UML: Unified Modeling Language

Modeling

- Describing a system at a high level of abstraction
  - A model of the system
  - Used for requirements and specification
- Many notations over time
  - State machines
  - Entity-relationship diagrams
  - Dataflow diagrams

Recent History: 1980's

- The rise of object-oriented programming
- New class of OO modeling languages
- By early '90's, over fifty OO modeling languages

Recent History: 1990's

- Three leading OO notations decide to combine
  - Grady Booch (BOOCH)
  - Jim Rumbaugh (OMT: Object Modeling Technique)
  - Ivar Jacobsen (OOSE: OO Soft. Eng)

- Why?
  - Natural evolution towards each other
  - Effort to set an industry standard

UML

- UML stands for

  Unified Modeling Language

- Design by committee
  - Many interest groups participating
  - Everyone wants their favorite approach to be "in"

UML

- UML stands for

  Unified Modeling Language

- Design by committee
  - Many interest groups participating
  - Everyone wants their favorite approach to be "in"

UML (Cont.)

- Resulting design is huge
  - Many features
  - Many loosely unrelated styles under one roof

- Could also be called

  Union of all Modeling Languages

Objectives of UML

- UML is a general purpose notation that is used to
  - visualize
  - specify
  - construct and
  - document

  the artifacts of a software system

This and Next Lectures

- We discuss
  - Use Case Diagrams — for functional models
  - Class Diagrams — for structural models
  - Object Diagrams — for structural models
  - Sequence Diagrams — for dynamic models
  - Activity Diagrams — for dynamic models
  - State Diagrams
- This is a subset of UML
  - But probably the most used subset

Development Process

- Requirements elicitation – High level capture of user/ system requirements
  - Use Case Diagram
- Identify major objects and relationships
  - Object and class diagrams
- Create scenarios of usage
  - Class, Sequence and Collaboration diagrams
- Generalize scenarios to describe behavior
  - Class, State and Activity Diagrams
- Refine and add implementation details
  - Component and Deployment Diagrams

Structural Diagrams

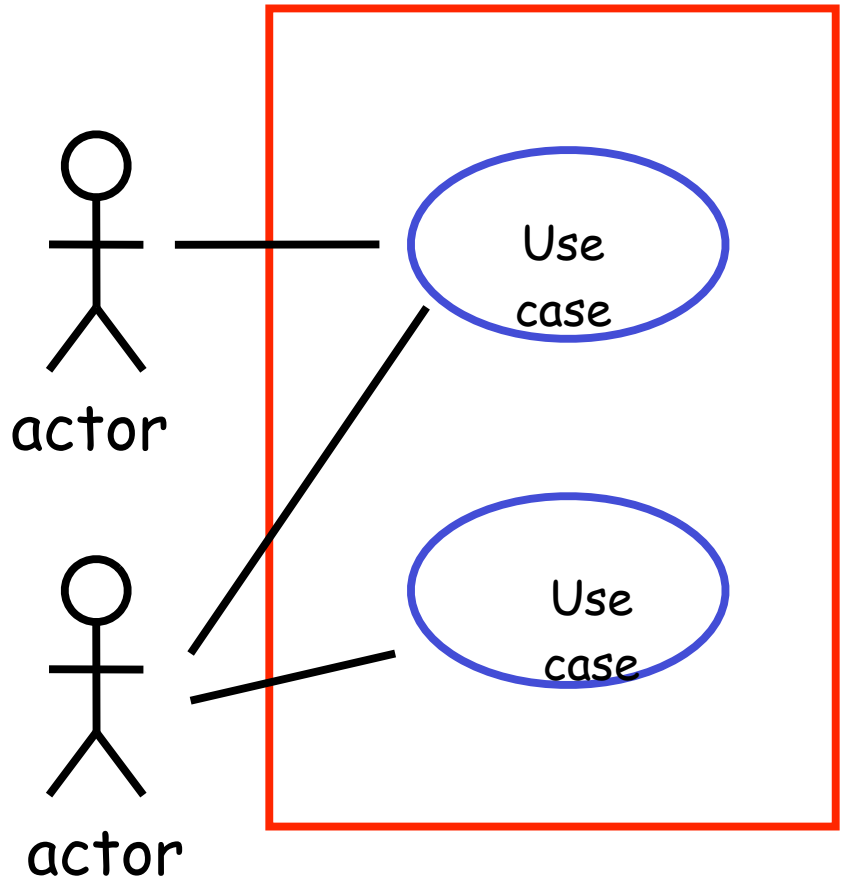
- **Class Diagram** – set of classes and their relationships. Describes interface to the class (set of operations describing services)
- **Object Diagram** – set of objects (class instances) and their relationships
- **Component Diagram** – logical groupings of elements and their relationships
- **Deployment Diagram** - set of computational resources (nodes) that host each component

Behavioral Diagram

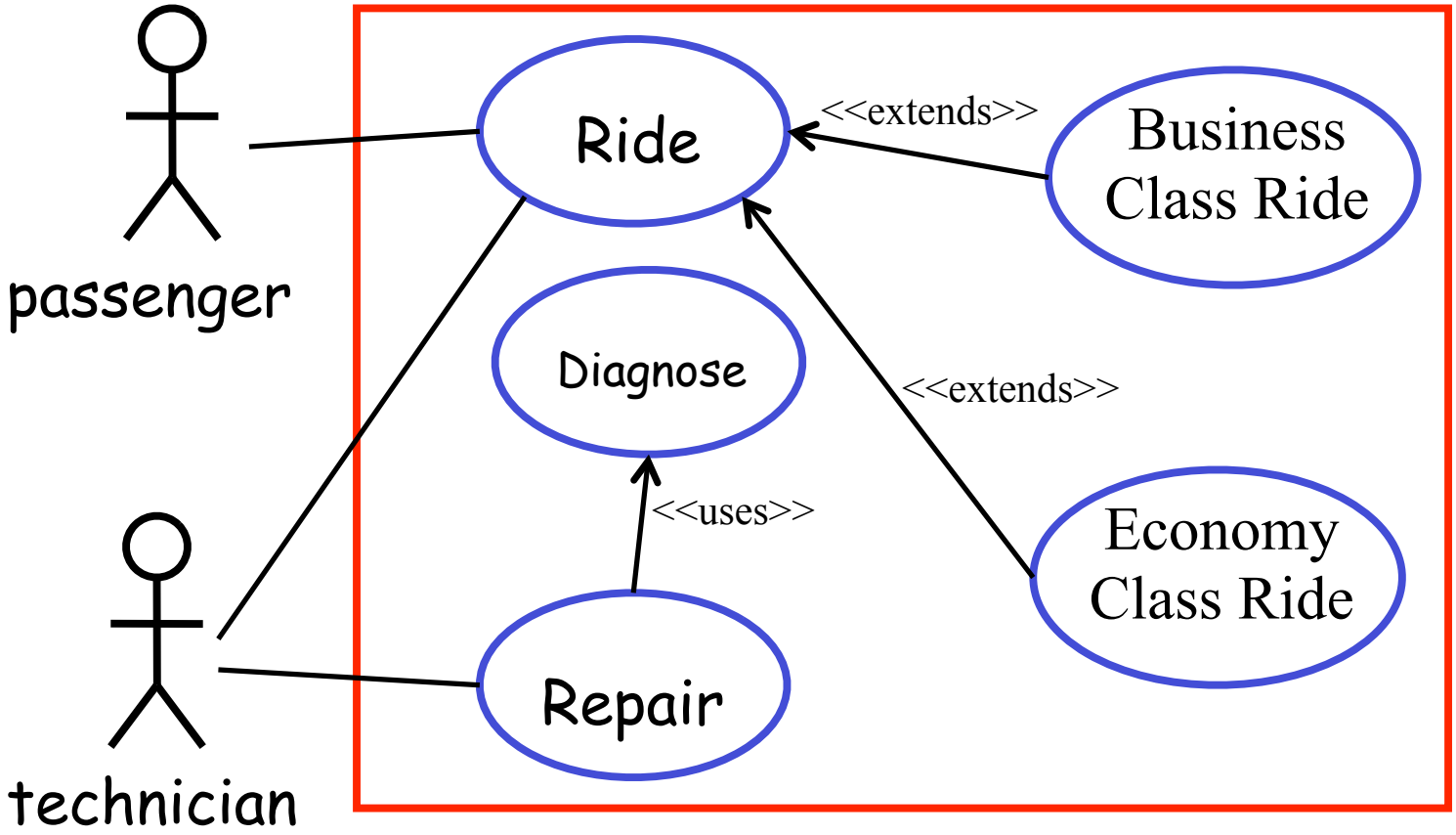
- **Use Case Diagram** – high-level behaviors of the system, user goals, external entities: actors
- **Sequence Diagram** – focus on time ordering of messages
- **Collaboration Diagram** – focus on structural organization of objects and messages
- **State (Machine) Diagram** – event driven state changes of system
- **Activity Diagram** – flow of control between activities

Use Case Diagram

- Elements
  - Actors
  - Use cases
  - Relations
- Use case diagram shows relationship between actors and use cases



Use Case Diagram Example



Example:
Project and Resource Management System

- A resource manager manages resources
- A project manager manages projects
- A system administrator is responsible for administrative functions of the system
- A backup system houses backup data for the system

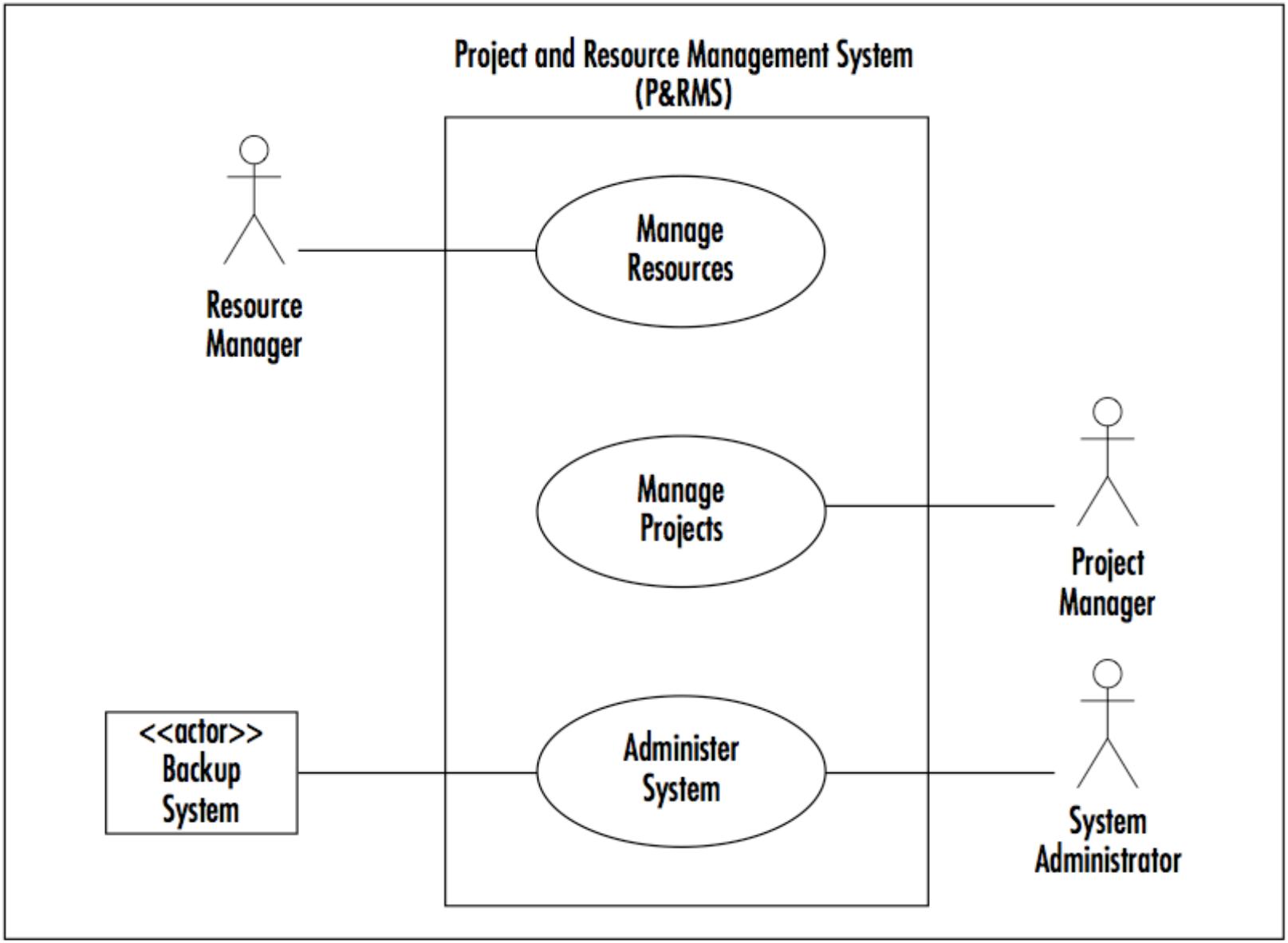


Figure 4-1: High-Level Use Case Diagram

Do these Use Cases Pass the Tests?

- Boss test?
- EBP test?
- Size test?

Manage Project Use Case

- A project manager can add, remove, and update a project
- Remove and update project requires to find project
- A project update may involve
  - Add, remove, or update activity
  - Add, remove, or update task
  - Assign resource to a task or unassign resource from a task

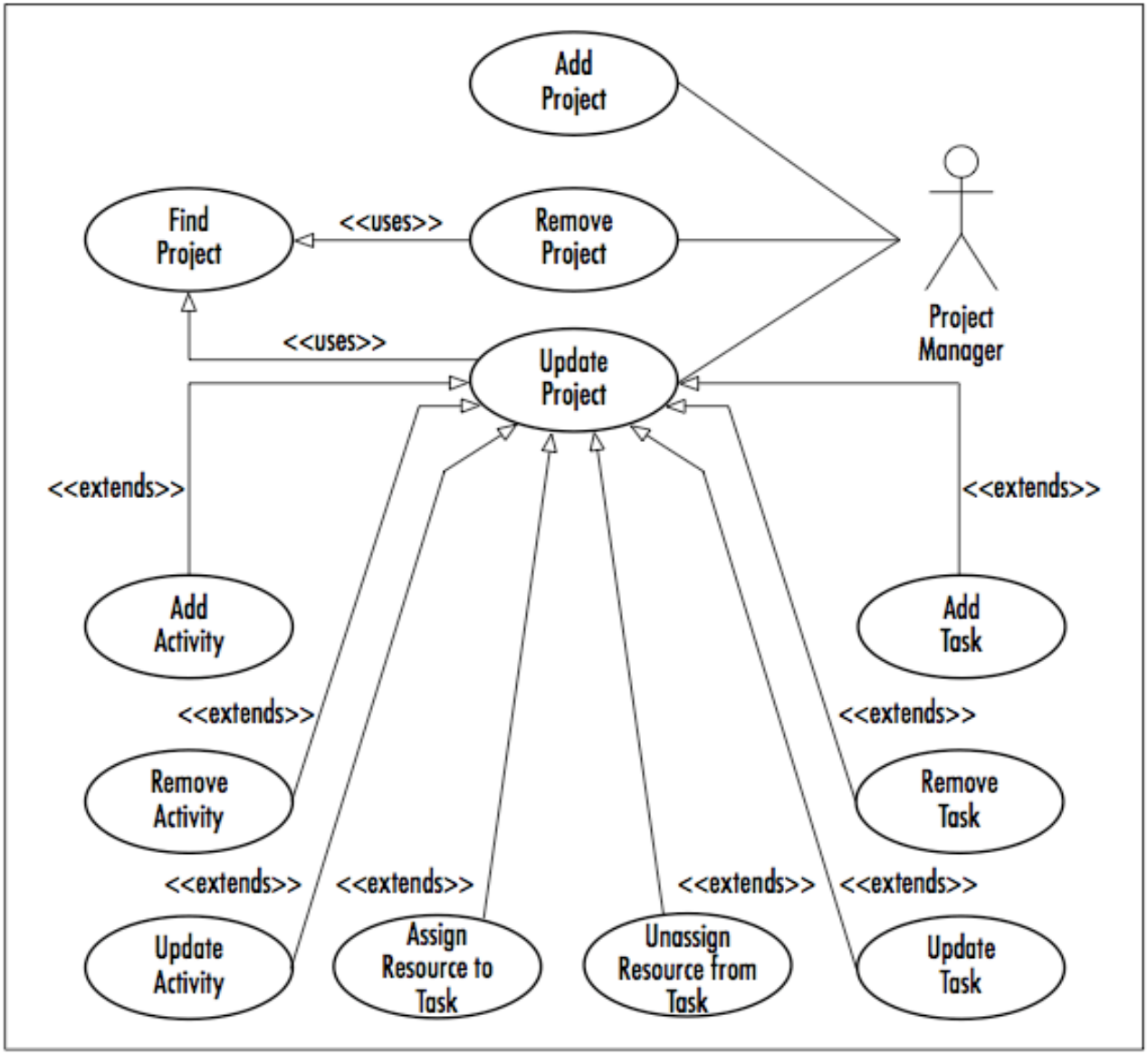
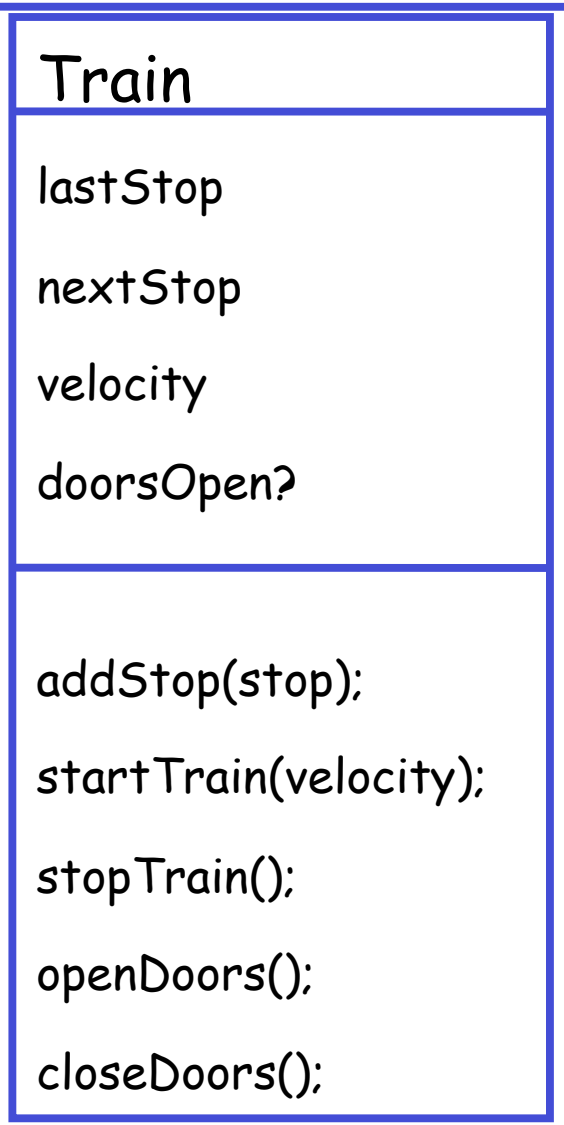


Figure 4-3: Manage Projects Use Case Diagram

Class Diagrams

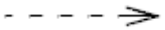
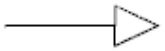
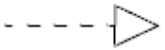
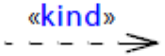
- Describe classes
  - In the OO sense
- Class diagrams are static -- they display what interacts but not what happens when they do interact
- Each box is a class
  - List fields
  - List methods



Class Diagrams: Relationships

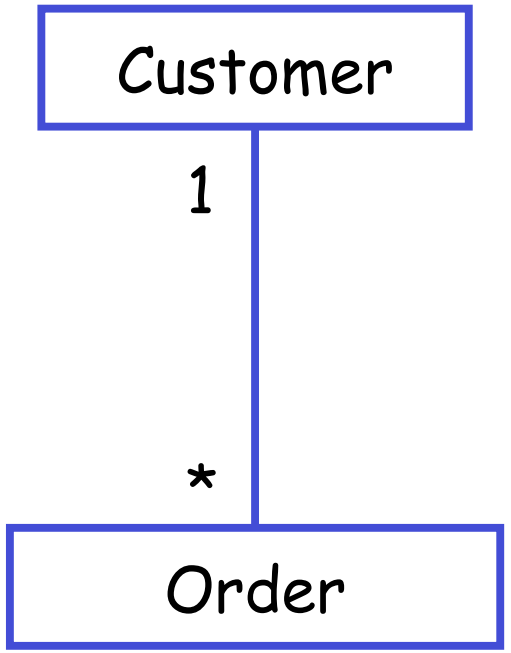
- Many different kinds of edges to show different relationships between classes
- Any examples?

Relationships in UML

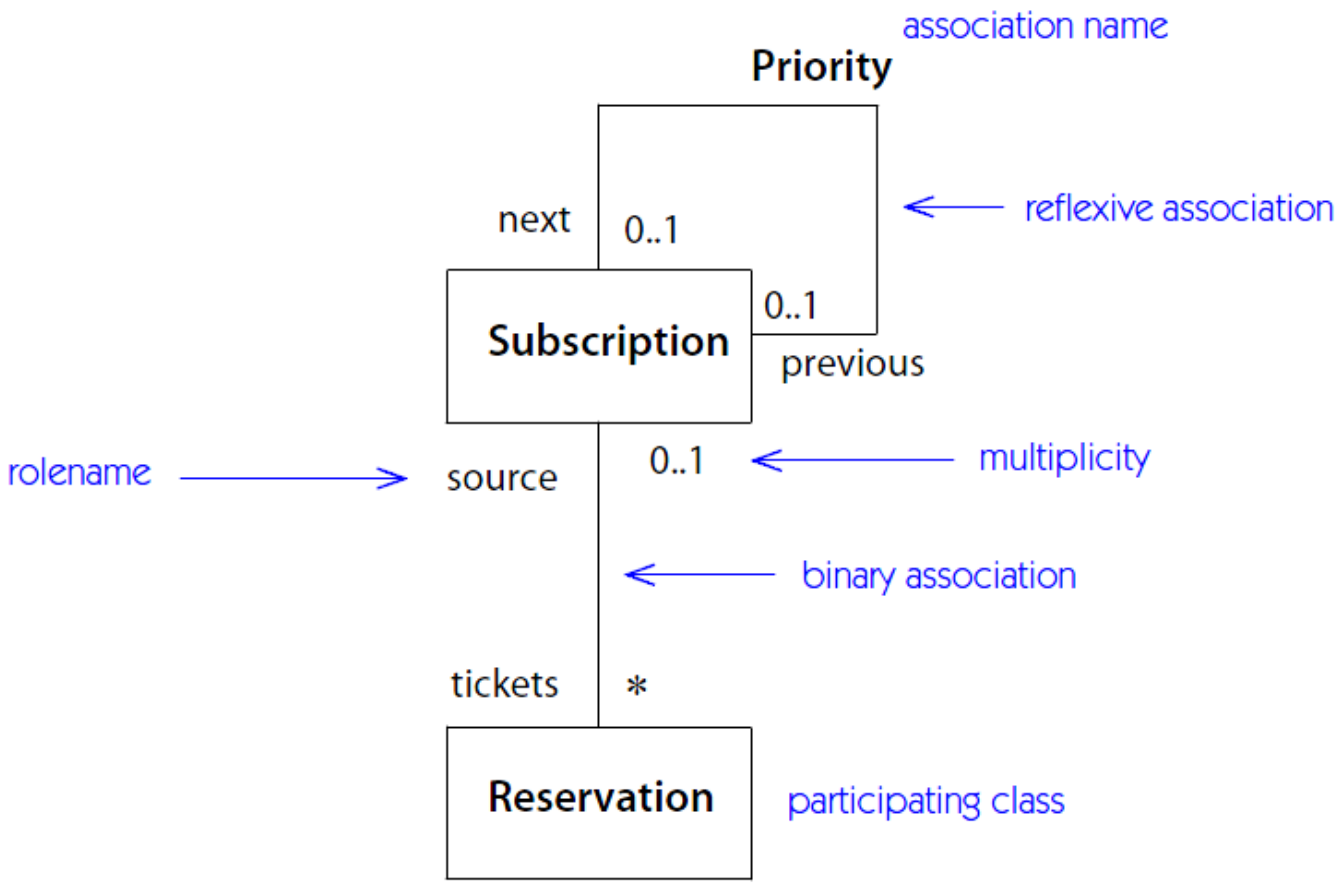
Relationship	*Function*	*Notation*
association	A description of a connection among instances of classes	
dependency	A relationship between two model elements	
generalization	A relationship between a more specific and a more general description, used for inheritance and polymorphic type declarations	
realization	Relationship between a specification and its implementation	
usage	A situation in which one element requires another for its correct functioning	

Association

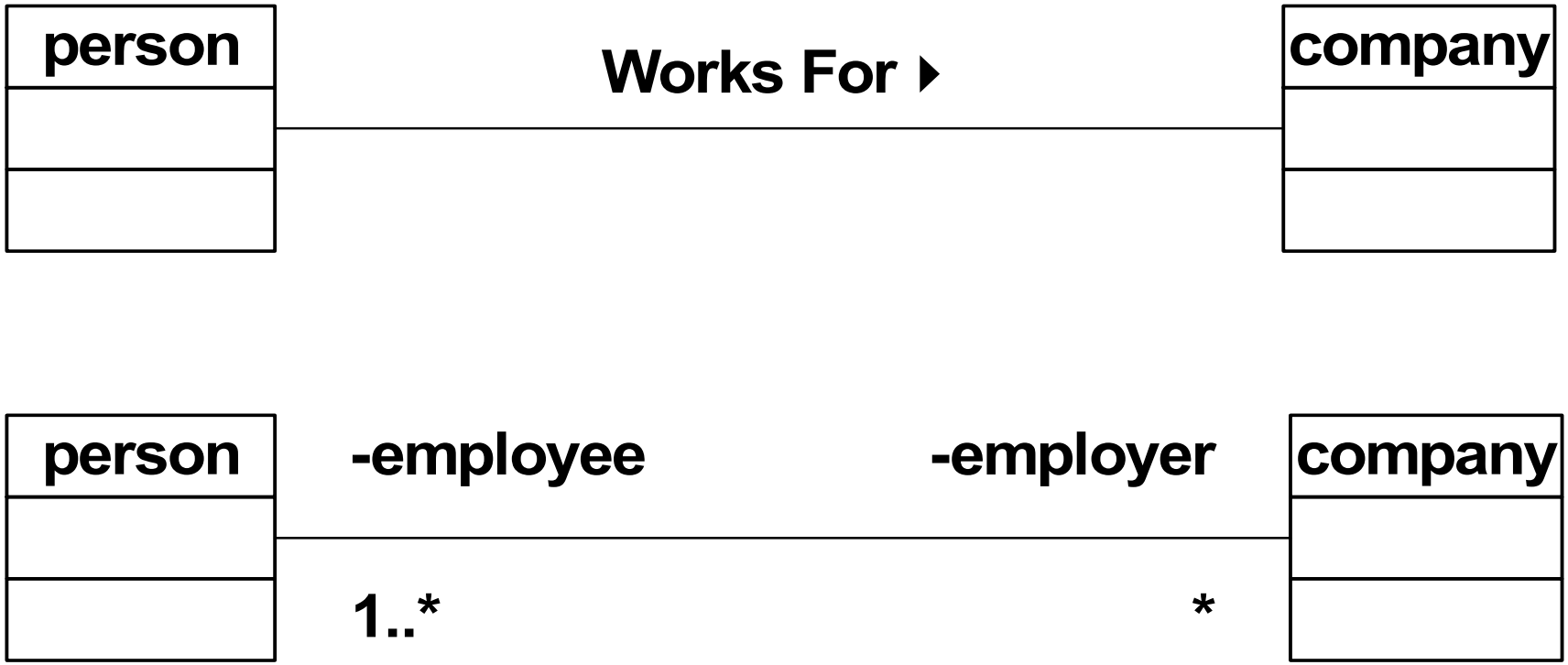
- Association between two classes
  - if an instance of one class must know about the other in order to perform its work.
- Label endpoints of edge with cardinalities
  - Use * for arbitrary
- Can be directional (use arrows in that case)



Association

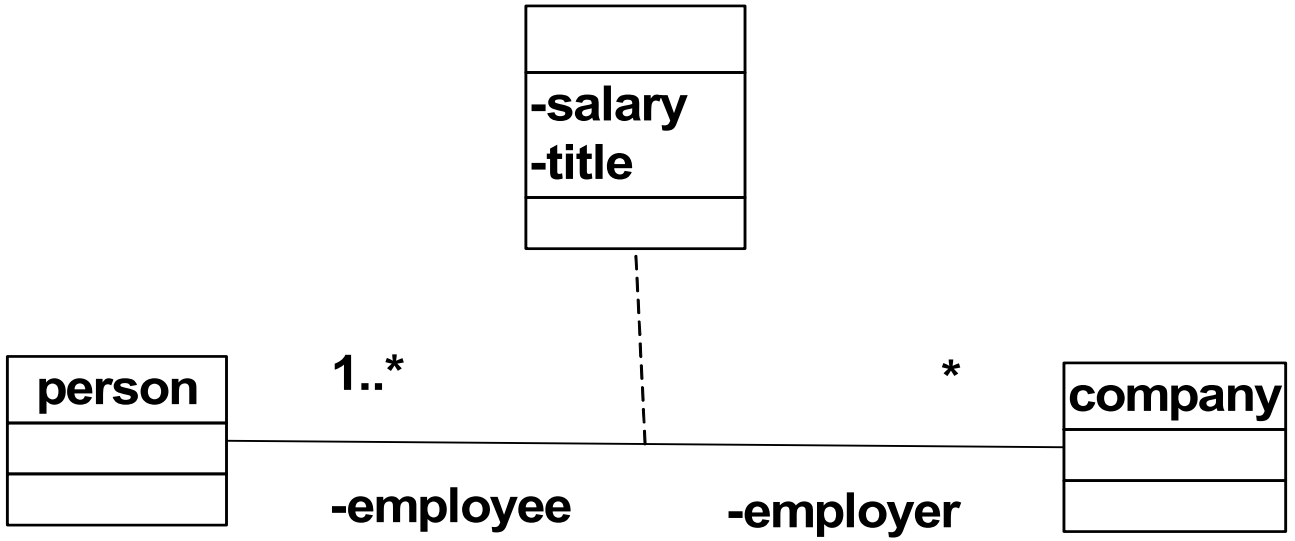


Examples of Association

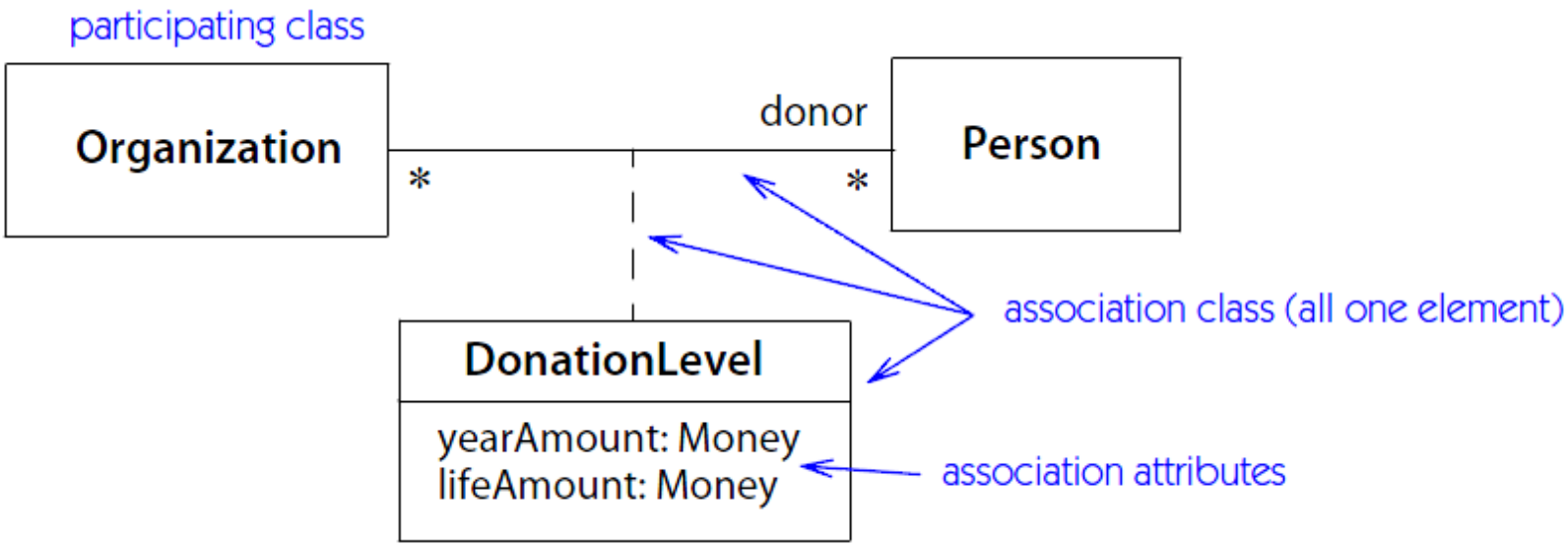


Link Attributes

- Associations may have properties in the same manner as objects/classes
- Salary and job title can be represented as

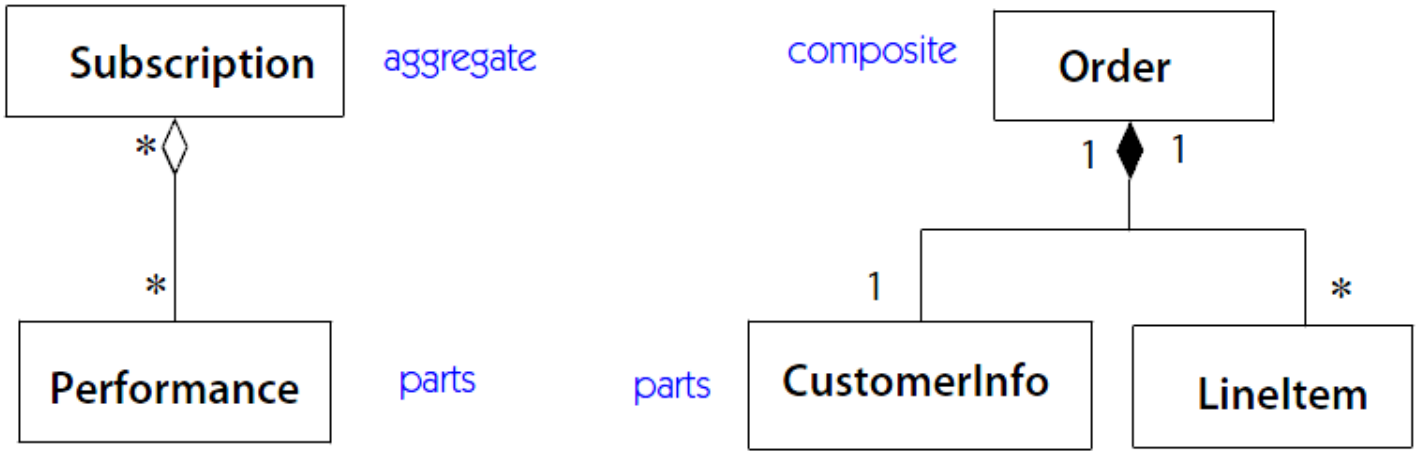


Association



Types of Association

Aggregation Composition

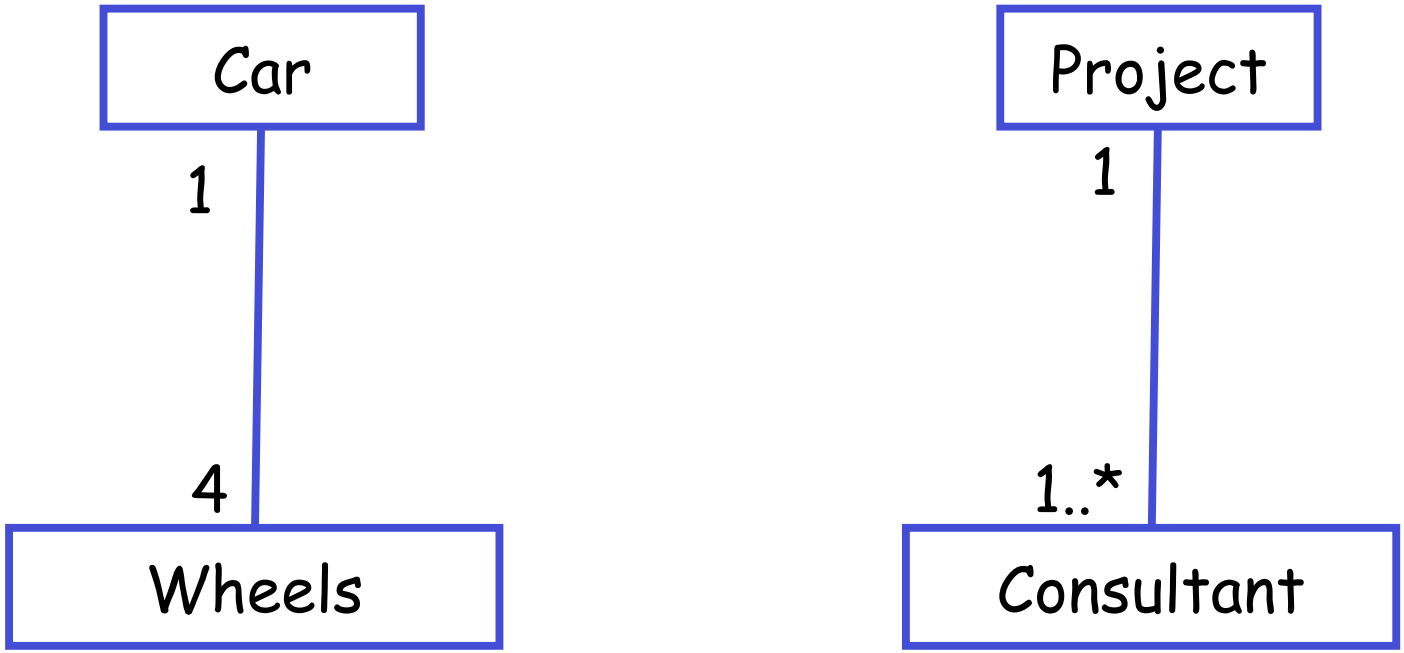


Aggregation

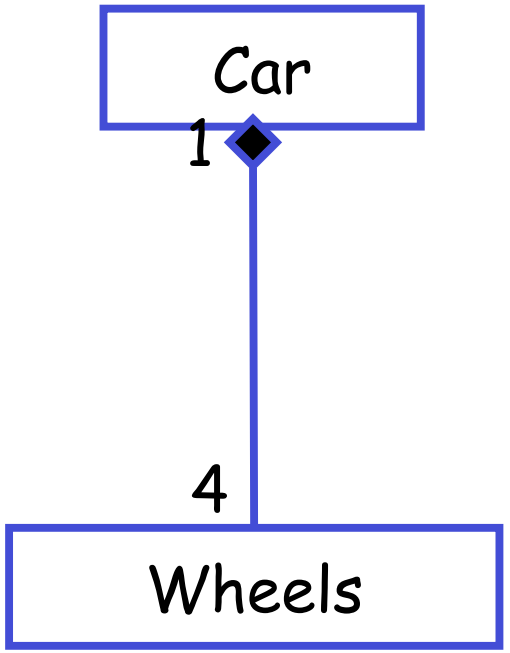
- An association in which one class belongs to a collection
  - Shared: An object can exist in more than one collections
  - No ownership implied
- Denoted by hollow diamond on the "contains" side

Composition

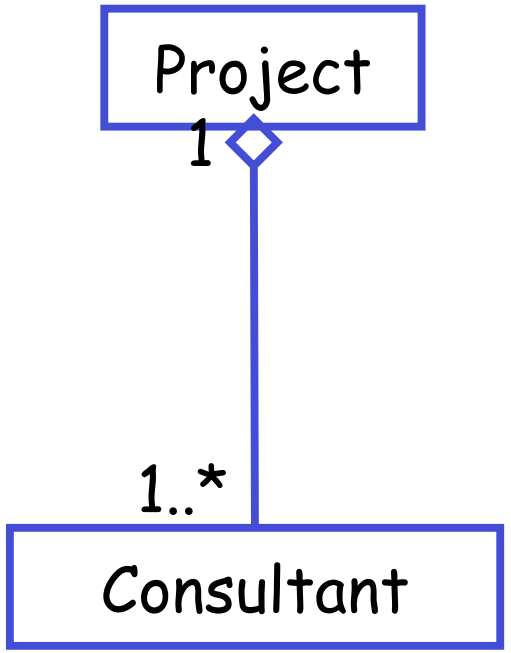
- An association in which one class belongs to a collection
  - No Sharing: An object cannot exist in more than one collections
  - Strong "has a" relationship
  - Ownership
- Denoted by filled diamond on the "contains" side

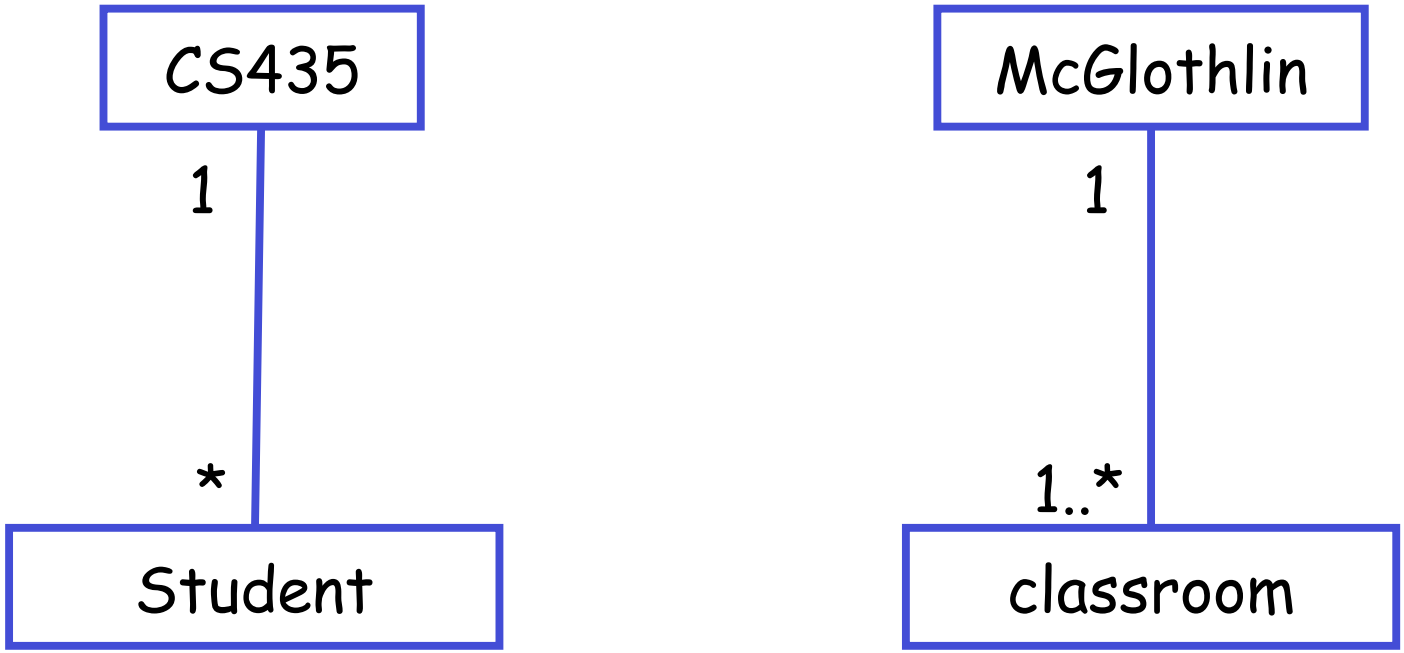

Car
1
4
Wheels
Project
1
1..*
Consultant

Composition

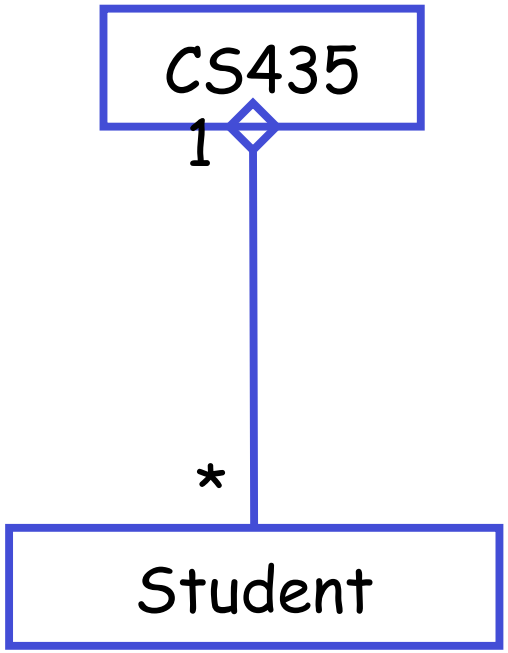


Aggregation

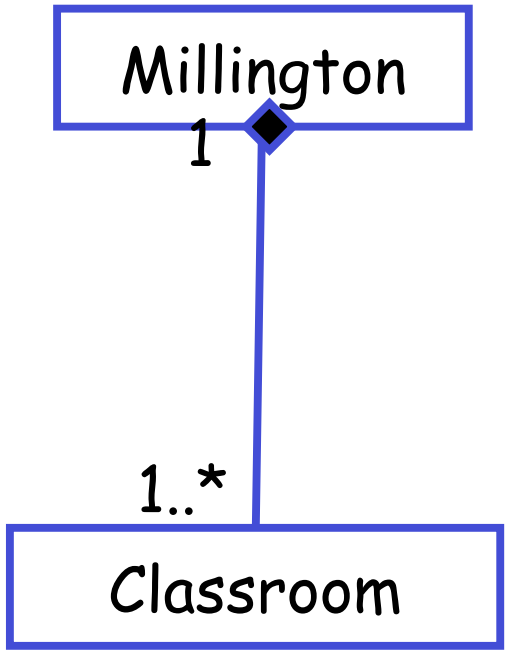



CS435
1
*
Student
McGlothlin
1
1..*
classroom

Aggregation

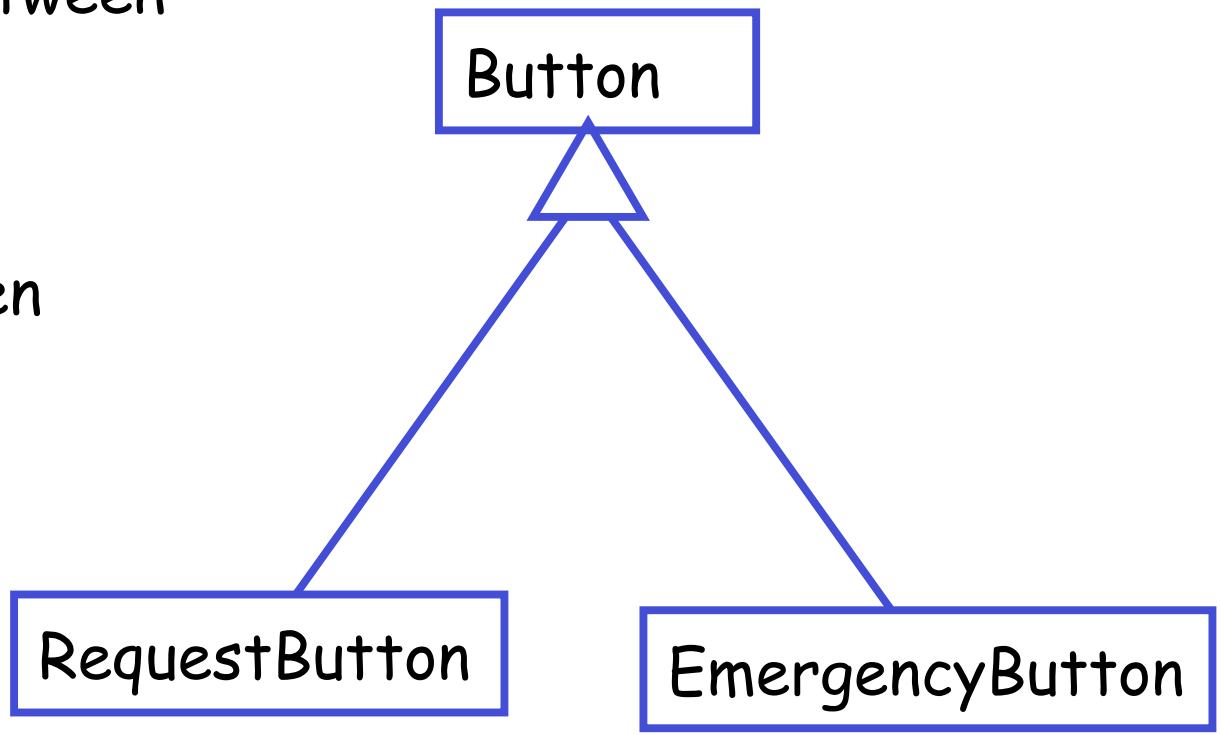


Composition

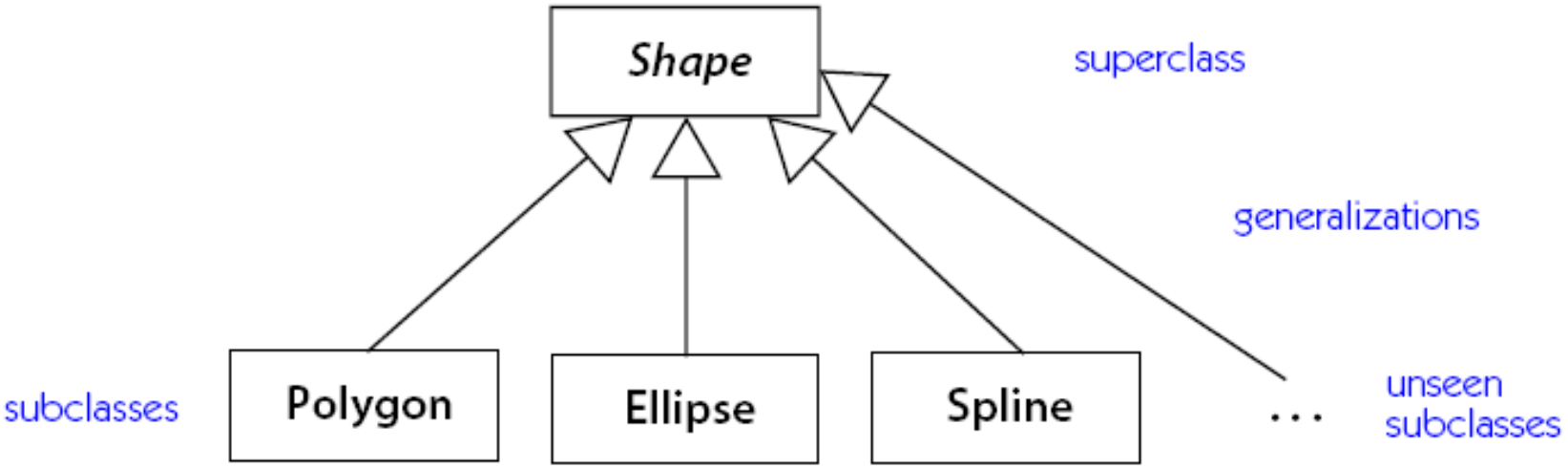


Generalization

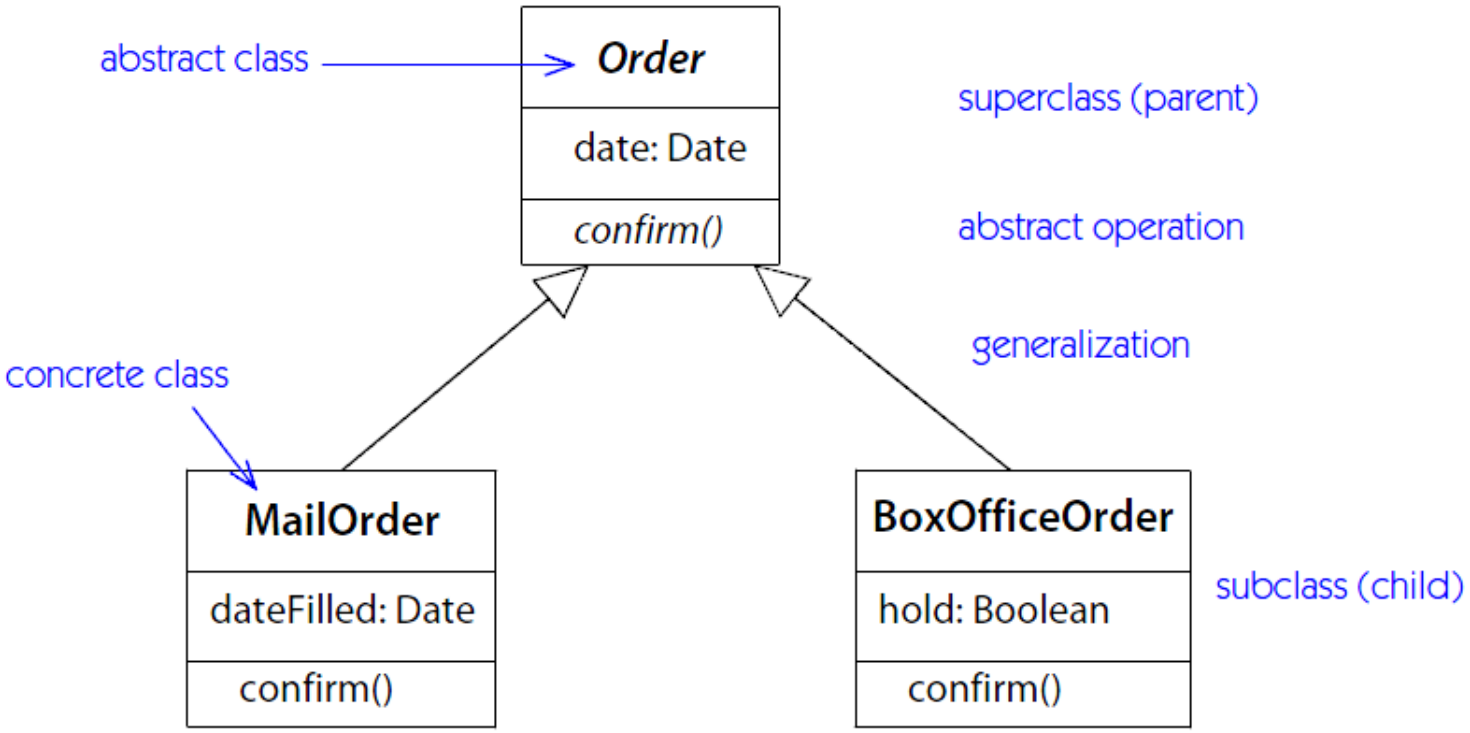
- Inheritance between classes
- Denoted by open triangle



Generalization

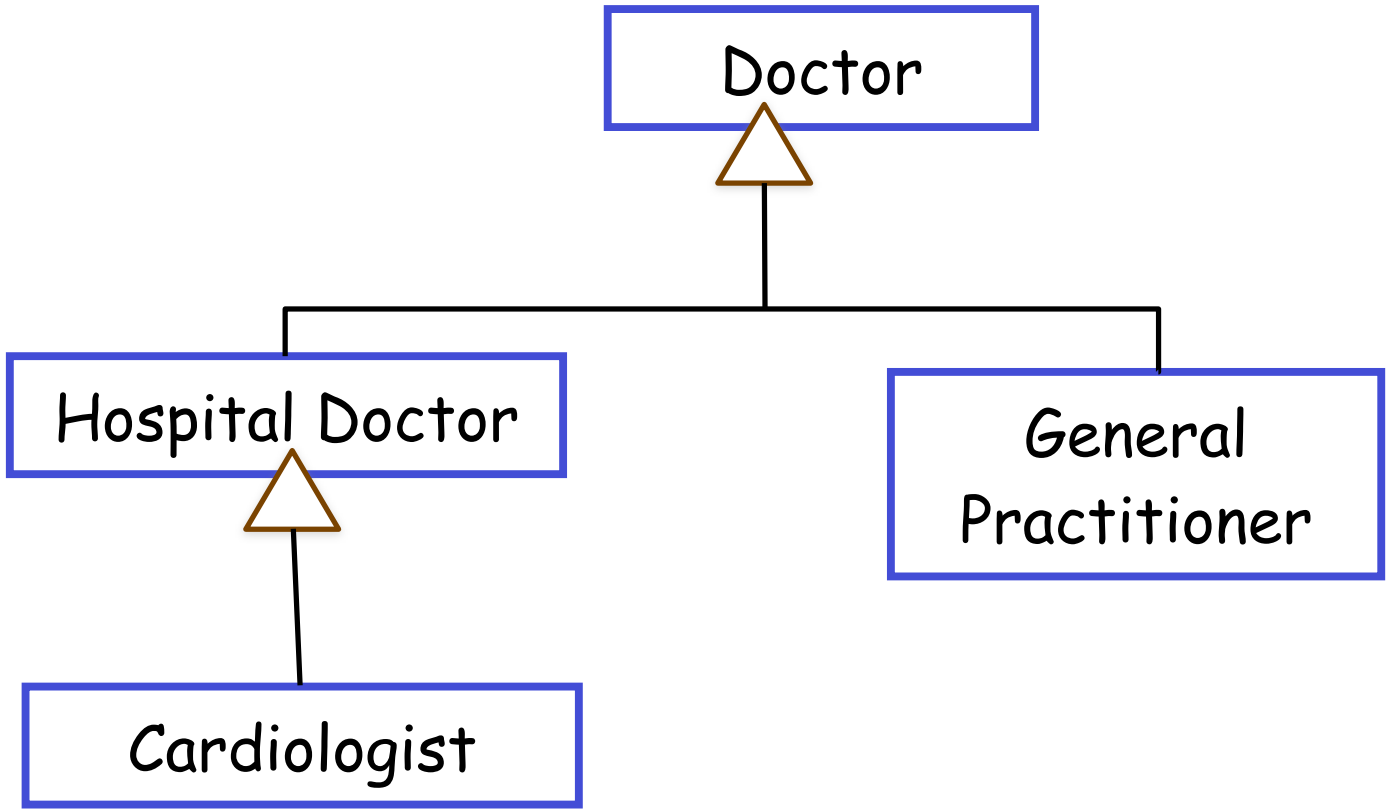


Generalization

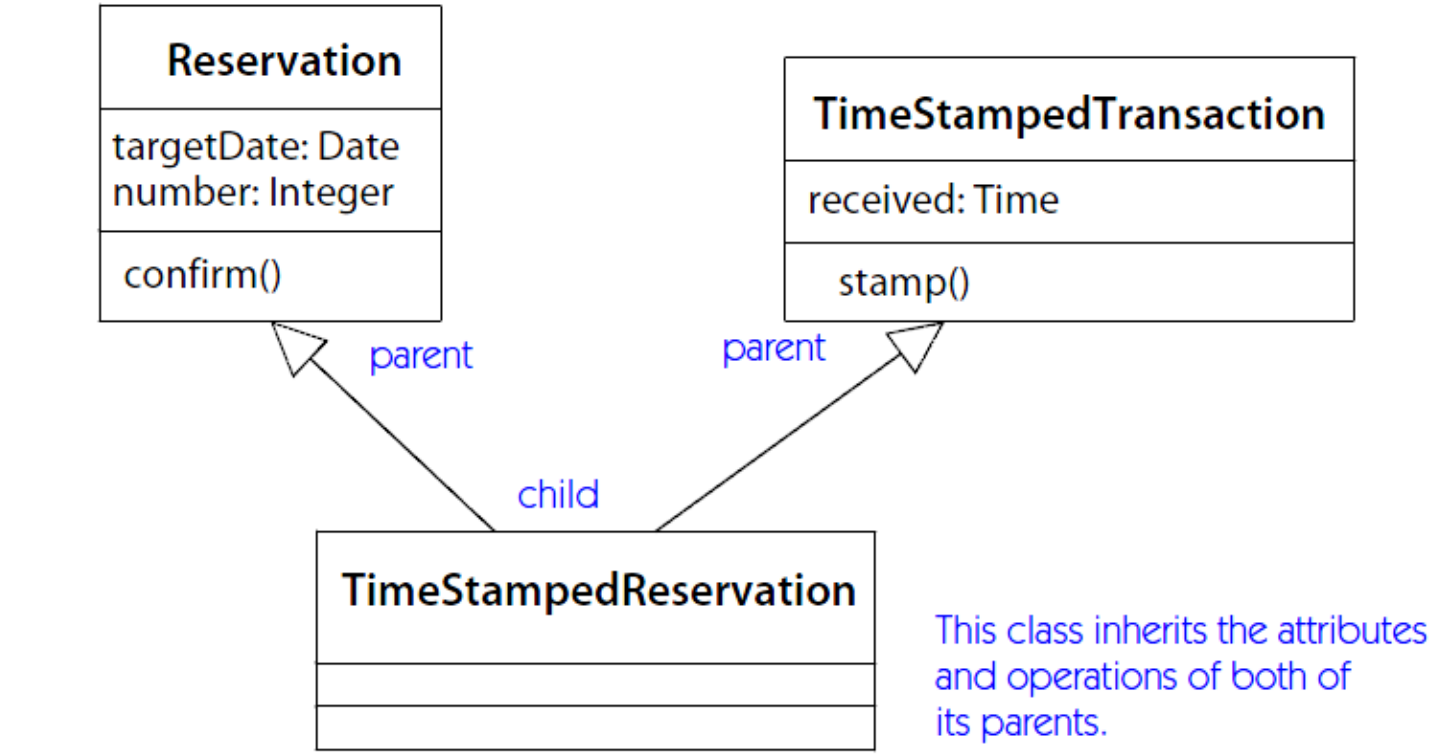


Generalization

- (Think subclassing)



Generalization

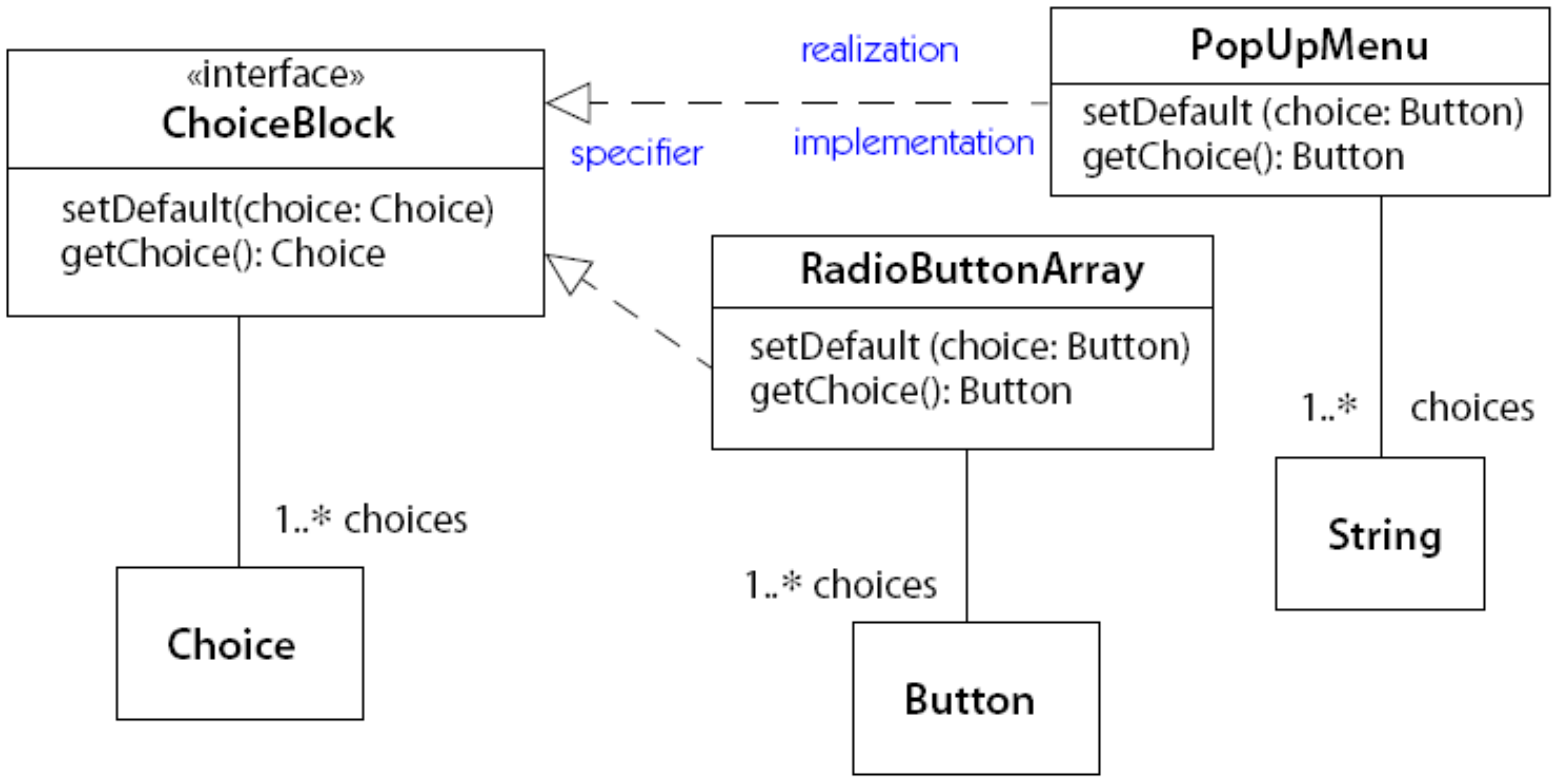


No new features are needed by the child.

Generalization

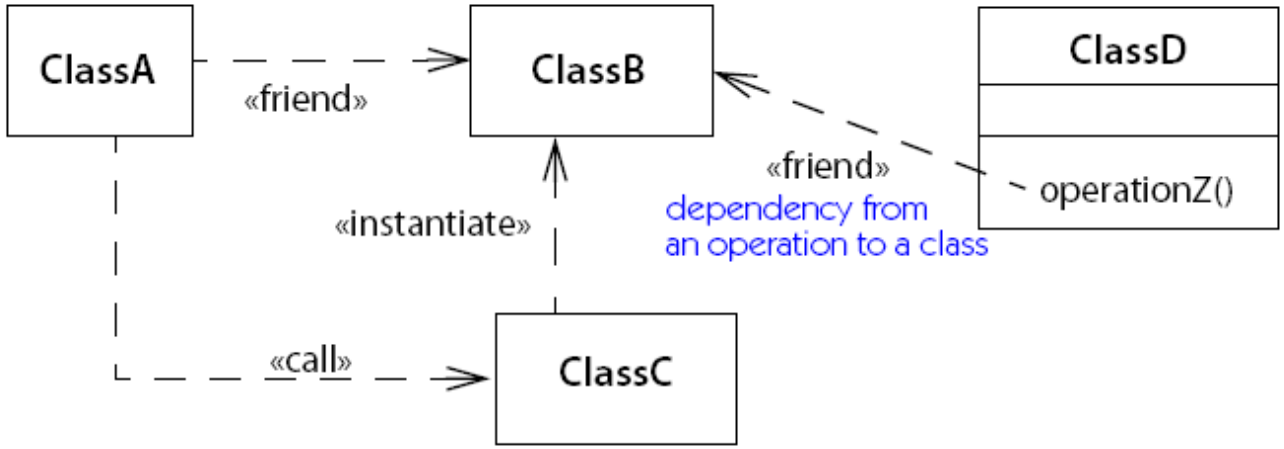
- An is-a relationship
- *Abstract* class

Realization



Dependency

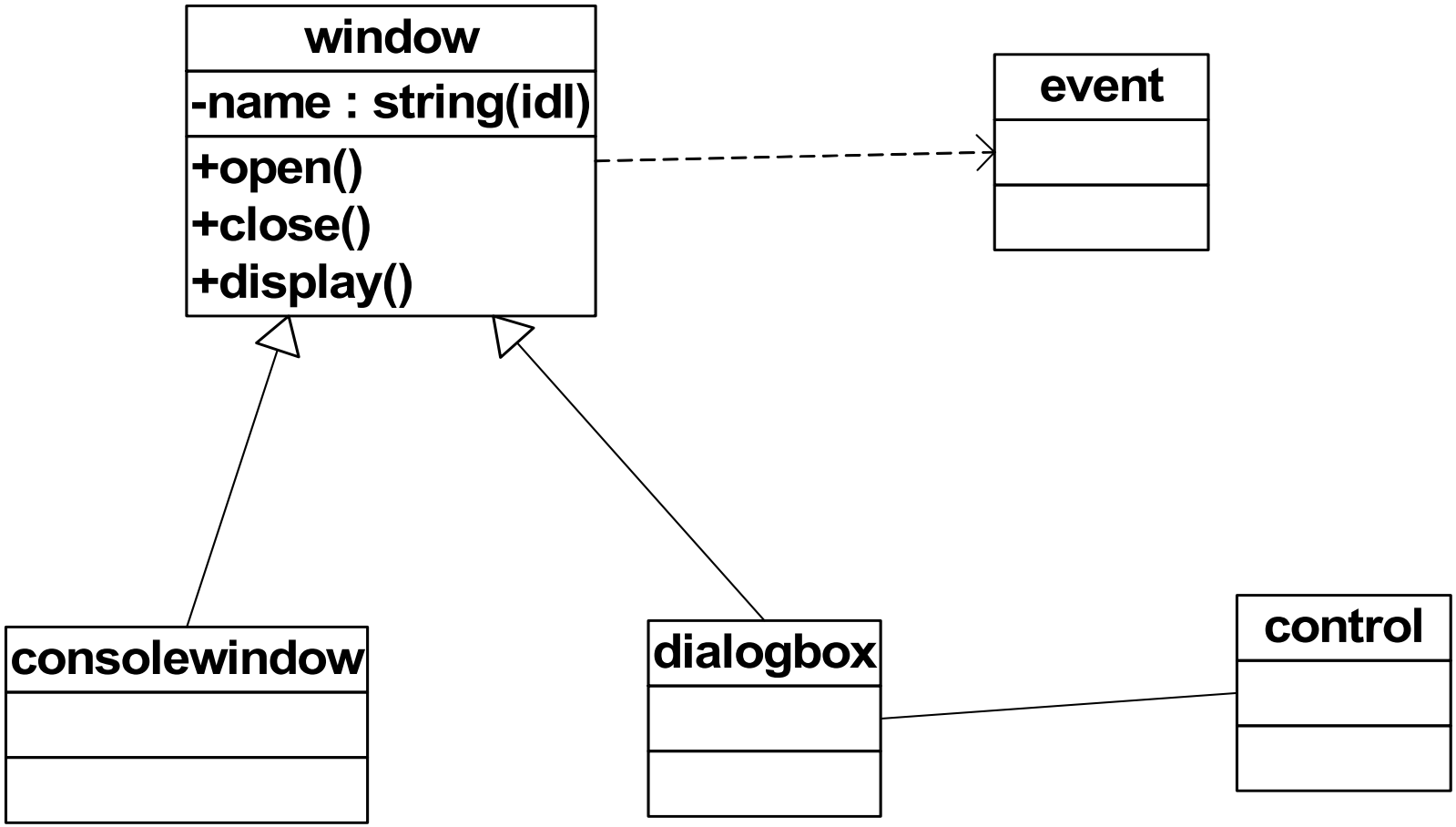
We use term dependencies for other relationships that do not fit sharper categories



Dependency	*Function*	*Keyword*
access	A private import of the contents of another package	access
binding	Assignment of values to the parameters of a template to generate a new model element	bind
call	Statement that a method of one class calls an operation of another class	call
creation	Statement that one class creates instances of another class	create
derivation	Statement that one instance can be computed from another instance	derive
instantiation	Statement that a method of one class creates instances of another class	instantiate
permission	Permission for an element to use the contents of another element	permit
realization	Mapping between a specification and an implementation of it	realize

instantiation	Statement that a method of one class creates instances of another class	instantiate
permission	Permission for an element to use the contents of another element	permit
realization	Mapping between a specification and an implementation of it	realize
refinement	Statement that a mapping exists between elements at two different semantic levels	refine
send	Relationship between the sender of a signal and the receiver of the signal	send
substitution	Statement that the source class supports the interfaces and contracts of the target class and may be substituted for it	substitute
trace dependency	Statement that some connection exists between elements in different models, but less precise than a mapping	trace
usage	Statement that one element requires the presence of another element for its correct functioning (includes call, creation, instantiation, send, and potentially others)	use

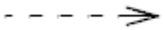
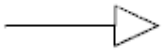
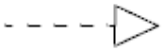
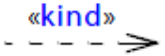
Example class diagram?



Which Relation is Right?

- **Aggregation** – aka is-part-of, is-made-of, contains
- Use **association** when specific (persistent) objects have multiple relationships (e.g., there was only one Bill Gates at MS and Steve Jobs at Apple)
- Use **dependency** when working with static objects, or if there is only one instance
- Do not confuse part-of with is-a

Relationships in UML

Relationship	*Function*	*Notation*
association	A description of a connection among instances of classes	
dependency	A relationship between two model elements	
generalization	A relationship between a more specific and a more general description, used for inheritance and polymorphic type declarations	
realization	Relationship between a specification and its implementation	
usage	A situation in which one element requires another for its correct functioning	

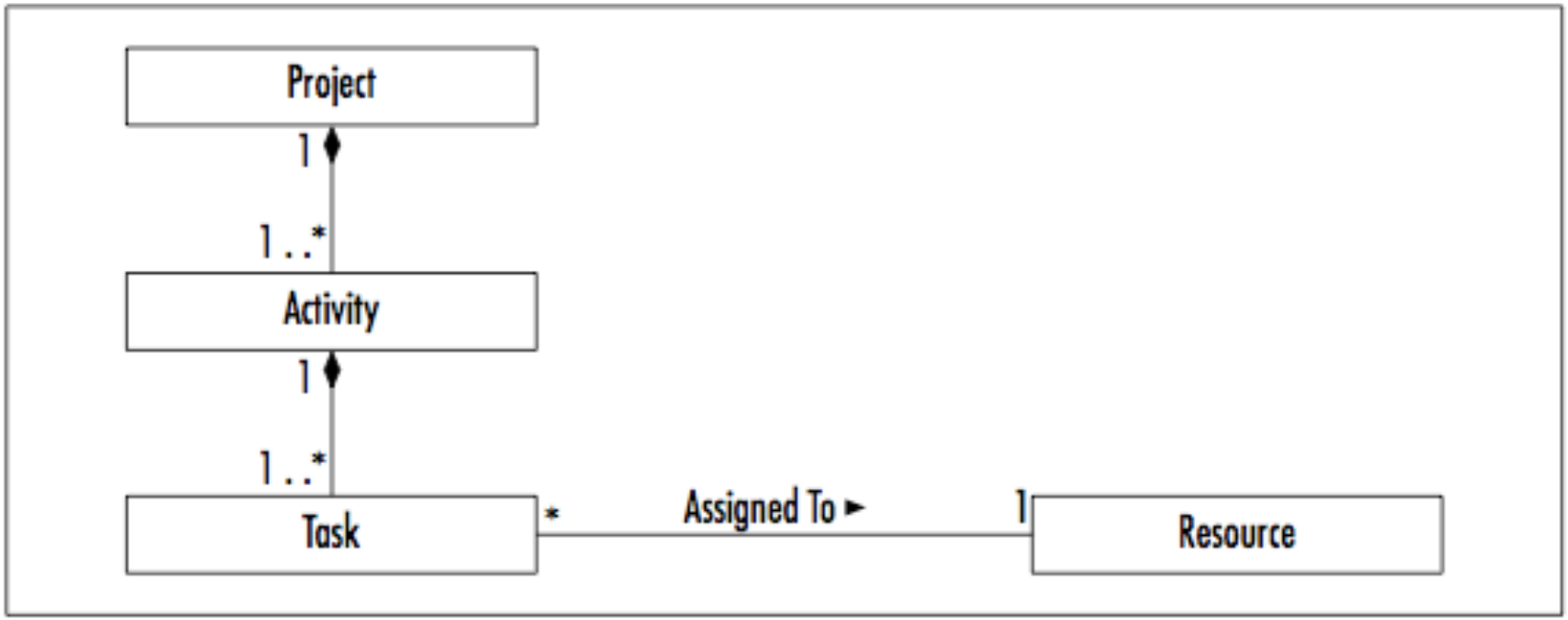


Figure 4-5: High-Level Project Class Diagram

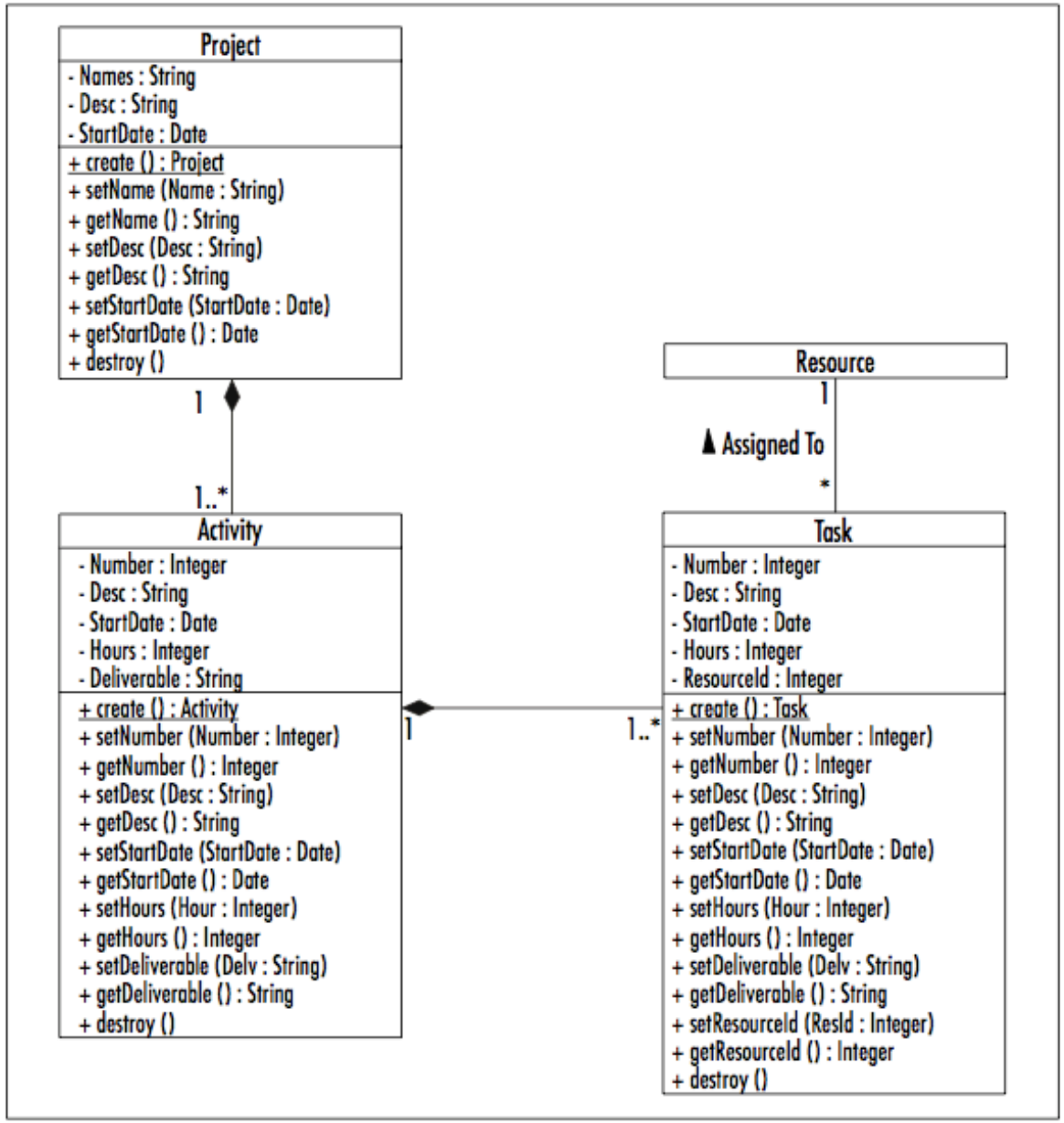


Figure 4-7: Detailed Project Class Diagram

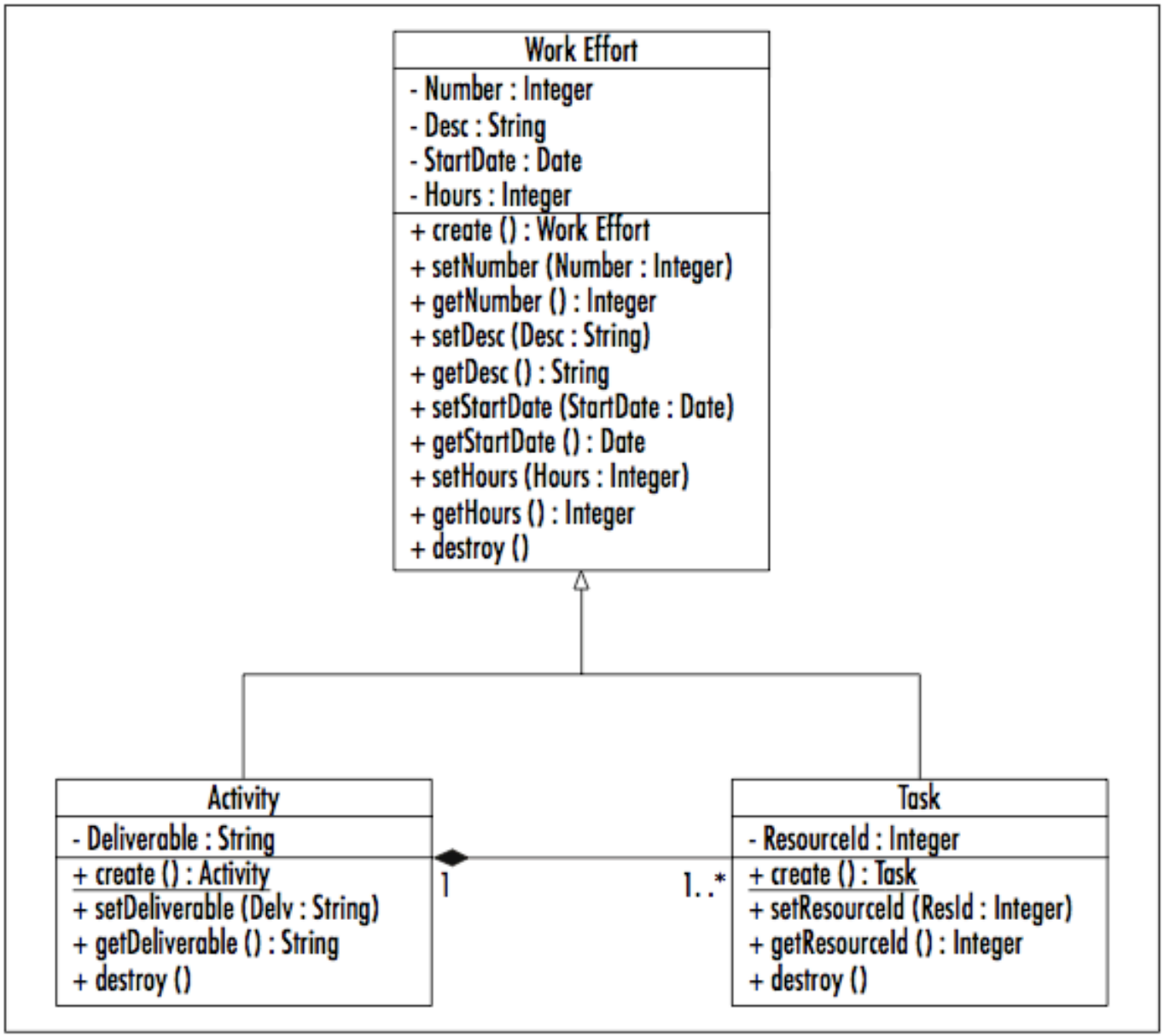


Figure 4-8: Detailed Activities and Tasks Class Diagram

Object Diagram

- Object diagram is an instantiation of a class diagram
- Represents a static structure of a system at a particular time

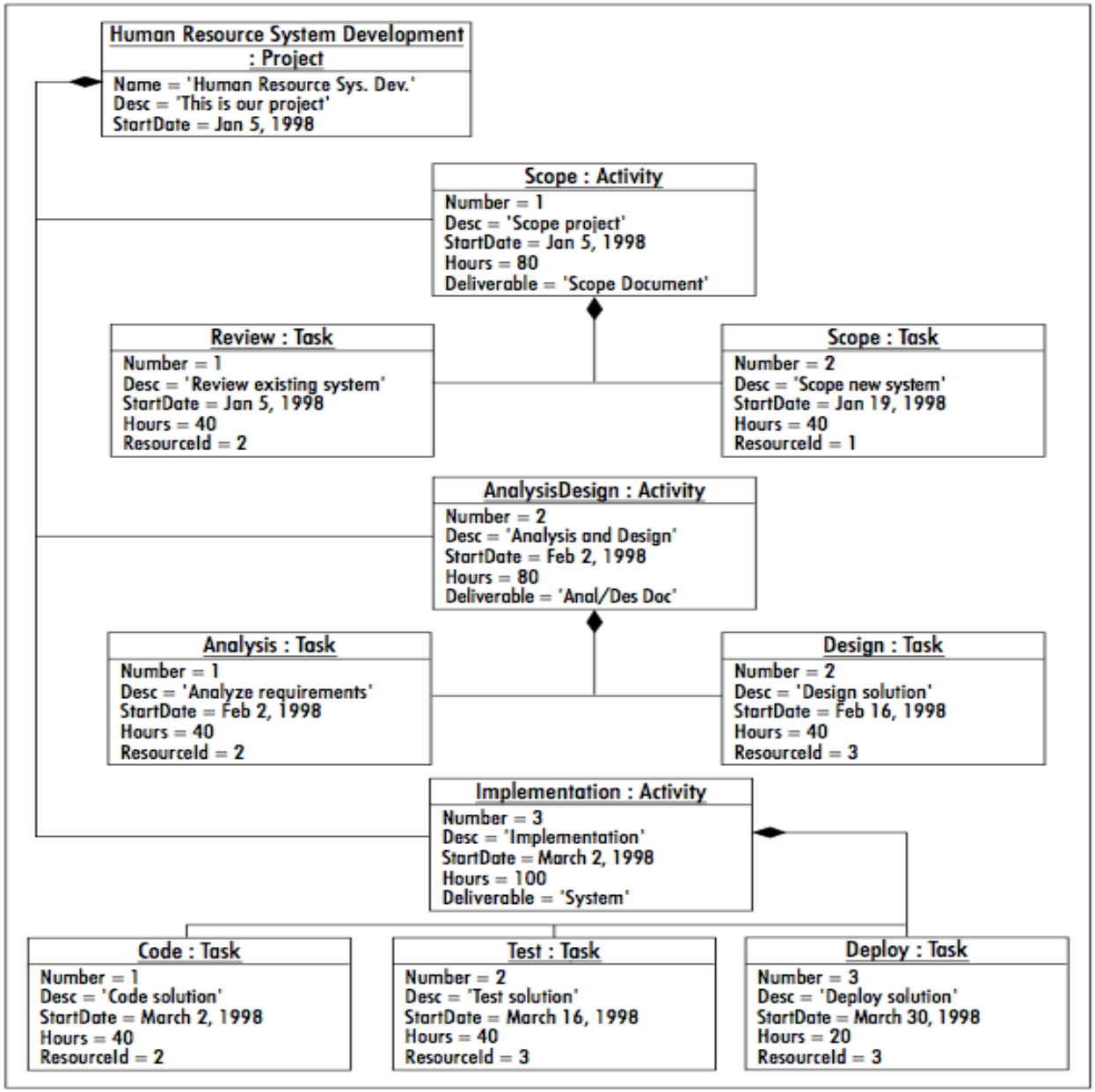


Figure 4-11: Project Object Diagram

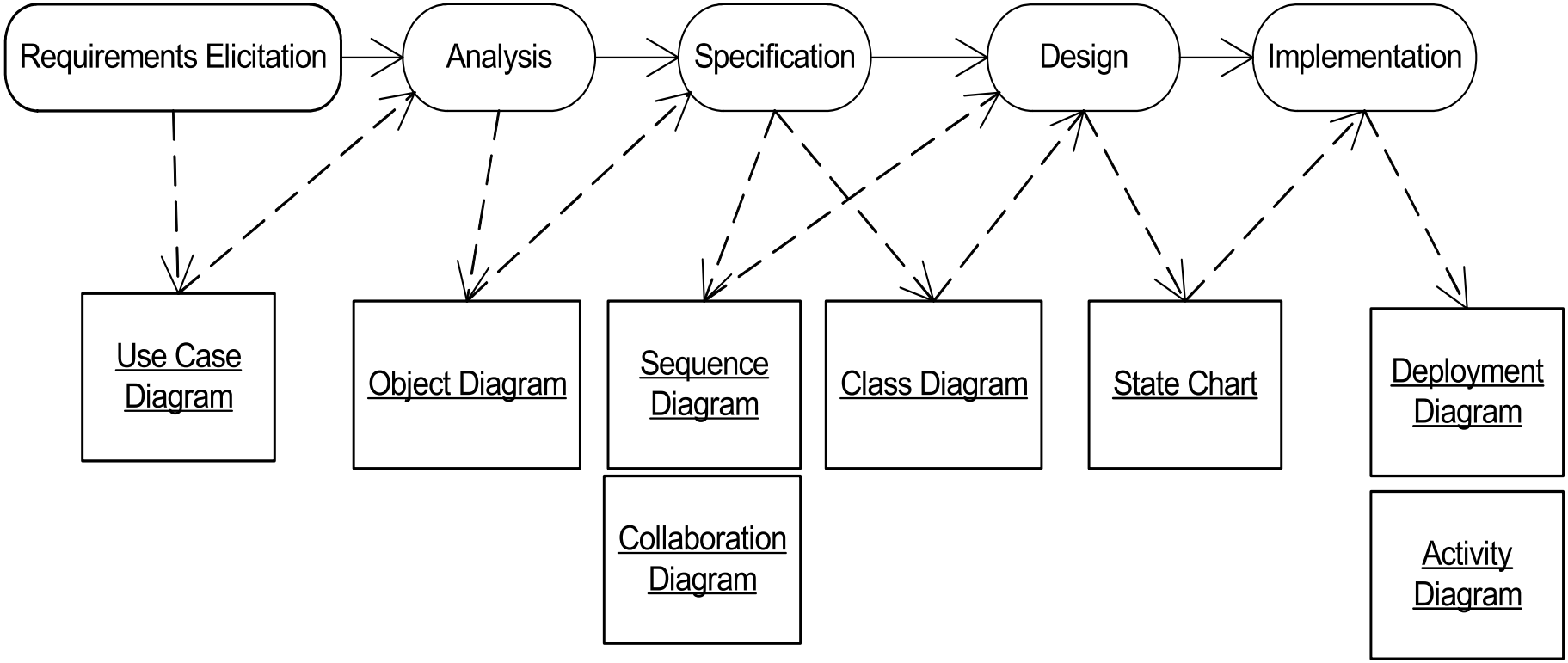
Sequence Diagrams

- Sequence diagrams
  - Refine use cases
  - Gives view of dynamic behavior of classes
    - Class diagrams give the static class structure
- Not orthogonal to other diagrams
  - Overlapping functionality
  - True of all UML diagrams

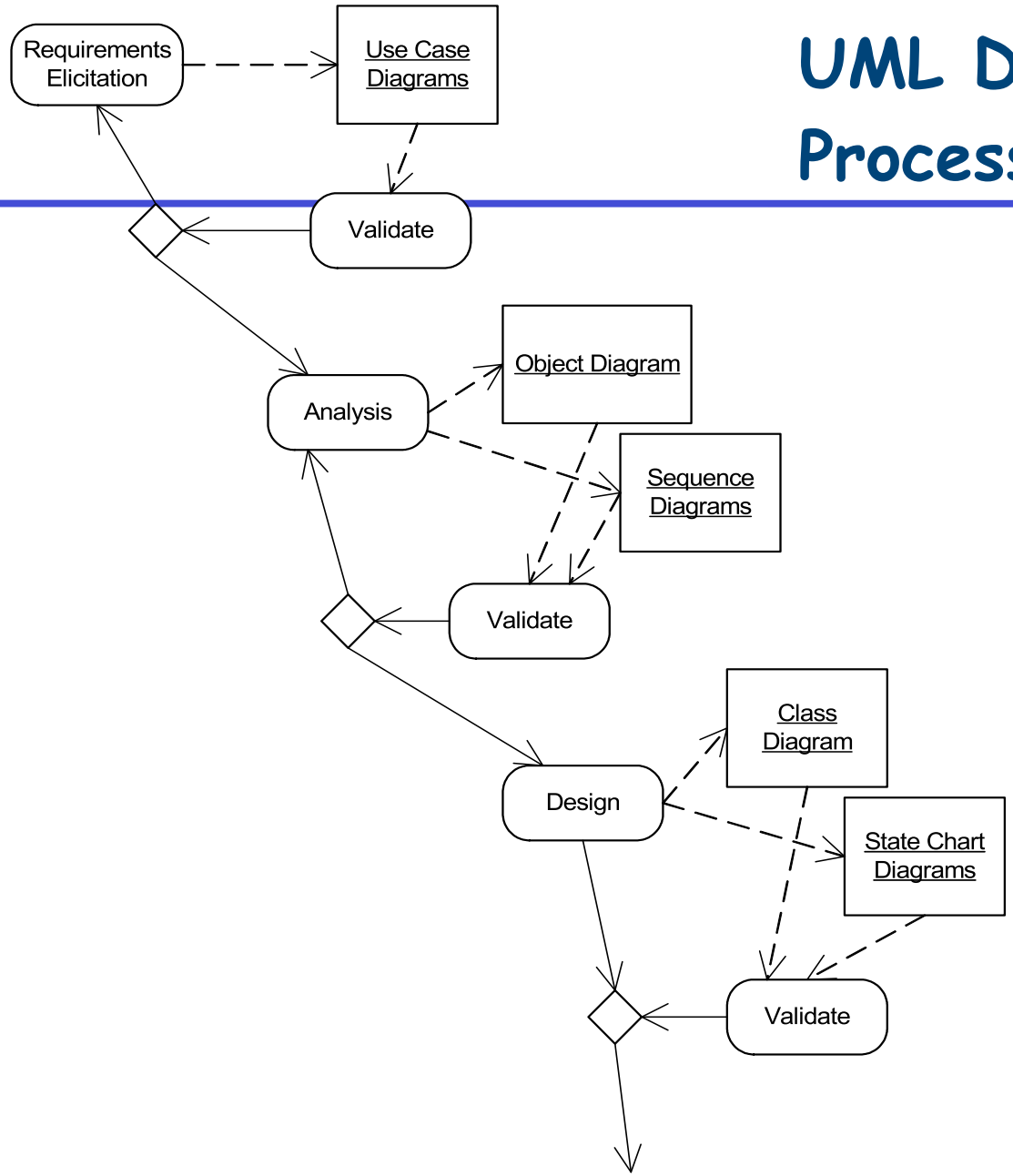
Development Process

- Requirements elicitation – High level capture of user/ system requirements
  - Use Case Diagram
- Identify major objects and relationships
  - Object and class diagrams
- Create scenarios of usage
  - Class, Sequence and Collaboration diagrams
- Generalize scenarios to describe behavior
  - Class, State and Activity Diagrams
- Refine and add implementation details
  - Component and Deployment Diagrams

UML Driven Process



UML Driven Process Model



Work Products

- Functional Model – Use Case diagrams
- Analysis Object Model – simple object/class diagram
- Dynamic Model – State and Sequence diagrams
- Object Design Model – Class diagrams
- Implementation Model – Deployment, and Activity diagrams

Acknowledgements

- Many slides courtesy of Rupak Majumdar