Instrukcja 7

Laboratoria 9, 10

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 przypadek użycia za pomocą diagramów sekwencji po wykonaniu pierwszego przypadku użycia podczas laboratorium 8, stanowiącego bazową logikę biznesową, z której korzystają kolejne przypadki użycia. 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 – rozszerzanie kodu źródłowego programu wykonanego podczas laboratoriów 7 i 8.**

1. Zdefiniować kolejne diagramy sekwencji operacji reprezentujących scenariusze poszczególnych przypadków użycia umieszczając je w projekcie UML założonym podczas realizacji instrukcji 2 i uzupełnianym podczas realizacji instrukcji 3-6.
2. Należy automatycznie uzupełniać definicję klas podczas modelowania kolejnych operacji za pomocą diagramów sekwencji. Należy rozwijać diagram klas utworzony podczas realizacji instrukcji 5 i 6.
3. Podzielić ten proces modelowania na kilka iteracji. Należy wykonać kolejne przypadki użycia, których wyniki wspierają działanie kolejnego modelowanego przypadku użycia w kolejnej iteracji **(wykład4, Dodatek 1 instrukcji)**. Pierwszy wykryty przypadek użycia należy modelować w 1-ej iteracji procesu projektowania (podczas realizacji instrukcji 6). 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 i 6.
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**.

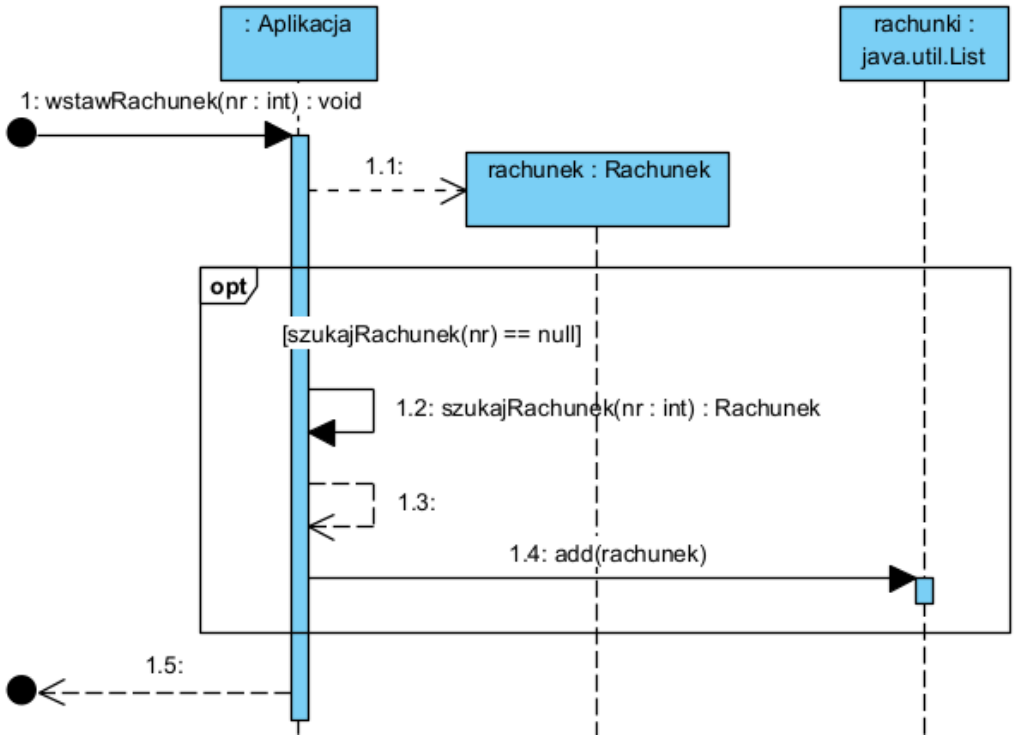
Dodatek 1

Przykład modelowania i implementacji przypadków użycia za pomocą diagramów sekwencji oraz diagramów klas i pakietów po wykonaniu bazowego przypadku użycia. Zastosowanie projektowych wzorców strukturalnych, wytwórczych i czynnościowych (cd. z instrukcji 2 - 5).

2-a iteracja: modelowanie przypadku użycia PU Dodawanie rachunku

1. Modelowanie i implementacja operacji **void wstawRachunek(int nr)** w klasie **Aplikacja.**

1.1. Diagram sekwencji operacji:

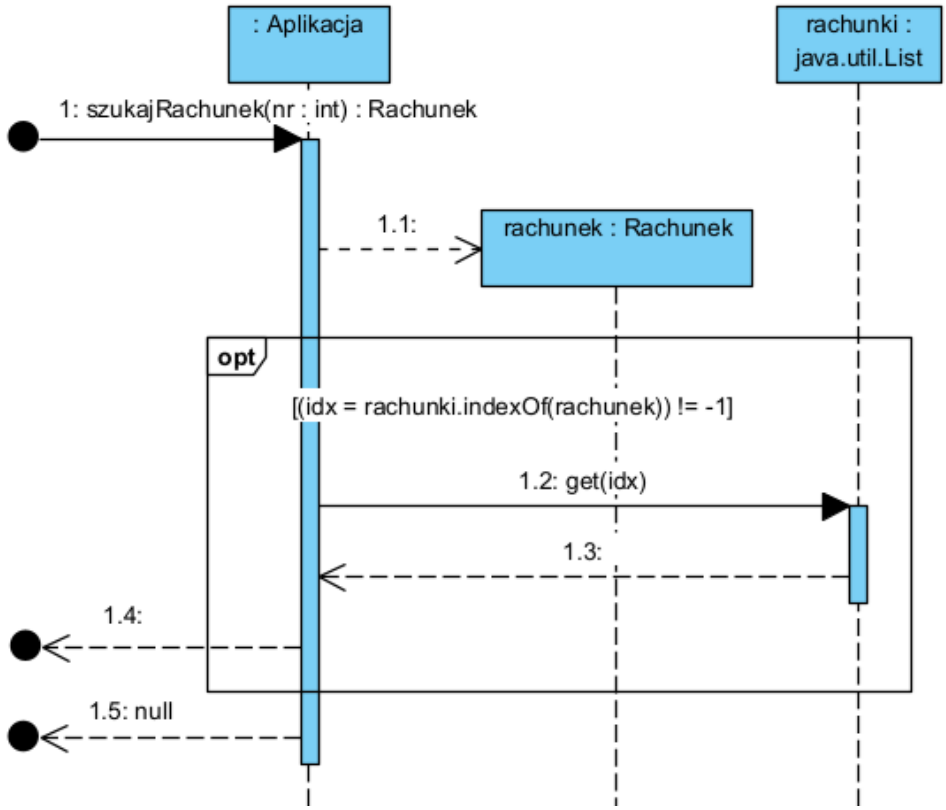


1.2. Kod operacji:

```
public void wstawRachunek(int nr) {
    Rachunek rachunek = new Rachunek(nr);
    if (szukajRachunek(nr) == null)
      rachunki.add(rachunek);
  }
```

2. Modelowanie i implementacja operacji **Rachunek szukajRachunek (int nr)** z klasy **Aplikacja** (modelownie **PU Szukanie Rachunku**).

2.1. Diagram sekwencji operacji:



2.2. Kod operacji:

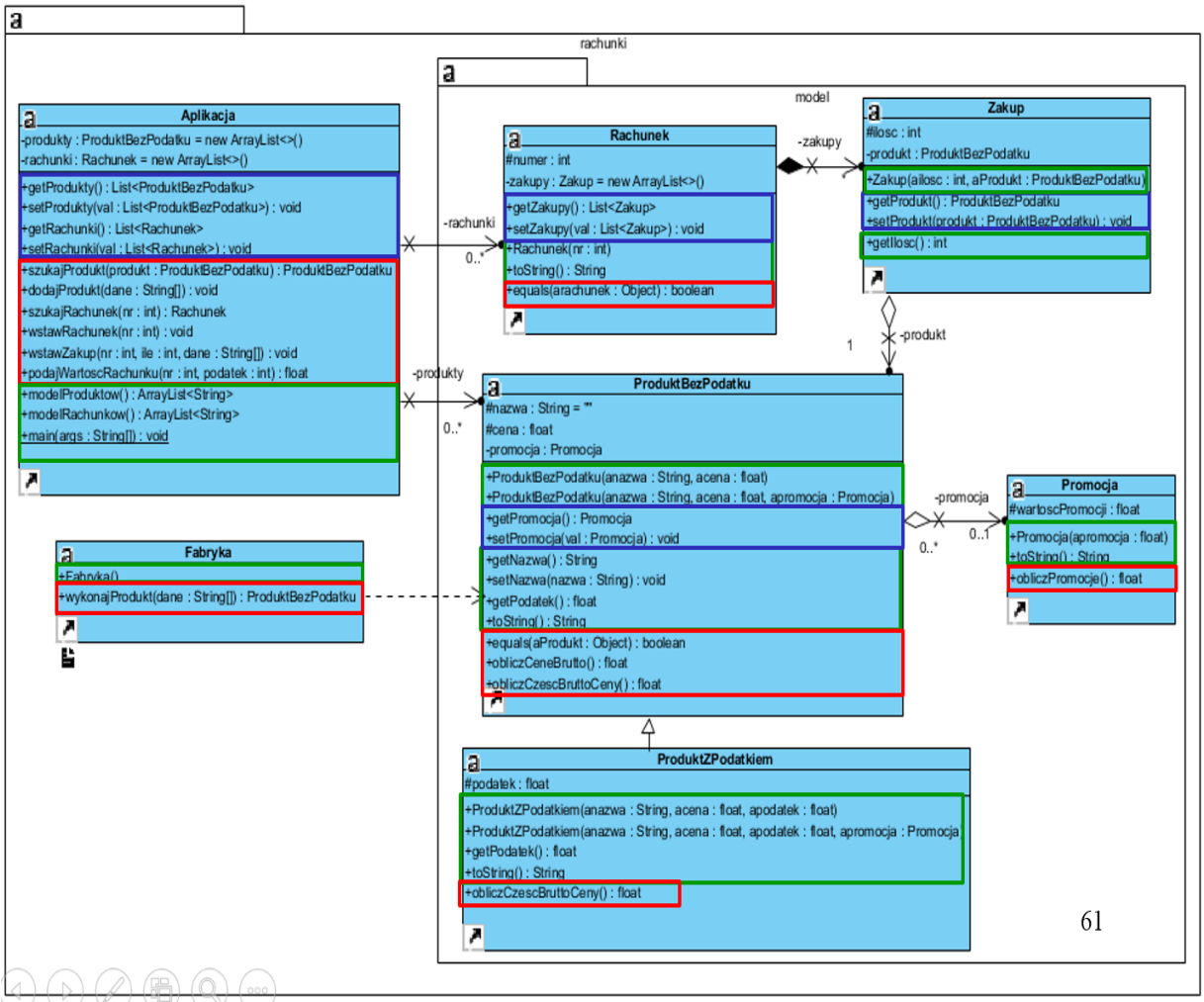
```
public  Rachunek szukajRachunek (int nr)  {
     Rachunek rachunek = new Rachunek(nr);
     int idx;
      if ((idx=rachunki.indexOf(rachunek)) != -1) {
          rachunek=rachunki.get(idx);
         return rachunek;
       }
      return null;
   }
```

2.3. Kod operacji boolean **equals(Object aRachunek)** w klasie **Rachunek**, wywoływanej w metodzie **indexOf** obiektu **Rachunki** typu **ArrayList**:

```
@Override
public boolean equals (Object aRachunek) {
    Rachunek rachunek= (Rachunek)aRachunek;
    return numer== rachunek.numer ;
}
```

3. Diagram klas zawierający elementy wynikające z wykonanych diagramów sekwencji w 2-iteracji

Projekt powiązań | Metody przypadków użycia | Decyzja projektowa



4. 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. Prezentacja wyników działania kodu z 1-ej iteracji oraz kodu z 2-iteracji, gdzie wyświetla się zawartość pustych rachunków (obiektów typu **Rachunek**).

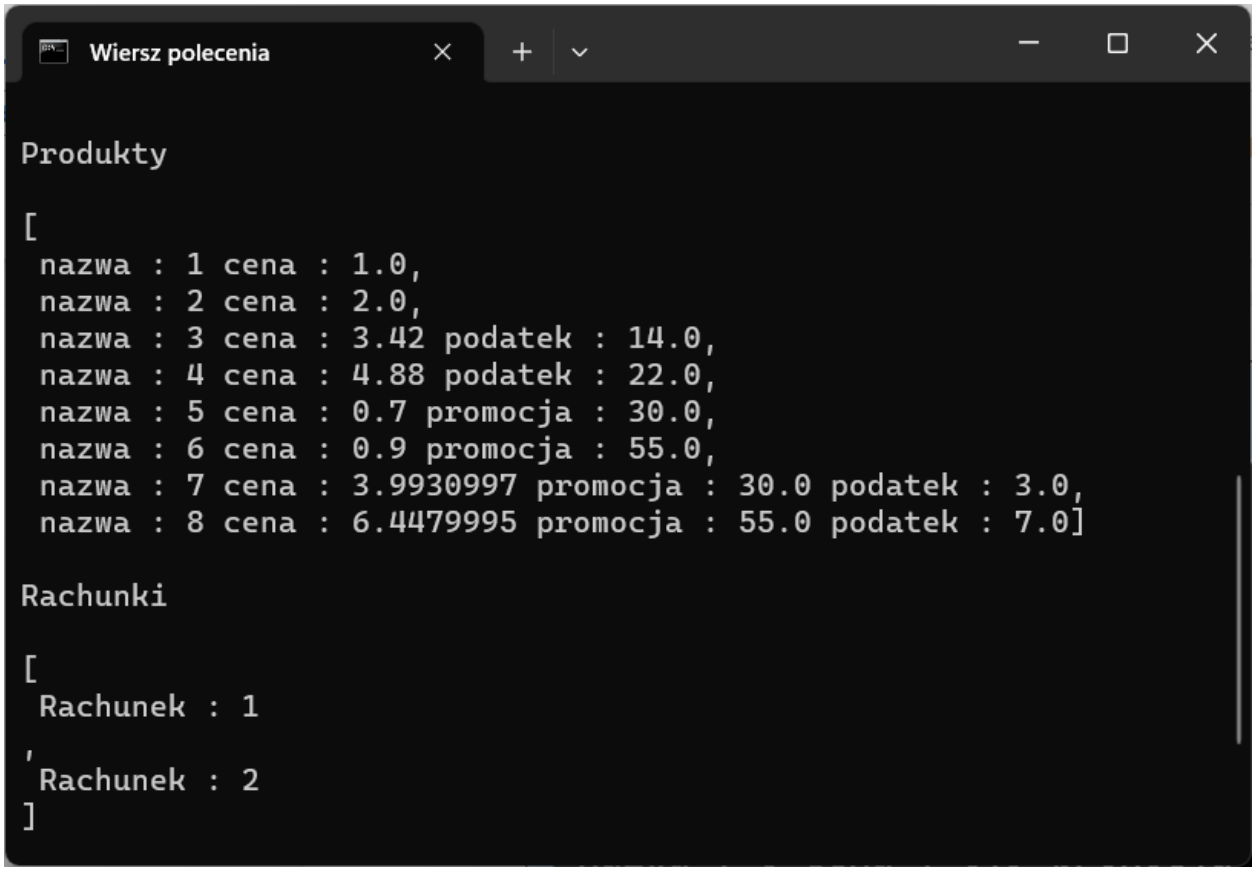
Klasa Rachunek

```
public Rachunek(int nr) {
      numer = nr;
}

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

Klasa Aplikacja (oparta na wzorcu Fasada)

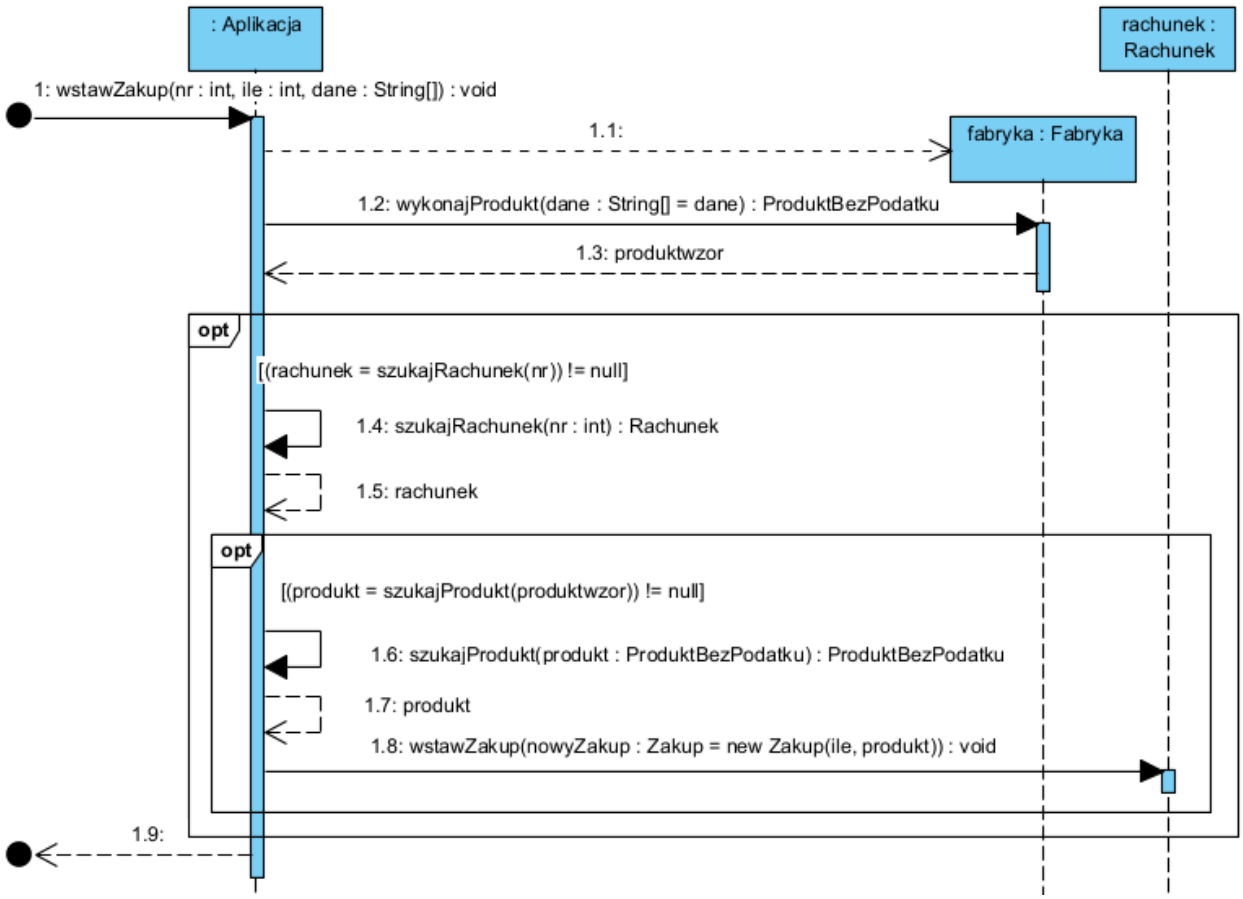
```
public ArrayList<String> modelRachunkow() {
      ArrayList<String> modelRachunkow = new ArrayList();
      for (Rachunek rachunek : rachunki)
         modelRachunkow.add("\n" + rachunek.toString());
      return modelRachunkow;
}
    //metodę main wykorzystano jedynie do testowania ręcznego zaprojektowanego kodu programu
 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());
      app.wstawRachunek(1);
      app.wstawRachunek(1);
      app.wstawRachunek(2);
      System.out.println("\nRachunki\n");
      System.out.println(app.modelRachunkow());
}
```



3-a iteracja: modelowanie przypadku użycia PU Dodawanie zakupu

1. Modelowanie i implementacja operacji **void wstawZakup(int nr, int ile, String dane[])** w klasie **Aplikacja.**

   1.1. Diagram sekwencji operacji:

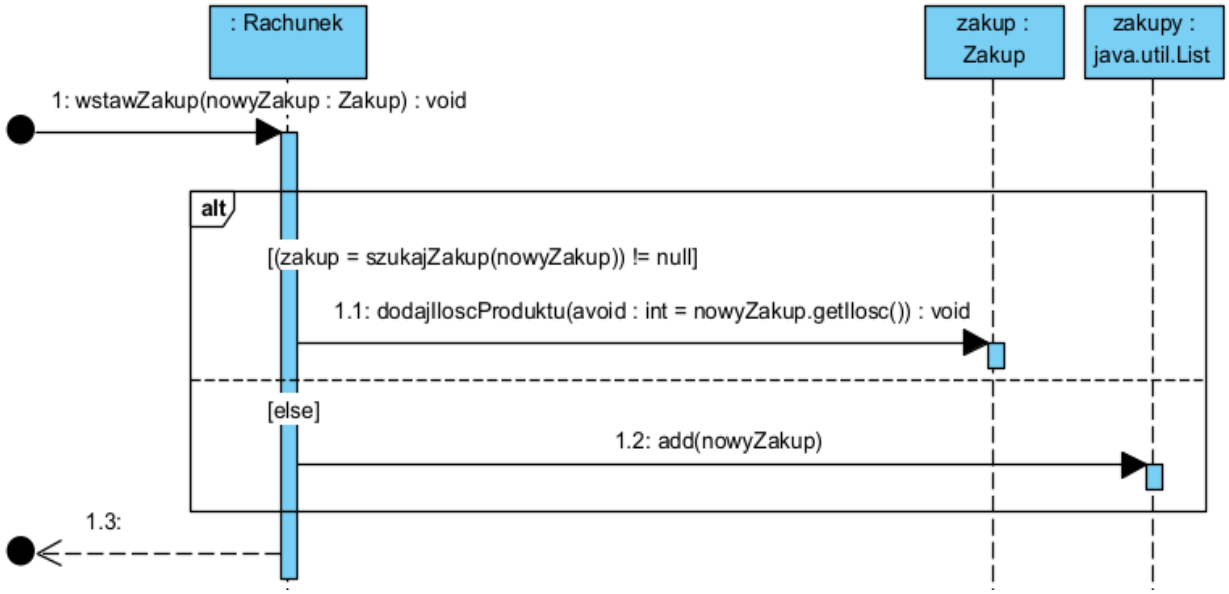


   1.2. Kod operacji:

```
public void wstawZakup(int nr, int ile, String dane[]) {
    Rachunek rachunek;
    Fabryka fabryka = new Fabryka();
    ProduktBezPodatku produkt1 = fabryka.wykonajProdukt(dane);      //1-a iteracja
    if ((rachunek = szukajRachunek(nr)) != null)                    //2-a iteracja
      if ((produkt1 = szukajProdukt(produkt1)) != null)             //1-a iteracja
        rachunek.wstawZakup(new Zakup(ile, produkt1);
}
```

2. Modelowanie i implementacja operacji **void wstawZakup(Zakup azakup)** z klasy **Rachunek.**

2.1. Diagram sekwencji operacji **void wstawZakup(Zakup azakup)** z klasy **Rachunek**:



2.2. Kod operacji **void wstawZakup(Zakup azakup)** z klasy **Rachunek**:

```
public void wstawZakup(Zakup azakup) {
   Zakup zakup;
   if ((zakup = szukajZakup(azakup)) != null)
     zakup.dodajIloscProduktu (azakup.getIlosc());
   else   zakupy.add(azakup);
  }
```

2.3. Kod operacji **public void dodajIloscProduktu(int avoid)** w klasie **Zakup**:

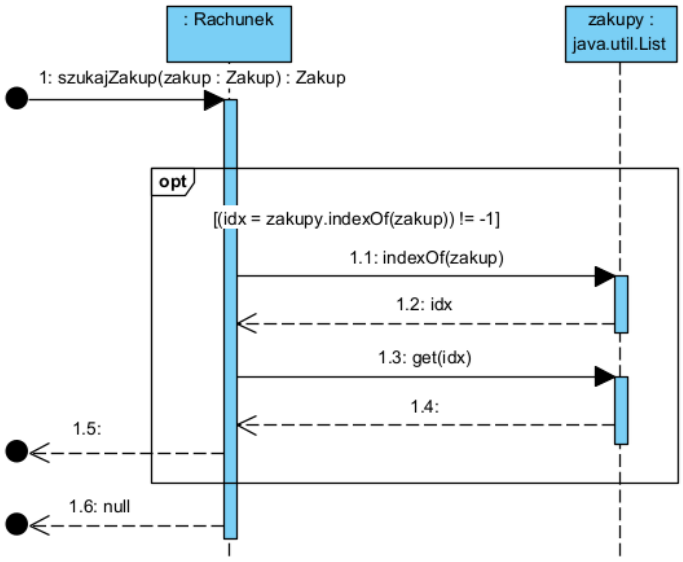
```
public void dodajIloscProduktu(int avoid) {
     ilosc += avoid;
  }
```

2.4. Kod operacji **int getIlosc()** z w klasie **Zakup**:

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

3. Modelowanie i implementacja operacji **Zakup szukajZakup(Zakup zakup)** z klasy **Rachunek**.

3.1. Diagram sekwencji operacji:

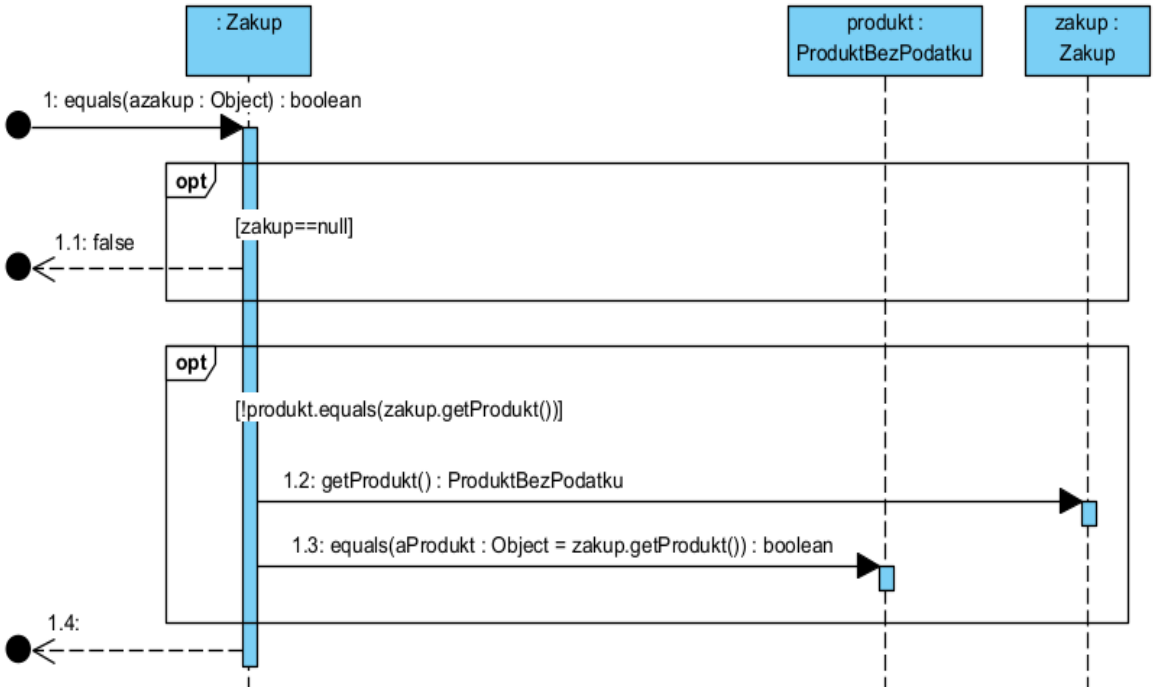


Kod operacji:

```
public Zakup szukajZakup(Zakup zakup) {
    int idx;
    if ((idx = zakupy.indexOf(zakup)) != -1) {
       zakup = zakupy.get(idx);
       return zakup;
    }
    return null;
  }
```

4. Modelowanie i implementacja operacji **boolean equals(Object azakup)** z klasy **Zakup**, wywoływanej w metodzie **indexOf** obiektu **zakupy** typu **ArrayList**.

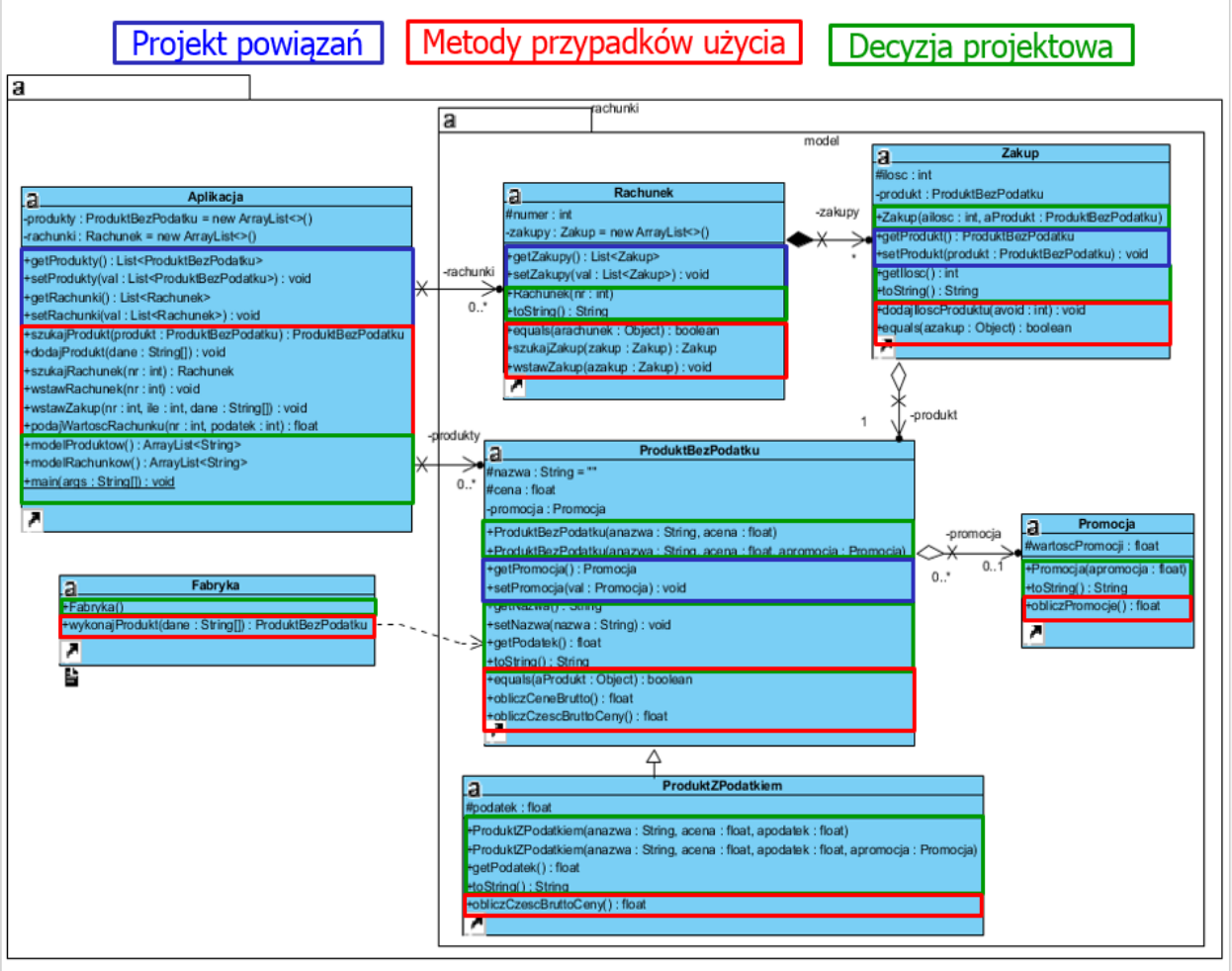
4.1. Diagram sekwencji operacji:



4.2. Kod operacji:

```
@Override
public boolean equals(Object azakup) {
    Zakup zakup = (Zakup) azakup;
    if (zakup == null)
        return false;
    return produkt.equals(zakup.getProdukt());          //1-a iteracja
}
```

5. Diagram klas zawierający elementy wynikające z wykonanych diagramów sekwencji w 3-iteracji.

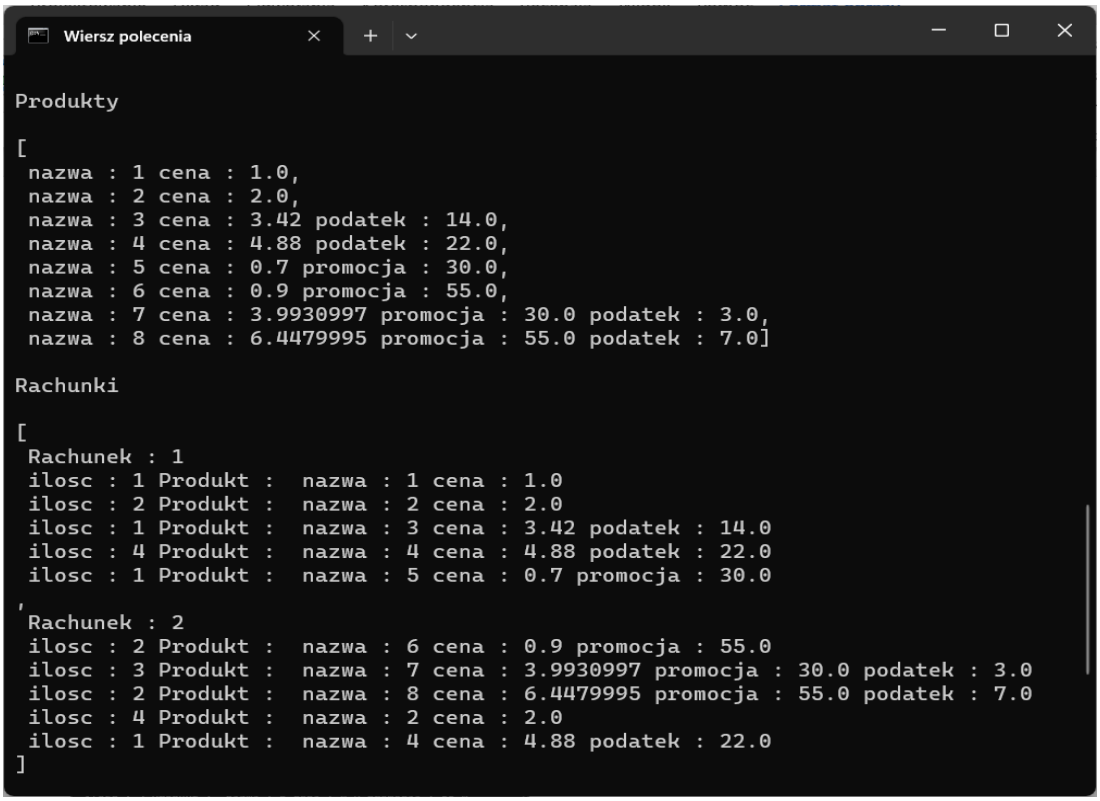


6. Wykonanie kodu źródłowego programu na podstawie wykonanego diagramu klas i diagramów sekwencji oraz pomocniczych metod do prezentacji wyników metod logiki biznesowej modelowanych za pomocą diagramów sekwencji – prezentacja wyników pierwszych trzech iteracji, gdzie dodatkowo prezentuje się zawartość obiektów typu **Rachunek**, zawierających kolekcję obiektów typu **Zakup**.

Klasa Zakup

```
public Zakup(int ailosc, ProduktBezPodatku aProdukt) {
    ilosc = ailosc;
    produkt = aProdukt;
}
```

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



Klasa Aplikacja (oparta na wzorcu Fasada)

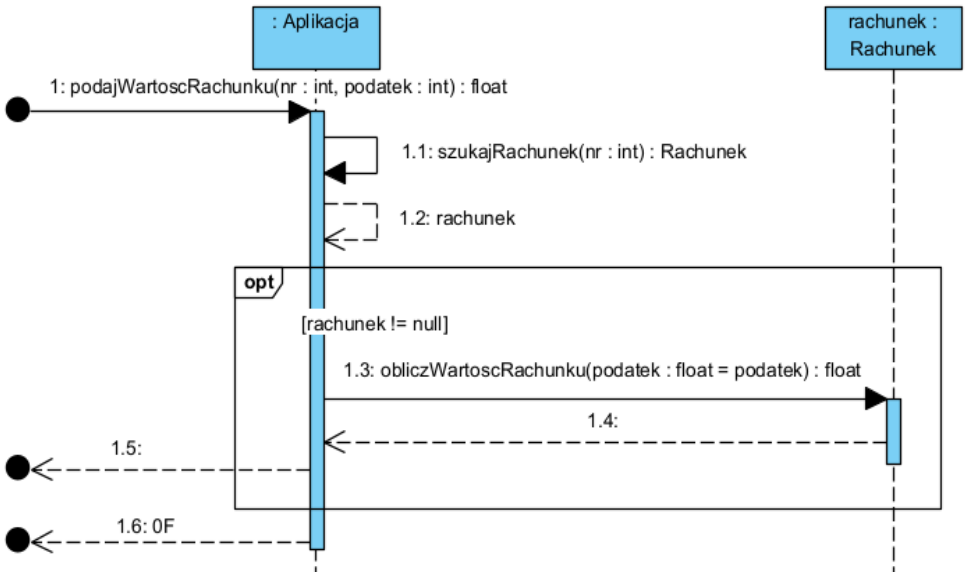
//metodę main wykorzystano jedynie do testowania ręcznego zaprojektowanego kodu programu

```
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.Wyswietl();
    app.dodajProdukt(dane4);  //app.Wyswietl();
    app.dodajProdukt(dane3);   //app.Wyswietl();
    String dane5[] = {"1", "5", "1", "30"};
    String dane6[] = {"1", "6", "2", "50"};
    String dane7[] = {"3", "7", "5.47", "3", "30"};
    String dane8[] = {"3", "8", "12.4", "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());
    app.wstawRachunek(1);
    app.wstawRachunek(1);
    app.wstawRachunek(2);
    app.wstawZakup(1, 1, dane1);
    app.wstawZakup(1, 2, dane2);
    app.wstawZakup(1, 1, dane3);
    app.wstawZakup(1, 4, dane4);
    app.wstawZakup(1, 1, dane5);
    app.wstawZakup(2, 1, dane6);
    app.wstawZakup(2, 3, dane7);
    app.wstawZakup(2, 1, dane8);
    app.wstawZakup(2, 4, dane2);
    app.wstawZakup(2, 1, dane4);
    app.wstawZakup(2, 1, dane6);
    app.wstawZakup(2, 1, dane8);
    System.out.println("\nRachunki\n");
    System.out.println(app.modelRachunkow());
}
```

4-a iteracja: modelowanie przypadku użycia PU Obliczanie wartosci rachunku

1. Modelowanie i implementacja operacji **float podajWartoscRachunku(int nr, int podatek)** z klasy **Aplikacja**

   1.1. Diagram sekwencji operacji:

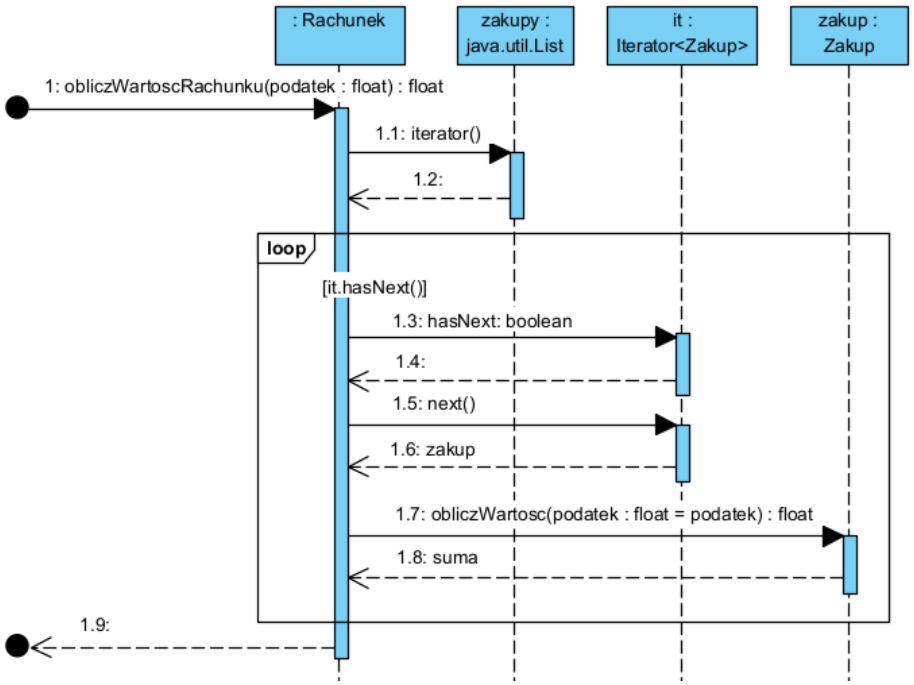


   1.2. Kod operacji:

```
public float podajWartoscRachunku(int nr, int podatek) {
     Rachunek rachunek;
     rachunek = szukajRachunek(nr);                    // 2-a iteracja
     if (rachunek != null)
        return rachunek.obliczWartoscRachunku(podatek);
     return 0F;
  }
```

2. Modelowanie i implementacja operacji **float obliczWartoscRachunku (int podatek)** z klasy **Rachunek**

   2.1. Diagram sekwencji operacji:

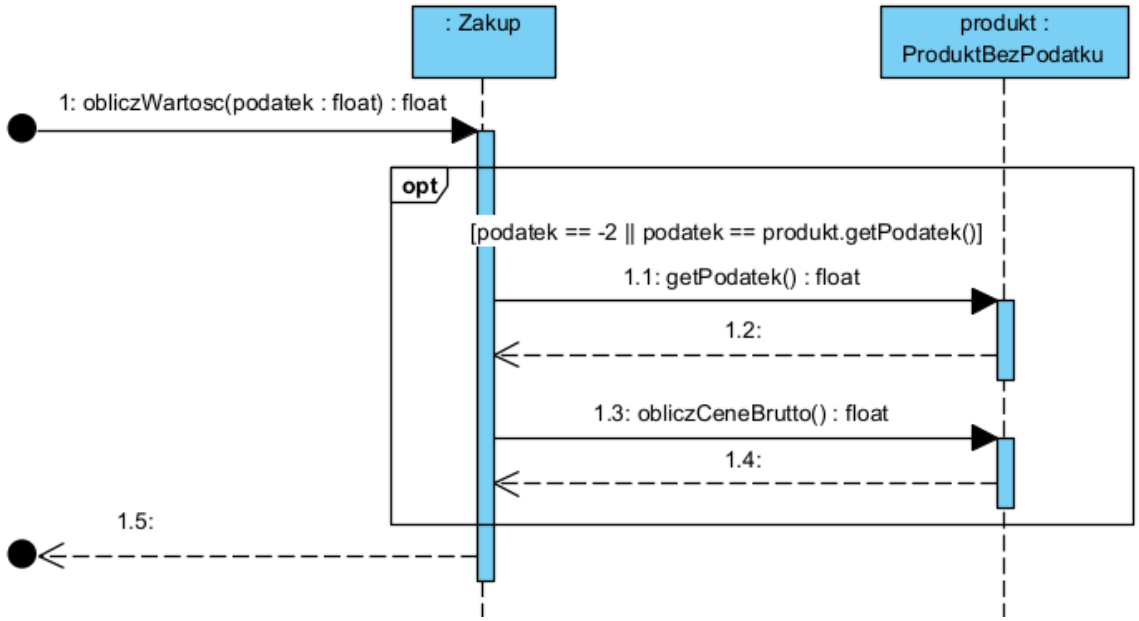


2.2. Kod operacji:

```
public float obliczWartoscRachunku(int podatek) {
    float suma = 0;
    Zakup zakup;
    Iterator<Zakup> it = zakupy.iterator();
    while (it.hasNext()) {
       zakup = it.next();
       suma += zakup.obliczWartosc(podatek);
    }
    return suma;
}
```

3. Modelowanie i implementacja operacji **float obliczWartosc(int podatek)** z klasy **Zakup**.

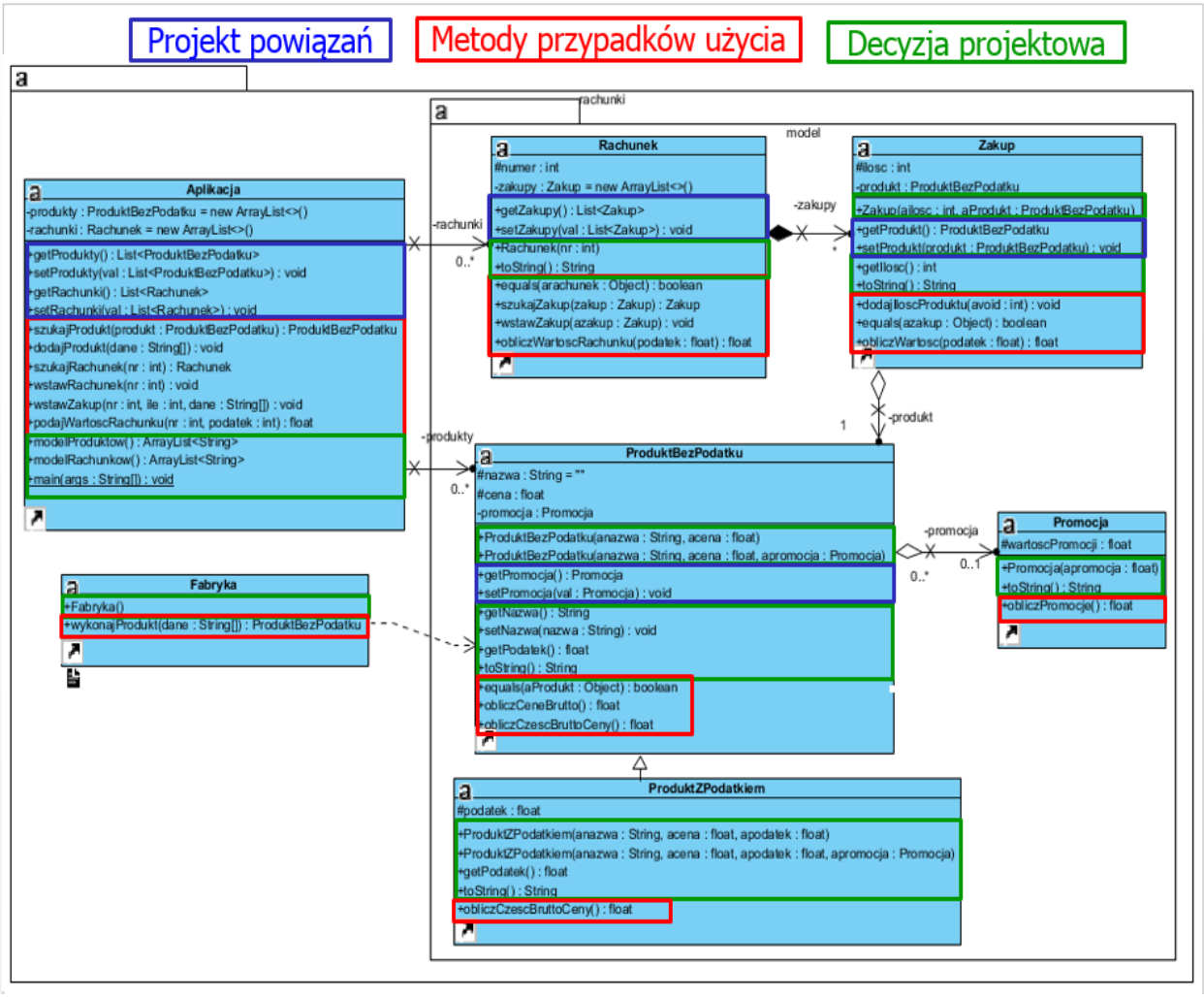
3.1. Diagram sekwencji operacji:



3.2. Kod operacji:

```
public float obliczWartosc(int podatek) {
    if (podatek == -2 || podatek == produkt.getPodatek()) {
       return ilosc * produkt.obliczCeneBrutto();          //1-a iteracja
    }
    return 0F;
  }
```

4. Diagram klas zawierający elementy wynikające z wykonanych diagramów sekwencji w 4-iteracji



5. Wykonanie kodu źródłowego programu na podstawie wykonanego diagramu klas i diagramów sekwencji oraz pomocniczych metod do prezentacji wyników metod logiki biznesowej modelowanych za pomocą diagramów sekwencji – prezentacja wyników czterech iteracji, gdzie prezentuje się zawartość obiektów typu **Rachunek** zawierających kolekcję obiektów typu **Zakup** oraz wartość tych rachunków.

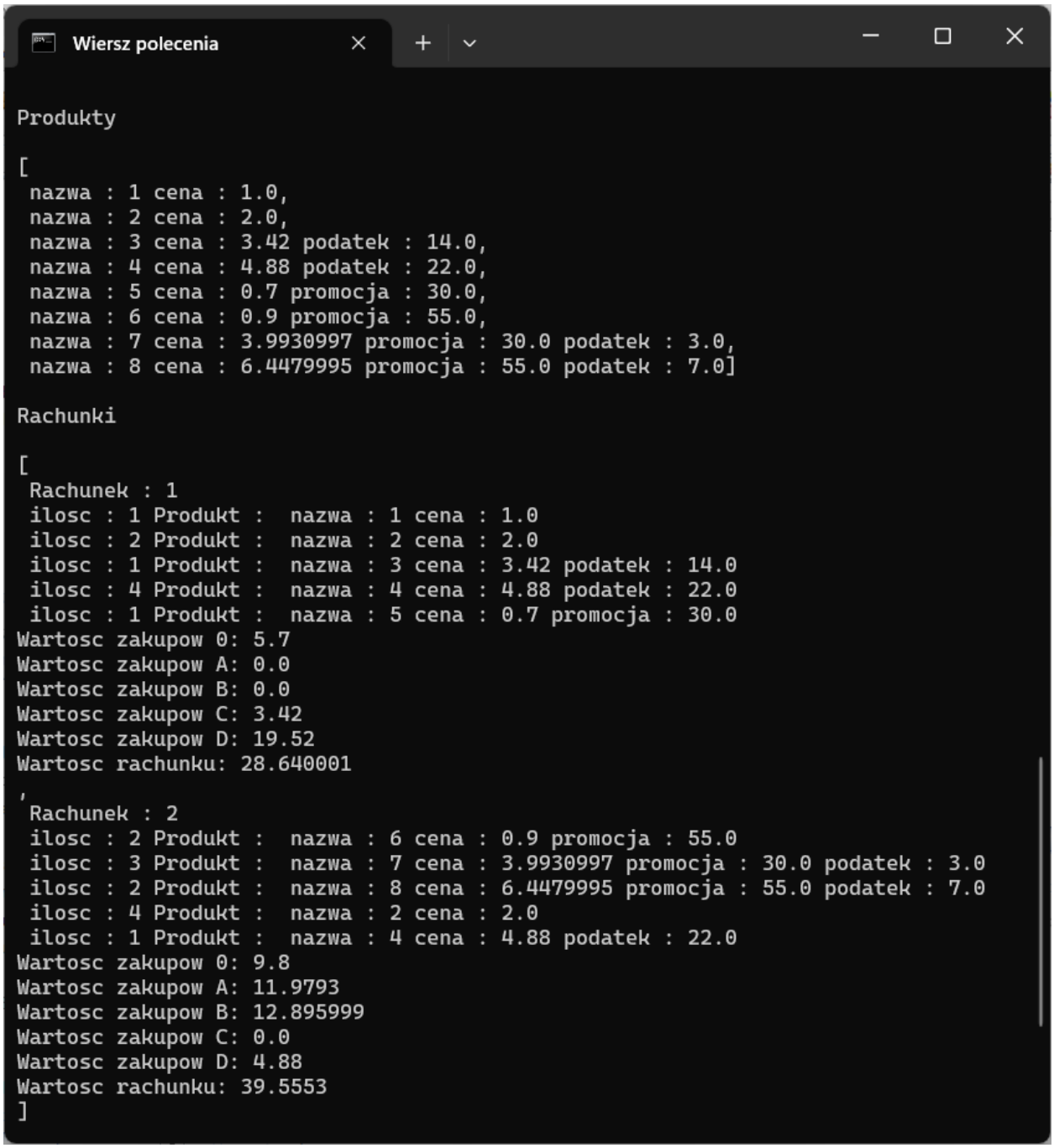
Rachunek

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

Klasa Aplikacja (oparta na wzorcu Fasada)

//metodę main wykorzystano jedynie do testowania ręcznego zaprojektowanego kodu programu

```
public static void main(String args[])
 {                                          //kod metody main po
    Aplikacja app=new Aplikacja();          //implementacji
   String dane1[]={"0","1","1"};            // 6-u przypadków użycia
   String dane2[]={"0","2","2"};             // identyczny jak po implementacji
   app.dodajProdukt(dane1);                 // 5-go przypadku użycia
   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","5"};
   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());
    app.wstawZakup(1, 1, dane1);
    app.wstawZakup(1, 2, dane2);
    app.wstawZakup(1, 1, dane3);
    app.wstawZakup(1, 4, dane4);
    app.wstawZakup(1, 1, dane5);
    app.wstawZakup(2, 1, dane6);
    app.wstawZakup(2, 3, dane7);
    app.wstawZakup(2, 1, dane8);
    app.wstawZakup(2, 4, dane2);
    app.wstawZakup(2, 1, dane4);
    app.wstawZakup(2, 1, dane6);
    app.wstawZakup(2, 1, dane8);
   System.out.println("\nRachunki\n");
   System.out.println(app.modelRachunkow());
  }
}
```



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)