CSc 131 – Computer Software Engineering

Analysis Modeling

Analysis Modeling

- Requirements analysis
- Flow-oriented modeling
- Scenario-based modeling
- Class-based modeling
- Behavioral modeling

(Source: Pressman, R. *Software Engineering: A Practitioner's Approach*. McGraw-Hill)

Requirements Analysis

Requirements Analysis

Requirements analysis

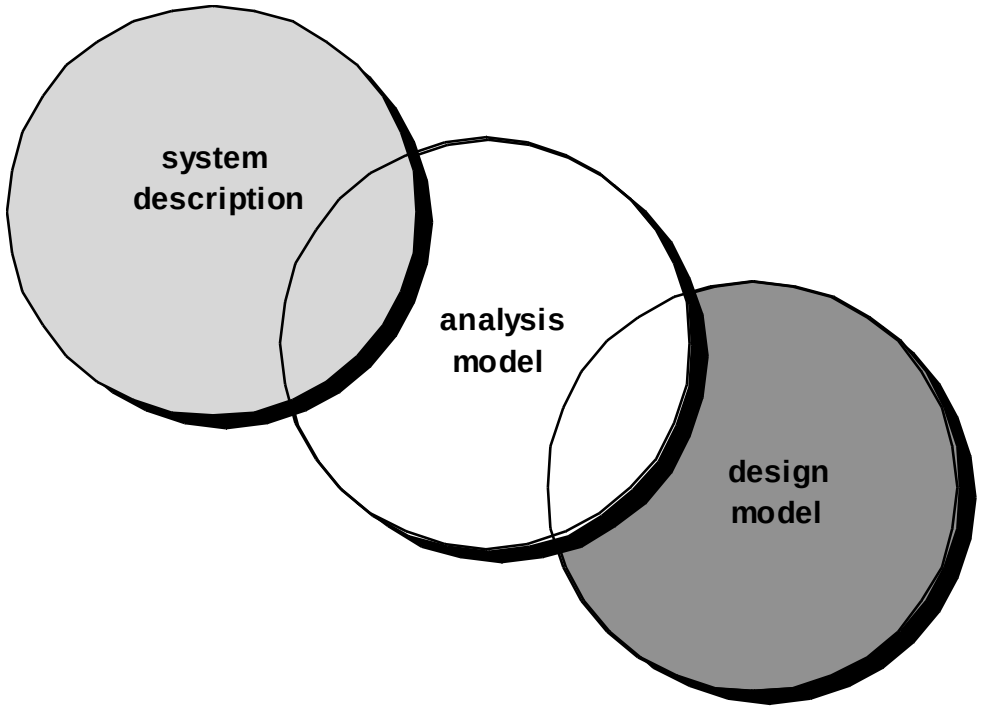
- ❑ specifies software's operational characteristics
- ❑ indicates software's interface with other system elements
- ❑ establishes constraints that software must meet

Requirements analysis allows the software engineer (called an *analyst* or *modeler* in this role) to:

- ❑ elaborate on basic requirements established during earlier requirement engineering tasks
- ❑ build models that depict user scenarios, functional activities, problem classes and their relationships, system and class behavior, and the flow of data as it is transformed.

This is later translated into architectural, interface, class/data and component-level designs

Analysis Phase: What is it?



Three objectives:

- To describe what the customer requires
- To establish a basis for the creation of a software design
- To define a set of requirements that can be validated once the software is built

Analysis Modeling Approaches

Structural analysis:

- The **data**: The model defines their *attributes* and *relationships*.
- The **processes** that *transform* the data: The model shows how they transform the data objects as they *flow* through the system.
- Considers data and the processes that transform the data as separate entities

Object-oriented analysis:

- Focus: **Classes** and their **inter-relationships**
- UML is predominantly object-oriented

A Set of Models

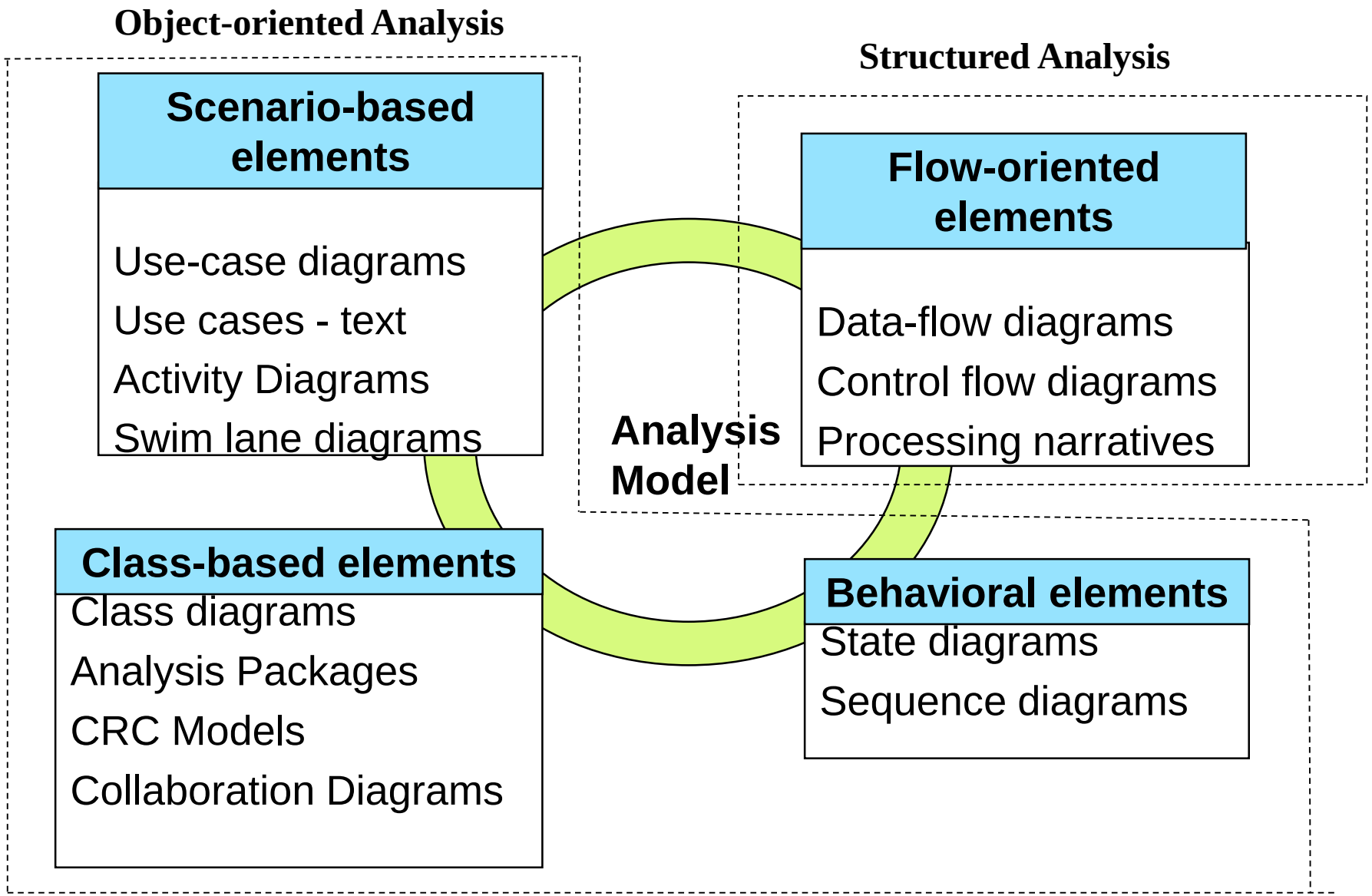
Flow-oriented modeling – provides an indication of how data objects are transformed by a set of processing functions

Scenario-based modeling – represents the system from the user's point of view

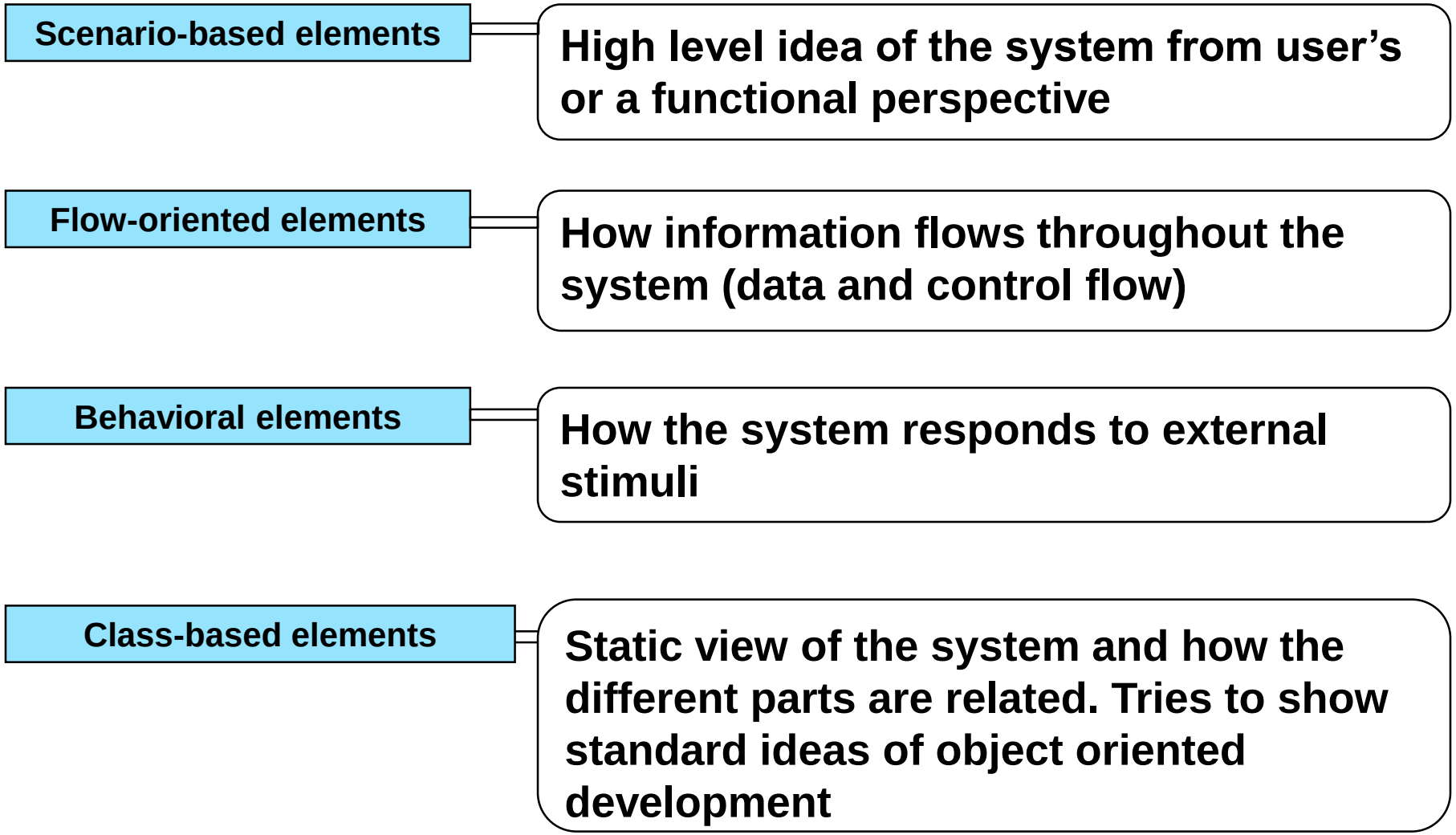
Class-based modeling – defines objects, attributes, and relationships

Behavioral modeling – depicts the states of the classes and the impact of events on these states

Elements of the Analysis Model



Elements of the Analysis Model



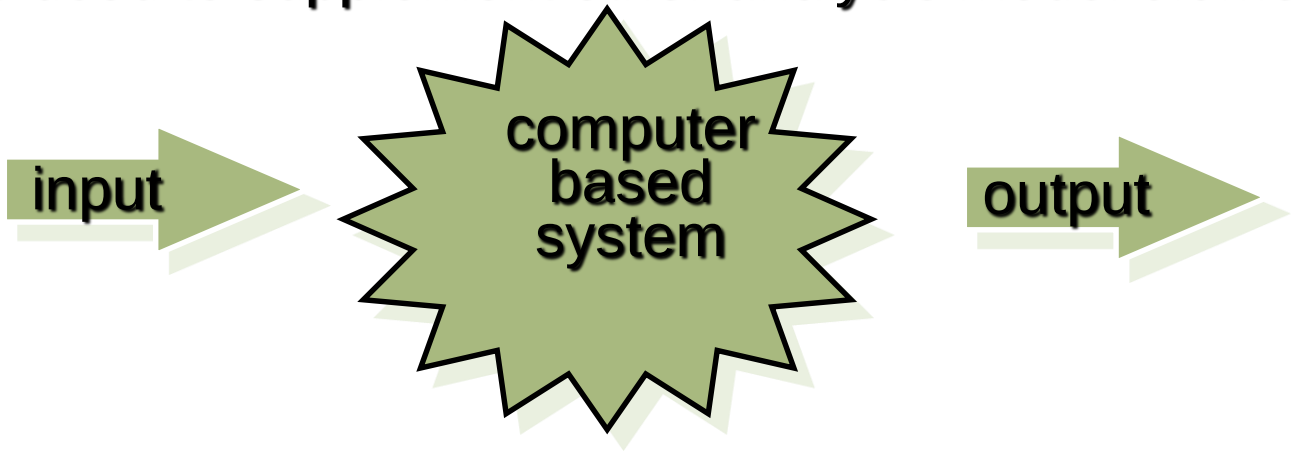
Flow-oriented Modeling

Flow-Oriented Modeling

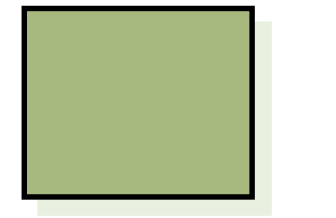
Represents how data objects are transformed at they move through the system

A data flow diagram (DFD) is the diagrammatic form that is used

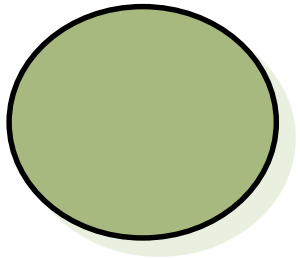
Considered by many to be an 'old school' approach, flow-oriented modeling continues to provide a view of the system that is unique—it should be used to supplement other analysis model elements



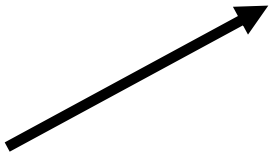
Flow Modeling Notation



external entity



process



data flow

data store

External Entity

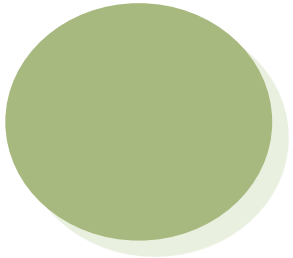
A producer or consumer of data

Examples: a person, a device, a sensor

Another example: computer-based system

Data must always originate somewhere and must always be sent to something

Process

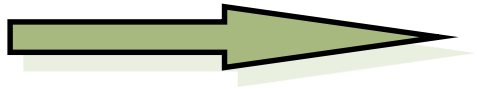


A data transformer (changes input to output)

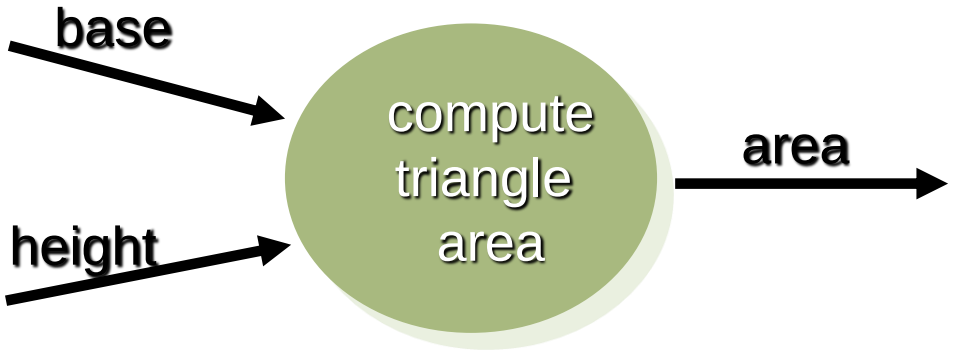
Examples: compute taxes, determine area, format report, display graph

Data must always be processed in some way to achieve system function

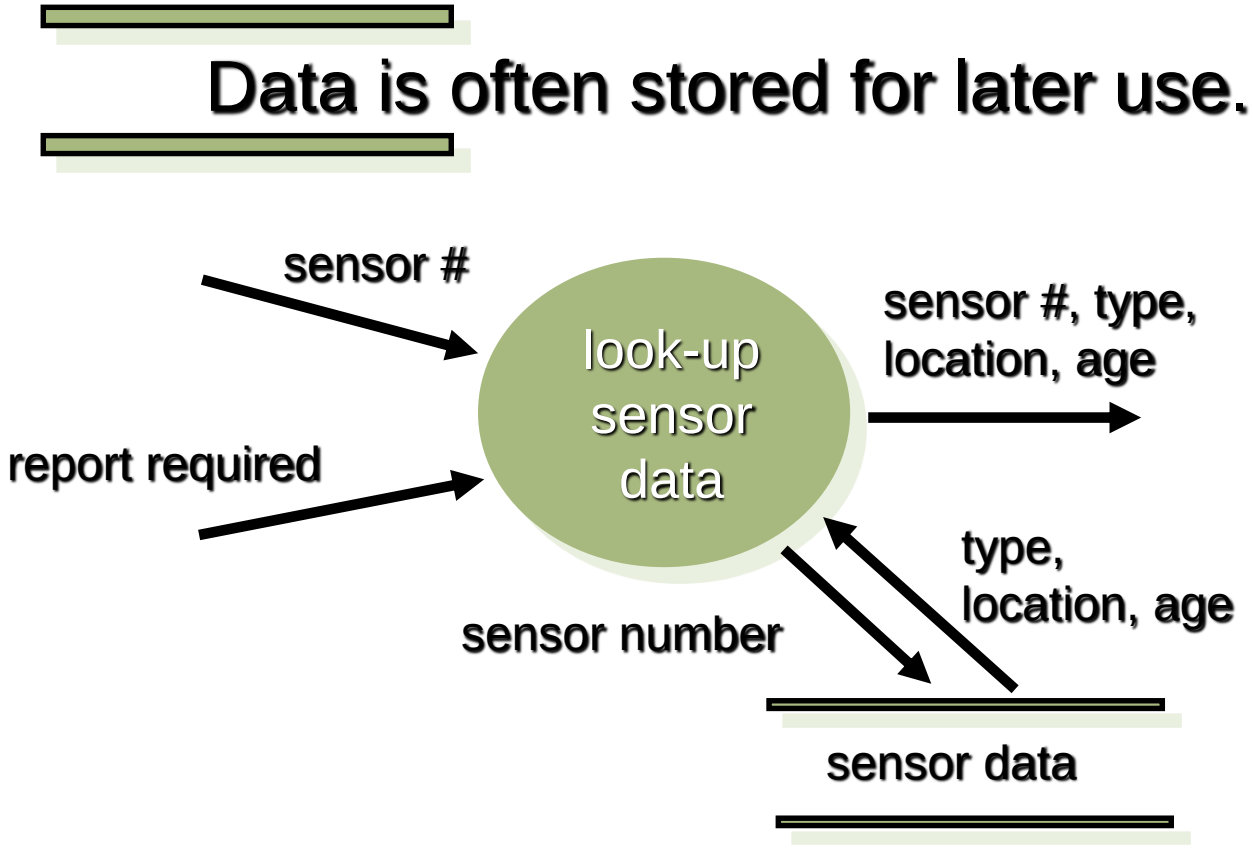
Data Flow



Data flows through a system, beginning as input and be transformed into output.



Data Stores



Data Flow Diagramming: Guidelines

all icons must be labeled with meaningful names

the DFD evolves through a number of levels of detail

always begin with a context level diagram (also called level 0)

always show external entities at level 0

always label data flow arrows

do not represent procedural logic

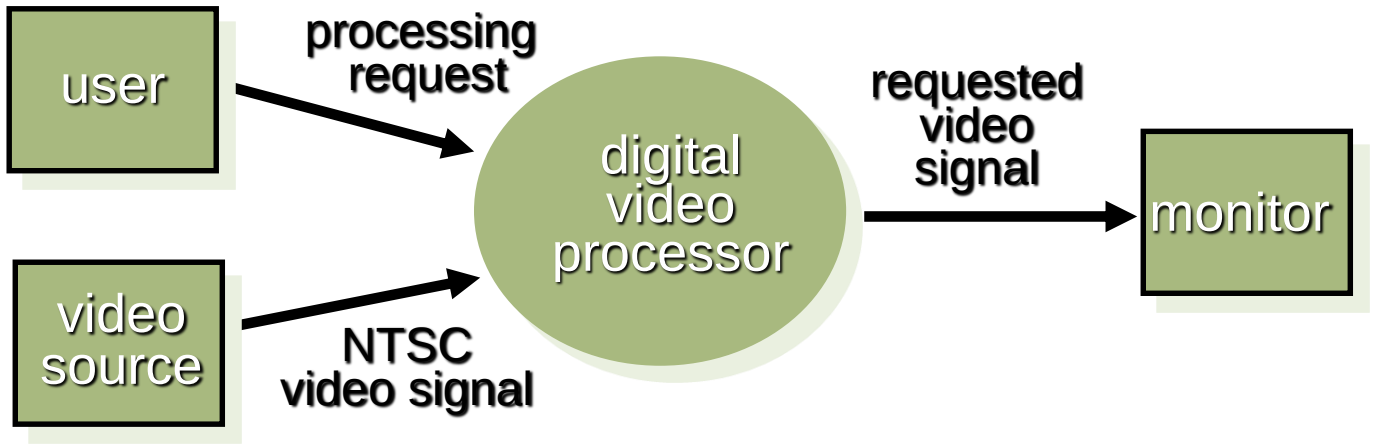
Constructing a DFD—I

review the data model to isolate data objects and use a grammatical parse to determine "operations"

determine external entities (producers and consumers of data)

create a level 0 DFD

Level 0 DFD Example



Constructing a DFD—II

write a narrative describing the transform

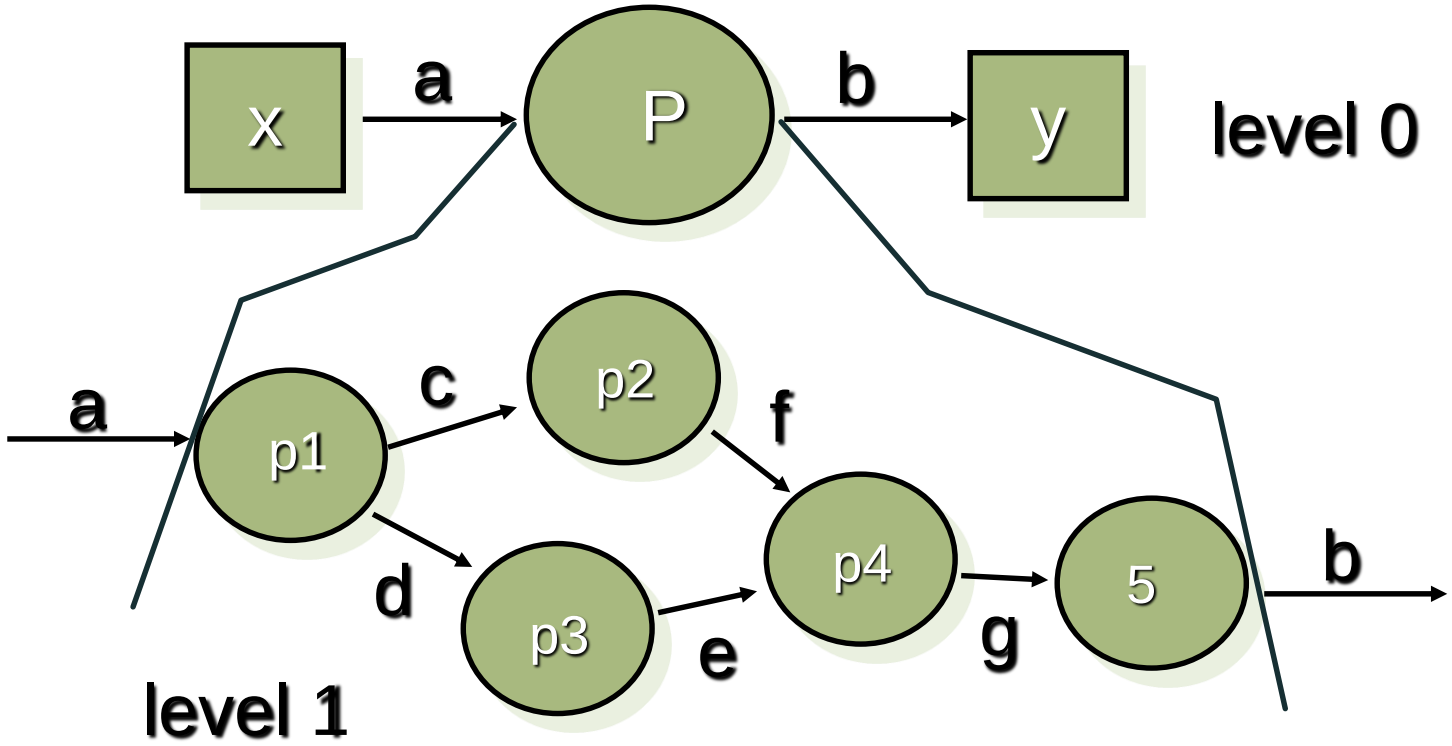
parse to determine next level transforms

“balance” the flow to maintain data flow continuity

develop a level 1 DFD

use a 1:5 (approx.) expansion ratio

The Data Flow Hierarchy



Any correspondence with a use case diagram?

Flow Modeling Notes

each bubble is refined until it does just one thing

the expansion ratio decreases as the number of levels increase

most systems require between 3 and 7 levels for an adequate flow model

a single data flow item (arrow) may be expanded as levels increase (data dictionary provides information)

Scenario-based Modeling

Scenario-based Modeling

Use-Cases

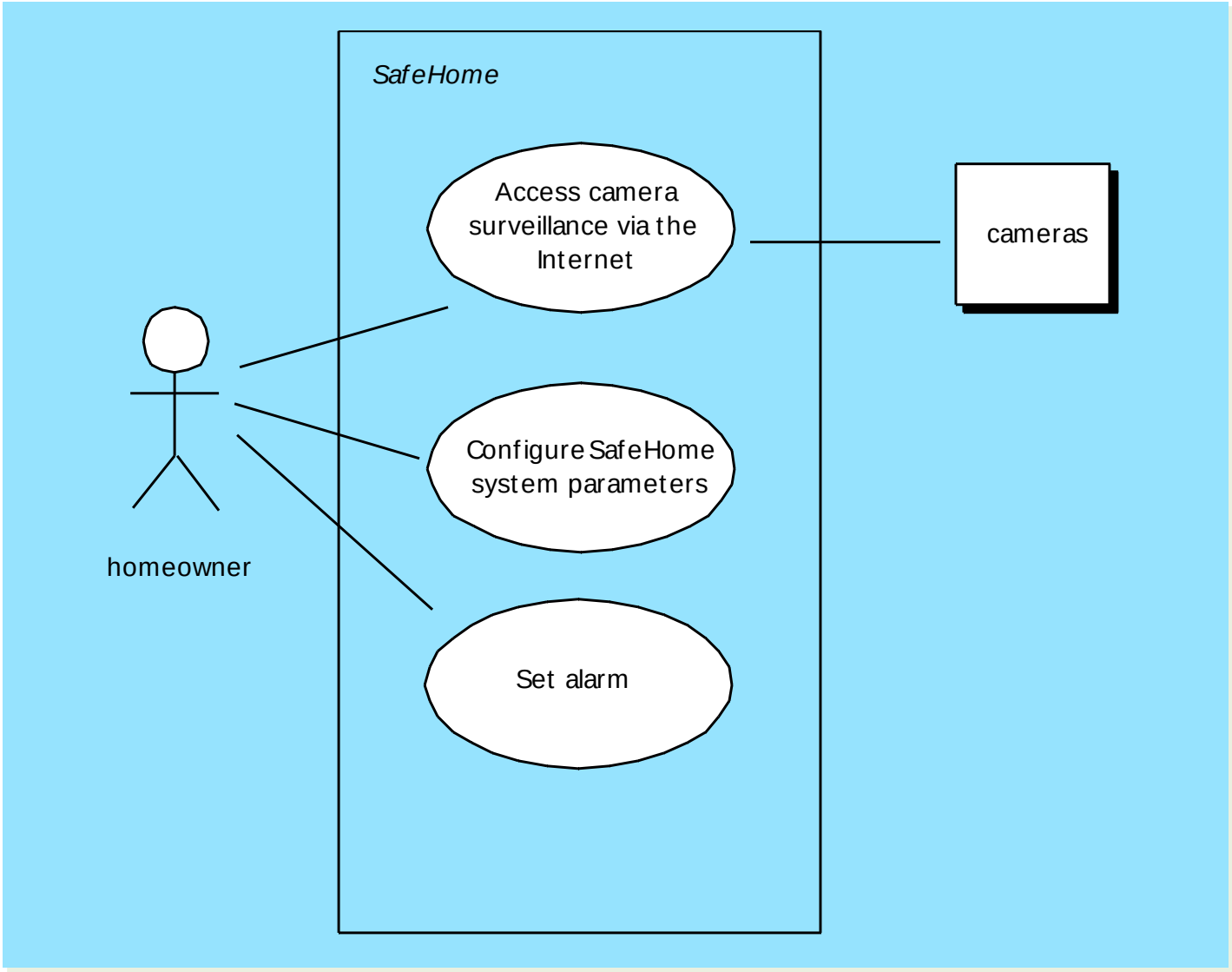
> "[Use-cases] are simply an aid to defining what exists outside the system (actors) and what should be performed by the system (use-cases)."
>
> --- **Ivar Jacobson**

a scenario that describes a "thread of usage" for a system

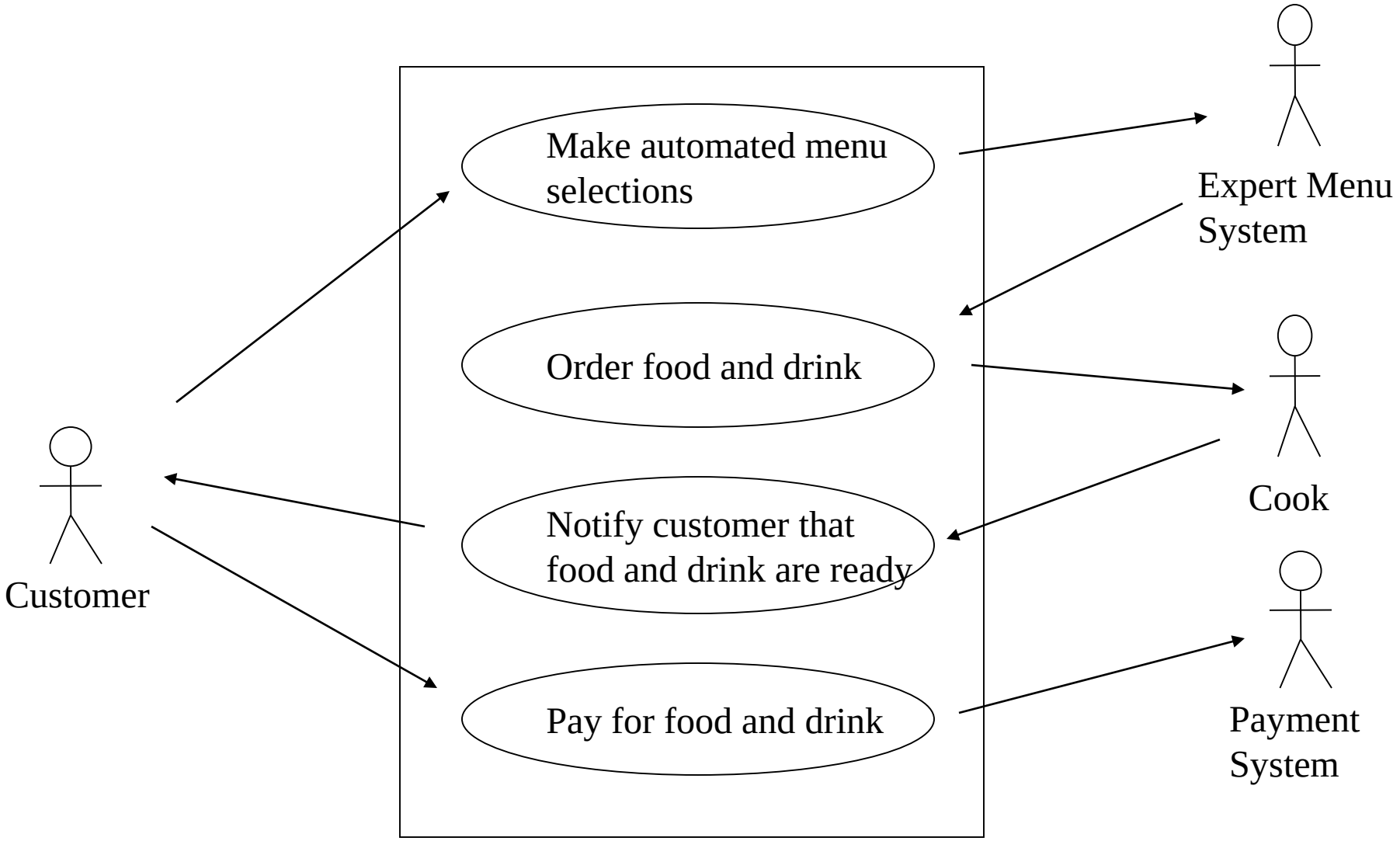
actors represent roles people or devices play as the system functions

users can play a number of different roles for a given scenario

Use-Case Diagram



Example Use Case Diagram



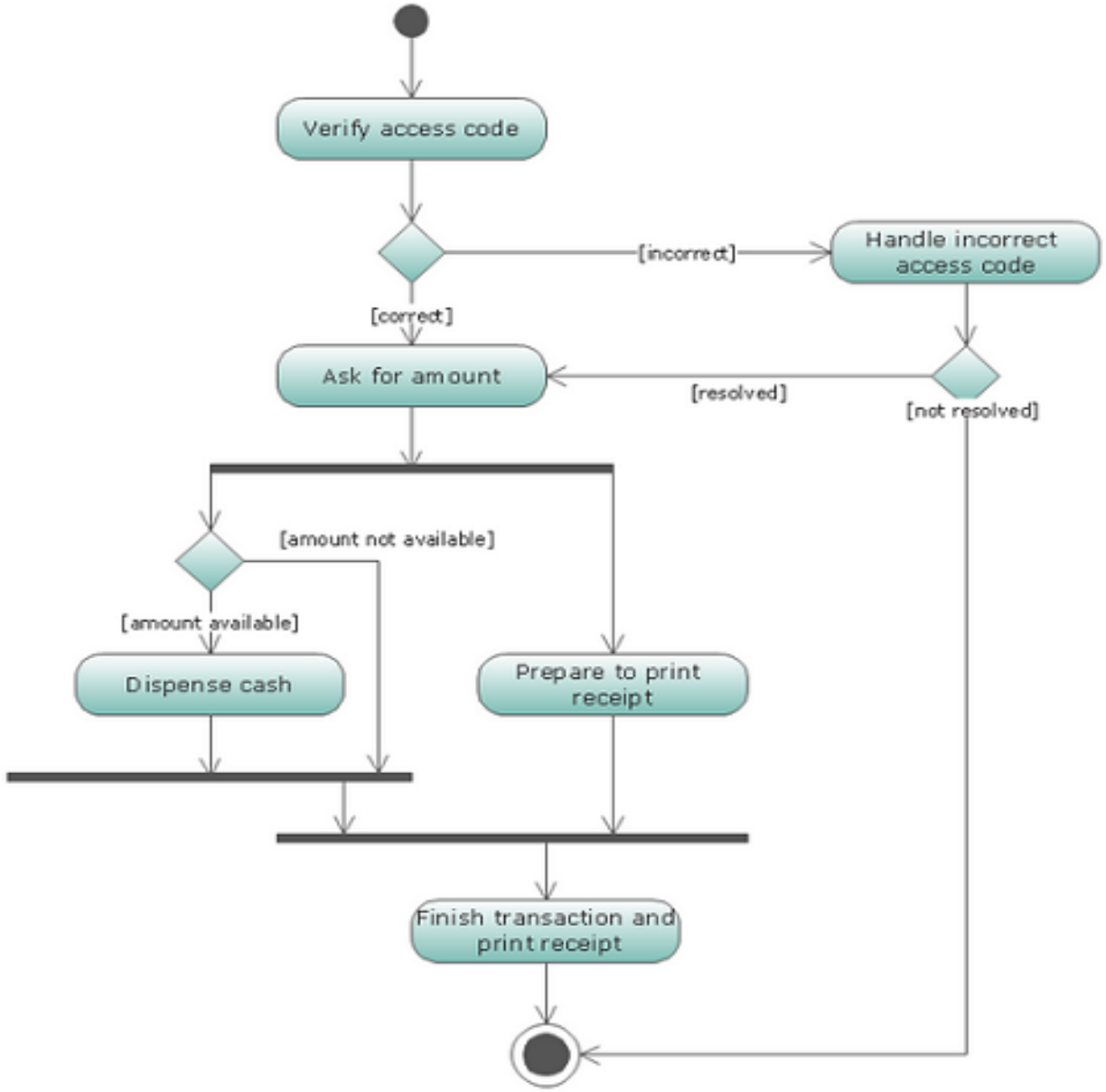
Activity Diagrams

Supplements the use case by providing a graphical representation of the flow of interaction within a specific scenario

Uses flowchart-like symbols

- **Rounded rectangle** - represent a specific system function/action
- **Arrow** - represents the flow of control from one function/action to another
- **Diamond** - represents a branching decision
- **Solid bar** – represents the fork and join of parallel activities

Example Activity Diagram for ATM Withdraw scenario



Class-based Modeling

Class-Based Modeling

Identify analysis classes by examining the problem statement

Use a “grammatical parse” to isolate potential classes

Identify the attributes of each class

Identify operations that manipulate the attributes

Identifying Analysis Classes

1) Perform a grammatical parse of the problem statement or use cases
2) Classes are determined by underlining each noun or noun clause
3) A class required to implement a solution is part of the solution space
4) A class necessary only to describe a solution is part of the problem space
5) A class should NOT have an imperative procedural name (i.e., a verb)
6) List the potential class names in a table and "classify" each class according to some taxonomy and class selection characteristics
7) A potential class should satisfy nearly all (or all) of the selection characteristics to be considered a legitimate problem domain class

Potential classes	General classification	Selection Characteristics

Grammatical Parsing

Write an informal description of the problem. The customer requirements document is one such description.

Underline all nouns in the description

Decide which of these are really objects which the project requires and organize them in related clusters

Grammatical Parsing

University **Bank** will be opening in **Oxford, Mississippi**, in **January**, 2000. We plan to use a full service automated teller machine (**ATM**) system.The ATM system will interact with the **customer** through a **display screen, numeric and special input keys**, a **bankcard reader,** a **deposit slot**, and a **receipt printer**.Customers may make **deposits**, withdrawals, and balance inquires using the ATM machine, but the update to **accounts** will be handled through an **interface** to the **Accounts system**.Customers will be assigned a Personal Identification Number (**PIN**) and clearance level by the **Security system**. The PIN can be verified prior to any transaction.In the future, we would also like to support routine operations such as a **change of address** or **phone number** using the ATM

Grammatical Parsing

University Bank will be opening in Oxford, Mississippi, in January, 2000. We plan to use a full service automated teller machine (ATM) system.The *ATM system* will interact with the *customer* through a *display screen*, *numeric* and *special input keys*, a *bankcard reader*, a *deposit* slot, and a *receipt printer*.Customers may make *deposits*, *withdrawals*, and *balance inquires* using the ATM machine, but the update to *accounts* will be handled through an interface to the *Accounts system*.Customers will be assigned a *Personal Identification Number (PIN)* and *clearance level* by the *Security system*. The PIN can be verified prior to any transaction.In the future, we would also like to support routine operations such as a *change of address* or *phone number* using the ATM

Typical Classes (a reminder)

External entities - printer, user, sensor

Things - reports, displays, signals

Occurrences or events (e.g., interrupt, alarm)

Roles (e.g., manager, engineer, salesperson)

Organizational units (e.g., division, team)

Places (e.g., manufacturing floor or loading dock)

Structures (e.g., sensors, four-wheeled vehicles, or computers)

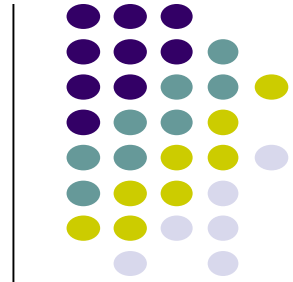
But, how do we select classes?

Selecting Classes—Criteria

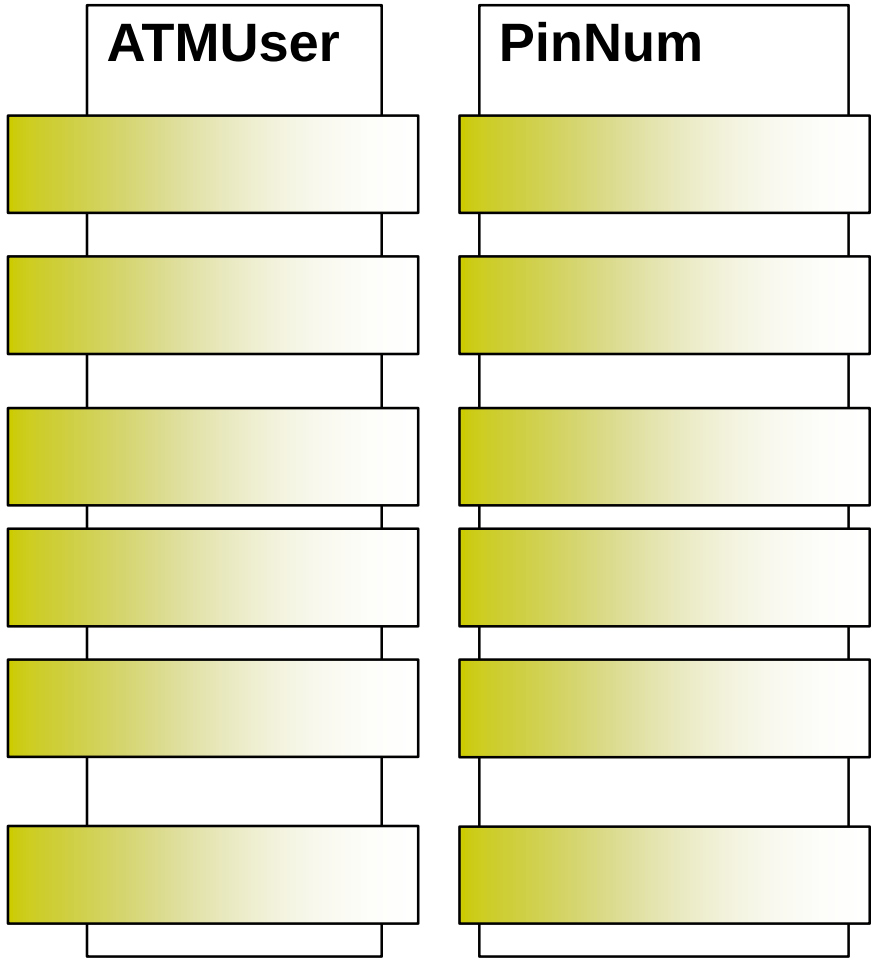
- **retained information** – information about it must be remembered
- **needed services** – operations that change the attributes
- **multiple attributes** – if it is only one attribute, probably should be part of another class
- **common attributes** – common things for all instances of a class
- **common operations** – for all instances of the class
- **essential requirements** – appear in the PROBLEM space (remember we're doing analysis modeling!)

COMING UP: SELECTING CLASSES—EXAMPLE

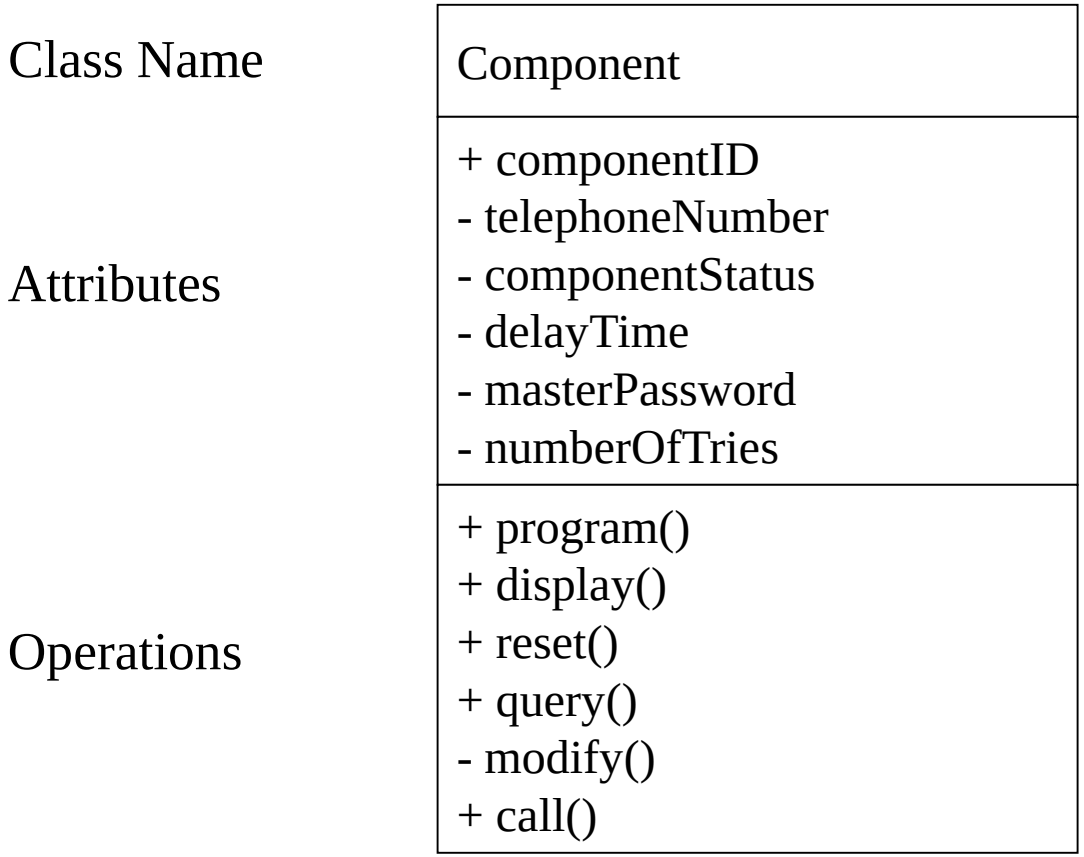
Selecting Classes—Example



- retained information
- needed services
- multiple attributes
- common attributes
- common operations
- essential requirements



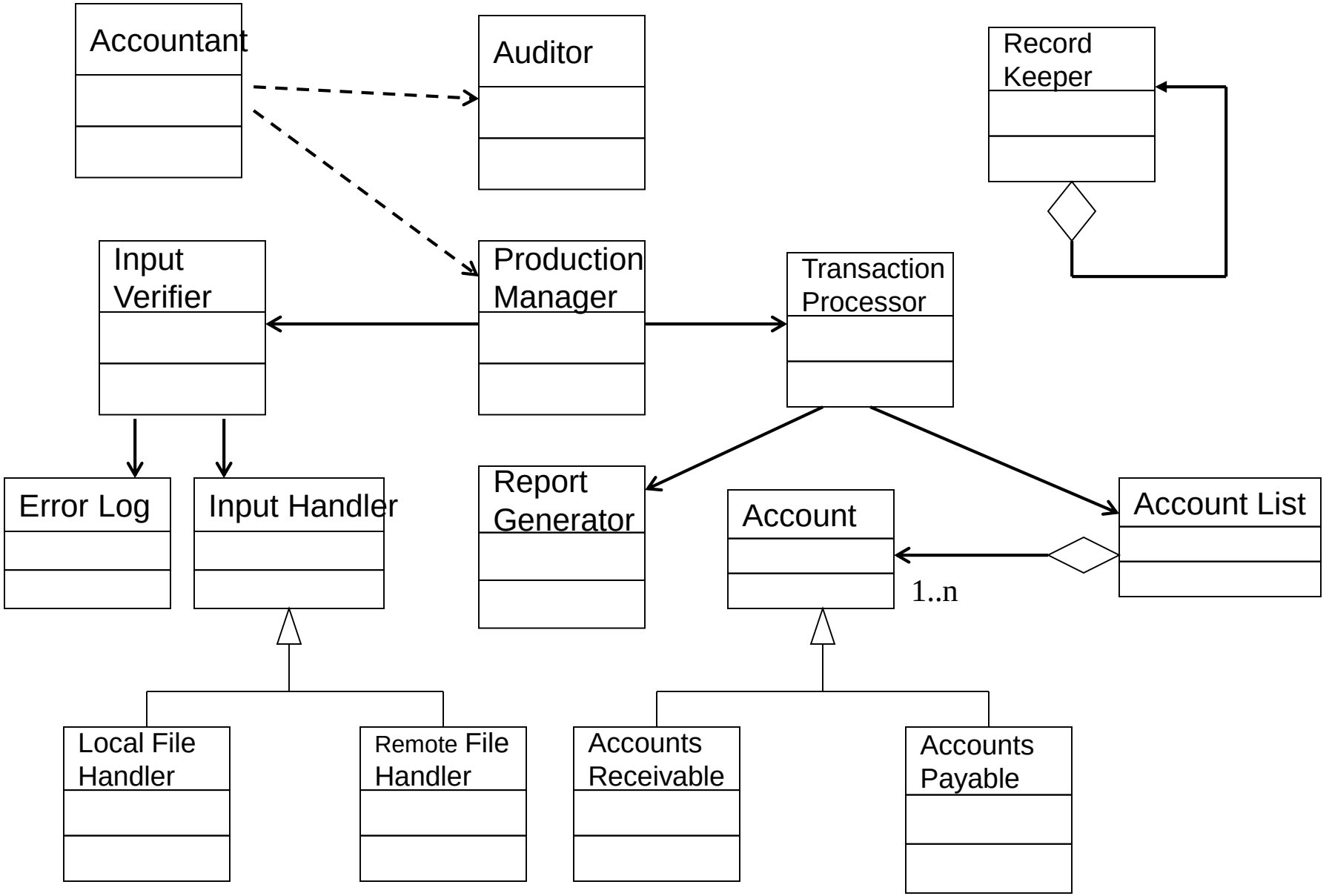
Example Class Box



Association, Generalization and Dependency (Ref: Fowler)

- Association
  - Represented by a solid line between two classes directed from the source class to the target class
  - Used for representing (i.e., pointing to) object types for attributes
  - May also be a part-of relationship (i.e., aggregation), which is represented by a diamond-arrow
- Generalization
  - Portrays inheritance between a super class and a subclass
  - Is represented by a line with a triangle at the target end
- Dependency
  - A dependency exists between two elements if changes to the definition of one element (i.e., the source or supplier) may cause changes to the other element (i.e., the client)
  - Examples
    - *One class calls a method of another class*
    - *One class utilizes another class as a parameter of a method*

Example Class Diagram



Behavioral Modeling

Creating a Behavioral Model

1) Identify events found within the use cases
2) Build a state diagram for each class, and if useful, for the whole software system

Identifying Events in Use Cases

An event occurs whenever an actor and the system exchange information

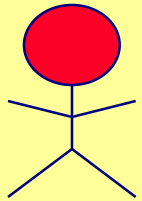
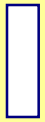
An event is NOT the information that is exchanged, but rather the fact that information has been exchanged

Sequence Diagram

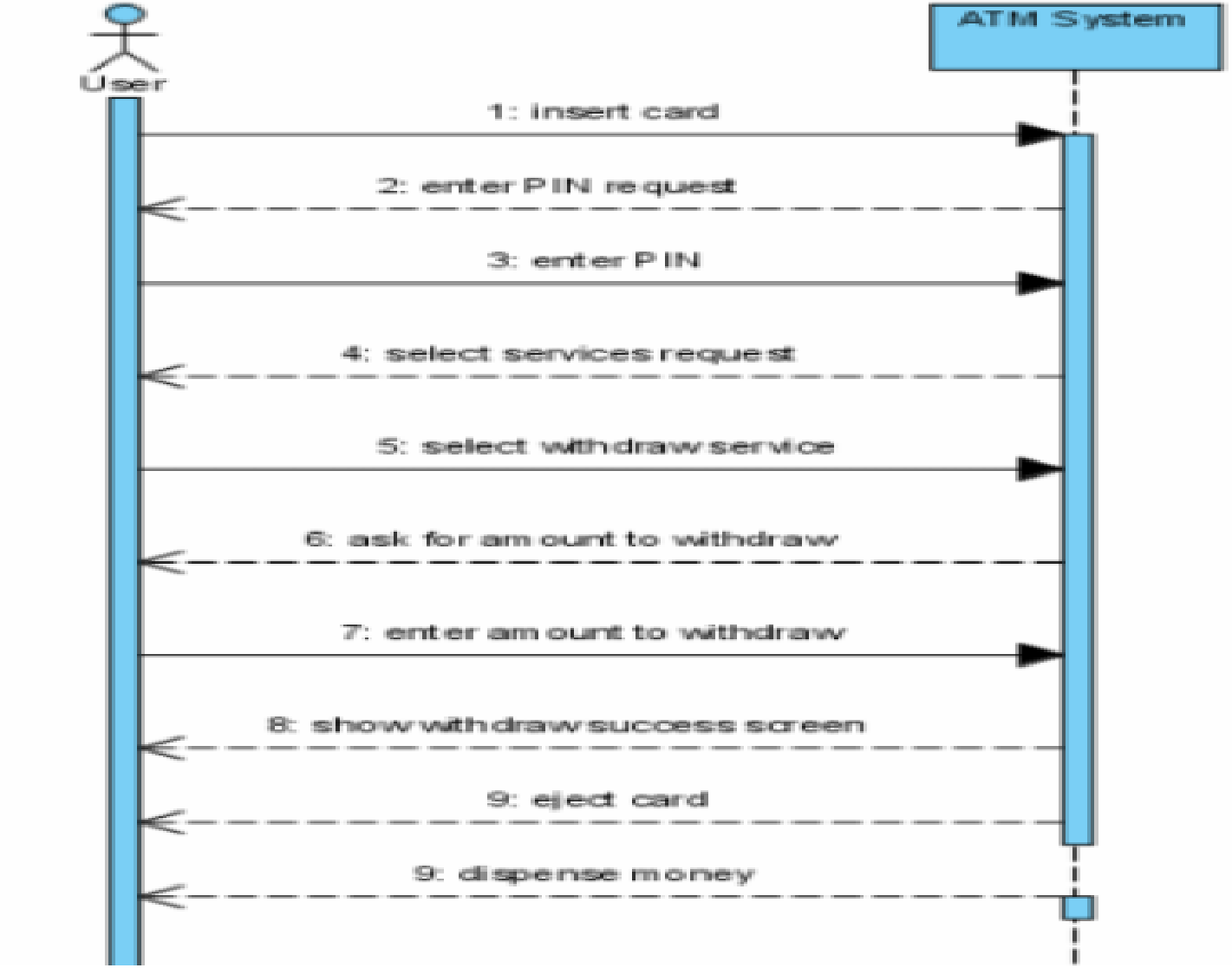
Illustrates the objects that participate in a use case and the messages that pass between them over time for *one* use case

In design, used to distribute use case behavior to classes

Sequence Diagram Syntax

AN ACTOR	
AN OBJECT	
A LIFELINE	
A FOCUS OF CONTROL	
A MESSAGE	
OBJECT DESTRUCTION	x

Example Sequence Diagram for Withdraw Use Case



Building a State Diagram

A state is represented by a rounded rectangle

A transition (i.e., event) is represented by a labeled arrow leading from one state to another

```
Syntax: trigger-signature [guard]/activity
```

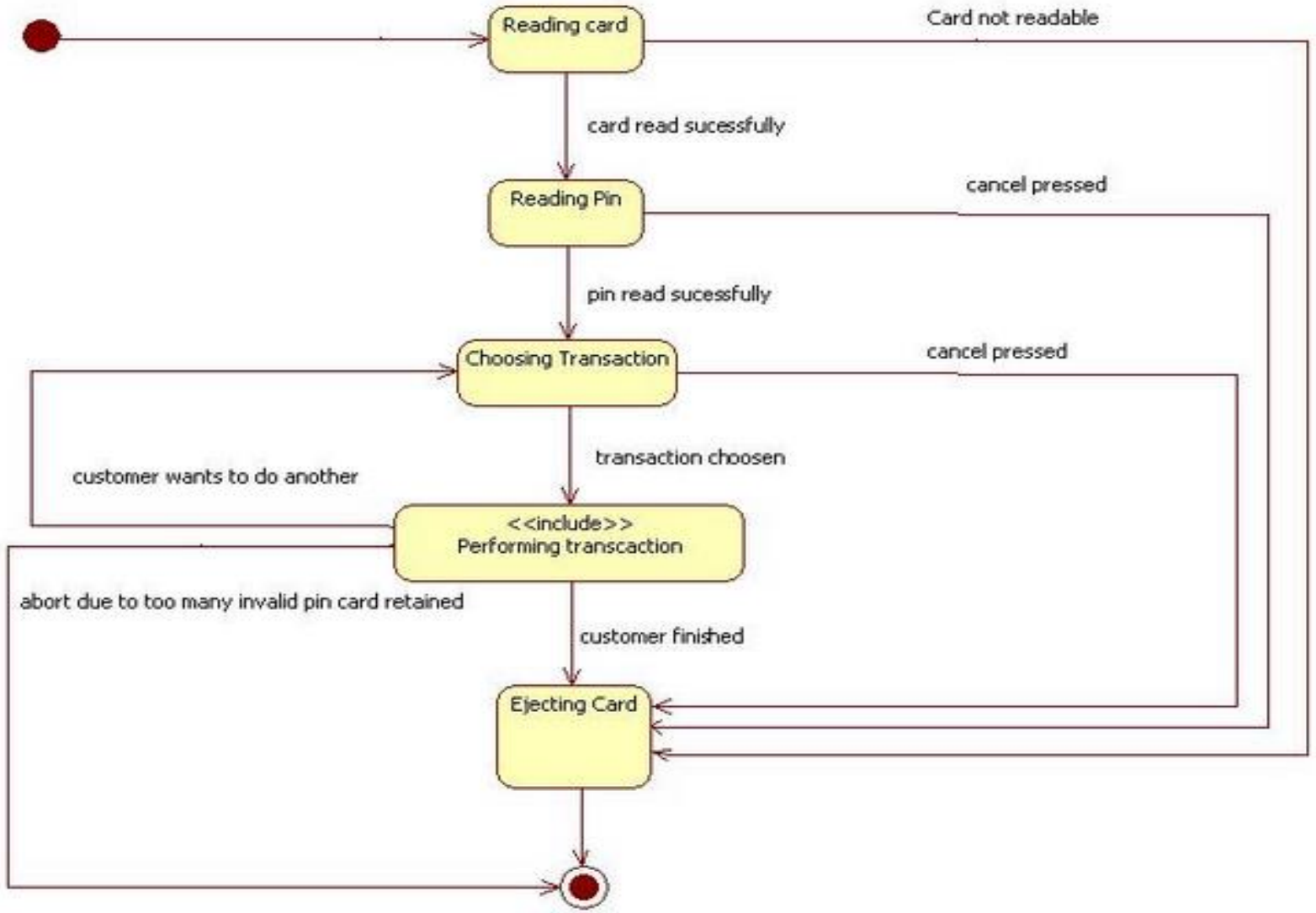
The active state of an object indicates the current overall status of the object as is goes through transformation or processing

- A state name represents one of the possible active states of an object

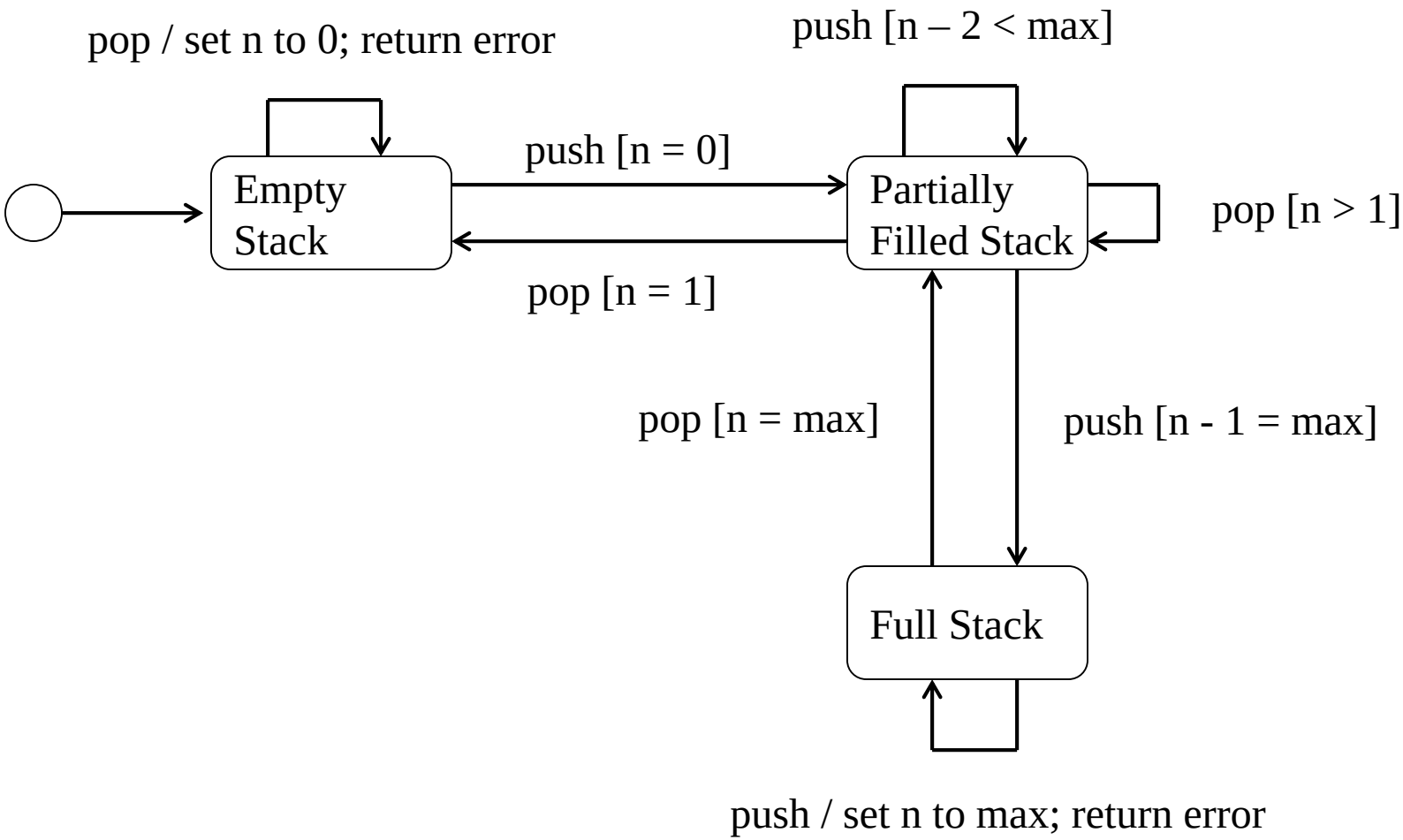
The passive state of an object is the current value of all of an object's attributes

- A guard in a transition may contain the checking of the passive state of an object

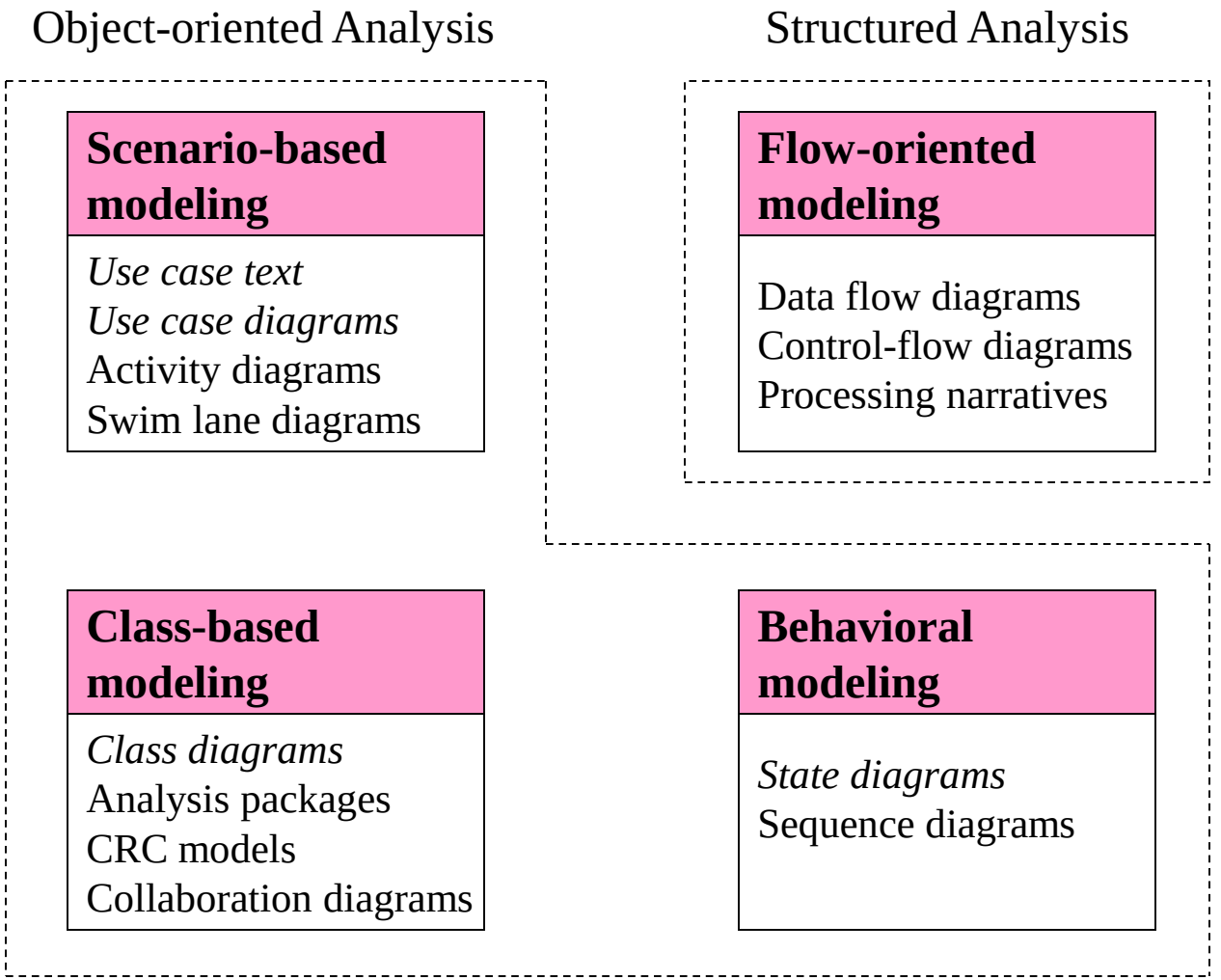
Example State Diagram



Example State Diagram



Summary: Elements of the Analysis Model



☺

Unified Model Language

UML is the industry standard for documenting OO models

Class Diagrams	visualize *logical structure* of system in terms of *classes, objects and relationships*
Use Case Diagrams	show external *actors and use cases* they participate in
Sequence Diagrams	visualize *temporal message ordering* of a *concrete scenario* of a use case
Collaboration (Communication) Diagrams	visualize *relationships* of objects exchanging messages in a *concrete scenario*
State Diagrams	specify the *abstract states* of an object and the *transitions* between the states