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**.

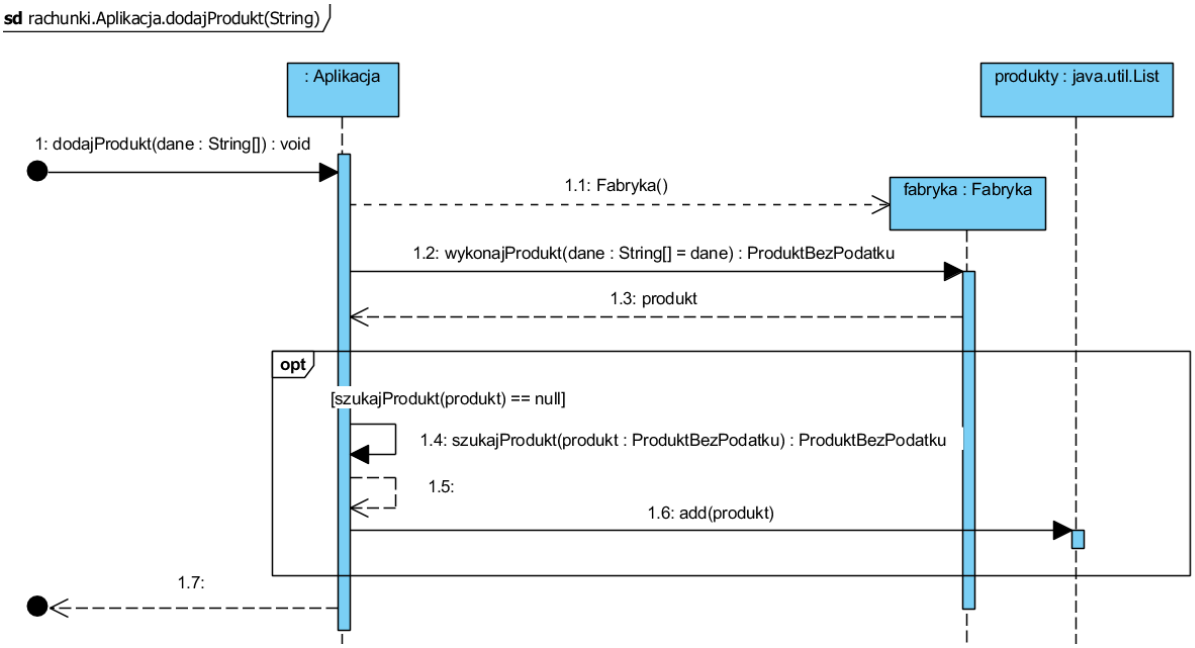
Dodatek 1

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 dodajProdukt(String [] dane)** w klasie **Aplikacja.**

   1.1. Diagram sekwencji operacji**:**

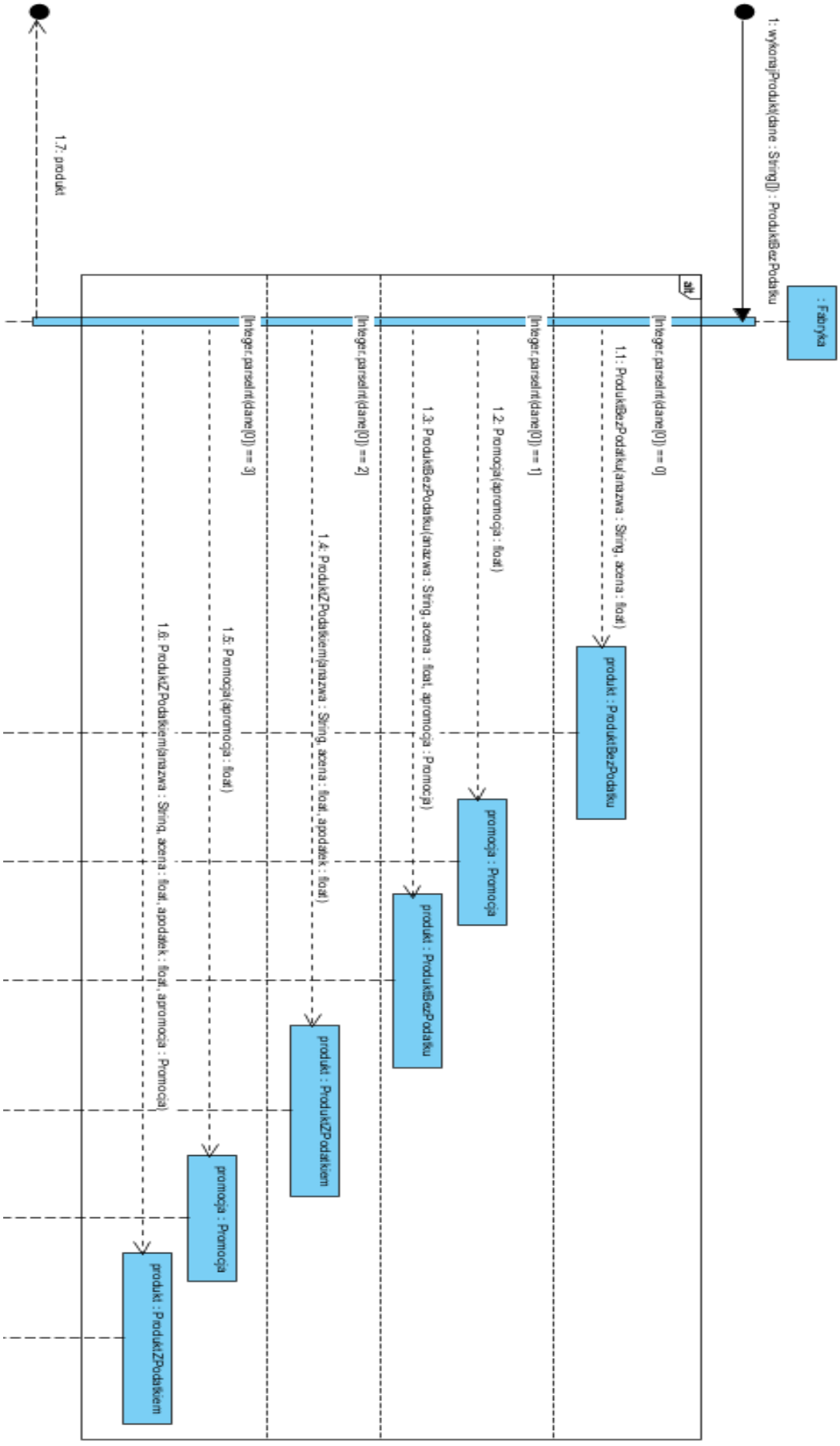


   1.2. Kod operacji**:**

```
public void dodajProdukt (String dane[])
{
  Fabryka fabryka = new Fabryka();
  ProduktBezPodatku produkt = fabryka.Podaj_produkt(dane);
   if (szukajProdukt(produkt) == null)
     produkty.add(produkt);
}
```

2. Modelowanie i implementacja operacji **ProduktBezPodatku wykonajProdukt(String dane[])** klasy **Fabryka** tworząca 4 różne warianty obiektów: typu **ProduktBezPodatku** bez utworzonego obiektu typu **Promocja**, typu **ProduktBezPodatku** z przypisanym obiektem typu **Promocja**, typu **ProduktZPodatkiem** bez utworzonego obiektu typu **Promocja**, typu **ProduktZPodatkiem** z przypisanym obiektem typu **Promocja**.

2.1. Diagram sekwencji operacji:

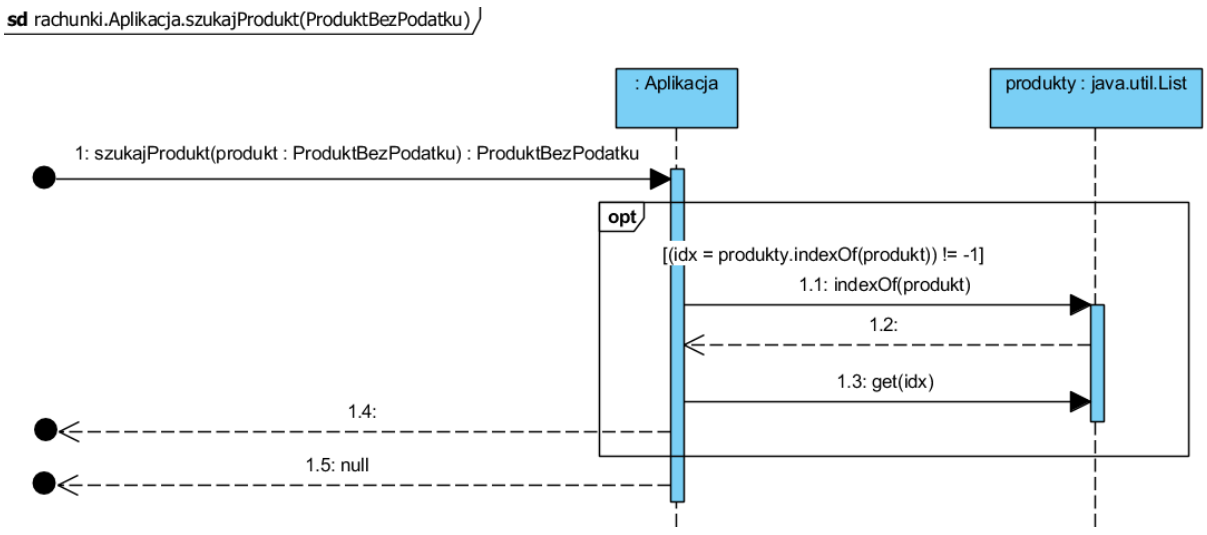


2.2. Kod operacji:

```
package rachunki;
public class Fabryka {
 public Fabryka() {   }
 public ProduktBezPodatku wykonajProdukt(String dane[]) {
      ProduktBezPodatku produkt = null;
      Promocja promocja;
      switch (Integer.parseInt(dane[0])) {
        case 0:
           produkt = new ProduktBezPodatku(dane[1], Float.parseFloat(dane[2]));
           break;
        case 1:
           promocja = new Promocja(Float.parseFloat(dane[3]));
           produkt = new ProduktBezPodatku(dane[1], Float.parseFloat(dane[2]), promocja);
           break;
        case 2:
           produkt = new ProduktZPodatkiem(dane[1], Float.parseFloat(dane[2]), Float.parseFloat(dane[3]));
           break;
        case 3:
           promocja = new Promocja(Float.parseFloat(dane[4]));
           produkt = new ProduktZPodatkiem(dane[1], Float.parseFloat(dane[2]), Float.parseFloat(dane[3]),
                          promocja);
           break;
      }
      return produkt;  }
}
```

3. Modelowanie i implementacja operacji **ProduktBezPodatku szukajProdukt (ProduktBezPodatku produkt)** w klasie **Aplikacja –** modelowanie i implementacja **PU Szukanie produktu**.

3.1. Diagram sekwencji operacji:



3.2. Kod operacji:

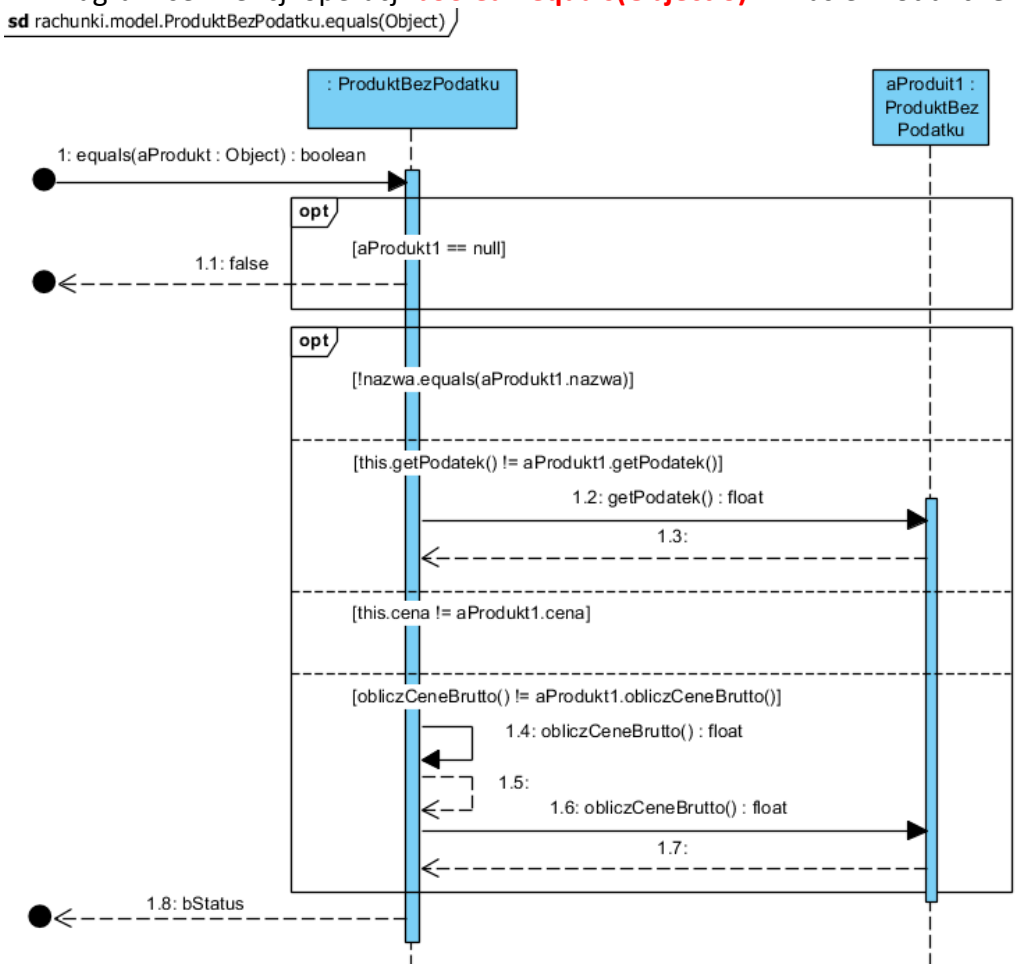
```
public  ProduktBezPodatku szukajProdukt (ProduktBezPodatku produkt)
  {   int idx;
     if ((idx=Produkty.indexOf(produkt))!=-1 )
      {   produkt=Produkty.get(idx);
        return produkt;  }
     return null;    }
```

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

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

```
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 **ProduktBezPodatku**:

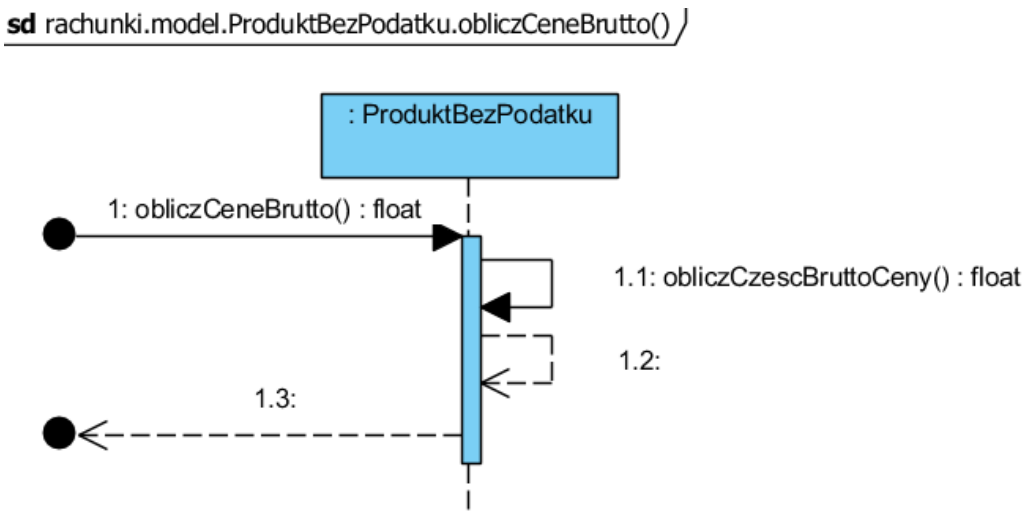


4.3. **Kod operacji equals w klasie ProduktBezPodatku**:

```
@Override
public boolean equals (Object aProdukt)
 {
    ProduktBezPodatku aProdukt1=(ProduktBezPodatku)aProdukt;
    if ( aProdukt1 == null ) return false;
    boolean bStatus = true;
   if ( !nazwa.equals(aProdukt1.nazwa)) bStatus = false;
   else
    if (this.getPodatek()!=aProdukt1.getPodatek())
       bStatus = false;
    else
      if (this.cena!=aProdukt1.cena)
        bStatus = false;
      else
        if (this.obliczCeneBrutto() != aProdukt1.obliczCeneBrutto())
          bStatus = false;
    return bStatus;
 }
```

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

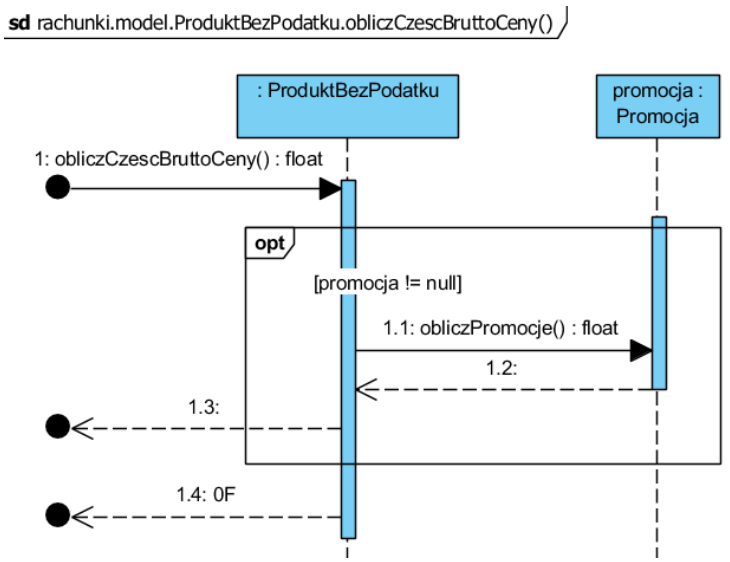
4.4.1. Diagram sekwencji operacji **float obliczCeneBrutto()** w klasie **ProduktBezPodatku**, która wywołuje metodę wirtualną **obliczCzescBruttoCeny(),** przedefiniowaną w klasie **ProduktZPodatkiem**: **obliczCeneBrutto()**



4.4.2. Kod metody **float obliczCeneBrutto()** w klasie **ProduktBezPodatku**:

```
public float obliczCeneBrutto()
  {      return cena + obliczCzescBruttoCeny();
  }
```

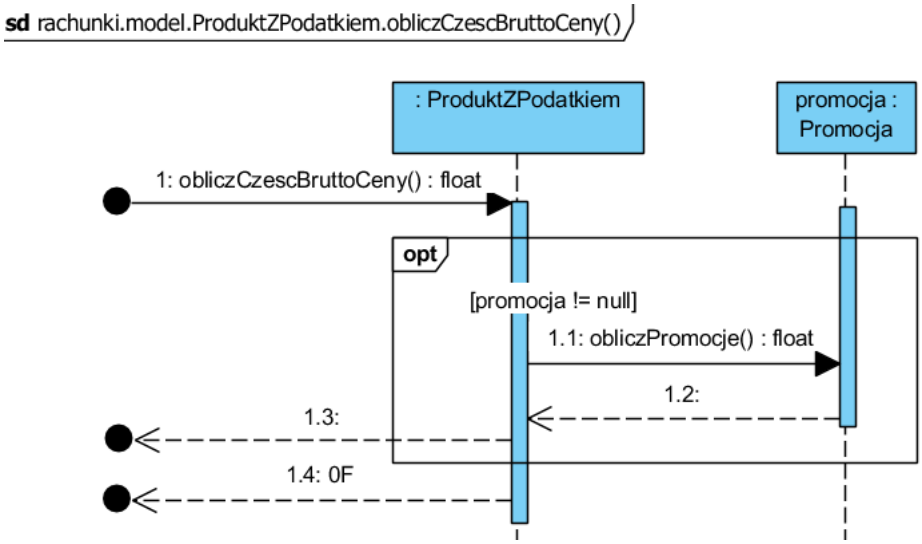
4.4.3. Diagram sekwencji metody **float obliczCzescBruttoCeny()** w klasie **ProduktBezPodatku**:



4.4.4. Kod metody **float obliczCzescBruttoCeny()** w klasie **ProduktBezPodatku**:

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

4.4.5. Diagram sekwencji metody **float obliczCzescBruttoCeny()** w klasie **ProduktZPodatkiem**:



4.4.6. Kod metody **float obliczCzescBruttoCeny()** w klasie **ProduktZPodatkiem**:

```
public  float obliczCzescBruttoCeny ()
 {
   float dodatek = 0;
   if (promocja  != null)
     dodatek= cena*(-promocja.obliczPromocje()/100);
   return  cena*podatek/100 + dodatek;
 }
```

4.4.7. Kod wirtualnych metod **float Podaj_podatek** w klasach **ProduktBezPodatku** i **ProduktZPodatkiem**, używanych w metodzie **equals** klasy **ProduktBezPodatku**, dziedziczonej przez klasę **ProduktZPodatkiem**:

//class ProduktBezPodatku

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

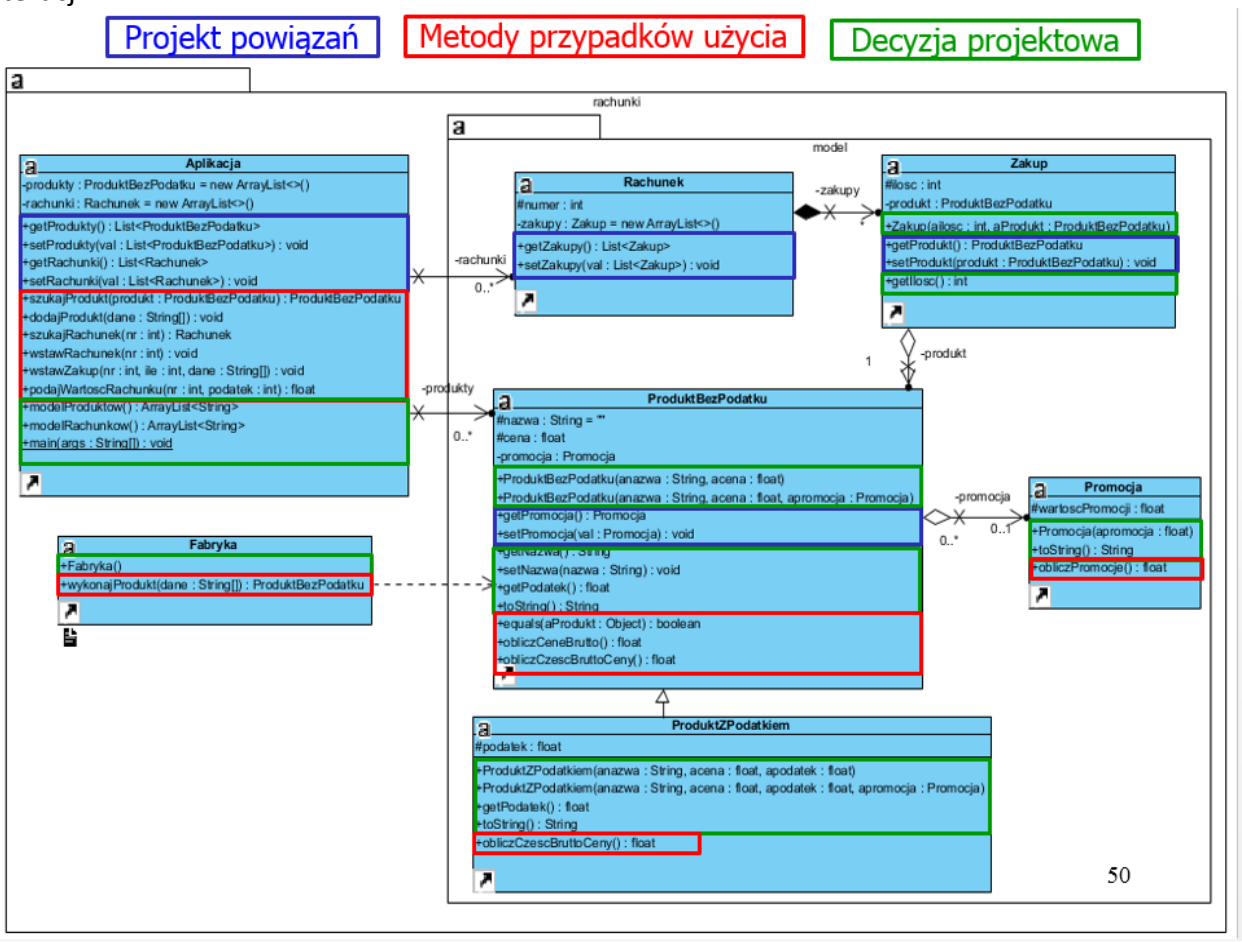
//class ProduktZPodatkiem

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

4.4.8. Kod metody **float obliczPromocje()** w klasie **Promocja** lub klasy pochodnej wykonującej **jakiś algorytm obliczania promocji** (koncepcja **wzorca czynnościowego Strategia**) np.

```
public  float obliczPromocje ()
  { if (wartoscPromocji<50)
        return wartoscPromocji;
    return wartoscPromocji *1.1F;
  }
```

5. Diagram klas zawierający elementy wynikające z wykonanych diagramów sekwencji w 1-ej 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 ProduktBezPodatku

```
public ProduktBezPodatku(String anazwa, float acena) {
      nazwa = anazwa;
      cena = acena;
   }
 public ProduktBezPodatku(String anazwa, float acena, Promocja 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(obliczCeneBrutto());
      if (Promocja != null) {
         sb.append(promocja.toString()); }
      return sb.toString();
   }

 public String getNazwa() {
      return nazwa;    }
```

Klasa ProduktZPodatkiem

```
 public ProduktZPodatkiem(String anazwa, float acena, float apodatek) {
    super(anazwa, acena);
    podatek = apodatek;
 }

public ProduktZPodatkiem(String anazwa, float acena, float apodatek, Promocja 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 Promocja

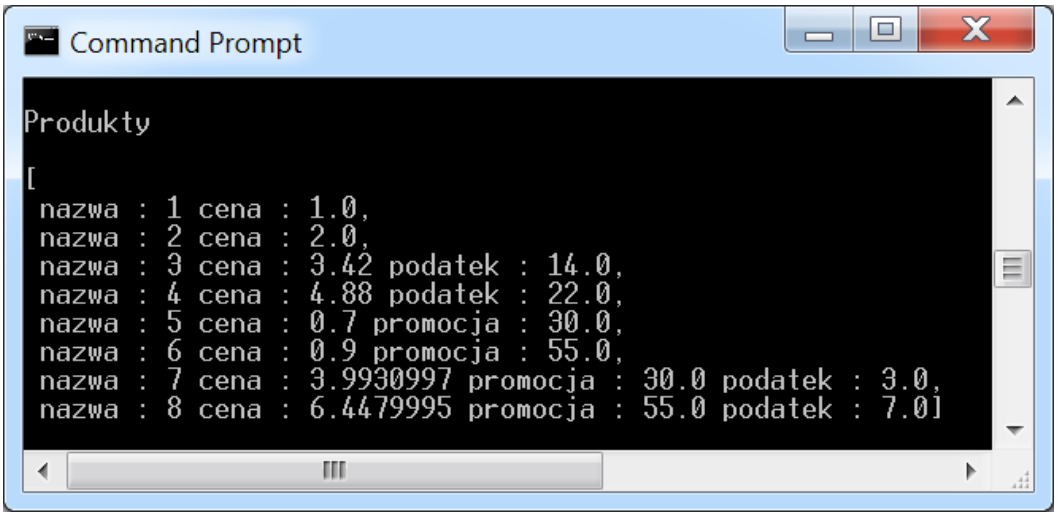
```
 public Pormocja(float apromocja) {
      promocja = apromocja;
   }

 @Override
 public String toString() {
      StringBuilder sb = new StringBuilder();
      sb.append(" promocja : ");
      sb.append(obliczPromocje());
      return sb.toString();
   }
```

Klasa Aplikacja

```
public ArrayList<String> modelProduktow() {
    ArrayList<String> modelProduktow = new ArrayList();
    for (ProduktBezPodatku produkt : produkty)
      modelProduktow.add("\n" + produkt.toString());
    return modelProduktow;
  }


public static void main(String args[]) {
    Aplikacja app = new Aplikacja();
    String dane1[] = {"0", "1", "1"};
    String dane2[] = {"0", "2", "2"};
    app.dodajProdukt(dane1);
    app.dodajProdukt(dane2);
    app.dodajProdukt(dane1);
    String dane3[] = {"2", "3", "3", "14"};
    String dane4[] = {"2", "4", "4", "22"};
    app.dodajProdukt(dane3);
    app.dodajProdukt(dane4);
    app.dodajProdukt(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", "12.46", "7", "50"};
    app.dodajProdukt(dane5);
    app.dodajProdukt(dane6);
    app.dodajProdukt(dane5);
    app.dodajProdukt(dane7);
    app.dodajProdukt(dane8);
    app.dodajProdukt(dane7);
    System.out.println("\nProdukty\n");
    System.out.println(app.modelProduktow());
  }
}
```



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)
2. Pomoc:**Drawing sequence diagrams.**
   (http://www.visual-paradigm.com/support/documents/vpumluserguide/94/2577/7025_drawingseque.html)