California State University, Sacramento
Computer Science Department

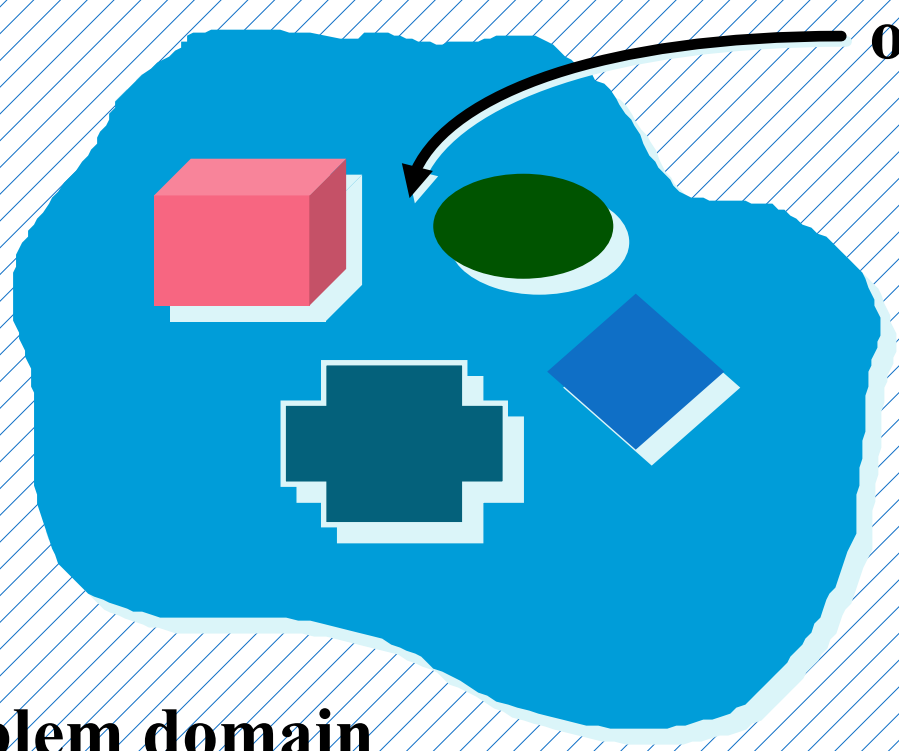
CSc 131

Fall 2022

Lecture # 8

Object-Oriented Design & UML Class Models

The OO Mindset



Classes

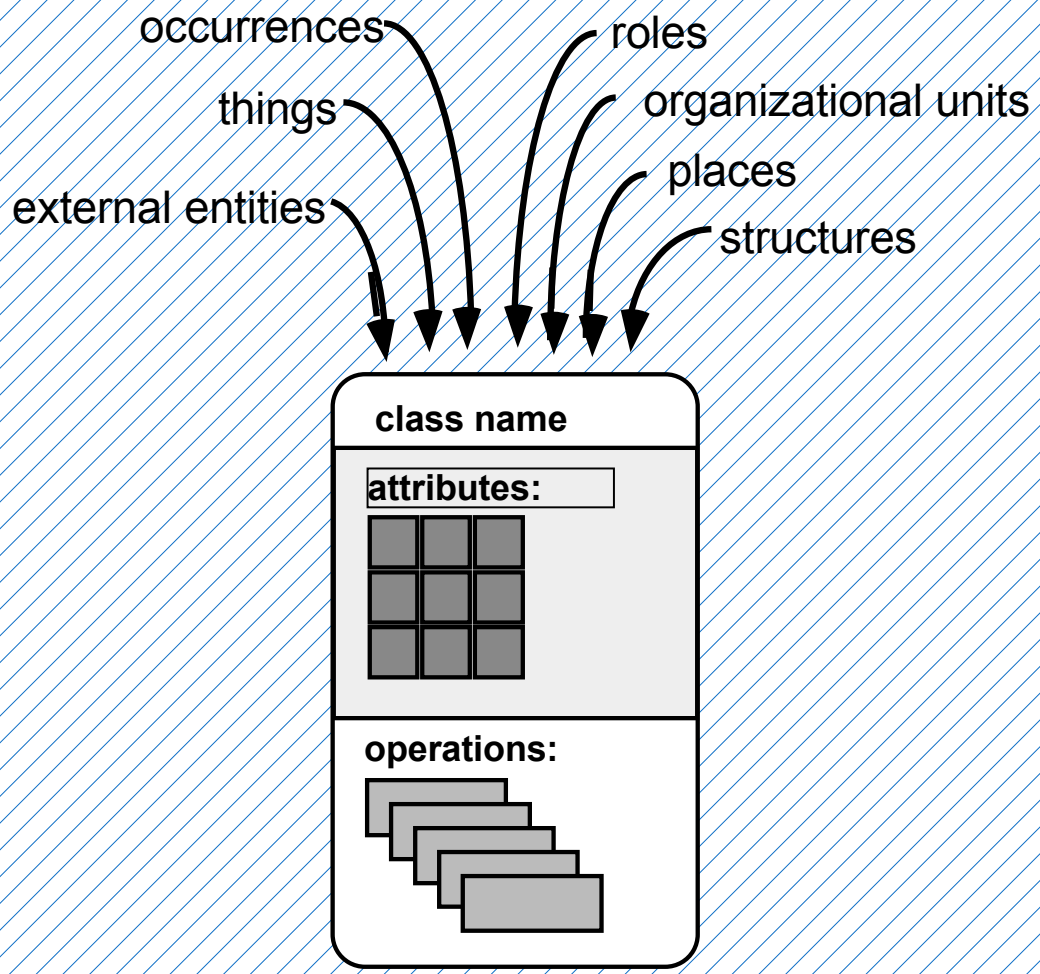
- Object-oriented thinking begins with the definition of a class often defined as:
  - template
  - generalized description
  - “blueprint” ... describing a collection of similar items

- A metaclass (also called a superclass) is a collection of classes.

What is a Class?

- A class is a description of a set of objects that share the same *attributes*, *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.

What is a Class?



What is an Object?

- Objects are key to understanding *object-oriented* technology.
- Many examples of real-world objects: your desk, your computer, your bicycle.
- These real-world objects share two characteristics: *state* and *behavior*.

Objects

Software objects are modeled after real-world objects in that they too have *state and behavior.*

- A software object maintains its state in one or more *variables*.
- A software object implements its behavior with *methods.*
- A method is a function (subroutine) associated with an object.

Object Types

- External entities: sensors, actuators, control panel, devices
- Information items : displays, commands, etc.
- Entities which establishes the context of the problem : controller, monitors, schedulers

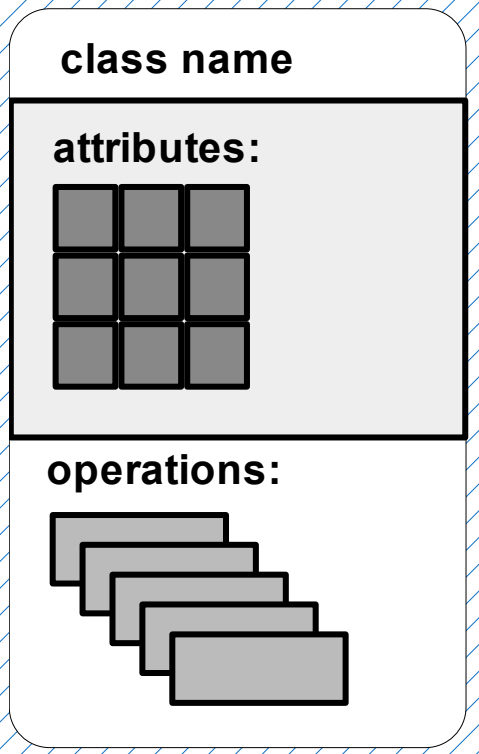
Identifying Classes

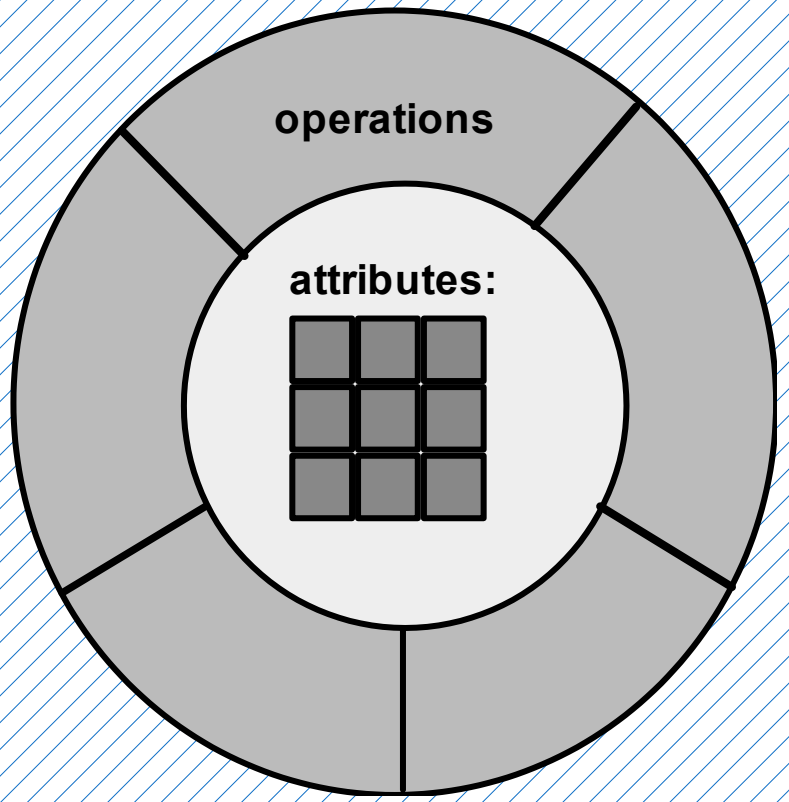
- An object may appear as a noun (ex. Measurement) or disguised in a verb (to measure)
- A method might appear as a verb (ex. Investigate) or disguised in a noun (investigation)
- Attributes describe some kind of characteristics for the object (adjectives).

Criteria for Evaluating Candidate Classes

- One or more attributes
- Needed functionality / Services (one or more methods)
- Common attributes (apply to all objects of a specific class)
- Common functionality/operations (apply to all objects of a specific class)

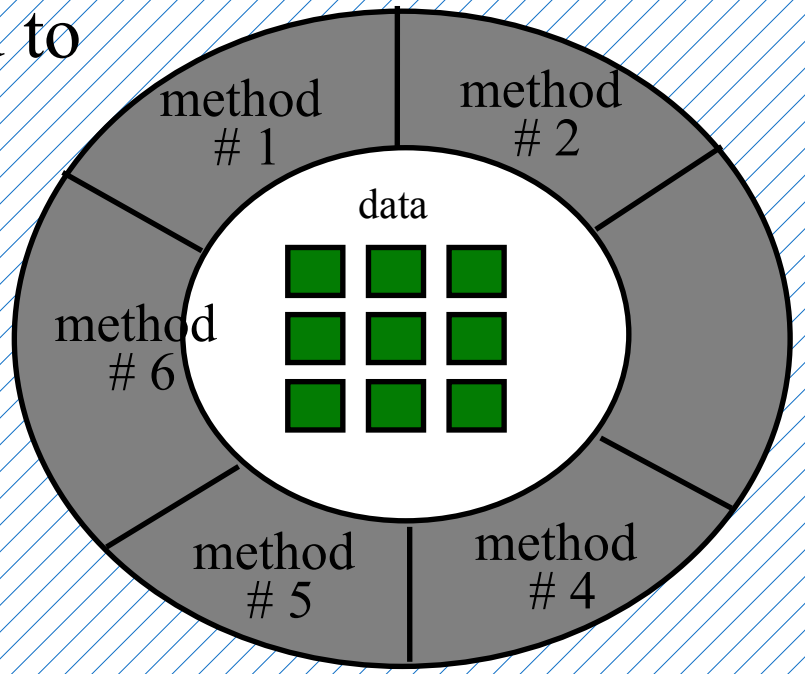
Building a Class





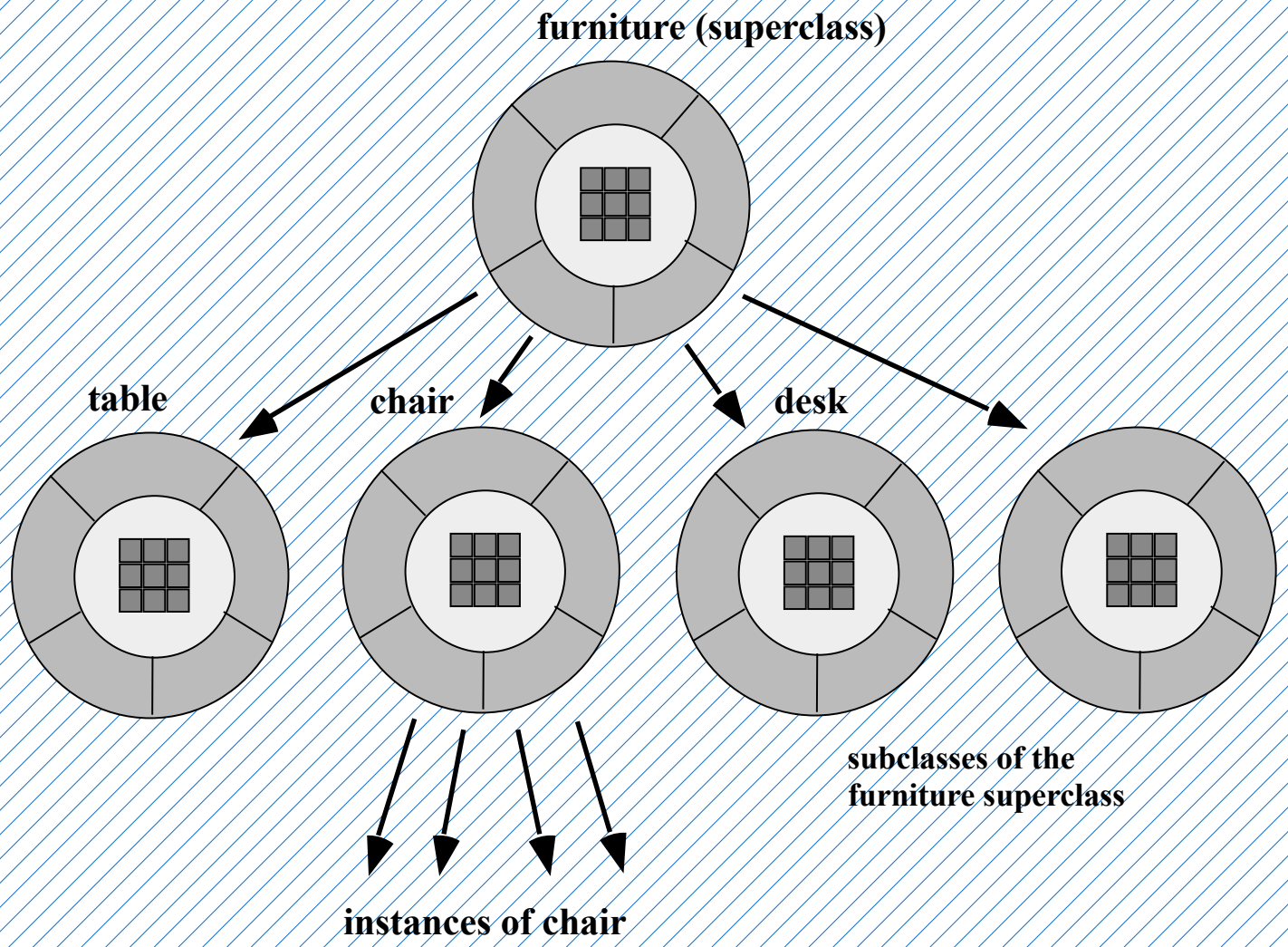
Encapsulation/Hiding

The object encapsulates both data and the logical procedures required to manipulate the data



Achieves “information hiding”

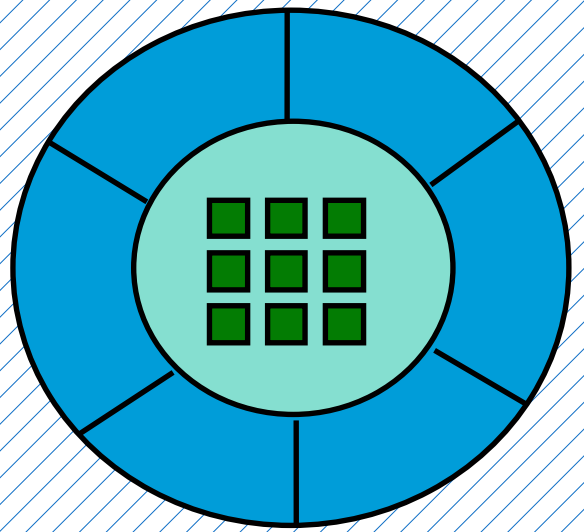
Class Hierarchy



Methods
(a.k.a. Operations, Services)

An executable procedure that is encapsulated in a class and is designed to operate on one or more data attributes that are defined as part of the class.

A method is invoked via message passing.



Encapsulation and Inheritance

- A class ***encapsulates*** data and operations that manipulates the data in a single package

- Benefits of encapsulation:
  - implementation details are hidden (information hiding)- reduces the propagation of side effects when changes occur.

  - Data structure and operations are in a single entity (class) – component reuse

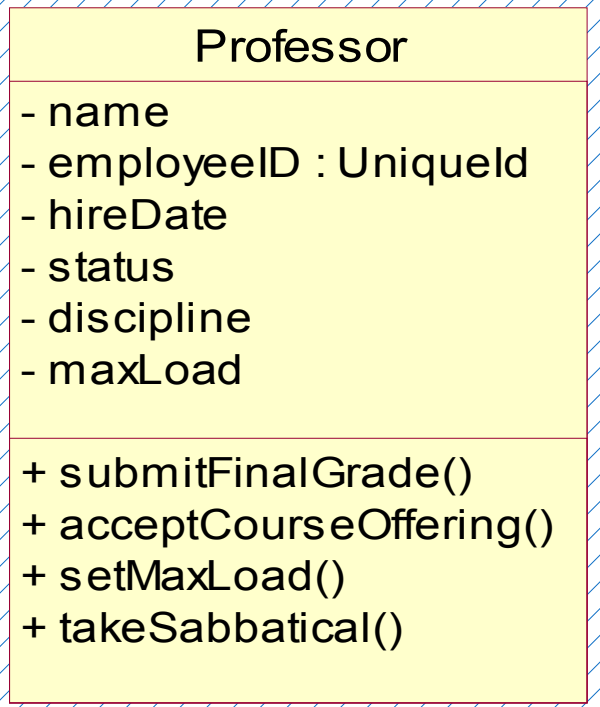
  - Object interfaces are simplified – by message passing and objects are not concerned with the details of internal data structures.

Inheritance

- It is a key concept in OO paradigm
- A subclass inherits all of the attributes and operations associated with its superclass.
- It differentiates between conventional and OO development.
- This implies that all data structures and algorithms that implemented for superclass are available for the subclass- ***Reuse is accomplished.***

Representing Classes in UML

- A class is represented using a rectangle with compartments.

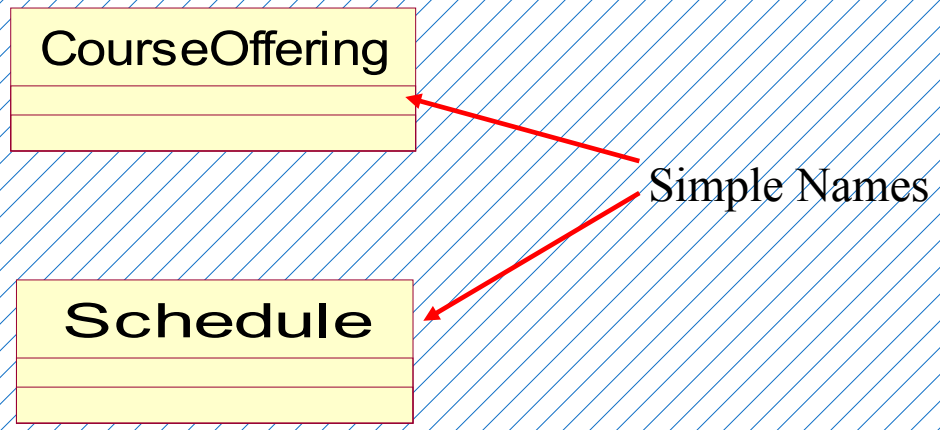


What Should You Name Your Classes?

Name classes to reflect what they represent.

The name should be a noun and not have a prefix or suffix.

Typically, capitalize the first letter in every word in the class name.

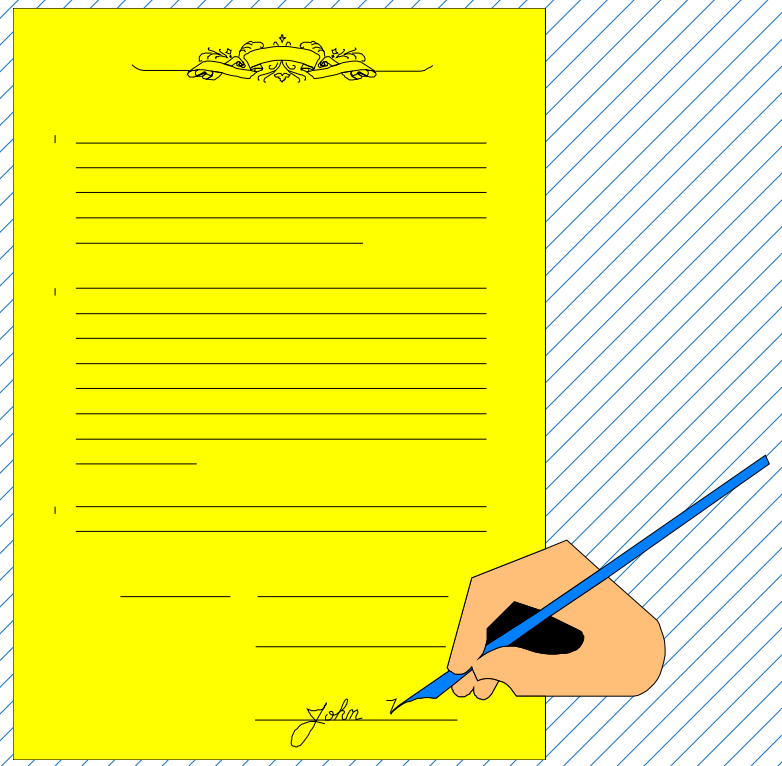


What is Class Responsibility?

A responsibility is a contract or obligation of the class.

The class responsibilities are carried out by the corresponding attributes and operations.

Responsibilities translate into operations and attributes as models are refined.



Classes Objects Need to Collaborate

Objects are useless unless they can collaborate together to solve a problem.

Each object is responsible for its own behavior and status.

No one object can carry out every responsibility on its own.

How do objects interact with each other?
They interact through messages.

Where do you find Classes?

Problem Statement

Use-case documentation

Use-case models

Glossary

Stakeholder Requests

Supplementary Specification

Vision document

Any other project documentation

How do you filter Nouns?

Traditional filtering nouns approach

Underline noun clauses in the use-case flow of events

Remove:

- Redundant candidates
- Vague candidates
- Actors (out of scope)
- Implementation constructs
- Operations

Pitfalls When filtering Nouns

When identifying nouns, be aware that

- Several terms may refer to the same object.
- One term may refer to more than one object.
- Natural language is very ambiguous.
- Any noun can be disguised as a verb and any verb can be disguised as a noun.
- Results are dependent on the author's writing skills.

Filtering nouns can identify many unimportant objects.

Find Classes from Use Case Behavior

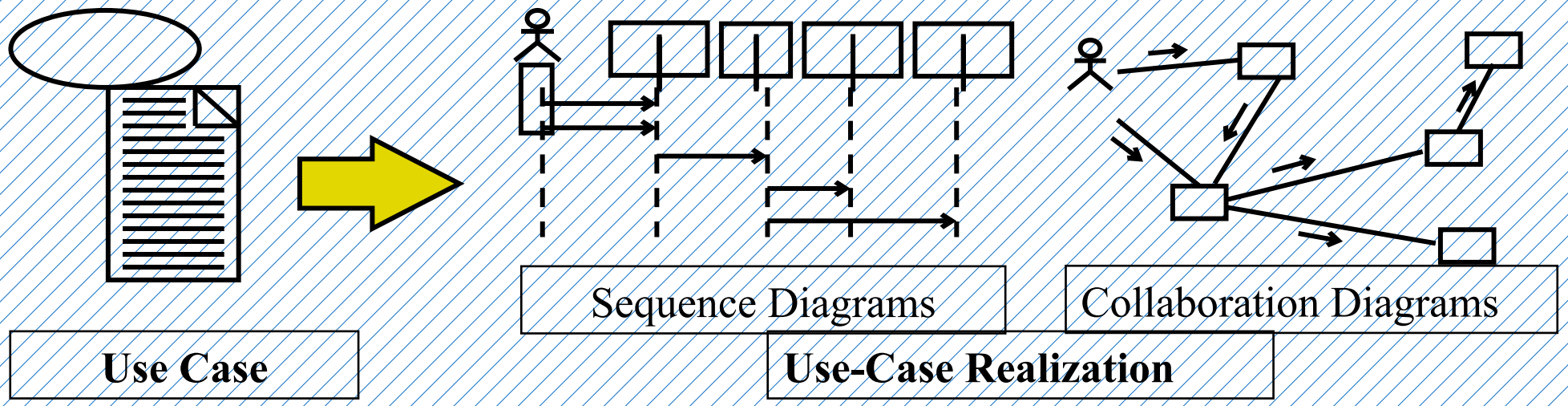
Identify a candidate set of model elements (classes) which are capable of performing the behavior described in use cases.

The complete behavior of a use case has to be distributed to classes.

Distribute Use Case Behavior to Classes

For each use-case flow of events:

- Identify classes
- Allocate use-case responsibilities to classes
- Model class interactions in interaction diagrams



Distribute Use Case Behavior to Classes

Purpose of this step:

To express the use-case behavior in terms of collaborating classes.

To determine the responsibilities of classes.

What is an Association Name?

To clarify its meaning, an association may be named.

The name is represented as a label placed midway along the association line.

An association name is usually a verb or verb phrase.

Good and Bad Examples of Association Names

Poor examples

A student *has a* schedule

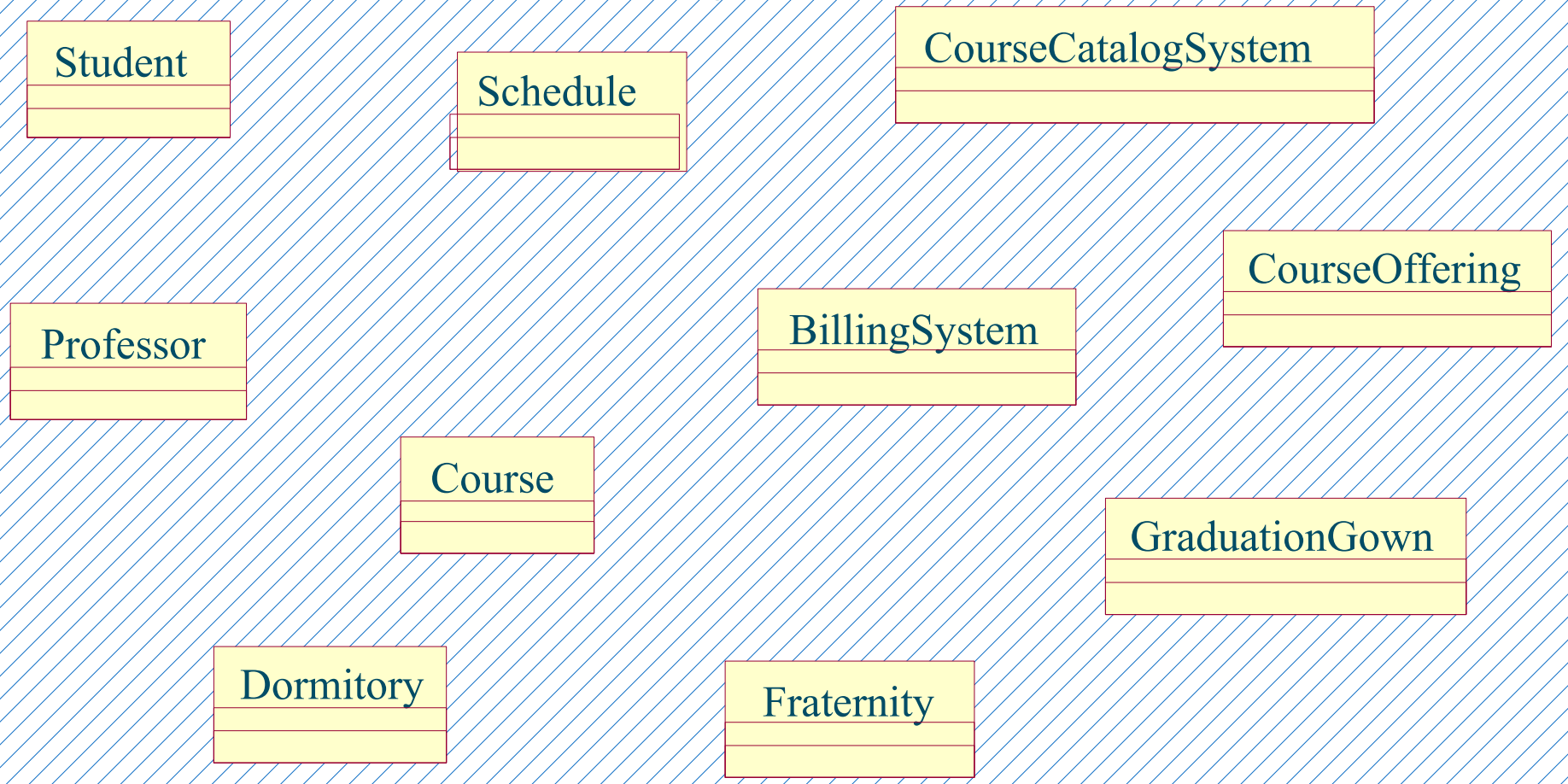
A department *contains a* professor

Good examples

A student *creates* a schedule

A department *employs* a professor

Example Classes for University Registration System



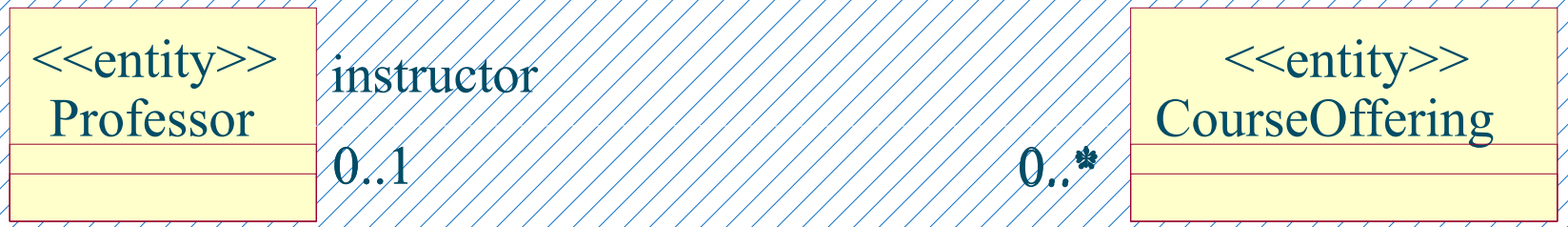
What is Multiplicity?

Multiplicity is the number of instances of one class relates to one instance of another class.

For each association, there are two multiplicity decisions to make, one for each end of the association.

For each instance of Professor, many Course Offerings may be taught.

For each instance of Course Offering, there may be either one or zero Professor as the instructor.



Multiplicity Indicators

Exactly one

Zero or more (many, unlimited)

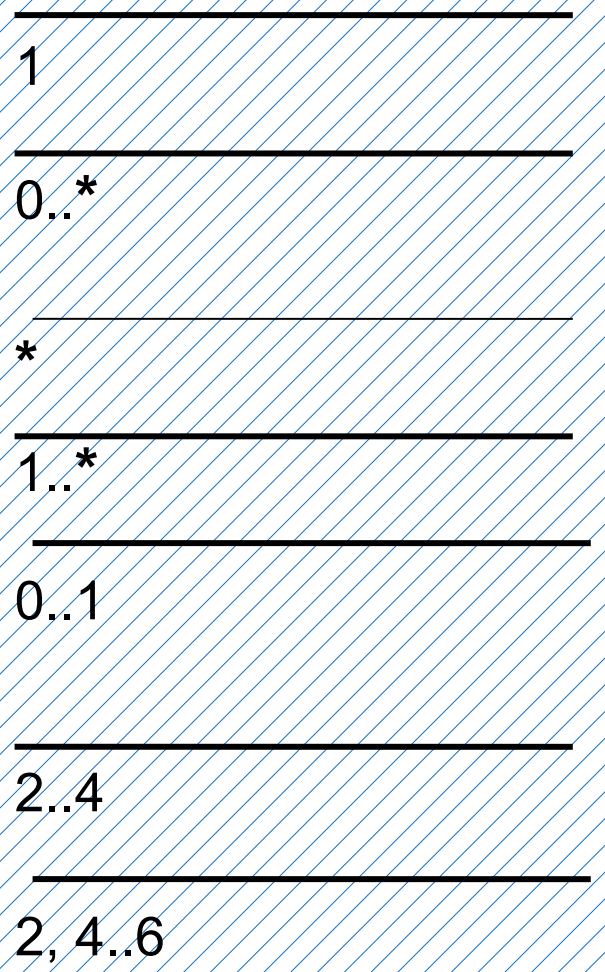
Many

One or more

Zero or one (optional scalar role)

Specified range

Multiple, disjoint ranges

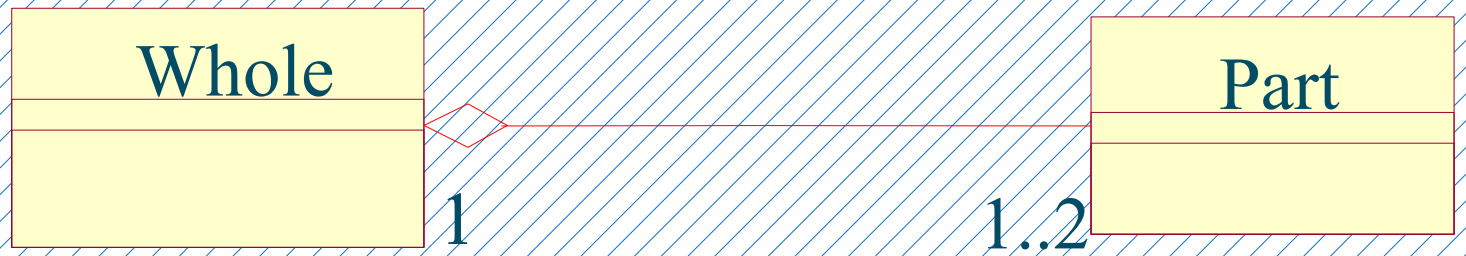


What is Aggregation?

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

An aggregation "Is a part-of" relationship.

Multiplicity is represented like other associations.



Aggregation

Aggregation is also a familiar concept from real life:

a forest is an aggregate of trees

a flock is an aggregate of sheep

Aggregation is a group/member association

The aggregate object may potentially exist without its constituent objects.

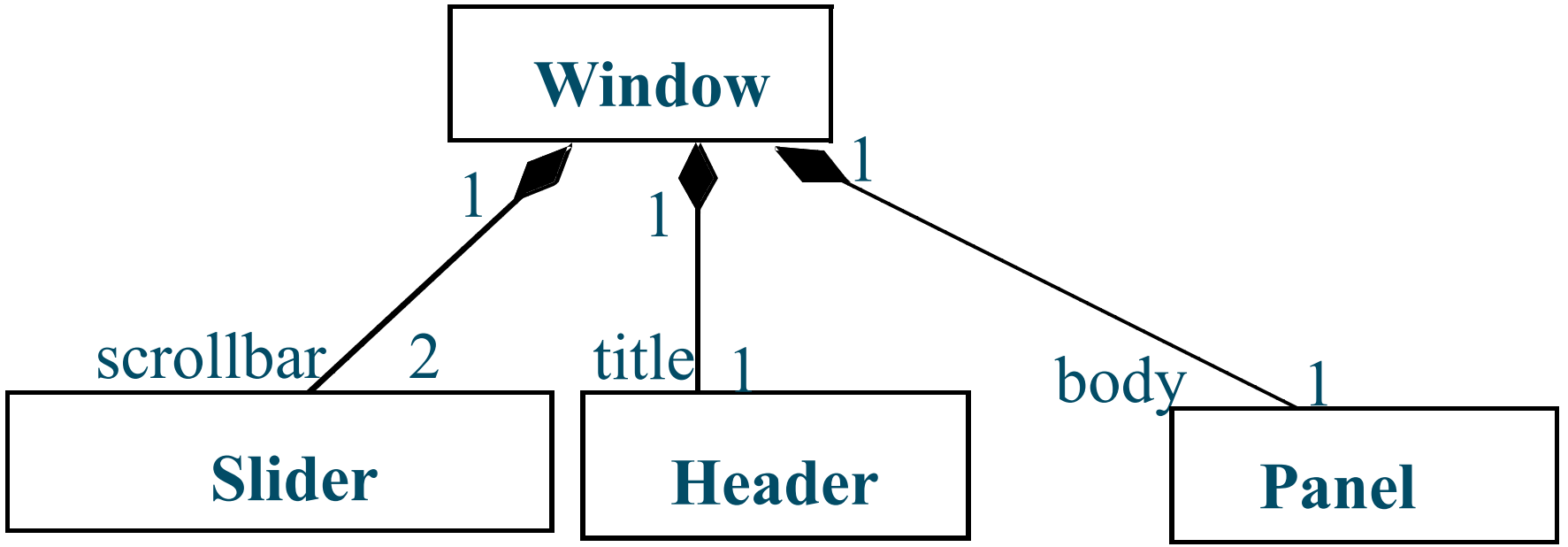
Composition

Composition is a common structure in software systems, because many composite objects appear in real life:

- a dog is a composite of a head, a body, a tail and 4 legs
- an email is composed of a header and a text message; the header is composed of a From: line a To: line, etc.
- The composite object does not exists without its components
- At any time, each component may be part of only one composite.

Composition

A solid diamond denotes ***composition***, a strong form of aggregation, where components cannot exist without the aggregate.



What is Generalization?

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

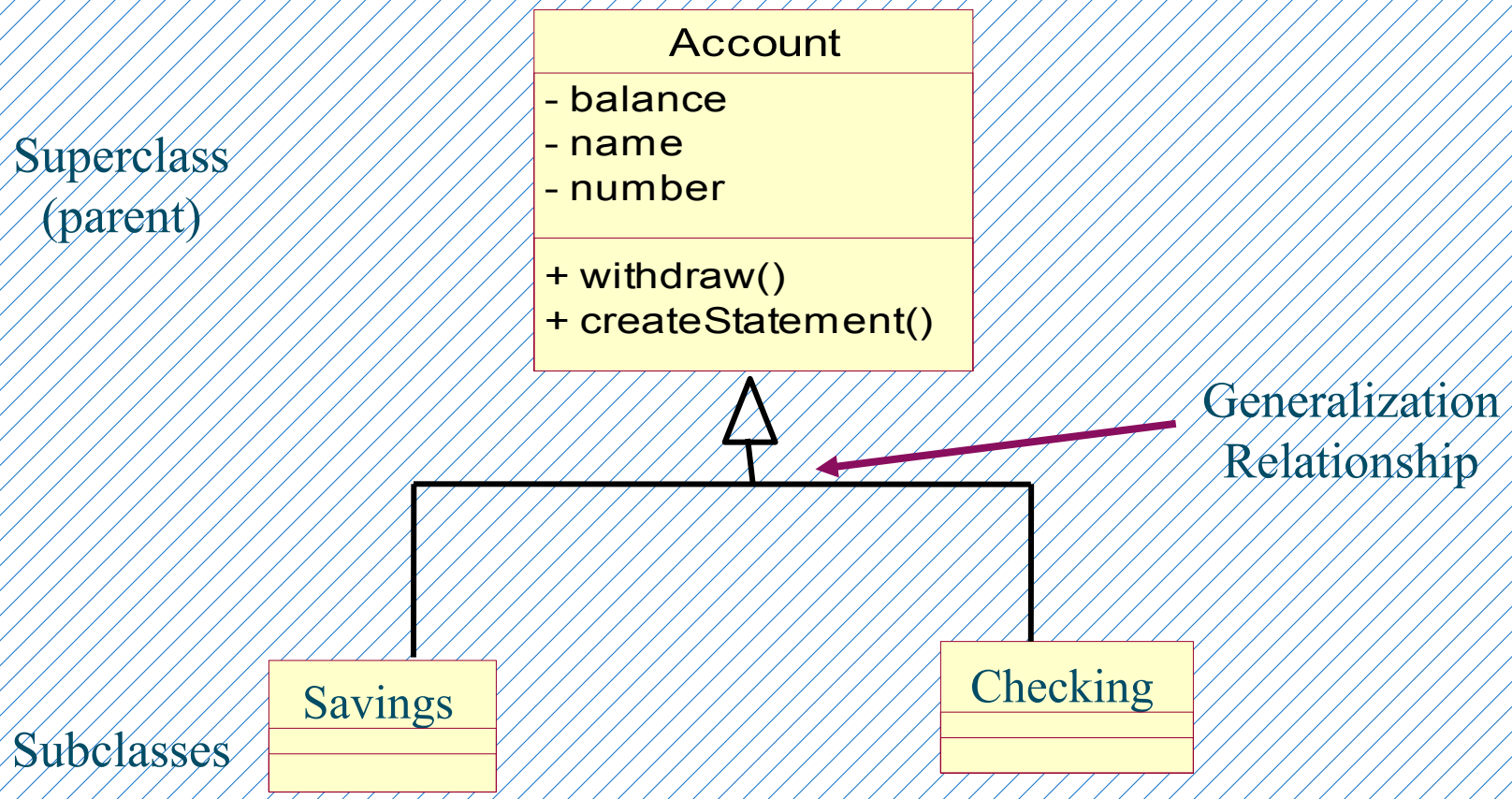
Defines a hierarchy of abstractions in which a subclass inherits from one or more superclasses

- Single inheritance
- Multiple inheritance

Is an “is a kind of” relationship

Example: Single Inheritance

One class inherits from another



Things are Going Well If:

All classes have meaningful, domain-specific names.

Each class has a small set of collaborators.

There are no “indispensable” classes.
A class that collaborates with everyone needs to be redefined.

The classes can handle a change in requirements.

Things are Not Going Well If:

A number of classes have no responsibilities.

A single responsibility gets assigned to several classes.

All classes collaborate with all other classes.

What Next:

- Continue with Object Oriented Design & UML Class Models
- Continue with “Design and implementation..”

Questions ?