Diagramy maszyn stanowych
Wykład 5 – część 1

Zofia Kruczkiewicz

Składnia elementów na diagramach UML

1. W prezentacji składni diagramów stanów UML (str.7-20, 22) o **charakterze tutorialowym** sposób definiowania składowych klas (stany, zdarzenia, akcje) odwzorowanych na elementy klas, jest jednym z przyjętych sposobów interpretowania specyfikacji języka UML w tutorialach – często odbiegająca od syntaktyki znanych języków obiektowych (Java, C++) i zazwyczaj uproszczona.
2. W prezentacji składni diagramu stanów UML (str.23, 24) oraz diagramów sekwencji (str.26, 28, 32, 35, 37) wykazując spójność modelowanego zachowania obiektów za pomocą tych dwóch typów diagramów prezentując sposób odwzorowania elementów diagramów, czyli zdarzenia, stany, akcje na elementy klas, itd jest jednym z przyjętych sposobów interpretowania specyfikacji języka UML w narzędziach UML (***VP CE***). **Diagramy sekwencji uzyskano generując diagramy z kodu Javy.**

Wniosek: W wielu narzędziach UML sposób definiowania elementów diagramów oparty na tej samej specyfikacji UML różni się. W prezentowanych materiałach przedstawiono te różnice, stosując dwa różne sposoby definiowania oparte na:

1) tutorialach (p.1): **https://sparxsystems.com/resources/tutorials/uml2/index.html**
2) narzędziu z serii Visual Paradigm CE - **VP CE** (p. 2: **http://zofia.kruczkiewicz.staff.iiar.pwr.wroc.pl/wyklady/IO_UML/Instrukcja_1_2.pdf**,

W mat. 2) diagramy sekwencji zostały wygenerowane z kodu Javy (inzynieria odwrotna),

w celu zwrócenia uwagi, że te różnice są naturalnym zjawiskiem, ale zawsze wspierającym programistów.

Diagramy stanów UML 2 – część piąta

Na podstawie

UML 2.0 Tutorial

https://sparxsystems.com/resources/tutorials/uml2/state-diagram.html

Diagramy stanów

1. **Diagramy stanów UML**

   https://sparxsystems.com/resources/tutorials/uml2/state-diagram.html

2. **Przykład diagramów stanów UML – modelowanie wpływu przypadków użycia na stany obiektu**

Diagramy stanów

1. Diagramy stanów UML

https://sparxsystems.com/resources/tutorials/uml2/state-diagram.html

Dwa rodzaje diagramów UML 2

Diagramy UML modelowania strukturalnego

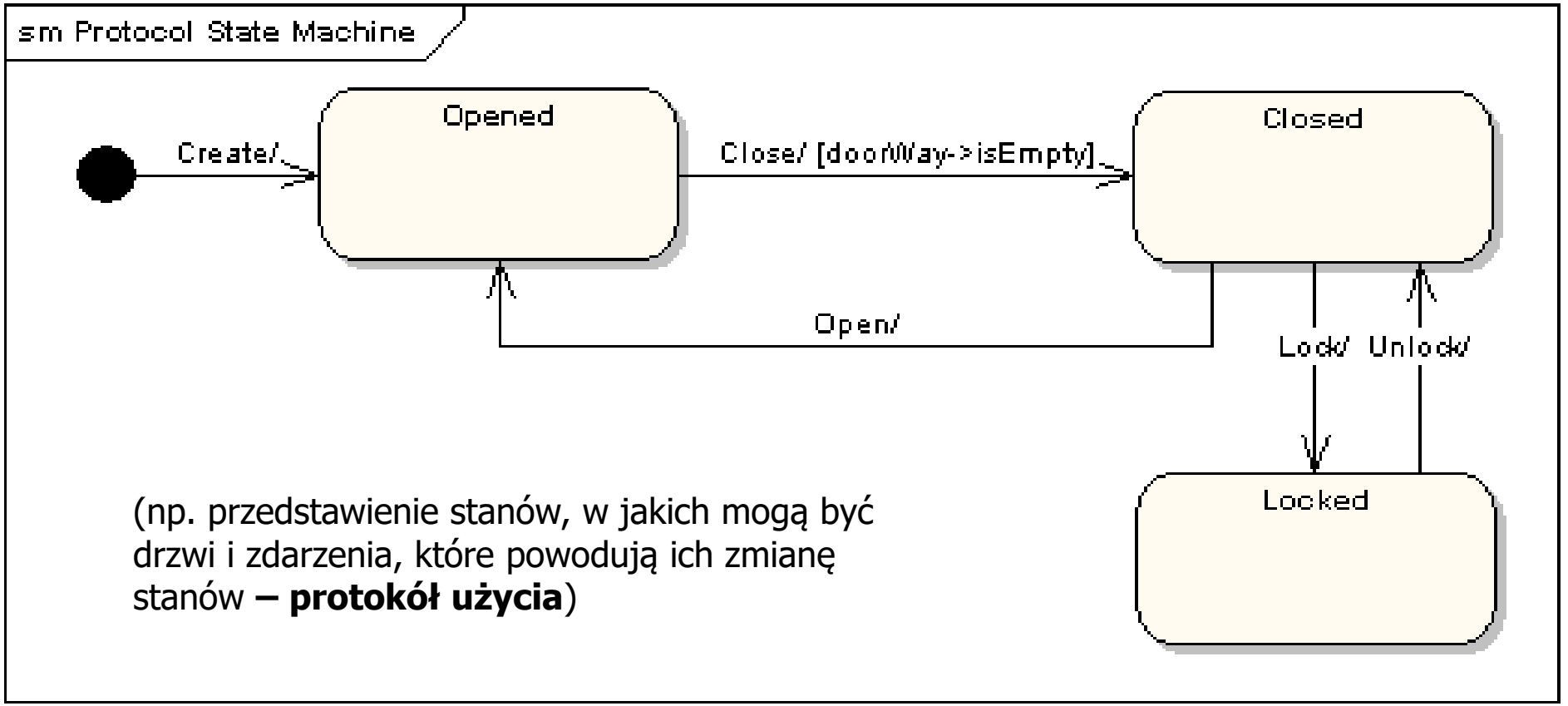
- Diagramy pakietów
- *Diagramy klas*
- Diagramy obiektów
- Diagramy mieszane
- Diagramy komponentów
- Diagramy wdrożenia

Diagramy UML modelowania zachowania

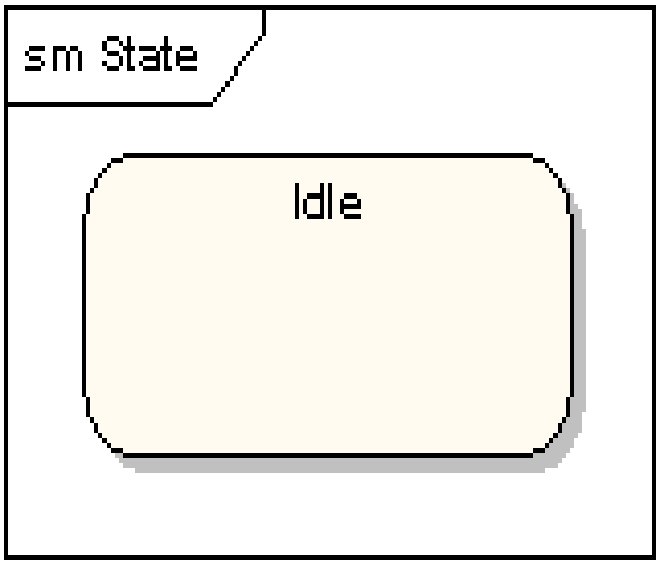
- *Diagramy przypadków użycia*
- *Diagramy aktywności*
- *Diagramy stanów*
- Diagramy komunikacji
- *Diagramy sekwencji*
- Diagramy czasu
- Diagramy interakcji

Diagram stanu opisuje zmiany stanu obiektu, podsystemu lub systemu pod wpływem działania operacji - iest szczególnie przydatny, gdy zachowanie obiektu jest złożone. Przedstawia on **maszynę stanów** jako przepływ sterowania między stanami.

Diagram stanów jest grafem złożonym z wierzchołków i krawędzi: wierzchołkami są **stany** (prostokąty o zaokrąglonych rogach), krawędziami są **przejścia** (strzałki).

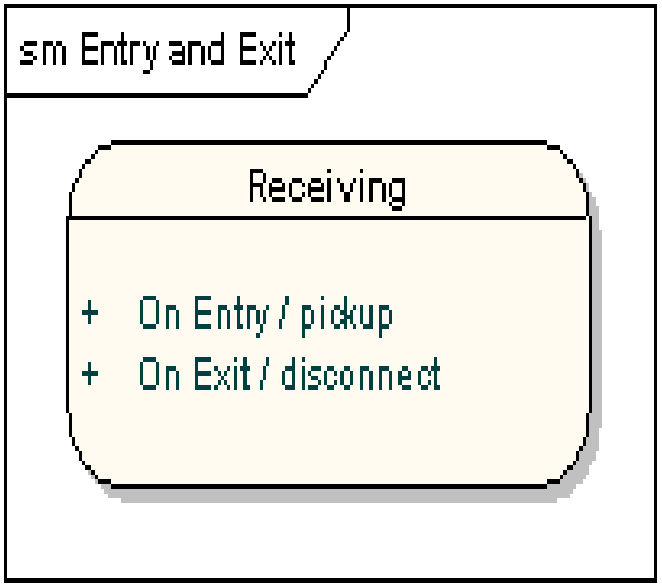


(np. przedstawienie stanów, w jakich mogą być drzwi i zdarzenia, które powodują ich zmianę stanów **– protokół użycia**)



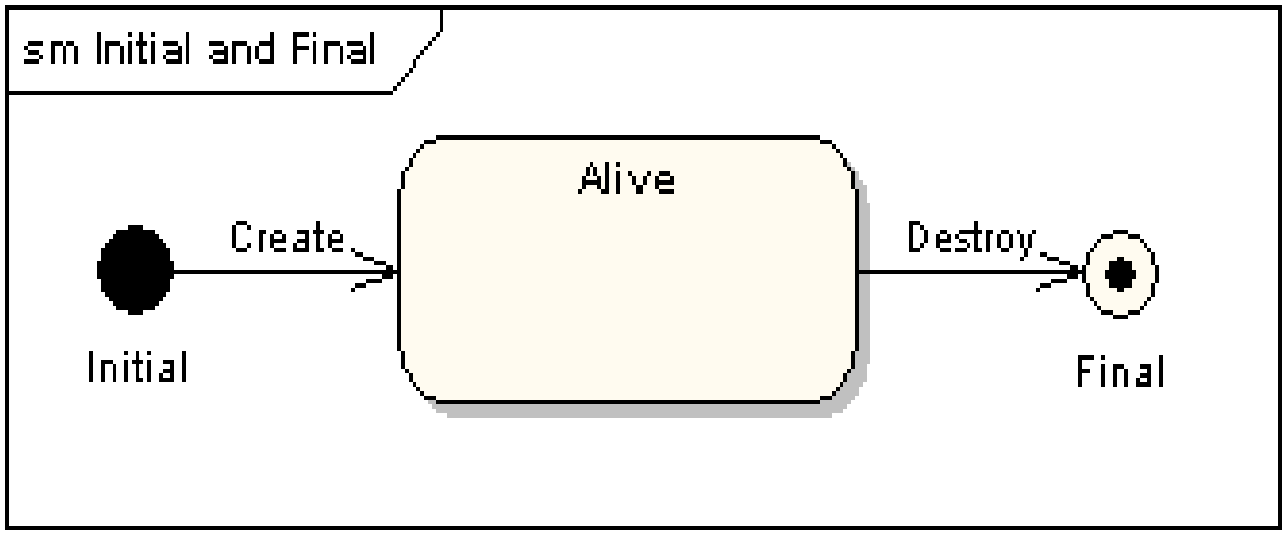
Stan jest okolicznością lub sytuacją, w jakiej znajduje się obiekt

- jest rezultatem poprzedniej aktywności
- spełnia jakiś warunek
- jest określony przez wartości własnych atrybutów i powiązań do innych zadań
- wykonuje pewne czynności
- czeka na jakieś zdarzenie



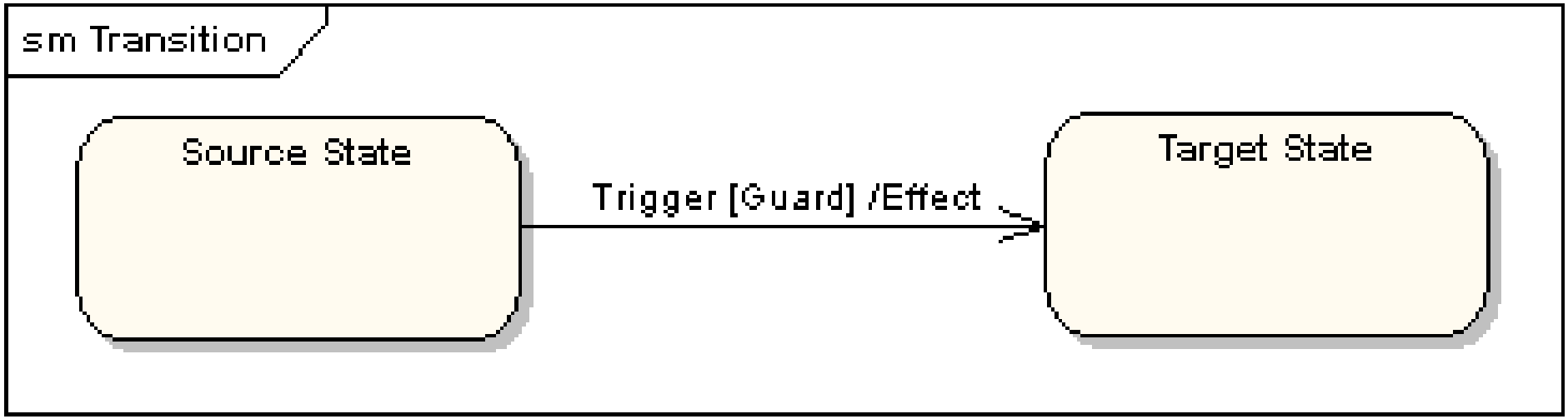
- **Nazwa** -unikatowy ciąg znaków, brak nazwy dla stanu anonimowego

- **Akcje wejściowe** (**entry**) i **wyjściowe** (**exit**) - akcje wykonywane odpowiednio przy wejściu do stanu i przy wyjściu)



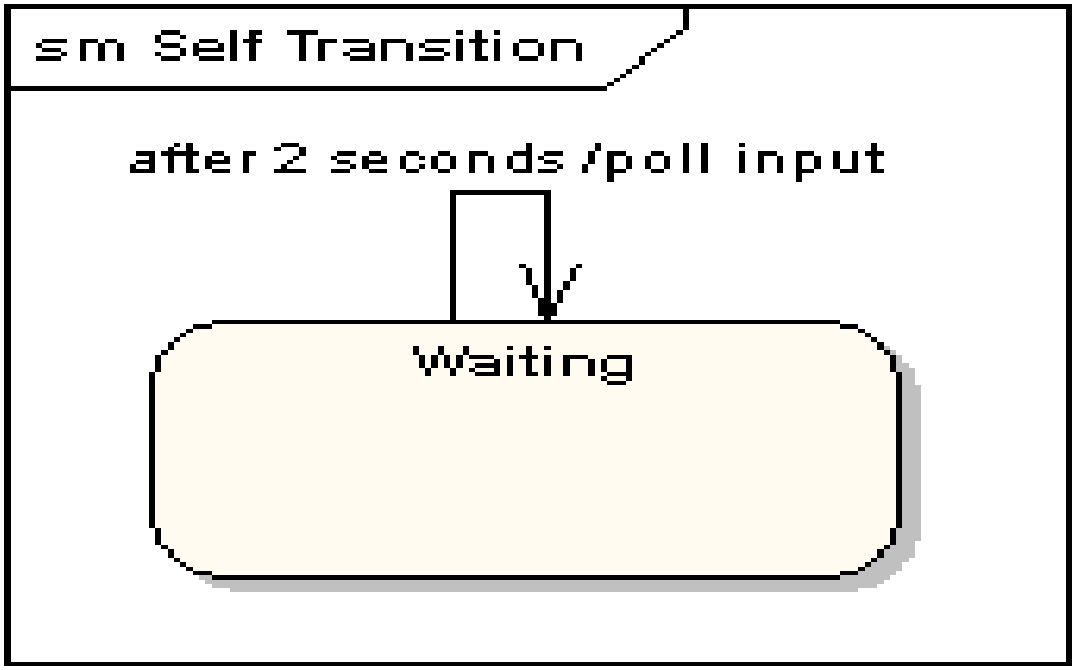
Stan początkowy (**Initial**) – może być tylko jeden stan początkowy

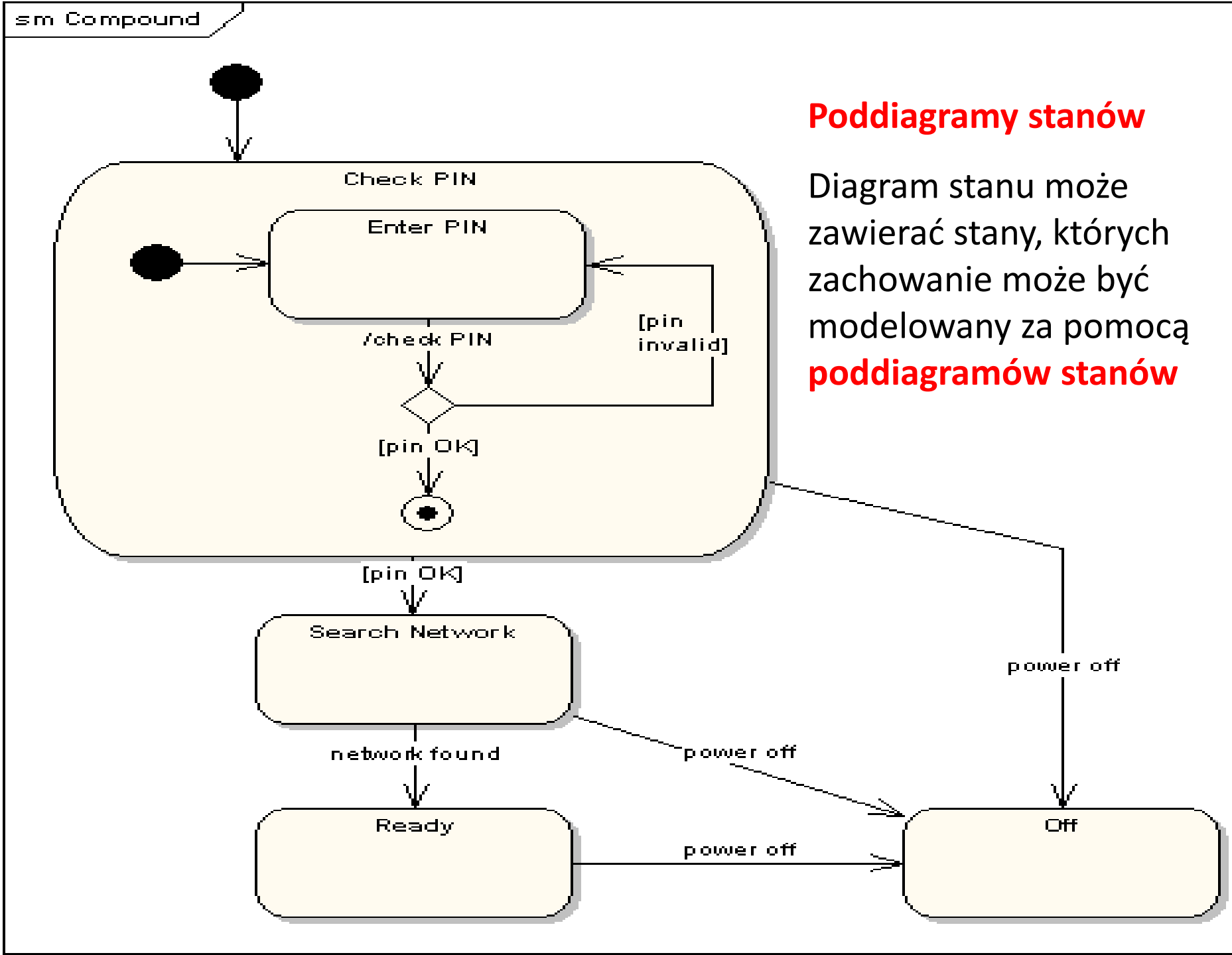
Stan końcowy (**Final**) – może być kilka stanów końcowych



Przejście (**Transition**) jest związkiem między dwoma stanami, który wskazuje, że np. obiekt znajdujący się w pierwszym stanie wykona pewne **akcje (Effect)** i przejdzie do drugiego stanu, ilekroć zaistnieje określone **zdarzenie (Trigger**) i będą spełnione określone **warunki (Guard).**

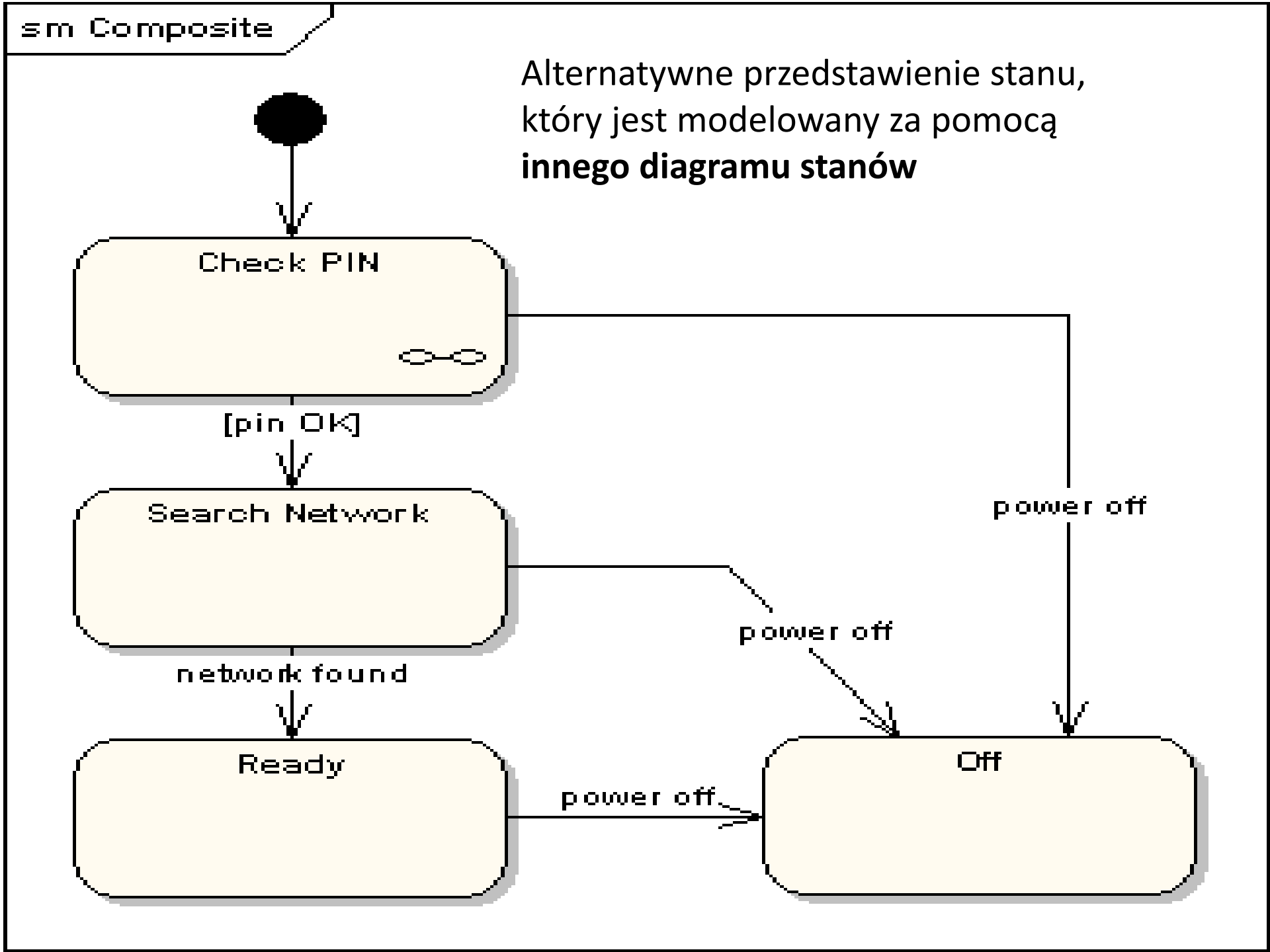
Przejście własne jest związkiem między tym samym stanem, który wskazuje, że np. obiekt znajdujący się w pewnym stanie wykona pewne **akcje** i powróci do tego samego stanu, ilekroć zaistnieje określone **zdarzenie i będą spełnione określone warunki**.

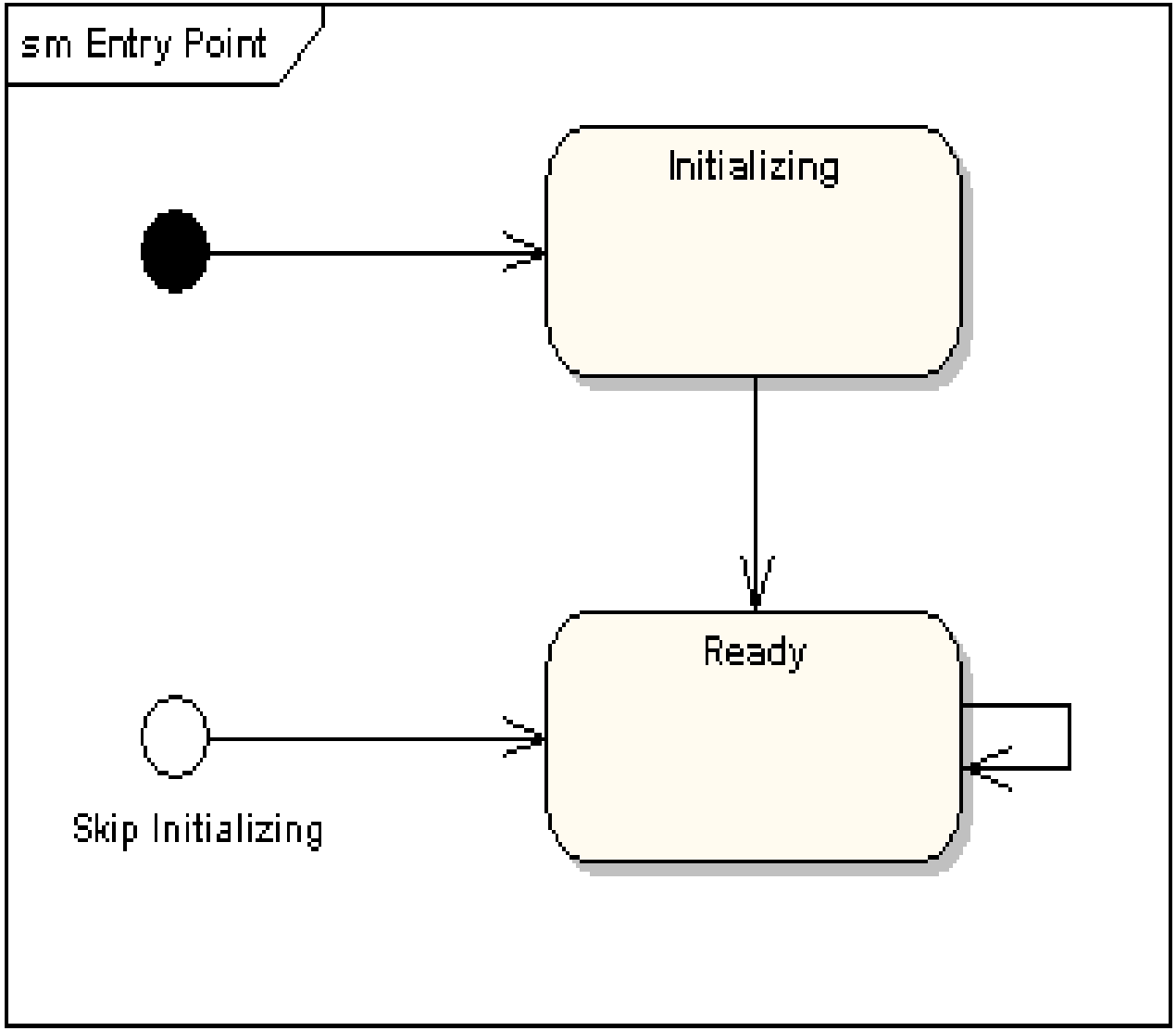




Poddiagramy stanów

Diagram stanu może zawierać stany, których zachowanie może być modelowany za pomocą **poddiagramów stanów**


sm Composite
Alternatywne przedstawienie stanu, który jest modelowany za pomocą **innego diagramu stanów**
Check PIN
[pin OK]
Search Network
network found
Ready
power off
power off
power off
Off



Stany początkowe w poddiagramach stanów

Wskazanie różnych stanów początkowych w **poddiagramie stanów**:

- rozpoczęcie stanu z inicjalizacją (normalne)
- bez inicjalizacji (wyjątkowe)

Punkty startowe w diagramach nadrzędnych

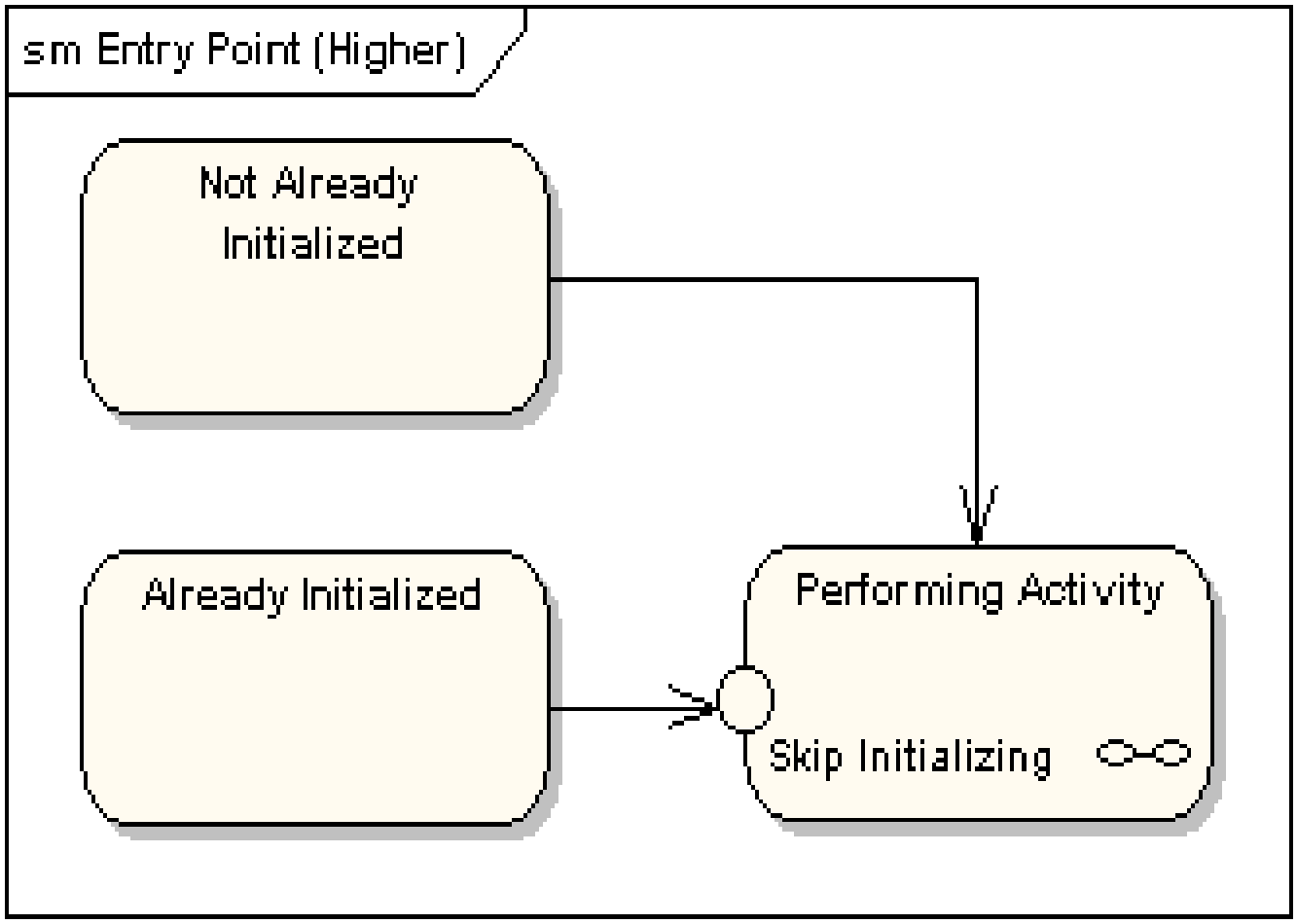
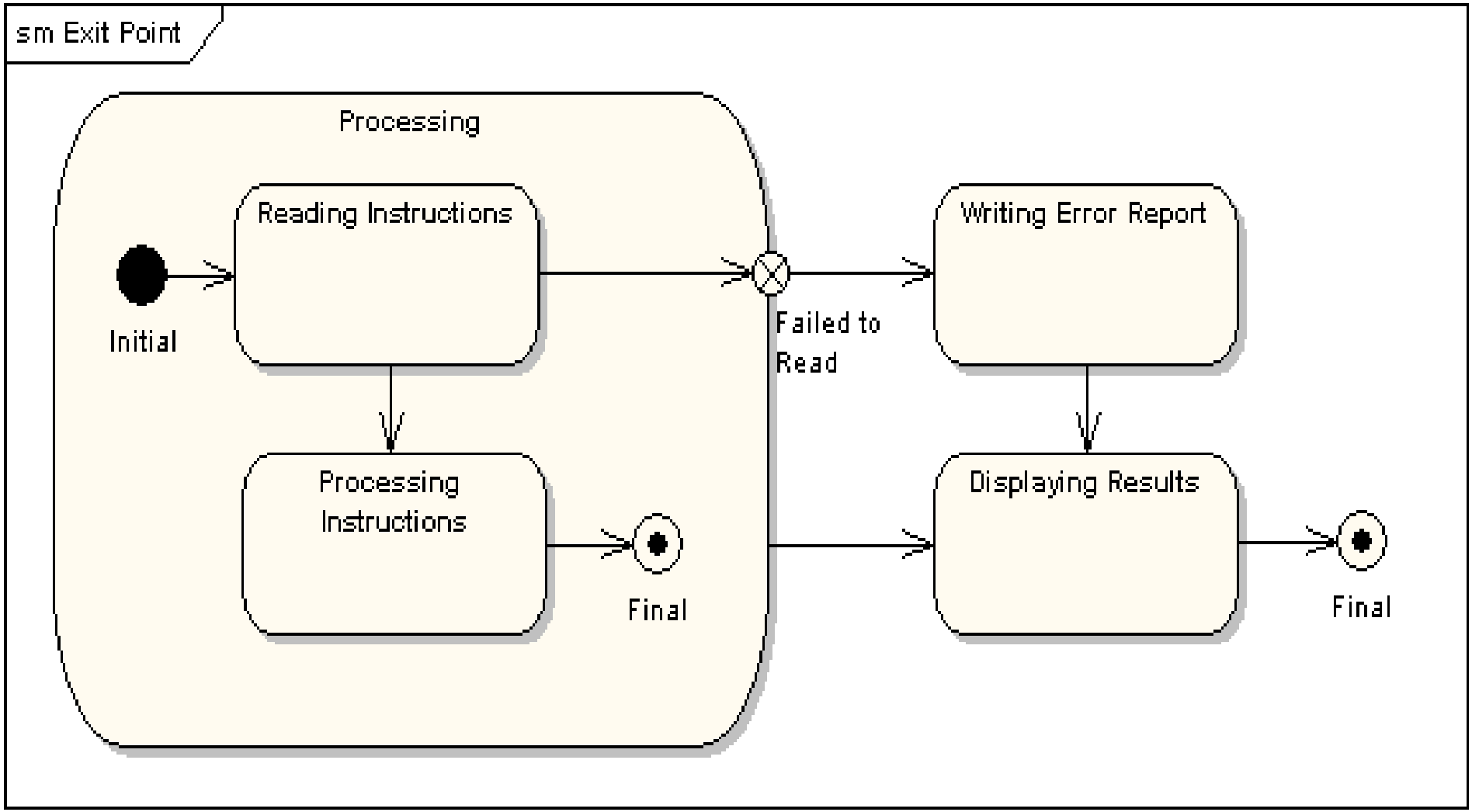


Diagram stanów zawierający różne punkty startowe dla poddiagramów stanów (reprezentowanych przez inne diagramy):

- rozpoczęcie stanu z inicjalizacją (normalne)
- bez inicjalizacji (wyjątkowe)

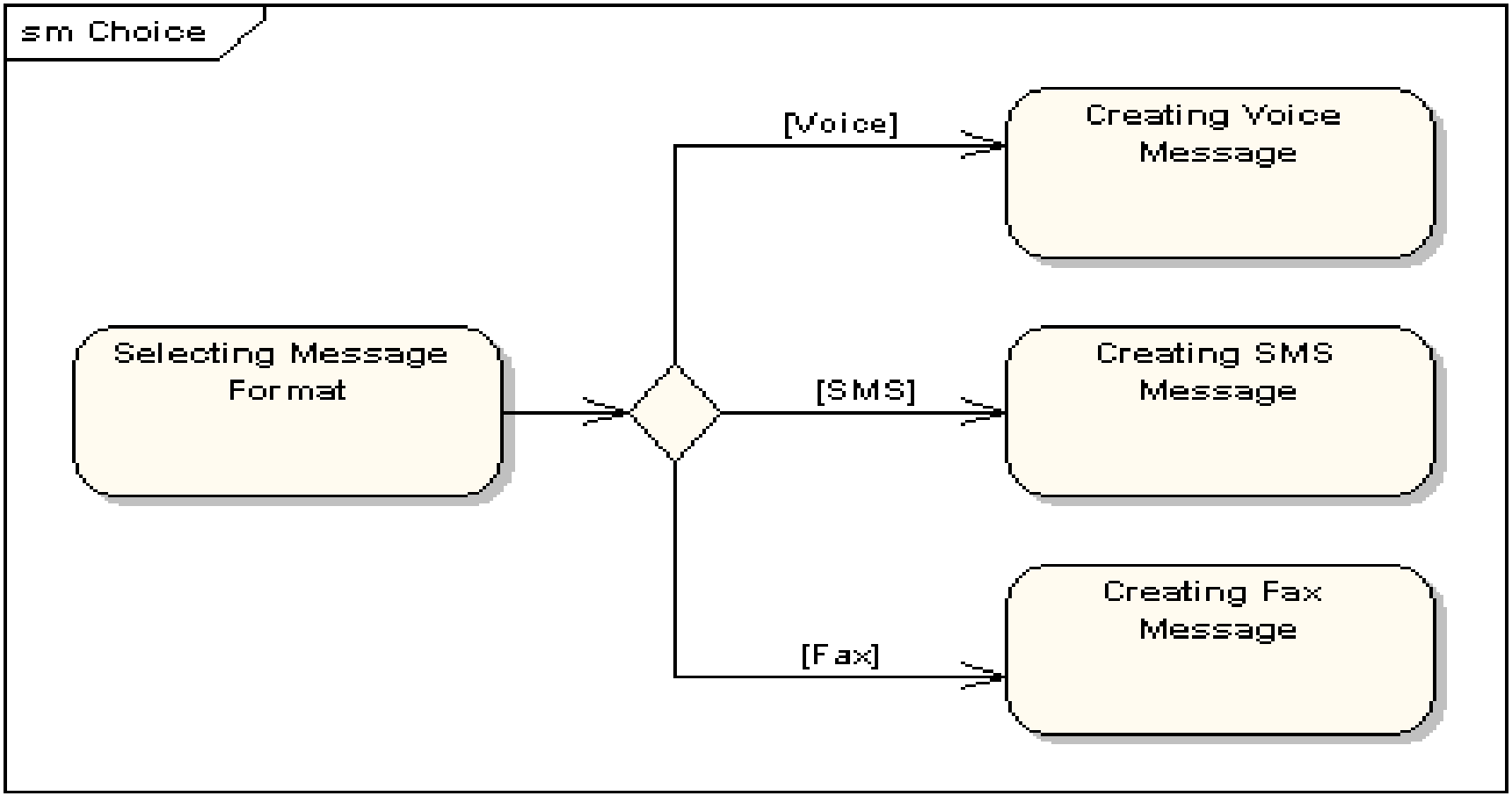
Punkt wyjścia – modelowanie osiągnięcia alternatywnych stanów końcowych (**Final**) przez obiekt



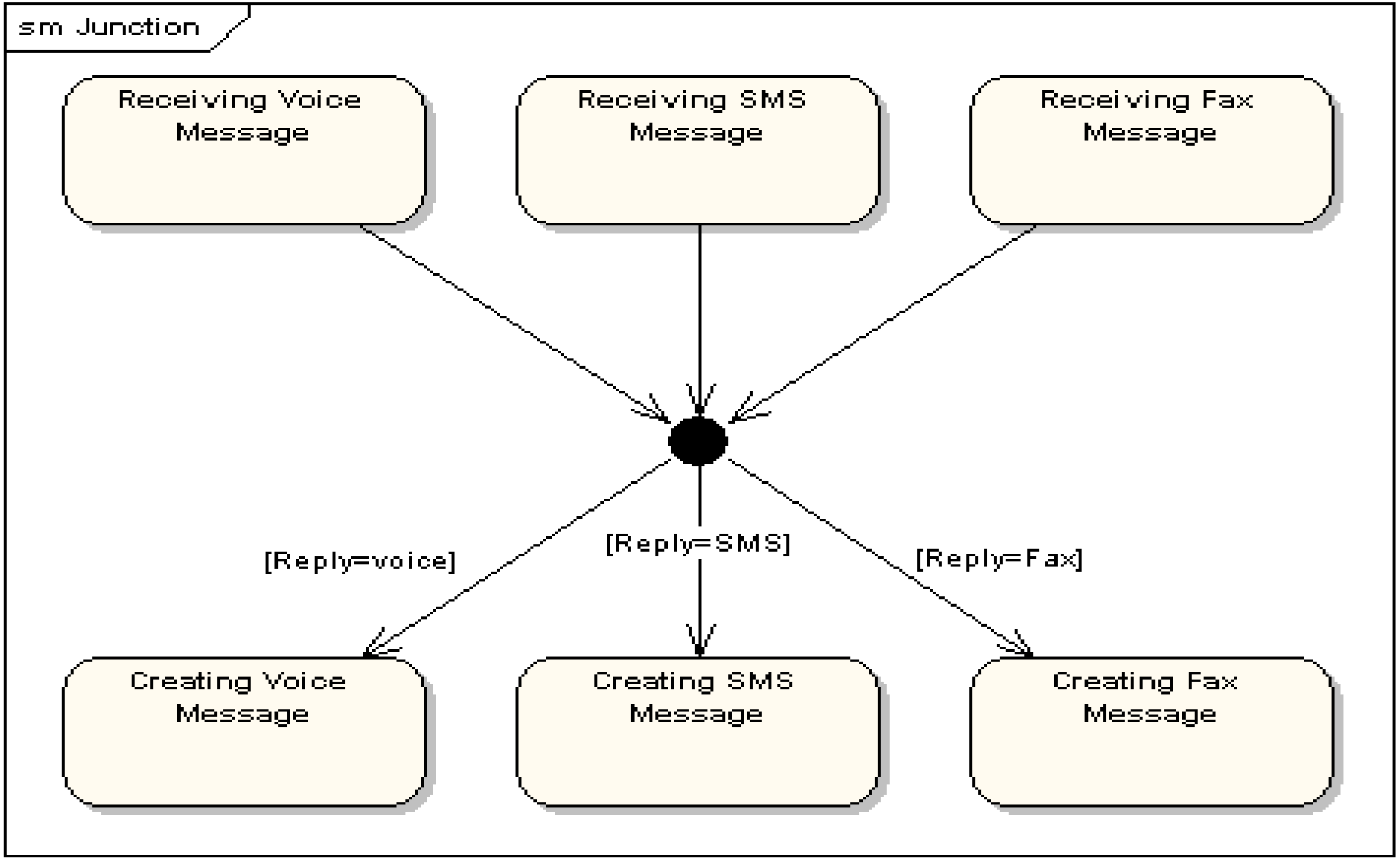
Pseudo stan wyboru:

• jedno przejście ze stanu wejściowego do **pseudo stanu wyboru (romb) i kilka przejść na wyjściu tego pseudo stanu**

•**w wyniku zdarzenia następuje przejście ze stanu wejściowego** (np. Selecting Message Format) **i na podstawie spełnionego warunku wybór przejścia do jednego ze stanów wyjściowych** (np. wybór przejścia na podstawie wybranego formatu wiadomości w stanie wejściowym); dynamiczny charakter wyboru przejścia

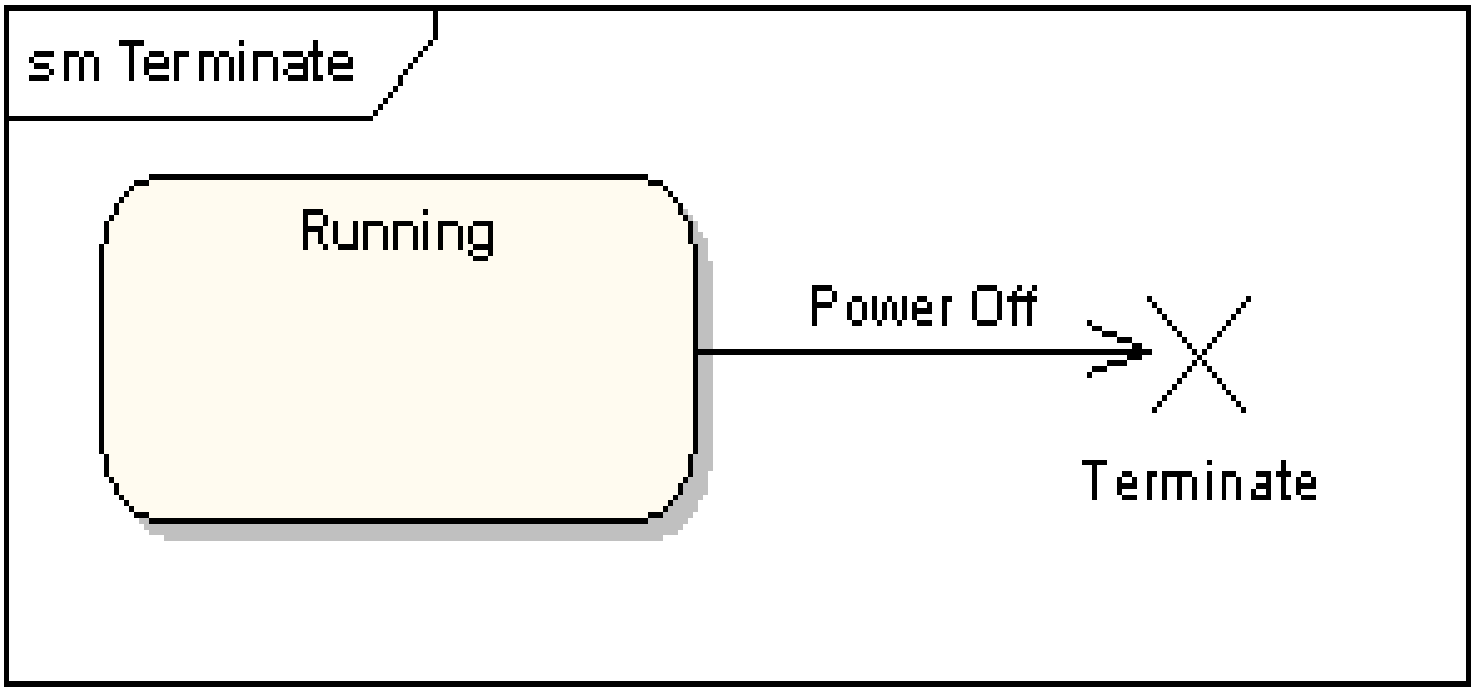


Pseudo stan typu połączenie – w pseudo stanie typu połączenie możliwość wyboru przejść do stanów wyjściowych po zdarzeniach zachodzących na przejściach ze stanów wejściowych

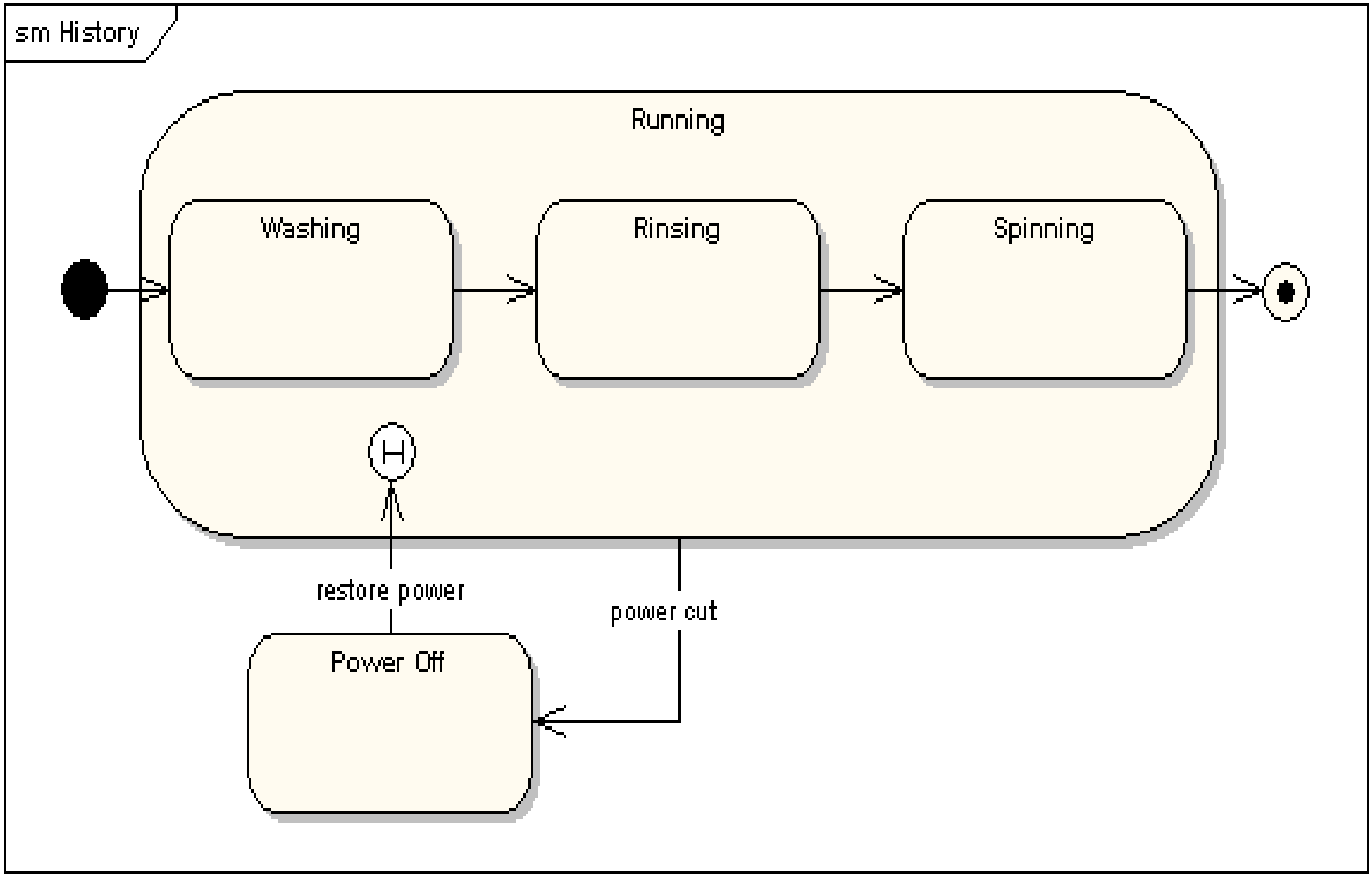


Pseudo stan typu zakończenie

oznacza zakończenie linii życia maszyny stanowej



Stany historyczne – przedstawiają stany wcześniejsze (historyczne) przed przerwaniem działania maszyny stanowej (np. w chwili załączenia zasilania maszyna stanowa zmywarki pamięta stan, w którym ma wznowić działanie)



Równolegle podstany

Stan może być podzielony między równoległe podstany wykonywane jednocześnie. Np. sterowanie przednimi (front) i tylnymi (rear) hamulcami odbywa się równolegle i musi być zsynchronizowane – wyrażone za pomocą symbolu rozdzielenia na pseudo stany oraz symbolu połączenia pseudo stanów. Równoległe podstany są używane do modelowania synchronizacji wątków

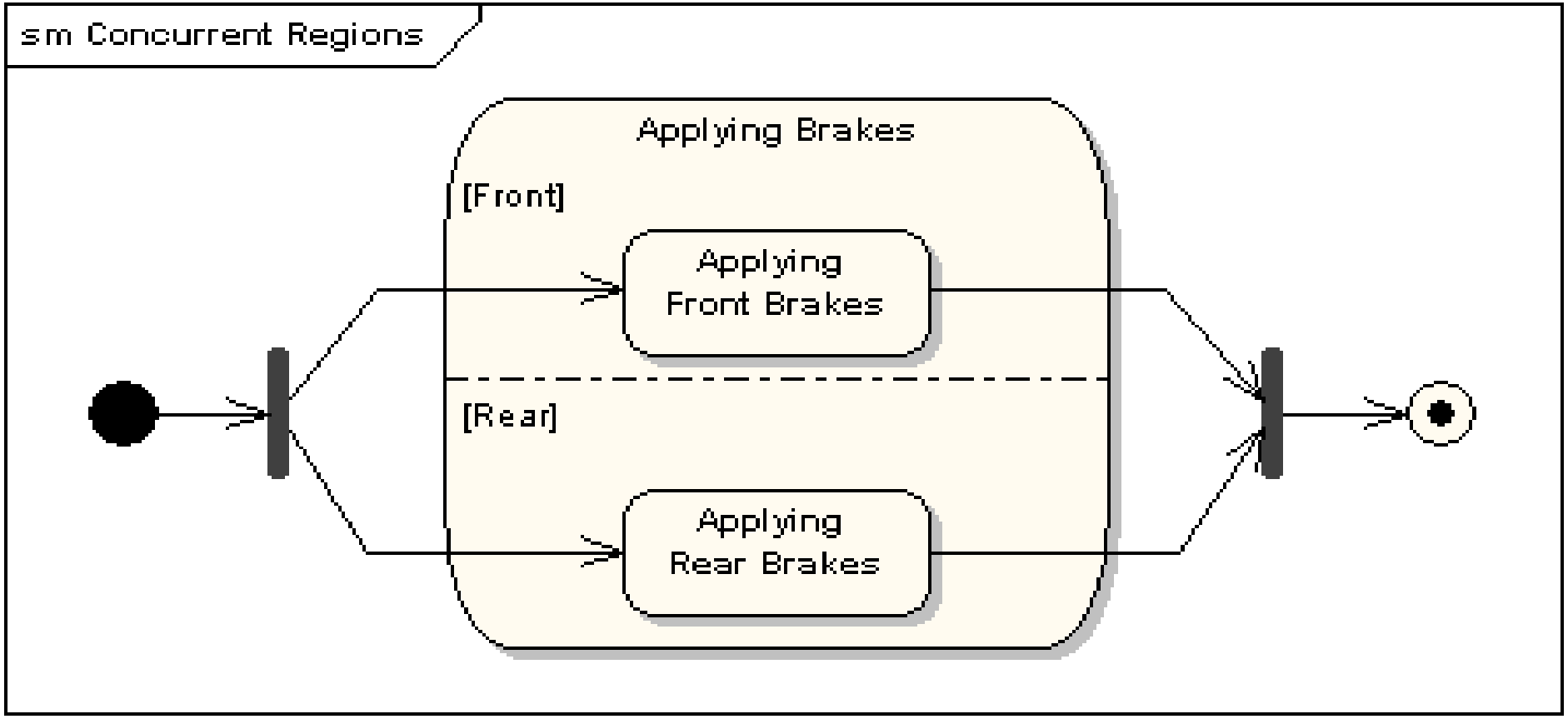


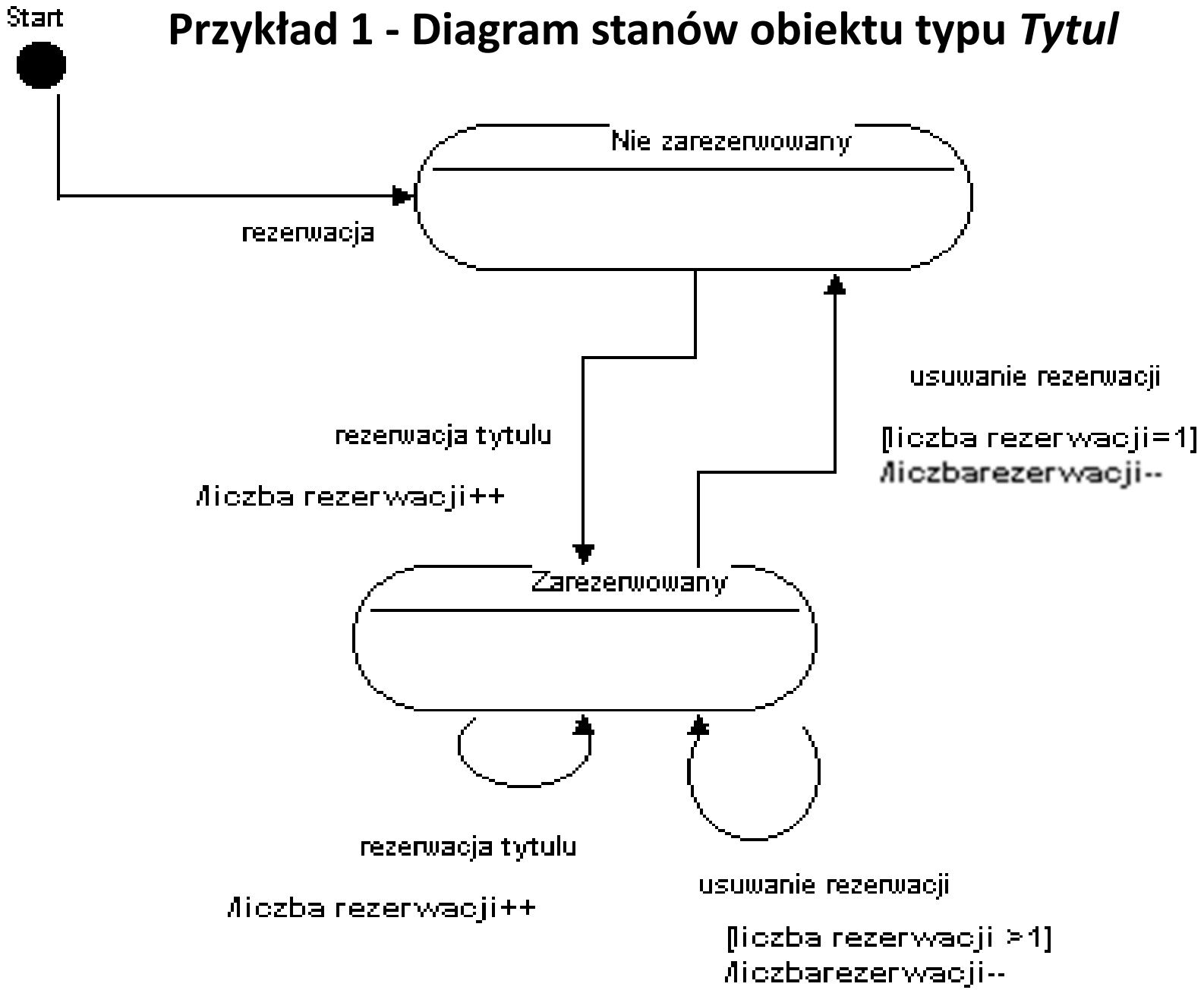
Diagram stanów

1. **Diagramy stanów UML**

https://sparxsystems.com/resources/tutorials/uml2/state-diagram.html

2. **Przykład diagramów stanów UML – modelowanie wpływu przypadków użycia na stany obiektu**

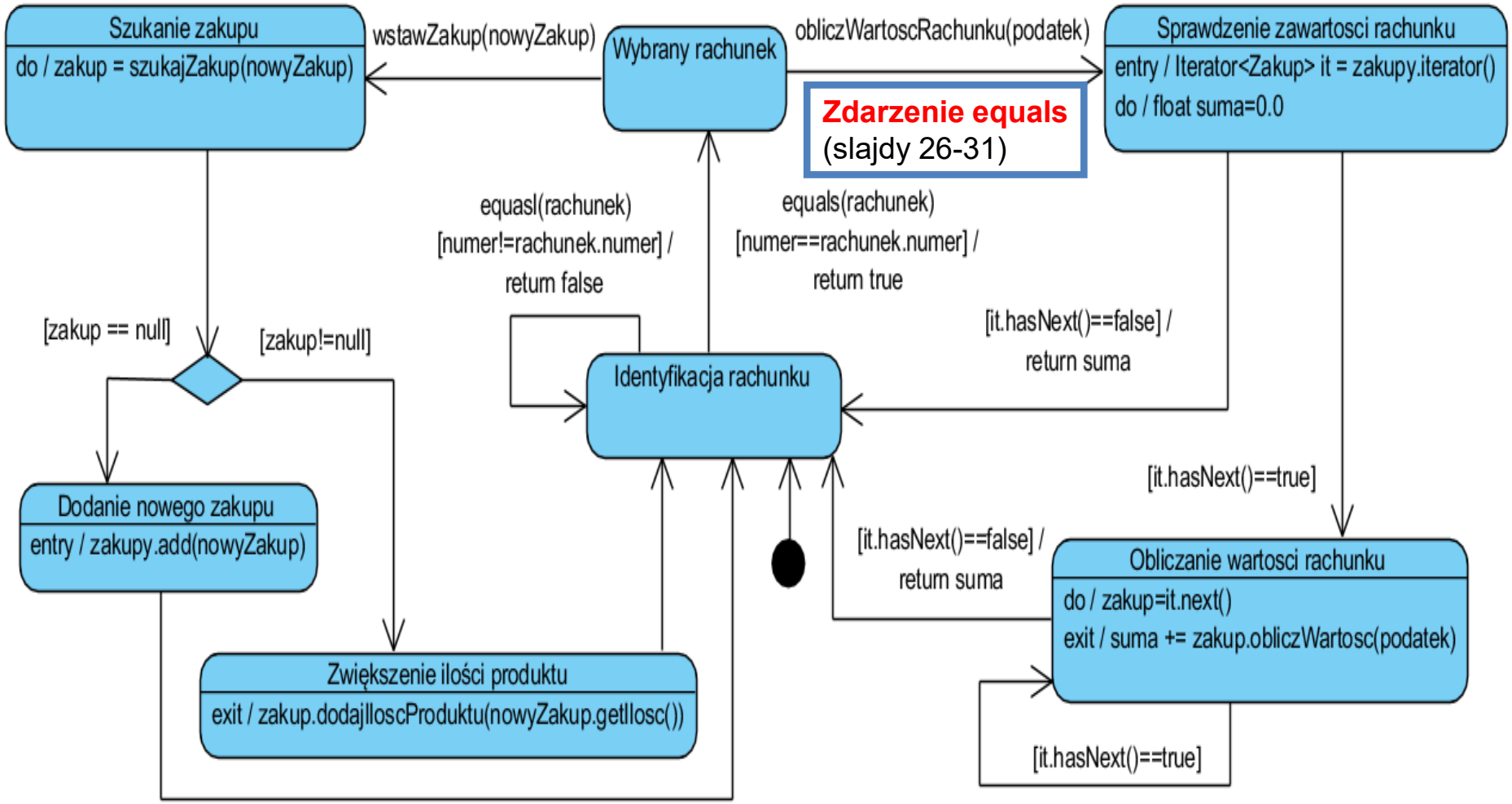
Przykład 1 - Diagram stanów obiektu typu *Tytul*



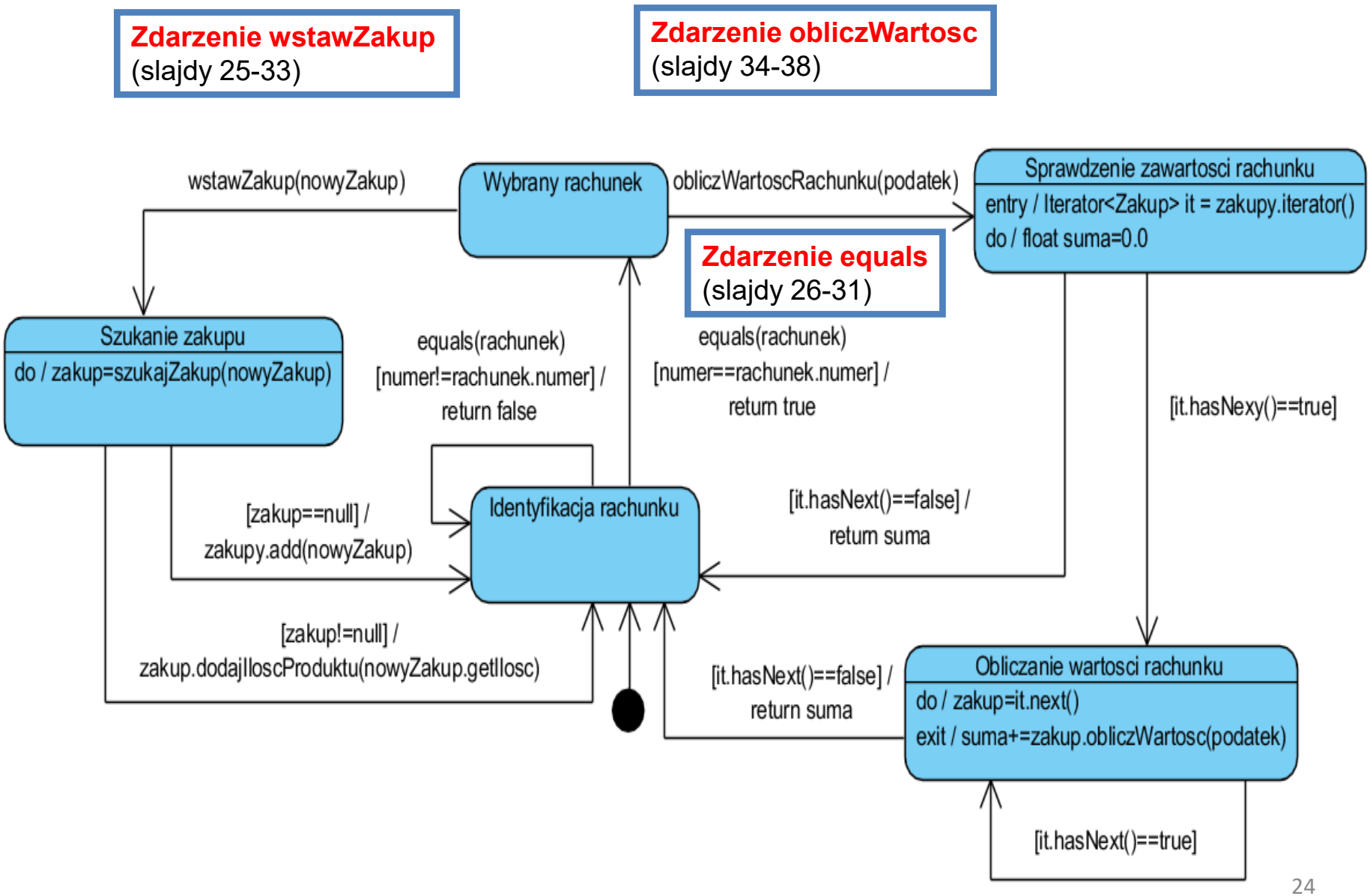
Przykład 2 – Diagram stanów klasy *Rachunek* (wersja 1)

Zdarzenie wstawZakup
(slajdy 25-33)

Zdarzenie obliczWartosc
(slajdy 34-38)

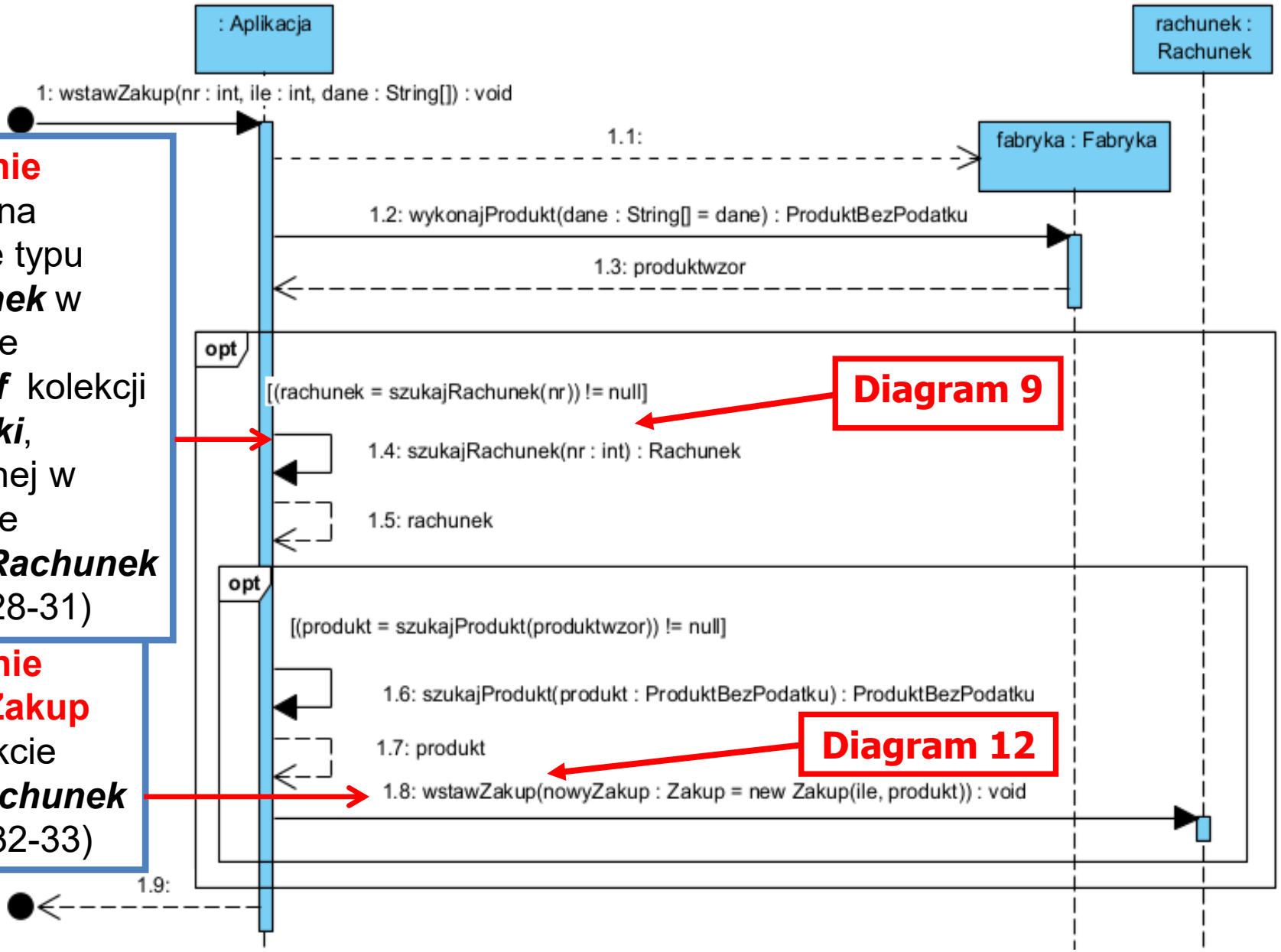


Przykład 2 (cd) – Diagram stanów klasy *Rachunek* (wersja 2)



Projekt przypadku użycia –
zdarzenie wstawZakup
„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

Wstawianie nowego zakupu – generowanie zdarzeń equals i wstawZakup na obiekcie typu Rachunek



Zdarzenie equals na obiekcie typu ***Rachunek*** w metodzie ***indexOf*** kolekcji ***rachunki***, wywołanej w metodzie ***szukajRachunek*** (slajdy 28-31)

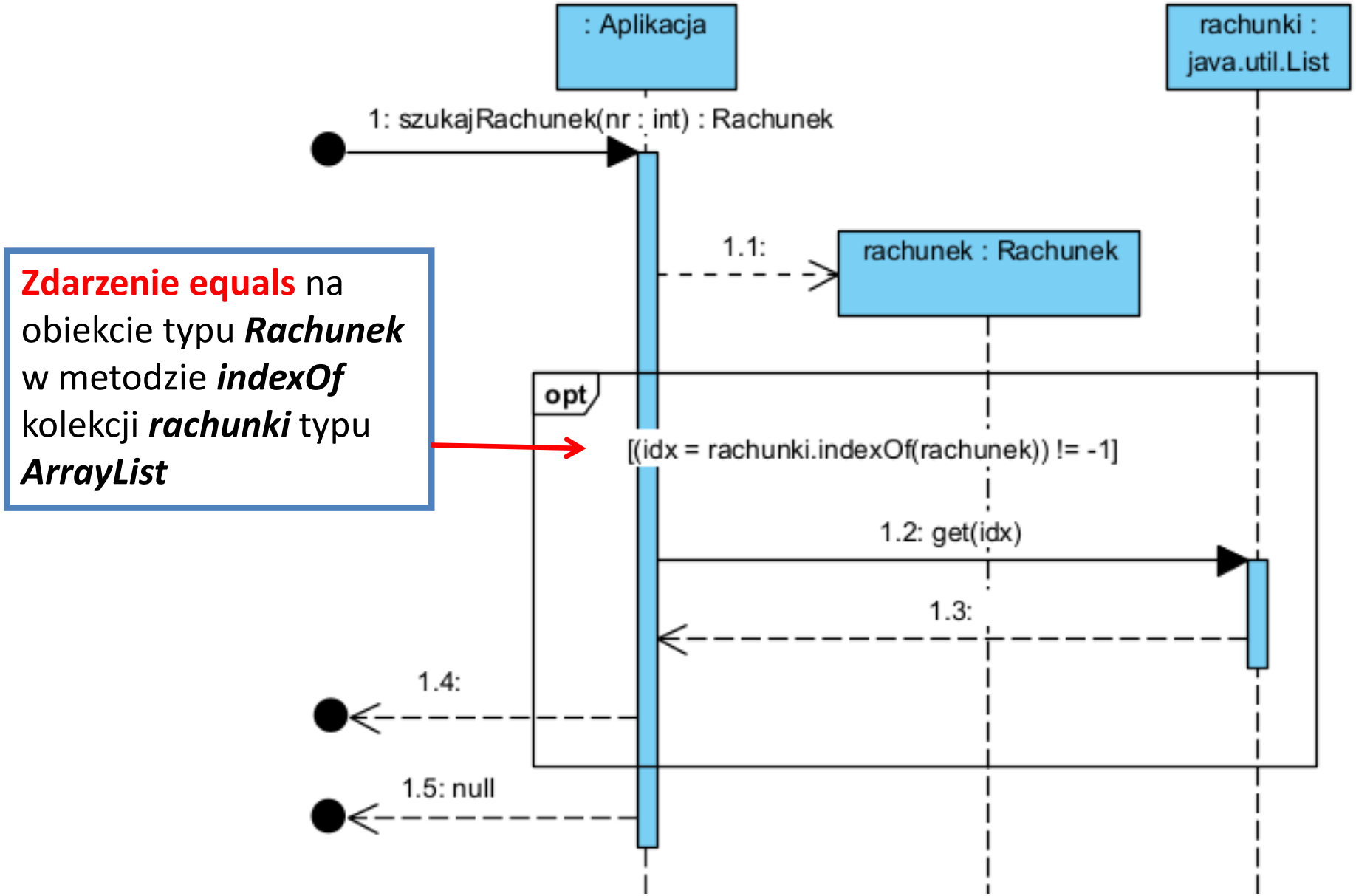
Zdarzenie wstawZakup na obiekcie typu ***Rachunek*** (slajdy 32-33)

```
                                                    //Aplikacja

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

PU Szukanie rachunku

(9) Rachunek szukajRachunek(int nr)



Zdarzenie equals na obiekcie typu ***Rachunek*** w metodzie ***indexOf*** kolekcji ***rachunki*** typu ***ArrayList***

```
                                                    //Aplikacja
private List <Rachunek> rachunki  = new ArrayList <>();

 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;
  }
```

//metoda indexOf obiektu rachunki klasy typu ArrayList

```
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;
   }
```

Zdarzenie equals na obiekcie o typu ***Rachunek*** podstawionym pod parametr typu ***Object*** metody ***indexOf*** kolekcji ***rachunki*** typu ***ArrayList***

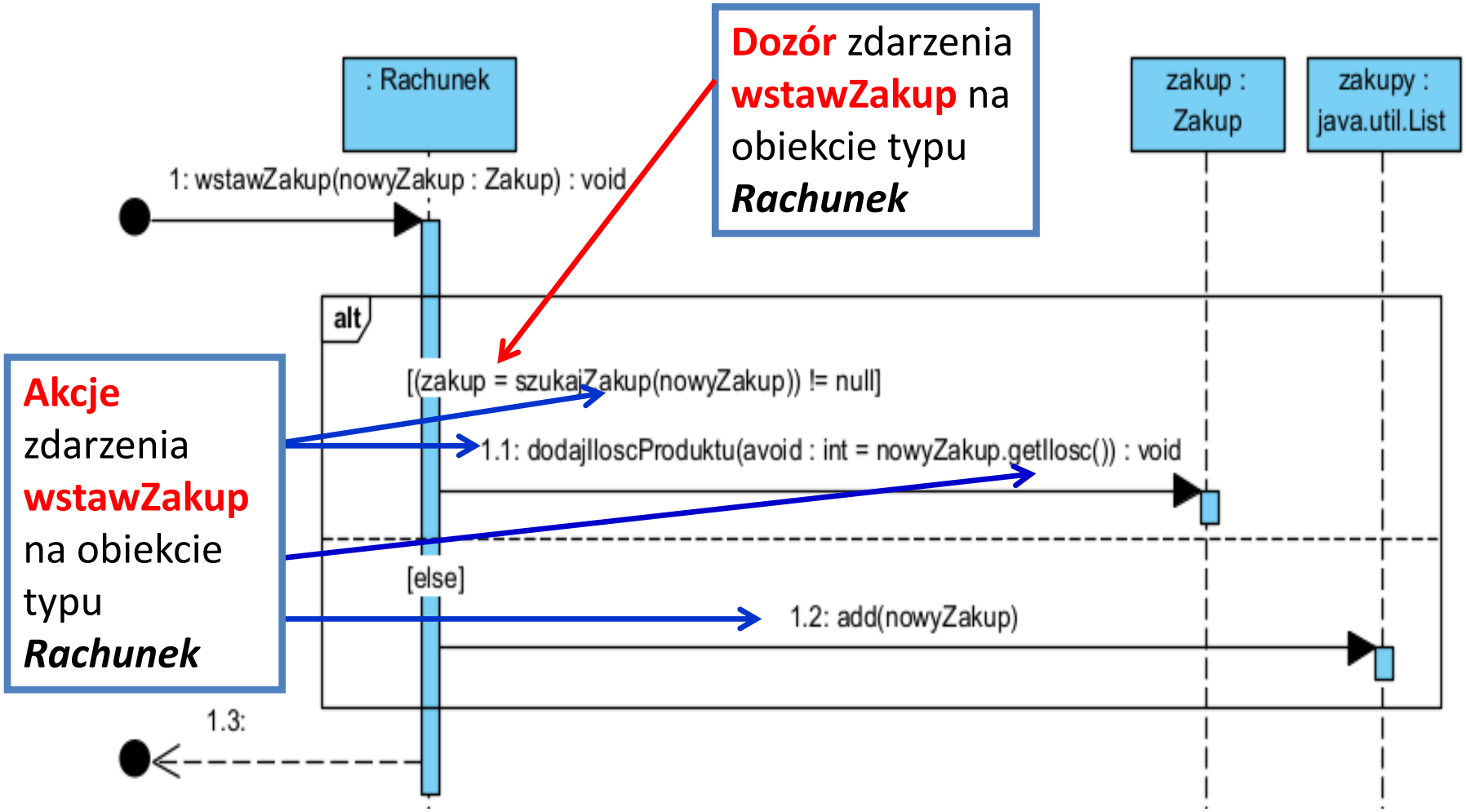
```
                                                    //Rachunek
 //metoda zdarzeniowa equals
// metody użyte w kodzie metody są akcjami zdarzenia
//instrukcje warunkowe mogą być użyte jako dozory

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

Dozór zdarzenia **equals** na obiekcie typu ***Rachunek***

Akcja zdarzenia **equals** na obiekcie typu ***Rachunek*** – zwrócenie wyniku dozoru **true** lub **false**

(12) void wstawZakup(Zakup azakup) – metoda zdarzeniowa na obiekcie typu Rachunek



//Rachunek

```
//metoda zdarzeniowa wstawZakup
// metody użyte w kodzie metody są akcjami zdarzenia
//instrukcje warunkowe mogą być użyte jako dozory

 private ArrayList<Zakup> zakupy = new ArrayList<Zakup>();

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

Projekt przypadku użycia

zdarzenie Podaj_wartosc

„**Obliczanie wartości rachunku**”

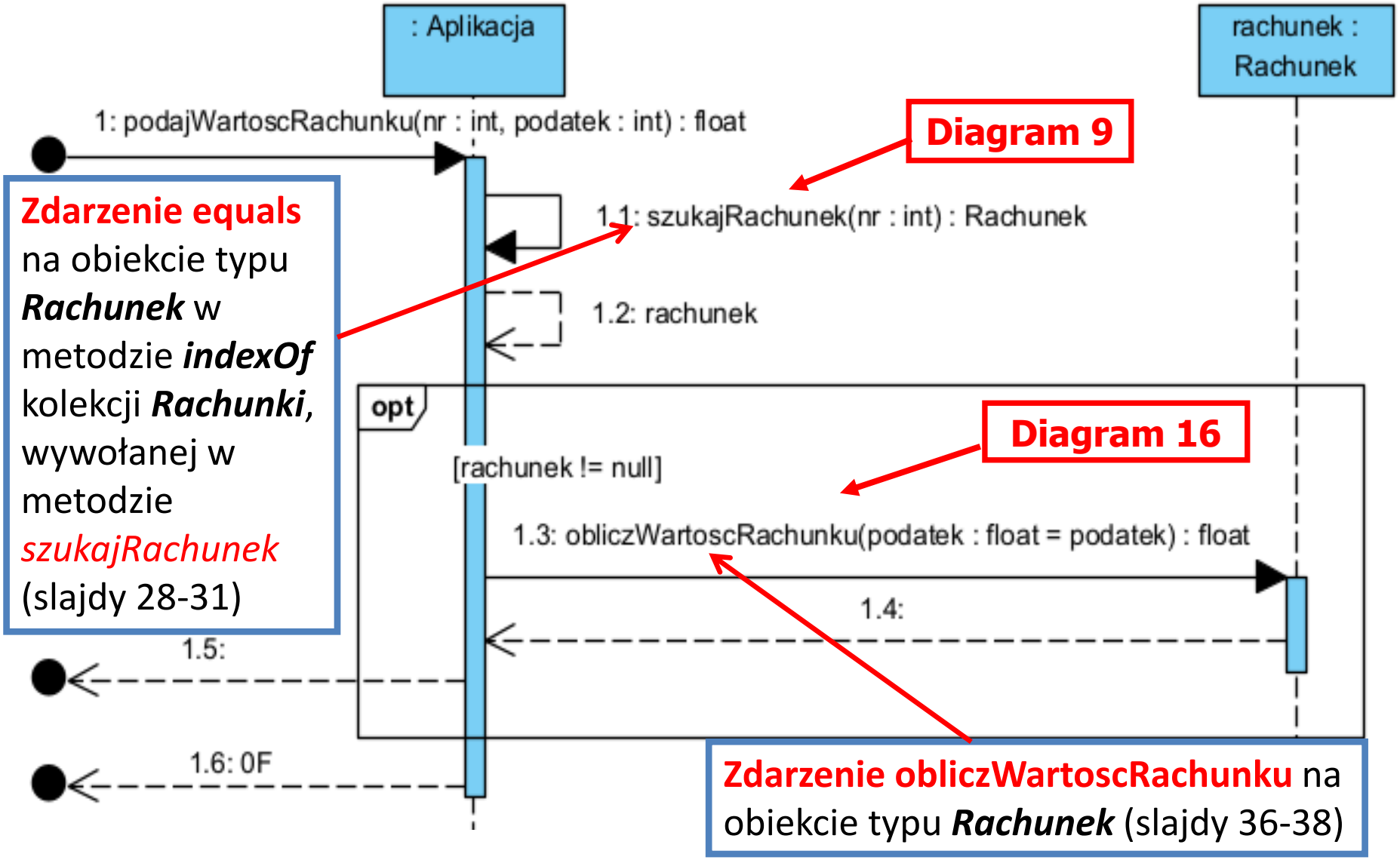
za pomocą diagramu sekwencji i diagramu klas.

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

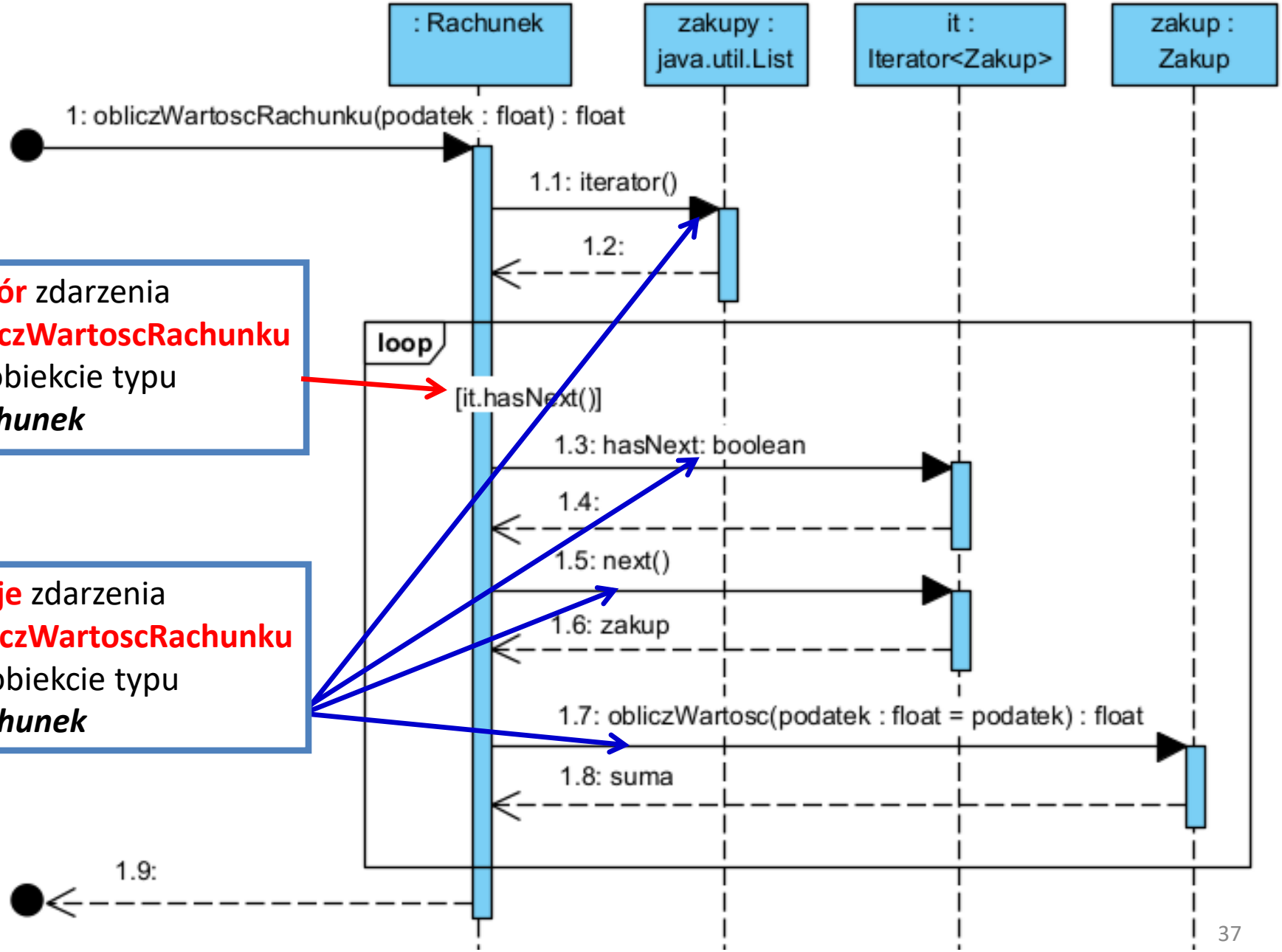
(15) Obliczanie wartosci rachunku – generowanie zdarzeń equals i podajWartosc na obiekcie typu Rachunek (float obliczWartoscRachunku(int nr, int podatek))



```
                                                    //Aplikacja

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

(16) float obliczWartoscRachunku(int podatek)



Dozór zdarzenia **obliczWartoscRachunku** na obiekcie typu ***Rachunek***

Akcje zdarzenia **obliczWartoscRachunku** na obiekcie typu ***Rachunek***

//Rachunek

```
//metoda zdarzeniowa obliczWartoscRachunku
// metody użyte w kodzie metody są akcjami zdarzenia
//instrukcje warunkowe mogą być użyte jako dozory

private ArrayList<Zakup> zakupy = new ArrayList<>();

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;
 }
```