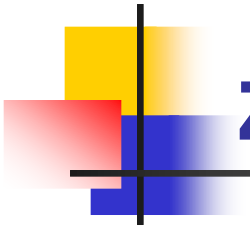
Zagadnienia (1/3)

- Rola modelu systemu w procesie analizy wymagań (inżynierii wymagań)
- Prezentacja różnego rodzaju informacji o systemie w zależności od rodzaju modelu. Budowanie pełnego obrazu systemu poprzez wzajemne uzupełnianie informacji obrazowanych przez poszczególne modele systemu
- Omówienie notacji oraz narzędzi typu CASE dla przykładowych modeli:
  - Data-flow – diagramy przepływów danych
  - ERD – diagramy związków encji
  - Diagramy obiektowe w UML (ang. Unified Modeling Language)

Zagadnienia (2/3)

- Omówienie poszczególnych typów diagramów wchodzących w skład specyfikacji UML:
  - Usecase diagram
  - Class diagram
  - Interaction diagram:
    - Sequence diagram
    - Collaboration diagram
  - State diagram
  - Activity diagram
  - Package diagram



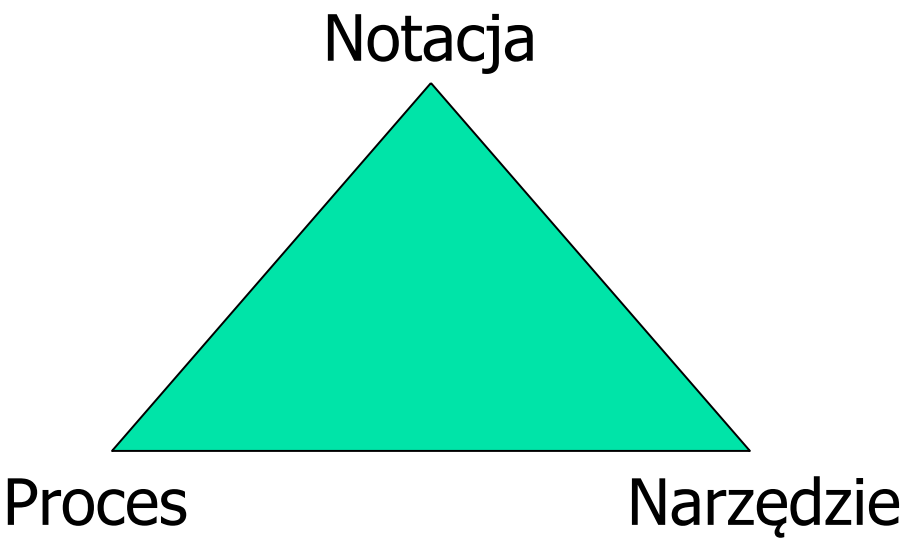
Zagadnienia (3/3)

- Deployment diagram

Modelowanie systemu

- Modelowanie systemu z wykorzystaniem różnorodnych modeli ma na celu zwiększenie rozumienia funkcjonalności jaką powinien realizować system
- Modele spełniają dwojakiego rodzaju funkcje:
  - służą do komunikacji z klientem/użytkownikiem
  - stanowią podstawę fazy analizy i implementacji wymagań
- Modele zawsze stanowią pewną abstrakcję systemu – różne modele w różnym stopniu opisują różne aspekty systemu
- Dla każdego projektowanego systemu w fazie analizy wymagań powinno powstać kilka różnych modeli, przy czym każdy z nich powinien koncentrować się na innych aspektach systemu (lub prezentować te same aspekty z różnych punktów widzenia)

Modelowanie – „Trójkąt sukcesu”



Typy modeli systemu

- Istnieje wiele różnorodnych modeli wykorzystujących odmienne notacje i skupiających się na różnych aspektach systemu:
  - przetwarzanie danych – obrazowanie procesów przetwarzania danych przez różne moduły wchodzące w skład systemu
  - kompozycja – obrazowanie w jaki sposób nadrzędne moduły systemu tworzone są z innych, podrzędnych modułów; wzajemne związki i relacje pomiędzy poszczególnymi modułami
  - zdarzenia-odpowiedzi – obrazowanie reakcji systemu na różnego rodzaju zdarzenia zarówno wewnętrzne jak i zewnętrzne
  - procesy – obrazowanie procesów i aktywności zachodzących w systemie

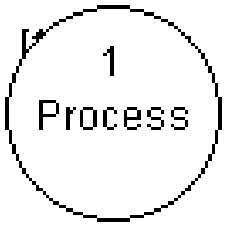
Data flow - diagramy przepływów danych

- Służą do prezentowania sposobu w jaki dane przepływają oraz są przetwarzane w systemie. Opisują również procesy przetwarzające dane
- Stanowią podstawę wielu istniejących modeli
- Prosta, intuicyjna i łatwa w zrozumieniu notacja

Data flow - podstawowe elementy

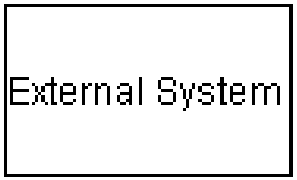
- Zbiorniki danych (ang. data stores) - składowa systemu przechowująca dane

- Procesy (ang. process) - dowolne operacje wykonywane przez system. W wyniku wykonania operacji może następować modyfikacja danych

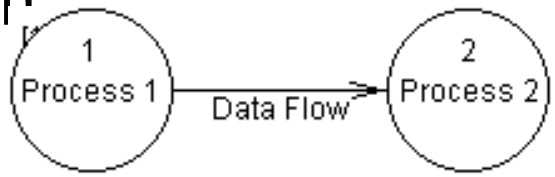


Data flow - podstawowe elementy

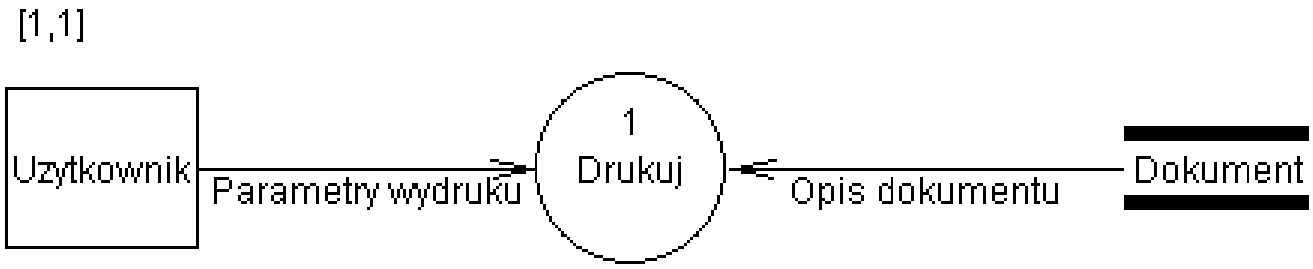
- Systemy zewnętrzne (ang. external systems) - zewnętrzne, w stosunku do modelowanego, system z którym wymieniane są dane.



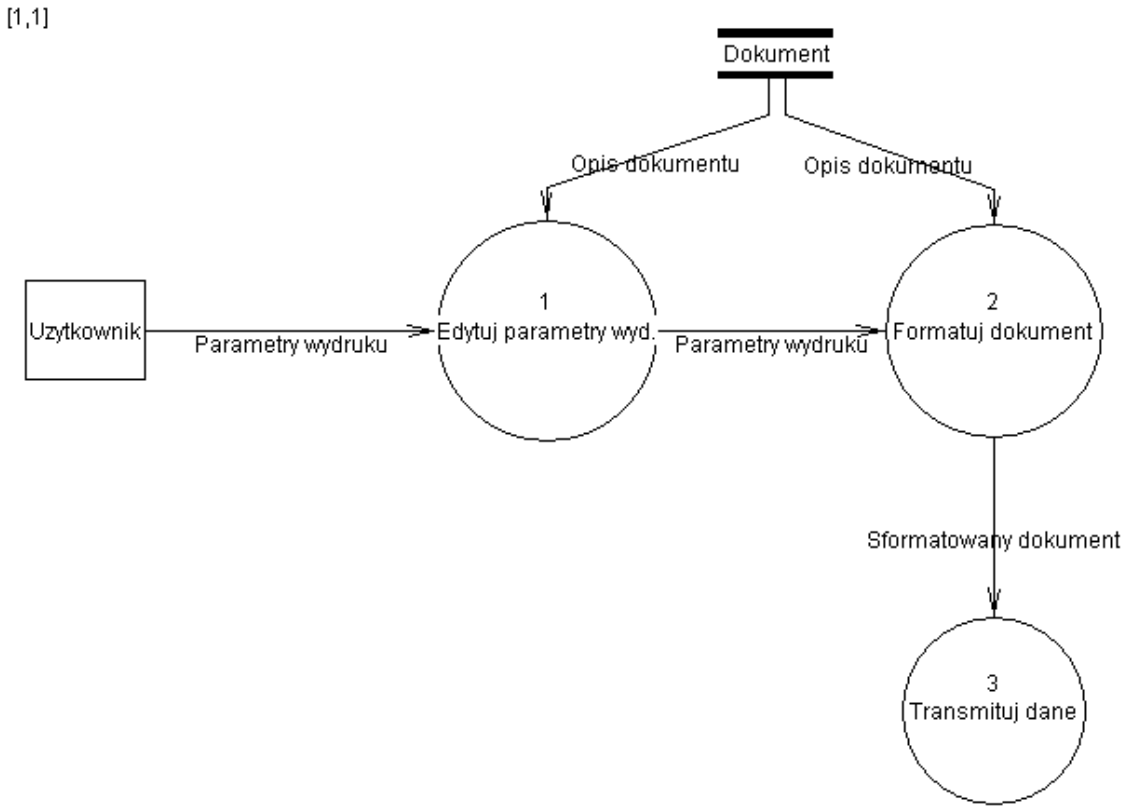
- Przepływ danych (ang. data flow) - opisuje dane przesyłane między procesami, systemami zewnętrznymi i zbiornikami danych



Data flow - przykład (1/2)

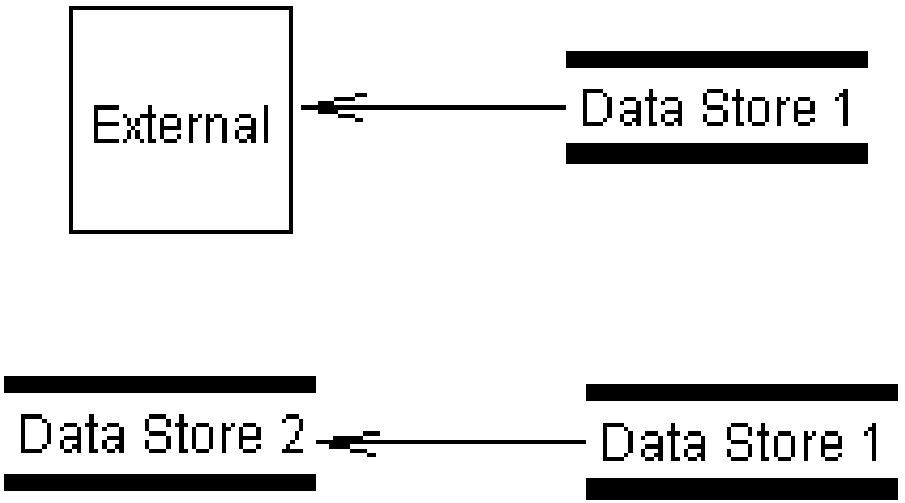


Data flow - przykład (2/2)



Data flow - niedopuszczalne przepływy danych

[1,1]



Data flow - zastosowanie

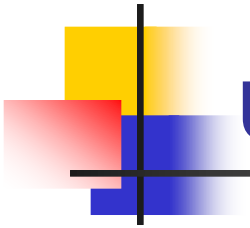
- Modelowanie przetwarzania danych na różnych poziomach począwszy od bardzo zgrubnych do bardzo szczegółowych modeli
- Opis architektury systemu obrazujący przepływ danych pamiędzy poszczególnymi elementami systemu
- Nie powinno się stosować diagramów tego rodzaju do opisu i definicji interfejsów

Modelowanie obiektowe

- Modele obiektowe opisują systemy poprzez wykorzystanie **klas obiektów** obrazujących poszczególne elementy/komponenty/idee tworzące system
- **Klasa obiektu** (ang. class) jest pewną abstrakcją (opisem) określajacą wspólne atrybuty i operacje jakie mogą zostać wykonane na konkretnych instancjach (obiektach) tej klasy. Klasę można traktować jako „wzorzec” tworzenia obiektów
- **Obiekt** (ang. object) stanowi abstrakcyjną reprezentację elementów z dziedziny danego systemu posiadającą tożsamość (ang. identity), stan (ang. state) i zachowanie (ang. behaviour). Obiekt może reprezentować konkretne rzeczy np. samochód lub komputer lub pewne idee czy abstrakcyjne pojęcia jak np. zainteresowania, transakcje bankowe itp. Każdy obiekt jest instancją pewnej klasy. Jeden obiekt nie może być instancją więcej niż **jednej** klasy

UML – modelowanie obiektowe (1/2)

- Unifikacja najlepszych technik/notacji istniejących w zakresie modelowania systemów z wykorzystaniem metodologii obiektowych. UML wprowadził standardową notację i zbiór diagramów
- Pierwsza wersja została ogłoszona w 1995 r. (wersja 0.8). Wersja 1.1 została ogłoszona i zaprezentowana dla OMG (Object Management Group) we wrześniu 1997 r.
- Specyfikację UML tworzą diagramy:
  - Usecase diagram
  - Class diagram
  - Sequence diagram
  - Collaboration diagram



UML – modelowanie obiektowe (2/2)

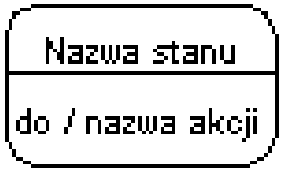
- State diagram
- Activity diagram
- Package diagram
- Deployment diagram

State diagram – diagram stanów

- Diagramy tego rodzaju wykorzystywane są do opisu zachowania obiektu
- Obrazują możliwe stany obiektu, zdarzenia jakie mogą być odbierane przez obiekt i akcje jakie mogą zostać wykonane w odpowiedzi na zdarzenia
- W większości obiektowych metodyk diagramy stanów wykorzystywane są do obrazowania zachowania pojedyńczych obiektów

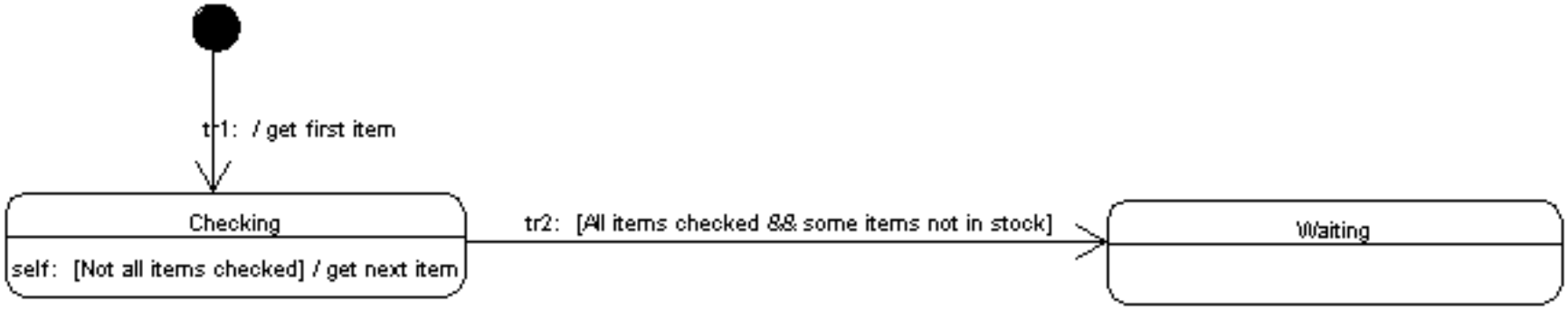
State diagram – terminologia/notacja

- **Stan** (ang. state) **-** w terminologii diagramów pojęcie stanu należy rozumieć jako stan obiektu, w którym wykonuje on pewną akcję lub oczekuje na zdarzenie, określony przez wartości poszczególnych atrybutów klasy tego obiektu



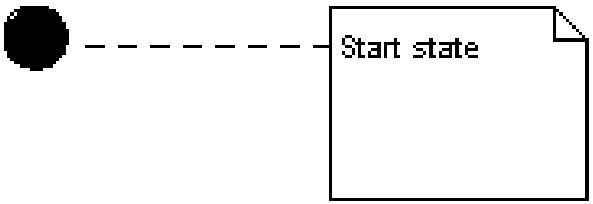
State diagram – terminologia/notacja

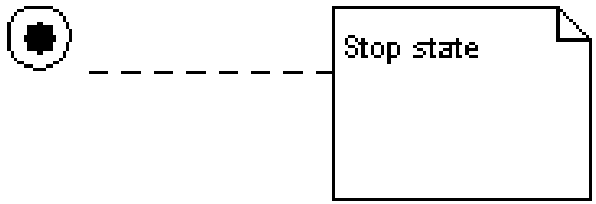
- **Zmiana stanu** (ang. state transition) – przejście z obecnego stanu do stanu następnego w wyniku zdarzenia lub zakończenia akcji wykonywanej przez obiekt w obecnym stanie.



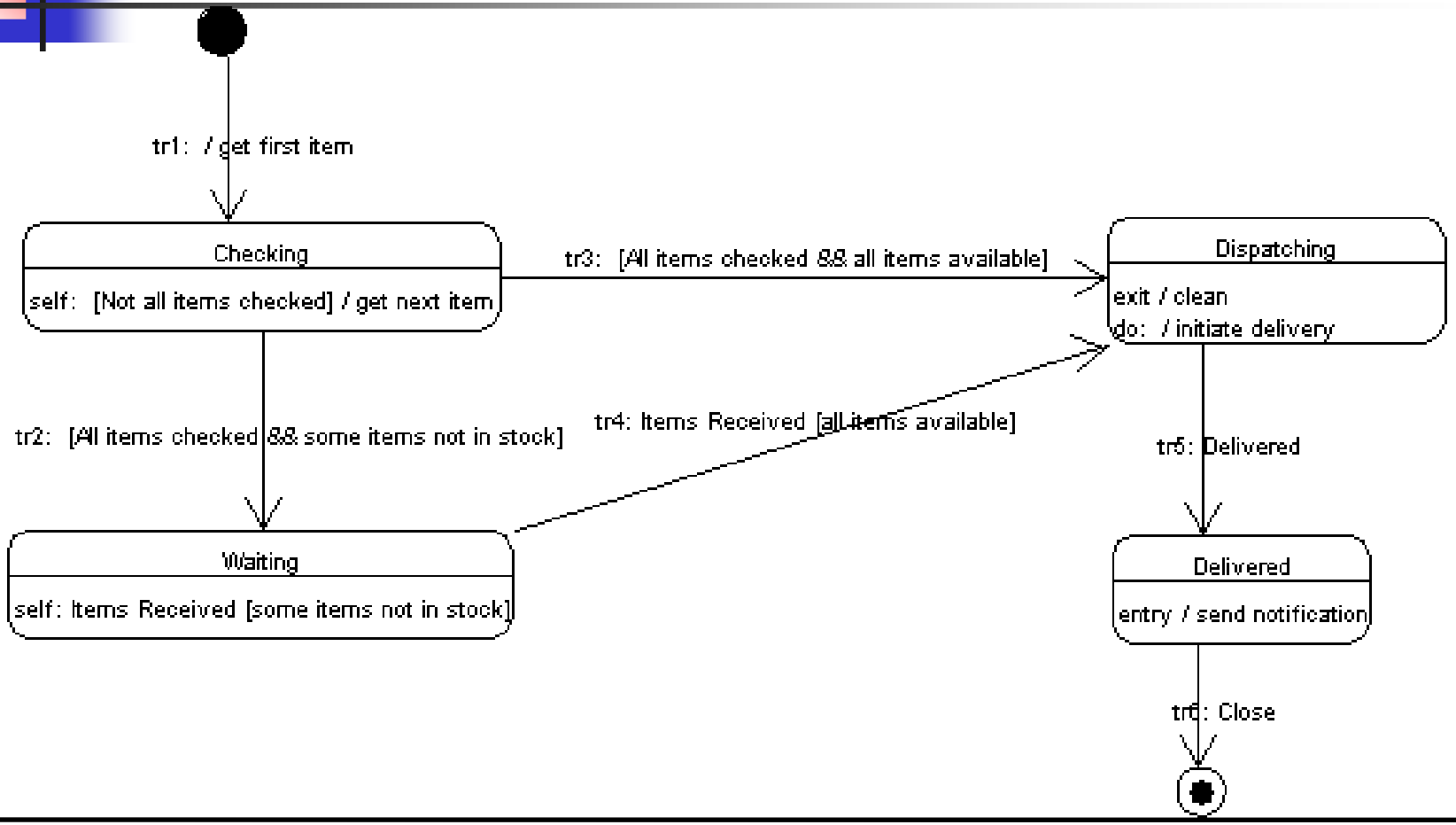
State diagram – terminologia/notacja

- **Stan początkowy i końcowy** – dwa specjalne stany określające odpowiednio początek diagramu stanów dla danego obiektu oraz jego koniec. Diagram może mieć jeden i tylko jeden stan początkowy i wiele stanów końcowych. Stan początkowy oznacza stan obiektu zaraz po jego utworzeniu, natomiast stan końcowy oznacza stan obiektu tuż przed jego usunięciem z systemu.

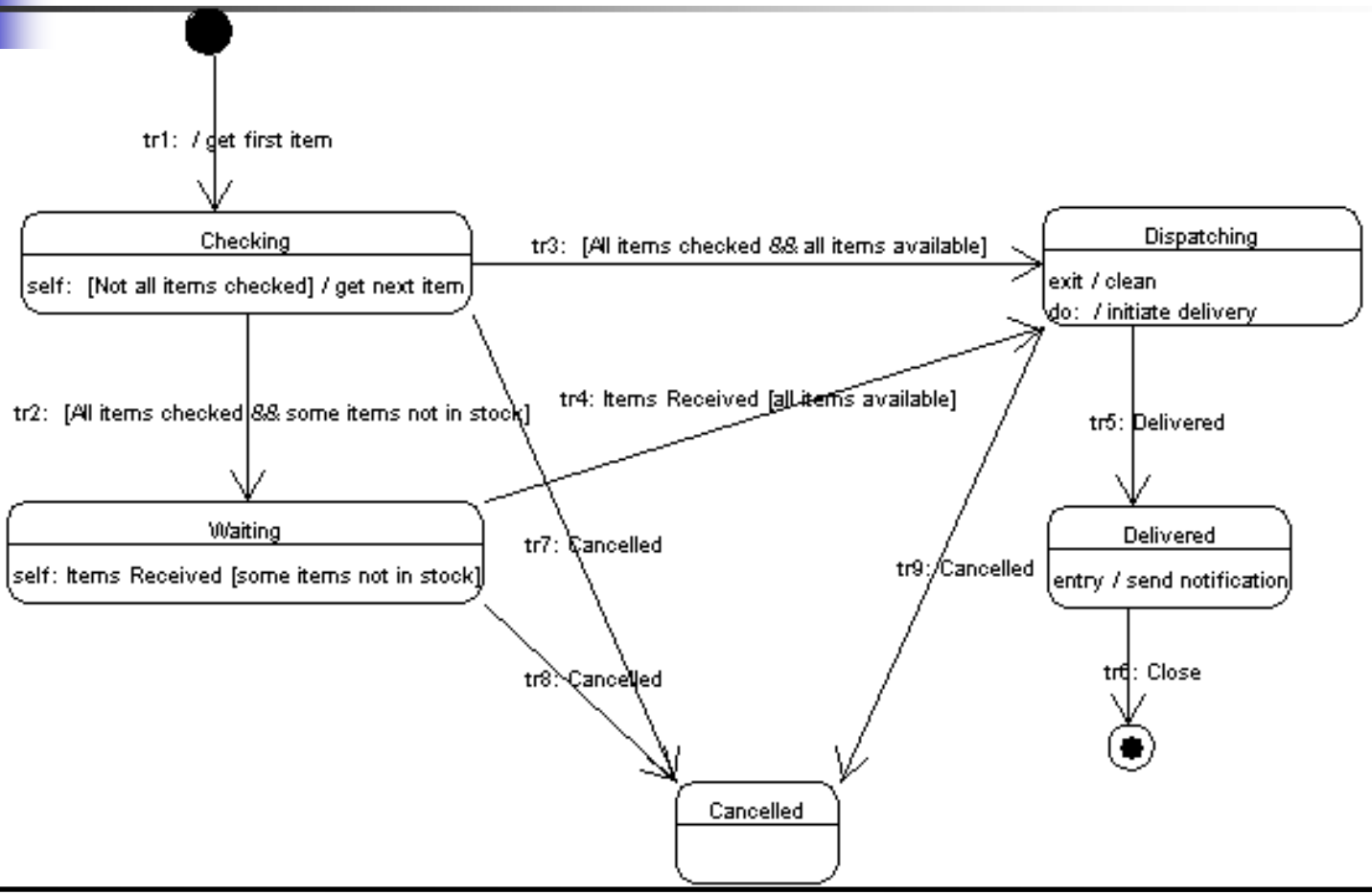




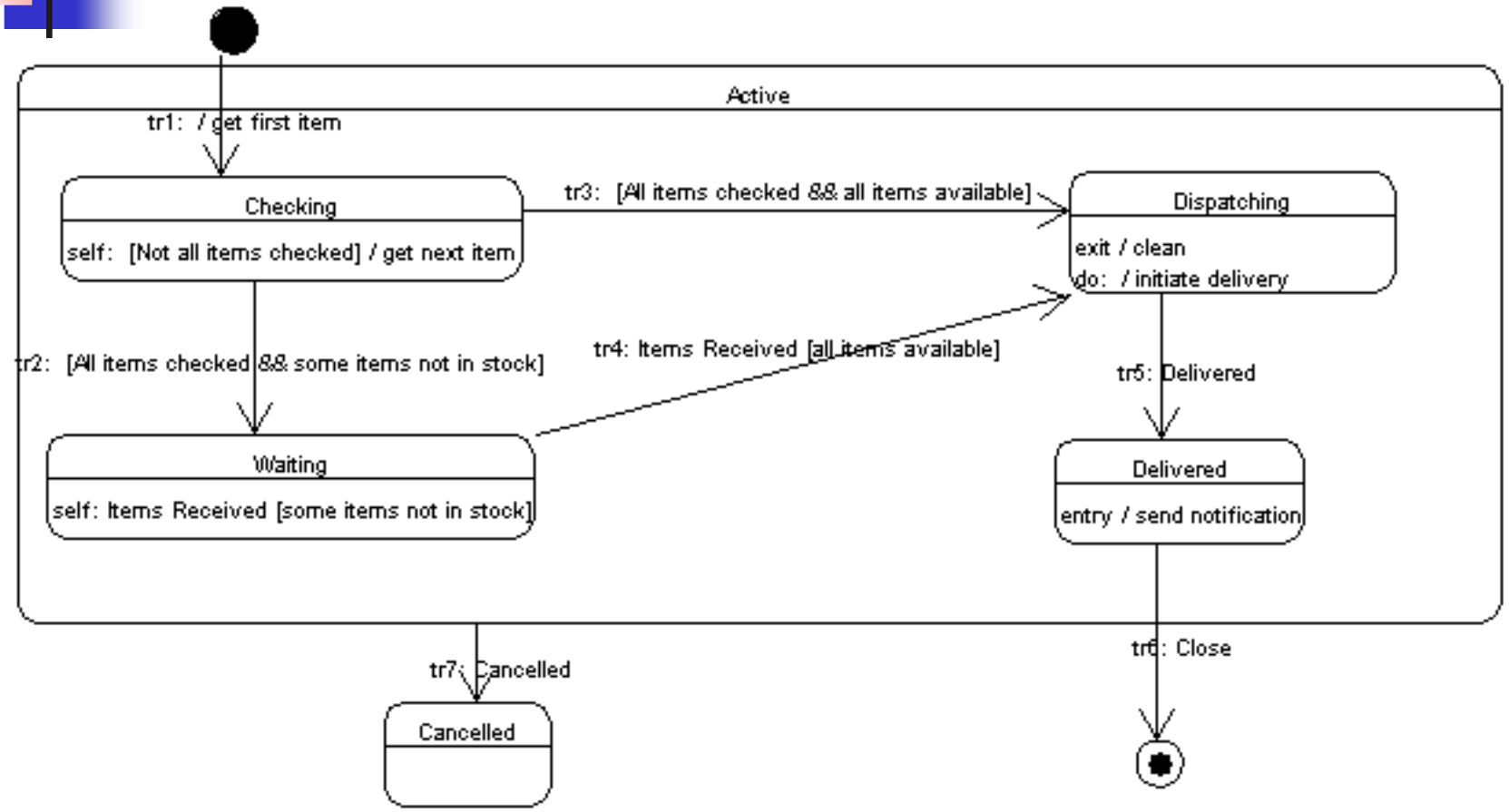
State diagram – przykład (1/3)



State diagram – przykład (2/3)



State diagram – przykład (3/3)



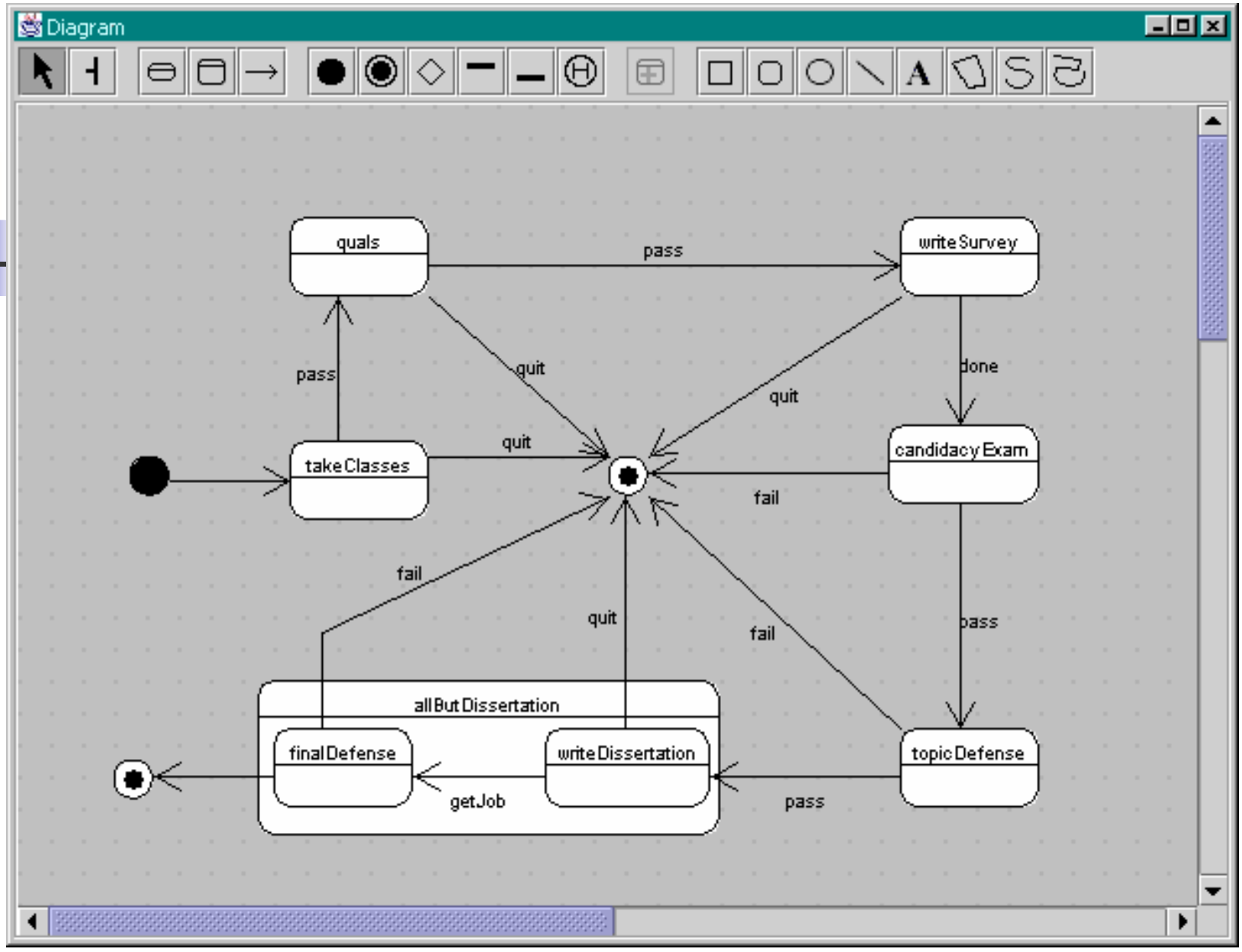
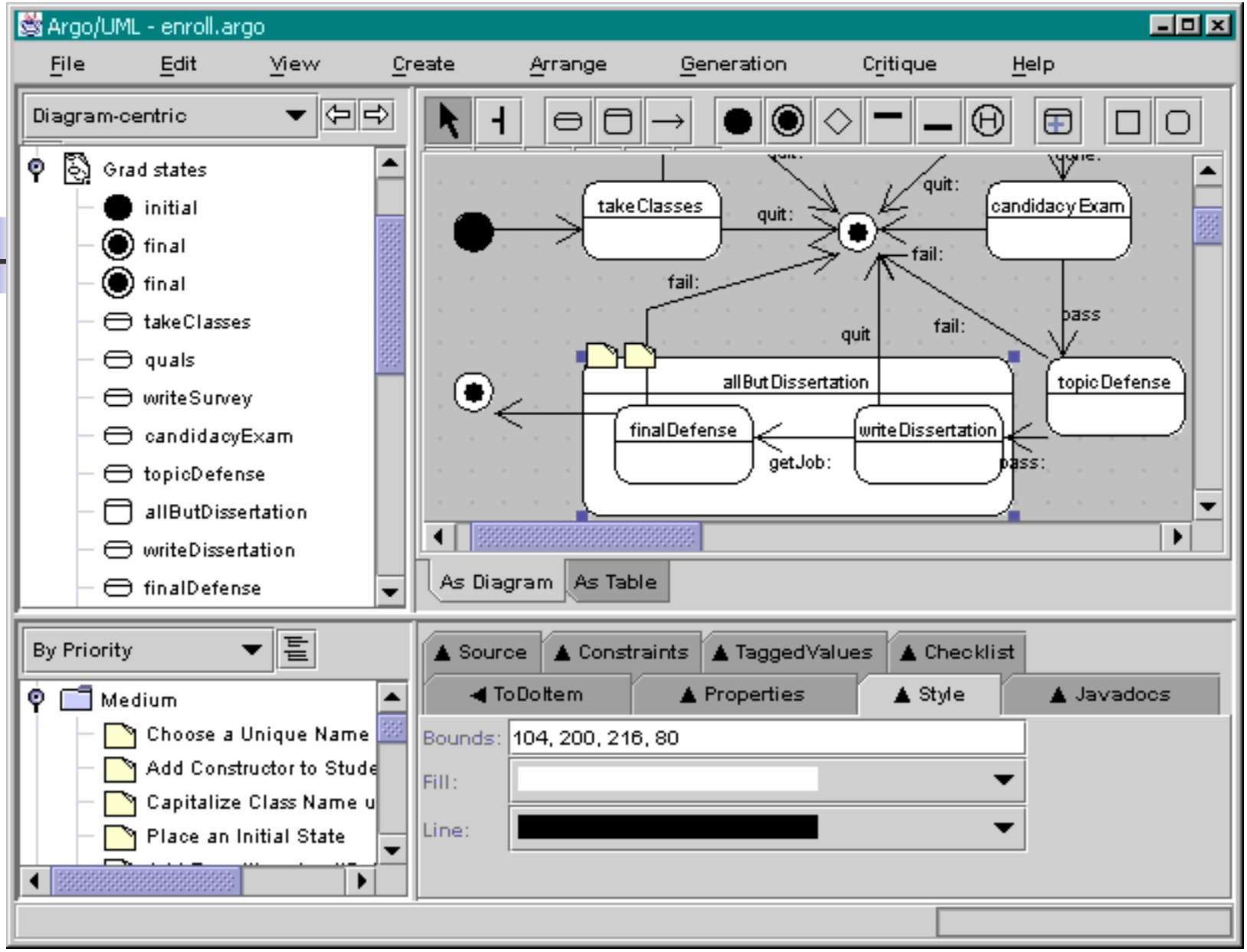


Diagram
quals
pass
write Survey
pass
quit
done
quit
take Classes
quit
candidacy Exam
fail
fail
quit
fail
pass
all But Dissertation
final Defense
write Dissertation
topic Defense
getJob
pass


Argo/UML - enroll.argo
File
Edit
View
Create
Arrange
Generation
Critique
Help
Diagram-centric
Grad states
initial
final
final
takeClasses
quals
writeSurvey
candidacyExam
topicDefense
allButDissertation
writeDissertation
finalDefense
takeClasses
quit:
candidacy Exam
fail:
fail:
quit
fail:
pass
allBut Dissertation
topic Defense
finalDefense
writeDissertation
getJob:
pass:
As Diagram
As Table
By Priority
Medium
Choose a Unique Name
Add Constructor to Stude
Capitalize Class Name u
Place an Initial State
Source
Constraints
TaggedValues
Checklist
ToDoItem
Properties
Style
Javadocs
Bounds:
104, 200, 216, 80
Fill:
Line:

State diagram – zastosowanie

- Opis zachowania pojedyńczych obiektów
- **Nie** powinny być stosowane do opisu zachowania **grupy wzajemnie** współpracujących obiektów (w takich przypadkach powininy być stosowane diagramy innego rodzaju np. activity)
- **Nie** powinny być stosowane dla **każdej klasy** istniejącej w modelu a tylko dla tych, które wykazują złożone zachowanie. W takich przypadkach diagramy stanów pomagają w zrozumieniu charakterystyki i zachowania obiektów
- Wykorzystywne są często do opisu interfejsu użytkownika

Activity diagram – diagram aktywności

- Diagramy tego rodzaju wykorzystywane są do opisu zachowania **grup obiektów**, wzajemnego powiązania pomiędzy operacjami wykonywanymi w określonej kolejności poprzez obiekty (bez wyraźnej identyfikacji i przypisywania akcji do poszczególnych instancji)

Activity diagram – terminologia/notacja

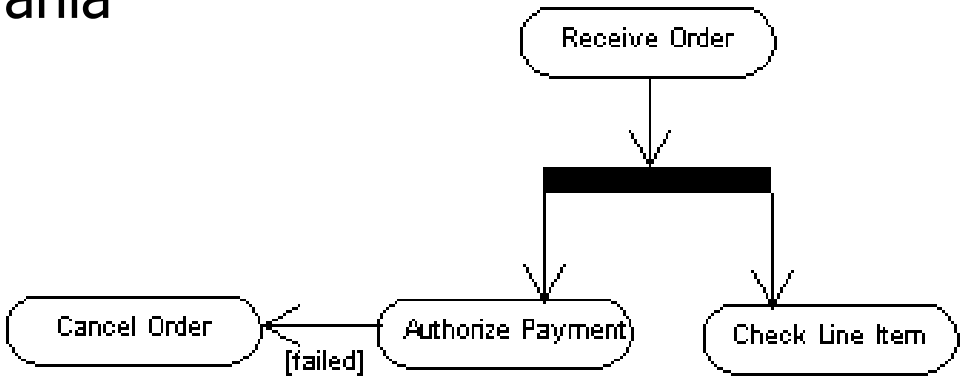
- **Aktywność** (ang. activity) **-** w terminologii diagramów aktywności należy rozumieć jako dowolne zadanie, operację, metodę jaką wykonuje obiekt. Każda aktywność może być połączona z inną aktywnością tworząc w ten sposób sekwencje zadań wykonywanych przez obiekt lub grupę obiektów

Activity diagram – terminologia/notacja

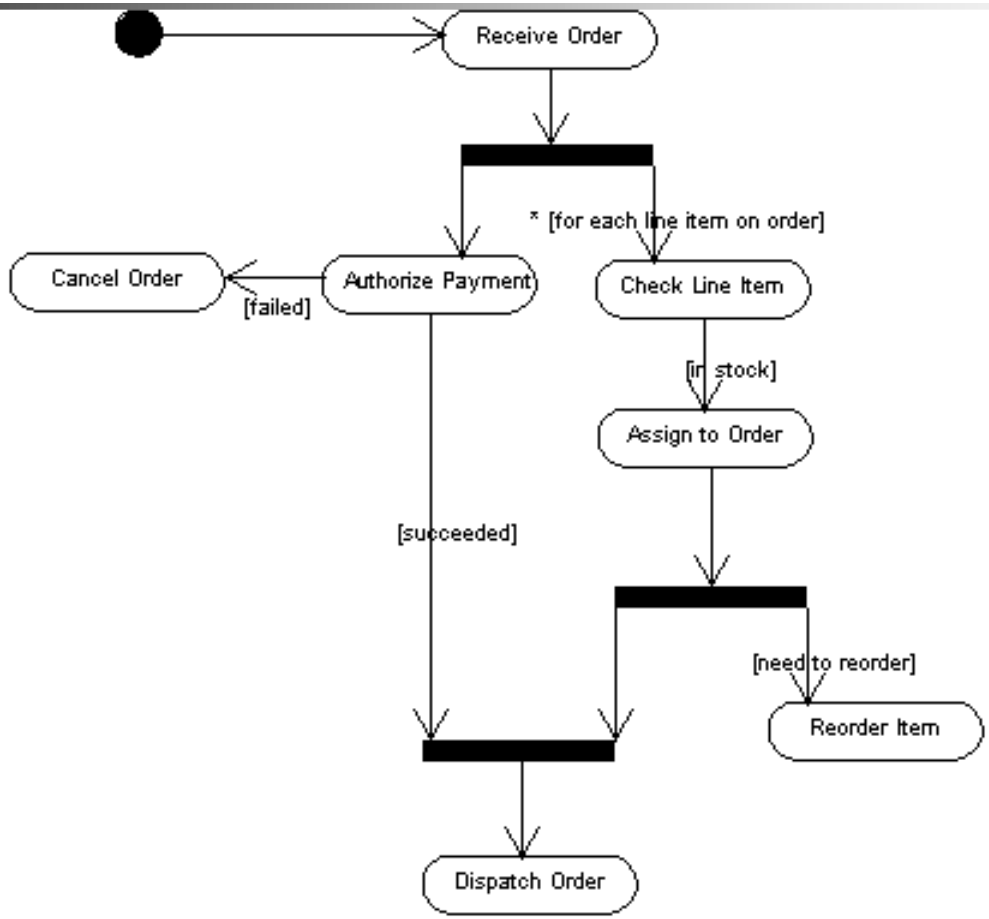
- **Przejścia pomiędzy aktywnościami** (ang. trigger) – przejście pomiędzy aktywnościami następuje w momencie zakończenia wykonywania operacji związanej z obecną aktywnością i spełnieniu warunków określonych dla danego przejścia. Przejście następuje tylko w przypadku gdy **wszystkie** warunki określone dla przejścia zostały spełnione

Activity diagram – terminologia/notacja

- **Punkt synchronizacji** (ang. synchronization bar) – umożliwia obrazowanie aktywności/operacji, które mogą być wykonywne równolegle przez różne obiekty. Punkty synchronizacji umożliwiają wskazanie miejsc, w których następuje zrównoleglenie wykonywania aktywności oraz miejsc, w których następuje oczekiwanie na zakończenie ich wykonywania



Activity diagram – przykład



Activity diagram – zastosowanie

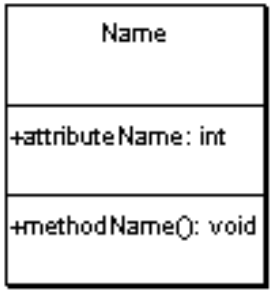
- Opisy zachowania grupy obiektów
- Modelowanie równoległego wykonywania aktywności/operacji (programowanie wielowątkowe)
- Szczegółowa analiza usecasów – zrozumienie akcji jakie należy wykonać w celu realizacji usecasu
- Modelowanie zachowania w przypadku gdy realizacja pewnej funkcjonalności obejmuje więcej niż jeden usecase
- Nie powinno się stosować aby zobrazować zachowanie pojedyńczych obiektów

Class diagram – diagram klas

- Diagram klas opisuje typy obiektów tworzących system i różnorodnych **statycznych** relacji/zależności występujących pomiędzy nimi
- Wyróżniamy dwa podstawowe rodzaje relacji statycznych:
  - powiązania/związki (ang. associations/relationships)
  - dziedziczenie/związek generalizacji-specjalizacji (ang. generalization)

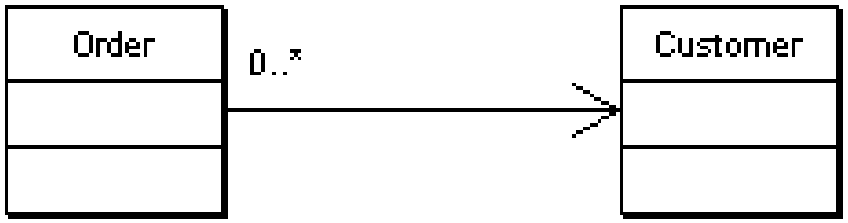
Class diagram – terminologia/notacja

- **Klasa** (ang. class) **–** na etapie analizy oznacza wzorzec grupy obiektów charakteryzujących się tymi samymi właściwościami. Na etapie projektowania i implementacji pojęcie to oznacza typ obiektowy w języku programowania. Klasa składa się z pól (atrybutów ang. attributes) i metod (ang. methods). Atrybuty opisują stan obiektu natomiast metody określają operacje jakie można wykonywać na obiektach tej klasy.



Class diagram – terminologia/notacja

- **Powiązania** (ang. associations) – reprezentują wzajemne relacje pomiędzy instancjami klas (np. pracownik pracuje w firmie, firma posiada wiele oddziałów itp.)



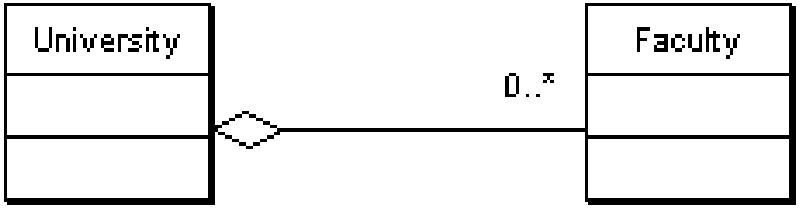
Class diagram – terminologia/notacja

- **Krotność powiązania** (ang. association multiplicity) – symbole na końcach linii odpowiadającej relacji opisujące liczby obiektów danej klasy, które mogą być powiązane w danej relacji z obiektami drugiej klasy

Krotność	Oznaczenie
Dokładnie jeden	1
Zero lub wiele	0..*
Zero lub jeden	0..1
Podana liczba	3
Zakres	2-8

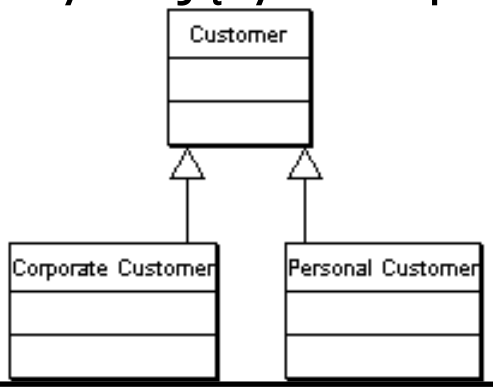
Class diagram – terminologia/notacja

- **Powiązania agregacji** (ang. aggregation) – oznacza związek, w którym obiekty pewnej klasy zawierają/składają się z obiektów innej klasy. Związek tego rodzaju nazywany jest również powiązaniem typu całość-część (ang. whole-part relationship)



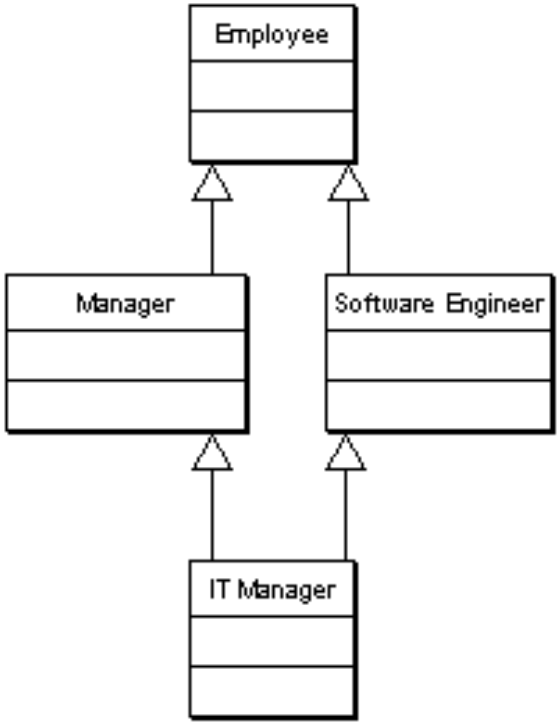
Class diagram – terminologia/notacja

- **Związek generalizacji-specjalizacji** (ang. generalization-specialization) – dwie klasy pozostają w związku generalizacji-specjalizacji, jeżeli jedna z nich, zwana specjalizacją, jest rodzajem drugiej, nazwanej generalizacją. Generalizacja jest więc uogólnieniem specjalizacji. Generalizacja może mieć wiele specjalizacji ale również specjalizacja może mieć wiele generalizacji. Dziedziczenie stanowi implementację związku generalizacji-specjalizacji w obiektowych językach programowania (np. C++, Java)

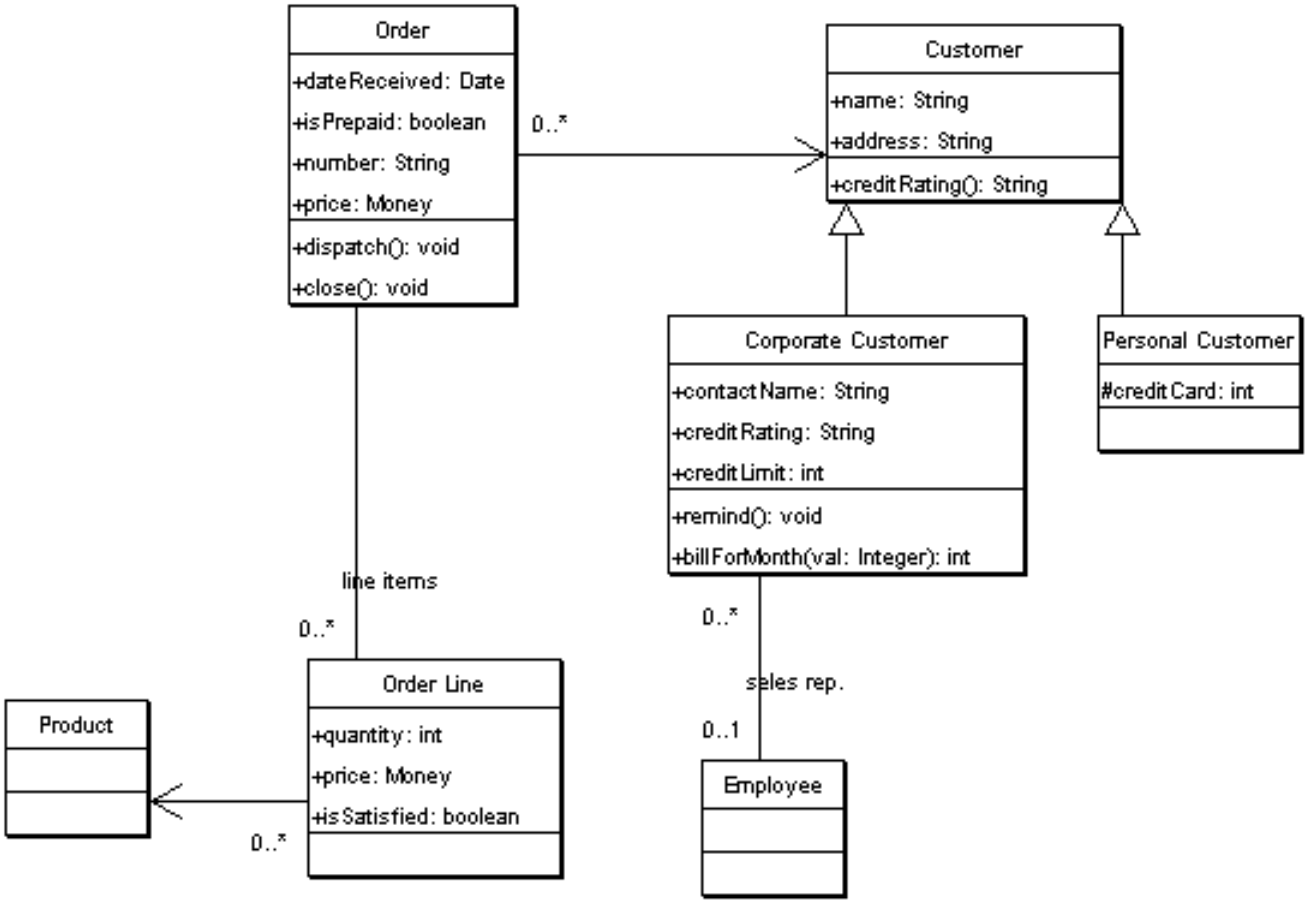


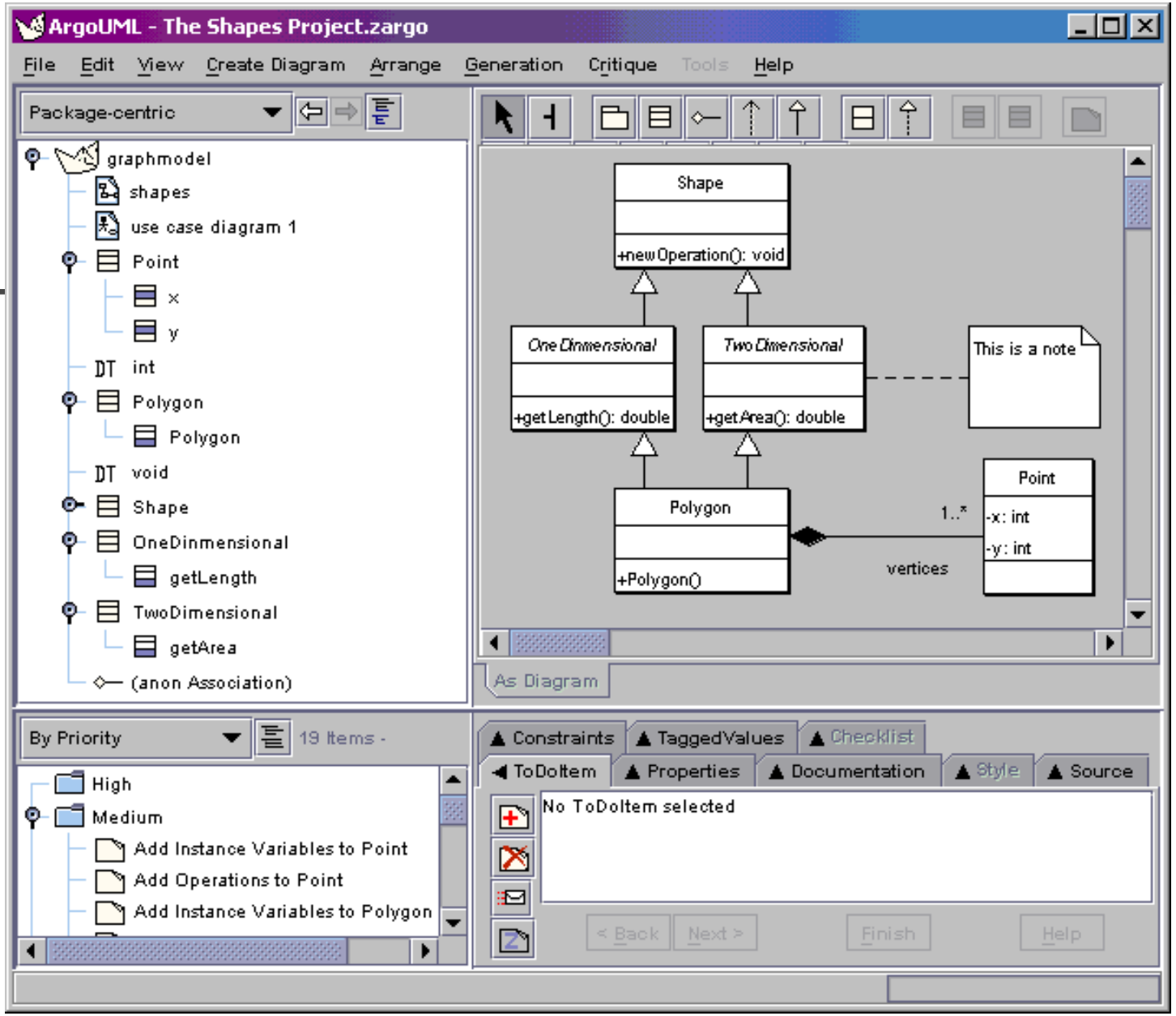
Class diagram – przykład

- Przykład wielu generalizacji i specjalizacji klasy



Class diagram – przykład




ArgoUML - The Shapes Project.zargo
File Edit View Create Diagram Arrange Generation Critique Tools Help
Package-centric
graphmodel
shapes
use case diagram 1
Point
x
y
int
Polygon
Polygon
void
Shape
OneDinmensional
getLength
TwoDimensional
getArea
(anon Association)
Shape
+newOperation(): void
One Dimmensional
+getLength(): double
Two Dimensional
+getArea(): double
This is a note
Polygon
+Polygon()
1..*
vertices
Point
-x: int
-y: int
As Diagram
By Priority
19 Items -
High
Medium
Add Instance Variables to Point
Add Operations to Point
Add Instance Variables to Polygon
Constraints
TaggedValues
Checklist
ToDoItem
Properties
Documentation
Style
Source
No ToDoItem selected
< Back
Next >
Finish
Help