Unified Modeling Language

Eran Kampf 2005

Agenda

- What is UML?
  - General definition
  - Goals
  - Some background
  - Why use UML?
- UML Diagrams
- Conclusions
- Bibliography

What is UML?

What is UML?

- UML – Unified Modeling Language
- Standard language for specifying, visualizing, constructing and documenting the artifacts of software systems.
- Collection of best engineering practices that have proven successful in modeling large and complex systems.

What is UML - Goals

- Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
- Provide extensibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development processes.
- Provide a formal basis for understanding the modeling language.
- Encourage the growth of the OO tools market.
- Support higher-level development concepts such as collaborations, frameworks, patterns and components.
- Integrate best practices.

Why use UML?

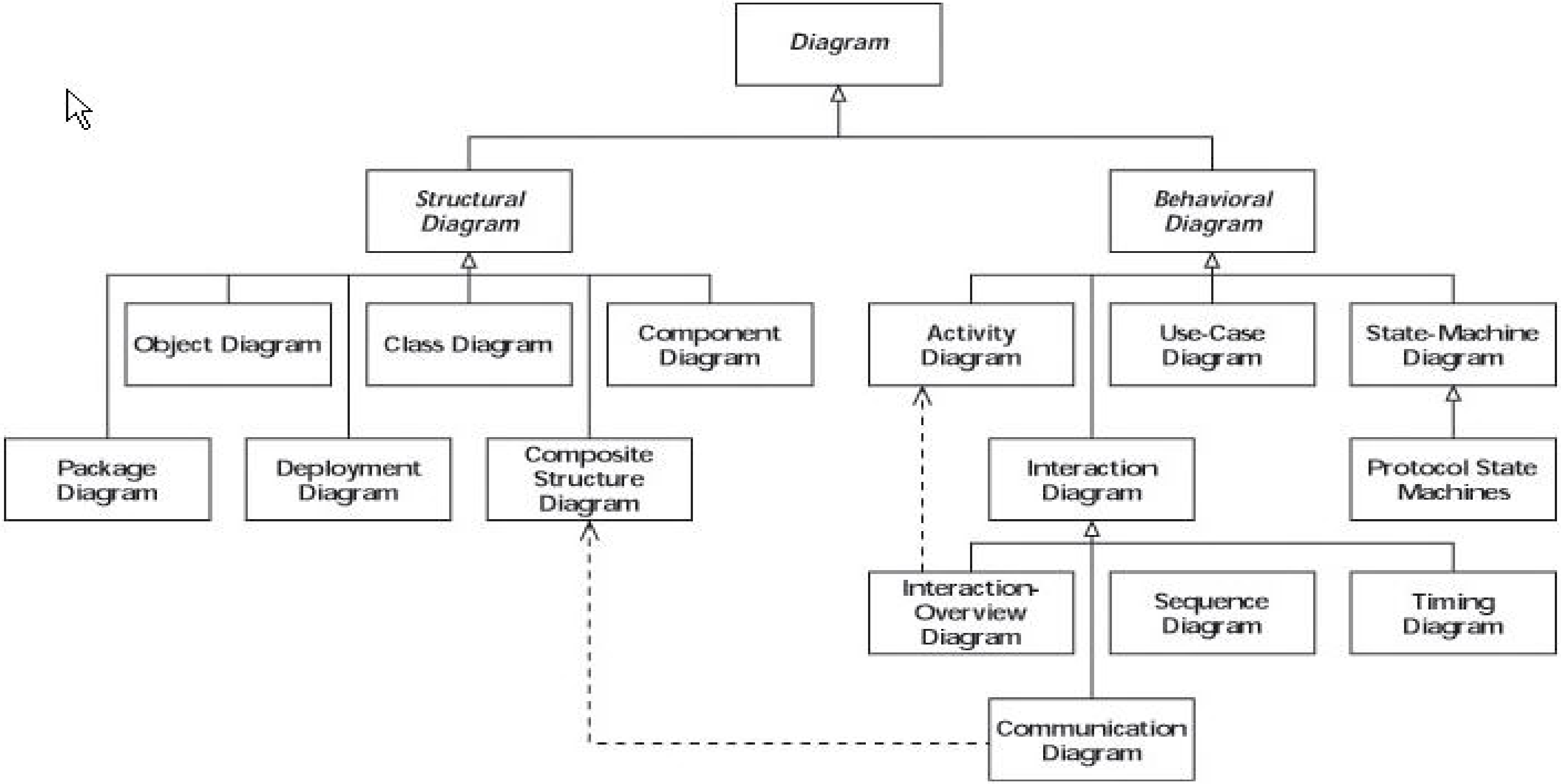
- Helps to reduce cost and time-to-market.
- Helps managing a complex project architecture.
- Helps to convey ideas between developers\designers\etc.

Background

- **1970** – Object-oriented modeling languages began to appear.
- **1996** – Release of UML 0.9 by by Grady Booch, Jim Rumbaugh of Rational Software Corporation, Ivar Jacobson of Objectory company.
- **1996** – Release of UML 1.0 by Digital Equipment, HP, I-Logix, IntelliCorp, IBM, ICON, MCI, Microsoft, Oracle, Rational, TI and Unisys.
- **1997** – Release of UML 1.1 by IBM, ObjecTime, Platinum, Ptech, Taskon, Reich and Softeam
- **2001** – Work on UML 2.0 specifications.

UML Diagrams

UML Diagrams



UML Diagrams – con

- **Structural diagrams** – Used to describe the building blocks of the system – features that do not change with time. These diagrams answer the question – What's there?
- **Behavioral diagrams –** Used to show how the system evolves over time (responds to requests, events, etc.)

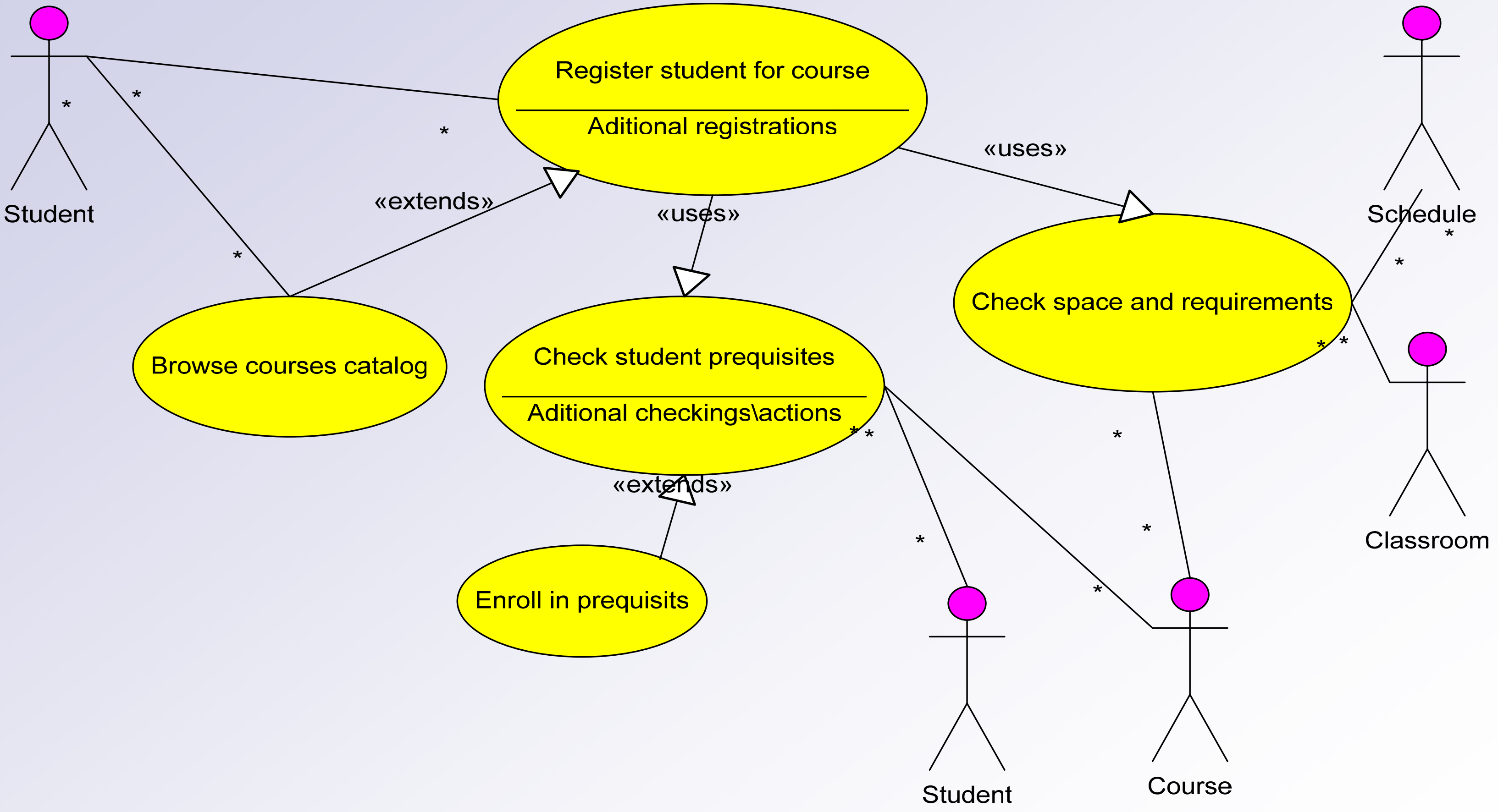
Use Case Diagrams

- Describes what a system does from the standpoint of an external observer.
- Emphasis on *what* a system does rather then *how*.
- **Scenario** – an example of what happens when someone interacts with the system.
- **Actor** – A user or another system that interacts with the modeled system.
- A use case diagram describes the relationships between *actors* and *scenarios*.
- Provides system requirements from the user's point of view.

Use Case Diagrams – cont.

- UML defines 3 kinds of associations:
  - □ **Association –** defines a relationship between an *actor* and a *use case*.
  - □ **Extend -** defines that instances of a *use case* may be augmented with some additional behavior defined in an extending *use case*.
  - □ **Uses -** defines that a *use case* uses a behavior defined in another *use case*.

Use Case Example



Class Diagrams

- Displays objects structure, contents and relationships.
- Class diagrams are static – display what interacts but not what happens when interaction occurs.

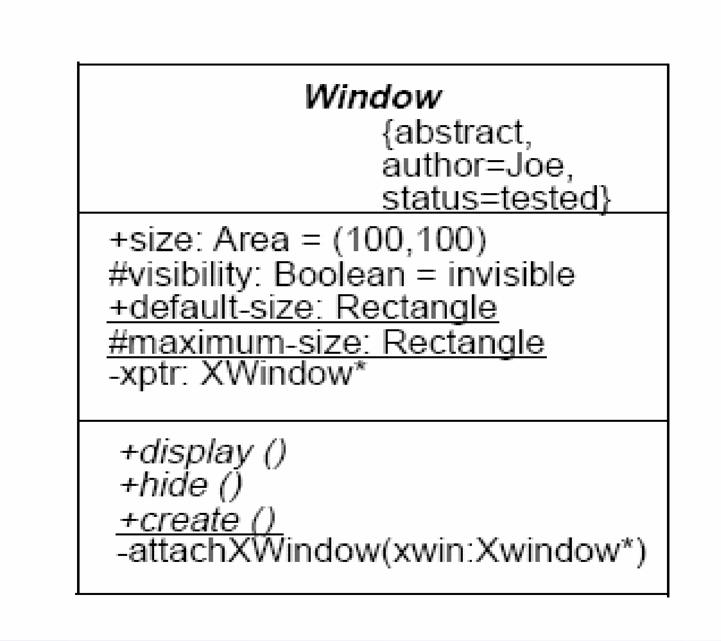
Class Diagrams – cont.

- Classes are represented by a rectangle divided to three parts: class name, attributes and operations.
- Attributes are written as:

  visibility name [multiplicity] : type-expression = initial-value
- Operations are written as:

  visibility name (parameter-list) : return type-expression
- Visibility is written as:
  + public
  # protected
  - private

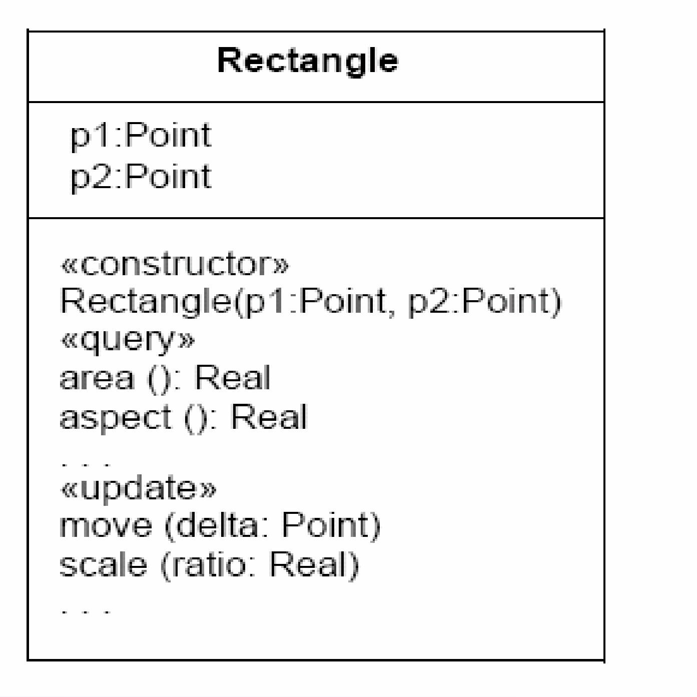
Class Diagrams – cont.



Window
{abstract,
author=Joe,
status=tested}

+size: Area = (100,100)
#visibility: Boolean = invisible
+default-size: Rectangle
#maximum-size: Rectangle
-xptr: XWindow*

+display ()
+hide ()
+create ()
-attachXWindow(xwin:Xwindow*)



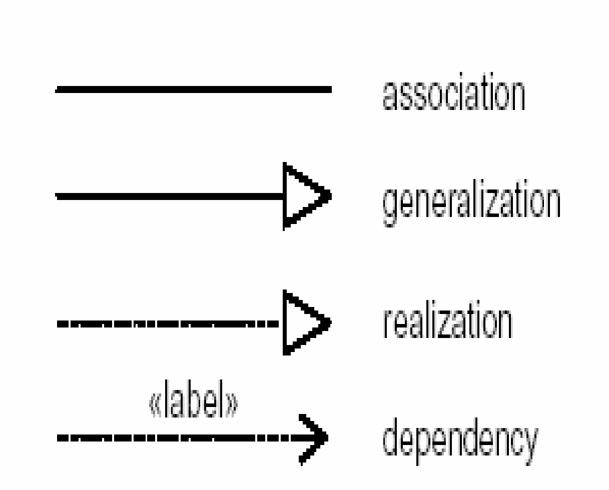
Rectangle

p1:Point
p2:Point

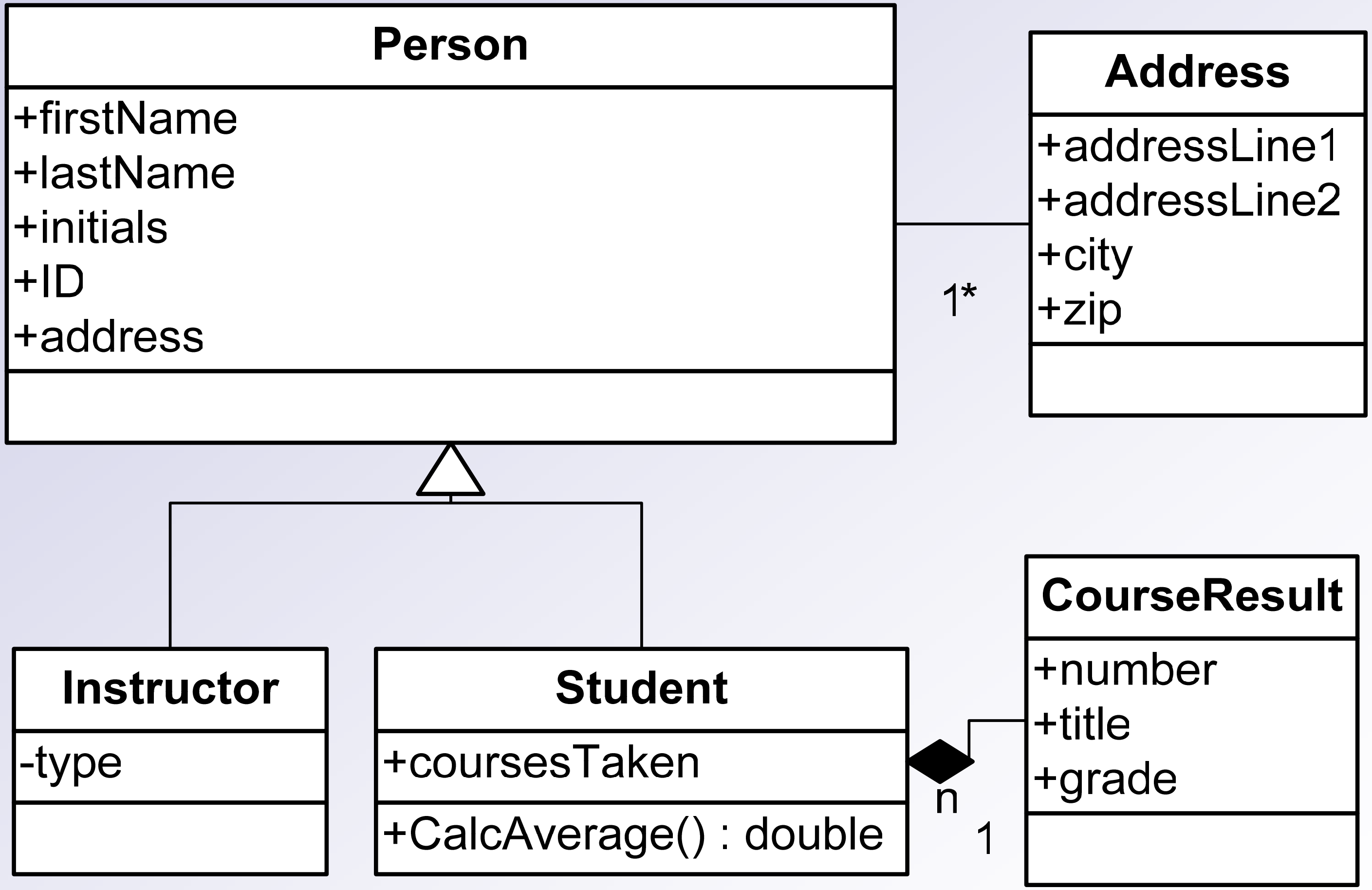
«constructor»
Rectangle(p1:Point, p2:Point)
«query»
area (): Real
aspect (): Real
. . .
«update»
move (delta: Point)
scale (ratio: Real)
. . .

Class Diagrams – Relationships

- Class Diagrams have 3 kinds of relationships:
  - □ **Association** – Two classes are associated if one class has to know about the other.
  - □ **Aggregation** – An association in which one class belongs to a collection in the other.
  - □ **Generalization** – An inheritance link indicating one class is a base class of the other.
  - □ **Dependency –** A labeled dependency between classes (such as friend classes, instaciation)



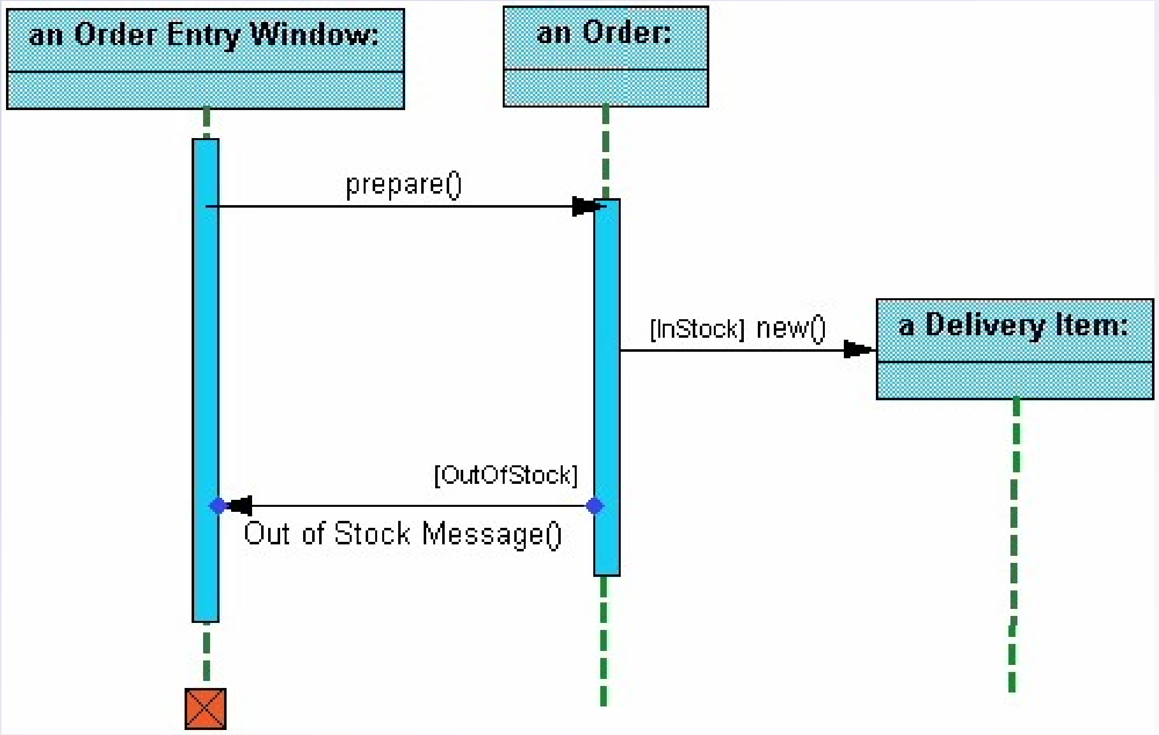
Class Diagram Example



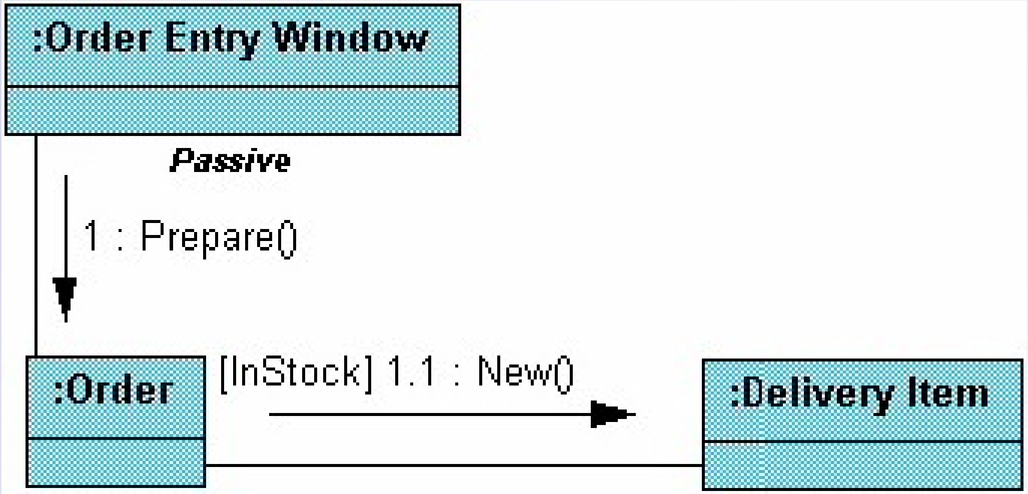
Interaction Diagrams

- Used to model the behavior of several objects in a use case.
- Demonstrates collaboration between the different objects.
- **Sequence Diagram** displays the time sequence of the objects participating in the interaction.
- **Collaboration Diagram** displays an interaction organized around the objects and their links to one another.

Sequence Diagram

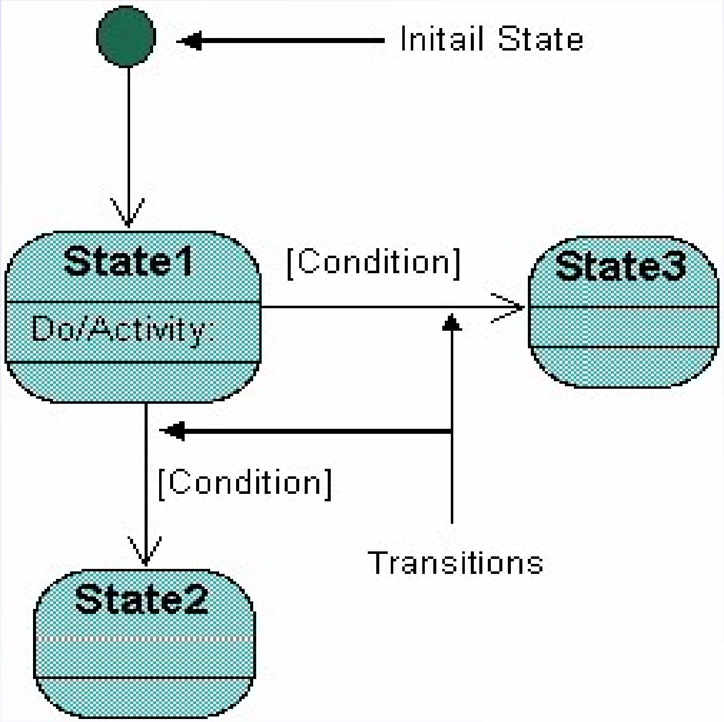


Collaboration Diagram



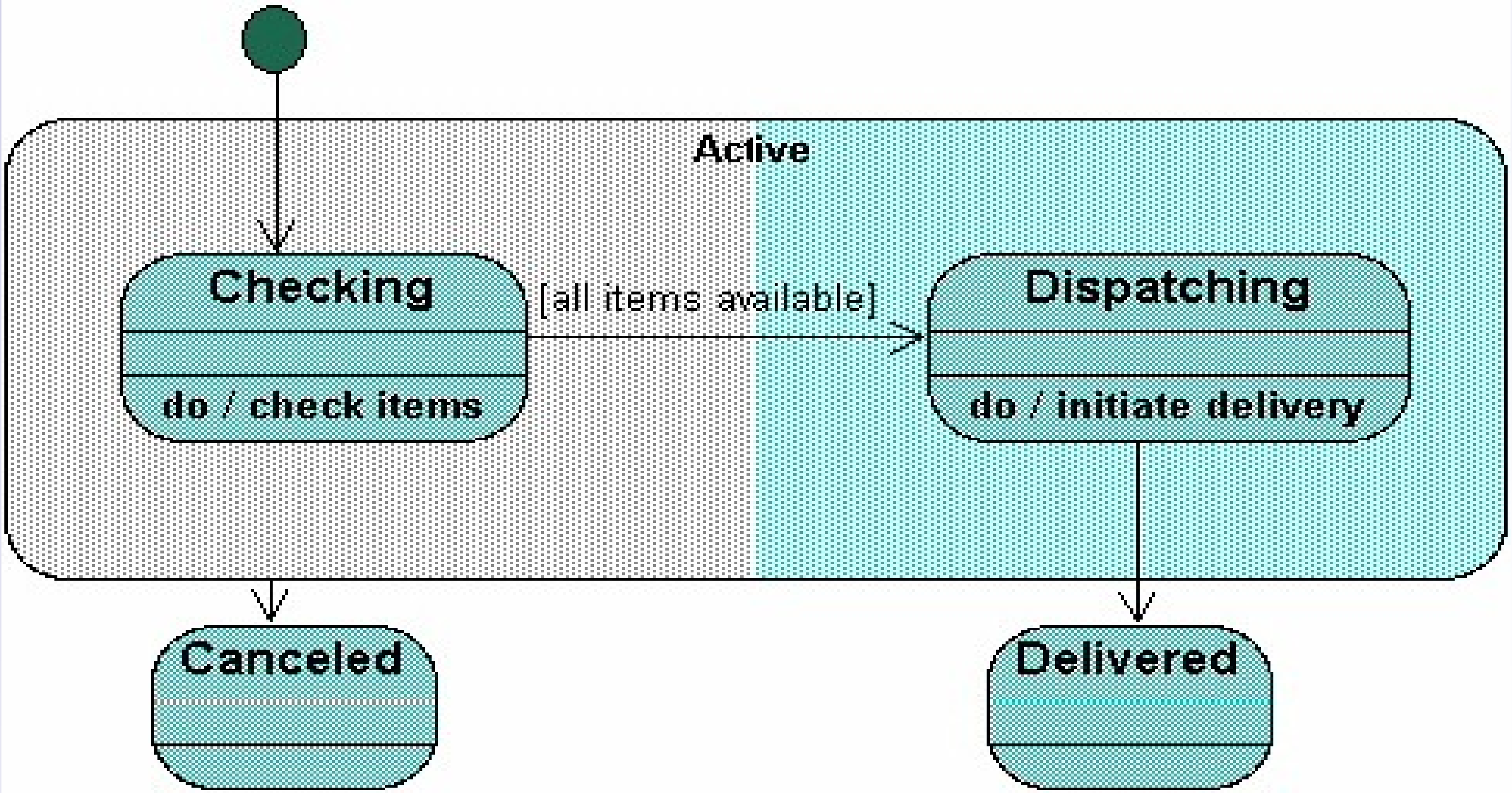
State Diagram

- State diagrams are used to describe the behavior of a system.
- State diagrams describe all of the possible states of an object as events occur.
- A state diagram begins with an initial object state (when the object is created).
- The state's activity section depicts what activities the object will be doing in this state.
- Conditions based on the activities can determine what the next state the object transitions to.



State Diagram Example

An Order object state diagram:



Activity Diagram

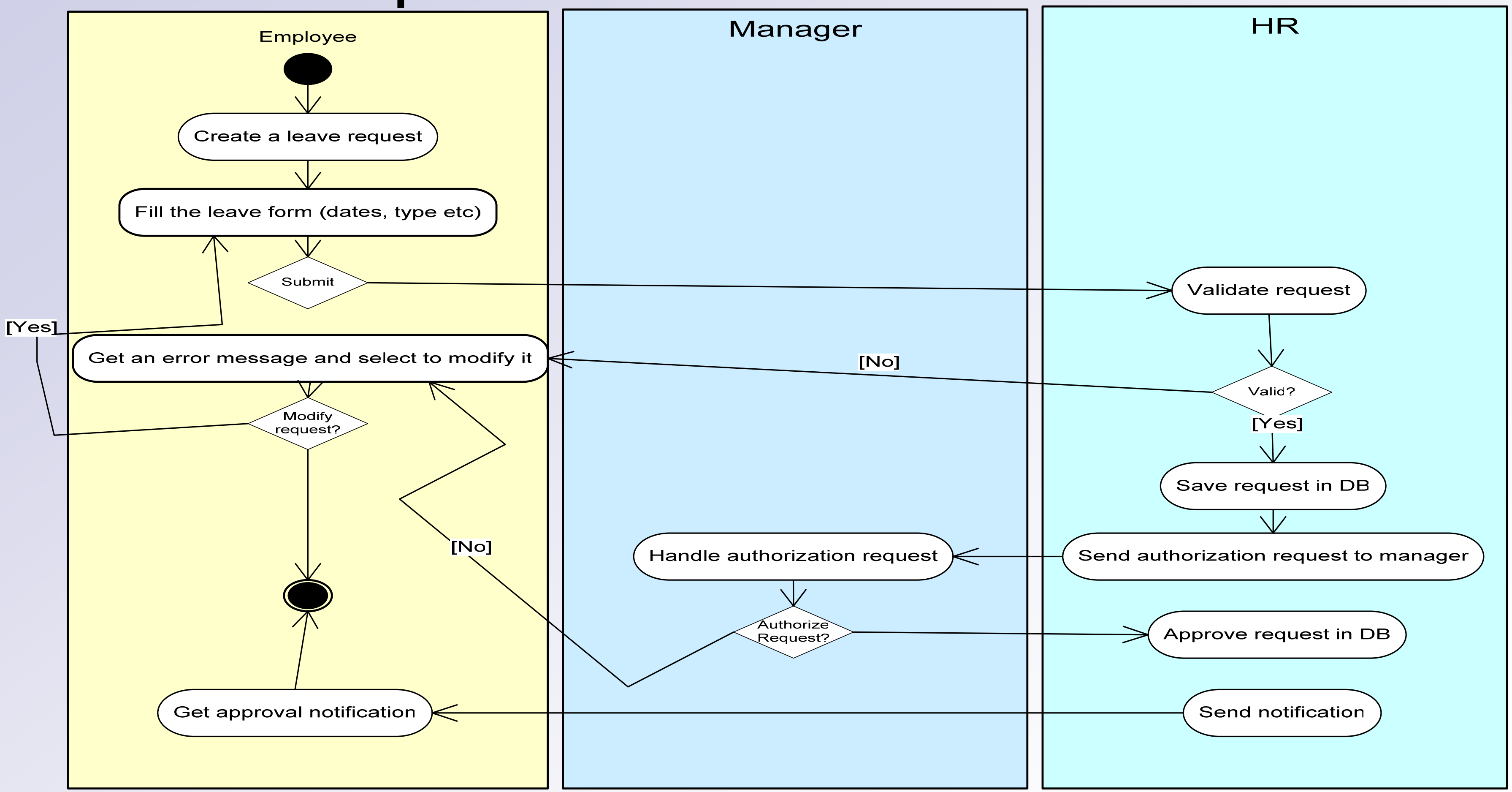
- Displays a workflow behavior of a system.
- Somewhat similar to a state diagram
  - Activities are states that represent the performance of actions or subactivities.
  - Transitions are triggered by the completion of actions or subactivities.

Activity Diagram

- Activity diagram notations:
  - **Swimlane** – Used to organize responsibility for actions and subactivities. Often corresponds to organizational units in a business model.
  - **Fork** - Splits an incoming transition into several concurrent outgoing transitions. All of the transitions fire together.
  - **Join** - Merges transitions from concurrent regions into a single outgoing transition. All the transitions fire together.
  - **Decision** – A state node that represents a decision. Each transition from this node depends on a Boolean condition.

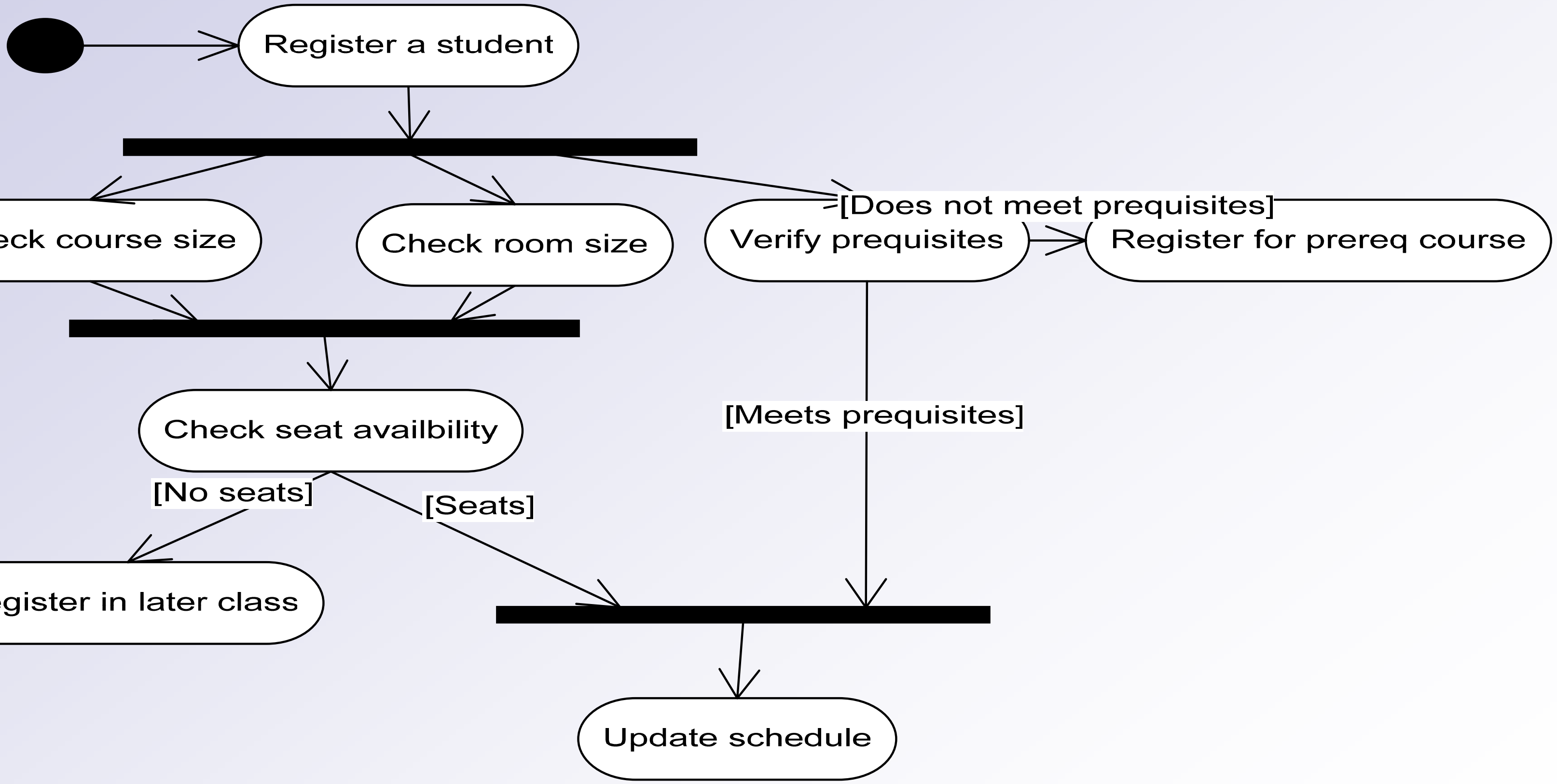
Activity Diagram

Leave request scenario



Activity Diagram

University course scheduling scenario



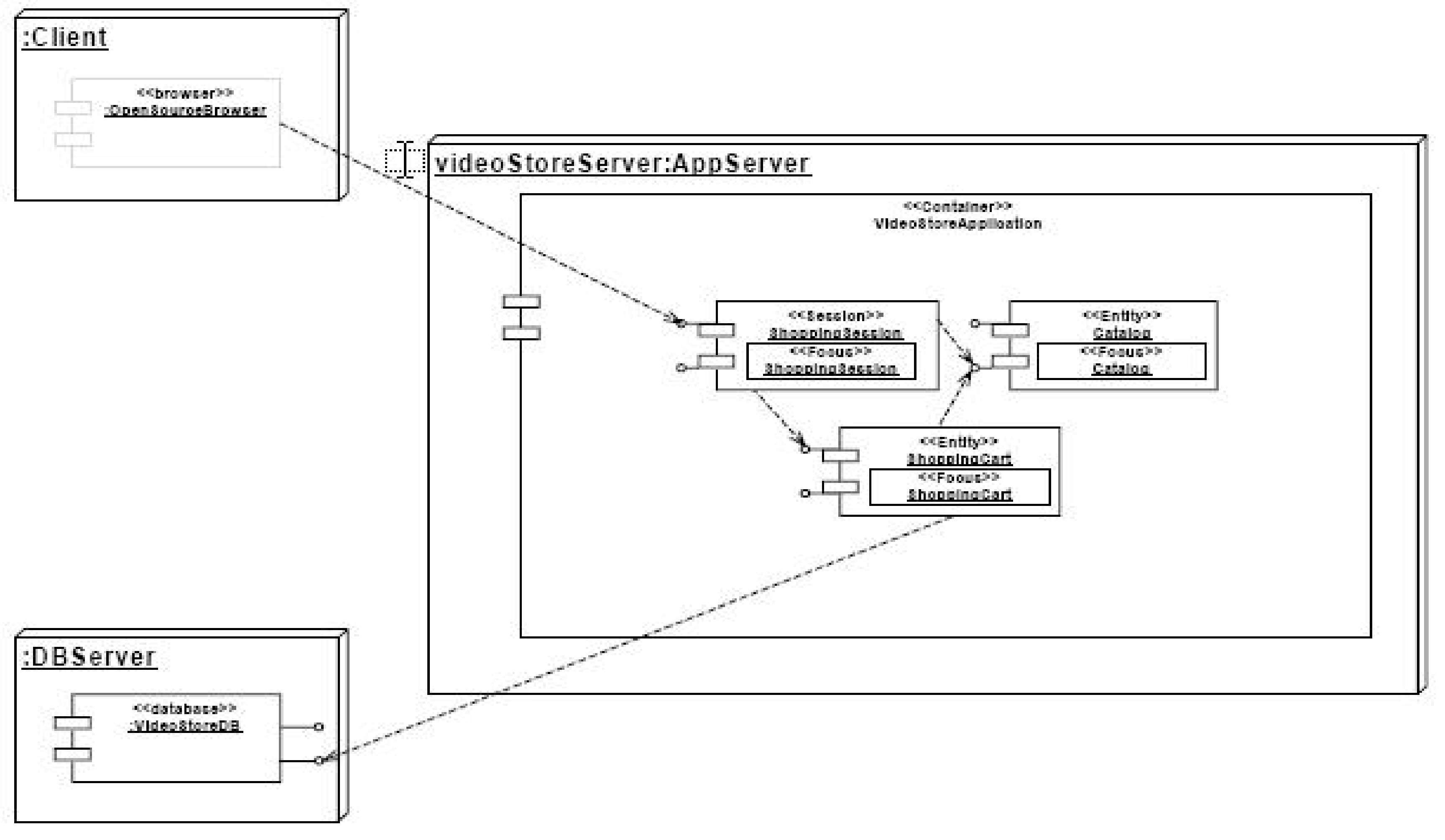
Implementation Diagrams

- show aspects of physical implementation:
  - Structure of components.
  - Run-time deployment systems.
- Two diagram types:
  - **Component diagram** – show the structure of components, including the classifiers that specify them and the artifacts that implement them.
  - **Deployment diagram** - show the structure of the nodes on which the components are deployed.
- These two diagrams are usually drawn together.

Implementation Diagrams Notations

- **Node**
  - A physical object that represents a processing resource.
  - generally, having at least a memory and often processing capability as well.
- **Component**
  - represents a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces.

Implementation Diagrams



UML and C++

- UML supports all the key concepts of OOP and C++.
- There are UML to C++ code generators on the market (and reverse engineering code to UML)

Conclusions

- UML provides a common ‘language’ for describing software projects (Not just for developers).
- Helps to define and understand the system.
- Increases efficiency and thus reduces costs and time-to-market.

The End

ביבליוגרפיה

- UML 2 for dummies – 2003.
- Kennesaw State University - CSIS 4650 - Spring 2001. By David Braun, Jeff Sivils, Alex Shapiro, Jerry Versteegh http://pigseye.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/index.htm
- OMG foundation – Official page for UML http://www.uml.org/
- Borland's UML tutorial http://bdn.borland.com/article/0,1410,31863,00.html
- Sun Microsystems Web Learning Center - WPB-120: Object-Oriented Programming with Java(TM) Technology Bundle.