Introduction to Object Orientation

and

Class Diagrams

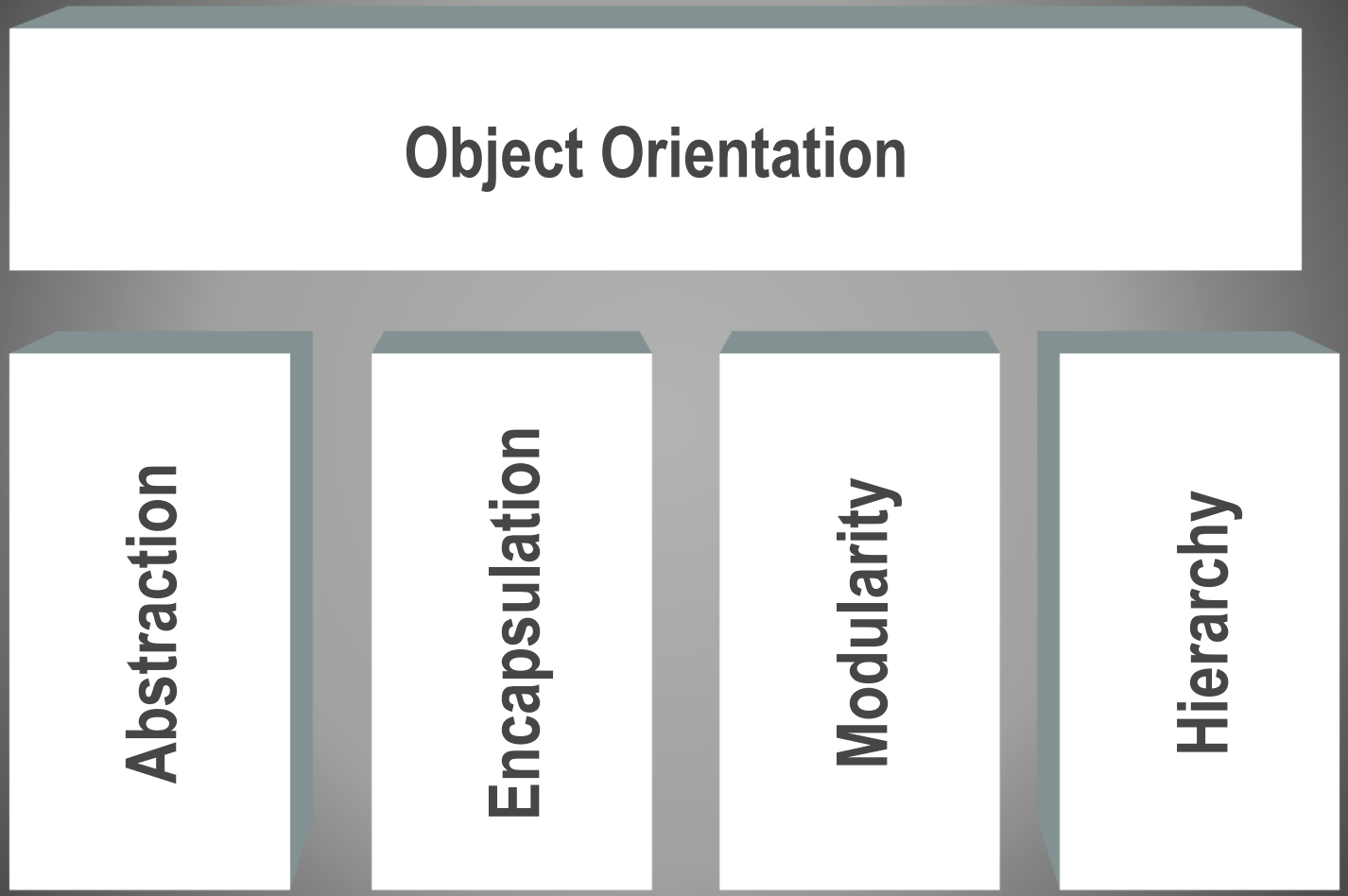
Objectives: Introduction to Object Orientation

- Understand the basic principles of object orientation
- Understand the basic concepts and terms of object orientation and the associated UML notation
- Appreciate the strengths of object orientation
- Understand some basic UML modeling mechanisms

Introduction to Object Orientation Topics

- Basic Principles of Object Orientation
- Basic Concepts of Object Orientation
- Strengths of Object Orientation
- General UML Modeling Mechanisms

Basic Principles of Object Orientation

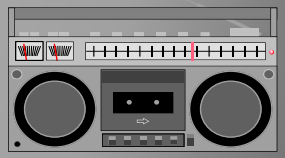


What is Abstraction?

Salesperson

Not saying <u>Which</u> salesperson – just a salesperson in general!!!

Customer

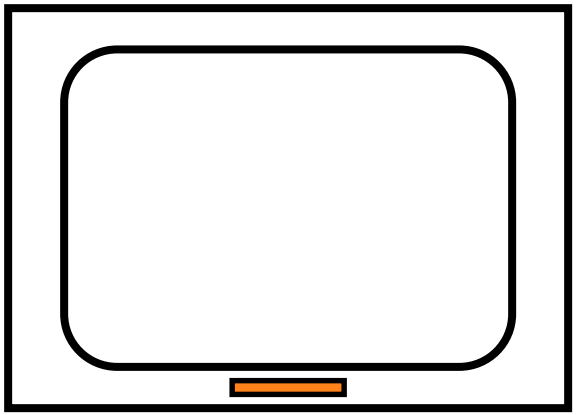


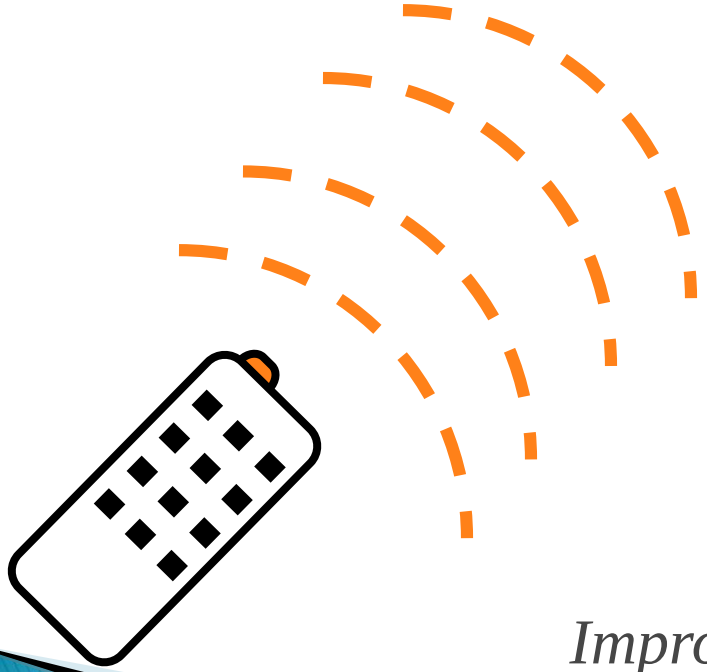
Product

Manages Complexity

What is Encapsulation?

- Hide implementation from clients
  - Clients depend on interface





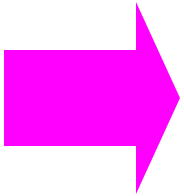
Improves Resiliency

What is Modularity?

- The breaking up of something complex into manageable pieces

Order Entry

Order Processing System



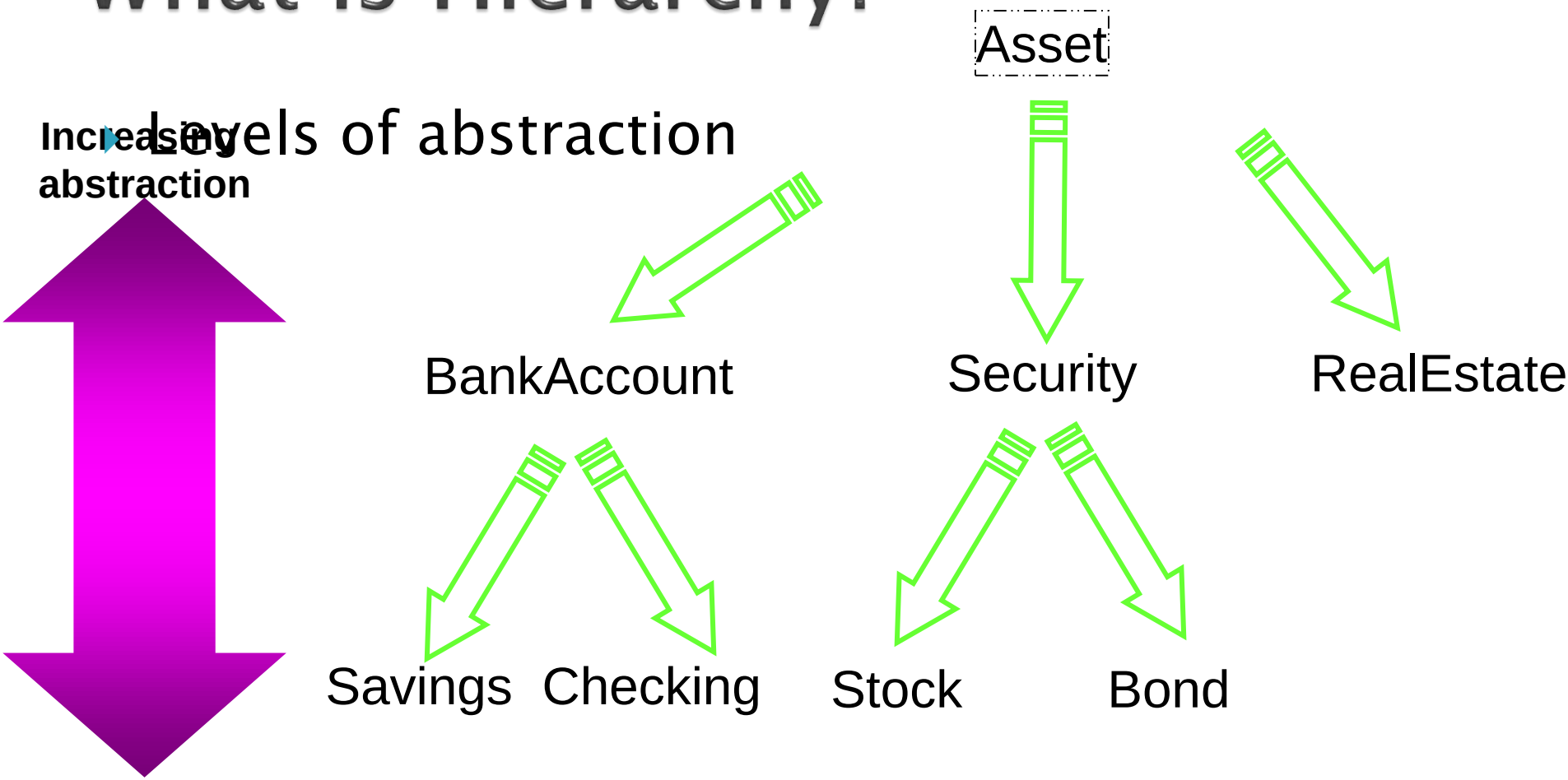
Order Fulfillment

Billing

Manages Complexity

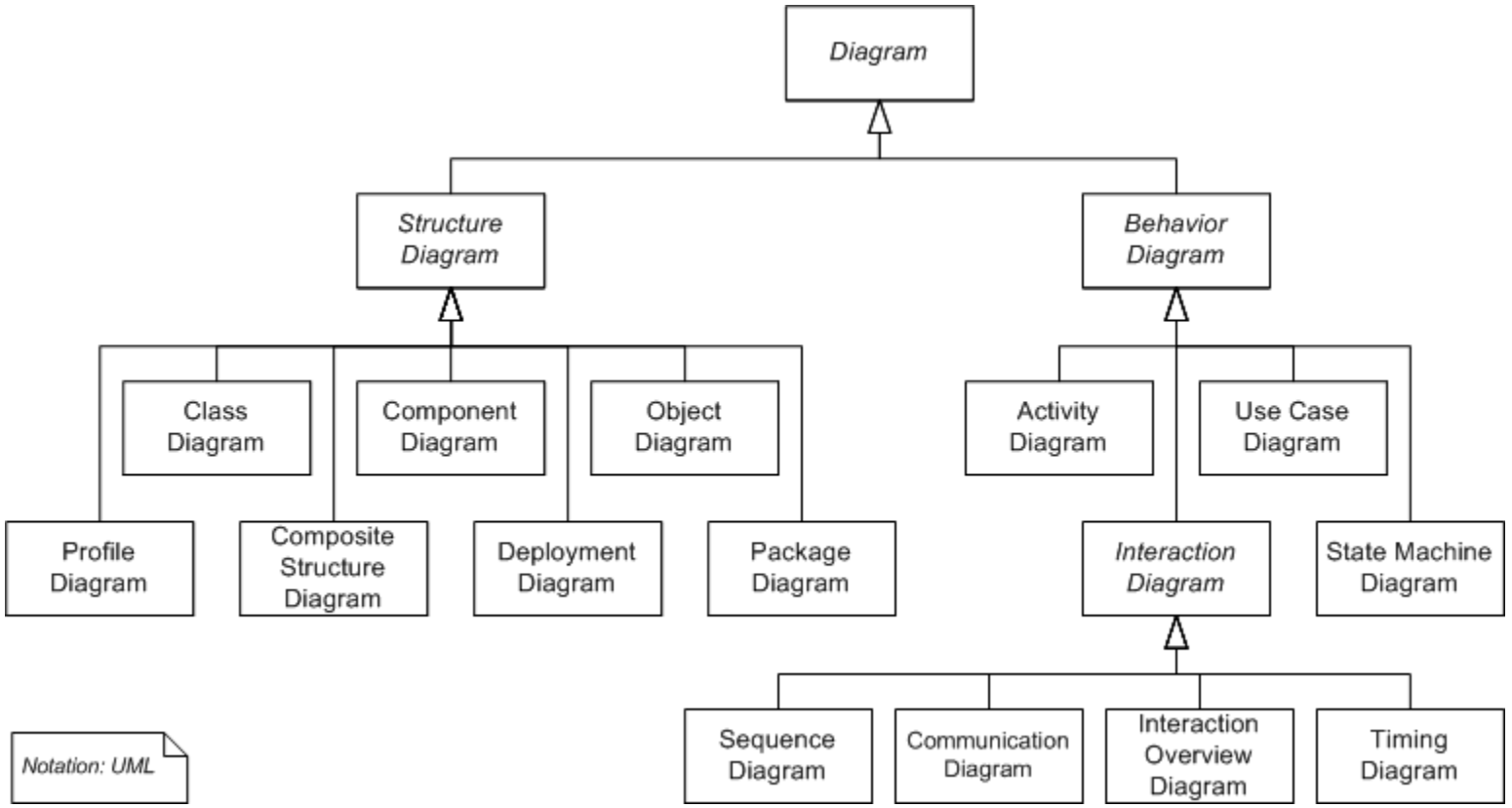
What is Hierarchy?

- Levels of abstraction



Elements at the same level of the hierarchy should be at the same level of abstraction

UML's Structural n Behavioral Hierarchy



Introduction to Object Orientation Topics

- Basic Principles of Object Orientation
- Basic Concepts of Object Orientation
- Strengths of Object Orientation
- General UML Modeling Mechanisms

Basic Concepts of Object Orientation

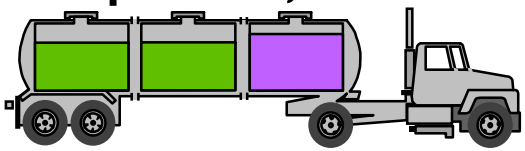
- Object
- Class
- Attribute
- Operation
- Interface (Polymorphism)
- Component
- Package
- Subsystem
- Relationships

Basic Concepts of Object Orientation

- Object
- Class
- Attribute
- Operation
- Interface (Polymorphism)
- Component
- Package
- Subsystem
- Relationships

What is an Object?

- Informally, an object represents an entity, either physical, conceptual, or software
  - Physical entity



Truck

  - Conceptual entity

Chemical Process

  - Software entity

Linked List

A More Formal Definition

- An object is a concept, abstraction, or thing with sharp boundaries and meaning for an application
- An object is something that has:
  - State
  - Behavior
  - Identity

Representing Objects

- An object is represented as rectangles with underlined names

: Professor

Class Name Only

ProfessorClark

Object Name Only

ProfessorClark : Professor

Class and Object Name

(stay tuned for classes)

Basic Concepts of Object Orientation

- Object
- Class
- Attribute
- Operation
- Interface (Polymorphism)
- Component
- Package
- Subsystem
- Relationships

What is a Class?

- A class is a description of a group of objects with common properties (attributes), behavior (operations), relationships, and semantics
  - An object is an instance of a class
- A class is an abstraction in that it:
  - Emphasizes relevant characteristics
  - Suppresses other characteristics

OO Principle: Abstraction

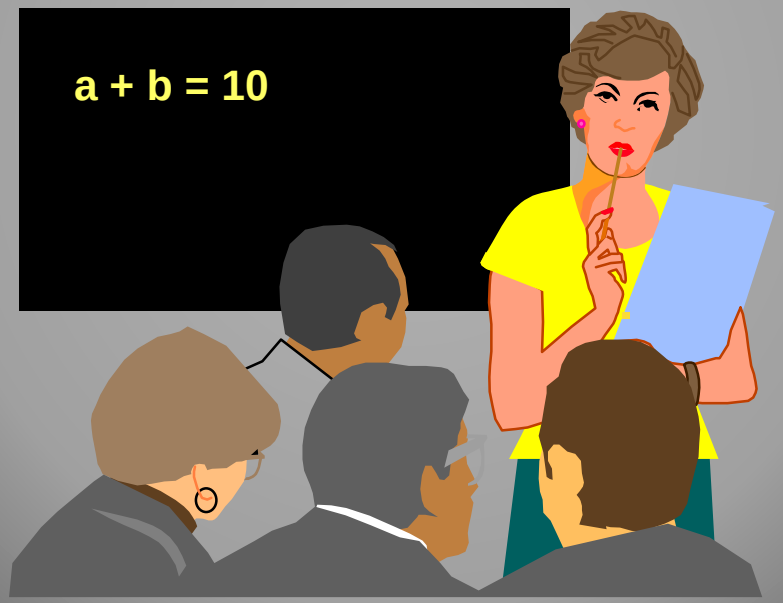
Sample Class

Class

Course

Properties

Name
Location
Days offered
Credit hours
Start time
End time

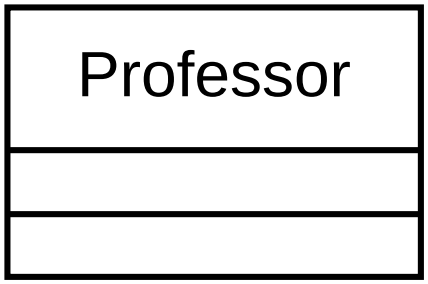


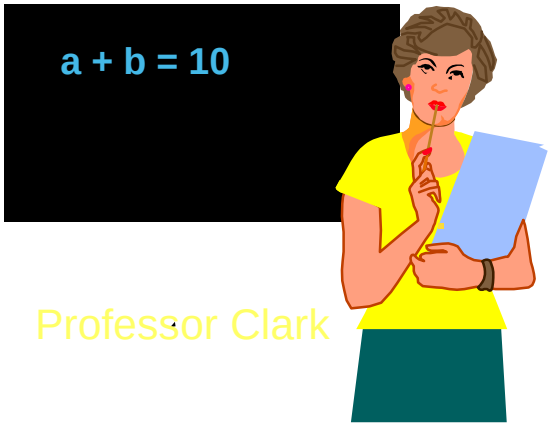
Behavior

Add a student
Delete a student
Get course roster
Determine if it is full

Representing Classes

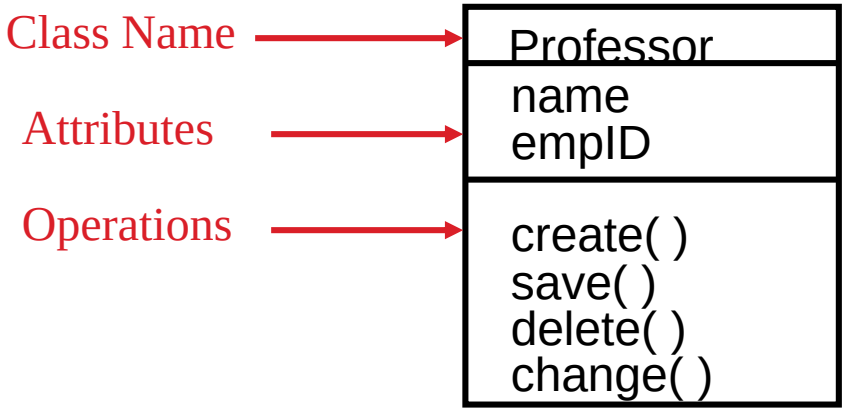
- A class is represented using a compartmented rectangle





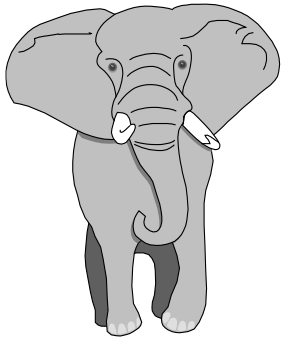
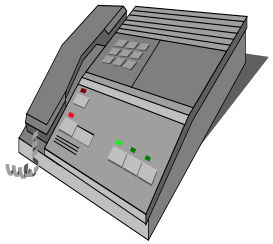
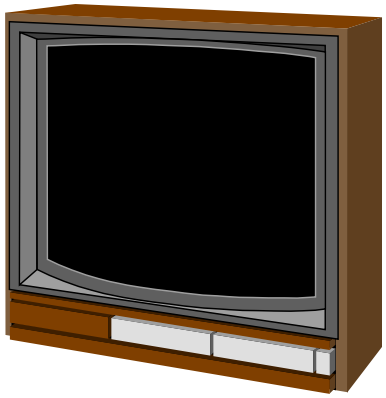
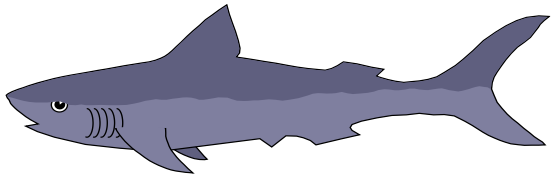
Class Compartments

- A class is comprised of three sections
  - The first section contains the class name
  - The second section shows the structure (attributes)
  - The third section shows the behavior (operations)



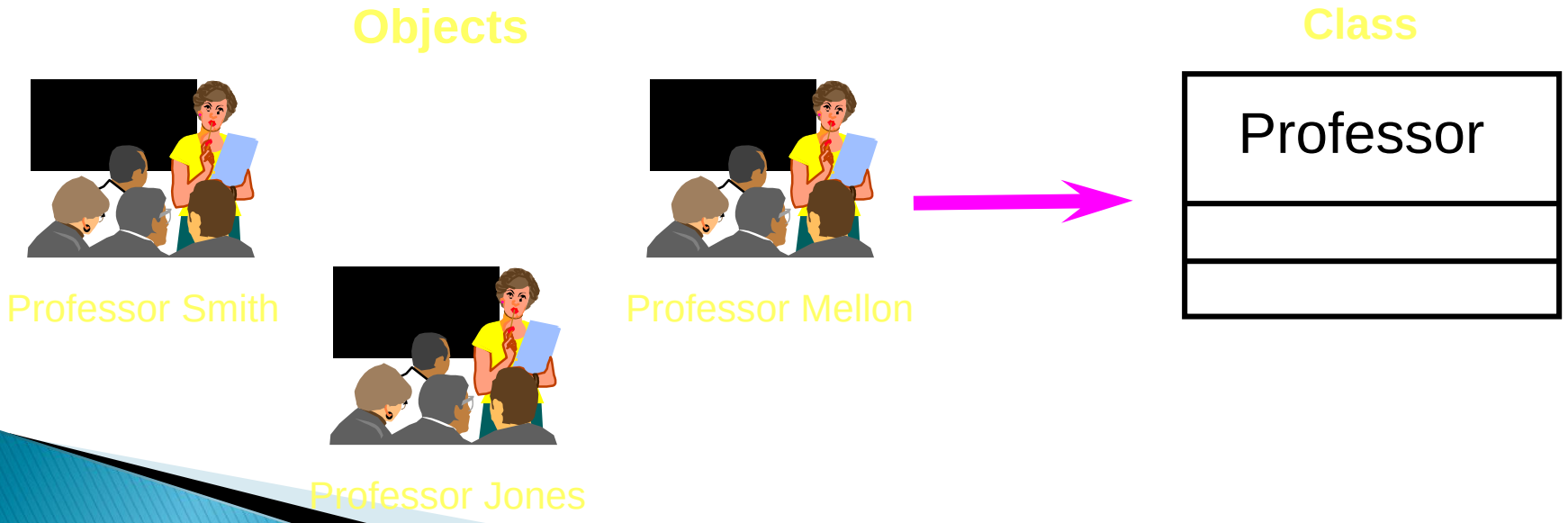
Classes of Objects

- How many classes do you see?


The Relationship Between Classes and Objects

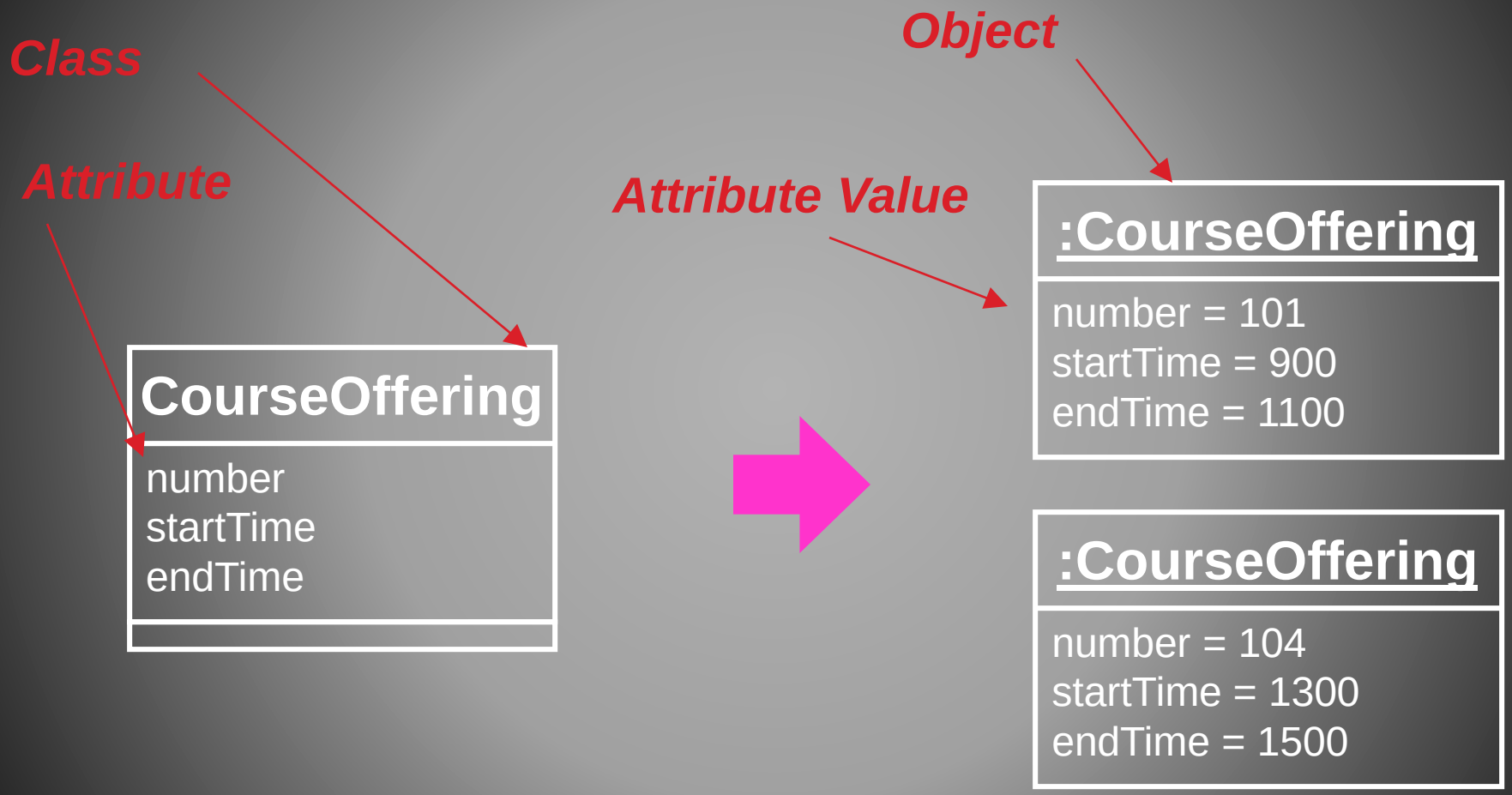
- A class is an abstract definition of an object
  - It defines the structure and behavior of each object in the class
  - It serves as a template for creating objects
- Objects are grouped into classes



Basic Concepts of Object Orientation

- Object
- Class
- Attribute
- Operation
- Interface (Polymorphism)
- Component
- Package
- Subsystem
- Relationships

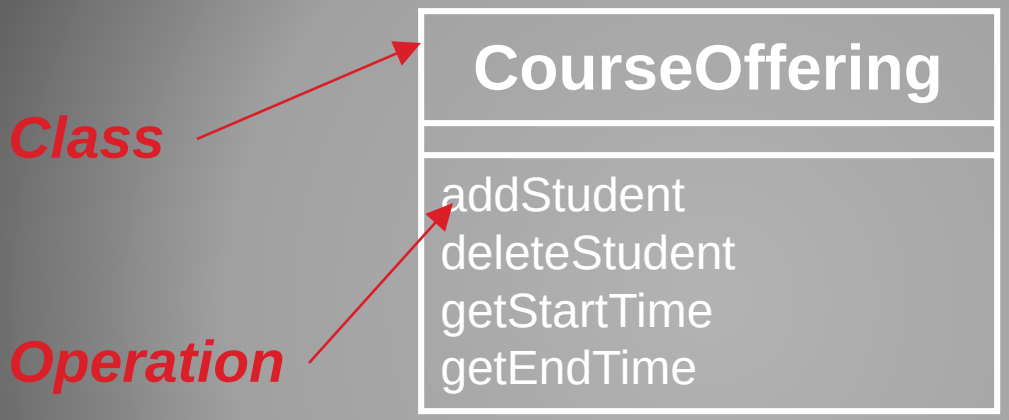
What is an Attribute?



Basic Concepts of Object Orientation

- Object
- Class
- Attribute
- Operation
- Interface (Polymorphism)
- Component
- Package
- Subsystem
- Relationships

What is an Operation?

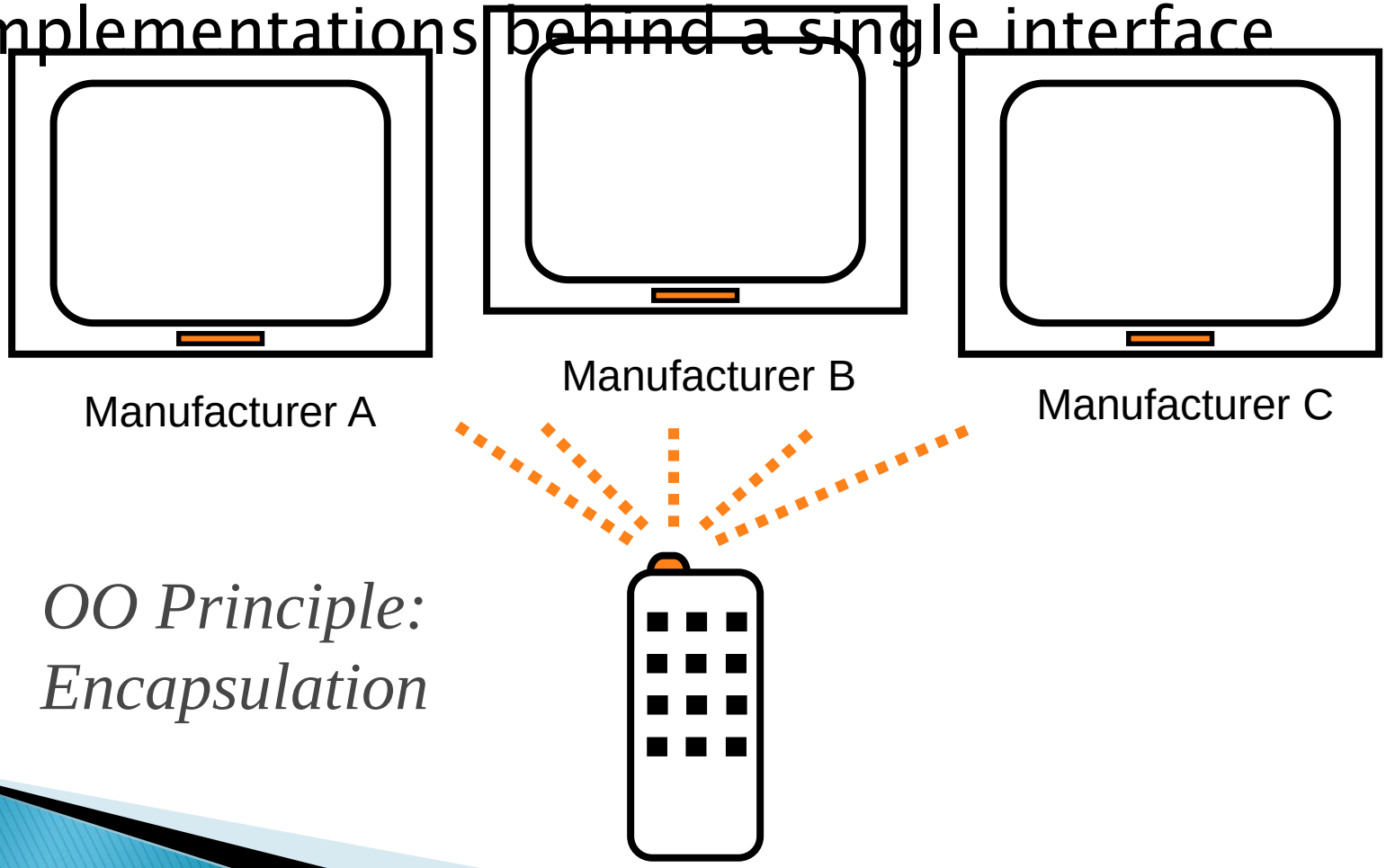


Basic Concepts of Object Orientation

- Object
- Class
- Attribute
- Operation
- Interface (Polymorphism)
- Component
- Package
- Subsystem
- Relationships

What is Polymorphism?

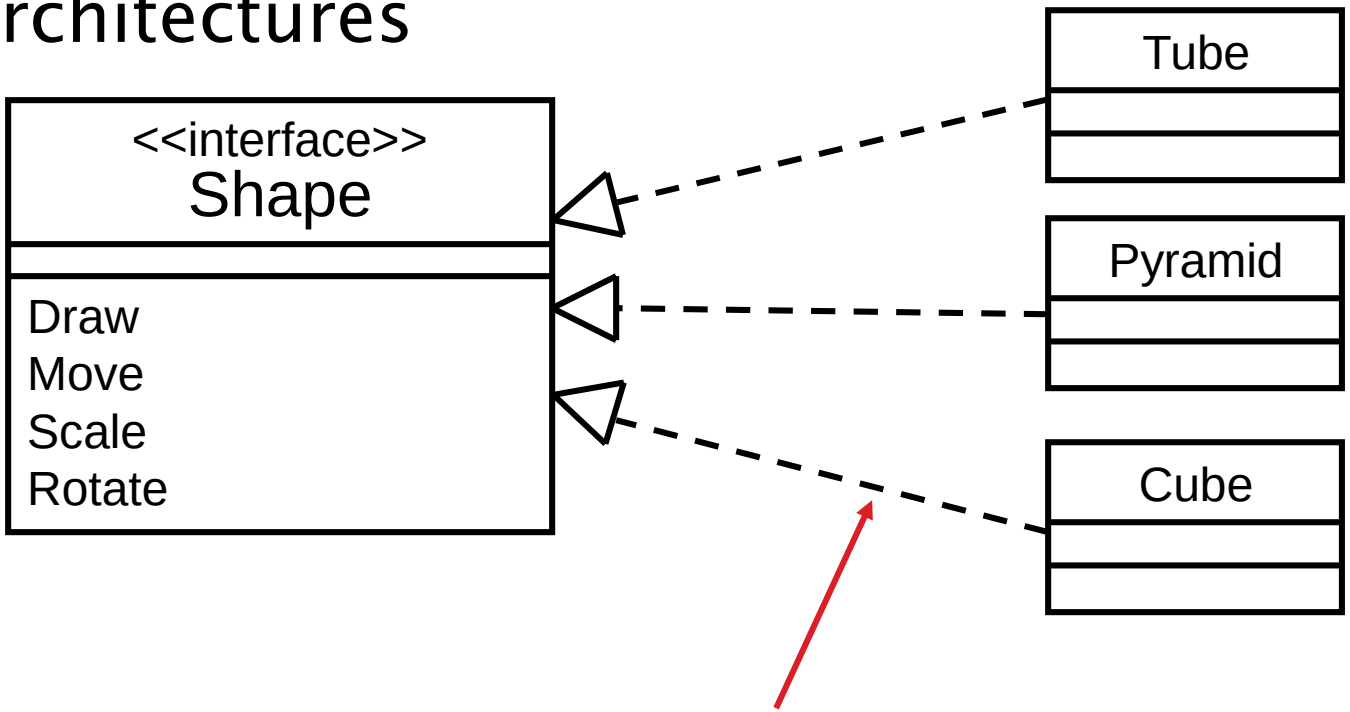
- The ability to hide many different implementations behind a single interface



OO Principle: Encapsulation

What is an Interface?

- Interfaces formalize polymorphism
- Interfaces support "plug-and-play" architectures

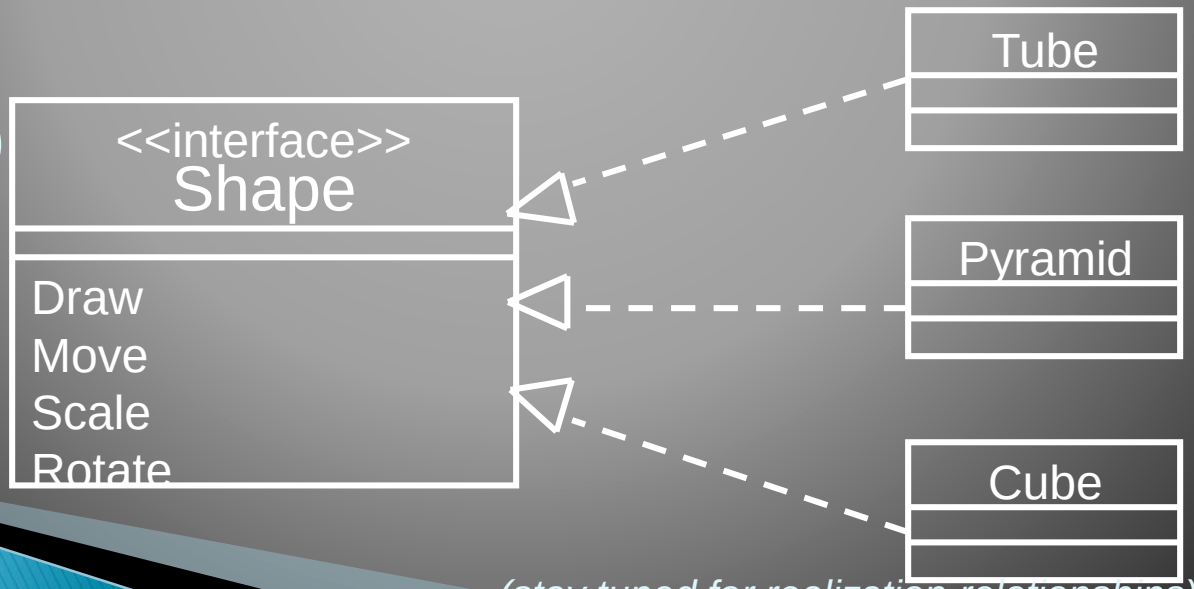


(stay tuned for realization relationships)

Interface Representations

Elided/Iconic Representation ("lollipop")

Canonical (Class/Stereotype) Representation



(stay tuned for realization relationships)

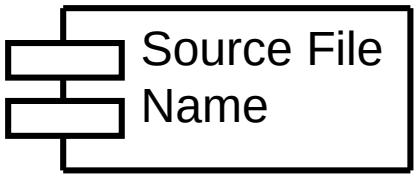
Basic Concepts of Object Orientation

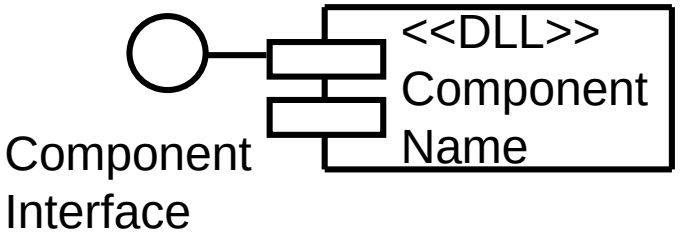
- Object
- Class
- Attribute
- Operation
- Interface (Polymorphism)
- Component
- Package
- Subsystem
- Relationships

What is a Component?

- A non-trivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture
- A component may be
  - A source code component
  - A run time components or
  - An executable component

OO Principle: Encapsulation



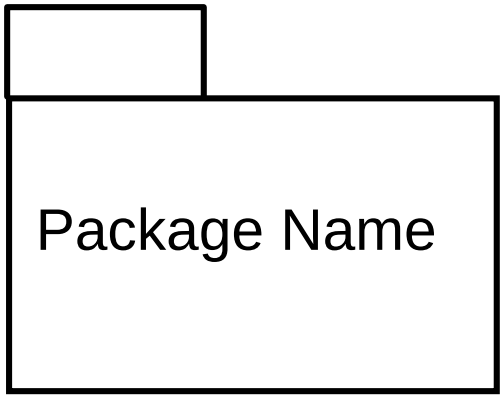


Basic Concepts of Object Orientation

- Object
- Class
- Attribute
- Operation
- Interface (Polymorphism)
- Component
- Package
- Subsystem
- Relationships

What is a Package?

- A package is a general purpose mechanism for organizing elements into groups
- A model element which can contain other model elements



OO Principle: Modularity

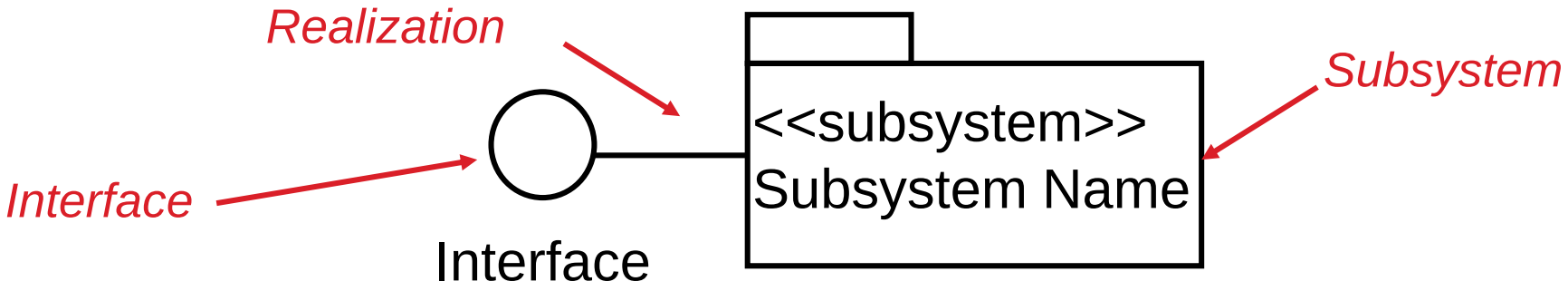
- Uses
  - Organize the model under development
  - A unit of configuration management

Basic Concepts of Object Orientation

- Object
- Class
- Attribute
- Operation
- Interface (Polymorphism)
- Component
- Package
- Subsystem
- Relationships

What is a Subsystem?

- A combination of a package (can contain other model elements) **and** a class (has behavior)
- Realizes one or more interfaces which define its behavior



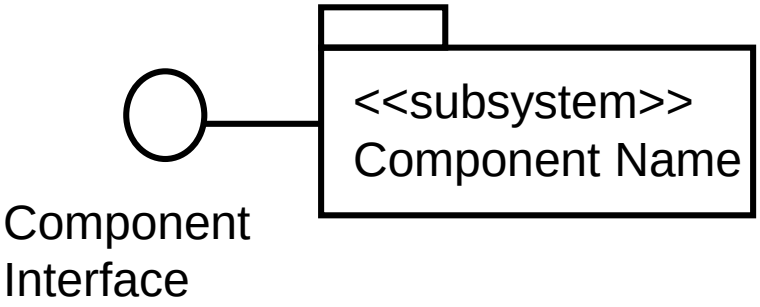
OO Principles: Encapsulation and Modularity

(stay tuned for realization relationship)

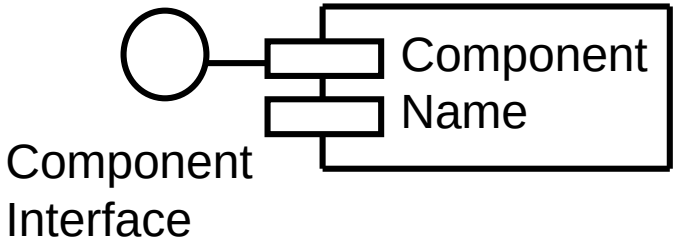
Subsystems and Components

- Components are the physical realization of an abstraction in the design
- Subsystems can be used to represent the component in the design

Design Model



Implementation Model



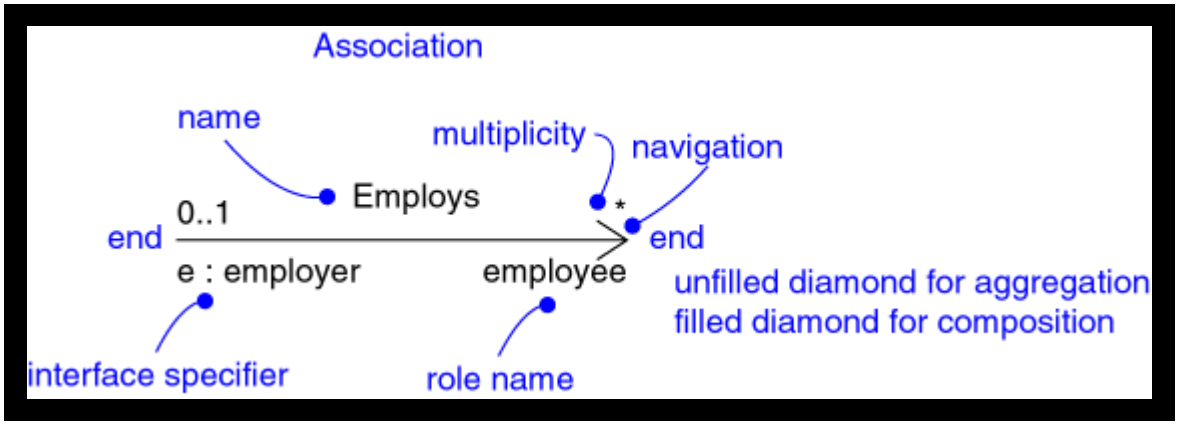
OO Principles: Encapsulation and Modularity

Basic Concepts of Object Orientation

- Object
- Class
- Attribute
- Operation
- Interface (Polymorphism)
- Component
- Package
- Subsystem
- Relationships

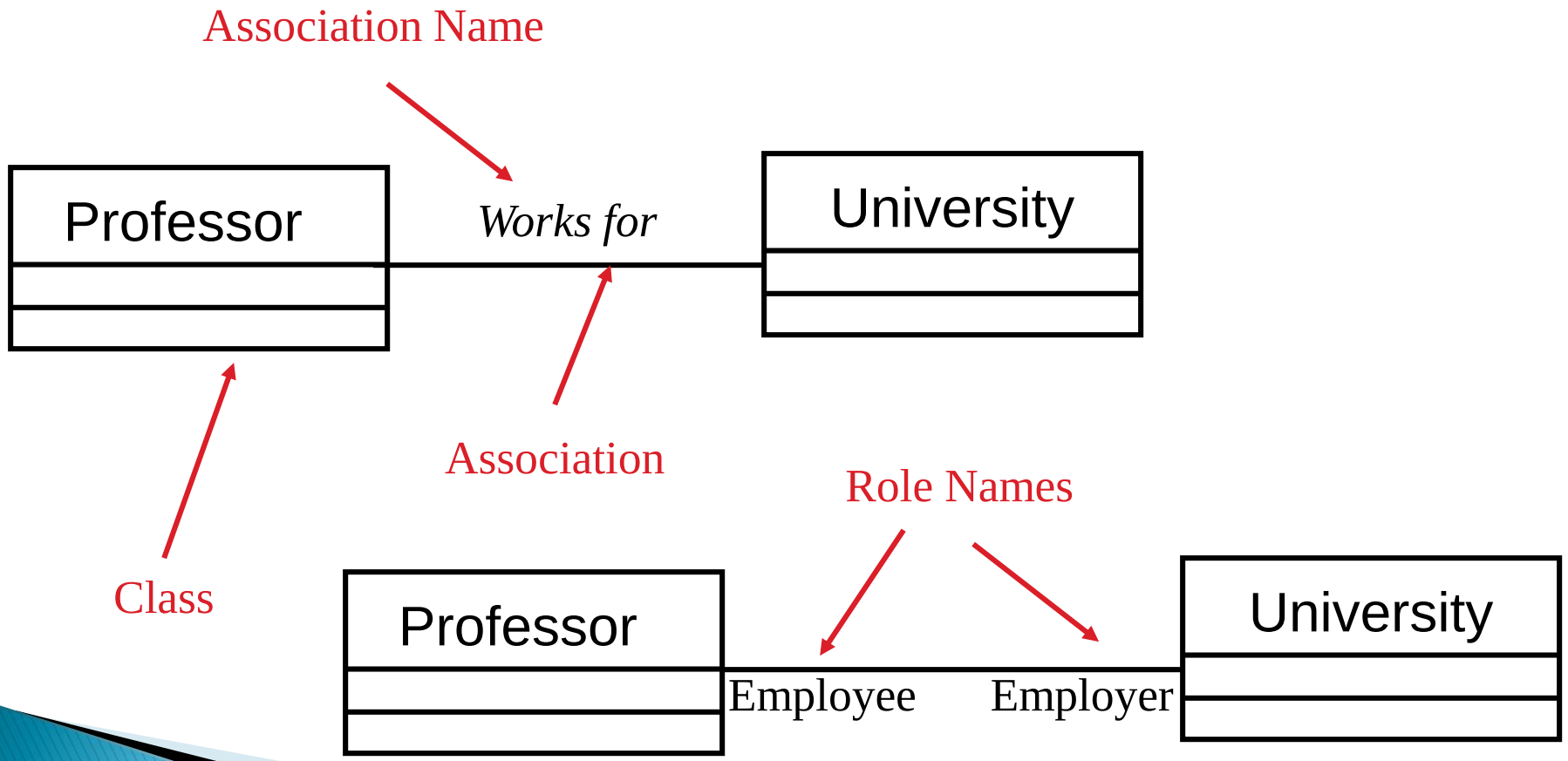
Relationships

- Association
  - Aggregation
  - Composition
- Dependency
- Generalization
- Realization



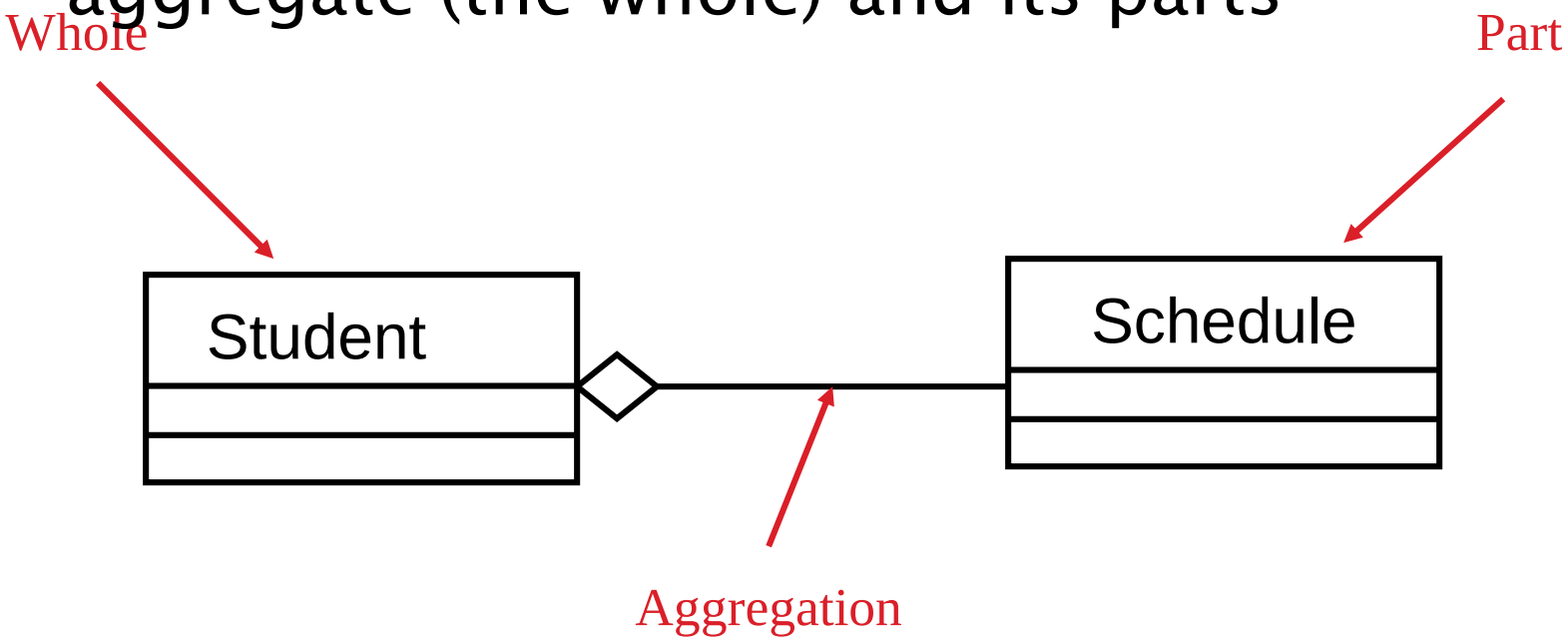
Relationships: Association

- Models a semantic connection among classes



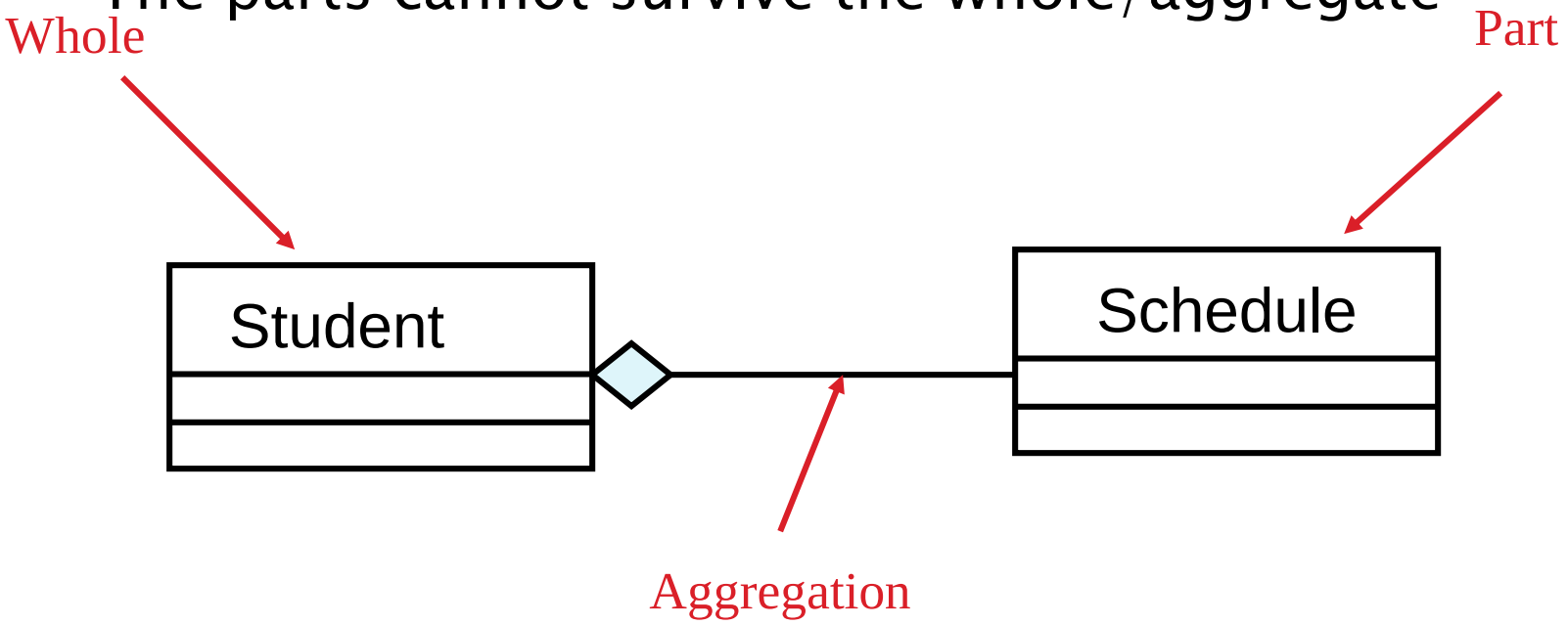
Relationships: Aggregation

- A special form of association that models a whole-part relationship between an aggregate (the whole) and its parts



Relationships: Composition

- A form of aggregation with strong ownership and coincident lifetimes
  - The parts cannot survive the whole/aggregate



Composition vs Aggregation

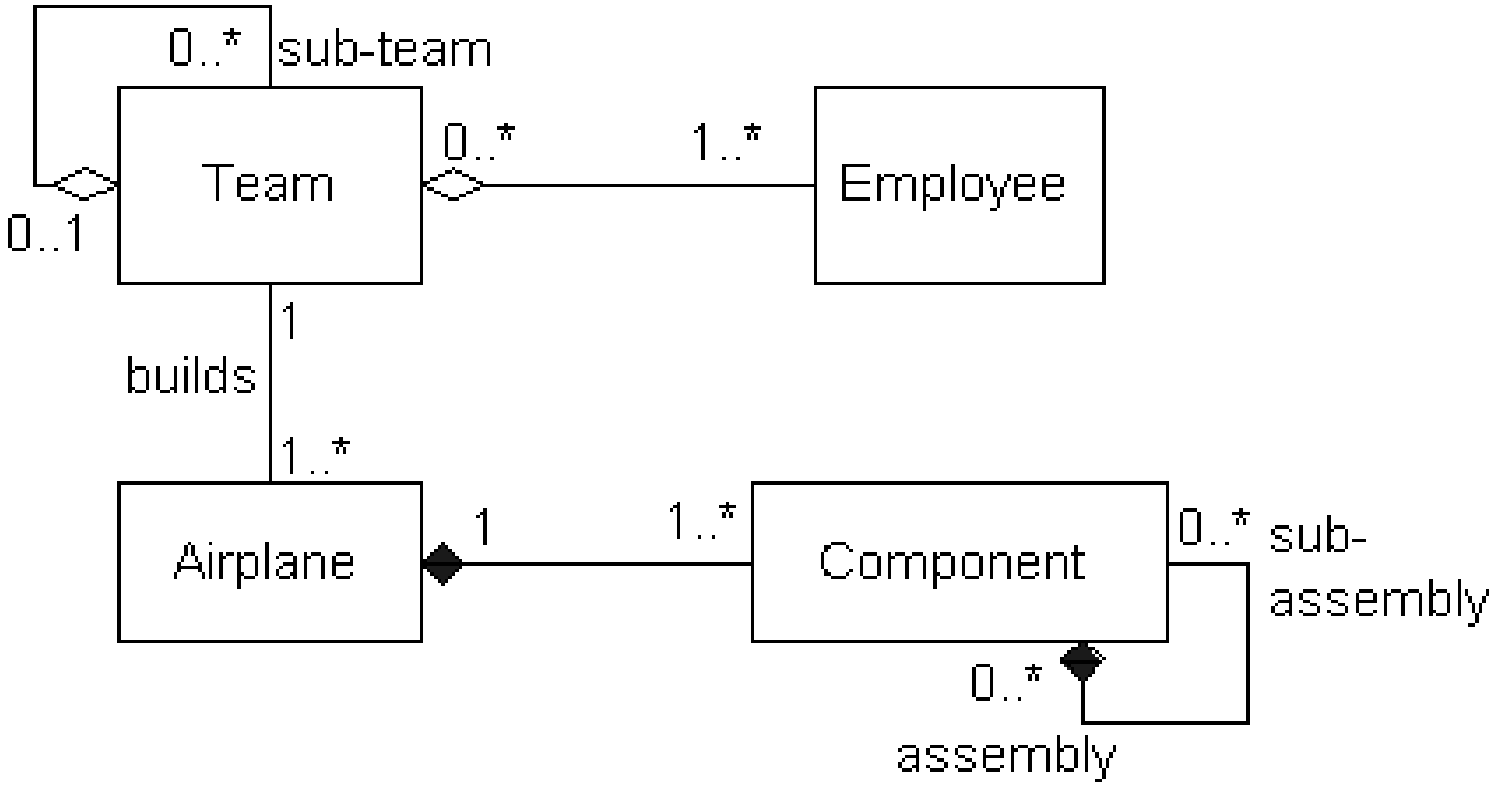
Composition is really a strong form of association

- components have only one owner
- components cannot exist independent of their owner
- components live or die with their owner
- e.g. Each car has an engine that can not be shared with other cars.

Aggregations

- may form "part of" the association, but may not be essential to it. They may also exist independent of the aggregate. e.g. Employees may exist independent of the team.

Association: Composition and Aggregation



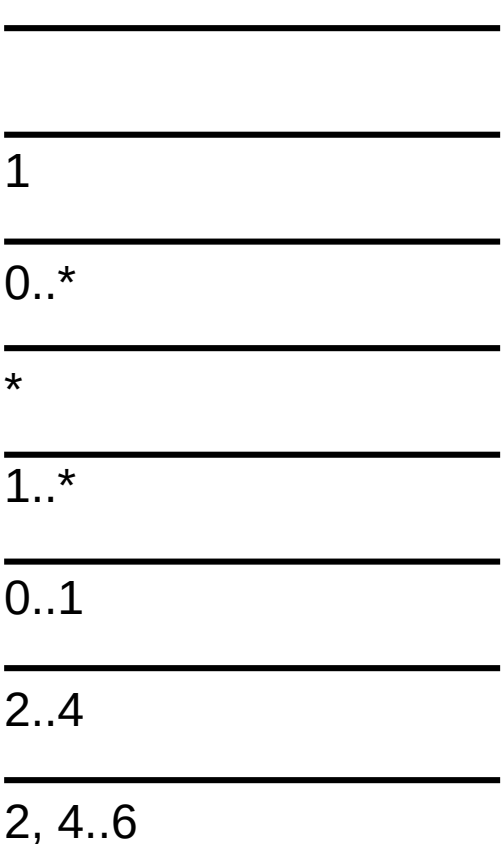
Association: Multiplicity and Navigation

- Multiplicity defines how many objects participate in a relationships
  - The number of instances of one class related to ONE instance of the other class
  - Specified for each end of the association
- Associations and aggregations are bi-directional by default, but it is often desirable to restrict navigation to one direction
  - If navigation is restricted, an arrowhead is added to indicate the direction of the navigation

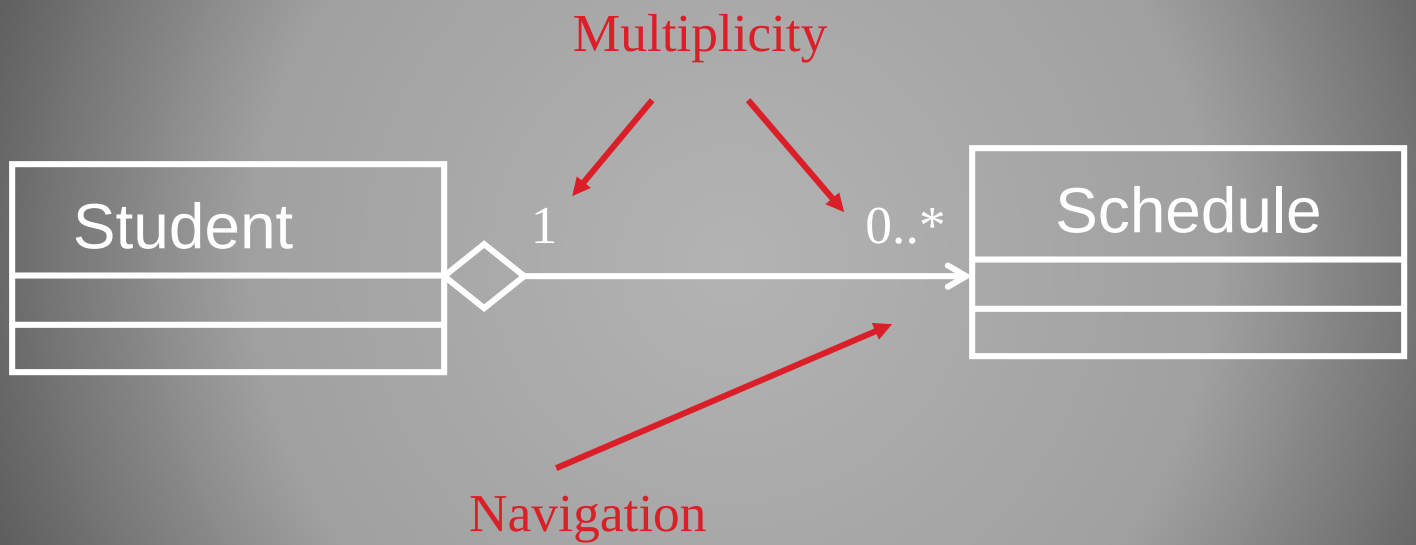
Association: Multiplicity

- Unspecified
- Exactly one
- Zero or more (many, unlimited)

- One or more
- Zero or one
- Specified range
- Multiple, disjoint ranges

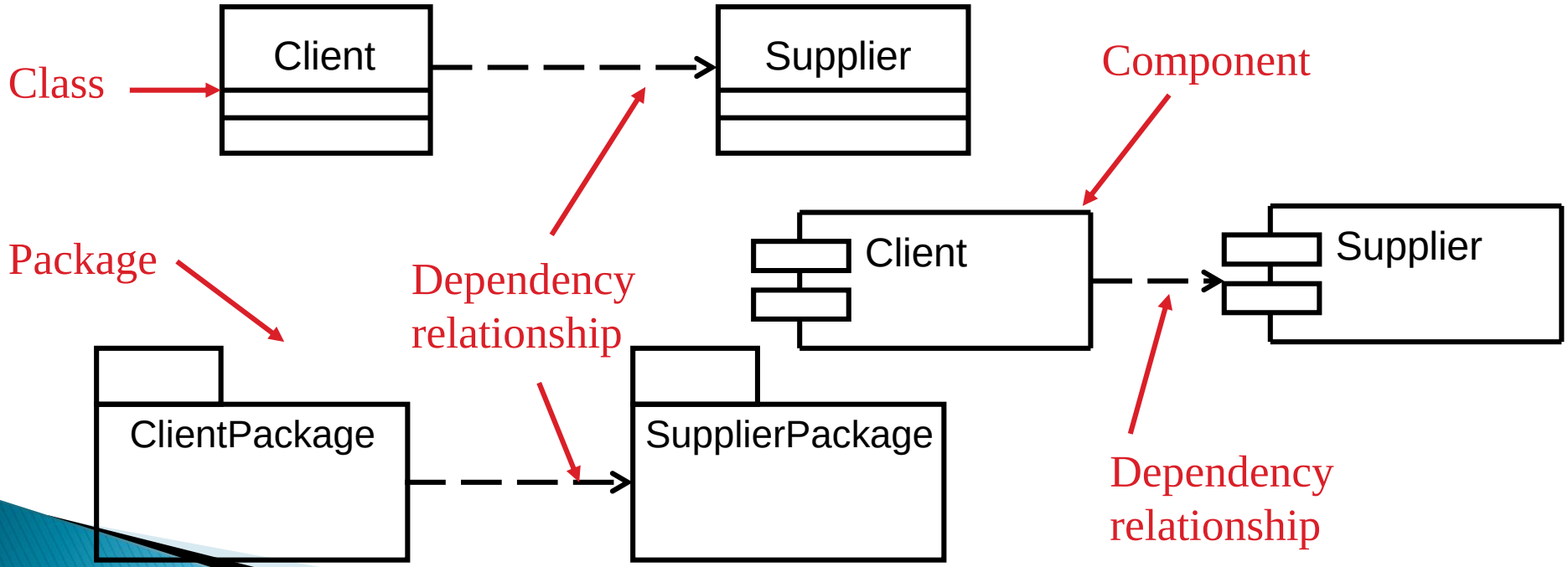


Example: Multiplicity and Navigation



Relationships: Dependency

- A relationship between two model elements where a change in one **may** cause a change in the other
- Non-structural, "using" relationship



Relationships: Generalization
Used for abstracting details in several layers

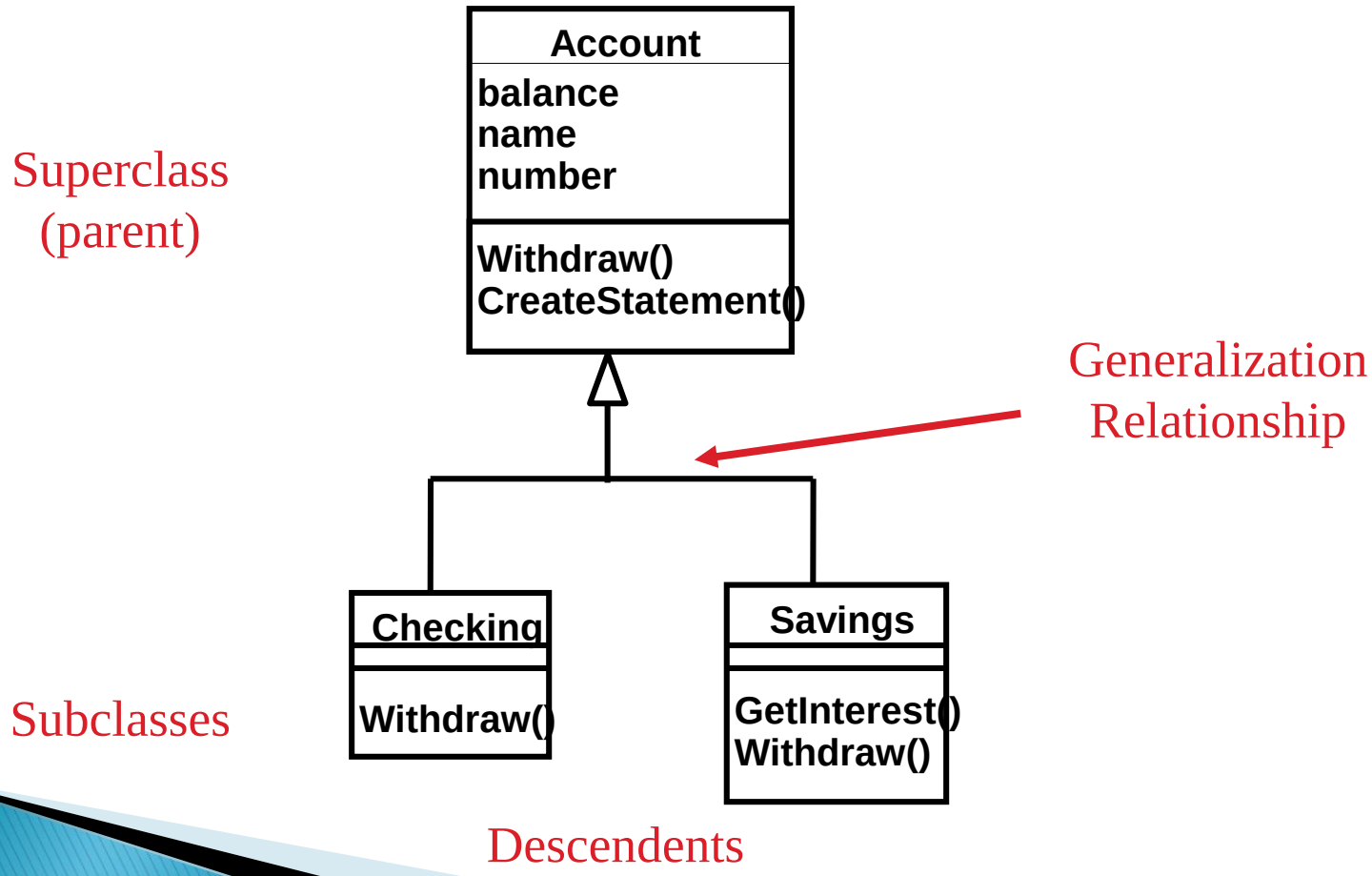
- A relationship among classes where one class shares the structure and/or behavior of one or more classes

- Defines a hierarchy of abstractions in which a subclass inherits from one or more super classes
  - Single inheritance
  - Multiple inheritance

- Generalization is an "is-a-kind of" relationship

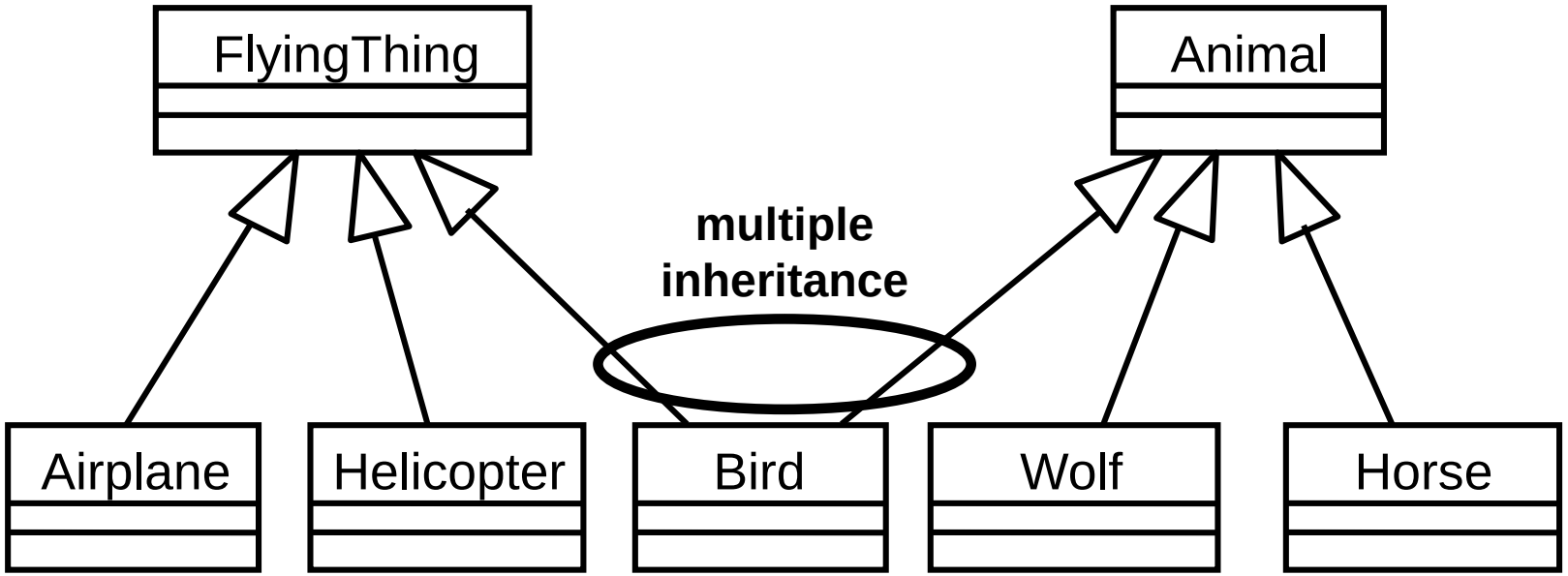
Example: Single Inheritance

- One class inherits from another



Example: Multiple Inheritance

- A class can inherit from several other classes



Use multiple inheritance only when needed, and always with caution !

What Gets Inherited?

- A subclass inherits its parent's attributes, operations, and relationships

- A subclass may:
  - Add additional attributes, operations, relationships
  - Redefine inherited operations

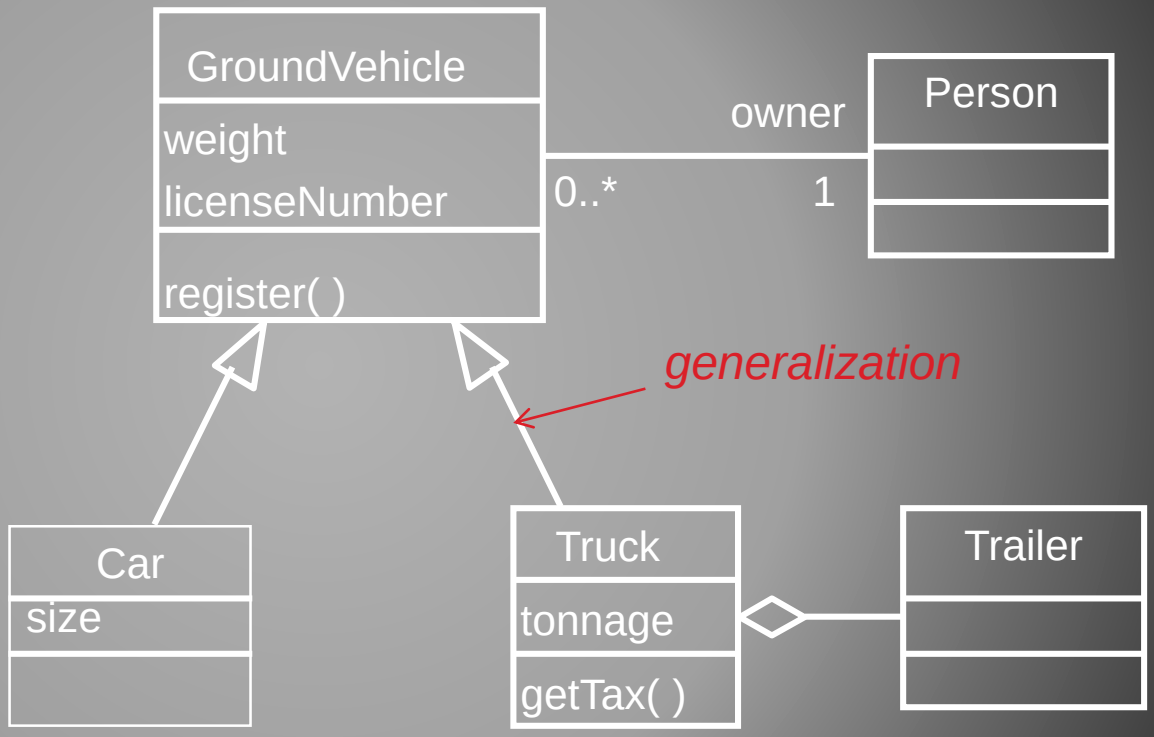
- Common attributes, operations, and/or relationships are shown at the highest applicable level in the hierarchy

Inheritance leverages the similarities among classes

Example: What Gets Inherited

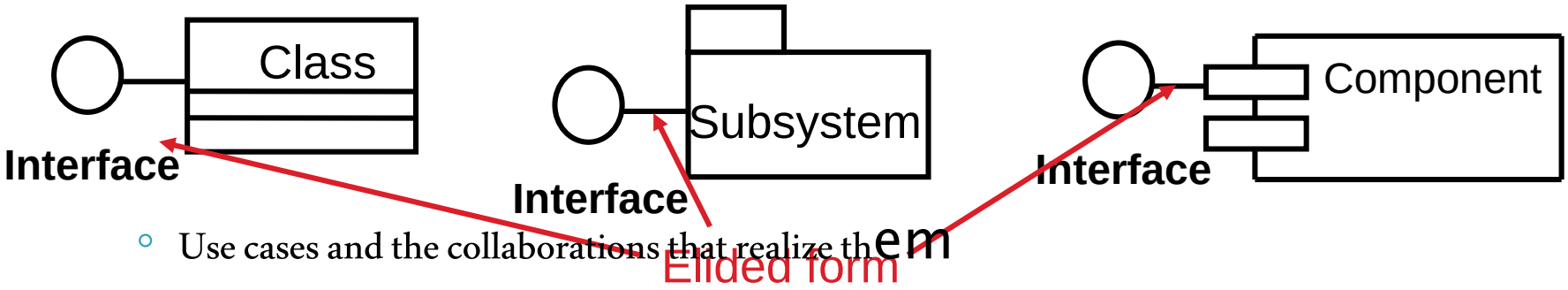
Superclass (parent)

Subclass

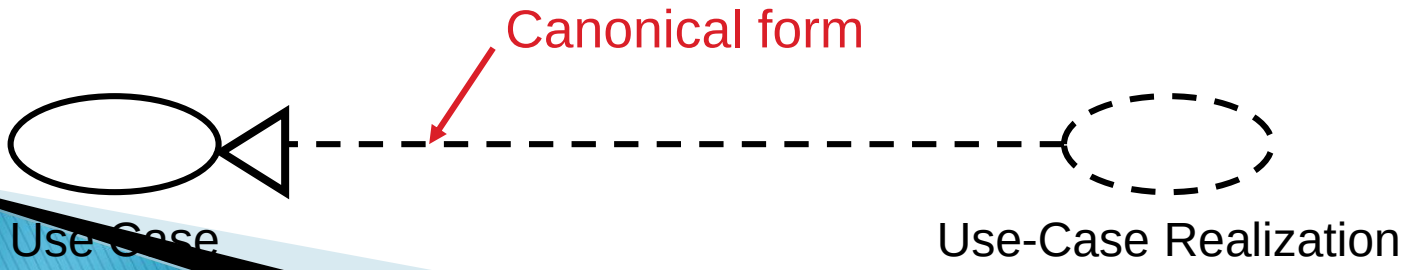


Relationships: Realization

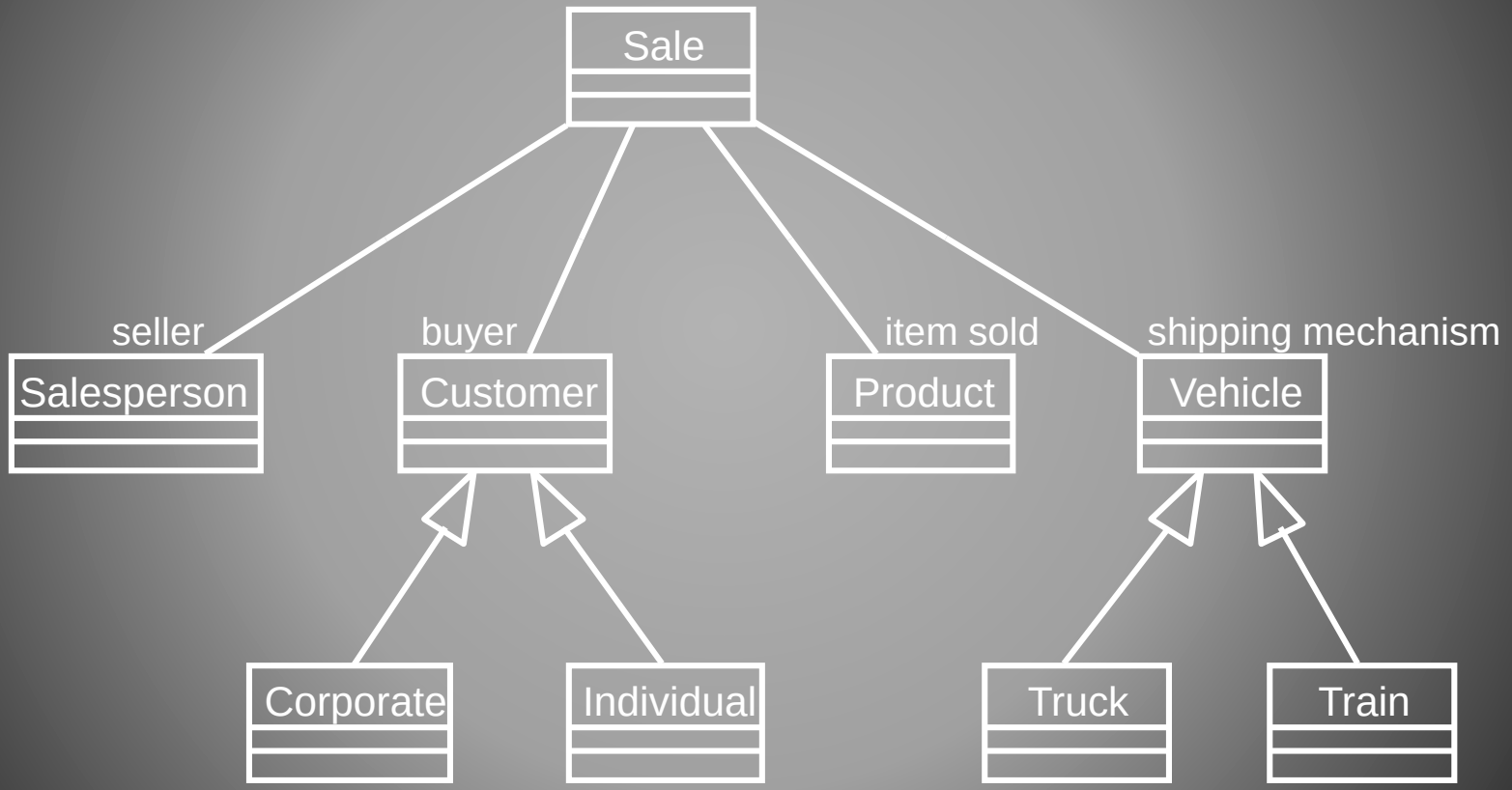
- One classifier serves as the contract that the other classifier agrees to carry out
- Found between:
  - Interfaces and the classifiers that realize them



  - Use cases and the collaborations that realize them

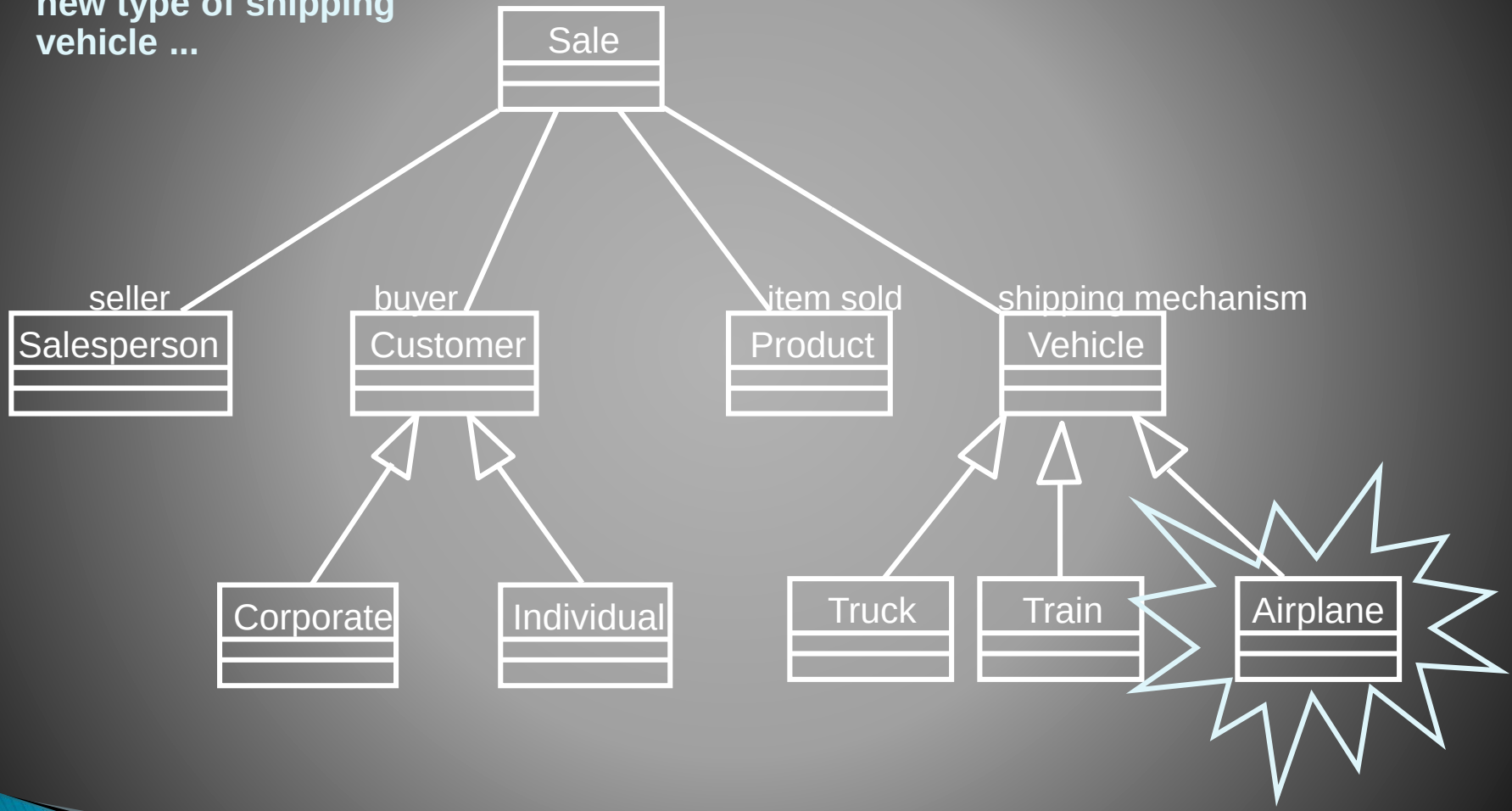


Class Diagram for the Sales Example



Effect of Requirements Change

Suppose you need a new type of shipping vehicle ...



Change involves adding a new subclass

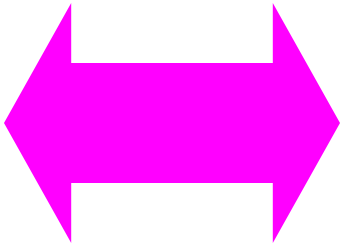
Introduction to Object Orientation Topics

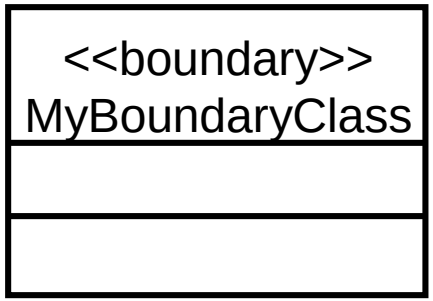
- Basic Principles of Object Orientation
- Basic Concepts of Object Orientation
- Strengths of Object Orientation
- General UML Modeling Mechanisms

Stereotypes

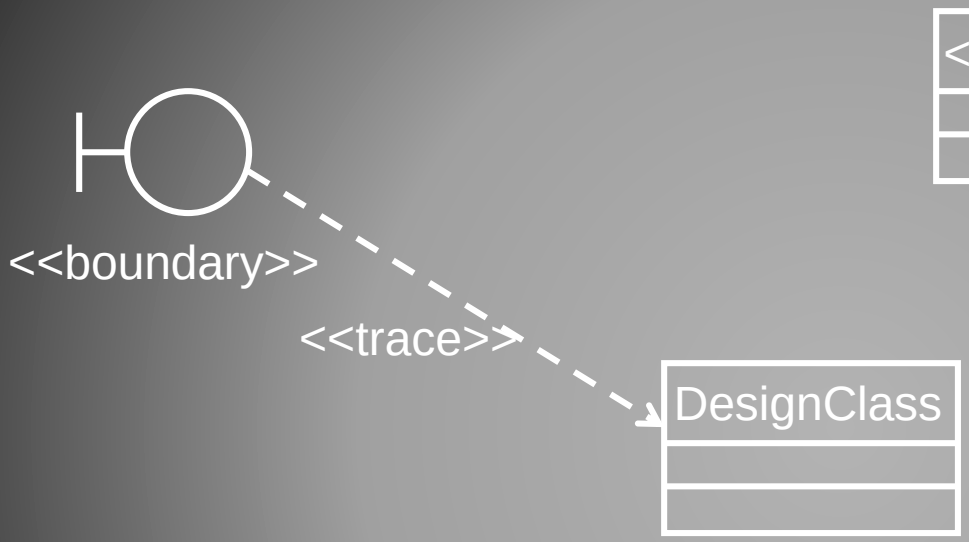
- Classify and extend the UML notational elements
- Define a new model element in terms of another model element
- May be applied to all modeling elements
- Represented with name in guillemets or as a different icon

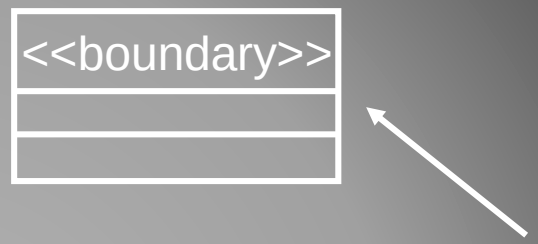
MyBoundaryClass





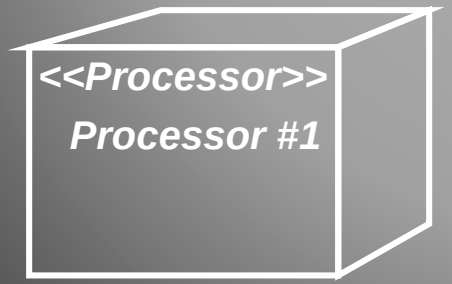
Example: Stereotypes

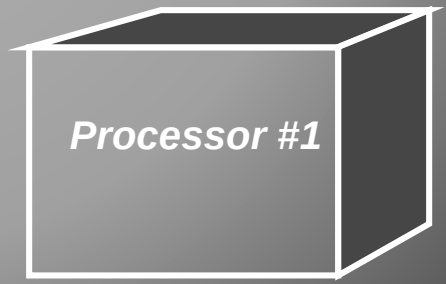




Sample boundary class (stereotype)
Stereotype of 'dependency relation'
Stereotype of <<Processor>>

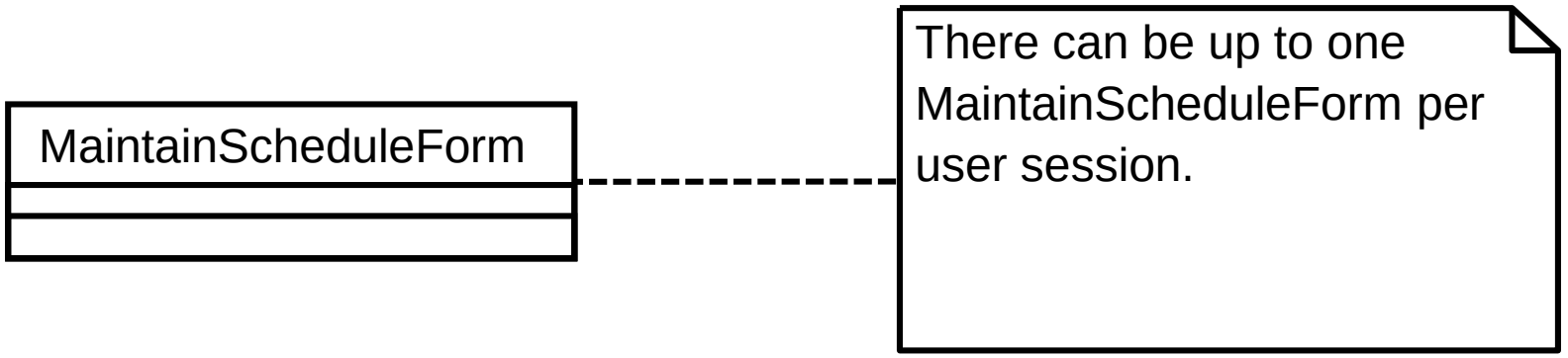
These create new symbols using accustomed graphics.





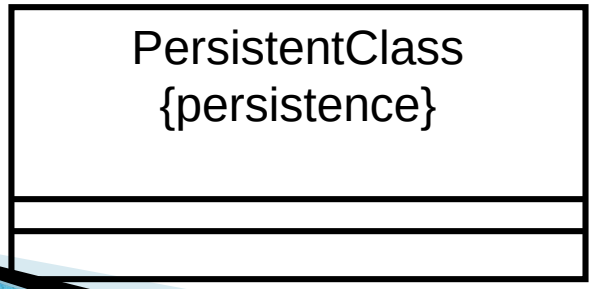
Notes

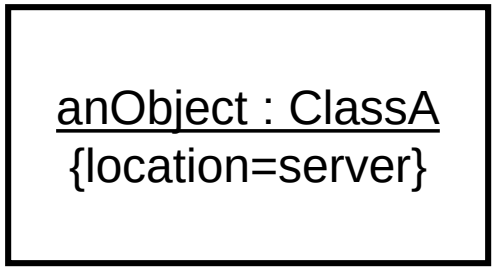
- A note can be added to any UML element
- Notes may be added to add more information to the diagram
- It is a ‘dog eared’ rectangle
- The note may be anchored to an element with a dashed line



Tagged Values

- Extensions of the properties, or specific attributes, of a UML element
- Some properties are defined by UML
  - Persistence
  - Location (e.g., client, server)
- Properties can be created by UML modelers for any purpose



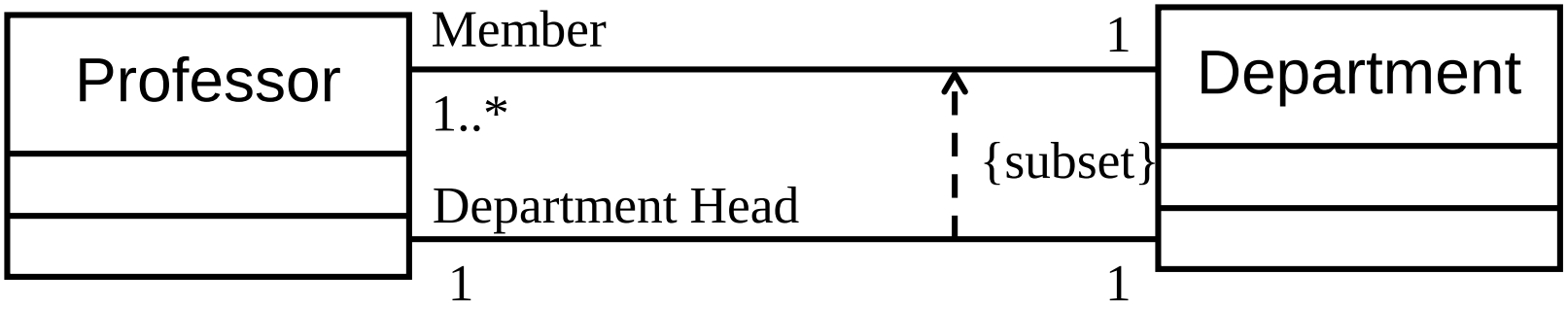


Constraints

- Constraints are functional relationship between the entities and object model. The entities include objects, classes, attributes, association, links.
- A constraint restricts the values that entities can assume.
- UML doesn't specify a particular syntax for constraints, other than they should appear between braces, so they may therefore be expressed using natural language, pseudo code, navigation expression or mathematical expression
- UML1.2 does prefer the use of a constraint language OCL i.e. Object Constraint Language, which is subset of UML.

Constraints

- Supports the addition of new rules or modification of existing rules



This notation is used to capture two relationships between Professor-type objects and Department-type objects; where one relationship is a subset of another….

Shows how UML can be tailored to correctly modeling exact relationships….

Examples: Constraints

- **Number of withdrawal transaction should be less than five per day.**

Constraint on the same class.

Transaction

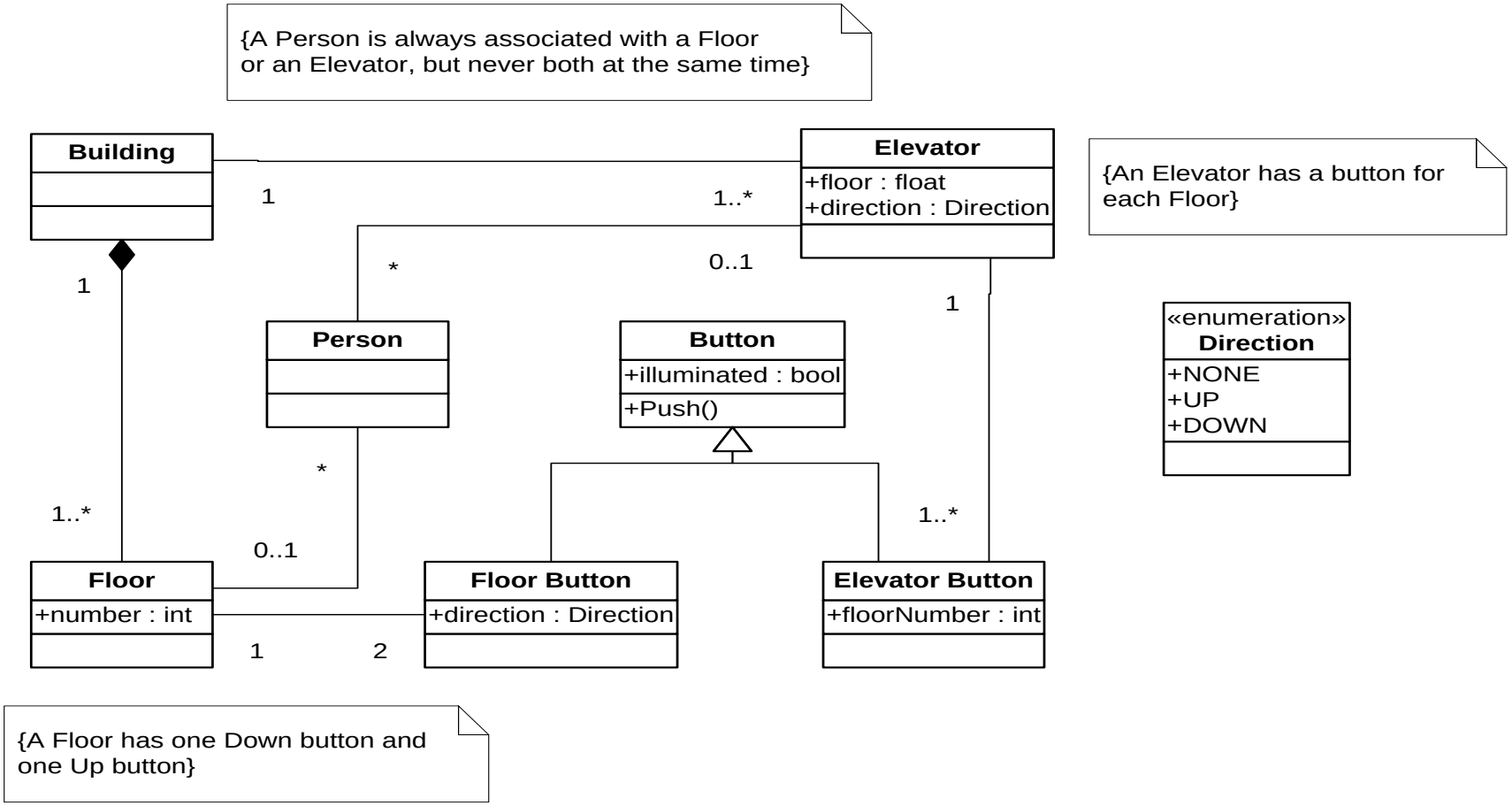
{No. of transaction <=5 /day}

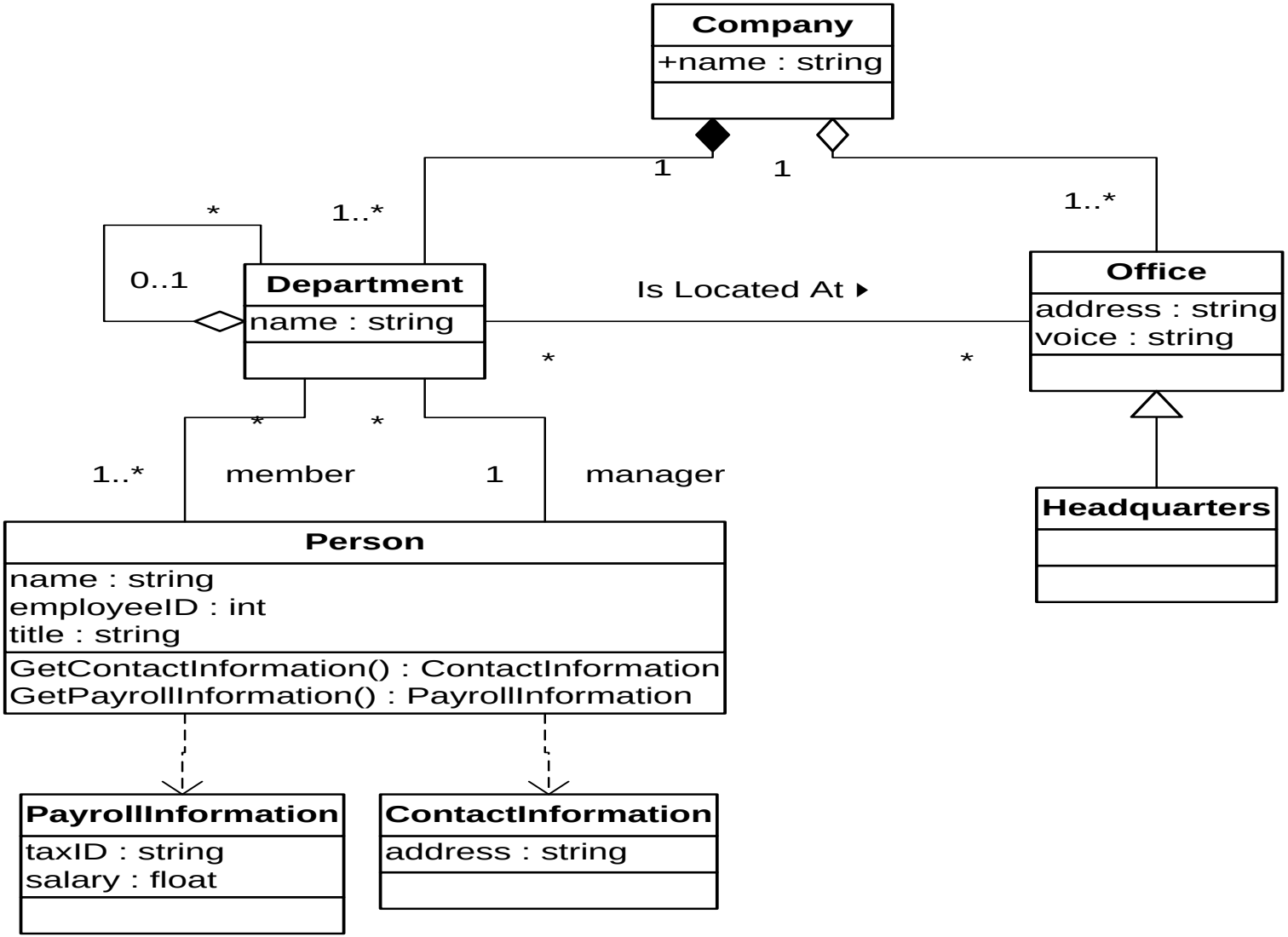
- No window will have an aspect ratio i.e. (length/width) of less than 0.8 or > 1.5

Window length/width

{0.8<=length/width <= 1.5}

Example: Class Diagram




Company
+name : string
1
1
1..*
*
1..*
0..1
Department
name : string
Is Located At
Office
address : string
voice : string
*
*
*
*
1..*
member
1
manager
Headquarters
Person
name : string
employeeID : int
title : string
GetContactInformation() : ContactInformation
GetPayrollInformation() : PayrollInformation
PayrollInformation
taxID : string
salary : float
ContactInformation
address : string

Thanx !!!