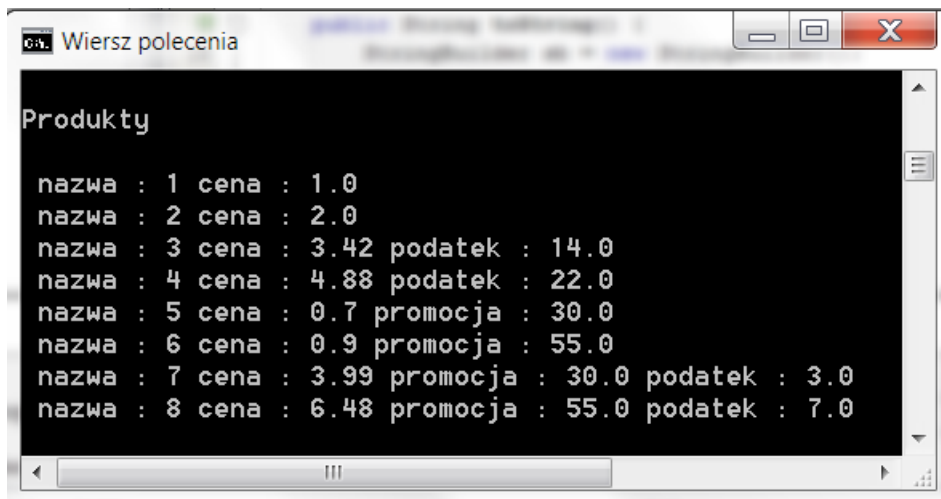
```

**Klasa TAplikacja**

```

public void Wyswietl_produkty() {
    for (TProdukt1 produkt : Produkty) {
        System.out.println(produkt);
    }
}
public static void main(String args[]) {
    TAplikacja app = new TAplikacja();
    String dane1[] = {"0", "1", "1"};
    app.Dodaj_produkty(dane1);
    String dane2[] = {"0", "2", "2"};
    app.Dodaj_produkty(dane2);
    String dane3[] = {"2", "3", "3", "14"};
    app.Dodaj_produkty(dane3);
    String dane4[] = {"2", "4", "4", "22"};
    app.Dodaj_produkty(dane4);
    String dane5[] = {"1", "5", "1", "30"};
    app.Dodaj_produkty(dane5);
    String dane6[] = {"1", "6", "2", "50"};
    String dane7[] = {"3", "7", "3", "3", "30"};
    app.Dodaj_produkty(dane6);
    app.Dodaj_produkty(dane7);
    String dane8[] = {"3", "8", "4", "7", "50"};
    app.Dodaj_produkty(dane8);
    app.Dodaj_produkty(dane7);
    System.out.println("\nProdukty\n");
    app.Wyswietl_produkty();
}

```



```
Wiersz poleceń
Produkt
nazwa : 1 cena : 1.0
nazwa : 2 cena : 2.0
nazwa : 3 cena : 3.42 podatek : 14.0
nazwa : 4 cena : 4.88 podatek : 22.0
nazwa : 5 cena : 0.7 promocja : 30.0
nazwa : 6 cena : 0.9 promocja : 55.0
nazwa : 7 cena : 3.99 promocja : 30.0 podatek : 3.0
nazwa : 8 cena : 6.48 promocja : 55.0 podatek : 7.0
```

## Dodatek 2

Tworzenie diagramów klas i sekwencji użycia w wybranym środowisku np Visual Paradigm

1. Pomoc: [Drawing class diagrams.](http://www.visual-paradigm.com/support/documents/vpumluserguide/94/2576/7190_drawingclass.html)  
([http://www.visual-paradigm.com/support/documents/vpumluserguide/94/2576/7190\\_drawingclass.html](http://www.visual-paradigm.com/support/documents/vpumluserguide/94/2576/7190_drawingclass.html))
2. Pomoc: [Drawing sequence diagrams.](http://www.visual-paradigm.com/support/documents/vpumluserguide/94/2577/7025_drawingseque.html)  
([http://www.visual-paradigm.com/support/documents/vpumluserguide/94/2577/7025\\_drawingseque.html](http://www.visual-paradigm.com/support/documents/vpumluserguide/94/2577/7025_drawingseque.html))