Technologia Programowania 2017/2018

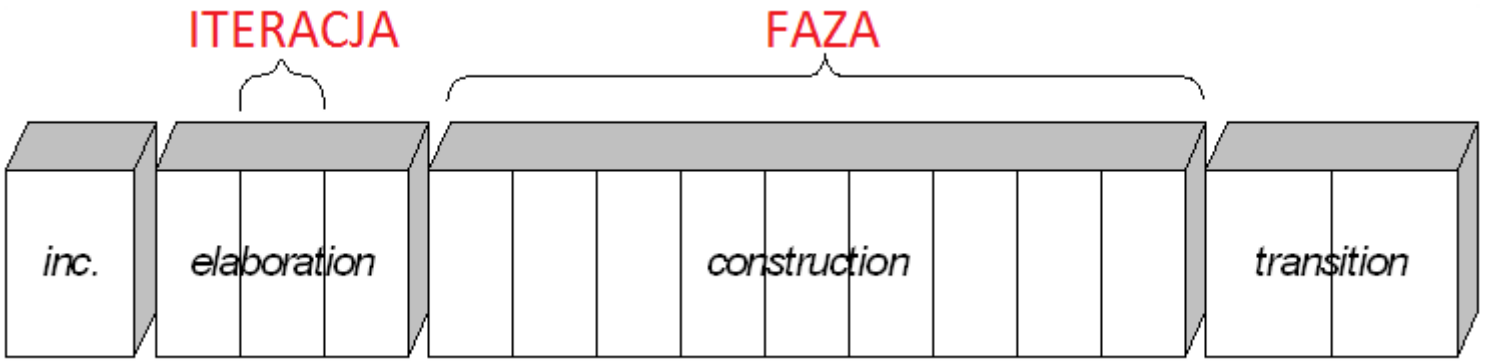
Wykład 2

„Wszystko, czego nie chcieliście wiedzieć o UML i dlatego o to nie pytaliście”

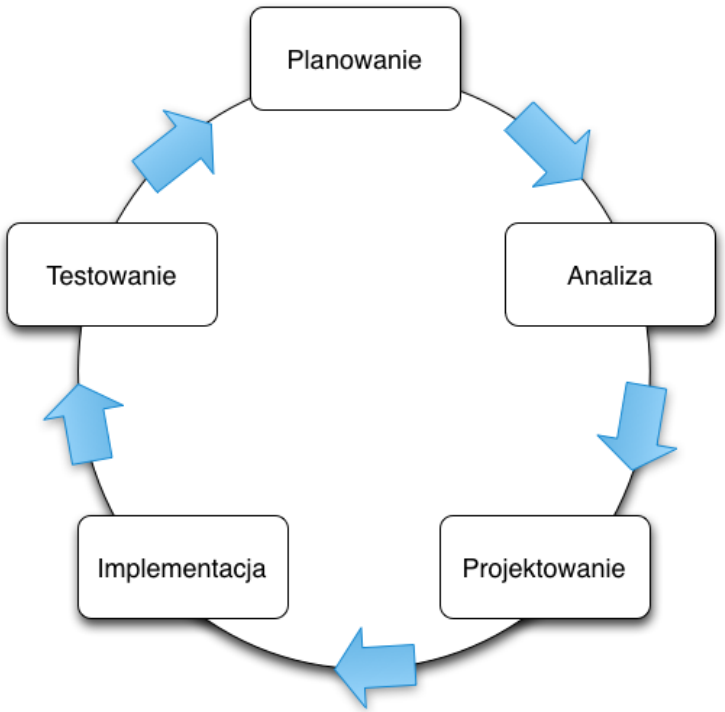
Jakub Lemiesz

Wiele z prezentowanych diagramów pochodzi z książki C. Larman'a „UML i wzorce projektowe”

Metodyki zwinne – gdzie jesteśmy?

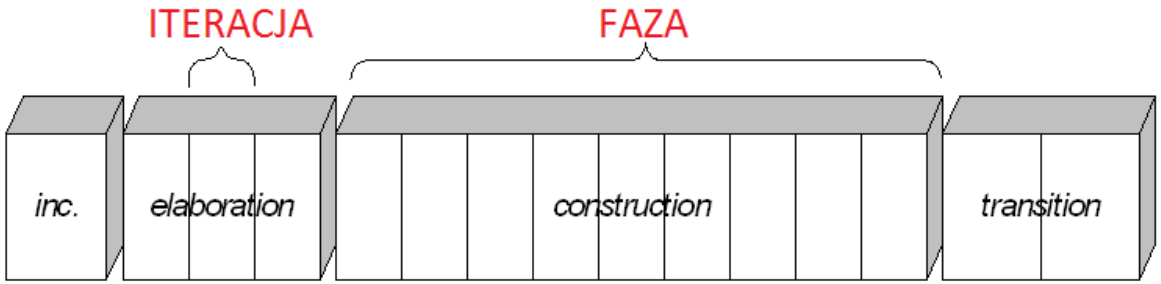


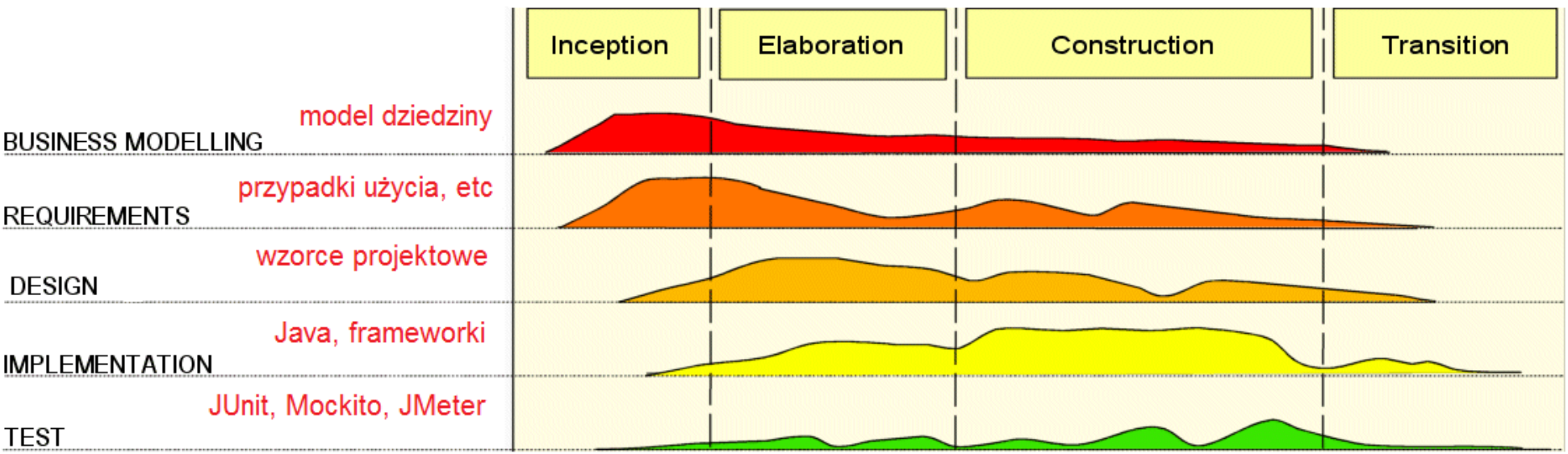
ITERACJA:



Źródło: usetheawesomenessluke.com

Metodyki zwinne – fazy na przykładzie RUP





Requirements: kolejność powstawania artefaktów

diagram przypadków użycia

↓

scenariusze (*główne, alternatywne...*)

↓

systemowe diagramy sekwencji

↓

operacje systemowe

operacje systemowe
model dziedziny
} → kontrakty operacji

↓

projektowanie

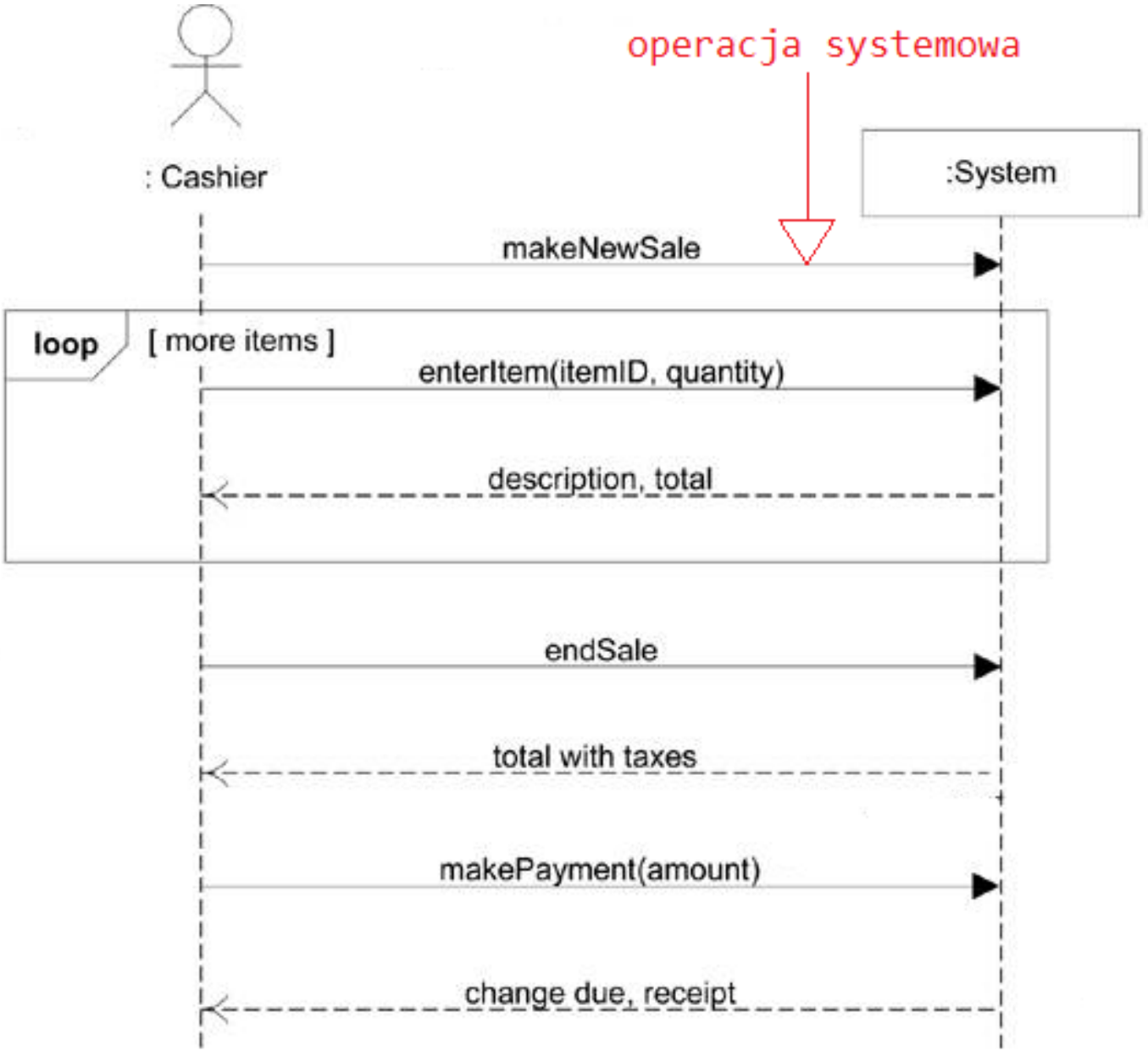
Systemowe diagramy sekwencji

Systemowe diagramy sekwencji powstają na podstawie scenariuszy i sugerowanych w nich operacji systemowych (*operacji przekraczających granice systemu*)

Main Success Scenario:

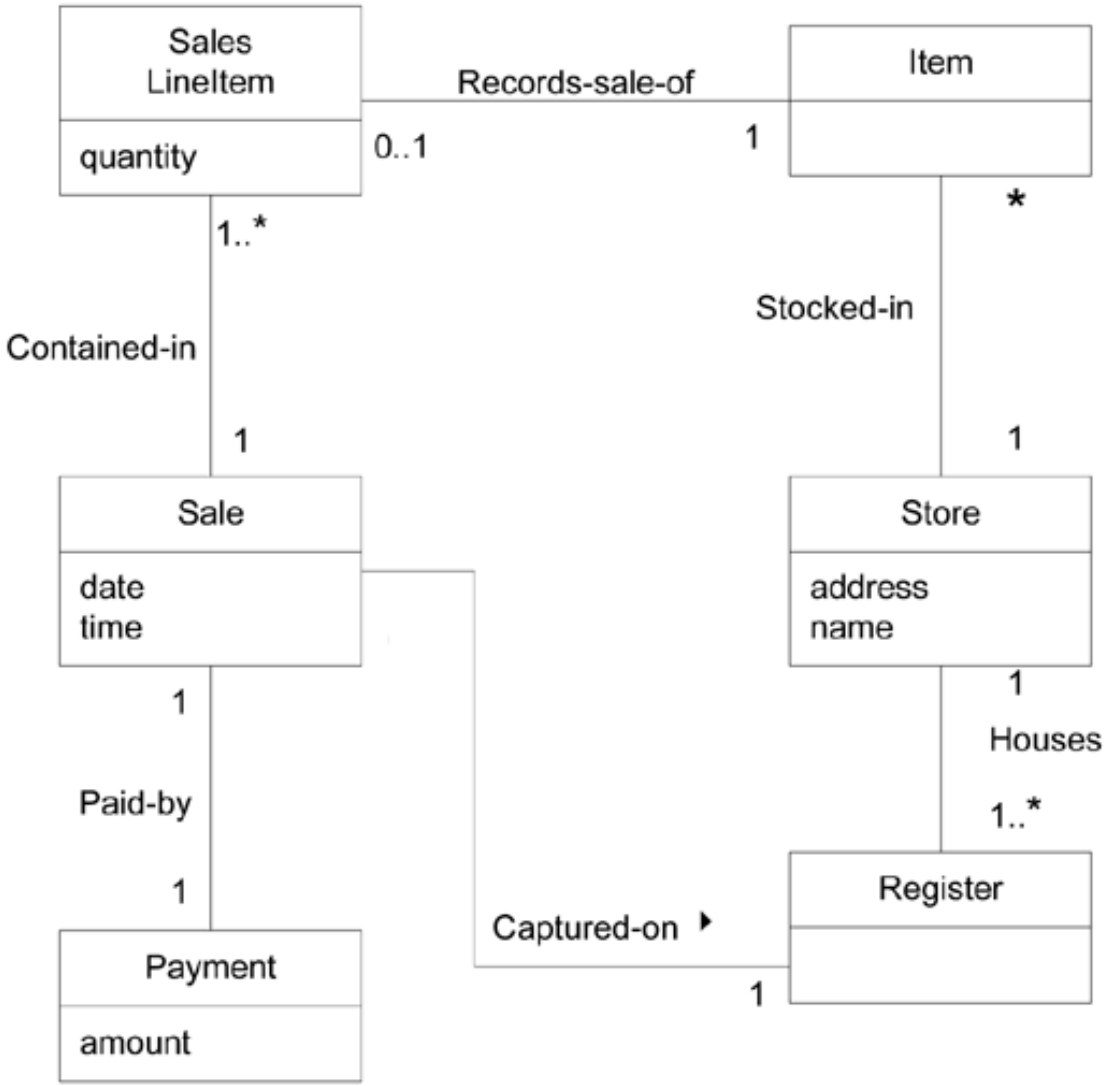
1. Customer arrives at a POS checkout with items to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records item and presents total.
5. Cashier repeats steps 3-4 for all items.
6. System presents total with taxes calculated.
7. ...

Systemowy diagram sekwencji



Kontrakty operacji

model dziedziny + operacje systemowe $\Longrightarrow$ kontrakty operacji



Kontrakty operacji — przykład

Operacja: `makePayment(amount: Money)`

Warunki początkowe: Odbywa się sprzedaż

Warunki końcowe:

1. Utworzona została instancja klasy `Payment` o nazwie `p` (*utworzenie instancji*)
2. `p.amount` otrzymał wartość parametru `amount` (*zmiana wartości atrybutu*)
3. `p` został powiązany z bieżącą sprzedażą, instancją `Sale` (*nawiązanie asocjacji*)
4. Bieżąca sprzedaż została powiązana z instancją `Register` w celu archiwizacji (*nawiązanie asocjacji*)

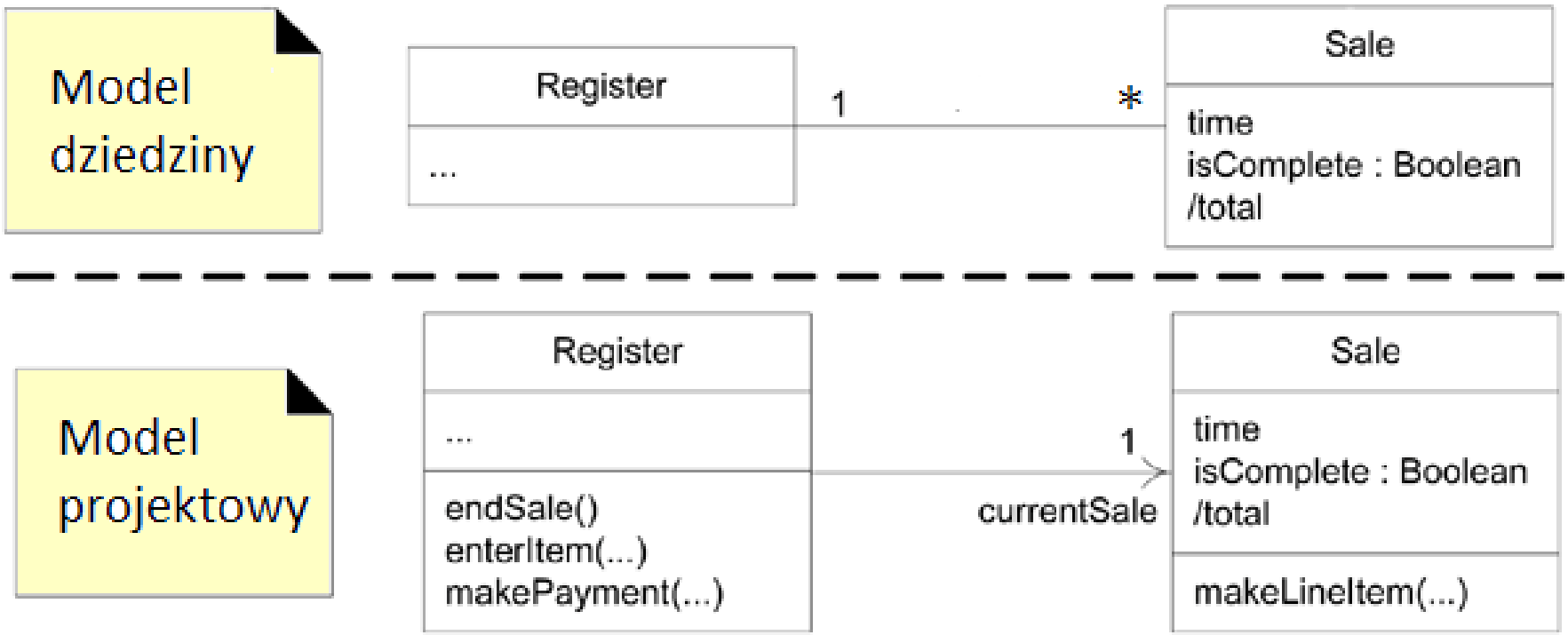
Analiza ⇒ Projektowanie

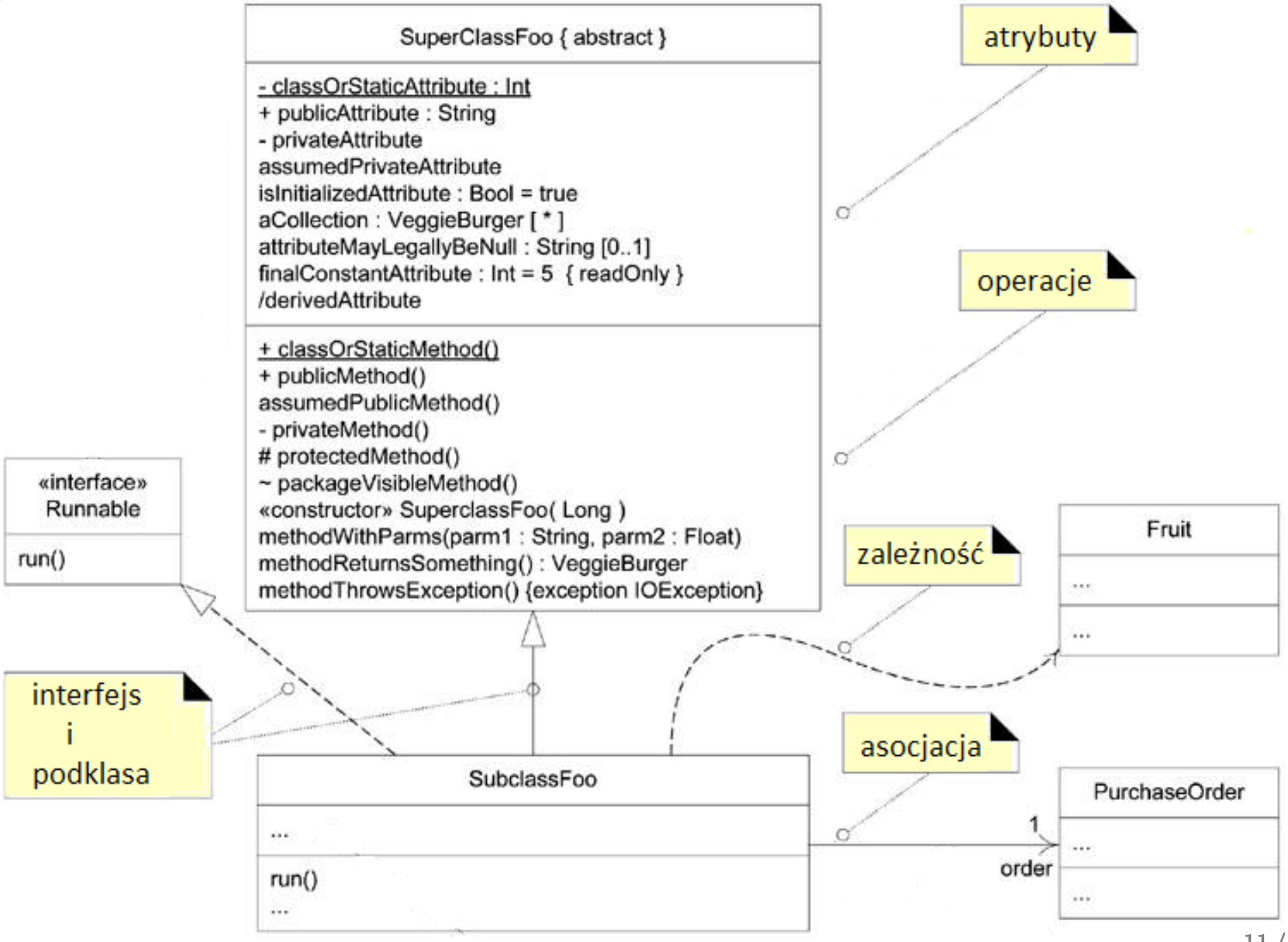
Kontrakty operacji:

- opisują zmiany stanu obiektów:
  1. utworzenie/usunięcie instancji
  2. nawiązanie/zerwanie asocjacji
  3. zmiana wartości atrybutu
- określają wstępne wymagania dla projektowego modelu klas

Projektowy model klas

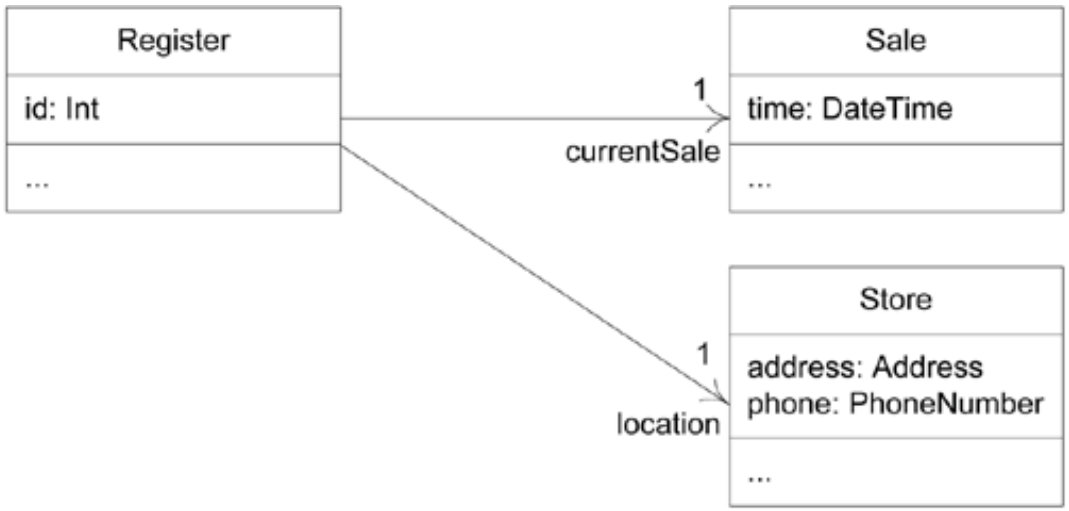
UML ma bogatą notacje – trzeba wiedzieć co dany element oznacza w danym kontekście




SuperClassFoo { abstract }
- classOrStaticAttribute : Int
+ publicAttribute : String
- privateAttribute
assumedPrivateAttribute
isInitializedAttribute : Bool = true
aCollection : VeggieBurger [*]
attributeMayLegallyBeNull : String [0..1]
finalConstantAttribute : Int = 5 { readOnly }
/derivedAttribute
+ classOrStaticMethod()
+ publicMethod()
assumedPublicMethod()
- privateMethod()
protectedMethod()
~ packageVisibleMethod()
«constructor» SuperclassFoo(Long)
methodWithParms(parm1 : String, parm2 : Float)
methodReturnsSomething() : VeggieBurger
methodThrowsException() {exception IOException}
atrybuty
operacje
«interface»
Runnable
run()
zależność
Fruit
...
...
interfejs
i
podklasa
asocjacja
SubclassFoo
...
run()
...
PurchaseOrder
...
...
1
order

Model klas – atrybuty vs. asocjacje

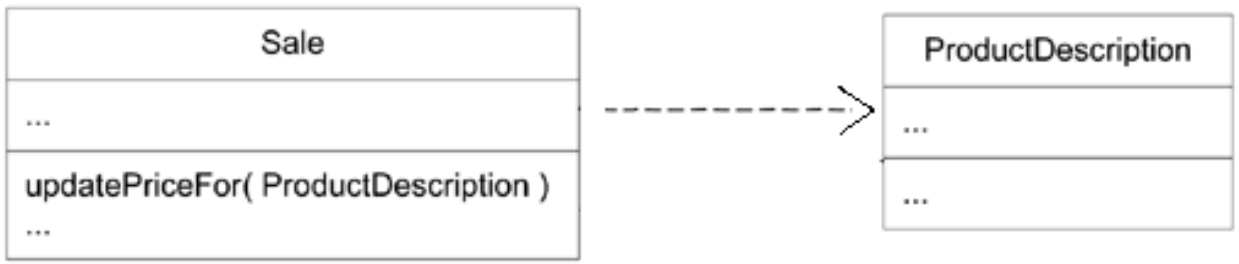
```
public class Register
{
 private int id;
 private Sale currentSale;
 private Store location;
...
```



- Strzałka nawigacyjna skierowana do celu
- Krotność i nazwa tylko po stronie celu
- Nazwa odpowiada nazwie atrybutu

Diagram klas – parametry operacji, operacje statyczne

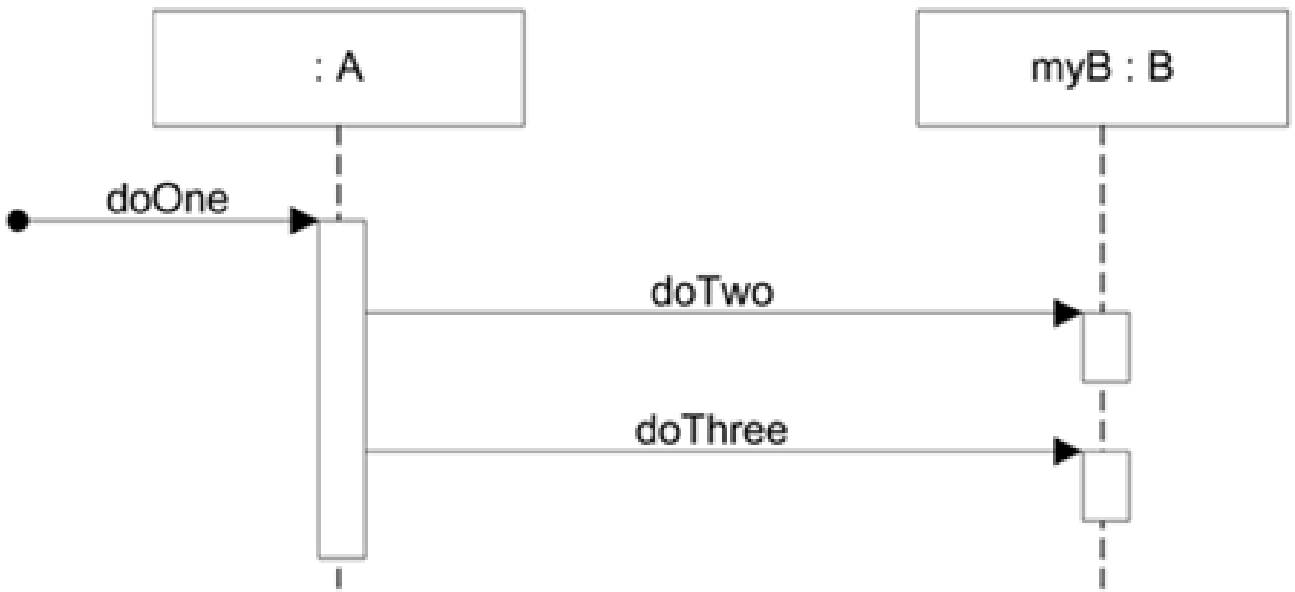
```
public class Sale{
    public void updatePriceFor(ProductDescription description){
```




```
public class Foo{
    public void doX(){
        System.runFinalization();   ...
```

Diagram sekwencji

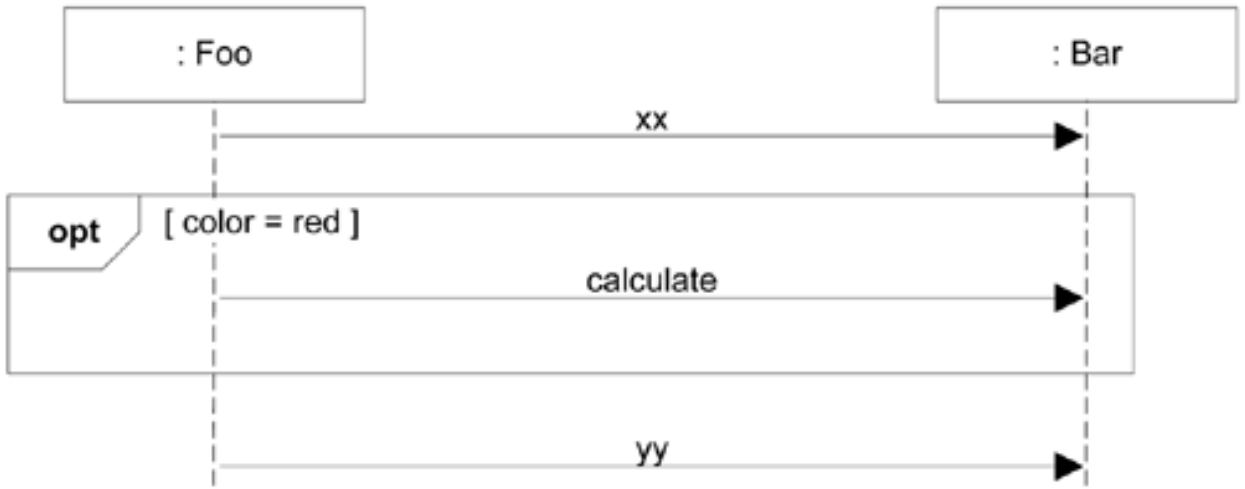
```
public class A {
private B myB = ...
public void doOne() {
   myB.doTwo();
   myB.doThree();
```



– elementy: instancja, linia życia, aktywność, komunikat,...

– nazwa instancji przed :ClassName gdy ma to znaczenie

Diagram sekwencji - ramki warunku



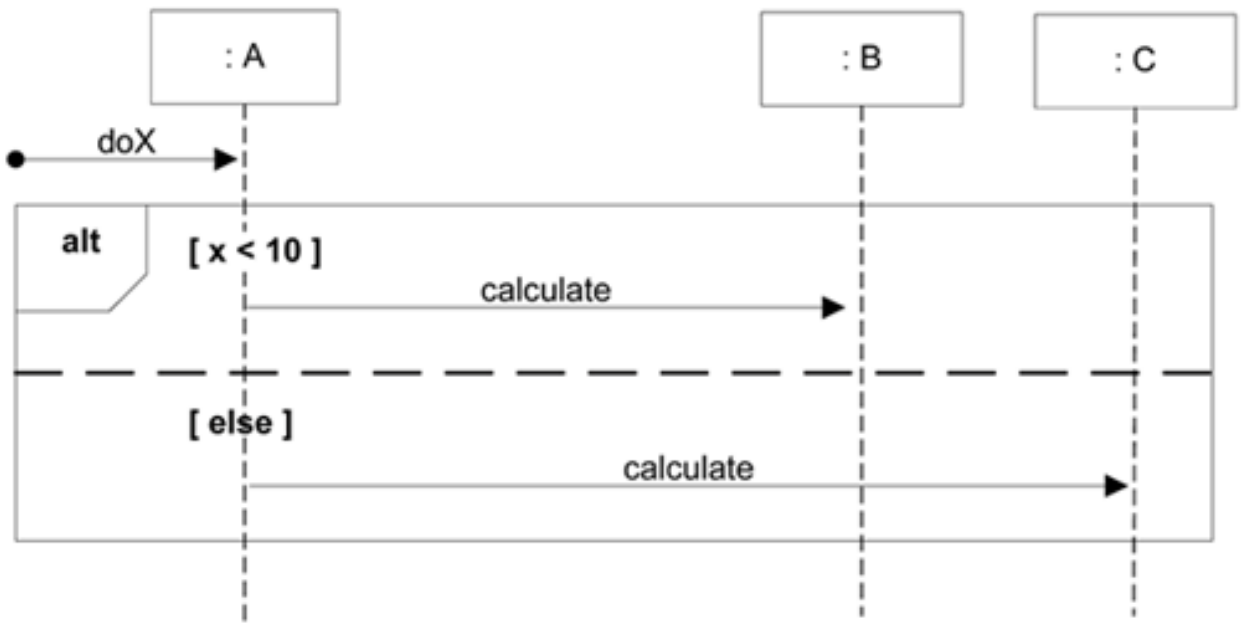


Diagram sekwencji - ramka pętli

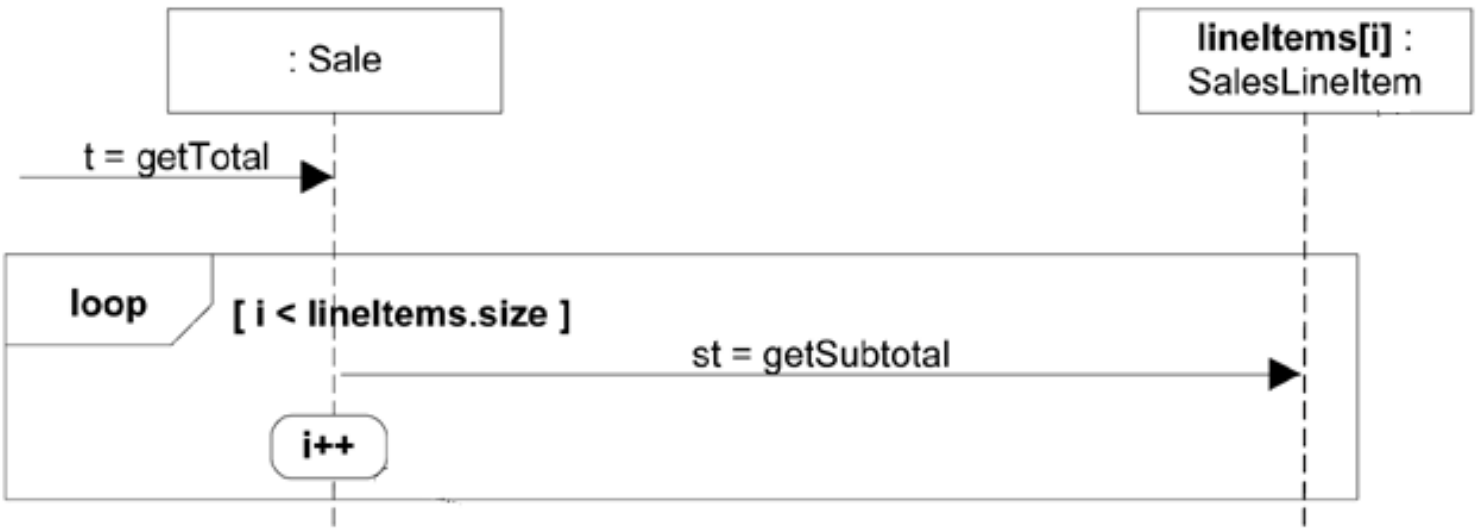


Diagram sekwencji - zagnieżdżanie ramek

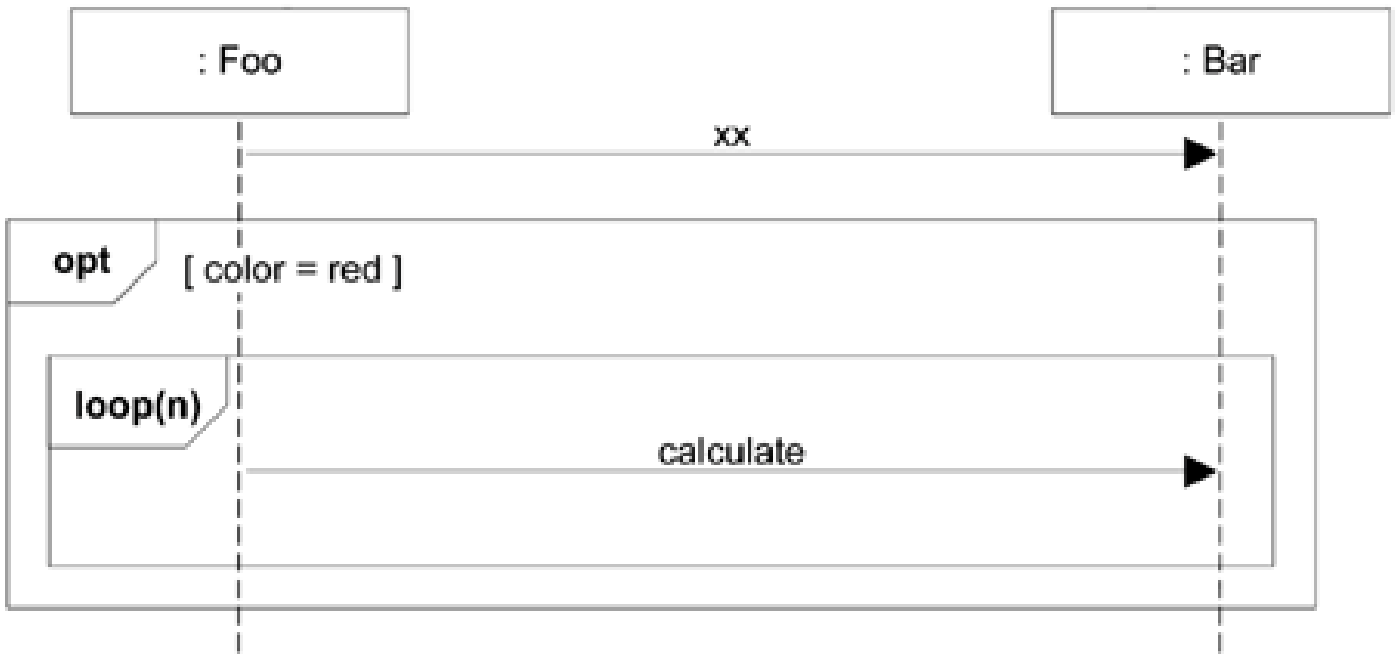


Diagram stanów - modelowanie stanu obiektu

Przedstawia stany w jakich może być obiekt (np. telefon) i zdarzenia (metody), które powodują zmianę stanu

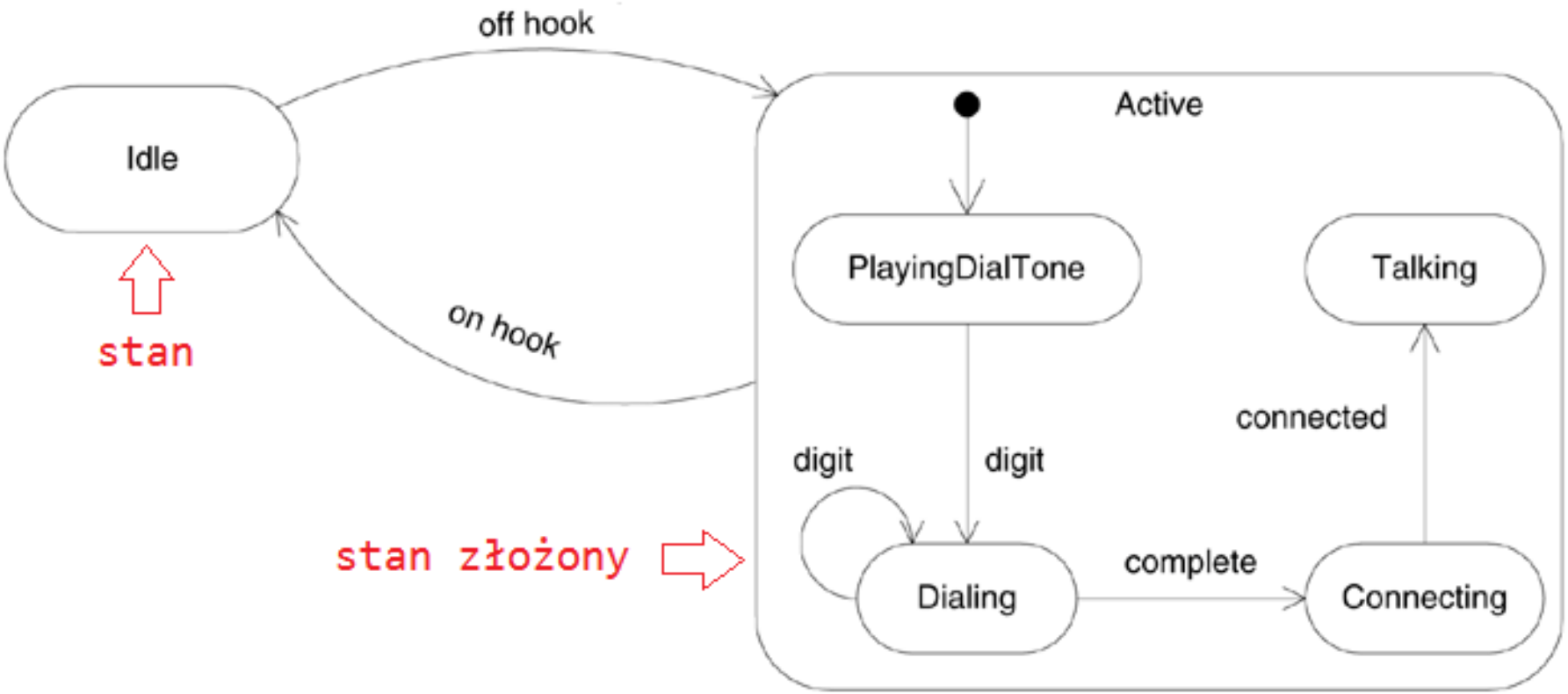


Diagram stanów – modelowanie logiki

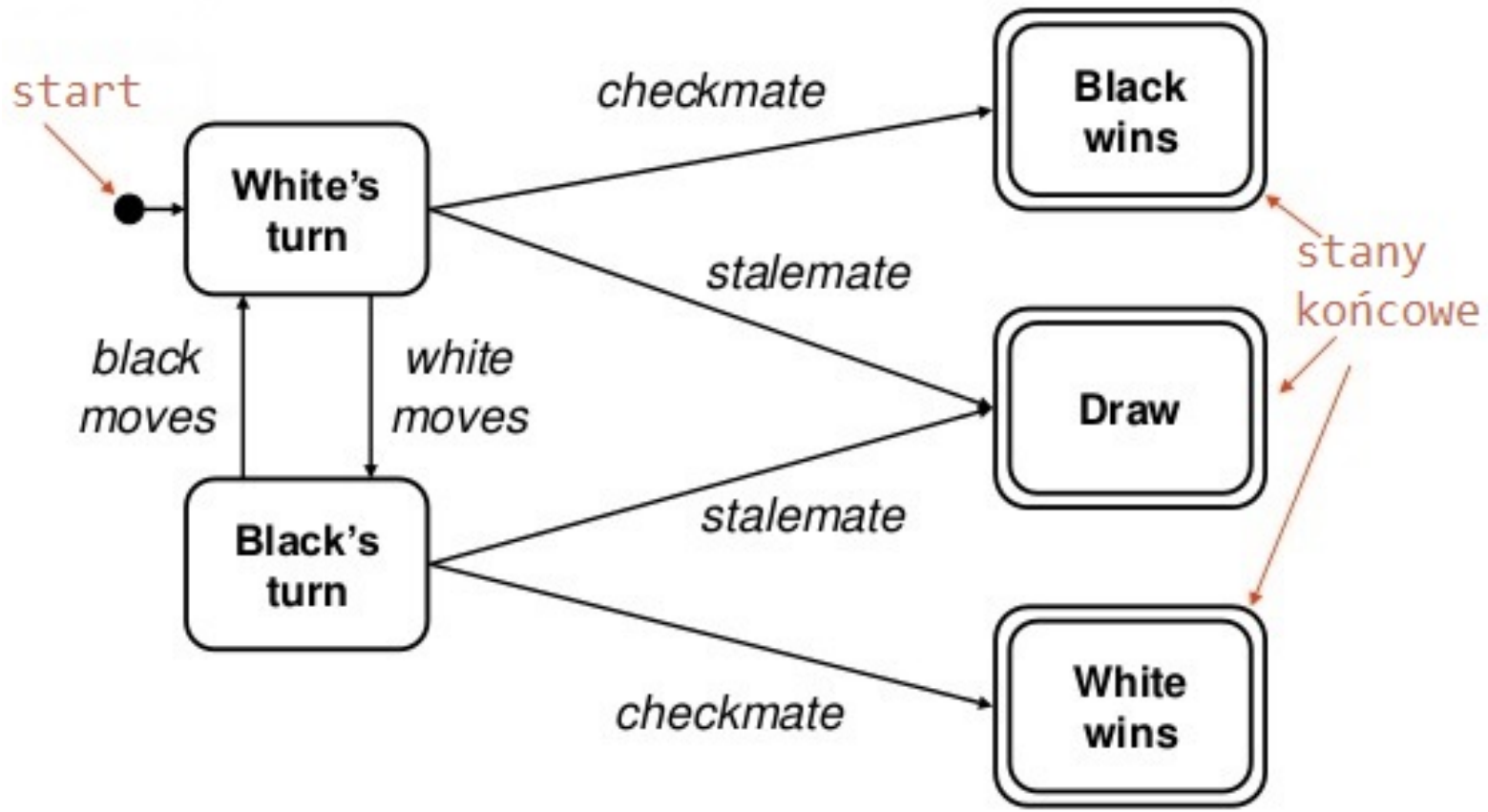


Diagram czynności

- Kiedy używać?
  - modelowanie przepływu danych i procesów
  - wizualizacja trudnych przypadków użycia (*np. wielu aktorów*)
- Elementy notacji:
  - akcja, grupa akcji, partycja, obiekt, sygnał
  - rozwidlenie, synchronizacja, decyzja, scalenie

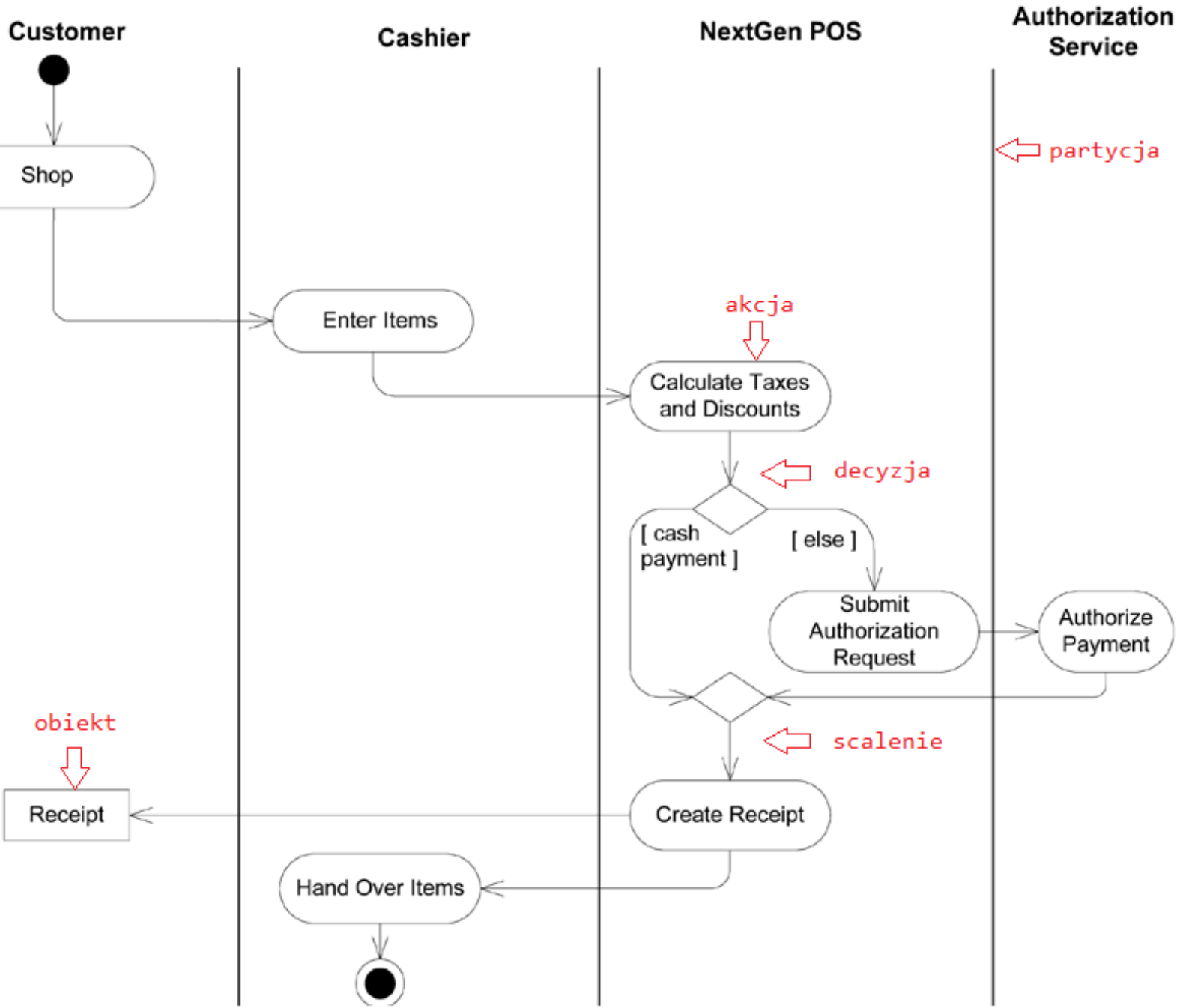
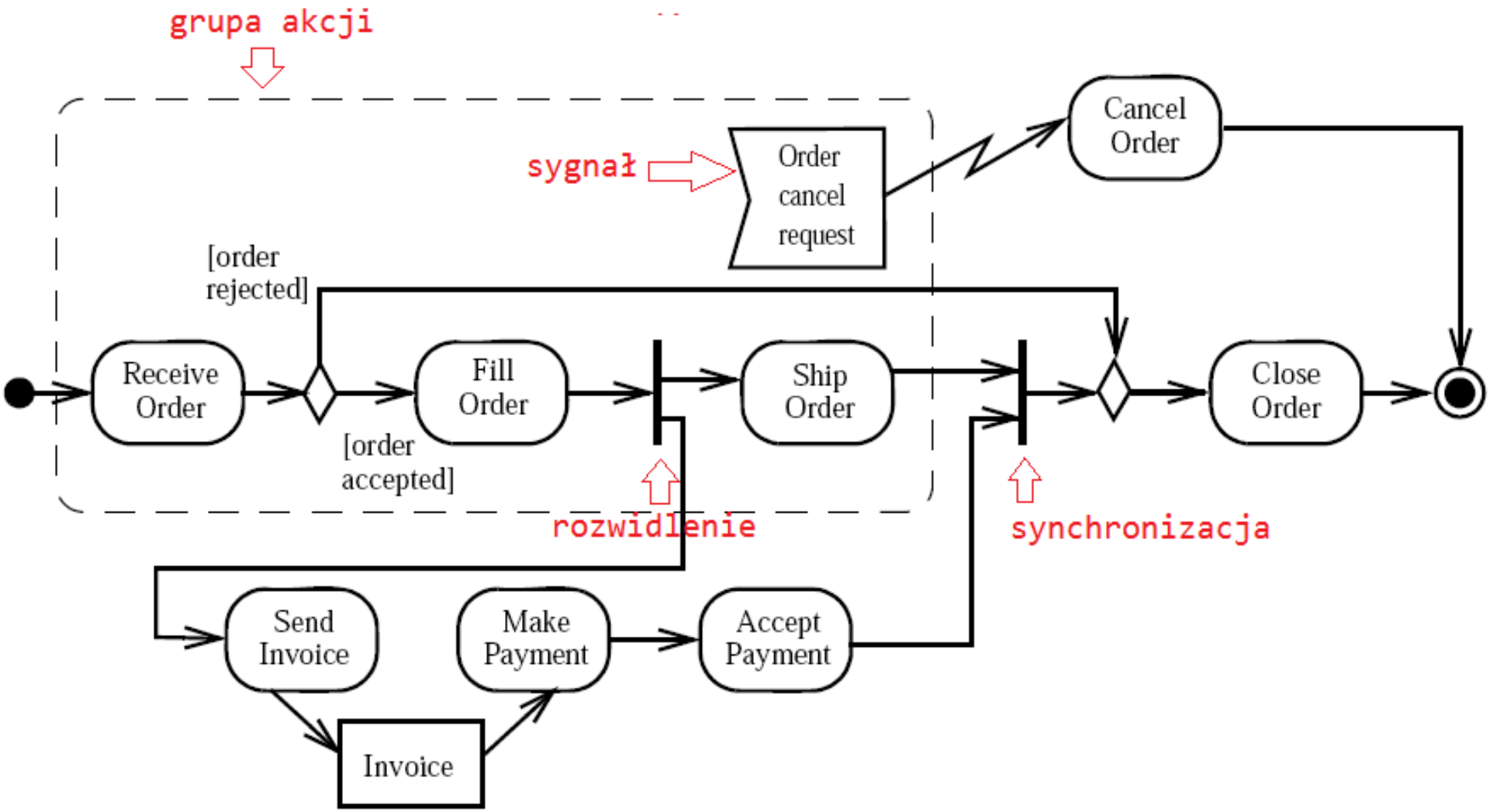
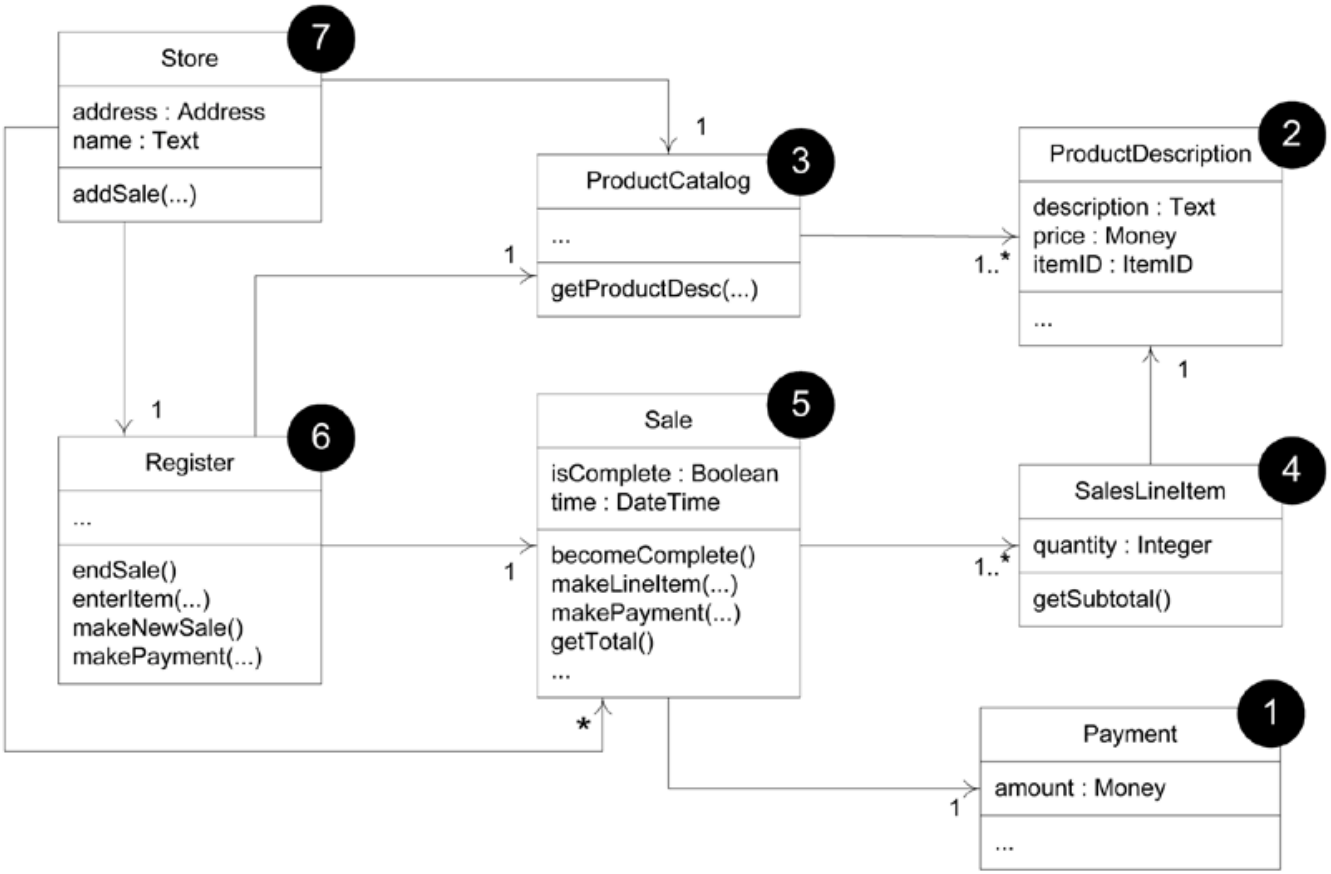

Customer
Cashier
NextGen POS
Authorization Service
partycja
Shop
Enter Items
akcja
Calculate Taxes and Discounts
decyzja
[cash payment]
[else]
Submit Authorization Request
Authorize Payment
obiekt
scalenie
Receipt
Create Receipt
Hand Over Items

Diagram czynności — przykład: zakupy online



Projekt systemu a jego implementacja

- m. dziedziny + kontrakty operacji + wzorce ⇒ diagram klas
- diagram klas + diagramy pomocnicze ⇒ klasy, ciała metod
- najpierw implementuj klasy najmniej powiązane (*przyrostowo*)



Narzędzia do UML

- Czy i jak używać UML?
  (*UML jako szkic, projekt, język programowania*)
- Czy używać narzędzi do UML?
- Forward engineering
  (*generacja kodu w oparciu o diagramy*)
- Reverse engineering
  (*generacja diagramów w oparciu o kod*)

Czy to co już wiemy wystarczy?

1. Projektując system musi podjąć wiele istotnych decyzji, a UML to tylko syntaktyka...
2. GRASP (*General Responsibility Assignment Software Patterns*)
   - 9 ogólnych wzorców przydziału odpowiedzialności do obiektów, ułatwiaja zrozumienie wzorców GoF
   - Cel: projektować w metodyczny i racjonalny sposób (∼ *nauka*)
3. 23 wzorce GoF (Gang of Four)
   - Gamma, Helm, Johnson, Vlissides: „*Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku*”
   - Schematy rozwiązań częstych problemów projektowych
   - Kodyfikacja wiedzy ekspertów, sprawdzonych podejść i strategii (∼ *wzorce architektoniczne w budownictwie*)

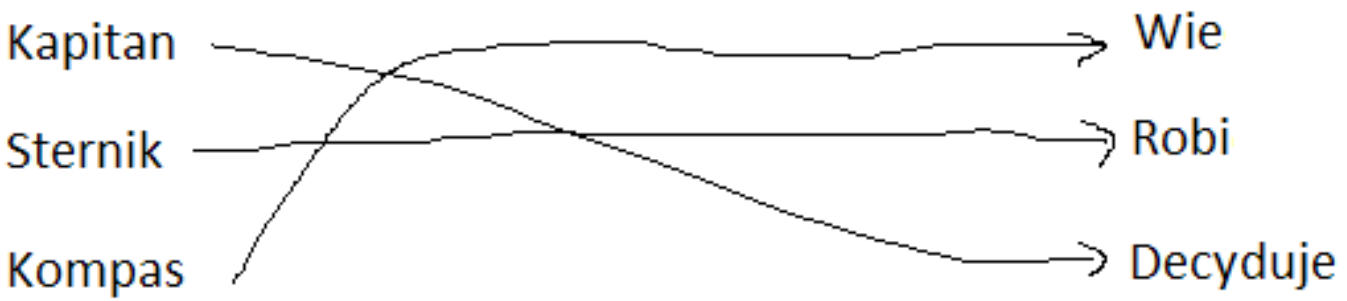
General Responsibility Assignment Software Patterns

- Czy to co robię to jest projekt obiektowy?
  (*polimorfizm, hermetyzacja, dziedziczenie, itp. to mało...*)
- Istotne jest rozłożenie odpowiedzialności – obiekt otrzymuje zobowiązania wobec innych obiektów i zapewnia ich wykonanie, ukrywając sposób ich realizacji
- GRASP to w praktyce zasady przydziału metod do klas, czyli rozkład odpowiedzialności w dążeniu do celu:

Kapitan	?	Wie
Sternik	?	Robi
Kompas	?	Decyduje

General Responsibility Assignment Software Patterns

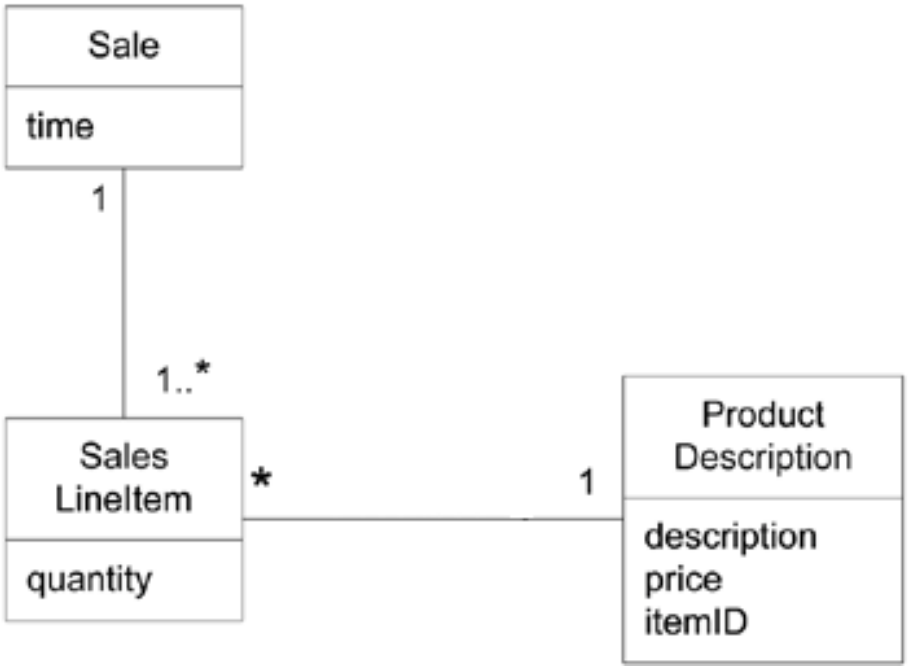
- Czy to co robię to jest projekt obiektowy?
  (*polimorfizm, hermetyzacja, dziedziczenie, itp. to mało...*)
- Istotne jest rozłożenie odpowiedzialności – obiekt otrzymuje zobowiązania wobec innych obiektów i zapewnia ich wykonanie, ukrywając sposób ich realizacji
- GRASP to w praktyce zasady przydziału metod do klas, czyli rozkład odpowiedzialności w dążeniu do celu:



Zasady GRASP

1. Expert
2. Creator
3. Low Coupling (*zasada ewaluacyjna*)
4. High Cohesion (*zasada ewaluacyjna*)
5. Polymorphism
6. Indirection
7. Pure Fabrication
8. Protected Variations
9. Controller

Expert - przykład



- `ProductDescription` - zna cenę produktu za sztukę
- `SalesLineItem` - zna liczbę sztuk danego produktu
- `Sale` - zna zakupione produkty

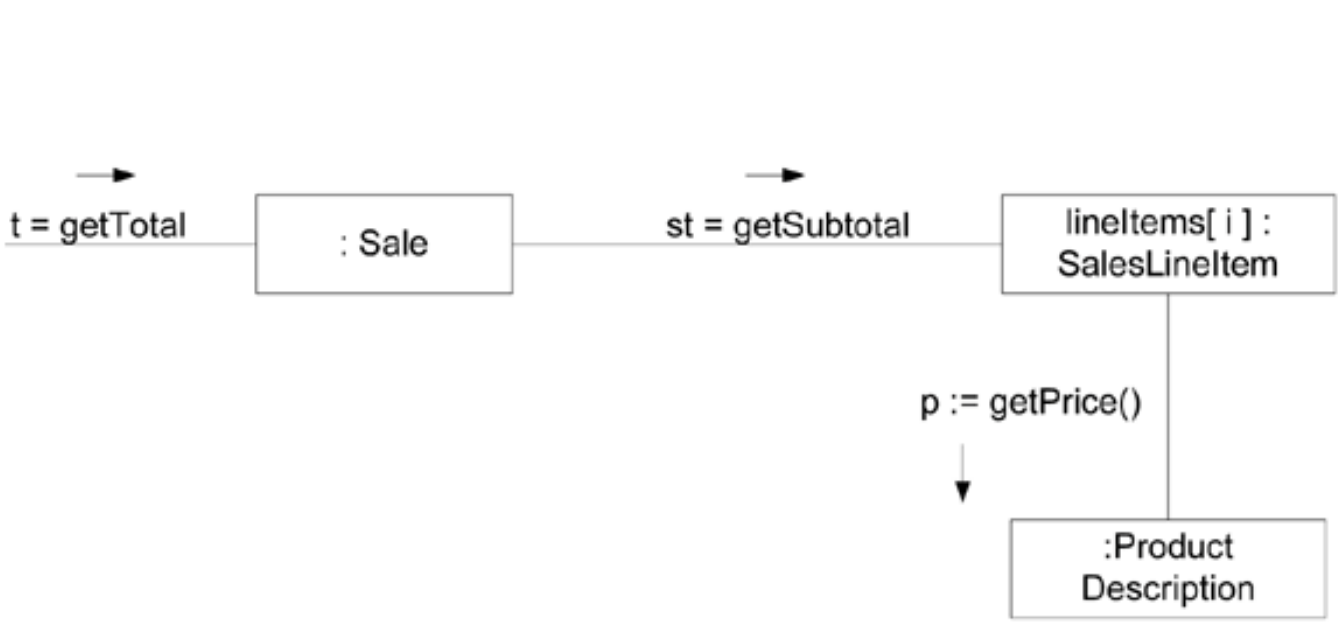
Która klasa powinnna wyliczać całkowitą kwotę sprzedaży?

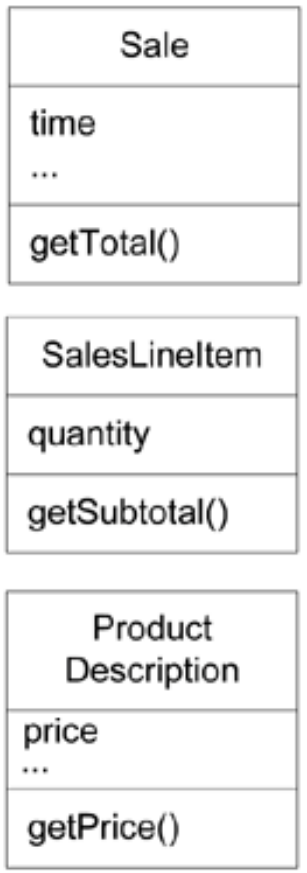
Expert - czyli ten, który wie, robi

1. Przydziel zobowiązanie „ekspertowi” - klasie, która ma informacje konieczne do jego realizacji
2. Gdy informacja jest rozproszona po różnych klasach konieczna może być współpraca ekspertów
3. Zysk: wspiera hermetyzację i równomierny podział odpowiedzialności, lżejsze i czytelniejsze klasy

Expert - przykład

Odpowiedź: 3 × Expert





Expert - zasada rozdzielenia poleceń i zapytań

Każda metoda powinna należeć do jednej z dwóch kategorii:

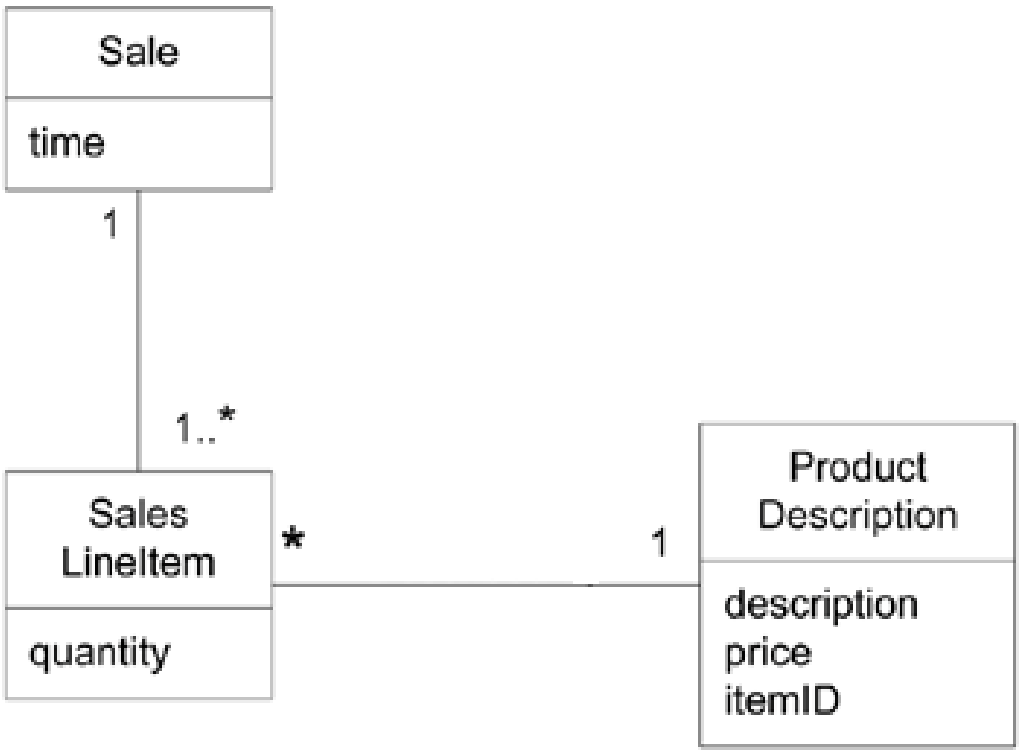
- metody-polecenia – wykonują działanie, nie zwracają wartości
- metody-zapytania – zwracają dane, nie zmieniają stanu obiektu

Taki podział 1) ułatwia zrozumienie działania programu,
2) pozwala na bezpiecznie wnioskowanie na temat jego stanu.

```
Bomb b = new Bomb();
String name = b.getName();
```

```
public class Bomb
{
  ...
  public String getName()
  {
    explode();
    return name;
```

Creator - łatwy przykład



Kto powinien być odpowiedzialny za utworzenie nowej instancji klasy SalesLineItem?

Creator

1. Nową instancje A powinna tworzyć klasa B, która:
   - ▷ zawiera, pamięta lub bezpośrednio używa A
   - ▷ posiada dane inicjalizacyjne dla A

   Gdy wiele klas spełnia warunki, wybierz B zawierające A
2. Zysk: mniejsza liczba powiązań, kod łatwy w utrzymaniu
3. Uwagi:
   - ▷ kolejność projektowania operacji typu „*create*”? (*createSystem na samym końcu*)
   - ▷ czasem lepiej oddelegować tworzenie instancji do specjalnej klasy (*GoF: wzorce fabryk*)

Creator - łatwy przykład