UML Class Diagrams and Packages

Hoang Huu Hanh, Hue University
hanh-at-hueuni.edu.vn

Agenda

- What is a Class Diagram?
- Essential Elements of a UML Class Diagram
- Packages and Class Diagrams
- Analysis Classes Approach
- Tips
- Object and Class Constructing

Hue University

What is a Class Diagram?

- A class diagram describes the types of objects in the system and the various kinds of static relationships that exist among them
  - A graphical representation of a static view on static elements
- A central modeling technique that is based on object-oriented principles
- The richest notation in UML

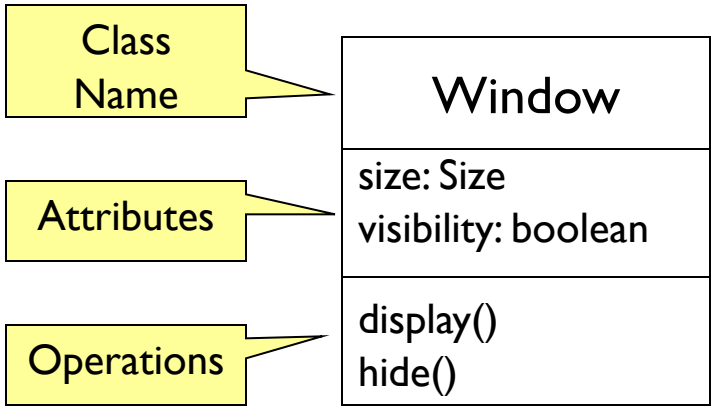
Essential Elements

- Class
- Attributes
- Operations
- Relationships
  - Associations
  - Generalization
  - Dependency
  - Realization
- Constraint Rules and Notes

Hue University

Classes

- A class is the description of a set of objects having similar attributes, operations, relationships and behavior.



Associations

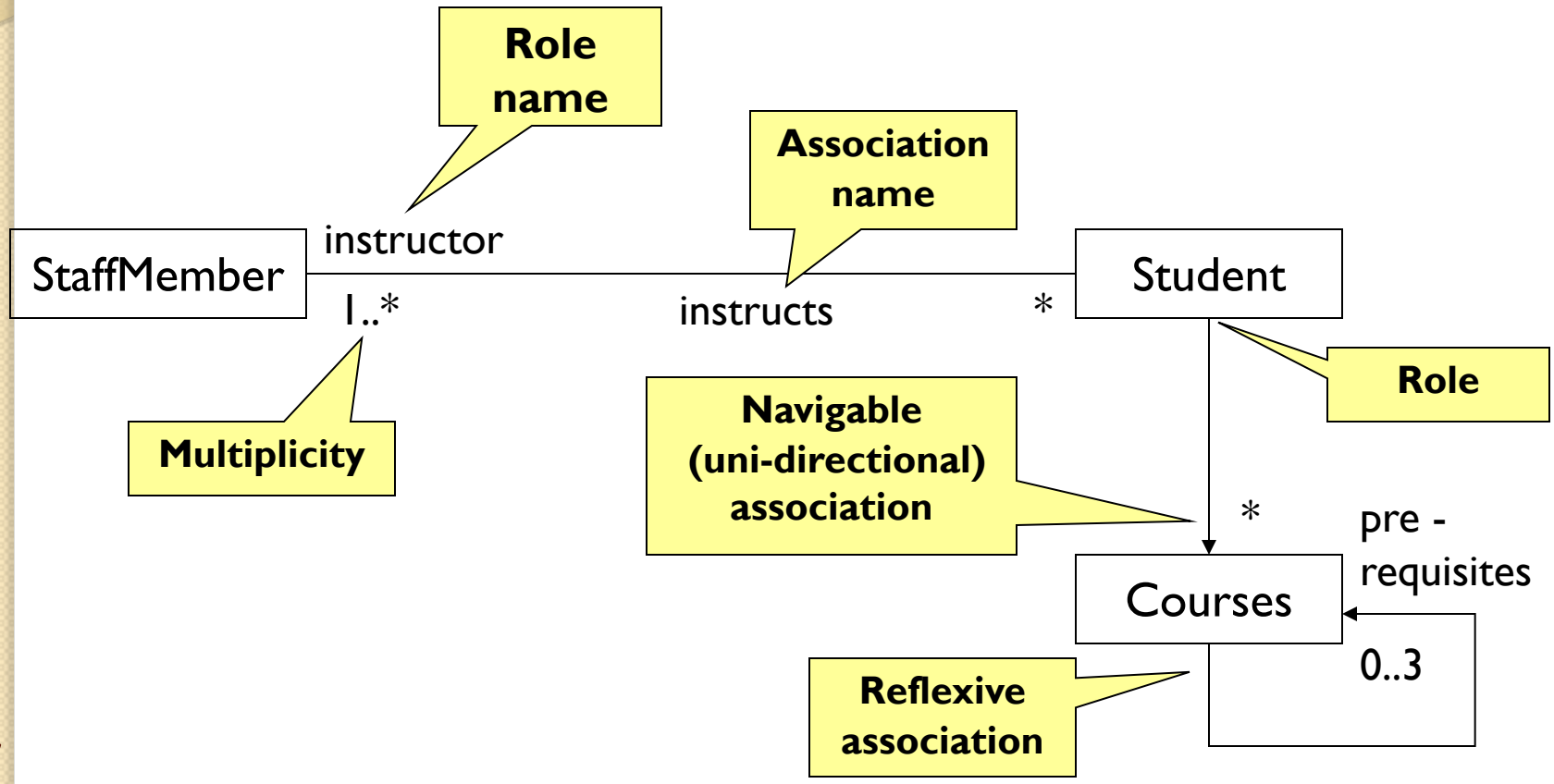
- A semantic relationship between two or more classes that specifies connections among their instances.
- A structural relationship, specifying that objects of one class are connected to objects of a second (possibly the same) class.
- Example: "An Employee works in a department of a Company"

Hue University

Associations (cont.)

- An association between two classes indicates that objects at one end of an association "recognize" objects at the other end and may send messages to them.
  - This property will help us discover less trivial associations using interaction diagrams.

Associations (cont.)



Associations (cont.)

- To clarify its meaning, an association may be named.
  - The name is represented as a label placed midway along the association line.
  - Usually a verb or a verb phrase.
- A **role** is an end of an association where it connects to a class.
  - May be named to indicate the role played by the class attached to the end of the association path.
    - Usually a noun or noun phrase
    - Mandatory for reflexive associations

Hue University

Associations (cont.)

- Multiplicity
  - The number of instances of the class, next to which the multiplicity expression appears, that are referenced by a **single** instance of the class that is at the other end of the association path.
  - Indicates whether or not an association is mandatory.
  - Provides a lower and upper bound on the number of instances.

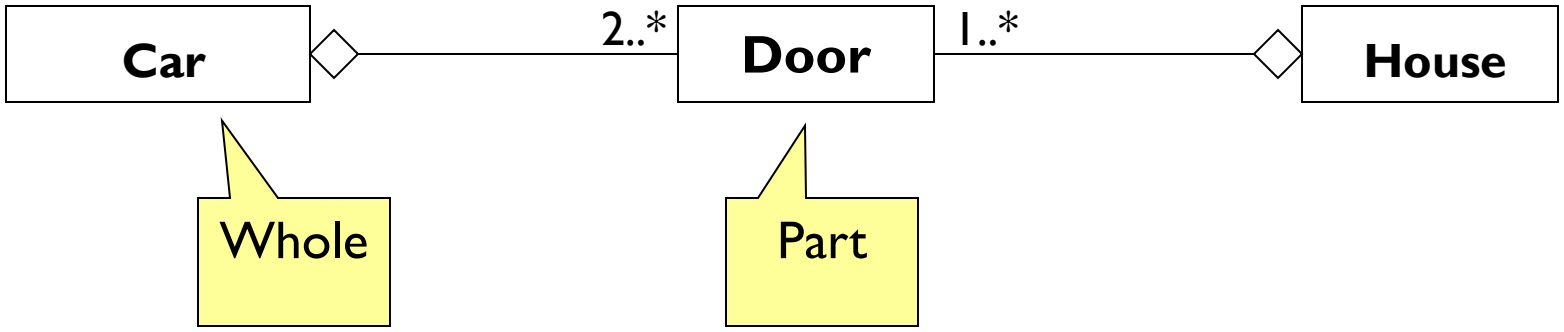
Associations (cont.)

- Multiplicity Indicators

Exactly one	1
Zero or more (unlimited)	* (0..*)
One or more	1..*
Zero or one (optional association)	0..1
Specified range	2..4
Multiple, disjoint ranges	2, 4..6, 8

Aggregation

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



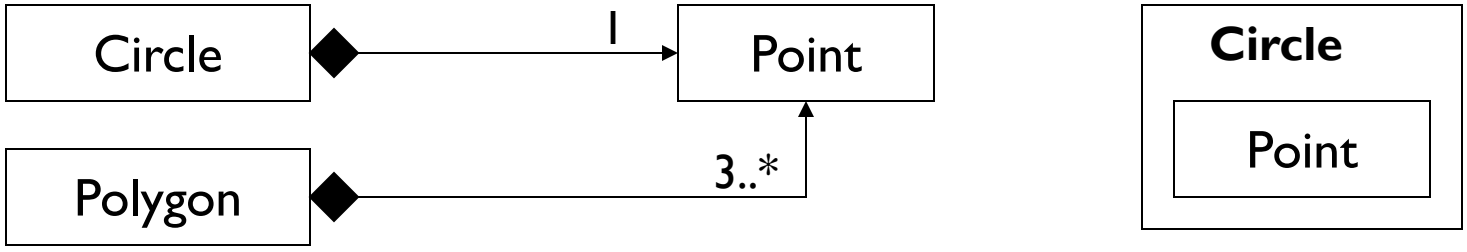
Hue University

Aggregation (cont'd)

- Aggregation tests:
  - Is the phrase "part of" used to describe the relationship?
    - A door is "part of" a car
  - Are some operations on the whole automatically applied to its parts?
    - Move the car, move the door.
  - Are some attribute values propagated from the whole to all or some of its parts?
    - The car is blue, therefore the door is blue.
  - Is there an intrinsic asymmetry to the relationship where one class is subordinate to the other?
    - A door is part of a car. A car is not part of a door.

Composition

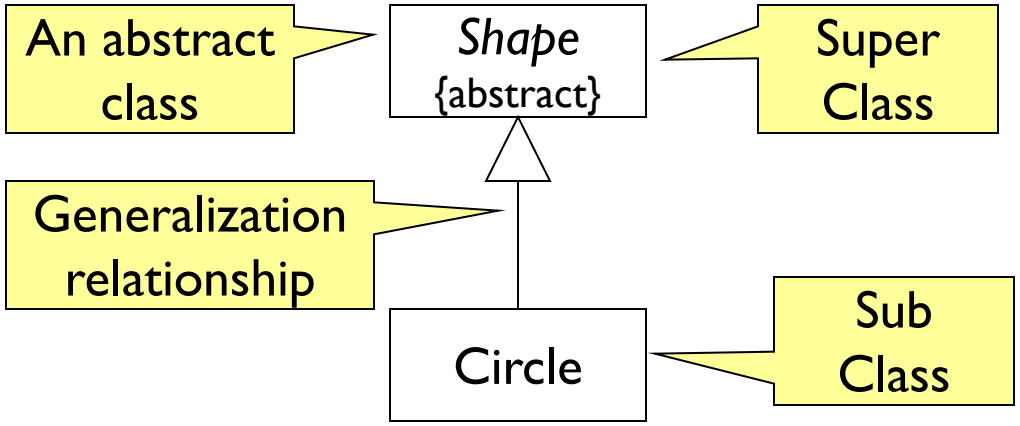
- A strong form of aggregation
  - The whole is the sole owner of its part.
    - The part object may belong to only one whole
  - Multiplicity on the whole side must be zero or one.
  - The life time of the part is dependent upon the whole.
    - The composite must manage the creation and destruction of its parts.



Generalization

- Indicates that objects of the specialized class (subclass) are substitutable for objects of the generalized class (super-class).
  - "is kind of" relationship.

{abstract} is a tagged value that indicates that the class is abstract. The name of an abstract class should be italicized



Hue University

Generalization

- A sub-class inherits from its super-class
  - Attributes
  - Operations
  - Relationships
- A sub-class may
  - Add attributes and operations
  - Add relationships
  - Refine (override) inherited operations
- A generalization relationship may not be used to model interface implementation.

Dependency

- A dependency indicates a semantic relation between two classes although there is no explicit association between them.
- A stereotype may be used to denote the type of the dependency.

Hue University

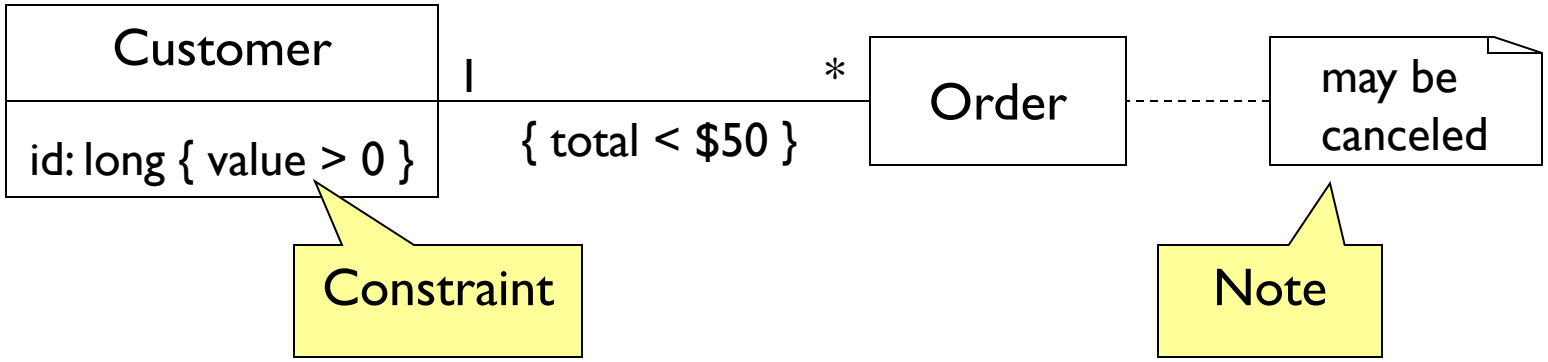
Realization

- A realization relationship indicates that one class implements a behavior specified by some interface
- An interface can be realized by many classes
- A class may realize many interfaces

Hue University

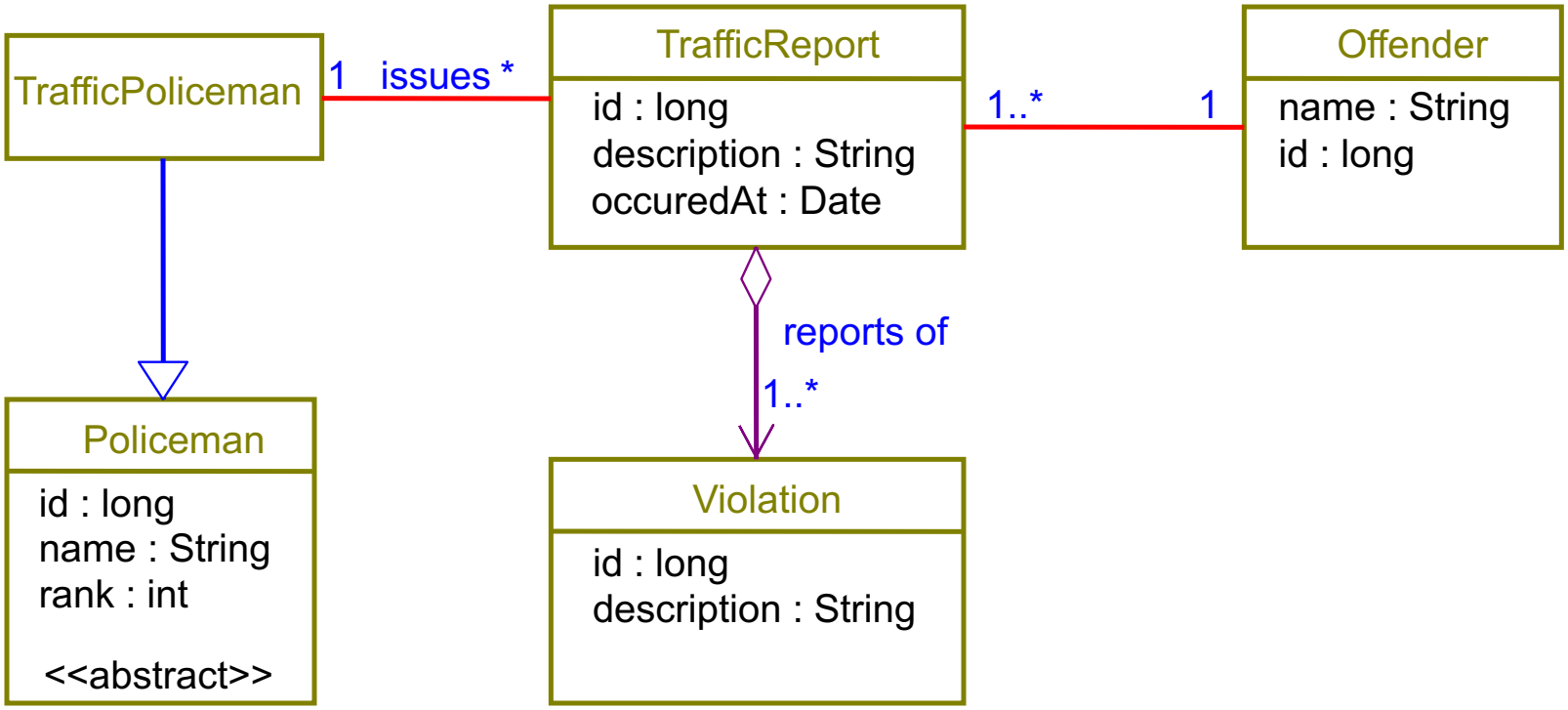
Constraint Rules and Notes

- Constraints and notes annotate among other things associations, attributes, operations and classes.
- Constraints are semantic restrictions noted as Boolean expressions.



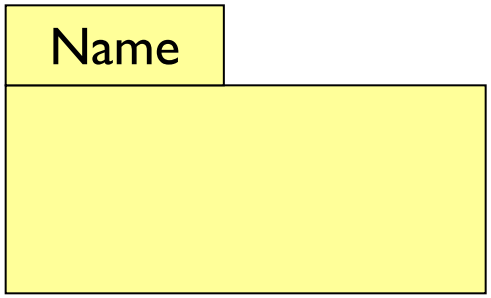
Hue University

TVRS Example



UML Packages

- A package is a general purpose grouping mechanism.
- Commonly used for specifying the logical architecture of the system.
- A package does not necessarily translate into a physical sub-system.



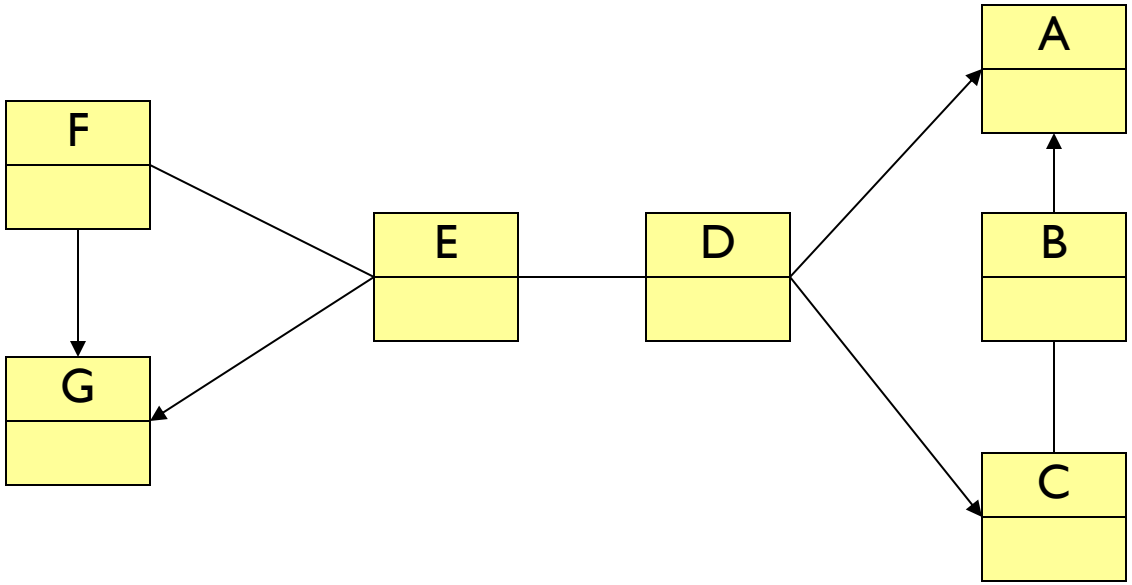
Hue University

UML Packages (cont'd)

- Emphasize the logical structure of the system (High level view)
- Higher level of abstraction over classes.
- Aids in administration and coordination of the development process.

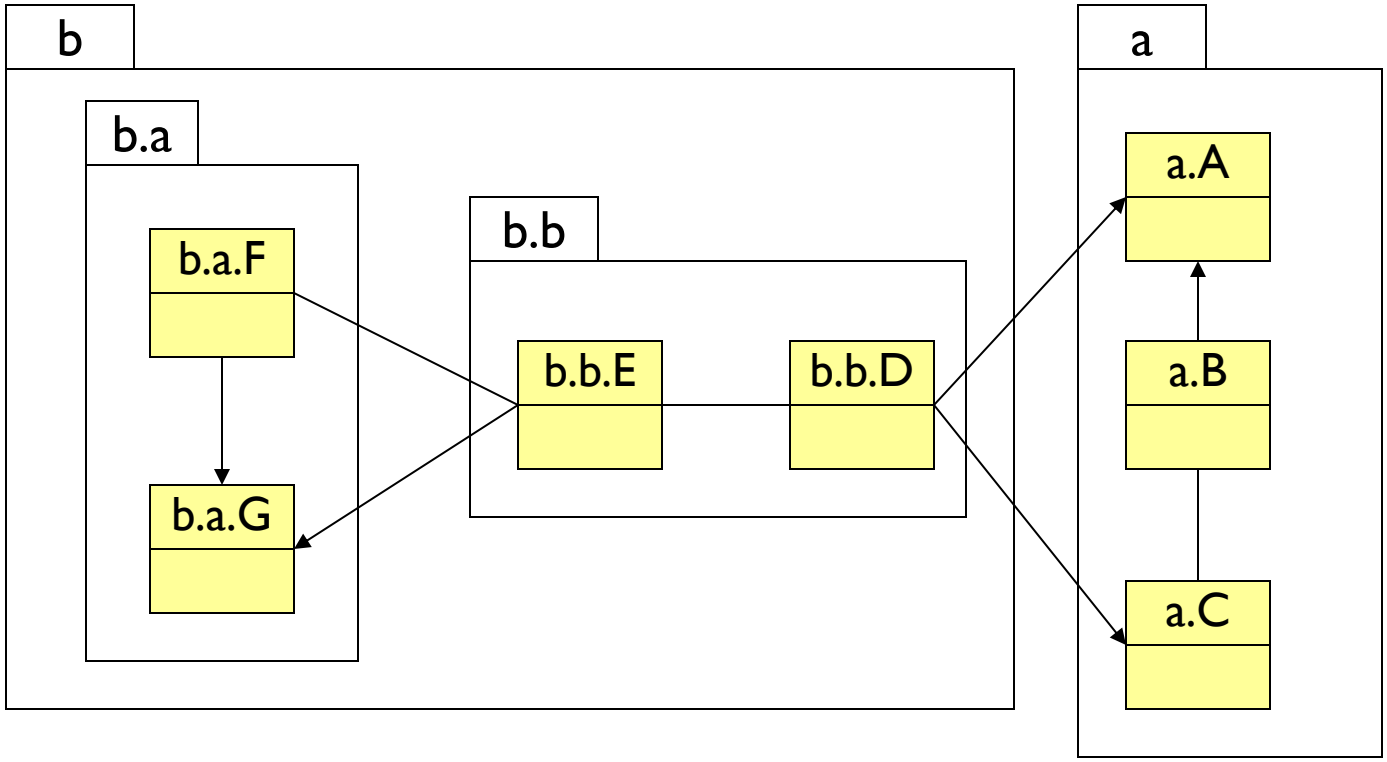
Packages and Class Diagrams

- Add package information to class diagrams



Packages and Class Diagrams

- Add package information to class diagrams



Analysis Classes

- A technique for finding analysis classes which uses three different perspectives of the system:
  - The boundary between the system and its actors
  - The information the system uses
  - The control logic of the system

Hue University

Boundary Classes

- Models the interaction between the system's surroundings and its inner workings
- User interface classes
  - Concentrate on what information is presented
  - Don't concentrate on visual asspects
  - Example: ReportDetailsForm
- System / Device interface classes
  - Concentrate on what protocols must be defined.
  - Don't concentrate on how the protocols are implemented

Hue University

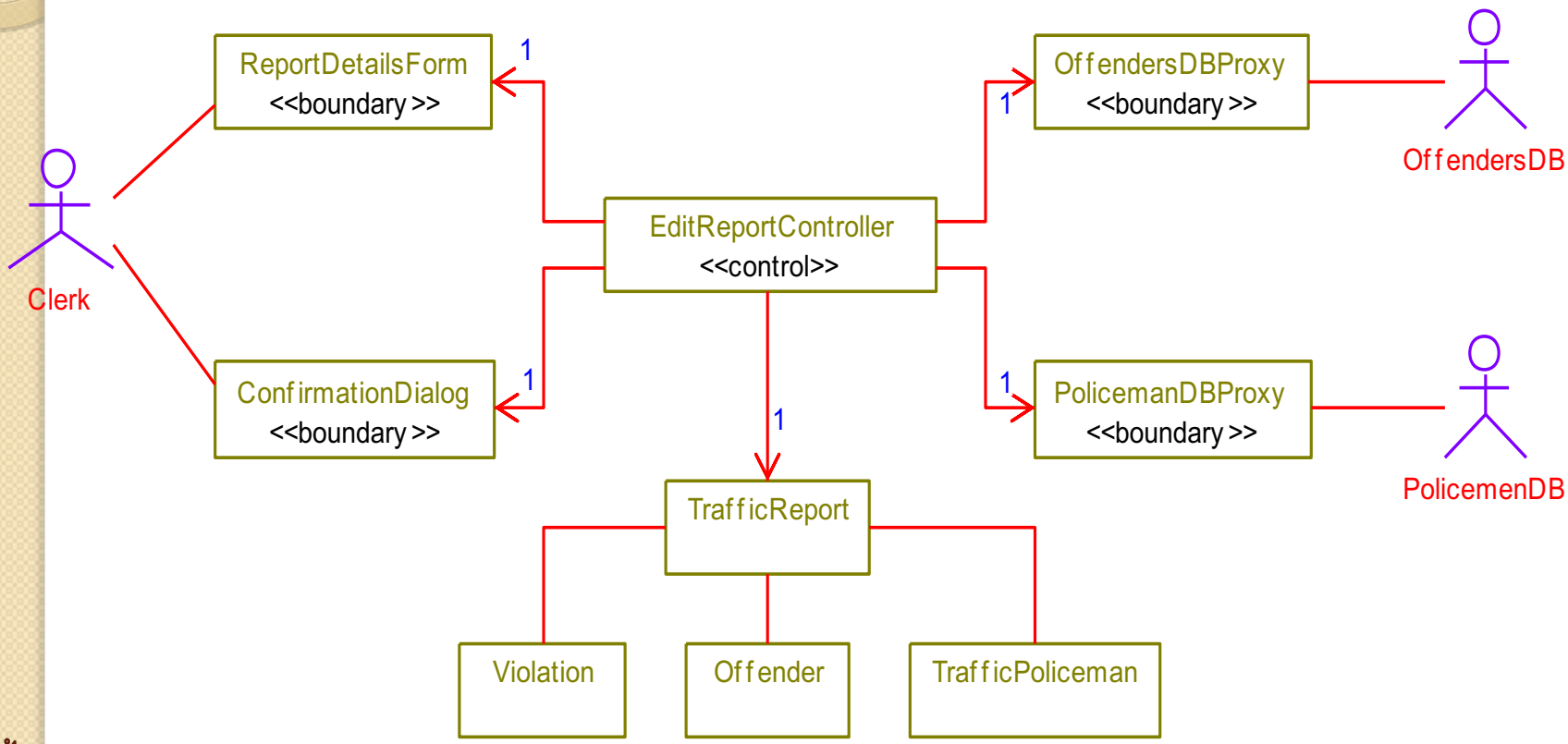
Entity Classes

- Models the key concepts of the system
- Usually models information that is persistent
- Can be used in multiple behaviors
- Example: Violation, Report, Offender.

Control Classes

- Controls and coordinates the behavior of the system
- Delegates the work to other classes
- Control classes decouple boundary and entity classes
- Example:
  - EditReportController
  - AddViolationController

TVRS Example



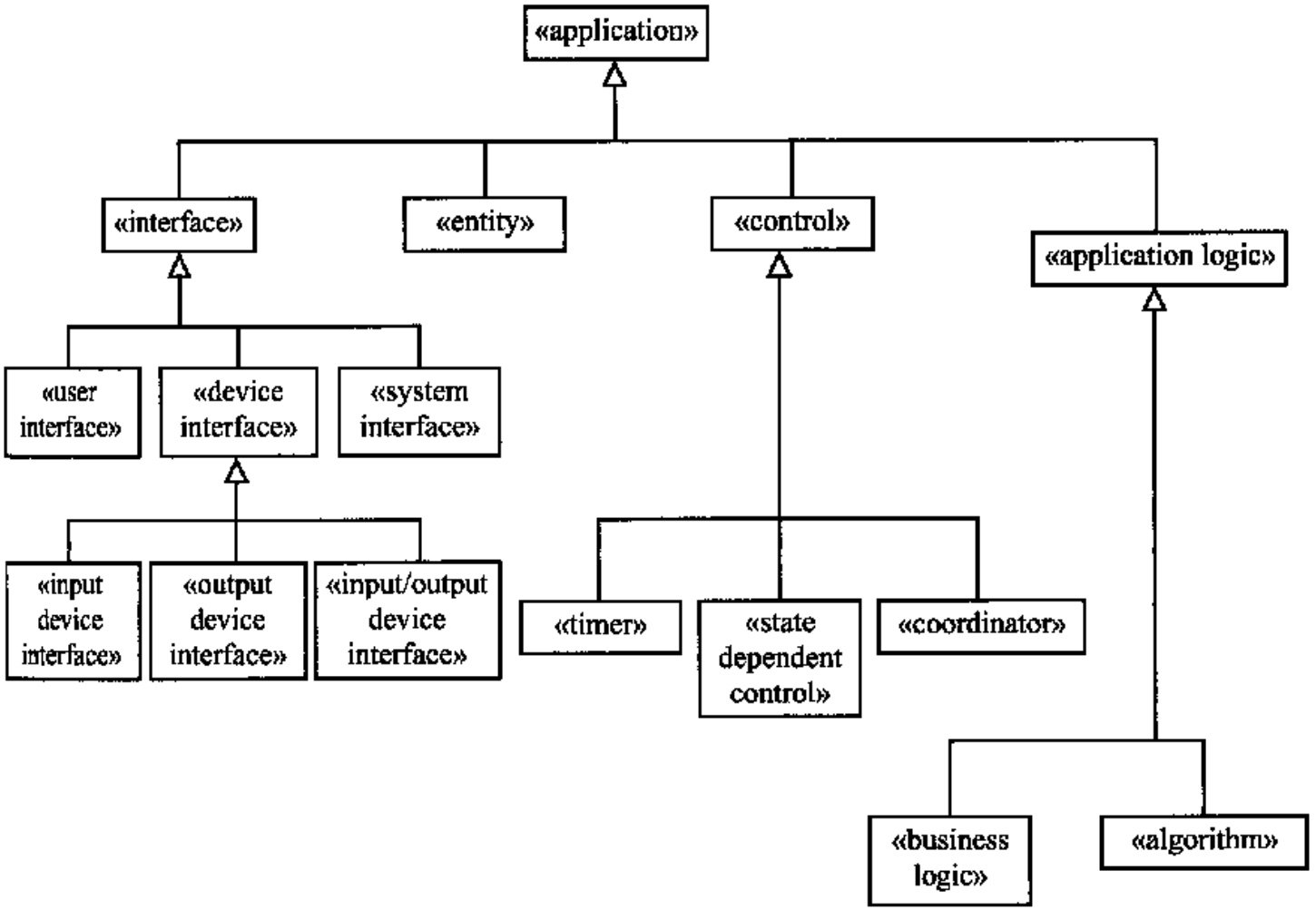
Object and Class Constructing

Hue University

Objectives

- Provide guidelines on how to determine the classes/objects in the system
- Define class/object structuring criteria

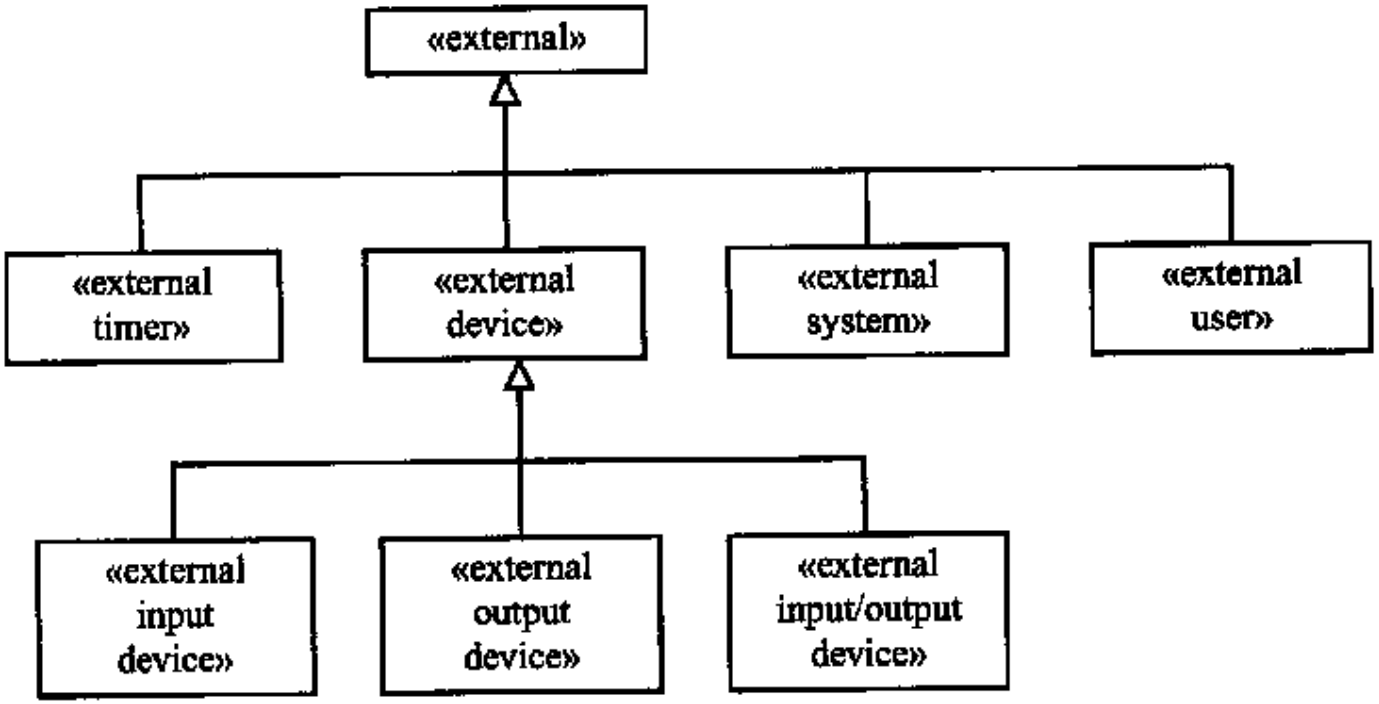
Categorization of Application Classes



External Classes and Interface Classes

- External classes are classes that are external to the system and that interface to the system.
- Interface (boundary) classes are classes internal to the system that interface to the external classes.

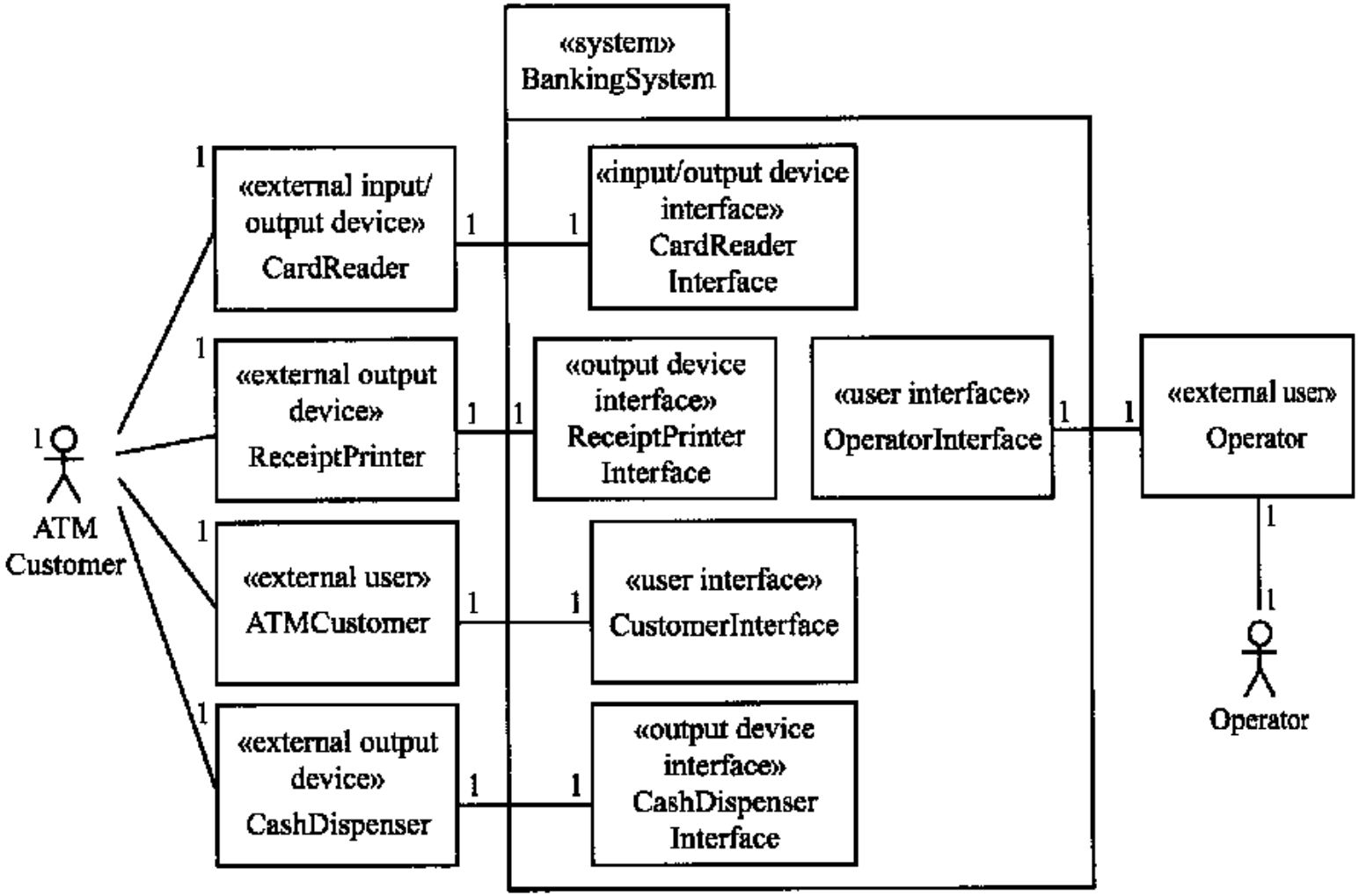
Categorization of External Classes



Identifying Interface Classes

- Each of the external classes interfaces to an interface class in the system.
  - An external user class interfaces to a user interface class
  - An external system class interfaces to a system interface class
  - An external input device class interfaces to an input device interface class
  - An **external output device** class interfaces to an output device interface class
  - An external I/O device class interfaces to an I/O device interface class
  - An external timer class interfaces to an internal timer class

Banking System: External Classes and Interface Classes



Entity Classes

- Store information
- Often mapped to relational database during design

Hue University

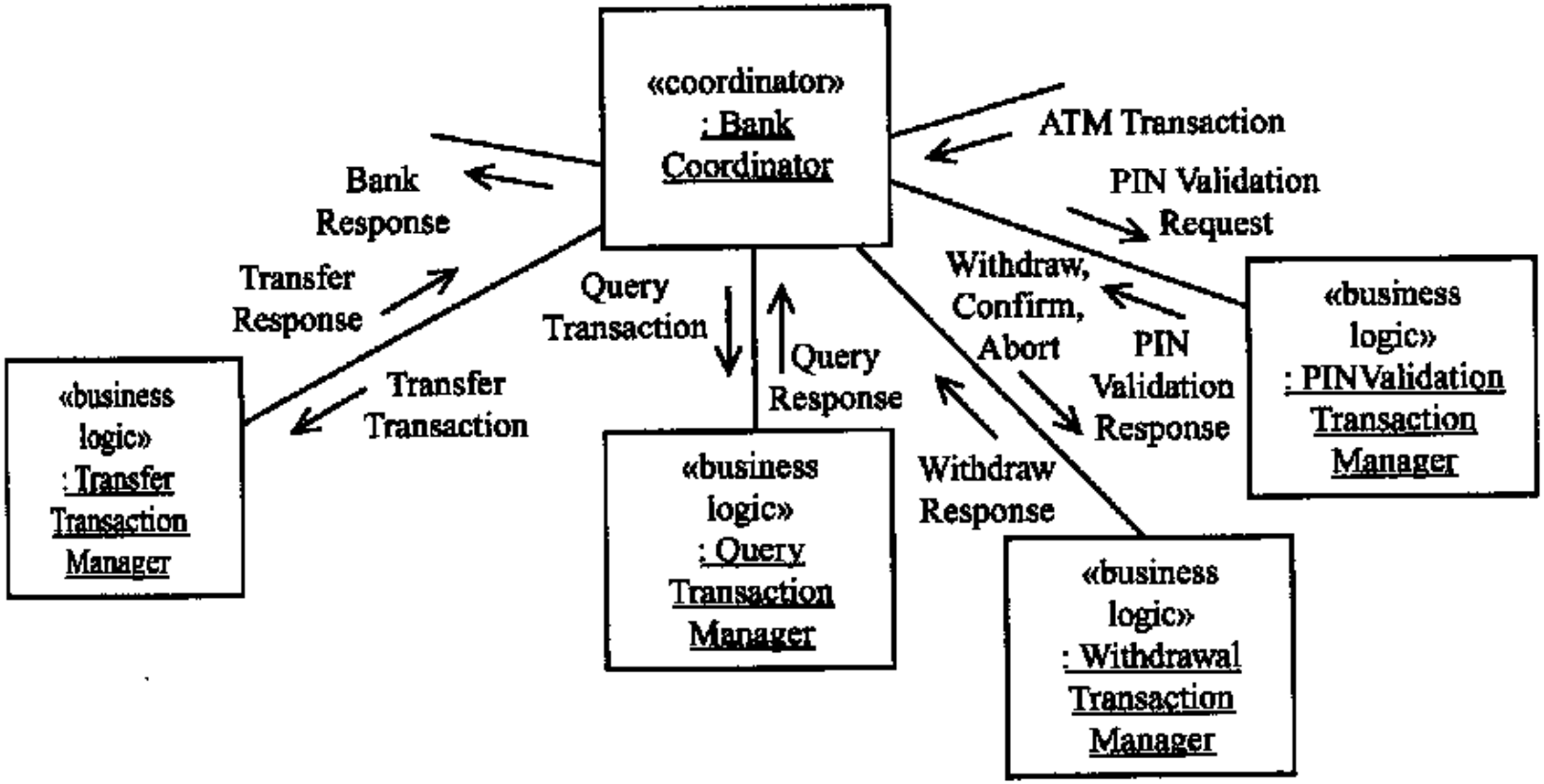
Control Classes

- A control class provides the overall coordination for execution of a use case.
- Makes overall decision
- Control objects decides when, and in what order, other objects participate in use case
  - Interface objects
  - Entity objects
- Simple use cases do not need control objects.

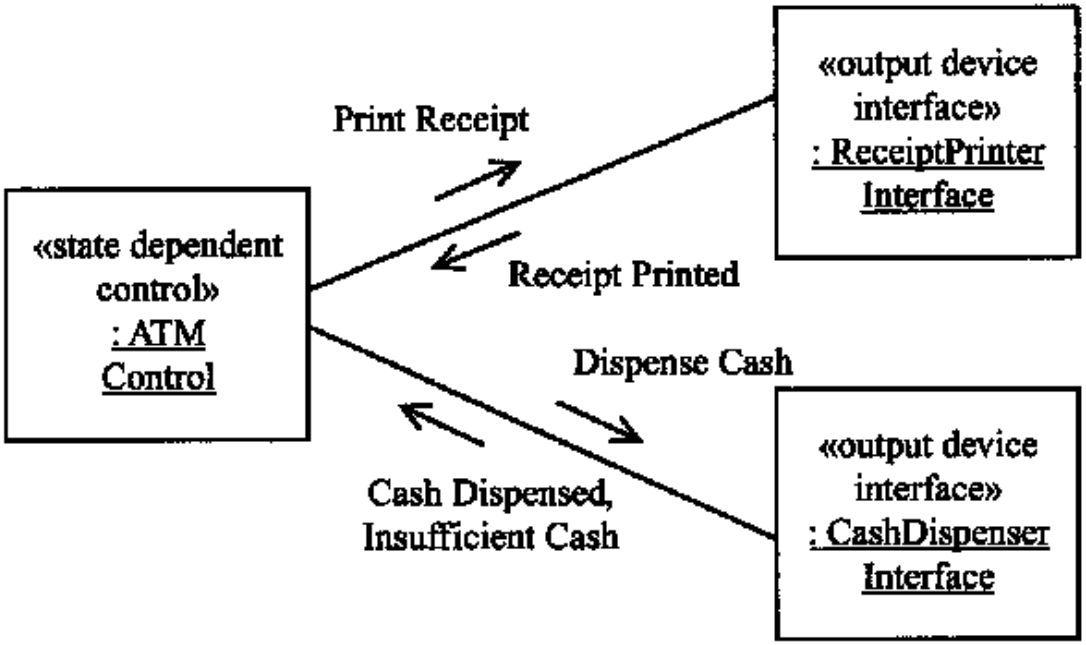
Kinds of Control Classes

- Coordinator class
  - Provides sequencing for use case
  - Is not state dependent
- State dependent control class
  - Defined by finite state machine
- Timer class
  - Activated periodically

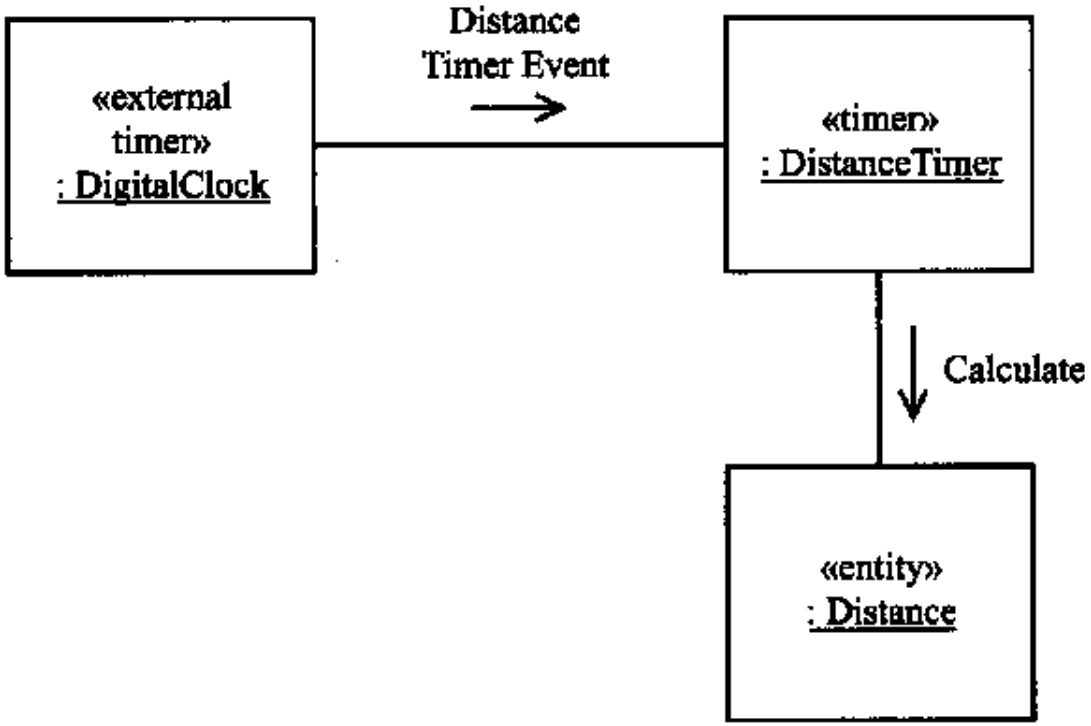
Example: Coordinator Object



Example: State Dependent Control Object



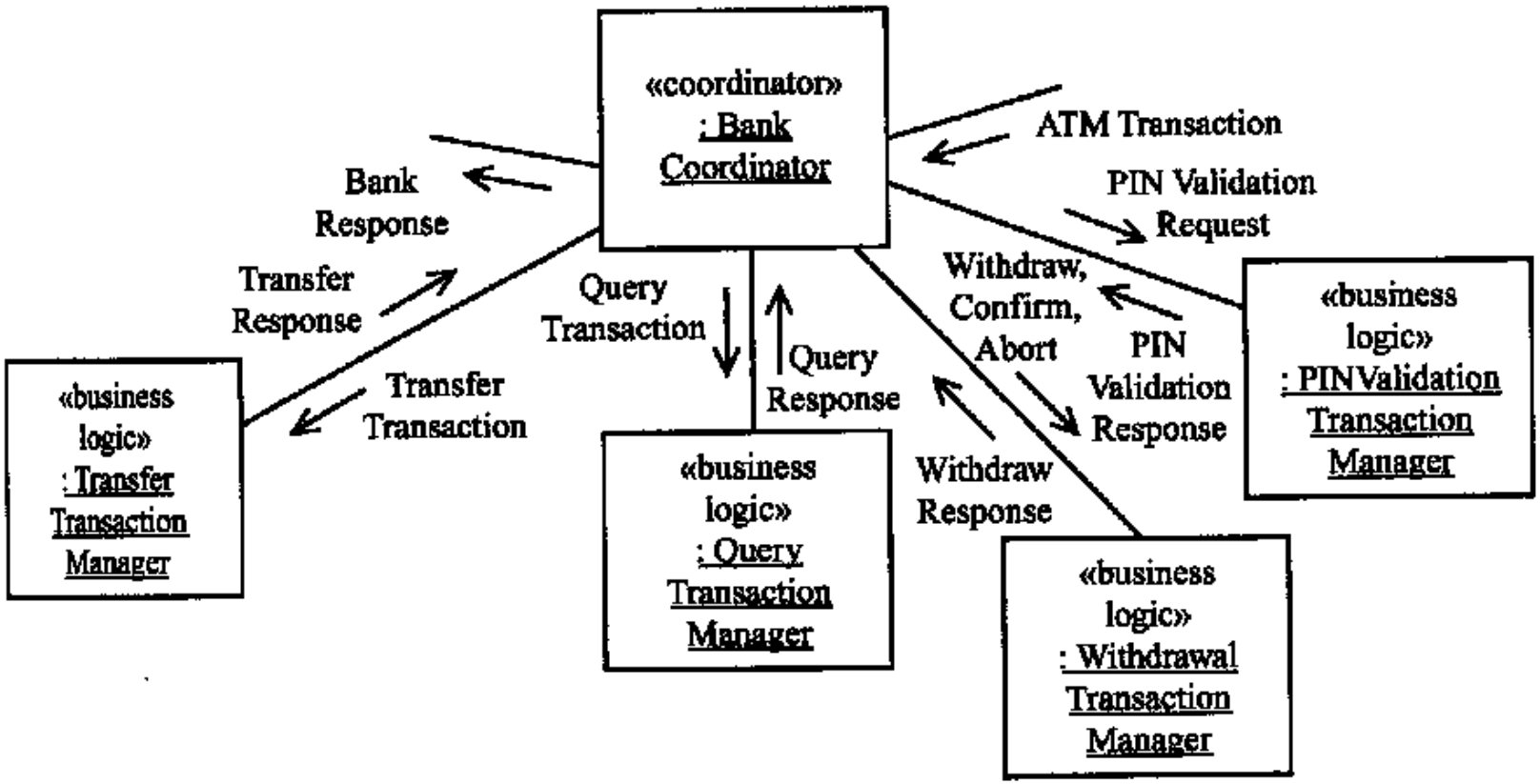
Example: Timer Object



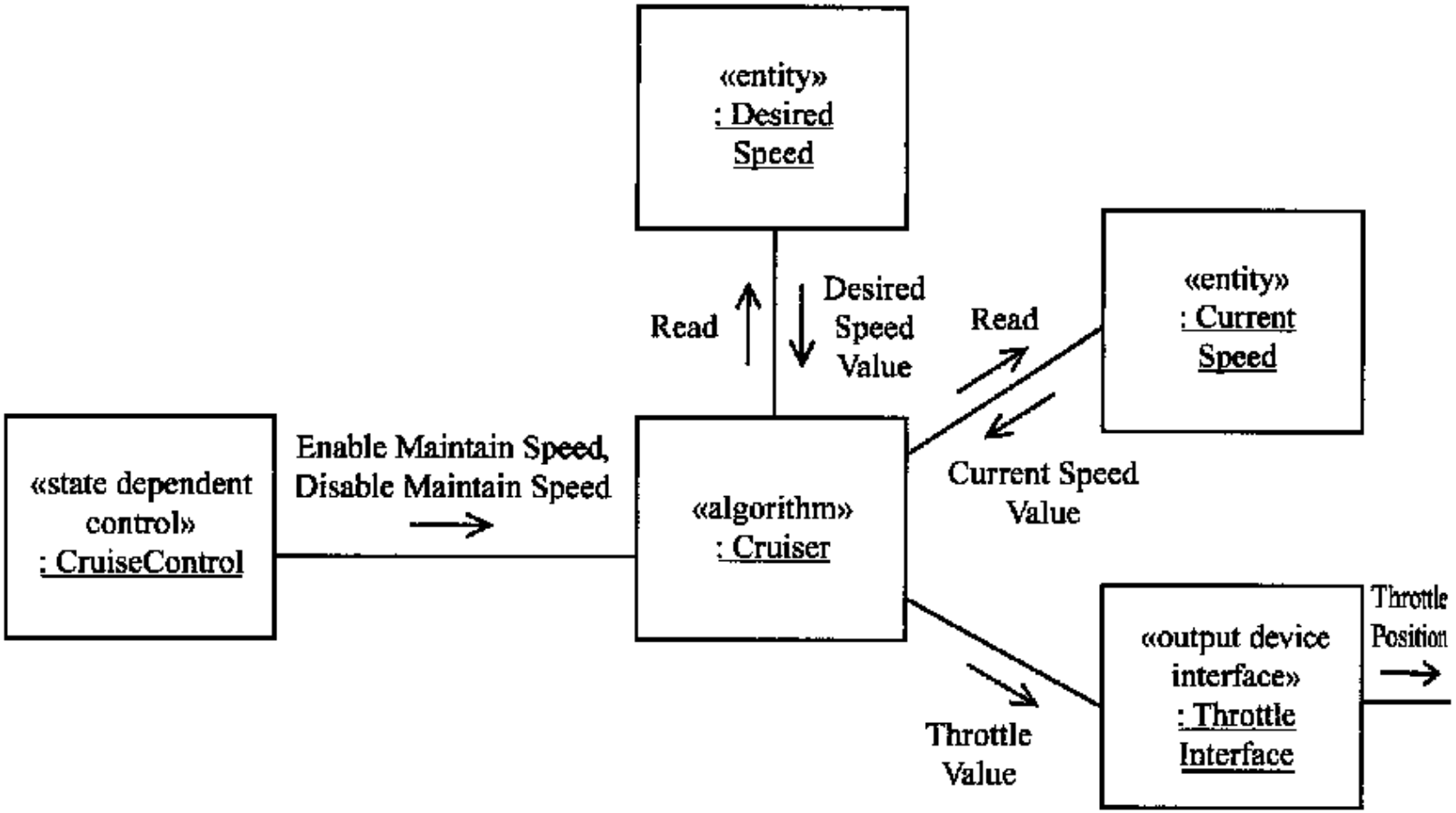
Application Logic Classes

- Business logic class
  - Defines business-specific application logic (rules) for processing a client request
  - Usually accesses more that one entity object
- Algorithm class
  - Encapsulates algorithm used in problem domain
  - More usual in scientific, engineering, real-time domains

Example: Business Logic Object



Example: Algorithm Object



Tips

- Don't try to use all the various notations.
- Don't draw models for everything, concentrate on the key areas.
- Draw implementation models only when illustrating a particular implementation technique.

Hue University

Note: Unifying Concepts

- classifier-instance dichotomy
  - e.g. an object is an instance of a class OR a class is the classifier of an object
- specification-realization dichotomy
  - e.g. an interface is a specification of a class OR a class is a realization of an interface
- analysis-time vs. design-time vs. run-time
  - modeling phases ("process creep")
  - usage guidelines suggested, not enforced

Thank You!

…

Hue University