System Modeling:

An Overview

Week 4: System Modeling, Part 1

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

System Modeling

- System modeling is the process of developing abstract models of a system, with each model presenting a different **view** or **perspective** of that system.

Four Perspectives

- An **external** perspective, where you model the context or environment of the system
- An **interaction** perspective, where you model the interactions between a system and its environment, or between the components of a system
- A **structural** perspective, where you model the organization of a system or the structure of the data that is processed by the system
- A **behavioral** perspective, where you model the dynamic behavior of the system and how it responds to events

- These perspectives have much in common with Kruchten's 4 + 1 view of system architecture (Kruchten 1995).

System Modeling

- Models are used in three ways.
  - ❖ Models are used during the <u>requirements engineering</u> process to help derive the requirements for a system.
    - ❖ They help the analysts and customers to understand the functionality of the system.
  - ❖ Models are used during the <u>design process</u> to describe the system to engineers implementing the system.
  - ❖ Models are used <u>after implementation</u> to document the system's structure and operation.

System Modeling (cont.)

- System modeling has now come to mean representing a system using some kind of graphical notation, which is now almost always based on notations in the Unified Modeling Language (UML).

- UML which has become a standard modeling language for modeling object-oriented systems.

System Modeling Using UML

- The UML has many diagram types and so supports the creation of many different types of system model.

❖ However, a survey in 2007 showed that most users of the UML thought that five diagram types could represent the essentials of a system (Erickson and Siau. “Theoretical and Practical Complexity of Modeling Methods.” *ACM* 50, no. 8 (2007): 46–51. doi:10.1145/1278201.1278205.)

❖ Class diagrams

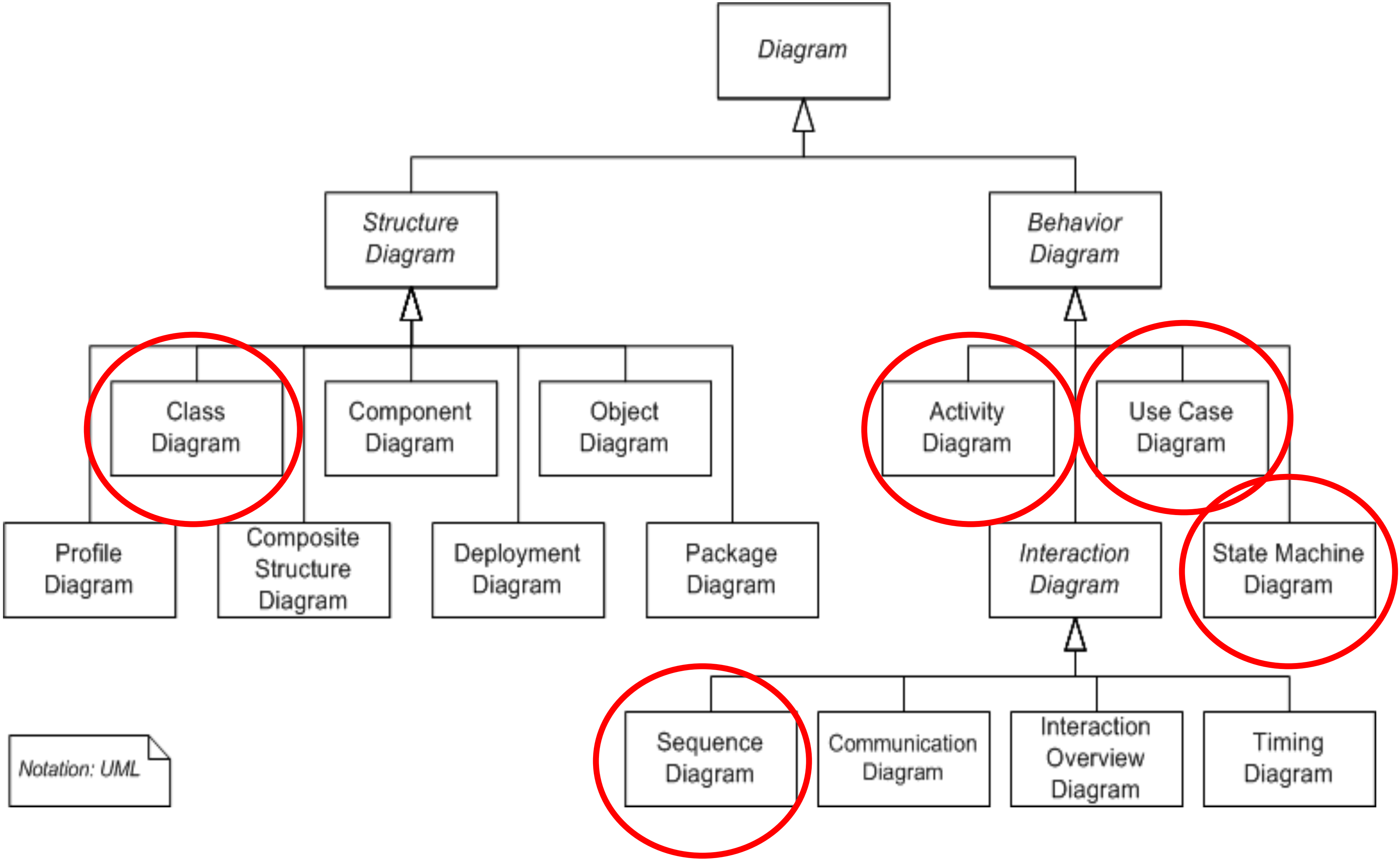
❖ Use case diagrams

❖ Sequence diagrams

❖ Activity diagrams

❖ State diagrams

Types of UML Diagrams



The Rise of UML

Week 4: System Modeling, Part 1

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

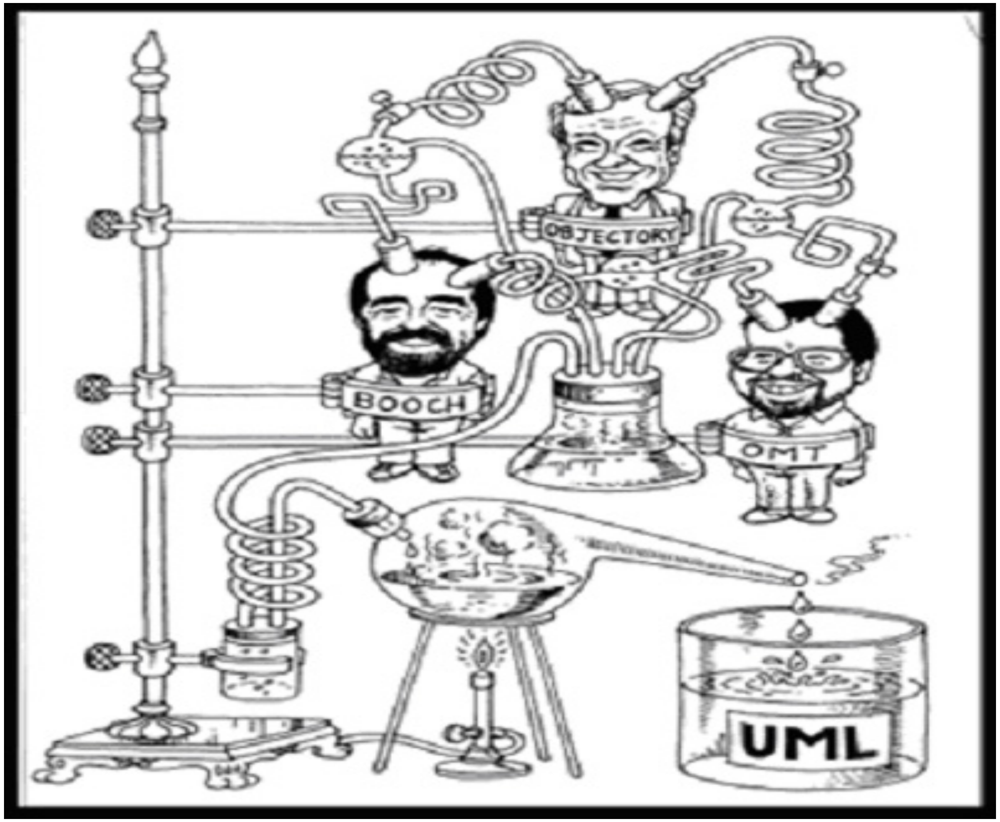
SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

The Rise of UML

- Structured design methods were invented in the 1970s to support function-oriented design methods.
  - ❖ Function-oriented methods use functions as their central design concept and often start by identifying the data flow through a system.
- They evolved in the 1980s and 1990s to support object-oriented design (OOD), which uses:
  - ❖ Objects as their central design concept
  - ❖ Use cases to describe the processes in the system's environment
- The unification of different structured methods for object-oriented design led to the development of the Unified Modeling Language (UML).

A Short History of UML

- **Before 1994**: James Rumbaugh invented the Object Modeling Technique for OOA; Grady Booch invented his Booch method for OOD; Ivar Jacobson invented his Objectory (for OOSE).



http://www.ibm.com/developerworks/cn/rational/theme/rational-rup/Umlbegin.gif

OOA, OOD, and OOP

- Object-oriented methods may be applied to different phases in the software life cycle: analysis, design, implementation, and so on.
  - ❖ Analysis phase (OOA): discover, model, and understand the requirements of the system
  - ❖ Design phase (OOD): create the <u>abstractions</u> and mechanisms necessary to meet the system's requirements as determined in the analysis phase
  - ❖ Implementation phase (OOP): convert OODs to concrete programs in a certain OO programming language, using the OO methods (three pillars of OOP)

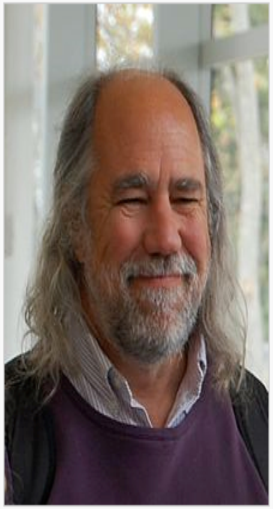
A Short History of UML

- **1994**: Rumbaugh joined Rational.
- **1995**: Jacobson joined Rational. The three methodologists were collectively referred to as the **Three Amigos**. (See next slide.)
- **1996**: Rational tasked the Three Amigos with the development of a nonproprietary Unified Modeling Language.
- **January 1997**: UML 1.0 specification draft was proposed to the OMG.
- **November 1997**: UML 1.1 (finalized semantics) was adopted by OMG.
- **1998–2004**: Subsequent minor revisions (UML 1.3, 1.4, and 1.5) fixed shortcomings and bugs.
- **2005**: UML 2.0, a major revision, was adopted by the OMG in 2005.

Grady Booch

From Wikipedia, the free encyclopedia

Grady Booch (born February 27, 1955) is an American software engineer, best known for developing the Unified Modeling Language with Ivar Jacobson and James Rumbaugh. He is recognized internationally for his innovative work in software architecture, software engineering, and collaborative development environments.


Grady Booch in 2011.

Contents [hide]

Biography [edit]

Booch earned his bachelor's degree in 1977 from the United States Air Force Academy and a master's degree in electrical engineering in 1979 from the University of California, Santa Barbara.[1]

Grady served as Chief Scientist of Rational Software Corporation since its founding in 1981 and through its acquisition by IBM in 2003, where he kept working until March, 2008.

James Rumbaugh

From Wikipedia, the free encyclopedia

James E. Rumbaugh (born August 22, 1947) is an American computer scientist and object-oriented methodologist[1] who is best known for his work in creating the Object Modeling Technique (OMT) and the Unified Modeling Language (UML).

Contents [hide]

Biography [edit]

Born in Bethlehem, Pennsylvania, Rumbaugh received a B.S. in physics from the Massachusetts Institute of Technology (MIT), an M.S. in astronomy from the California Institute of Technology (Caltech), and received a Ph.D. in computer science from MIT under Professor Jack Dennis.[1]

Rumbaugh started his career in the 1960s at Digital Equipment Corporation (DEC) as a lead research scientist. From 1968 to 1994 he worked at the General Electric Research and Development Center developing technology, teaching, and consulting. At General Electric he also led the development of Object-modeling technique (OMT), an object modeling language for software modeling and designing.

In 1994, he joined Rational Software, where he worked with Ivar Jacobson and Grady Booch ("the Three Amigos") to develop Unified Modeling Language (UML). Later they merged their software development methodologies, OMT, OOSE and Booch into the Rational Unified Process (RUP). In 2003 he moved to IBM, after its acquisition of Rational Software. He retired in 2006.[1]

Ivar Jacobson

From Wikipedia, the free encyclopedia

Ivar Hjalmar Jacobson (born 1939) is a Swedish computer scientist and software engineer, known as major contributor to UML, Objectory, Rational Unified Process (RUP), aspect-oriented software development and Essence.

Ivar Jacobson

Born	September 2, 1939 (age 75) Ystad, Sweden
Residence	Switzerland
Nationality	Swedish
Fields	Electrical Engineering, Computer Science, Software Engineering
Institutions	Ericsson, Objective Systems, Rational Software, IBM, Ivar Jacobson International
Alma mater	Chalmers Institute of Technology in Gothenburg, Royal Institute of Technology in Stockholm
Known for	components and component architecture, use-cases and use-case driven development, SDL,

Contents [hide]

Biography [edit]

Ivar Jacobson was born in Ystad, Sweden on 2 September 1939. He received his Master of Electrical Engineering degree at Chalmers Institute of Technology in Gothenburg in 1962 and a Ph.D. at the Royal Institute of Technology in Stockholm in 1985 on a thesis on Language Constructs for Large Real Time Systems.

What Is UML?

- It is a language.
  - ❖ It is not simply a notation for drawing diagrams.
  - ❖ But also a complete language for capturing knowledge (semantics) about a subject (domain) and expressing knowledge (syntax).
- It is a modeling language.
  - ❖ Modeling involves understanding/capturing the essence of a subject/domain.
- It is a unifying modeling language:
  - ❖ It unifies the IT industry’s best engineering practices (principles, techniques, methods, and tools).
- “The UML is a language for visualizing, specifying, constructing, and documenting all the artifacts of a software system.”

  —The Three Amigos, *The UML User Guide*, 2nd ed.

Three Building Blocks of UML

- Things
  - ❖ Structural things: the static parts (nouns) of UML models (e.g., classes)
  - ❖ Behavioral things: the dynamic parts (verbs) of UML models (e.g., actions)
  - ❖ Grouping things: the organizational parts of UML models (e.g., packages)
  - ❖ Annotational things: the explanatory parts of UML models (e.g., notes)
- Relationships
  - ❖ Dependency: A change to one element may affect the other.
  - ❖ Association: A class is linked to the other.
  - ❖ Generalization: A class is a special/general case of the other.
  - ❖ Realization: A class carries out a contract the other guarantees.
- Diagrams

Class Diagrams—Classes

Week 4: System Modeling, Part 1

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

SYRACUSE UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE

Diagrams

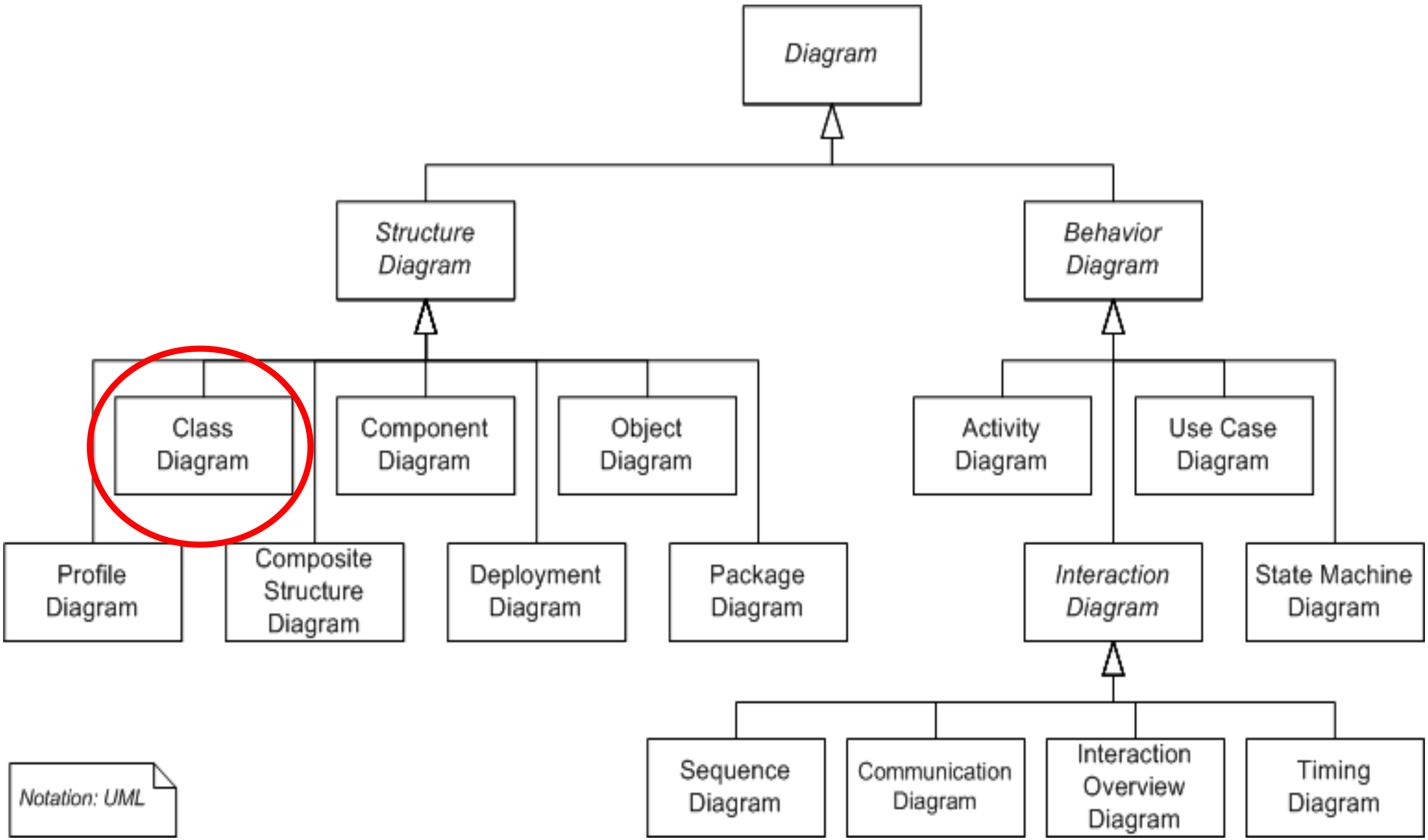
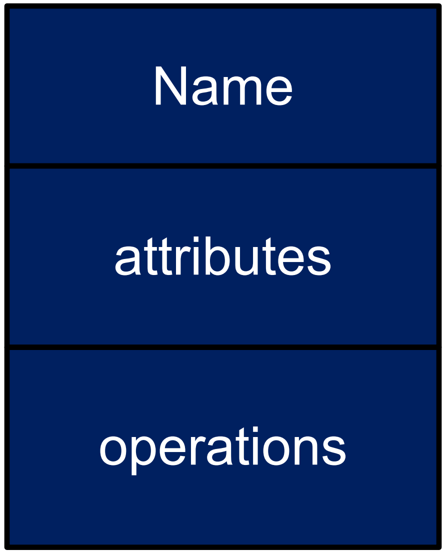


Diagram
Structure Diagram
Behavior Diagram
Class Diagram
Component Diagram
Object Diagram
Activity Diagram
Use Case Diagram
Profile Diagram
Composite Structure Diagram
Deployment Diagram
Package Diagram
Interaction Diagram
State Machine Diagram
Sequence Diagram
Communication Diagram
Interaction Overview Diagram
Timing Diagram
Notation: UML

Classes

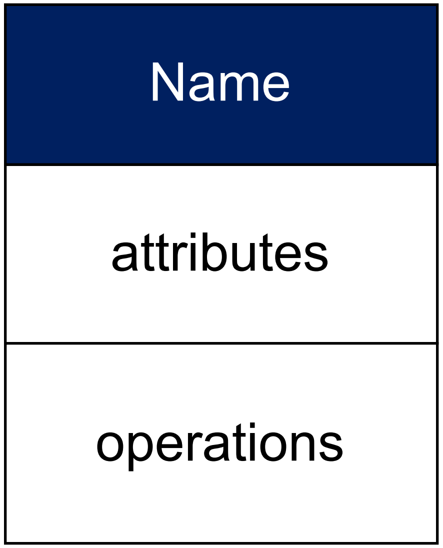


Classes are the most important building blocks of any OO system.

A class is a description of a set of objects that share the same attributes, operations, relationships, and semantics.

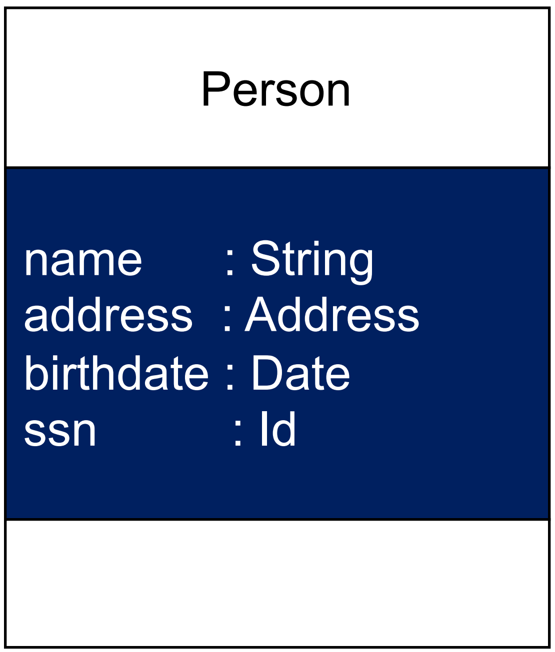
Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations in separate, designated compartments.

Class Names



The name of the class is the only required field in the graphical representation of a class. It always appears in the top-most compartment.

Class Attributes



An attribute is a named property of a class that describes the object being modeled.

In the class diagram, attributes appear in the second compartment just below the name-compartment.

Class Attributes (cont.)

Person
name : String address : Address birthdate : Date / age : Date ssn : Id

Attributes are usually listed in the form:

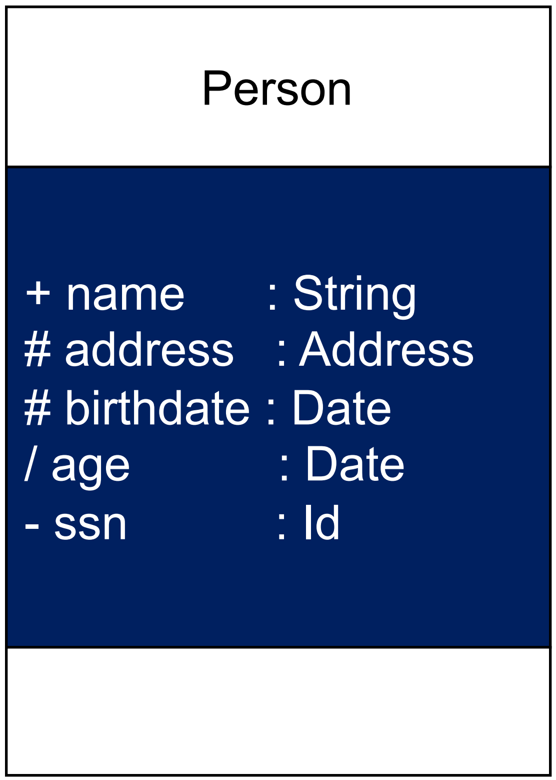
attributeName : Type

A derived attribute is one that can be computed from other attributes but doesn't actually exist.

For example, a Person's age can be computed from his birth date. A derived attribute is designated by a preceding "/" as in:

/ age : Date

Class Attributes (cont.)



Attributes can be:

+ public
# protected
- private
/ derived

Class Attributes (cont.)

Person
+ name : String # address : Address # birthdate : Date / age : Date - ssn : **Id[0..1]**

Attributes can be:

+ \+ public
+ \# protected
+ \- private
+ / derived

Class Operations

Person
name : String address : Address birthdate : Date ssn : Id
eat() sleep() work() play()

Operations describe the class behavior and appear in the third compartment.

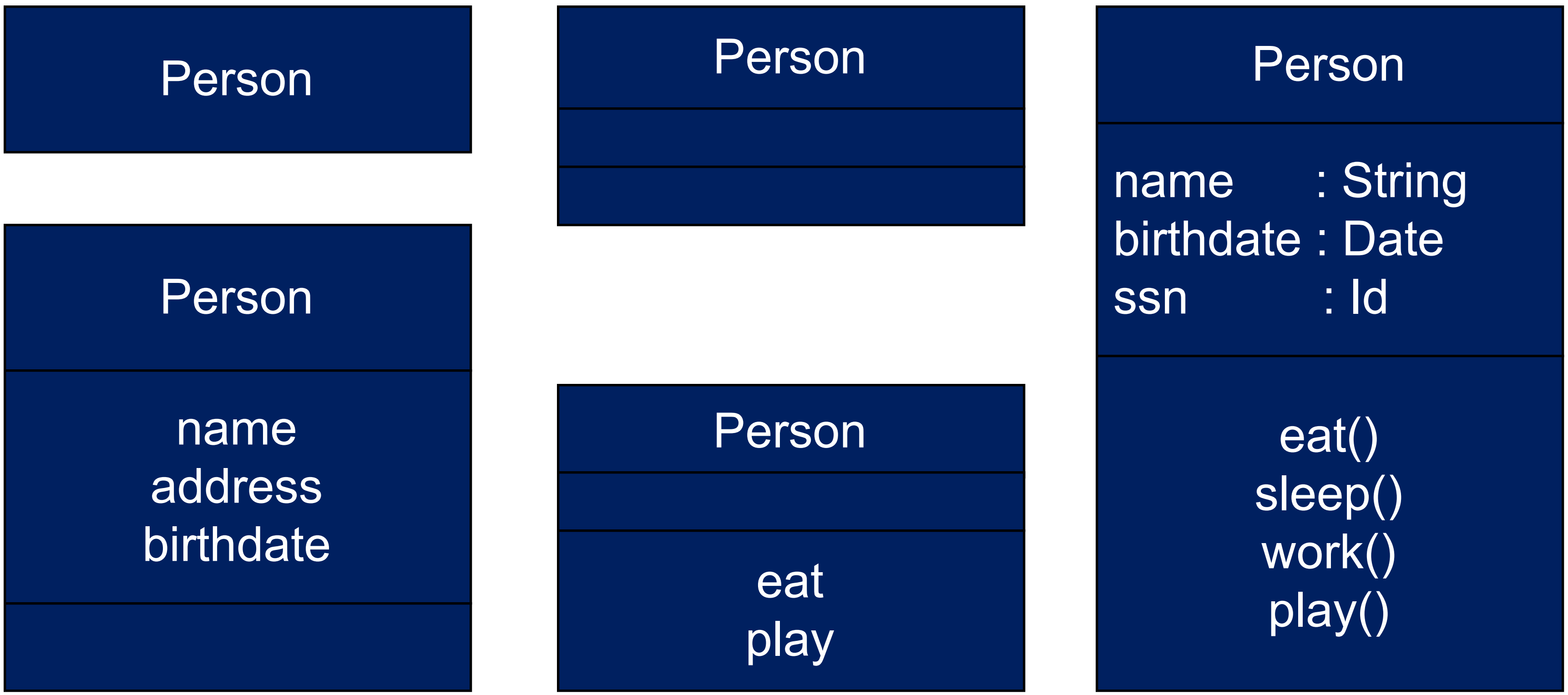
Class Operations (cont.)

PhoneBook
newEntry (n : Name, a : Address, p : PhoneNumber, d : Description) getPhone (n : Name, a : Address) : PhoneNumber

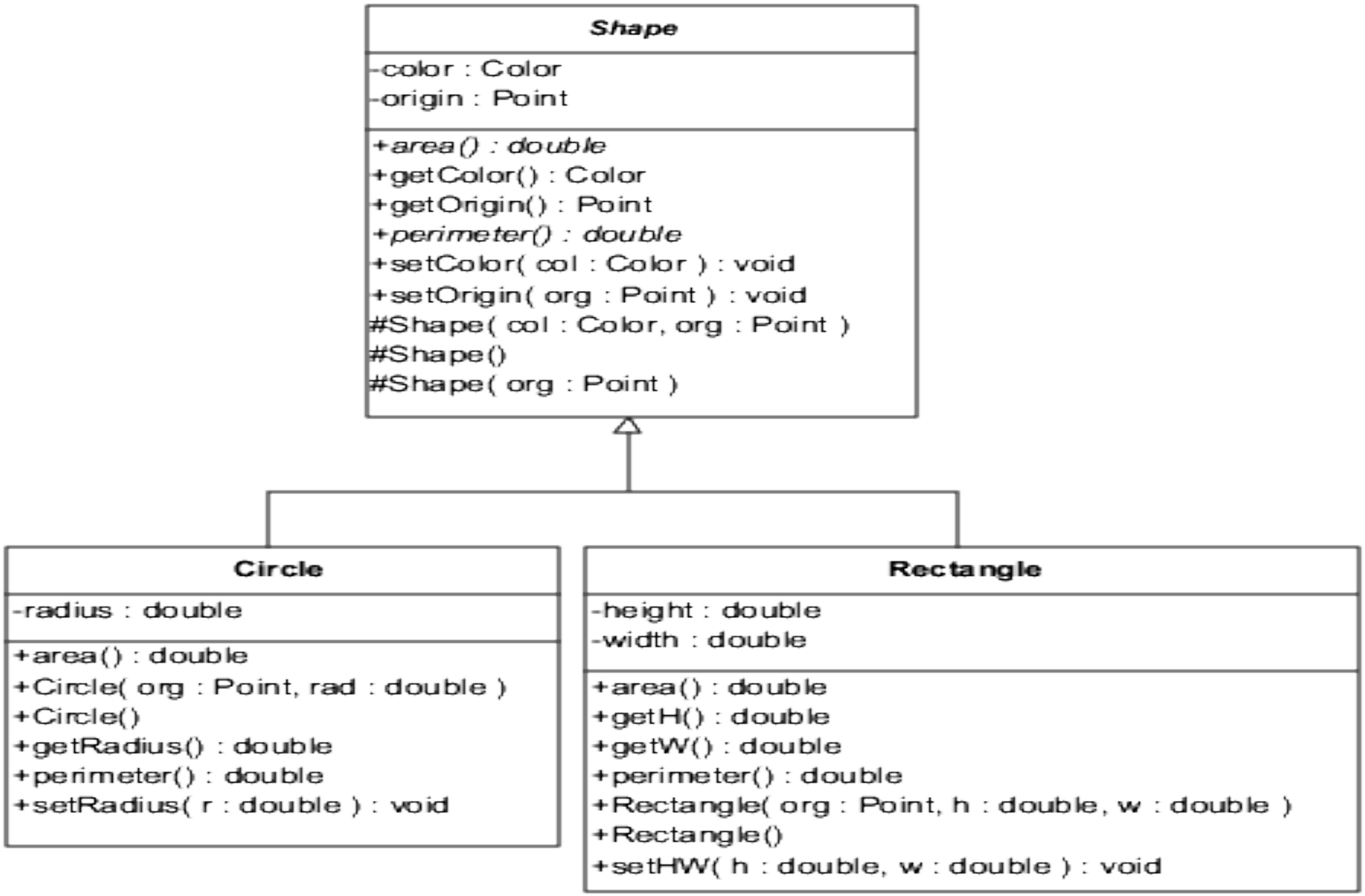
You can specify an operation by stating its signature: listing the name, type, and default value of all parameters, and, in the case of functions, a return type.

Depicting Classes

When drawing a class, you don't need to show attributes and operation in every diagram.



Depicting Classes



Class Responsibilities

A class may also include its responsibilities in a class diagram.

A responsibility is a contract or obligation of a class to perform a particular service.

SmokeAlarm
Responsibilities -- sound alert and notify guard station when smoke is detected. -- indicate battery state

Class Diagrams—Relationships

Week 4: System Modeling, Part 1

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

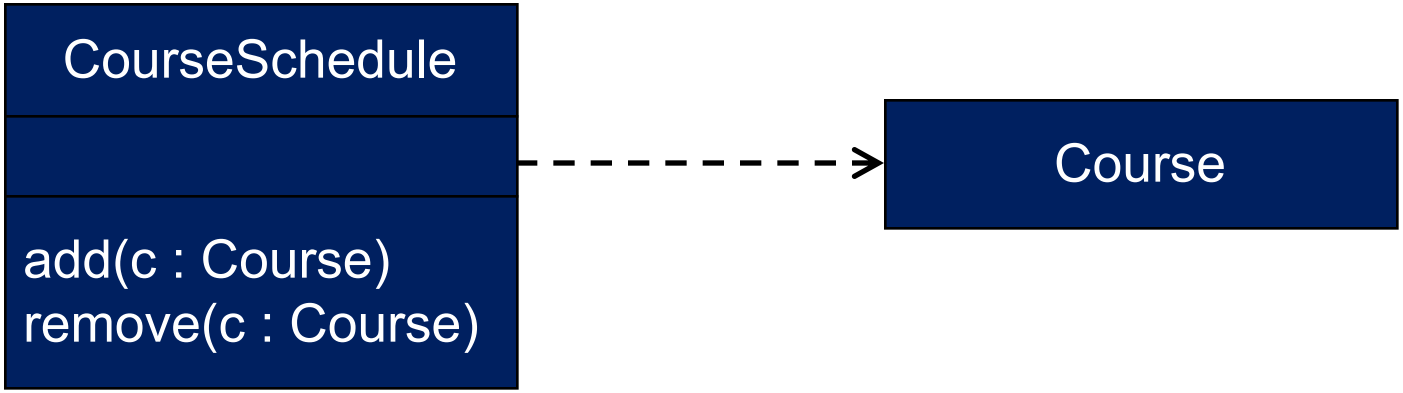
Three Building Blocks of UML

- Things
  - ❖ Structural things: the static parts (nouns) of UML models (e.g., classes)
  - ❖ Behavioral things: the dynamic parts (verbs) of UML models (e.g., actions)
  - ❖ Grouping things: the organizational parts of UML models (e.g., packages)
  - ❖ Annotational things: the explanatory parts of UML models (e.g., notes)
- Relationships
  - ❖ Dependency: A change to one element may affect the other.
  - ❖ Generalization: A class is a special/general case of the other.
  - ❖ Association: A class is linked to the other.
  - ❖ Realization: A class carries out a contract the other guarantees.
- Diagrams

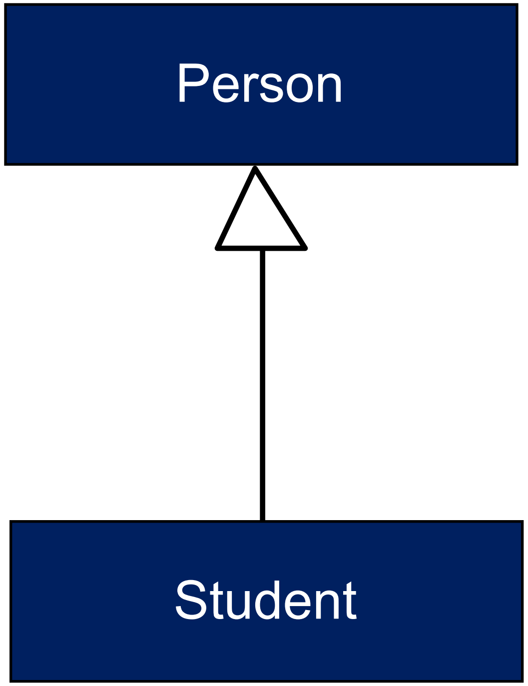
Relationships: Dependency

A dependency indicates a semantic relationship between two or more elements.

The dependency from CourseSchedule to Course exists because Course is used in both the **add** and **remove** operations of CourseSchedule.



Relationships: Generalization



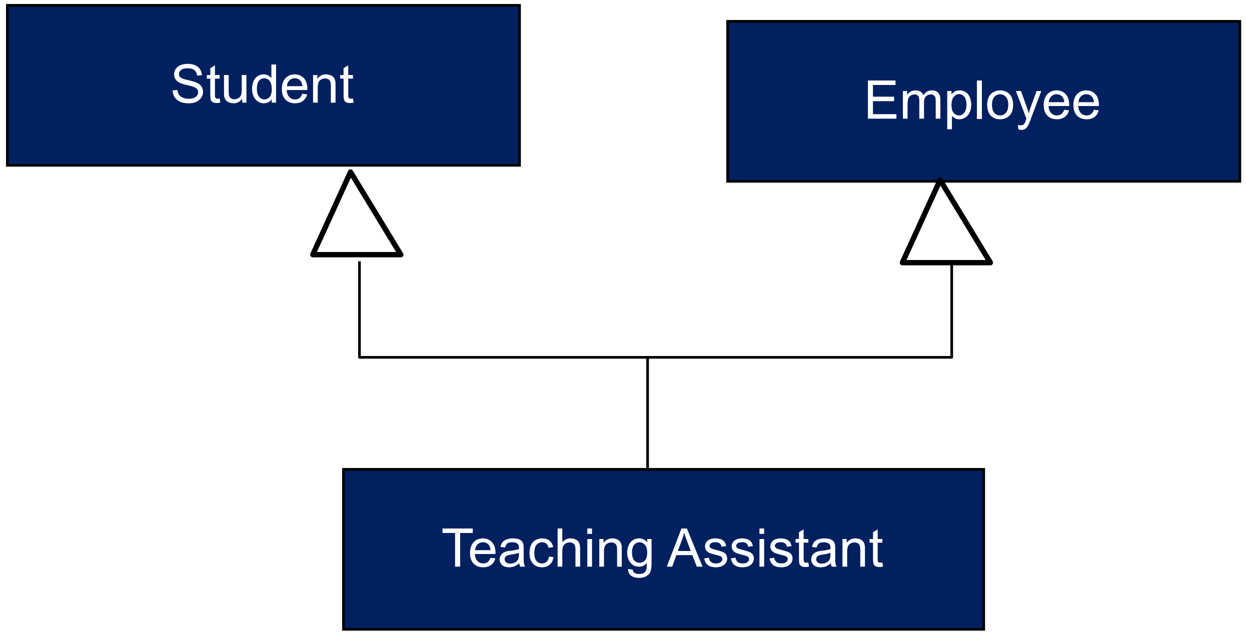
A generalization connects a subclass to its superclass.

It denotes an inheritance of attributes and behavior from the superclass to the subclass.

It also indicates a specialization in the subclass of the more general superclass.

Relationships: Generalization (cont.)

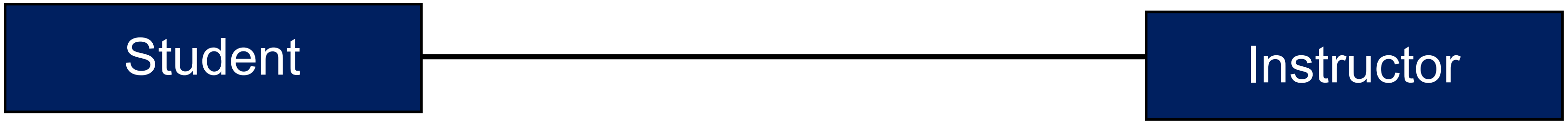
UML permits a class to inherit from multiple superclasses, although some programming languages (e.g., Java) do not permit multiple inheritance.



Relationships: Association

If two classes in a model need to communicate with each other, there must be link between them.

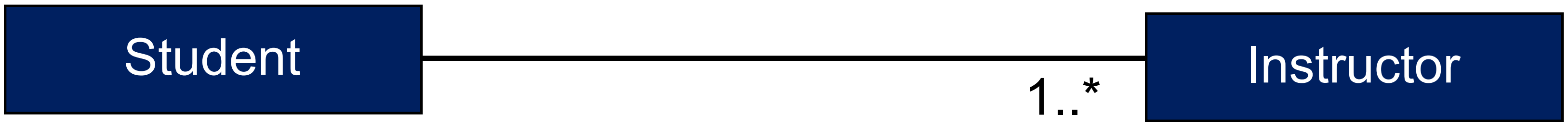
An association denotes that link.



Relationships: Association (cont.)

We can indicate the multiplicity of an association by adding multiplicity adornments to the line denoting the association.

The example indicates that a Student has one or more Instructors:

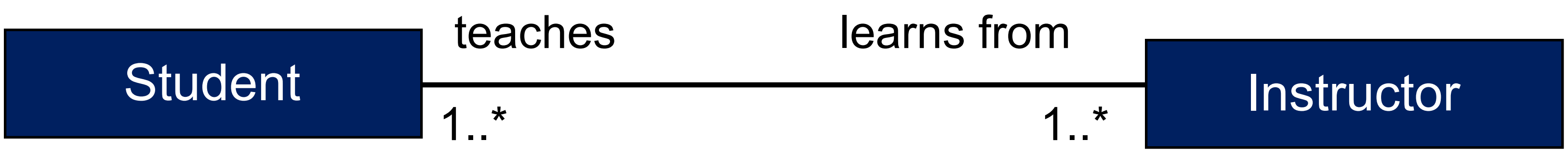


Relationships: Association (cont.)

The example indicates that every Instructor has one or more Students:

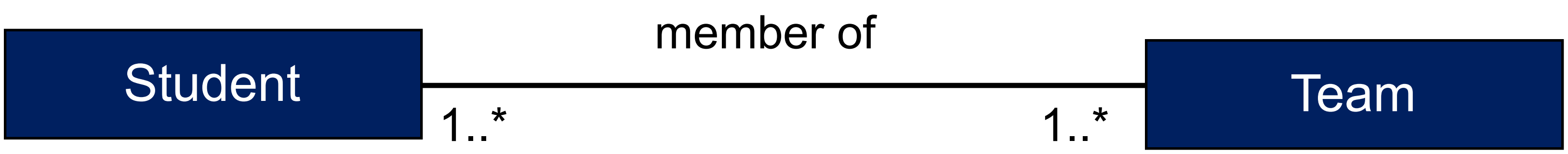
Relationships: Association (cont.)

We can also indicate the behavior of an object in an association (i.e., the role of an object) using role names.



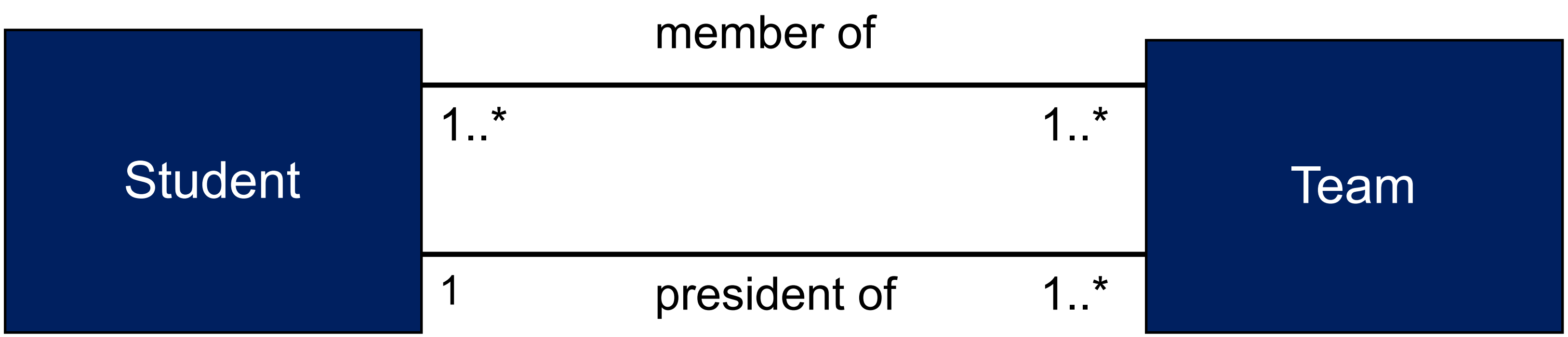
Relationships: Association (cont.)

We can also name the association.

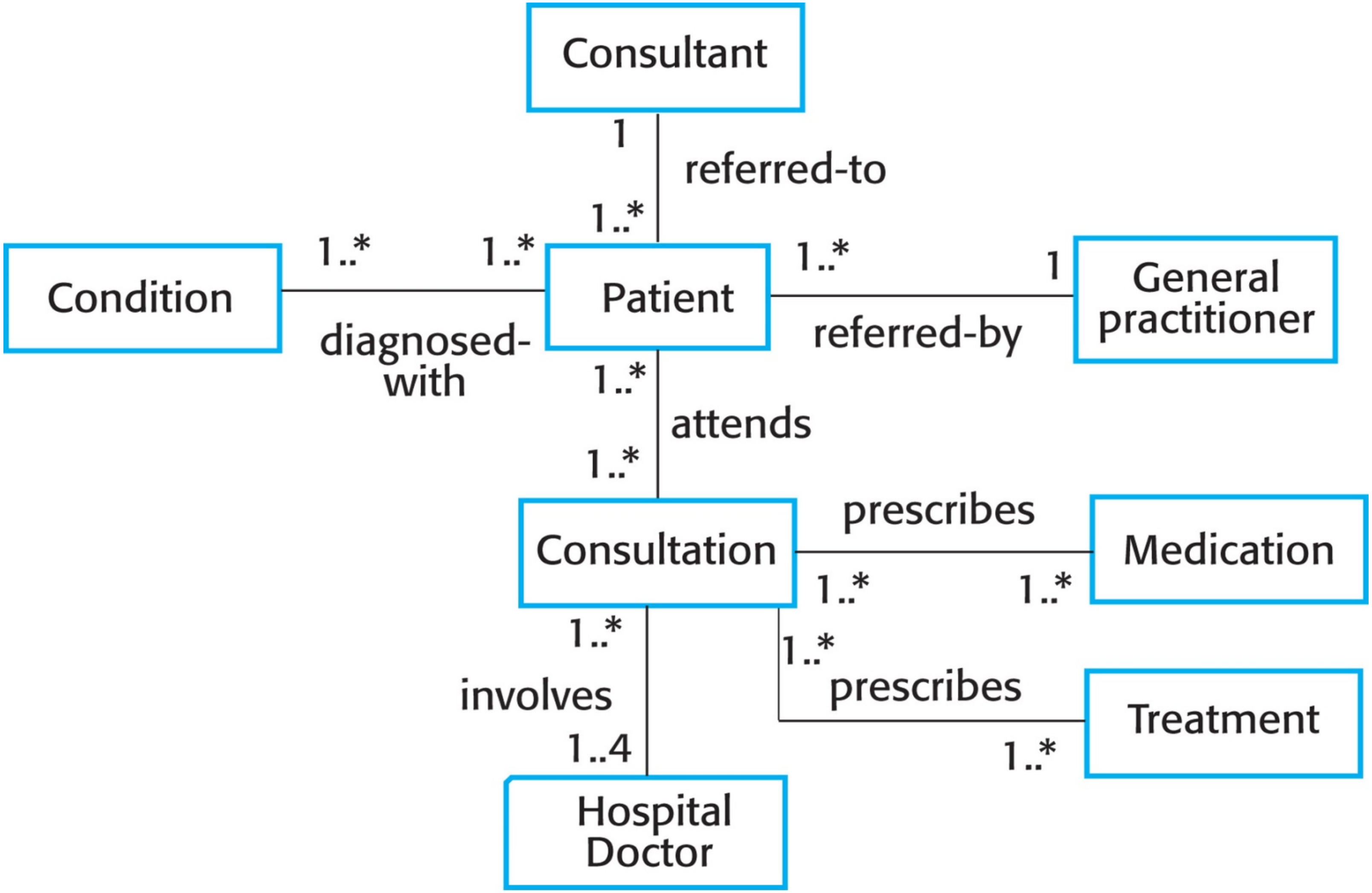


Relationships: Association (cont.)

We can specify dual associations.



Classes/Associations in Mentcare

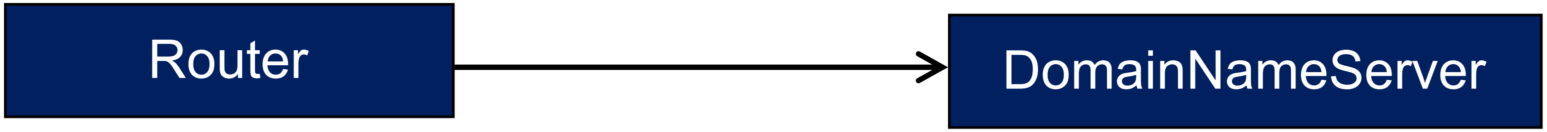


Relationships: Association (cont.)

We can <u>constrain</u> the association relationship by defining the navigability of the association.

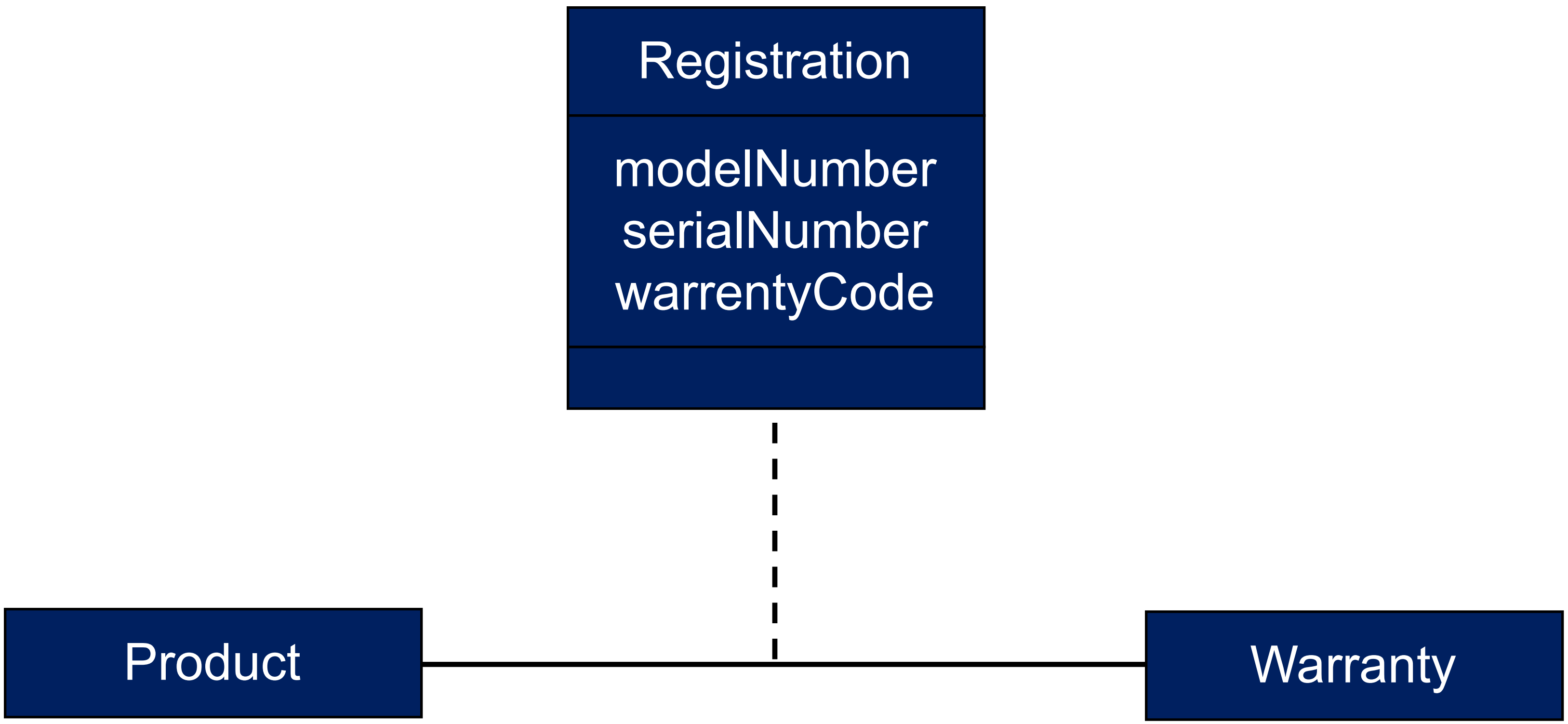
Here, a Router object requests services from a DNS object by sending messages to (invoking the operations of) the server.

The direction of the association indicates that the server has no knowledge of the Router.



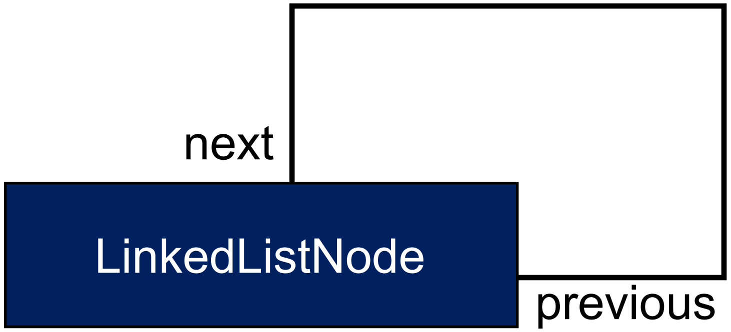
Relationships: Association (cont.)

Associations can also be objects themselves, called link classes or an association classes.



Relationships: Association (cont.)

A class can have a self-association.

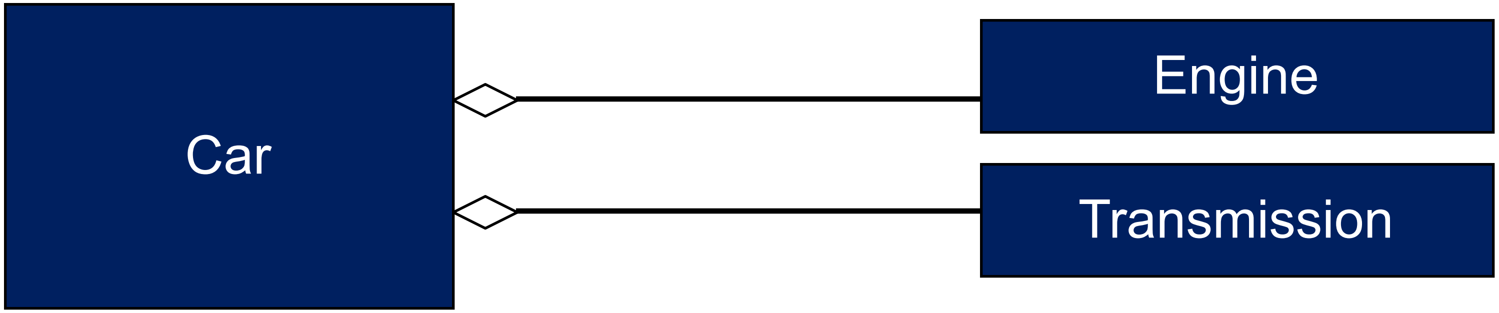


Relationships: Association (cont.)

We can model objects that contain other objects by way of special associations called aggregations and compositions.

An **aggregation** specifies a whole-part relationship between an aggregate (a whole) and a constituent part, where the part can exist independently from the aggregate.

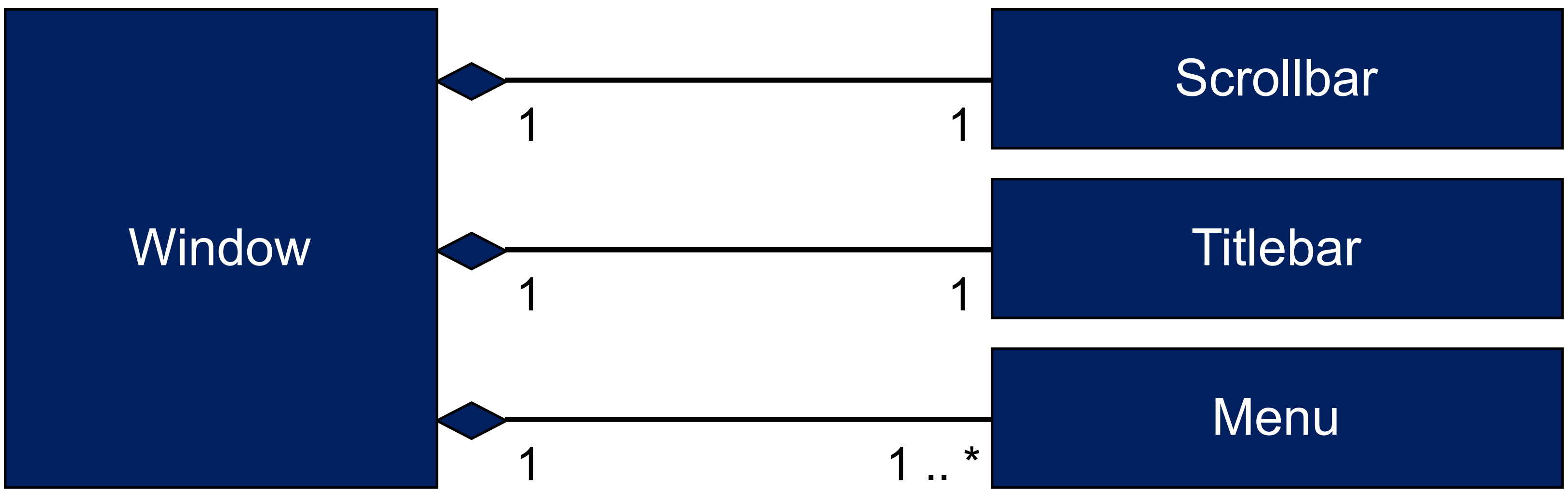
Aggregations are denoted by a hollow diamond on the association.



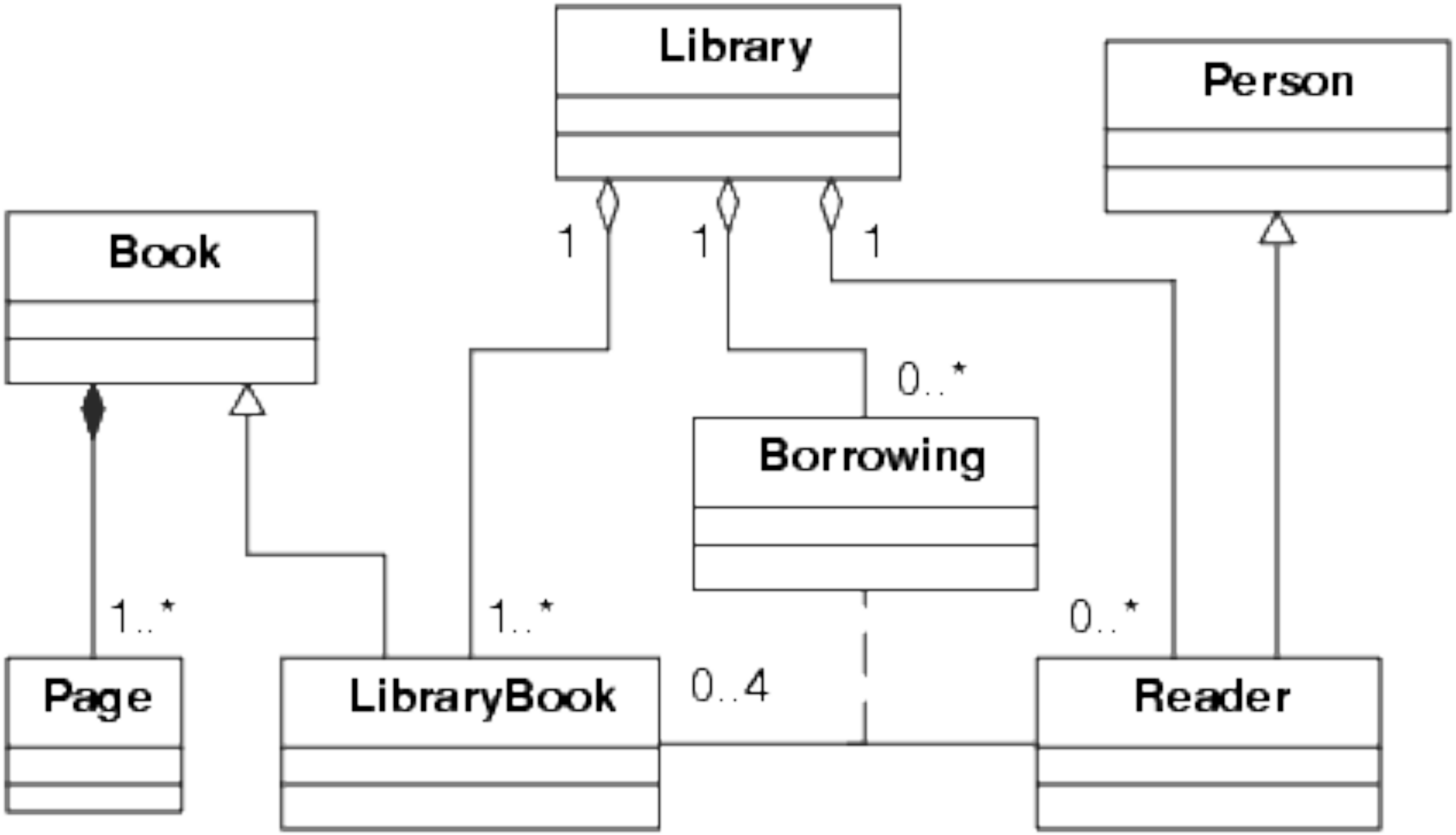
Relationships: Association (cont.)

A composition specifies a stronger whole-part relationship that indicates a strong (lifetime) ownership of parts by the whole. They live and die as a whole.

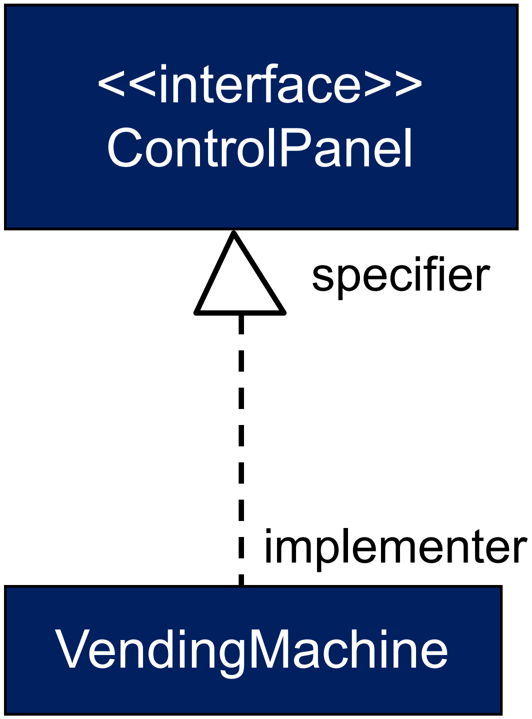
Compositions are denoted by a filled diamond on the association.



Relationships: Association (cont.)



Relationships: Realization



A realization relationship connects a class with an interface that supplies its behavioral specification.

It is rendered by a dashed line with a hollow triangle towards the specifier.

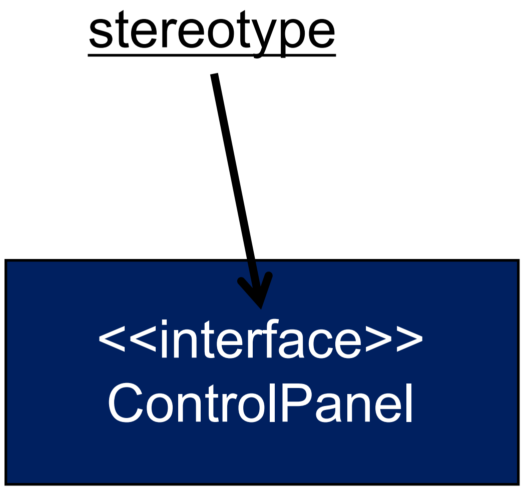
Class Diagrams— Interface and Stereotypes

Week 4: System Modeling, Part 1

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

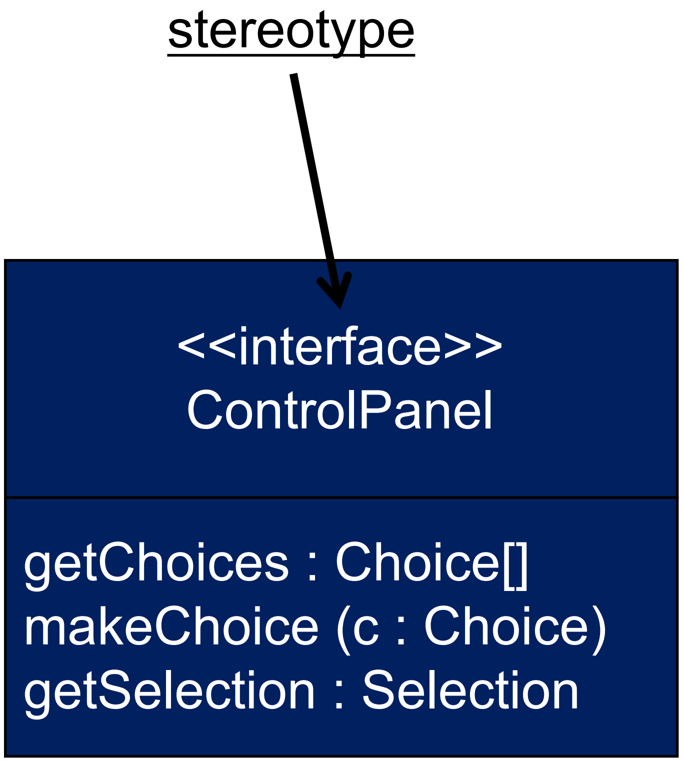
Interfaces



An interface is a named set of operations that specifies the behavior of objects without showing their inner structure.

It can be rendered in the model by a one- or two-compartment rectangle, with the stereotype <<interface>> above the interface name.

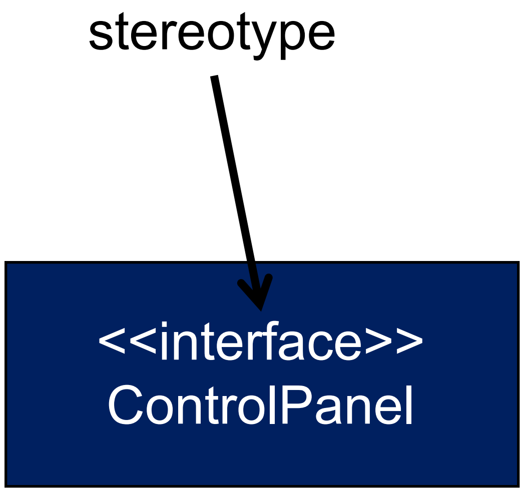
Interfaces (Two Rectangles)



An interface is a named set of operations that specifies the behavior of objects without showing their inner structure.

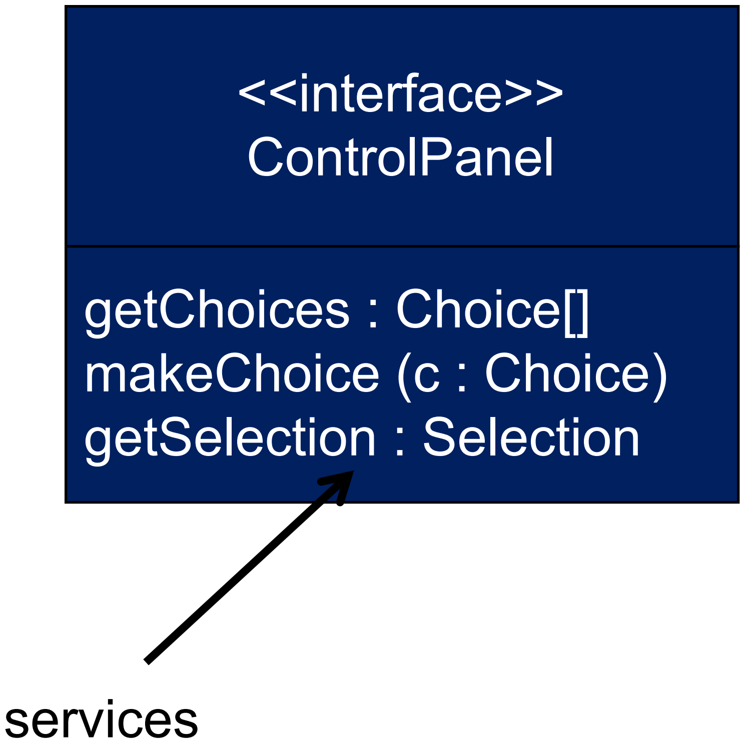
It can be rendered in the model by a one- or two-compartment rectangle, with the stereotype <<interface>> above the interface name.

Stereotype



An extension of the vocabulary of the UML that allows you to create new kinds of building blocks derived from existing ones but specific to your problem.

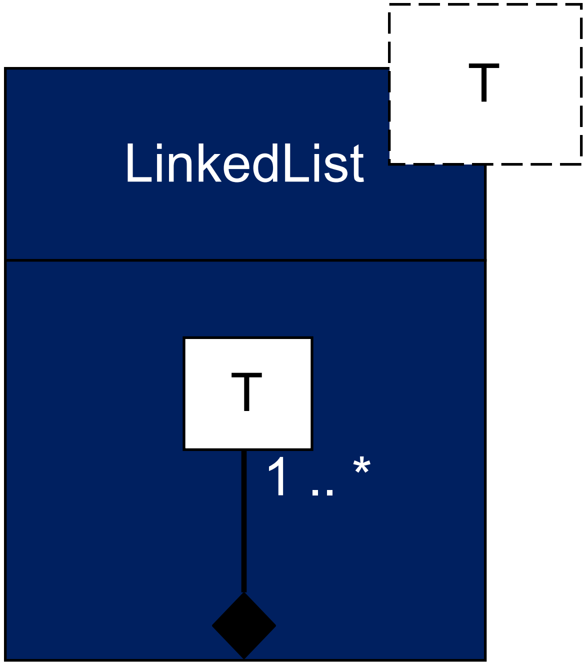
Interface Services



Interfaces do not get instantiated.

They have no attributes or state. Rather, they specify the services offered by a related class.

Parameterized Classes

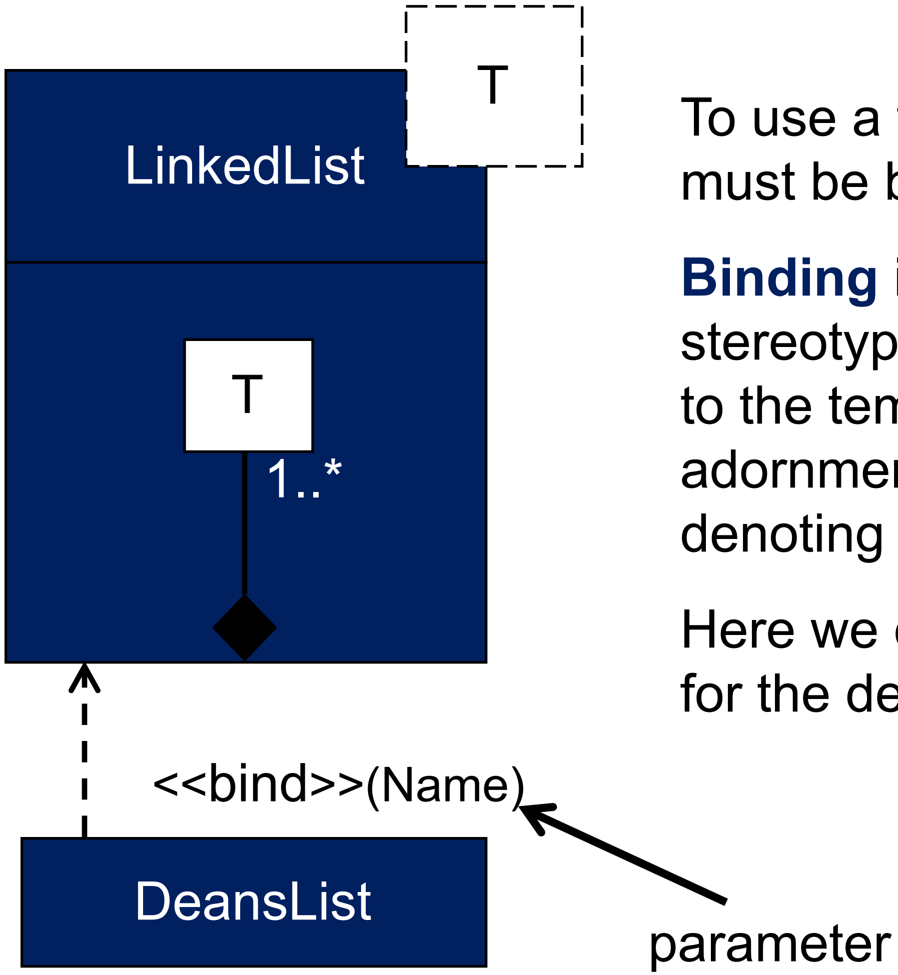


A parameterized class or template defines a family of potential elements.

A **template** is rendered by a small <u>dashed rectangle</u> superimposed on the upper-right corner of the class rectangle.

The dashed rectangle contains a list of formal parameters for the class.

Parameterized Classes

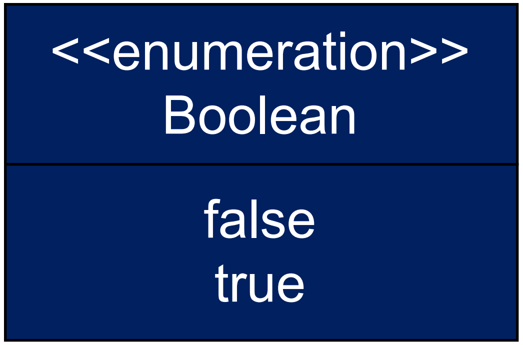


To use a template, the parameter must be bound.

Binding is done with the <<bind>> stereotype and a parameter to supply to the template. These are adornments to the dashed arrow denoting the dependency relationship.

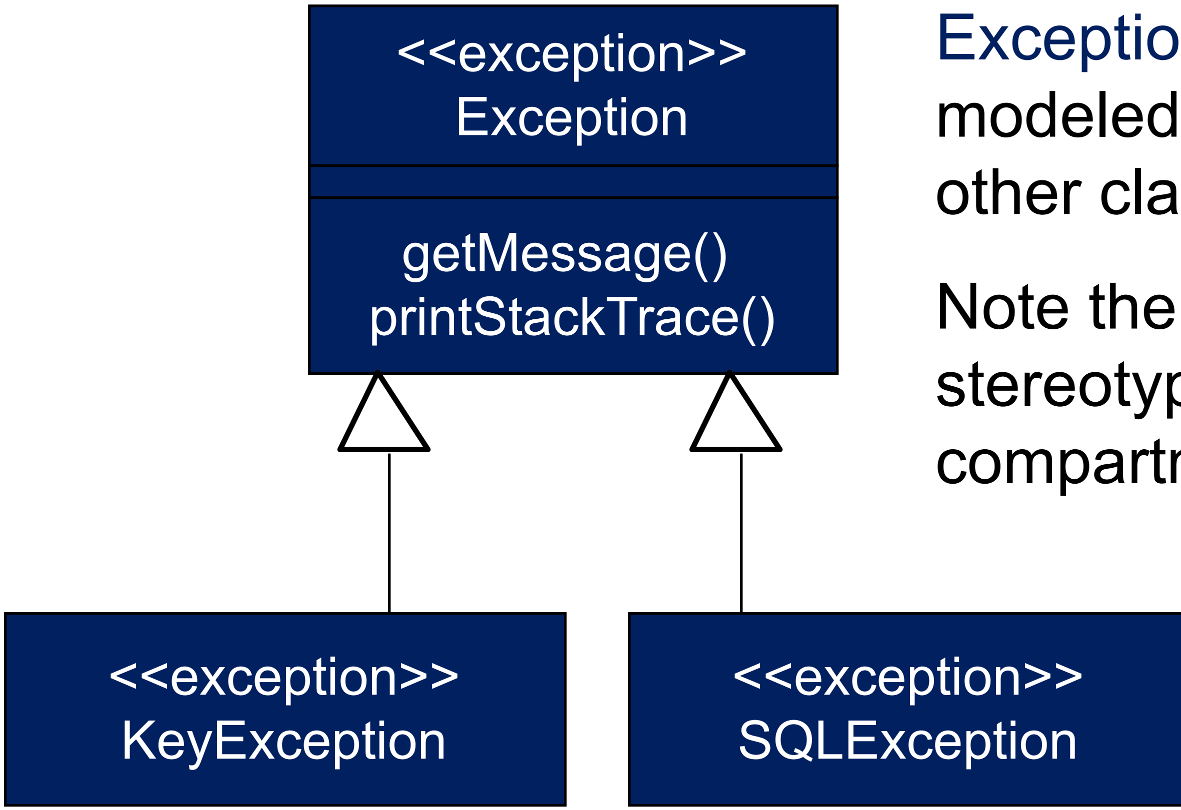
Here we create a linked-list of names for the dean's list.

Enumeration



An enumeration is a user-defined data type that consists of a name and an ordered list of enumeration literals.

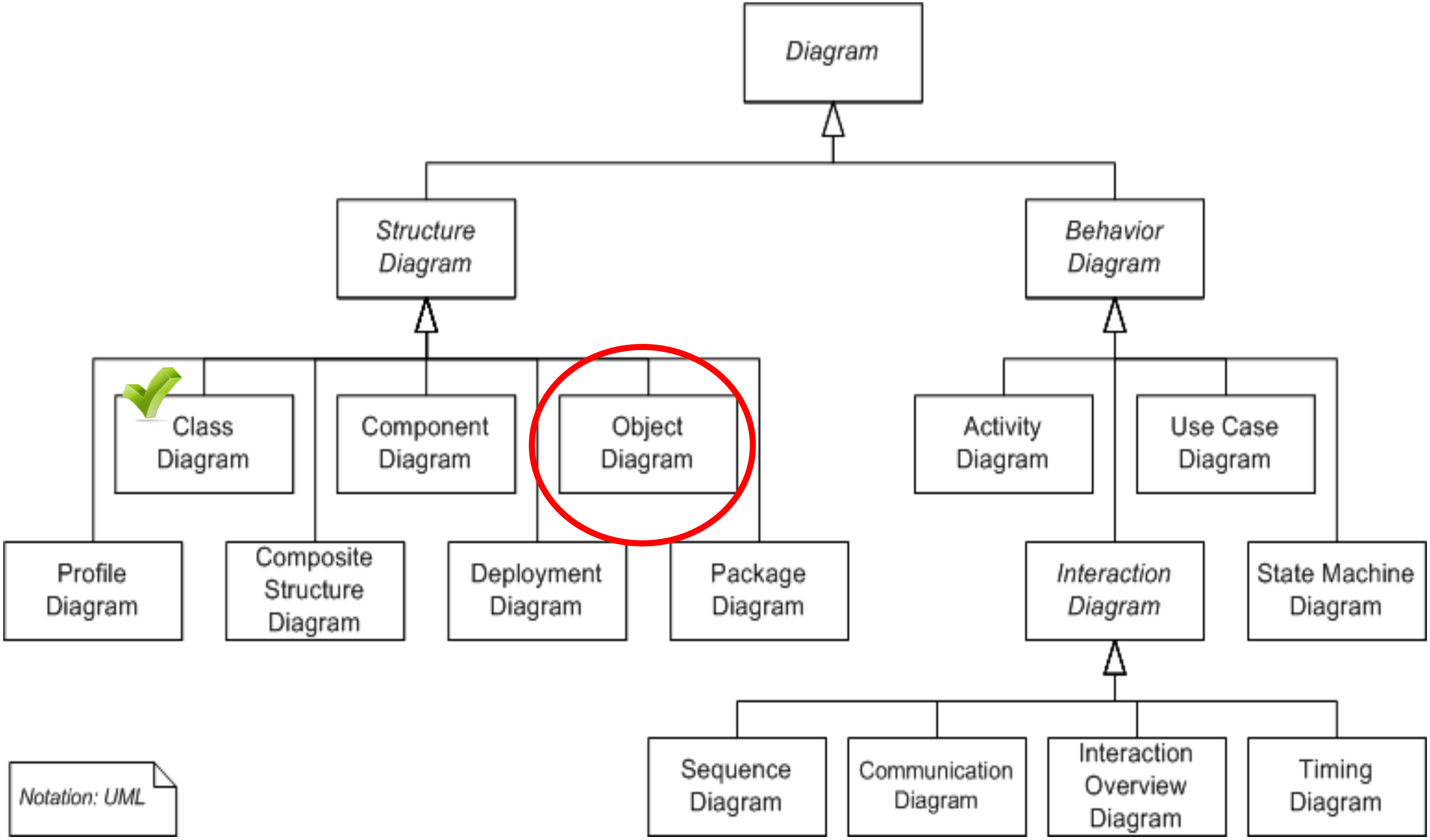
Exceptions



Exceptions can be modeled just like any other class.

Note the <<exception>> stereotype in the name compartment.

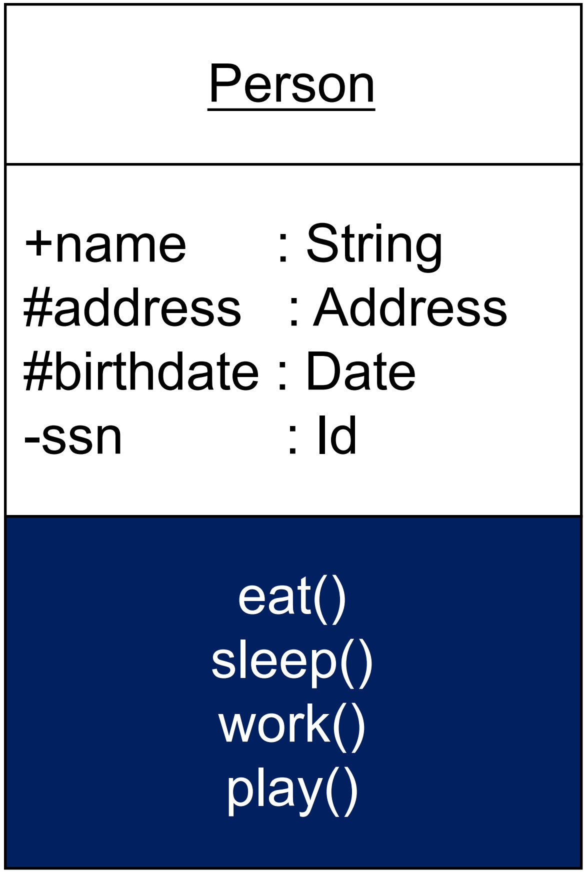
UML Diagrams



Object Diagram

- The object is represented in the same way as the class.
  - ❖ The only difference is the <u>name</u>, which is underlined as shown.

- An object is the actual implementation of a class, which is known as the instance of a class.
  - ❖ It has the same usage as the class.



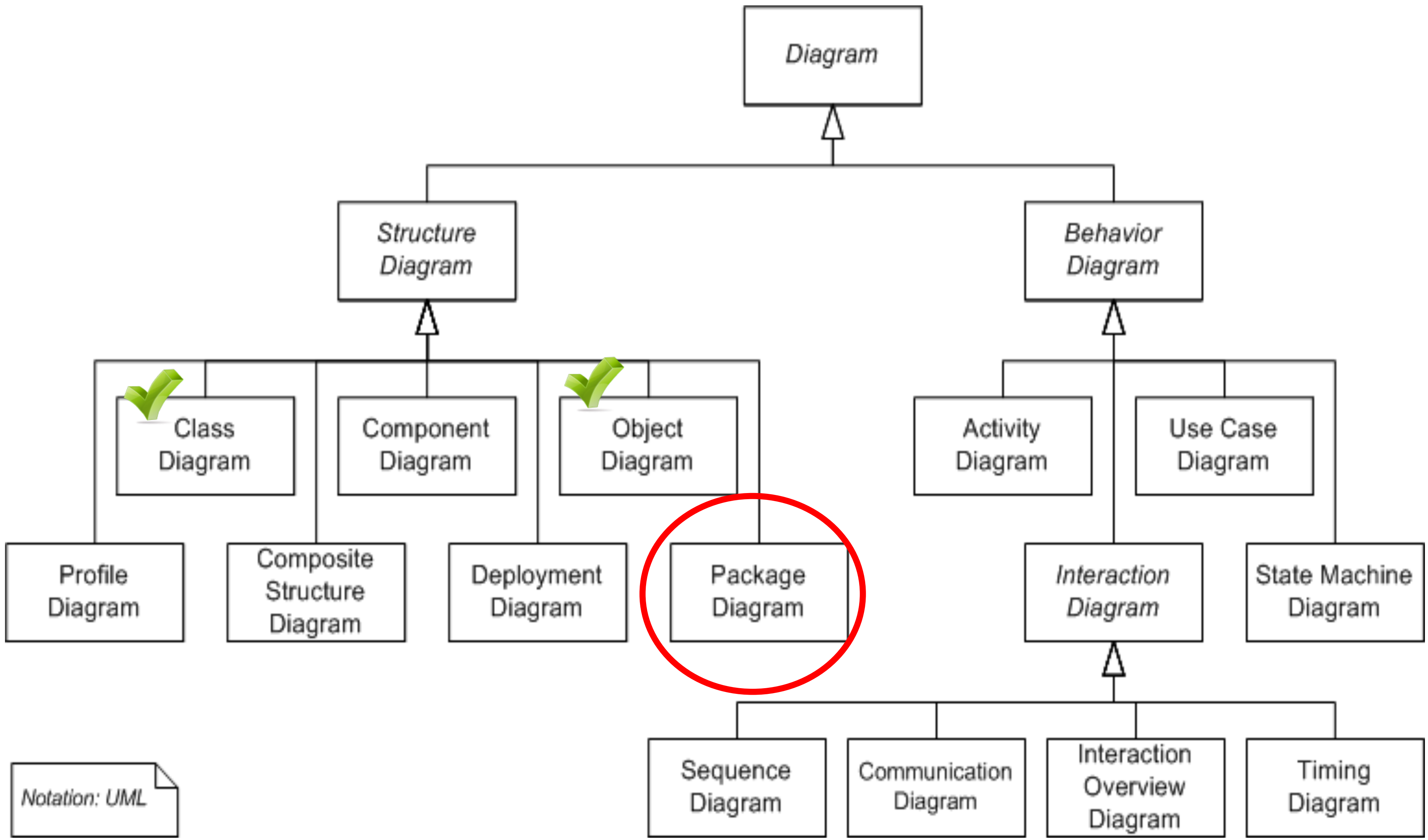
Package Diagrams

Week 4: System Modeling, Part 1

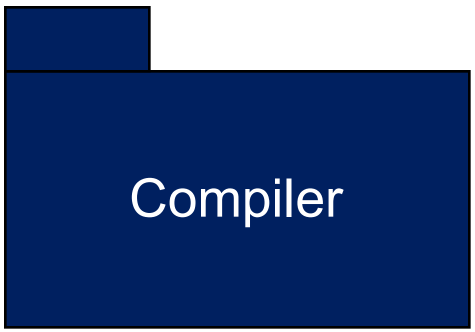
Edmund Yu, PhD
Associate Professor
esyu@syr.Edu

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

UML Diagrams



Packages



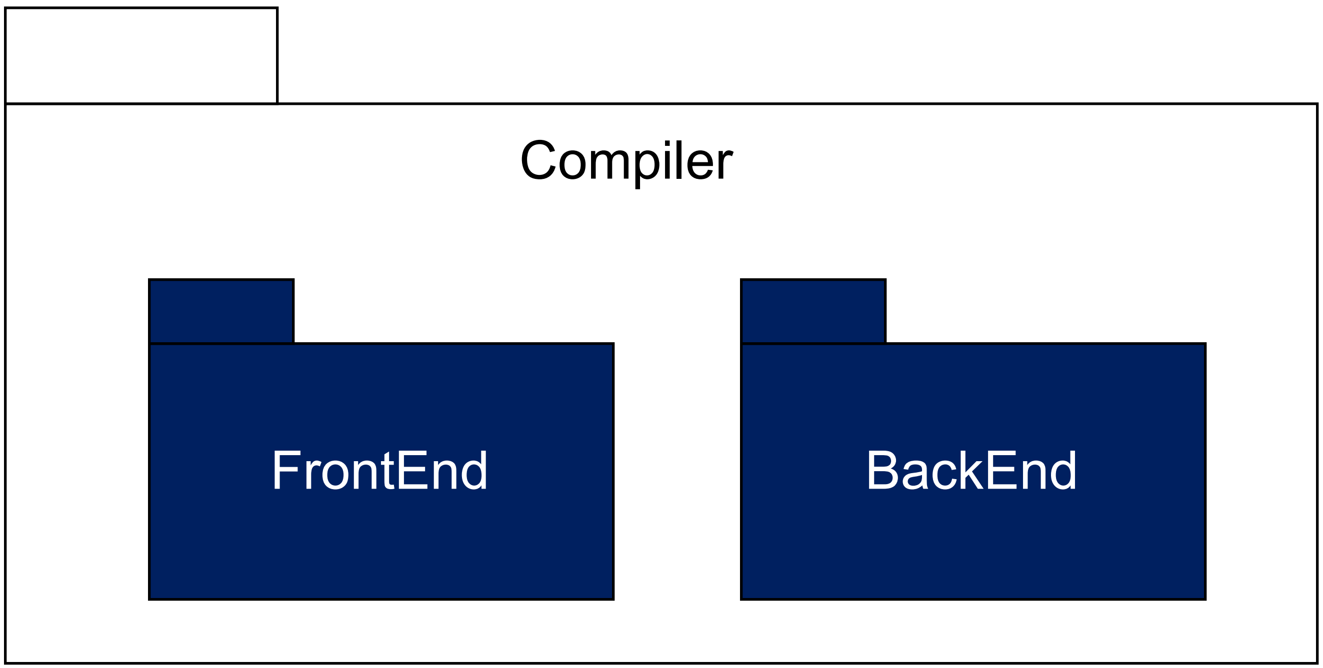
A package is a container-like element for organizing other elements into groups.

A package can contain classes and other packages and diagrams.

Packages can be used to provide controlled access between classes in different packages.

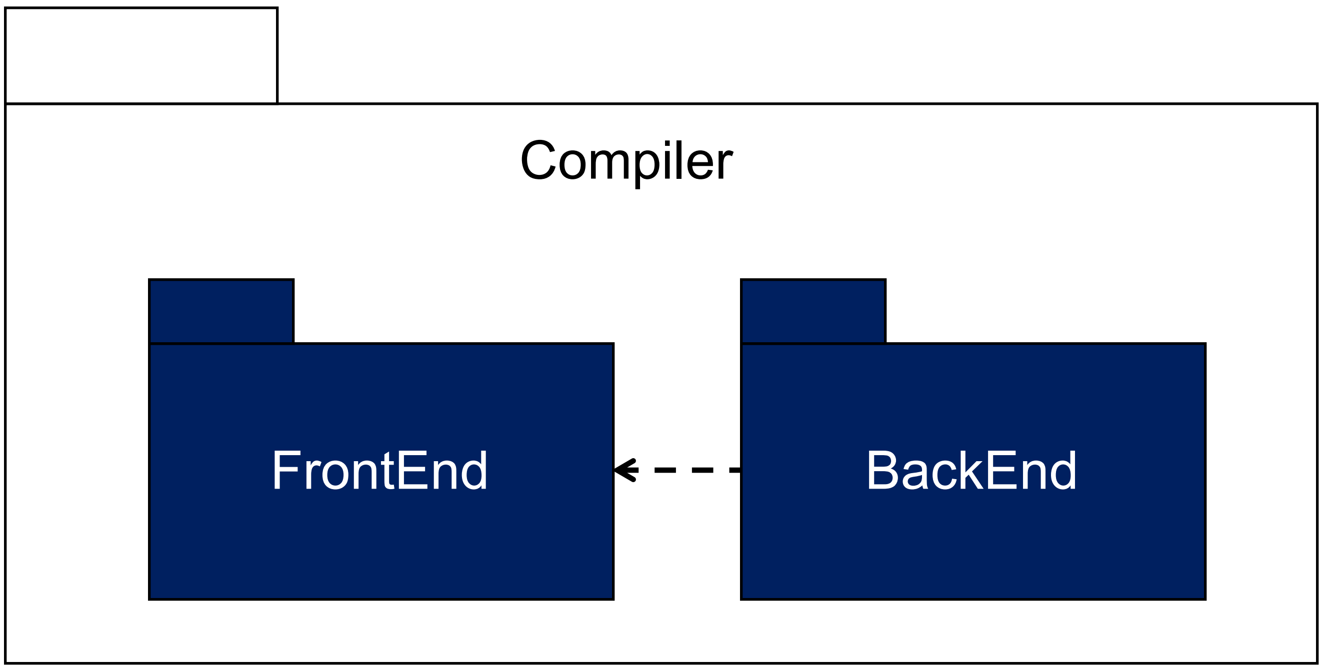
Packages

Classes in the FrontEnd package and classes in the BackEnd package cannot access each other in this diagram.

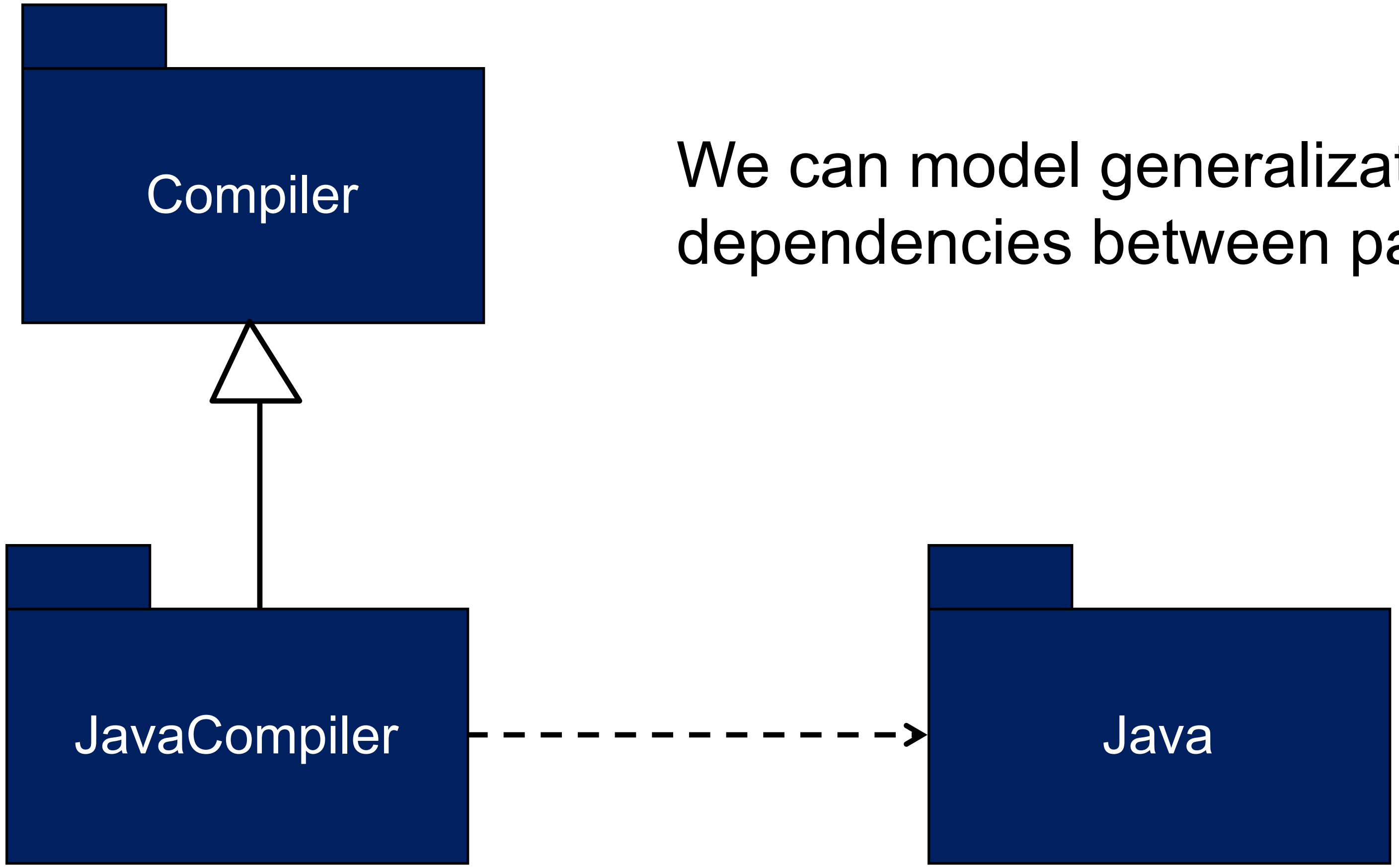


Packages

Classes in the BackEnd package now have access to the classes in the FrontEnd package.

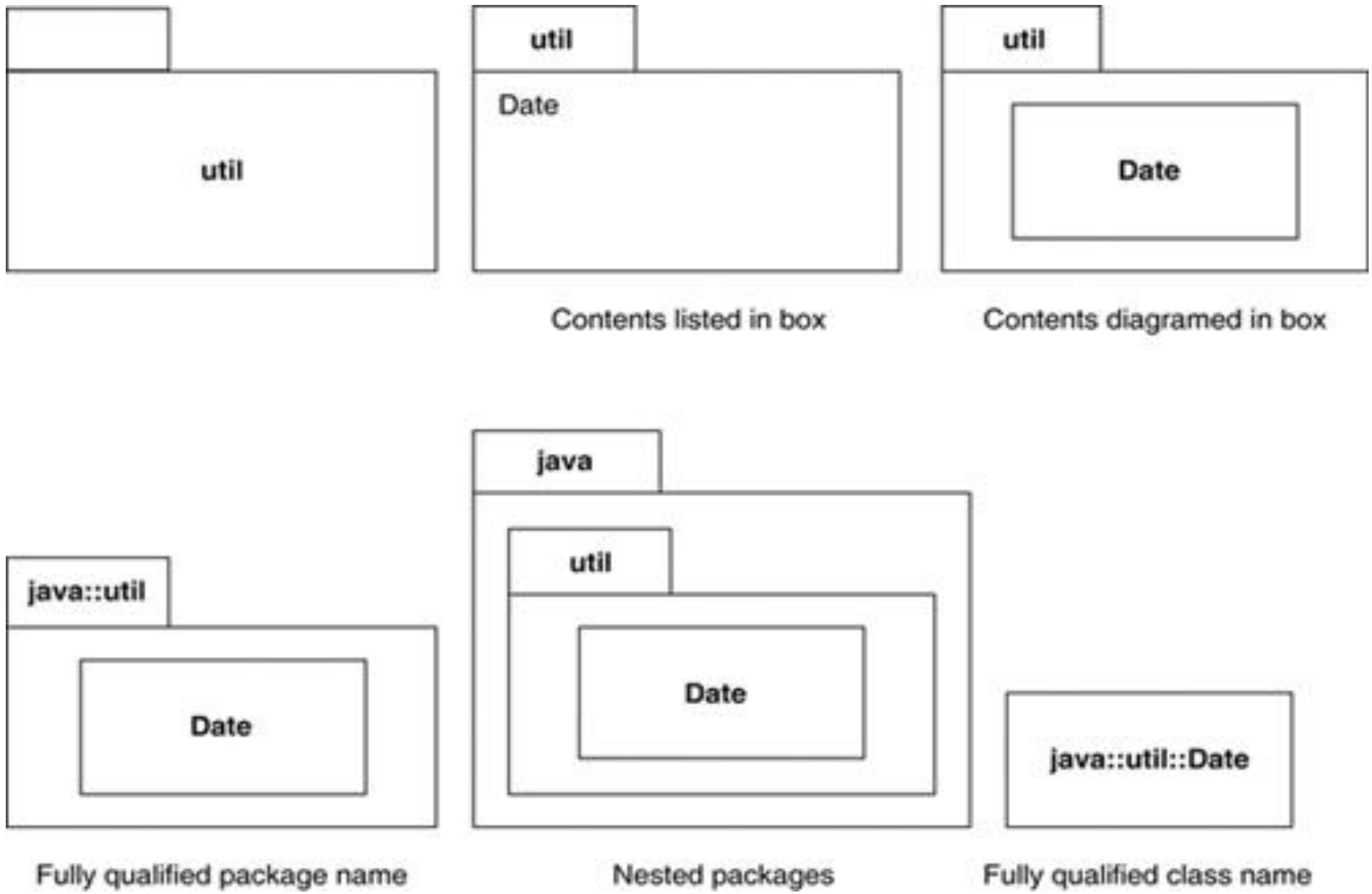


Packages

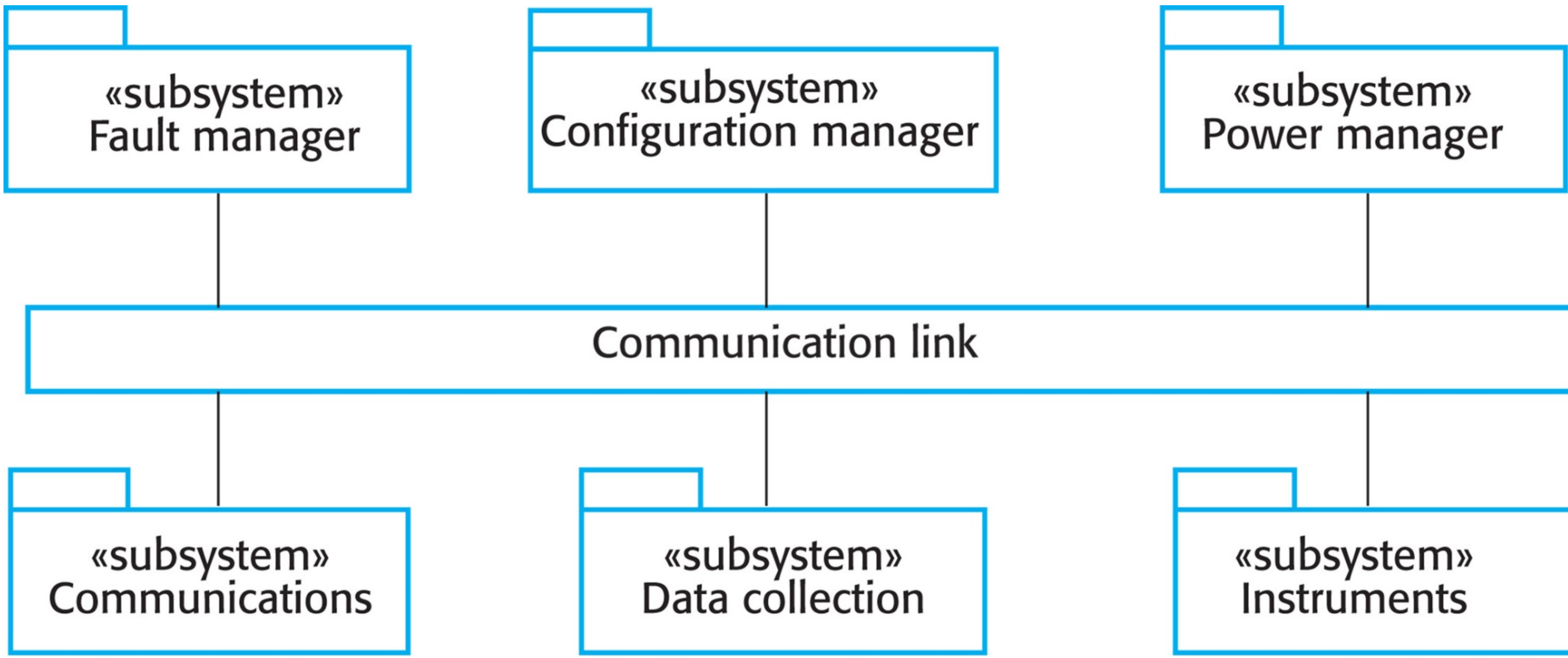


We can model generalizations and dependencies between packages.

Packages: java.util.Date



A Package Diagram (Weather Station)



A Package Diagram (Weather Station)