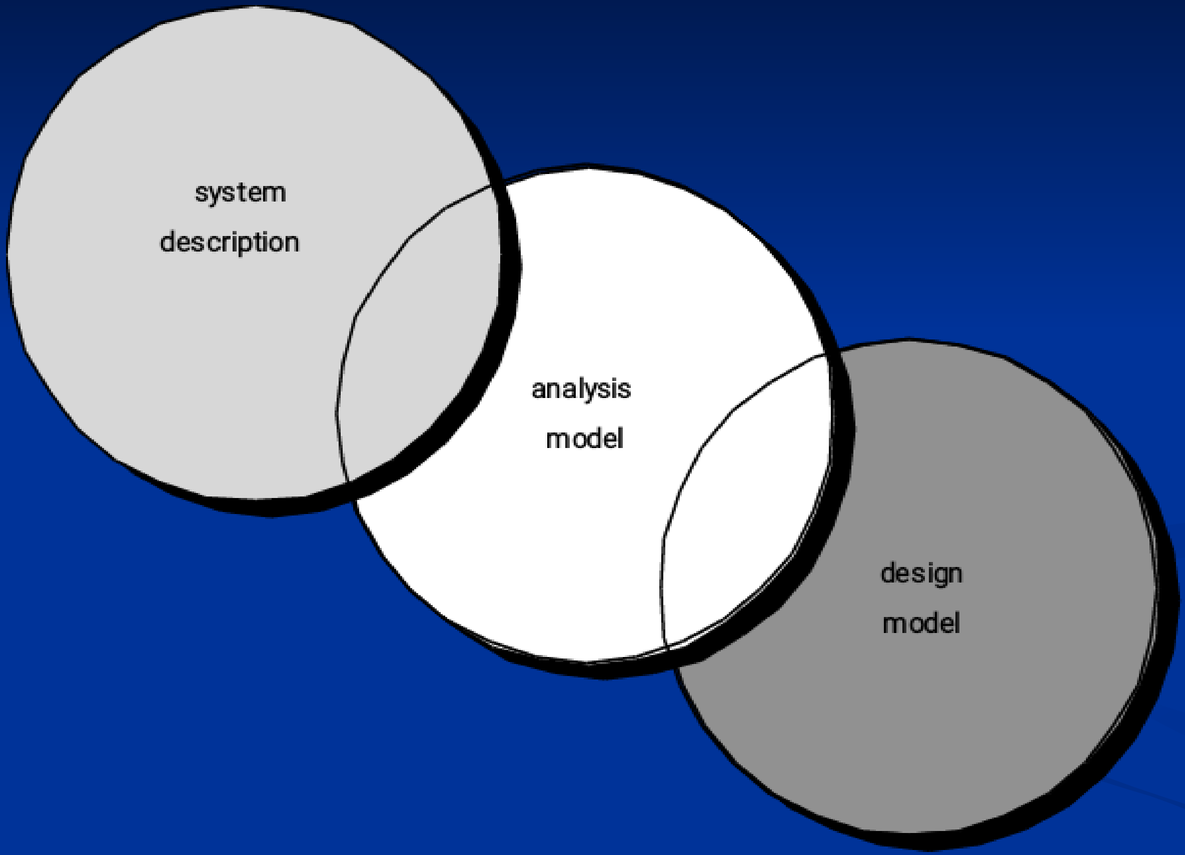
Analysis Modeling

.

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.

A Bridge



Rules of Thumb

- The model should focus on requirements that are visible within the problem or business domain. The level of abstraction should be relatively high.
- Each element of the analysis model should add to an overall understanding of software requirements and provide insight into the information domain, function and behavior of the system.
- Delay consideration of infrastructure and other non-functional models until design.
- Minimize coupling throughout the system.
- Be certain that the analysis model provides value to all stakeholders.
- Keep the model as simple as it can be.

Domain Analysis

- Define the domain to be investigated.
- Collect a representative sample of applications in the domain.
- Analyze each application in the sample.
- Develop an analysis model for the objects.

Data Modeling

- examines data objects independently of processing
- focuses attention on the data domain
- creates a model at the customer’s level of abstraction
- indicates how data objects relate to one another

What is a Data Object?

Object —something that is described by a set of attributes (data items) and that will be manipulated within the software (system)

- each instance of an object (e.g., a book) can be identified uniquely (e.g., ISBN #)
- each plays a necessary role in the system i.e., the system could not function without access to instances of the object
- each is described by attributes that are themselves data items

Typical Objects

- *external entities* **(printer, user, sensor)**
- *things* (e.g, 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)
- *structures* **(e.g., employee record)**

Data Objects and Attributes

A data object contains a set of attributes that act as an aspect, quality, characteristic, or descriptor of the object

object: automobile

---

attributes:
- **make**
- **model**
- **body type**
- **price**
- **options code**

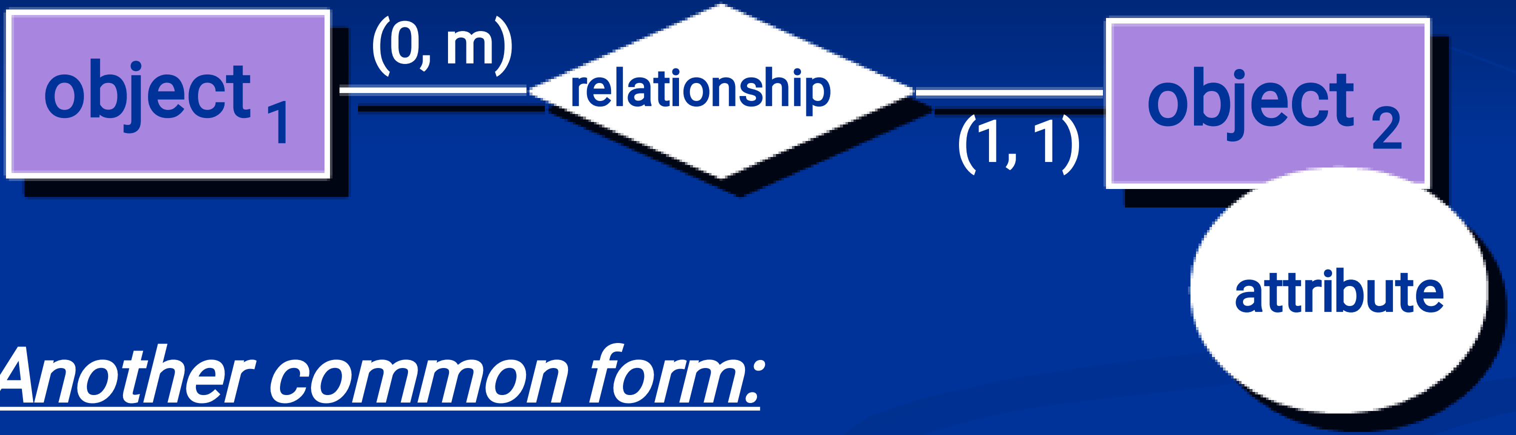
What is a Relationship?

relationship —indicates “connectedness”;
a "fact" that must be "remembered"
by the system and cannot or is not computed
or derived mechanically

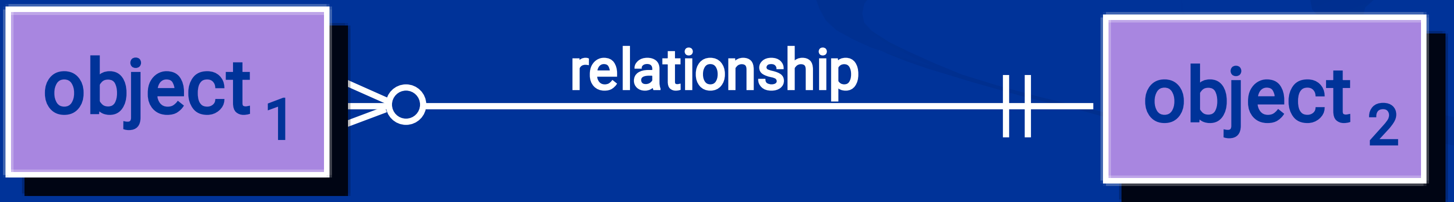
- several instances of a relationship can exist
- objects can be related in many different ways

ERD Notation

One common form:



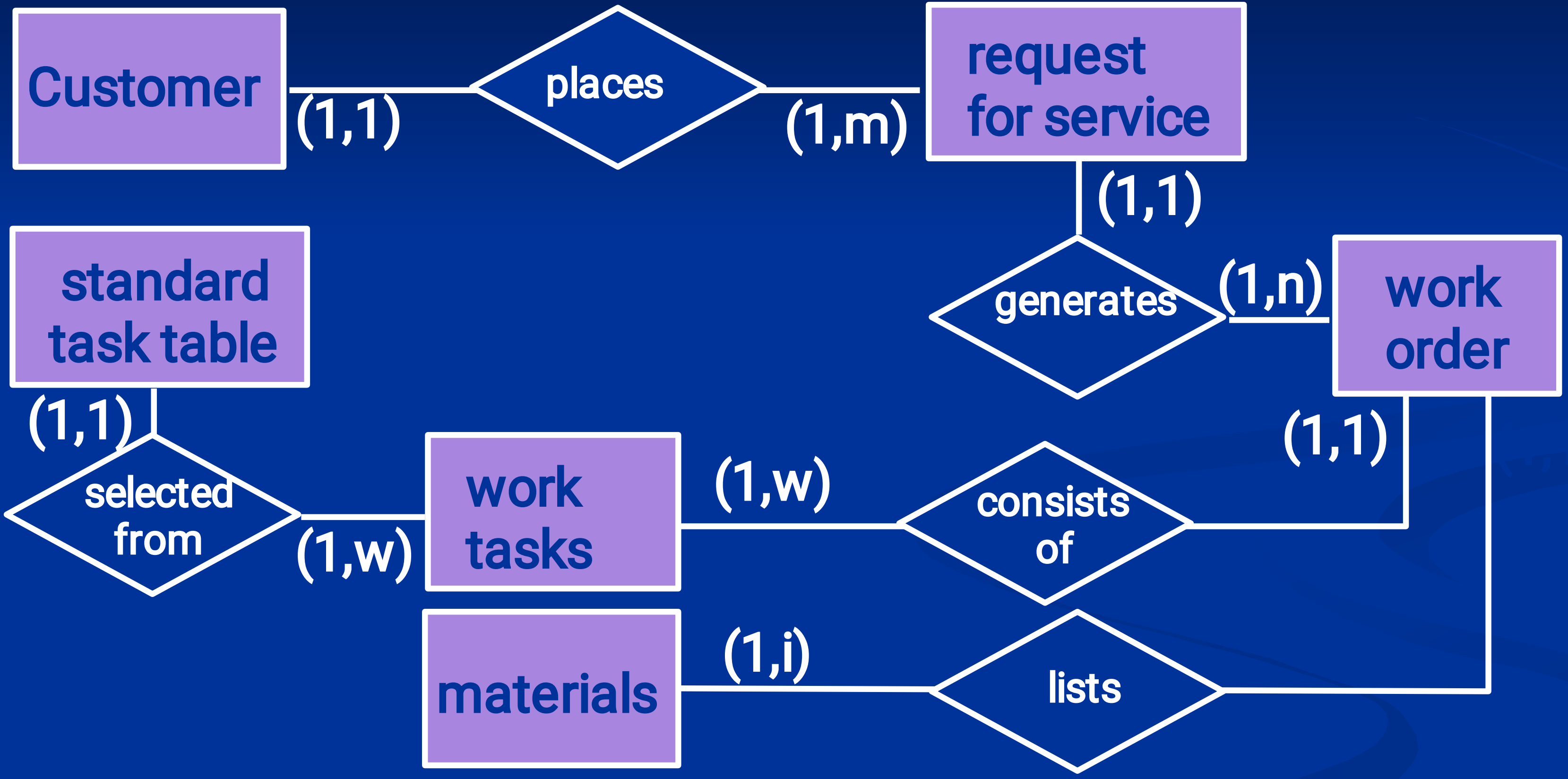
Another common form:



Building an ERD

- Level 1—model all data objects (entities) and their “connections” to one another
- Level 2—model all entities and relationships
- Level 3—model all entities, relationships, and the attributes that provide further depth

The ERD: An Example



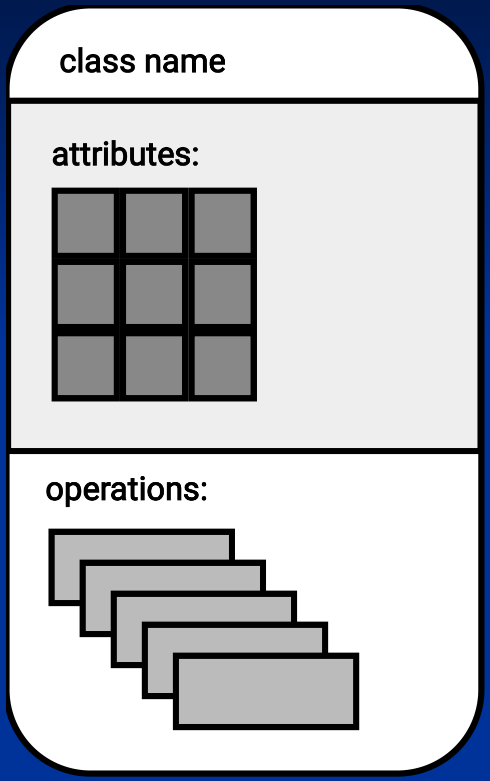
Object-Oriented Concepts

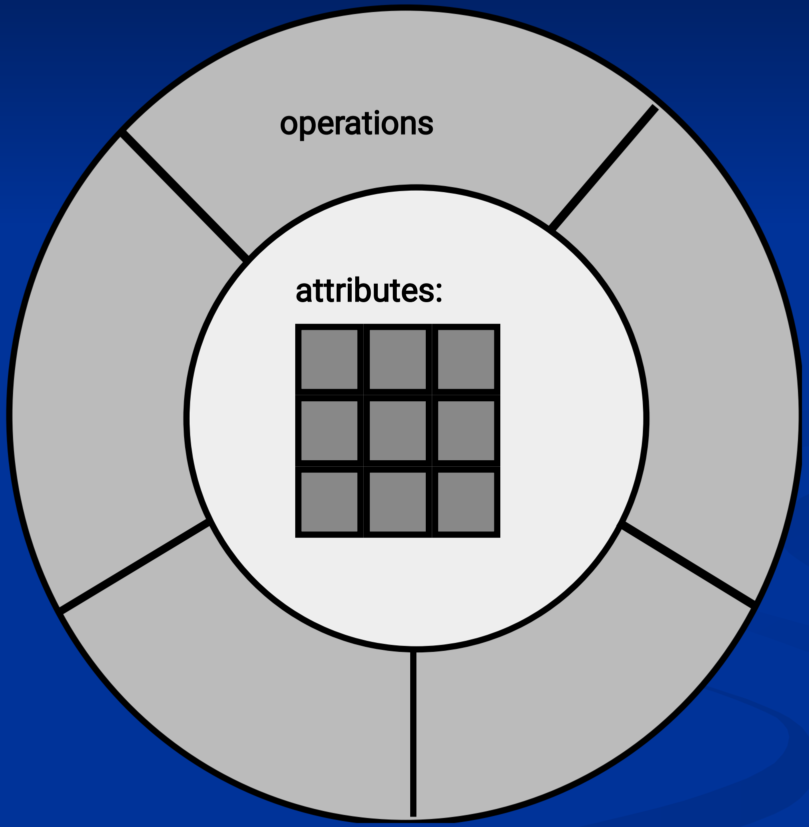
- Must be understood to apply class-based elements of the analysis model
- Key concepts:
  - Classes and objects
  - Attributes and operations
  - Encapsulation and instantiation
  - Inheritance

Classes

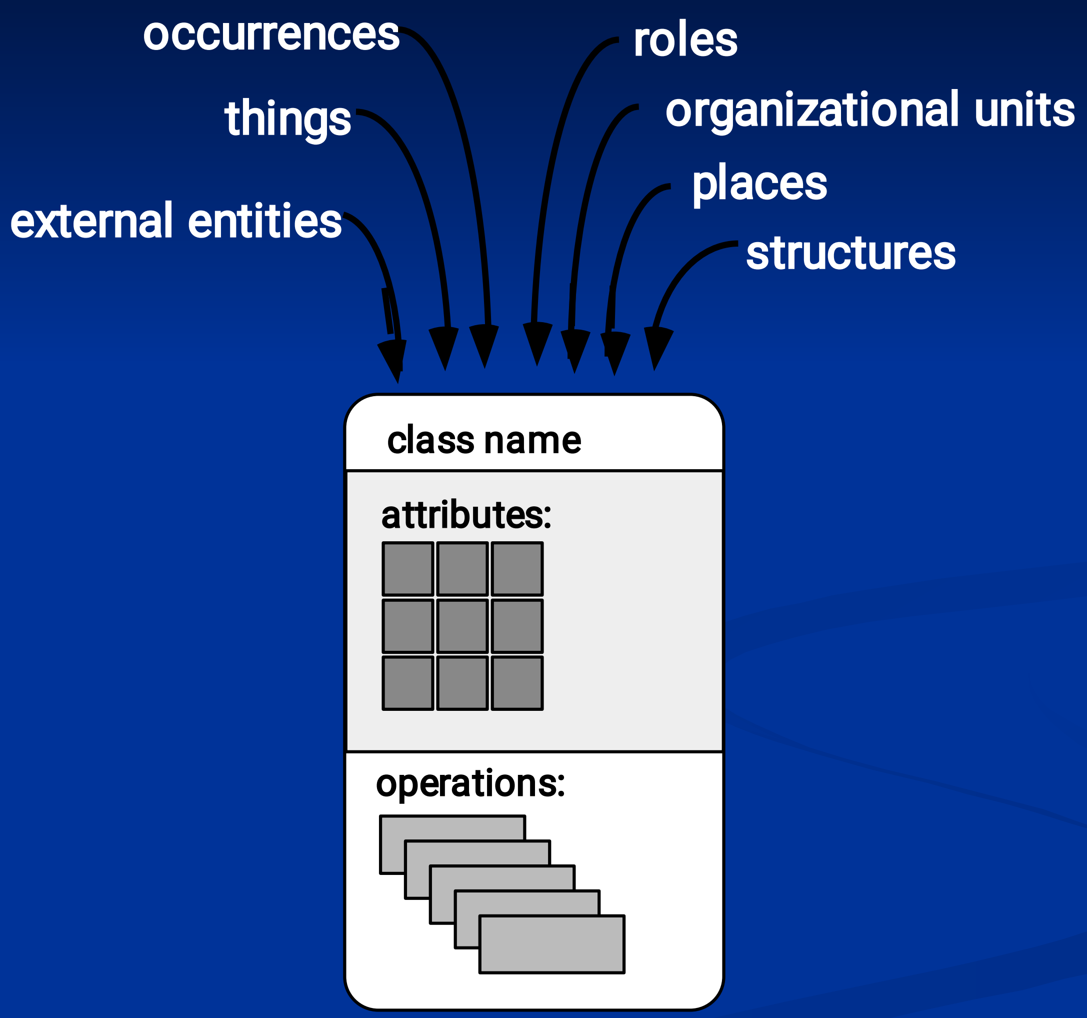
- object-oriented thinking begins with the definition of a class, often defined as:
  - template
  - generalized description
  - “blueprint” ... describing a collection of similar items
- a metaclass (also called a **superclass**) establishes a hierarchy of classes
- once a class of items is defined, a specific instance of the class can be identified

Building a Class



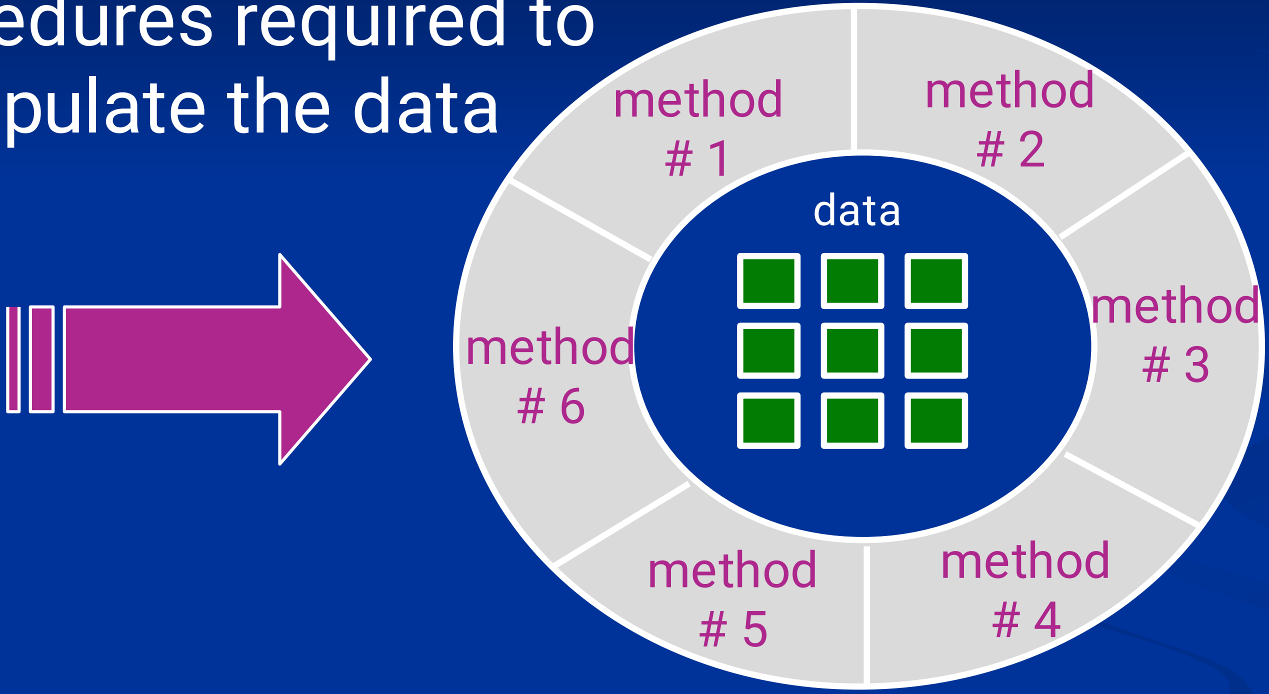


What is a Class?

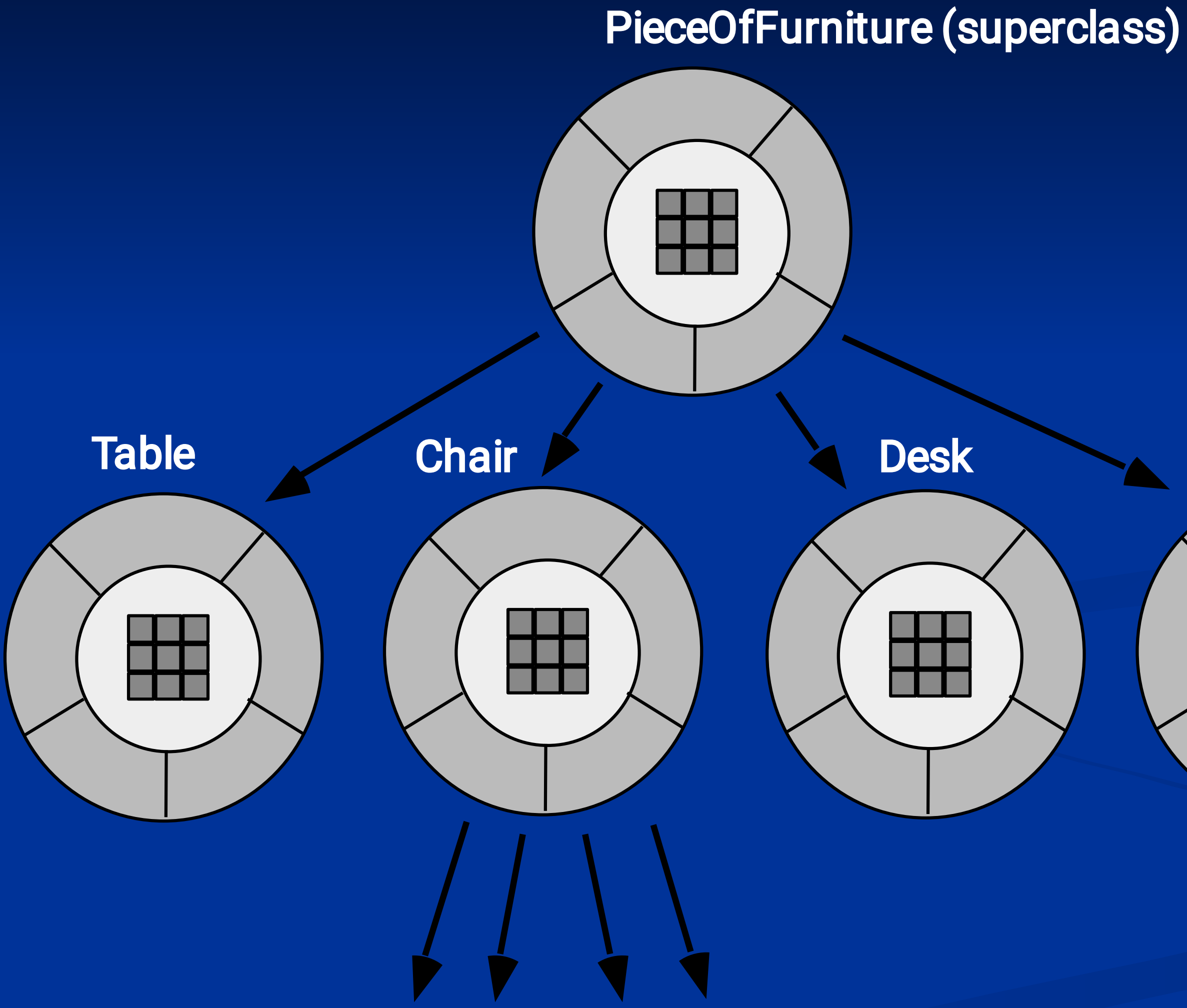


Encapsulation/Hiding

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



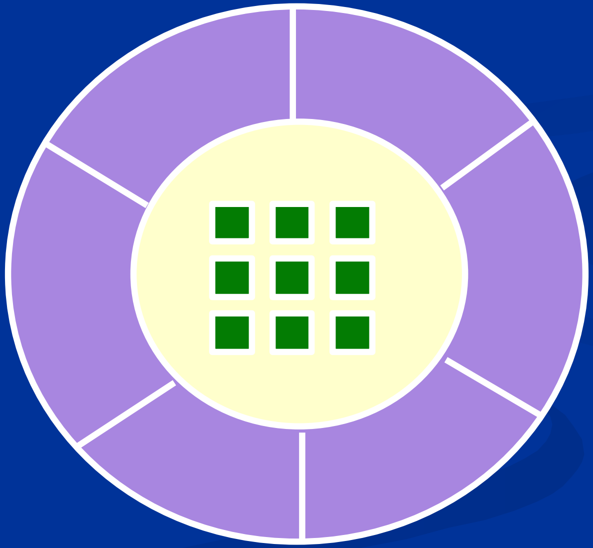
Class Hierarchy



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.



Scenario-Based Modeling

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

(1) What should we write about?

(2) How much should we write about it?

(3) How detailed should we make our description?

(4) How should we organize the description?

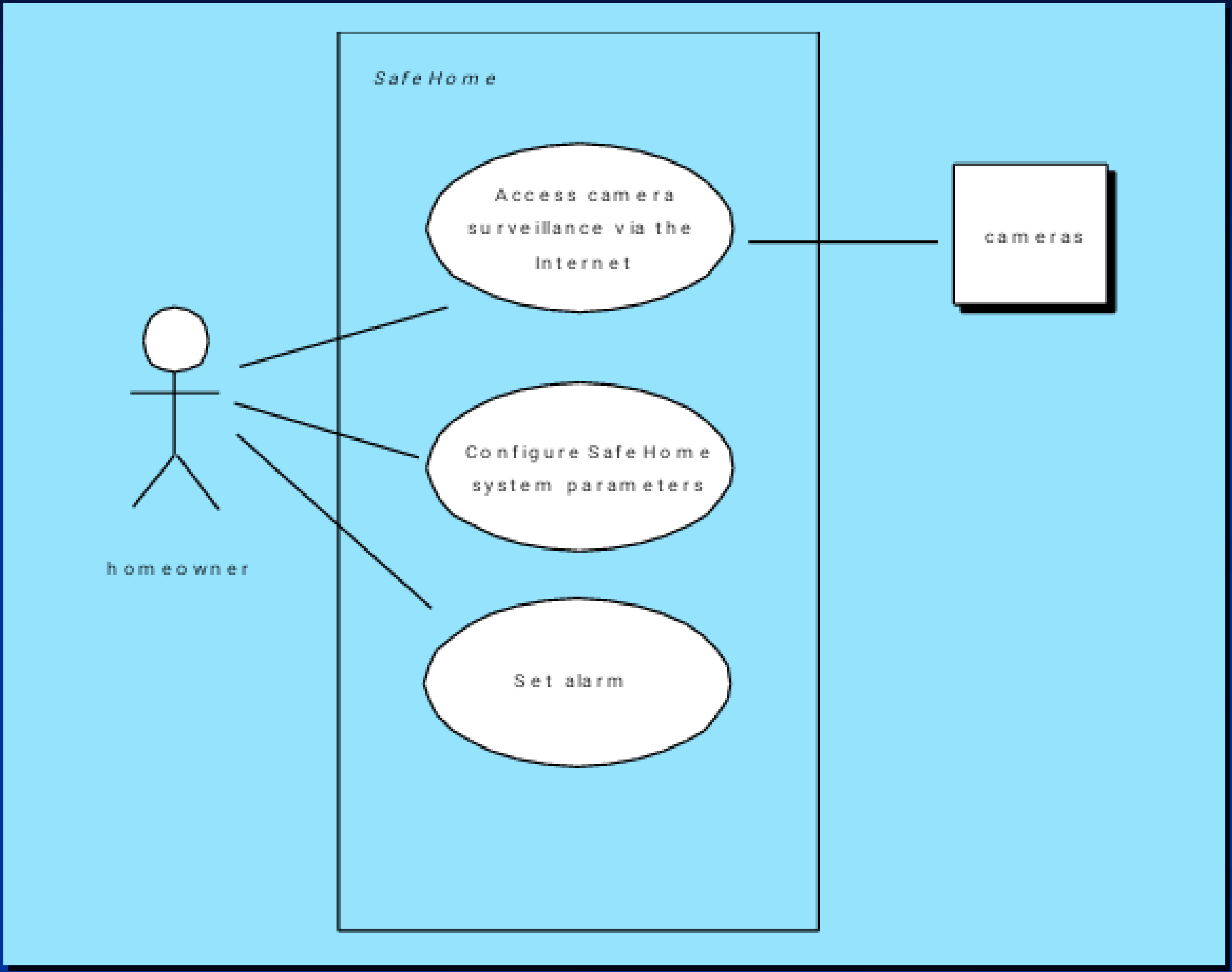
Use-Cases

- 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

Developing a Use-Case

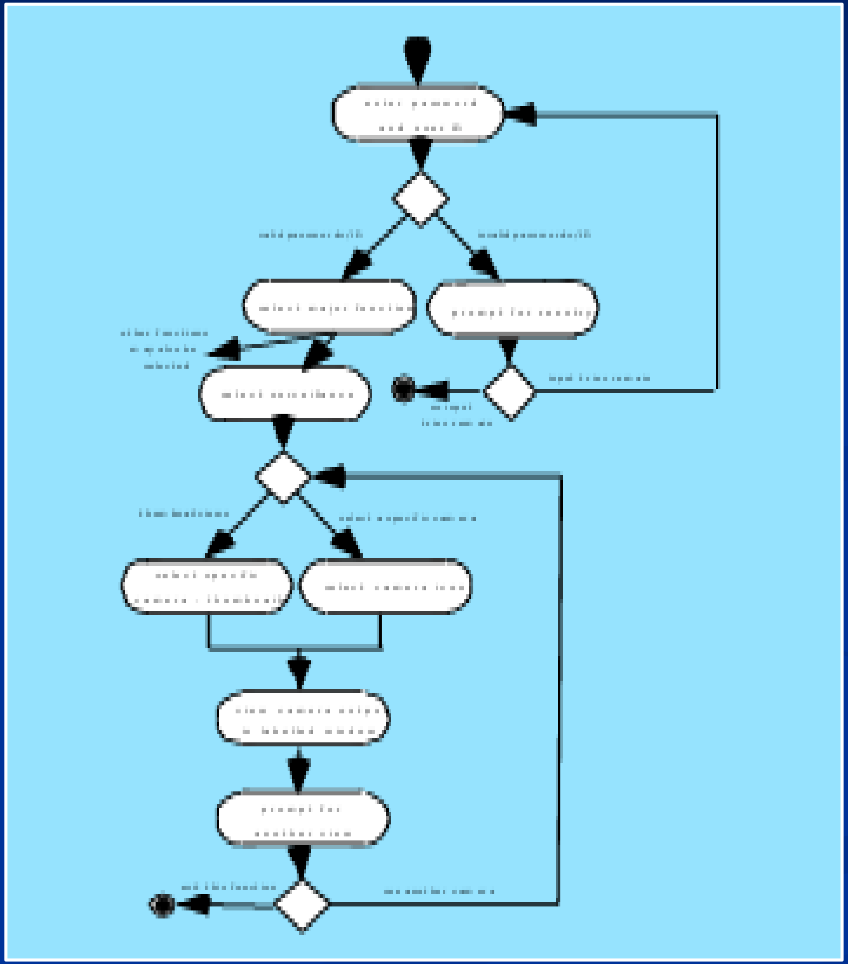
- What are the main tasks or functions that are performed by the actor?
- What system information will the the actor acquire, produce or change?
- Will the actor have to inform the system about changes in the external environment?
- What information does the actor desire from the system?
- Does the actor wish to be informed about unexpected changes?

Use-Case Diagram


SafeHome
Access camera surveillance via the Internet
Configure SafeHome system parameters
Set alarm
homeowner
cameras

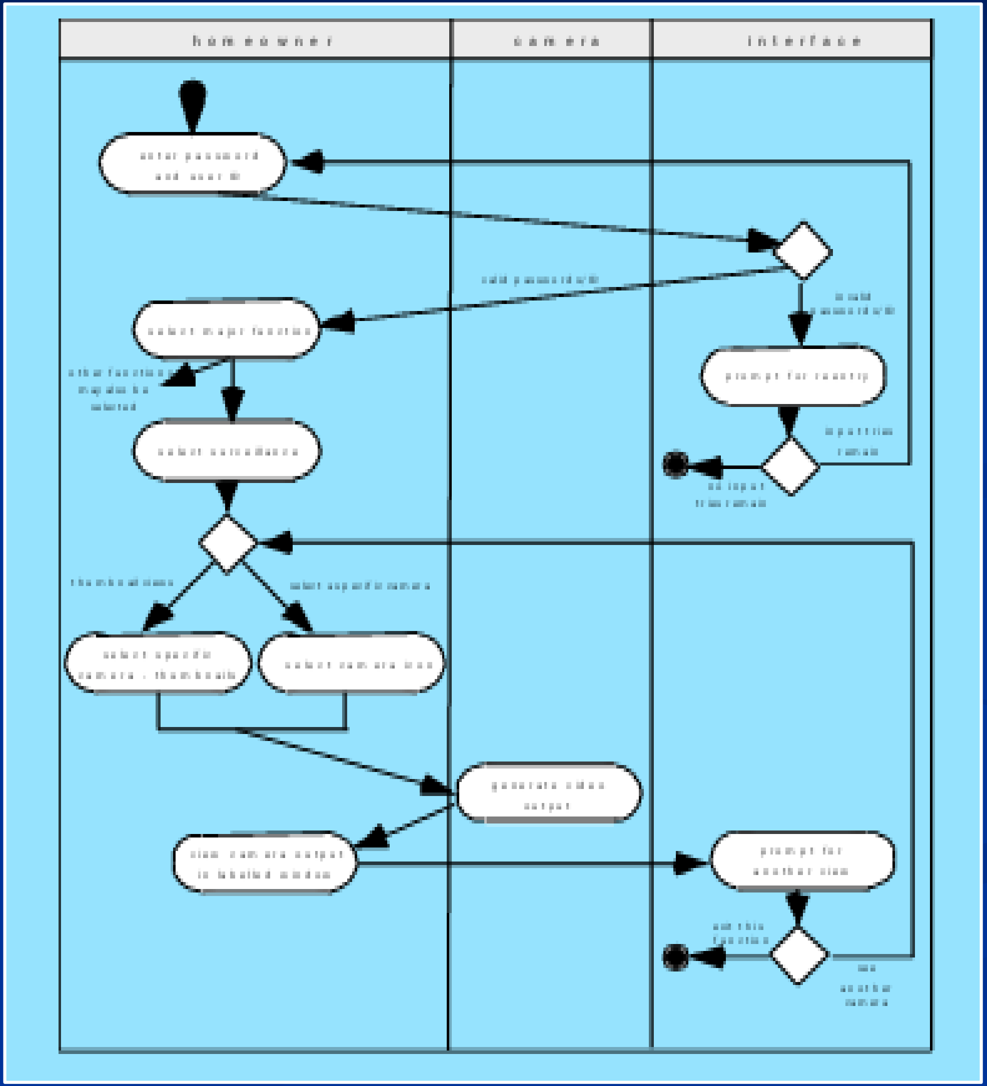
Activity Diagram

Supplements the use-case by providing a diagrammatic representation of procedural flow



Swimlane Diagrams

Allows the modeler to represent the flow of activities described by the use-case and at the same time indicate which actor (if there are multiple actors involved in a specific use-case) or analysis class has responsibility for the action described by an activity rectangle



Data Flow Diagram

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.

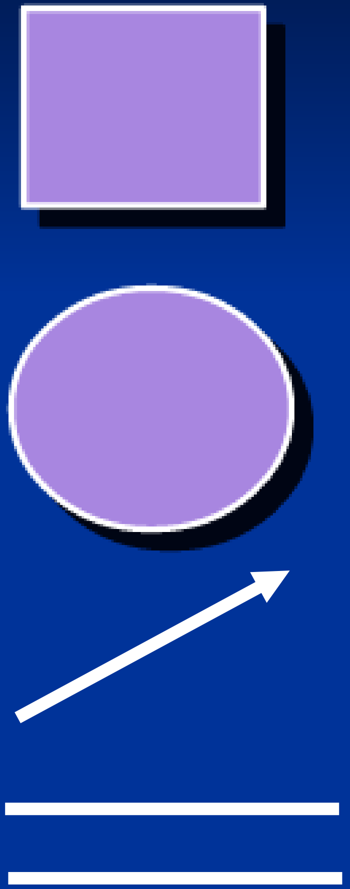
It should be used to supplement other analysis model elements

Flow-Oriented Modeling

Use a Data Flow Diagram (DFD) to show the relationships among the business processes within an organization to:

- external systems,
- external organizations,
- customers,
- other business processes.

Flow Modeling Notation



external entity

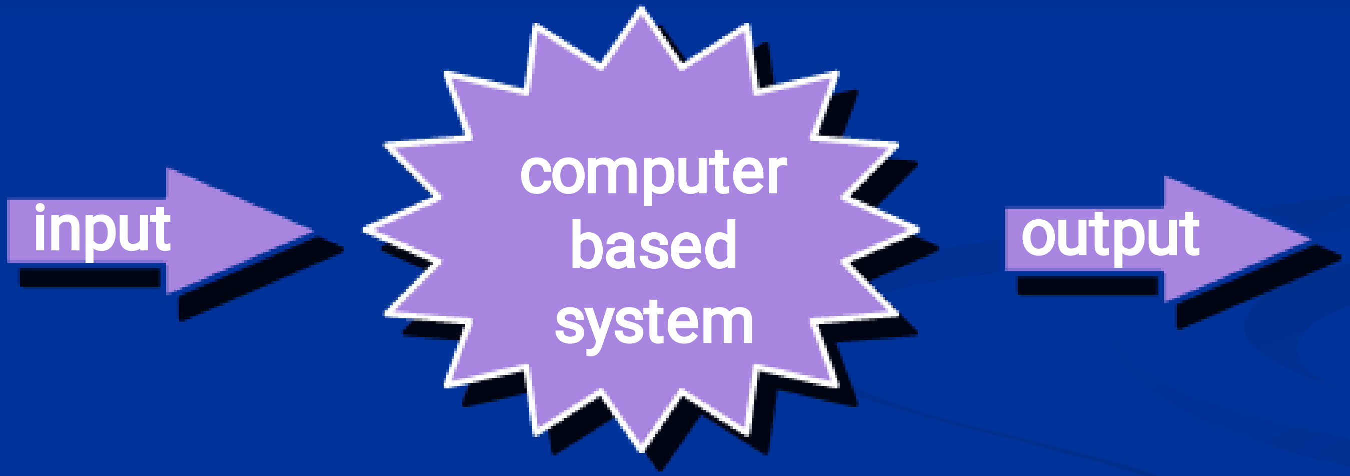
process

data flow

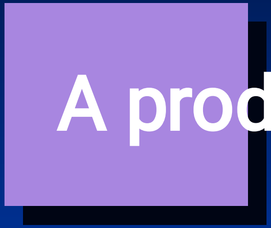
data store

The Flow Model

Every computer-based system is an information transform ....



External Entity



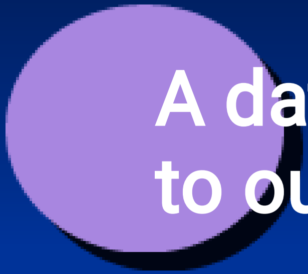
A producer or consumer of data

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



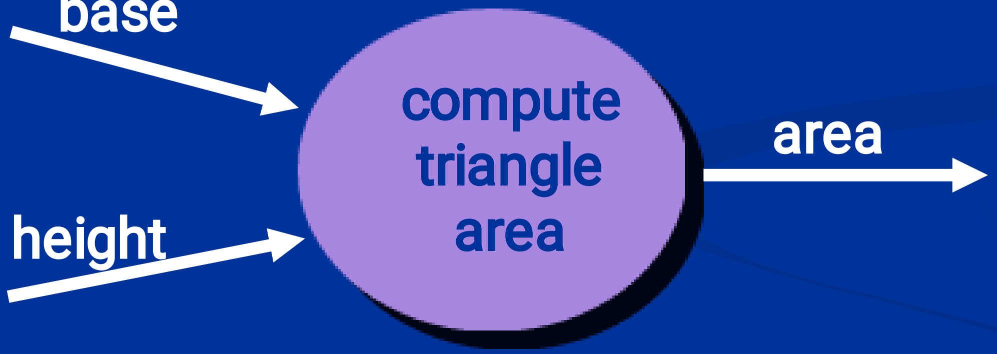
A data transformer (changes input to output)

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

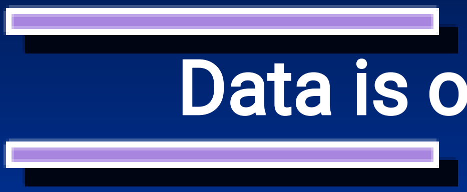
Data must always be processed in some way to achieve system function

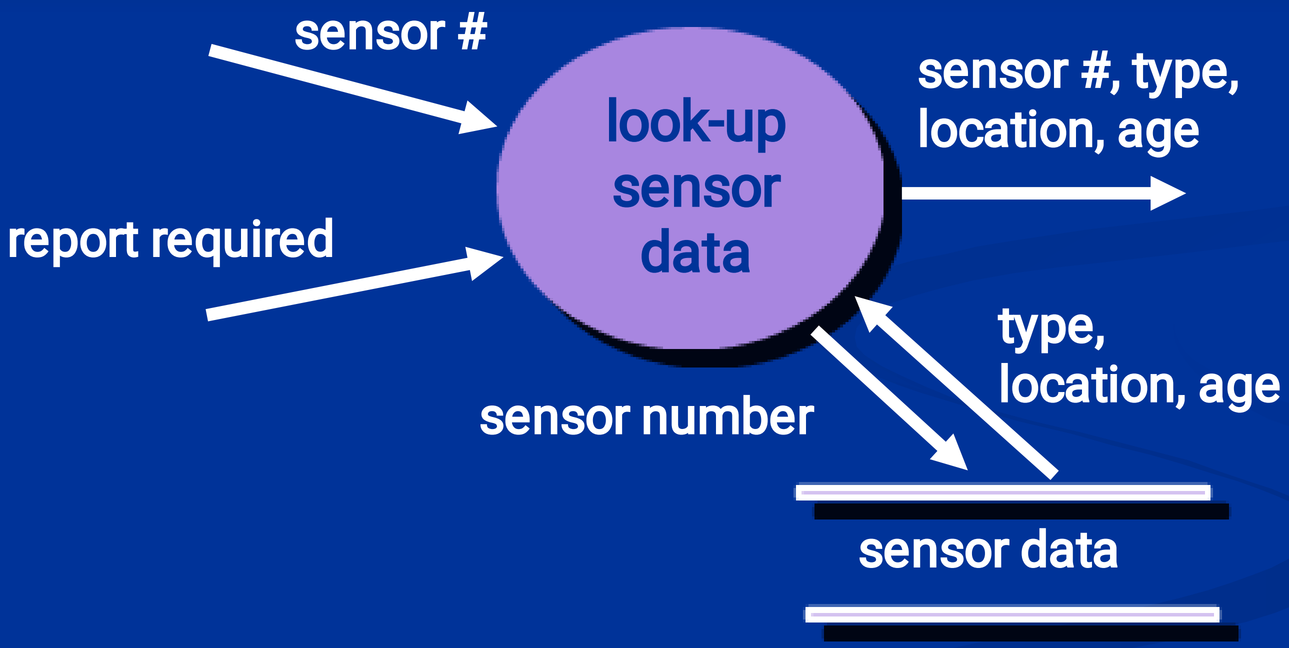
Data Flow

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



Data Stores


Data is often stored for later use.


sensor #
report required
look-up sensor data
sensor #, type, location, age
sensor number
type, location, age
sensor data

Methods

Data flow diagrams are used to describe how the system transforms information.

They define how information is processed and stored and identify how the information flows through the processes.

Methods

When building a data flow diagram, the following items should be considered:

1. where does the data that passes through the system come from and where does it go,

2. what happens to the data once it enters the system (i.e., the inputs) and before it leaves the system (i.e., the outputs),

3. what delays occur between the inputs and outputs (i.e., identifying the need for data stores) .

STEPS TO DRAW A DATA FLOW DIAGRAM

1. Start from the context diagram. Identify the parent process and the external entities with their net inputs and outputs.

2·Place the external entities on the diagram. Draw the boundary.

3·Identify the data flows needed to generate the net inputs and outputs to the external entities.

4·Identify the business processes to perform the work needed to generate the input and output data flows.

5· Connect the data flows from the external entities to the processes.

6· Identify the data stores.

STEPS TO DRAW A DATA FLOW DIAGRAM

7· Connect the processes and data stores with data flows.

8· Apply the Process Model Paradigm to verify that the diagram addresses the processing needs of all external entities.

9· Apply the External Control Paradigm to further validate that the flows to the external entities are correct.

10· Continue to decompose to the nth level DFD. Draw all DFDs at one level before moving to the next level of decomposing detail.

Modelling Rules

1. All processes must have at least one data flow in and one data flow out.
2. All processes should modify the incoming data, producing new forms of outgoing data.
3. Each data store must be involved with at least one data flow.
4. Each external entity must be involved with at least one data flow.
5. A data flow must be attached to at least one process

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

Level 0 - Context Diagram

- models system as one process box which represents scope of the system
- identifies external entities and related inputs and outputs
- 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

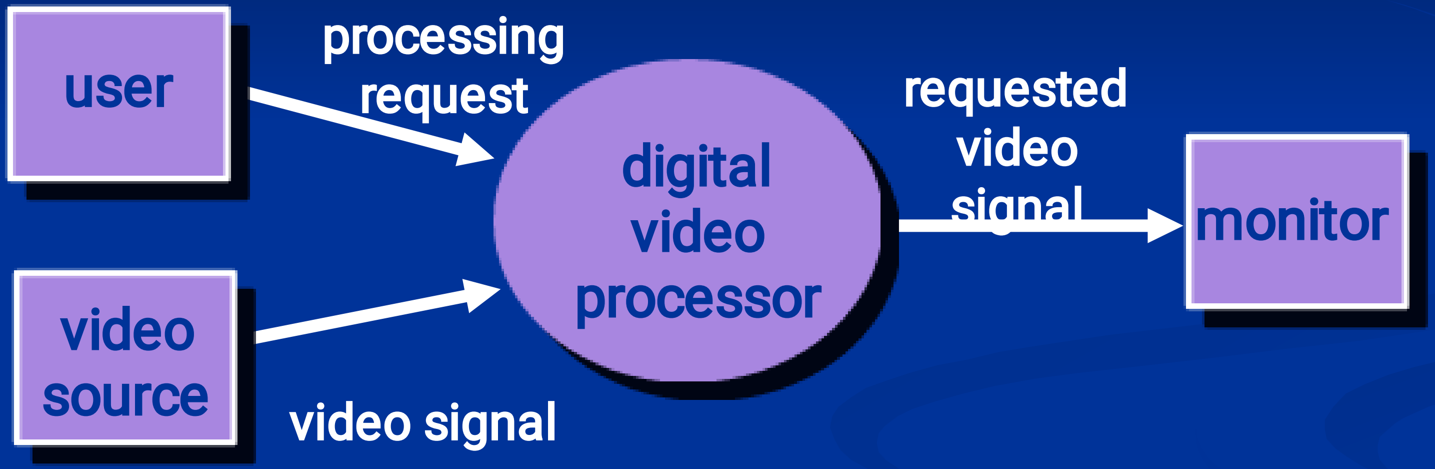
Level 1 - overview diagram

- gives overview of full system
- identifies major processes and data flows between them
- identifies data stores that are used by the major processes
- boundary of level 1 is the context diagram
- 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

Level 2 - detailed diagram

- level 1 process is expanded into more detail
- each process in level 1 is decomposed to show its constituent processes
- boundary of level 2 is the level 1 process

Level 0 DFD Example



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

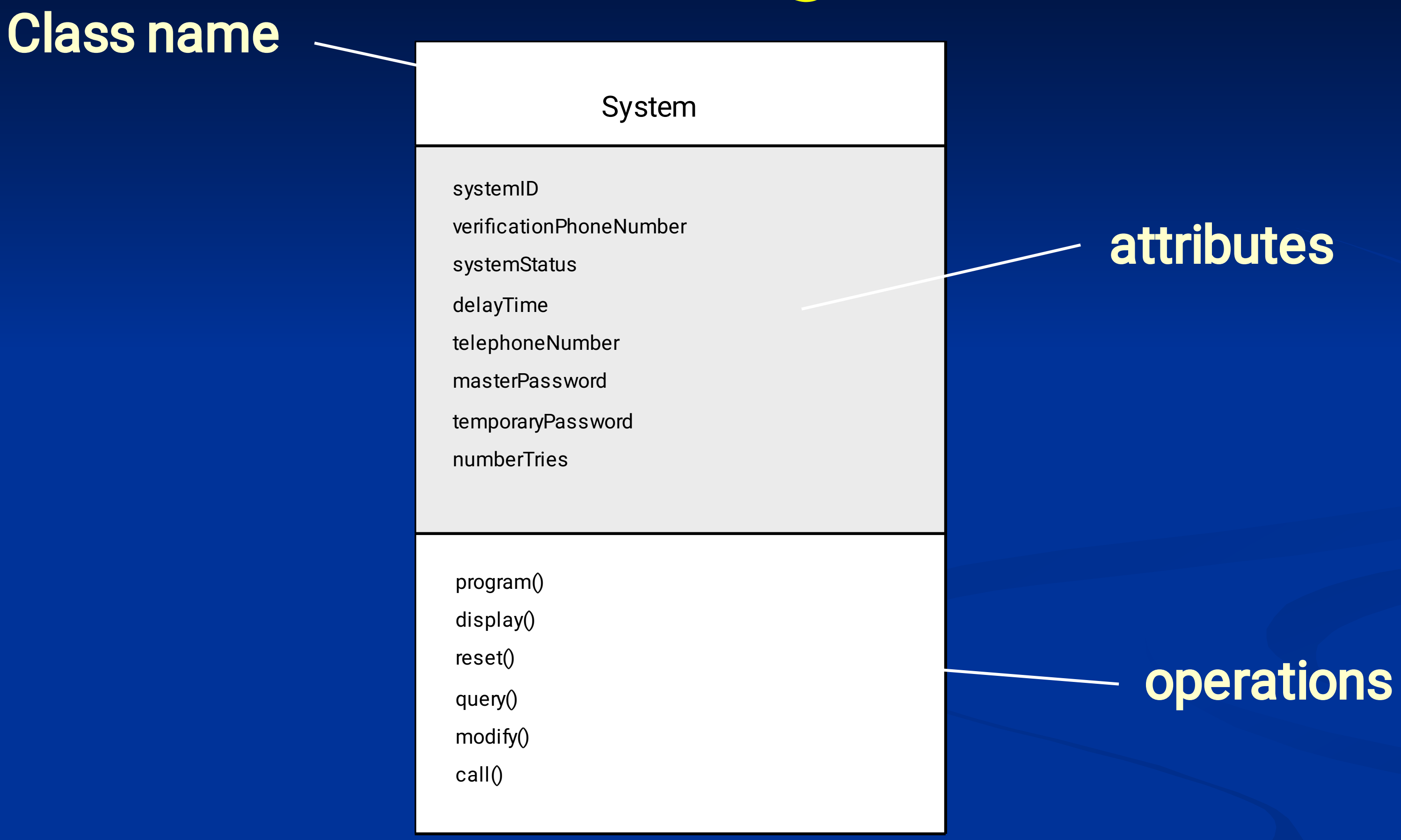
Analysis Classes

- *External entities* (e.g., other systems, devices, people) that produce or consume information to be used by a computer-based system.
- *Things* (e.g, reports, displays, letters, signals) that are part of the information domain for the problem.
- *Occurrences or events* (e.g., a property transfer or the completion of a series of robot movements) that occur within the context of system operation.
- *Roles* **(e.g., manager, engineer, salesperson) played by people who interact with the** system.
- *Organizational units* (e.g., division, group, team) that are relevant to an application.
- *Places* **(e.g., manufacturing floor or loading dock) that establish the context of the** problem and the overall function of the system.
- *Structures* (e.g., sensors, four-wheeled vehicles, or computers) that define a class of objects or related classes of objects.

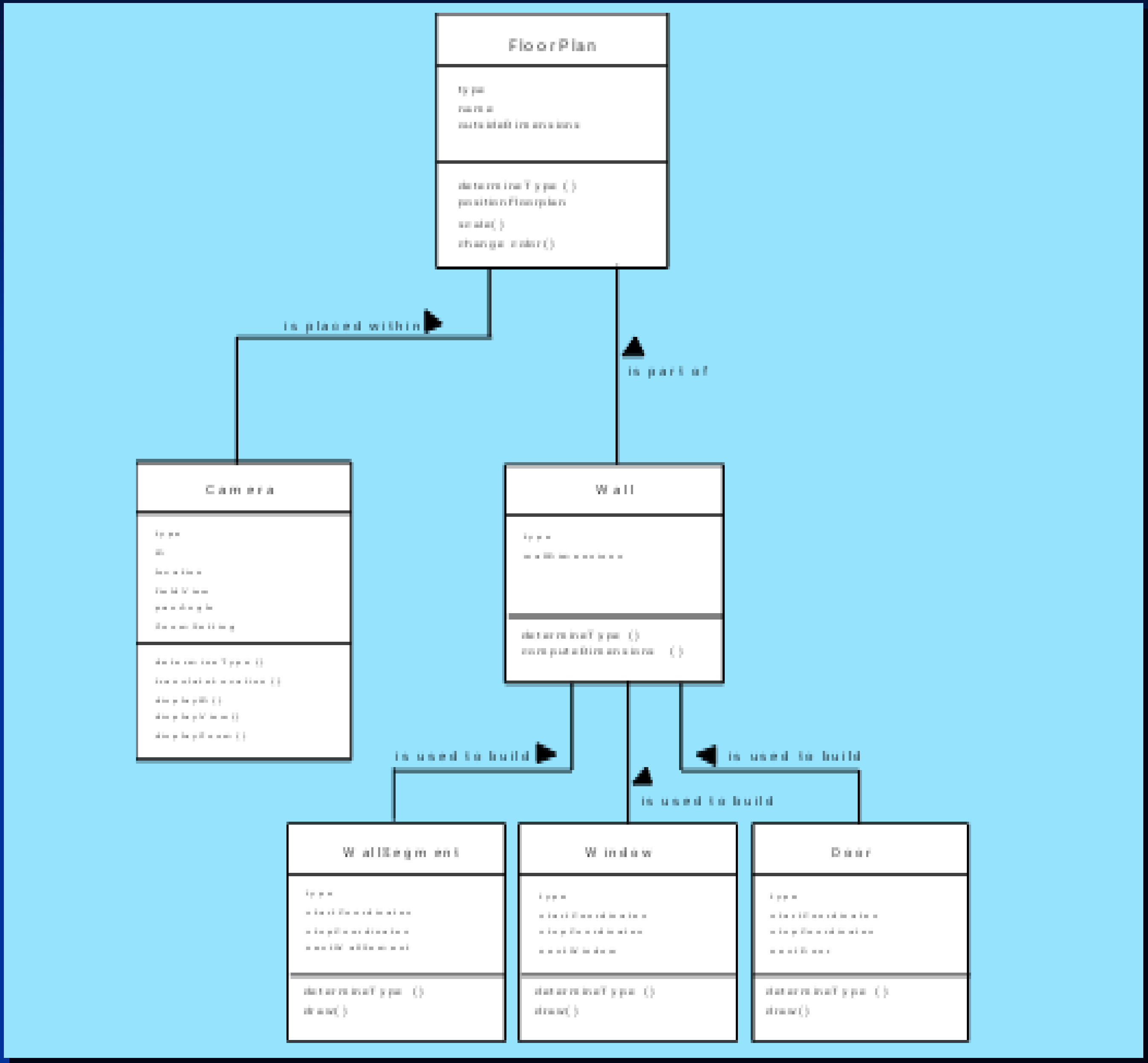
Selecting Classes—Criteria

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

Class Diagram



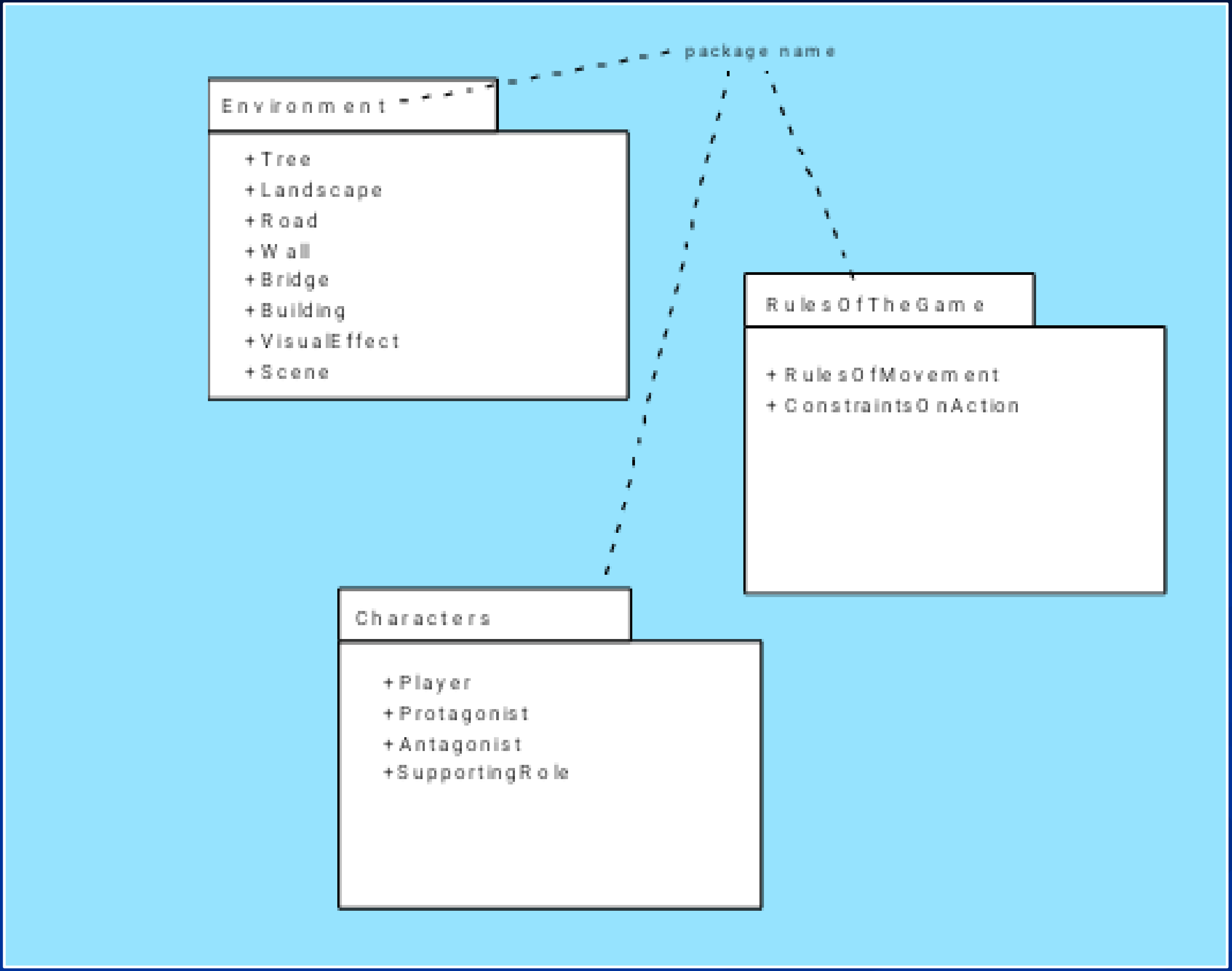
Class Diagram



Analysis Packages

- Various elements of the analysis model (e.g., use-cases, analysis classes) are categorized in a manner that packages them as a grouping
- The plus sign preceding the analysis class name in each package indicates that the classes have public visibility and are therefore accessible from other packages.
- Other symbols can precede an element within a package. A minus sign indicates that an element is hidden from all other packages and a # symbol indicates that an element is accessible only to packages contained within a given package.

Analysis Packages



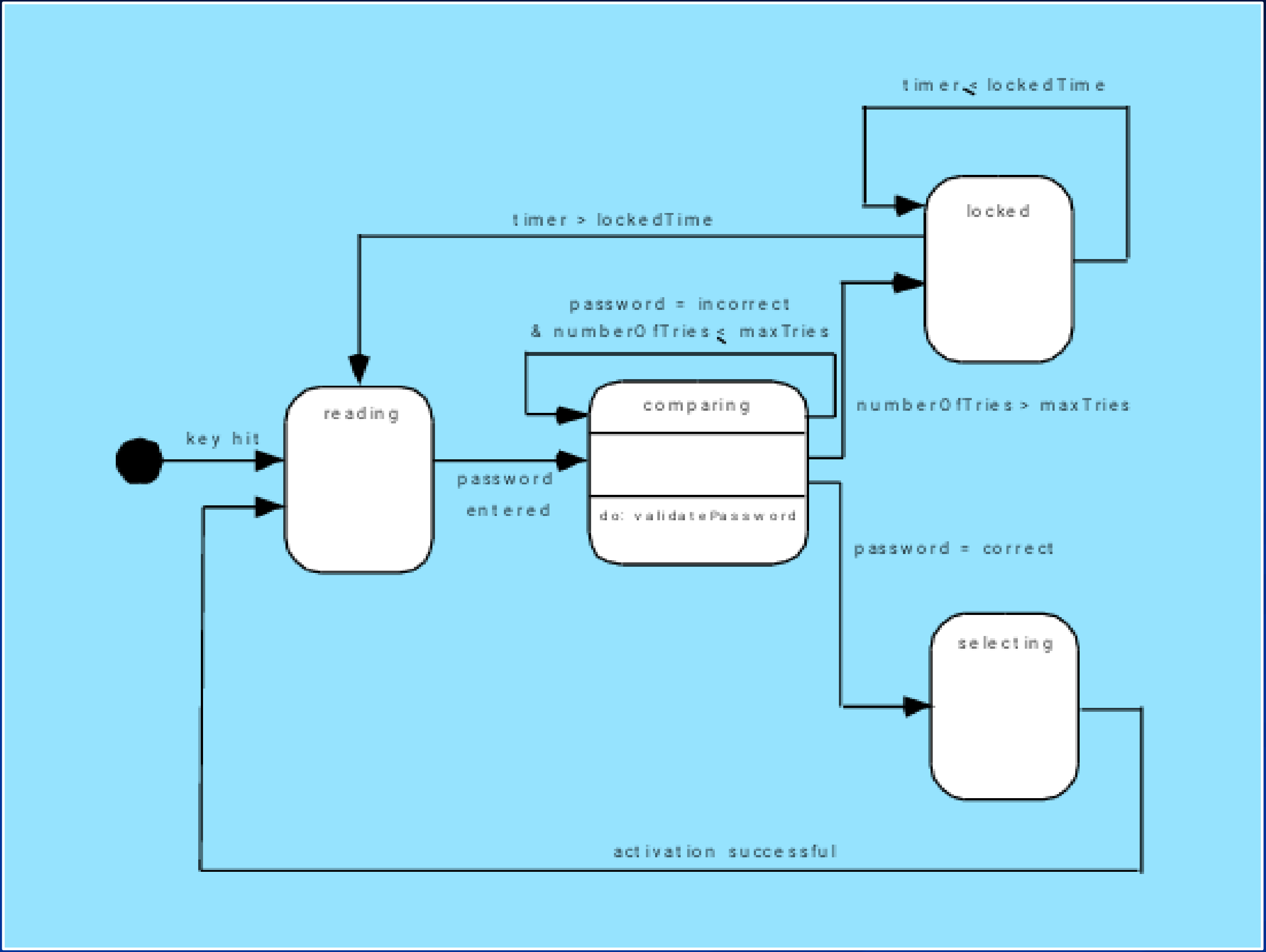
Behavioral Modeling

- The behavioral model indicates how software will respond to external events or stimuli. To create the model, the analyst must perform the following steps:
  - Evaluate all use-cases to fully understand the sequence of interaction within the system.
  - Identify events that drive the interaction sequence and understand how these events relate to specific objects.
  - Create a sequence for each use-case.
  - Build a state diagram for the system.
  - Review the behavioral model to verify accuracy and consistency.

State Representations

- In the context of behavioral modeling, two different characterizations of states must be considered:
  - the state of each class as the system performs its function and
  - the state of the system as observed from the outside as the system performs its function
- The state of a class takes on both passive and active characteristics [CHA93].
  - A *passive state* is simply the current status of all of an object's attributes.
  - The *active state* of an object indicates the current status of the object as it undergoes a continuing transformation or processing.

State Diagram for the ControlPanel Class



The States of a System

- state—a set of observable circum-stances that characterizes the behavior of a system at a given time
- state transition—the movement from one state to another
- event—an occurrence that causes the system to exhibit some predictable form of behavior
- action—process that occurs as a consequence of making a transition

Behavioral Modeling

- make a list of the different states of a system (How does the system behave?)
- indicate how the system makes a transition from one state to another (How does the system change state?)
  - indicate event
  - indicate action
- draw a **state diagram or a sequence** diagram

Sequence Diagram

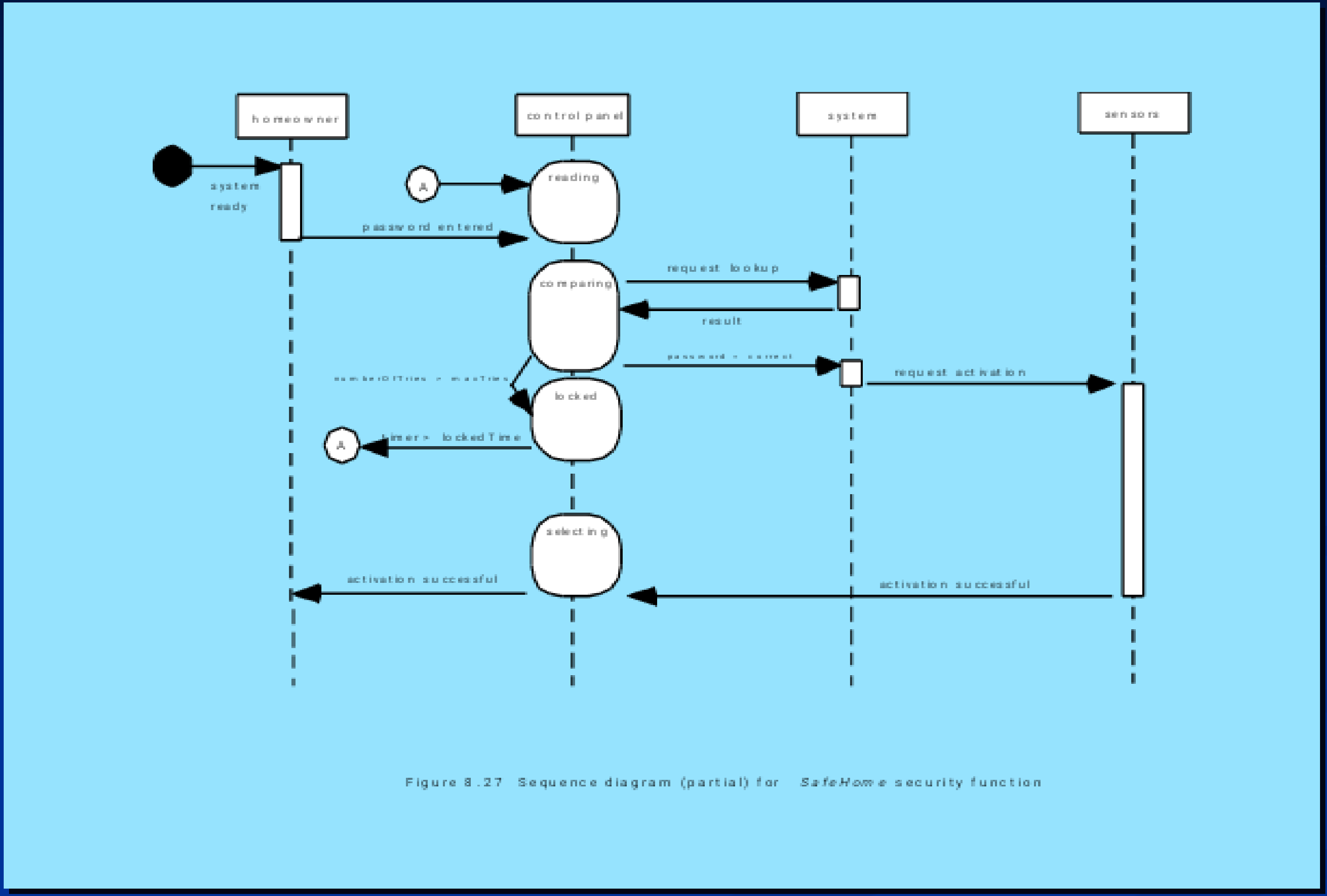


Figure 8.27 Sequence diagram (partial) for *SafeHome* security function