Design and Modeling using UML

Marwadi University

Computer Engineering Diploma

Unit 5:-
Design and Modeling using UML

Software Engineering (09CE2402)

Design Concepts

- **Software Design --** An iterative process of transforming requirements into a “blueprint” for constructing the software.

Characteristics of Good Software design

- **Correctness:** A good design should correctly implement all the functionalities identified in the SRS document.
- **Understandability:** A good design is easily understandable.
- **Efficiency:** It should be efficient.
- **Maintainability:** It should be easily amenable(controllable) to change.

Effective Modular Design

- Modularization is the process of dividing a software system into multiple independent modules where each module works independently
- A modular design reduces complexity, facilitates change (a critical aspect of software maintainability), and results in easier implementation by encouraging parallel development of different parts of a system.

Effective Modular Design

- **Modular Decomposability** - provides systematic means for breaking problem into subproblems.
- **Modular Composability** - supports reuse of existing modules in new systems
- **Modular Understandability** - module can be understood as a stand-alone unit
- **Modular Continuity** - side-effects due to module changes minimized
- **Modular Protection** - side-effects due to processing errors minimized

Functional Independence

Functional Independence

- **Process of dividing system into independently implementable parts**
- Each module should address a specific sub function of requirements and have a simple interface.
- Functional independent modules are easier to develop, maintain, and test
- Error propagation is reduced and reusable modules are possible

Need of Functional Independence

❑ **Error isolation:**

- Functional independence reduces error propagation.
- The reason behind this is that if a module is functionally independent, its degree of interaction with the other modules is less.
- Therefore, any error existing in a module would not directly effect the other modules.

Need of Functional Independence

❑**Scope of reuse:**

- Reuse of a module becomes possible.
- Because each module does some well-defined and precise function, and the interaction of the module with the other modules is simple and minimal.
- Therefore, a module can be easily taken out and reused in a different program.

Need of Functional Independence

❑**Understandability**

- Complexity of the design is reduced, because different modules can be understood in isolation as modules are more or less independent of each other.

Functional Independence

- Assessed using two qualitative criteria:

1) **Cohesion**

2) **Coupling**

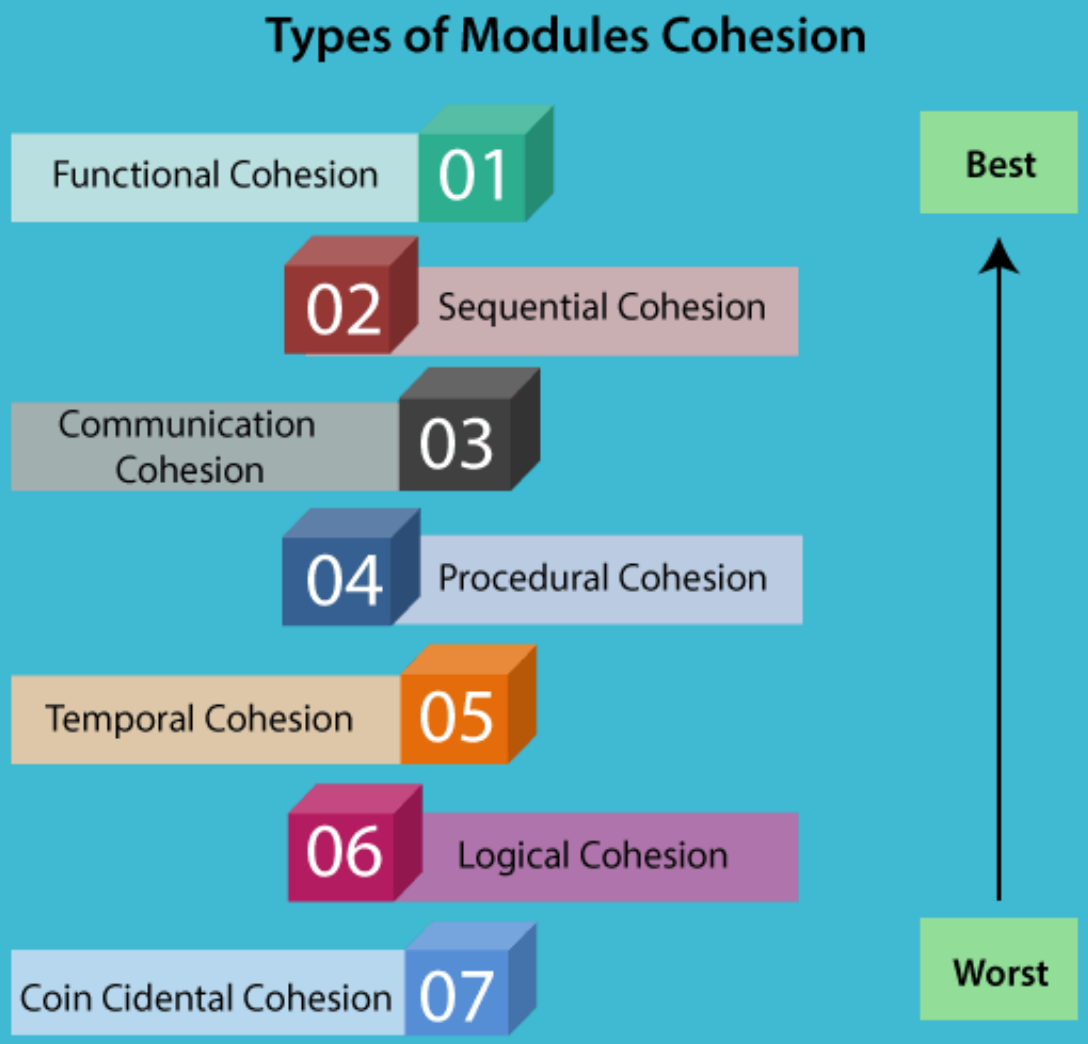
(1) Cohesion - degree to which a module focuses on just one thing

(2) Coupling - degree to which a module is connected to other modules and to the outside world

COHESION

- A cohesive module performs a single task within a software procedure, requiring little interaction with procedures being performed in other parts of a program.
- Stated simply, a cohesive module should (ideally) do just one thing.
- A good software design will have high cohesion.

Types of Cohesion



Types of Cohesion

- **Functional Cohesion:** Functional Cohesion is said to exist if the different elements of a module, cooperate to achieve a single function.
- A functional cohesion performs the task and functions.

Types of Cohesion

- **Sequential Cohesion:** A module is said to possess sequential cohesion if the element of a module form the components of the sequence, where the output from one component of the sequence is input to the next.
- Example: getting data from database

1. Get result set from sql command
2. prepare result set
3. return result set

Types of Cohesion

- **Communicational Cohesion:** Two elements operate on the same input data or contribute towards the same output data.
  - Example- update record in the database and send it to the printer.
- **Procedural Cohesion:** Elements of procedural cohesion ensure the order of execution. Actions are still weakly connected and unlikely to be reusable.
  - Example- calculate student GPA, print student record, calculate cumulative GPA, print cumulative GPA.

Types of Cohesion

- **Temporal Cohesion:** When a module includes functions that are associated by the fact that all the methods must be executed in the same time, the module is said to exhibit temporal cohesion.
- Example1: a function which is called after catching an exception which closes open files, creates an error log, and notifies the user.
- Example2: A "Termination" procedure which does the following:
    a. Writes array contents to disk files.
    b. Closes all files.

Types of Cohesion

- **Logical Cohesion:** A module is said to be logically cohesive if all the elements of the module perform a similar operation. For example Error handling, data input and data output, etc.

Types of Cohesion

❑**Coincidental Cohesion:** Coincidental cohesion is when parts of a module are grouped arbitrarily, the only relationship between the parts is that they have been grouped together.

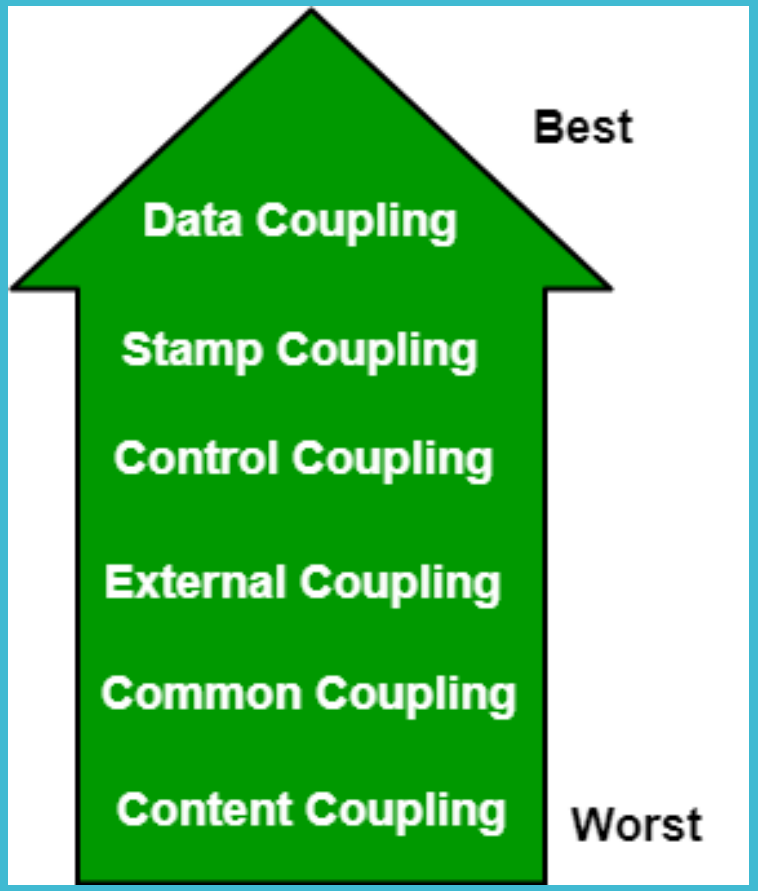
- The elements are not related(unrelated).
- It is accidental and the worst form of cohesion.
- Example- print next line and reverse the characters of a string in a single component.

Coupling

COUPLING

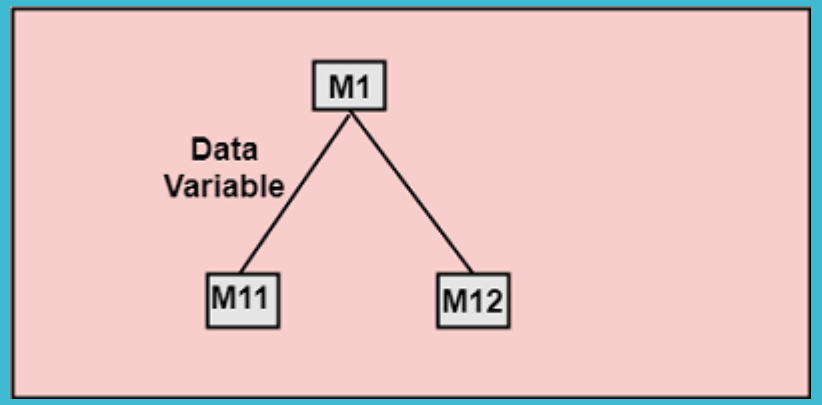
- Coupling is the measure of the degree of interdependence between the modules.
- A good software will have low coupling.

Types of Coupling



Types of Coupling

(1) Data Coupling: When data of one module is passed to another module, this is called data coupling.



- Example-customer billing system.

Types of Coupling

(2) Stamp Coupling: In stamp coupling, the complete data structure is passed from one module to another module.

For example, passing structure variable in C or object in C++ language to a module.

Types of Coupling

(3) Control Coupling: If the modules communicate by passing control information, then they are said to be control coupled.

- one module controls the flow of another, such as by passing it a variable or other information
- Example- sort function that takes comparison function as an argument.

Types of Coupling

(4) External Coupling: In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware.

- Example- protocol, external file, device format, etc.

Types of Coupling

(5) Common Coupling: The modules have shared data such as global data structures.

- The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change.
- So it has got disadvantages like difficulty in reusing modules, reduced ability to control data accesses and reduced maintainability.

Types of Coupling

(6) Content Coupling: In a content coupling, one module can modify the data of another module or control flow is passed from one module to the other module.

- This is the worst form of coupling and should be avoided.

The Three Models

The Three Models in UML

- **The Class Model:** The class model is the graphical representation of the structure of the system and also the relation between the objects and classes in the system.
- **The State Model:** A state diagram, sometimes known as a state machine diagram, is a type of behavioral diagram in the Unified Modeling Language (UML) that shows transitions between various objects.
- **The Interaction Model:** The Interaction model represents the collaboration of individual objects, the “interaction” aspects of a system.

The Class Model

The Class Model

The class model captures the static structure of a system by characterizing the objects in the system, the relationship between the objects and the attributes and operations for each class of objects.

Object and Class concepts

Objects:-

- The purpose of class modeling is to describe objects.
- An object is a concept, abstraction, or thing with identity that has meaning for an application.
- All objects have identity and are distinguishable.
- Two apples with the same color, size, shape and texture are still individual apples.

Object and Class concepts

Classes:-

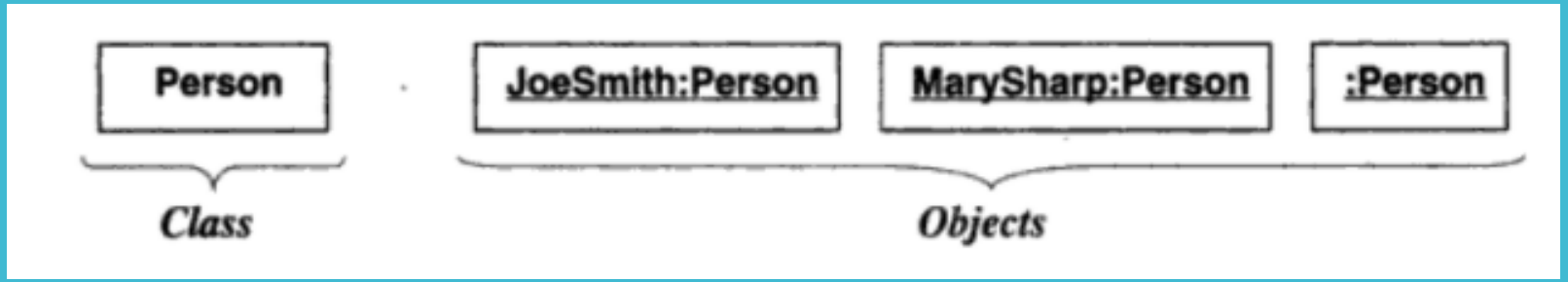
- An object is an instance-or occurrence of a class.
- A class describes a group of objects with the same properties(attributes), behavior(Operations), kinds of relationships.

E.x:- Person, company, process, and window all are classes

Class Diagram

Class Diagram

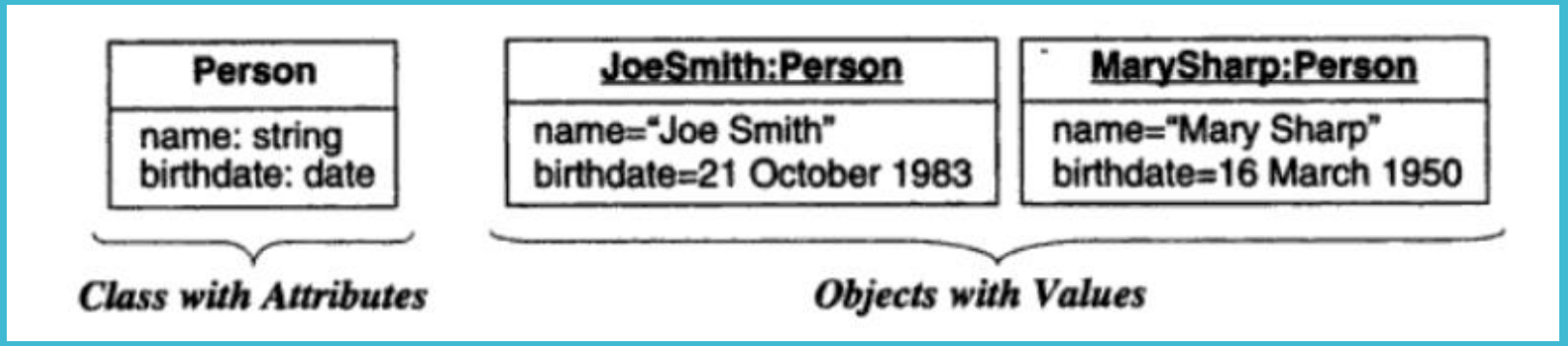
- Class diagram provide a graphic notation for modeling classes and their relationships, thereby describing possible objects.



- The UML Symbol for a class is a box.
- Objects names are separated by colon symbol along with class name

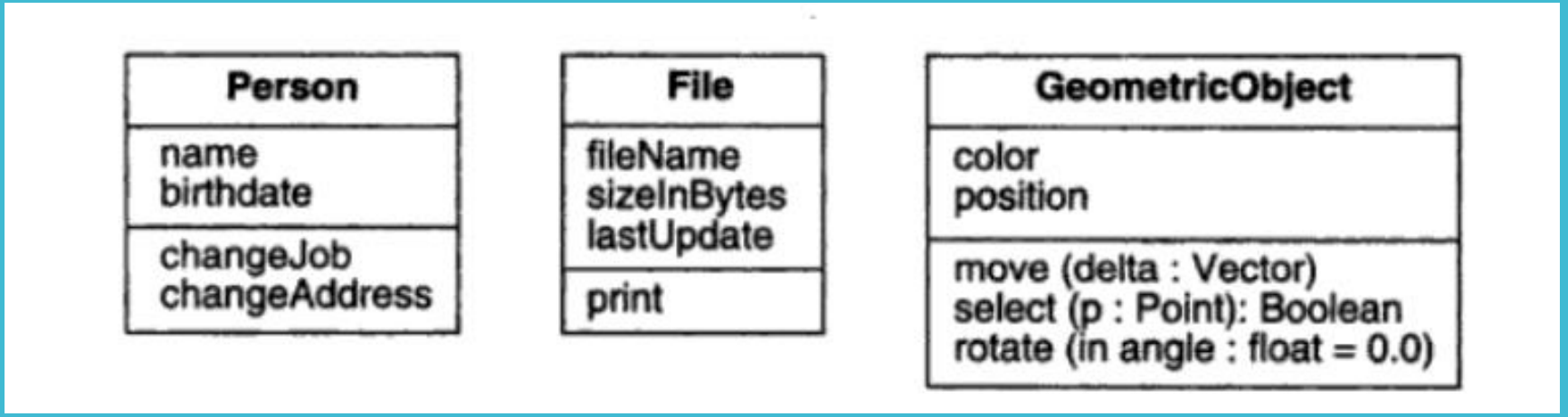
Value and Attributes

- A value is a piece of data.
- An attribute is a named property of a class that describes a value held by each object of the class.
- Name, Birthdate, and weight are attributes of Person object.
- Color, Model year and weight are attributes of Car object.
- Each attribute has a value for each object.
- For example, attribute Birthdate has value “21 October 1983”.



Operations and Methods

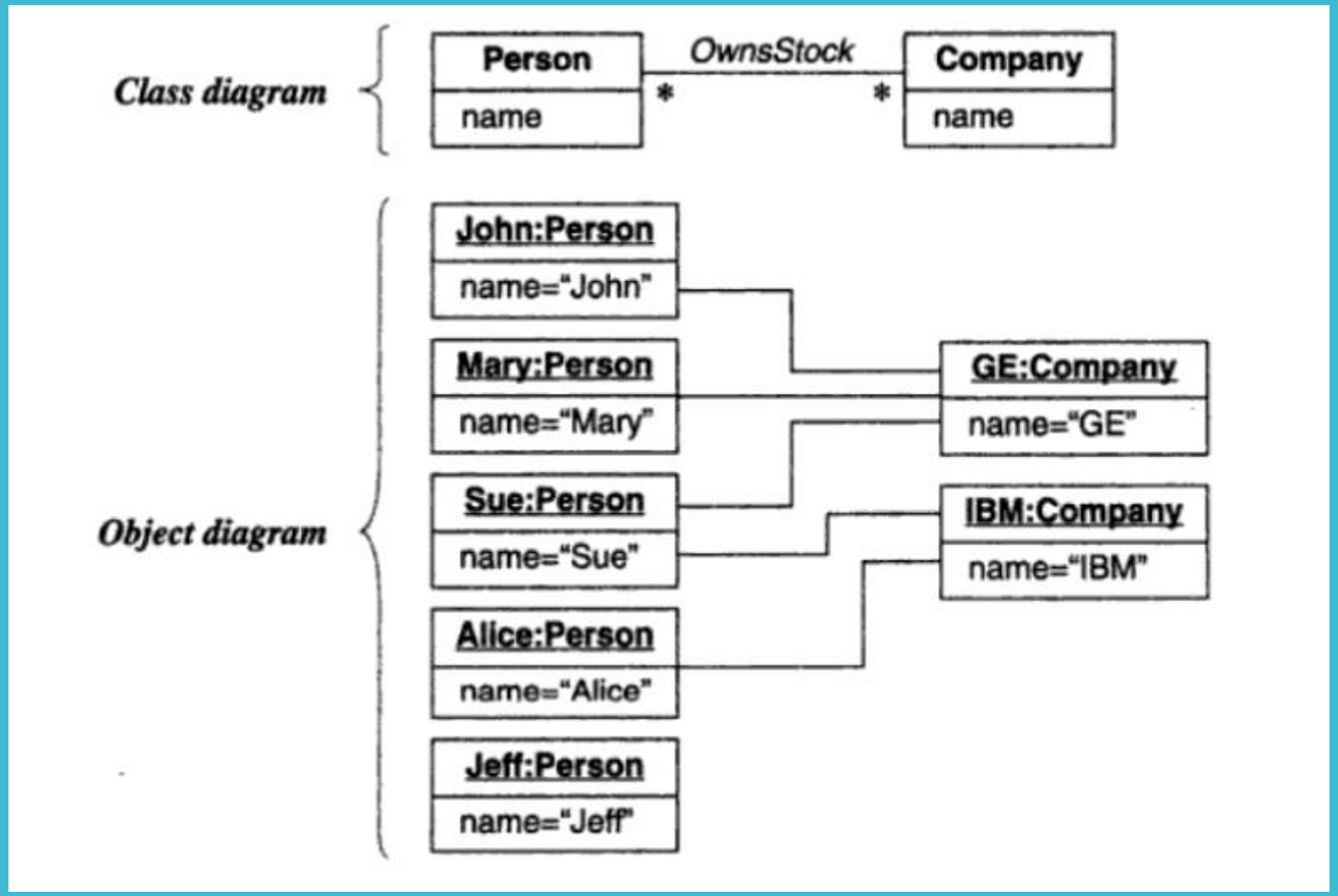
- An operation is a function or procedure that may be applied to or by objects in a class.
- Hire, Fire, Pay dividend are operations on class Company.
- Open, Close are operations on class Window.
- The UML notation is to list operations in the third compartment of the class box.



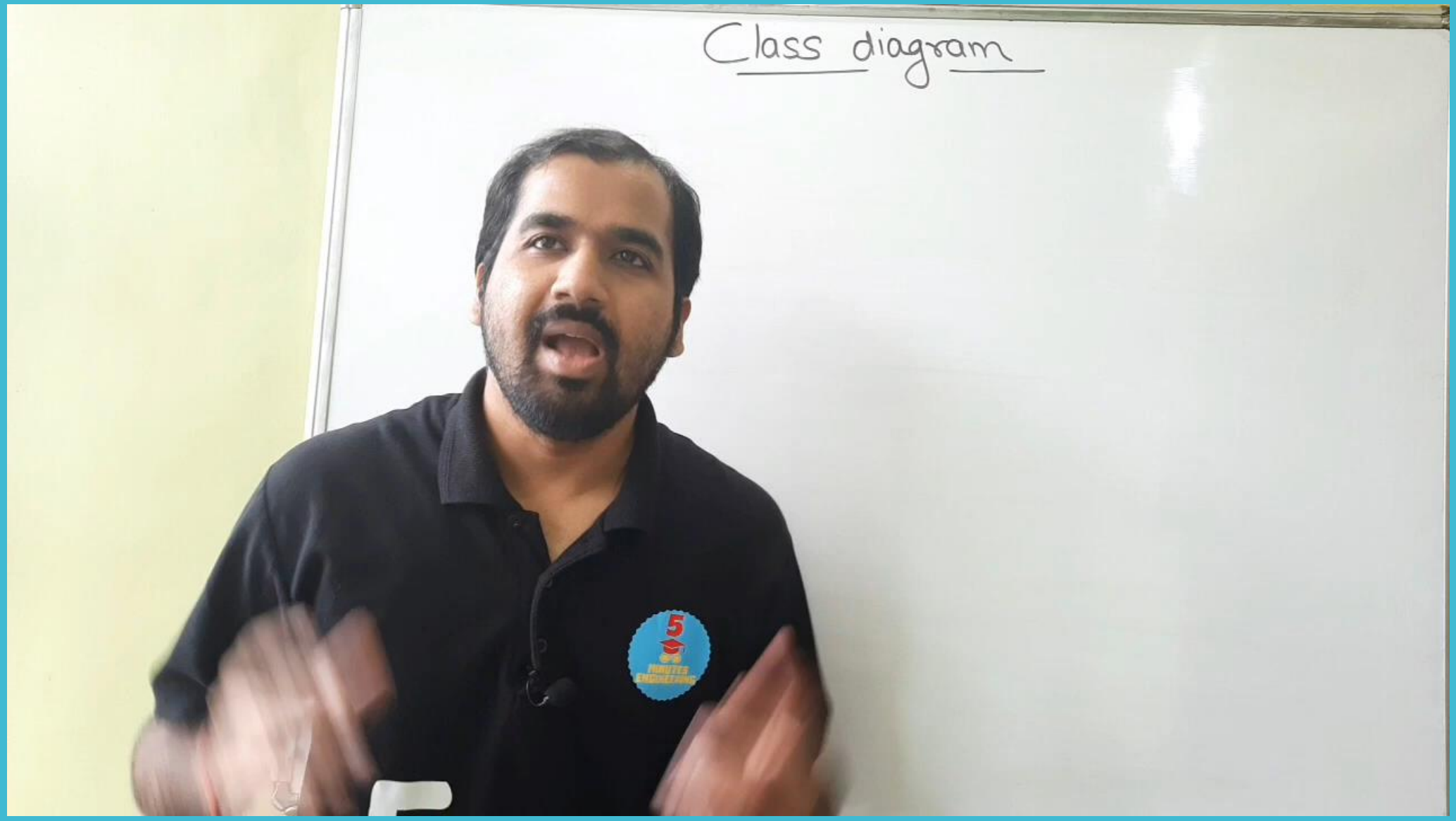
Link and Associations

- A link is a physical or conceptual connection among objects.
- For example, joe smith works for simplex company.
- An Association is a description of a group of links with common structure and common semantics.
- For example, a person WorksFor a company.
- The links of an association connect objects from the same classes.

Link and Associations

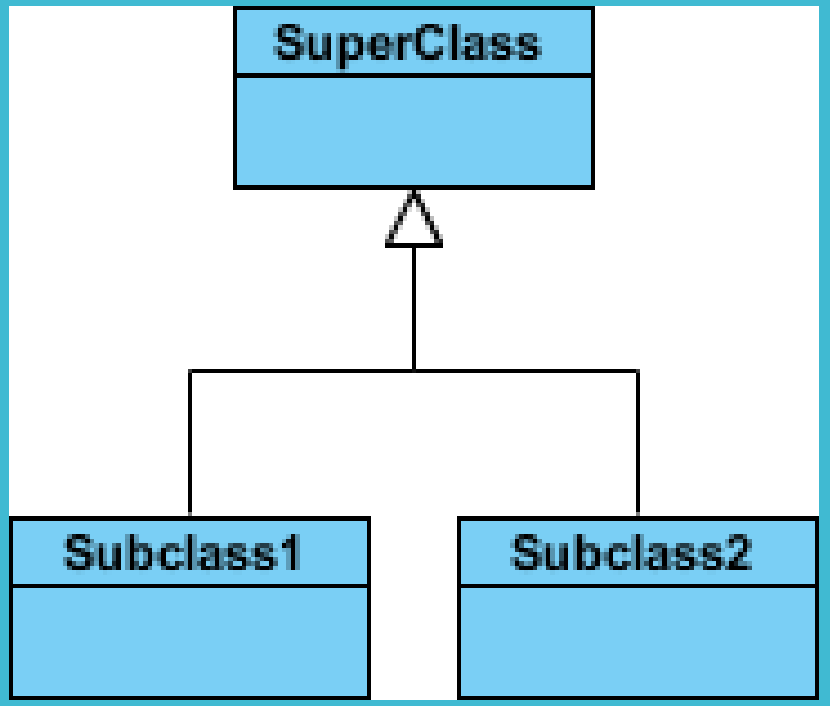


Class Diagram Video

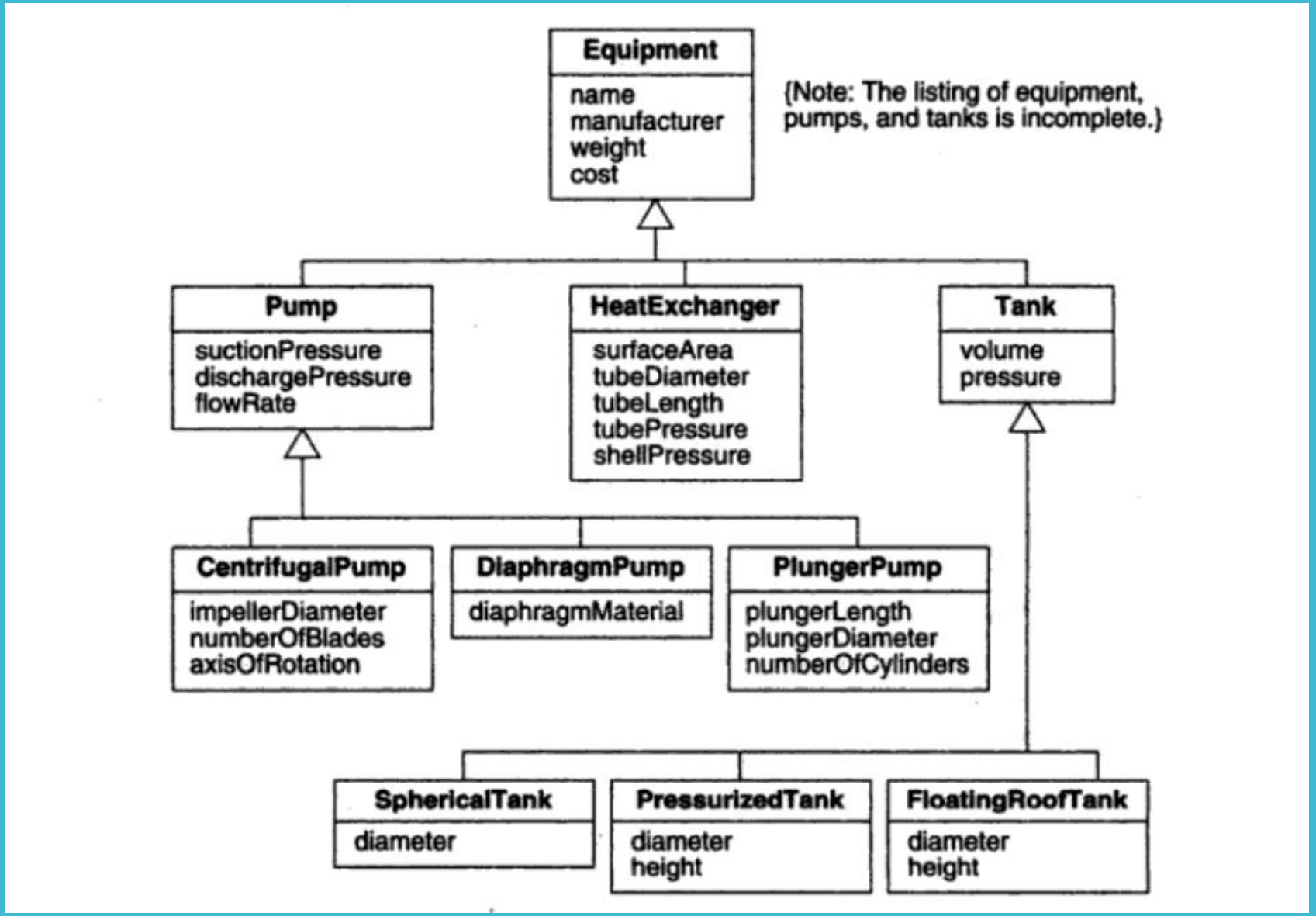


Inheritance (Generalization)

- Represents an "is-a" relationship.
- An abstract class name is shown in italics.
- SubClass1 and SubClass2 are specializations of Super Class.
- A solid line with a hollow arrowhead that point from the child to the parent class

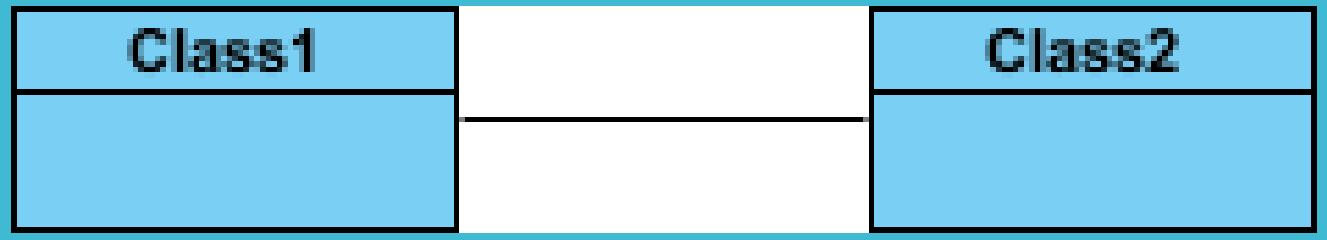


Inheritance Example



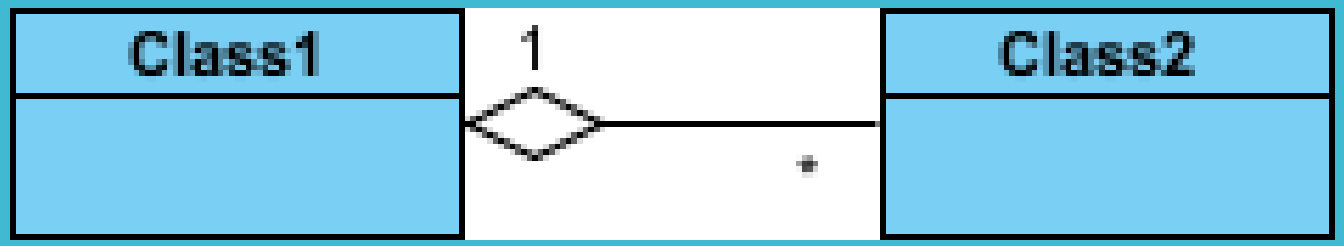
ASSOCIATION

- A structural link between two peer classes.
- There is an association between Class1 and Class2
- A solid line connecting two classes



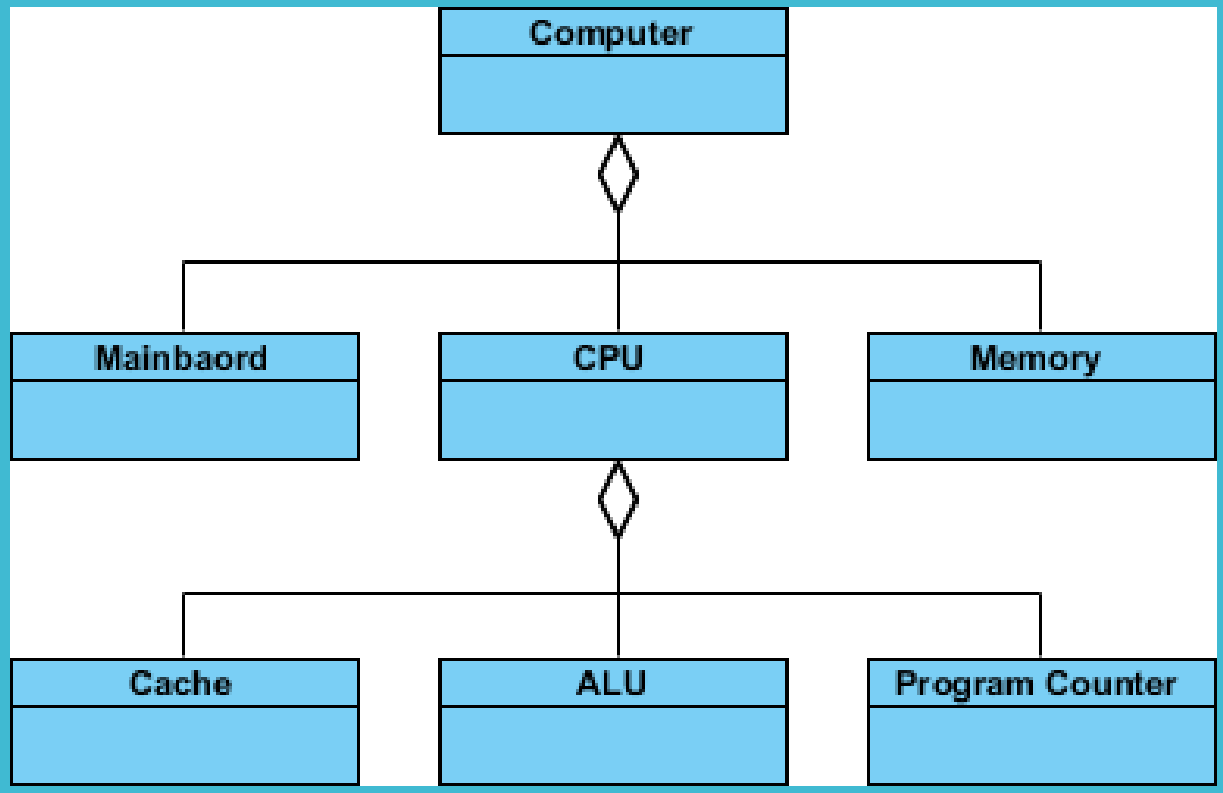
Aggregation

- A special type of association. It represents a "part of" relationship.
- Class2 is part of Class1.
- Many instances (denoted by the *) of Class2 can be associated with Class1.
- Objects of Class1 and Class2 have separate lifetimes.
- A solid line with an unfilled diamond at the association end connected to the class of composite



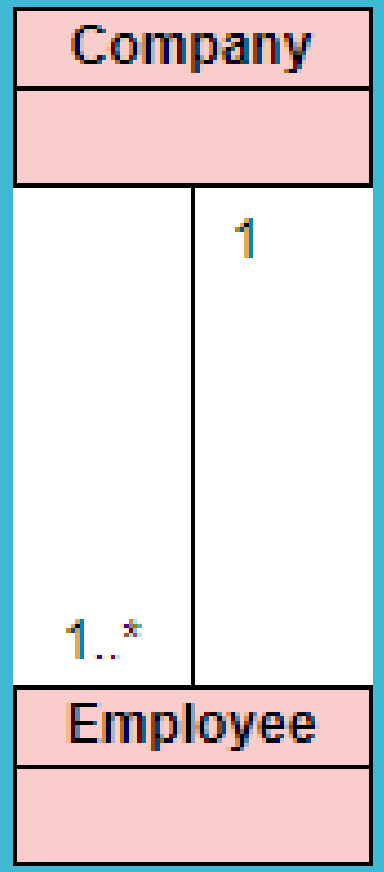
Aggregation Example - Computer and parts

- An aggregation is a special case of association denoting a "consists-of" hierarchy
- The aggregate is the parent class, the components are the children classes

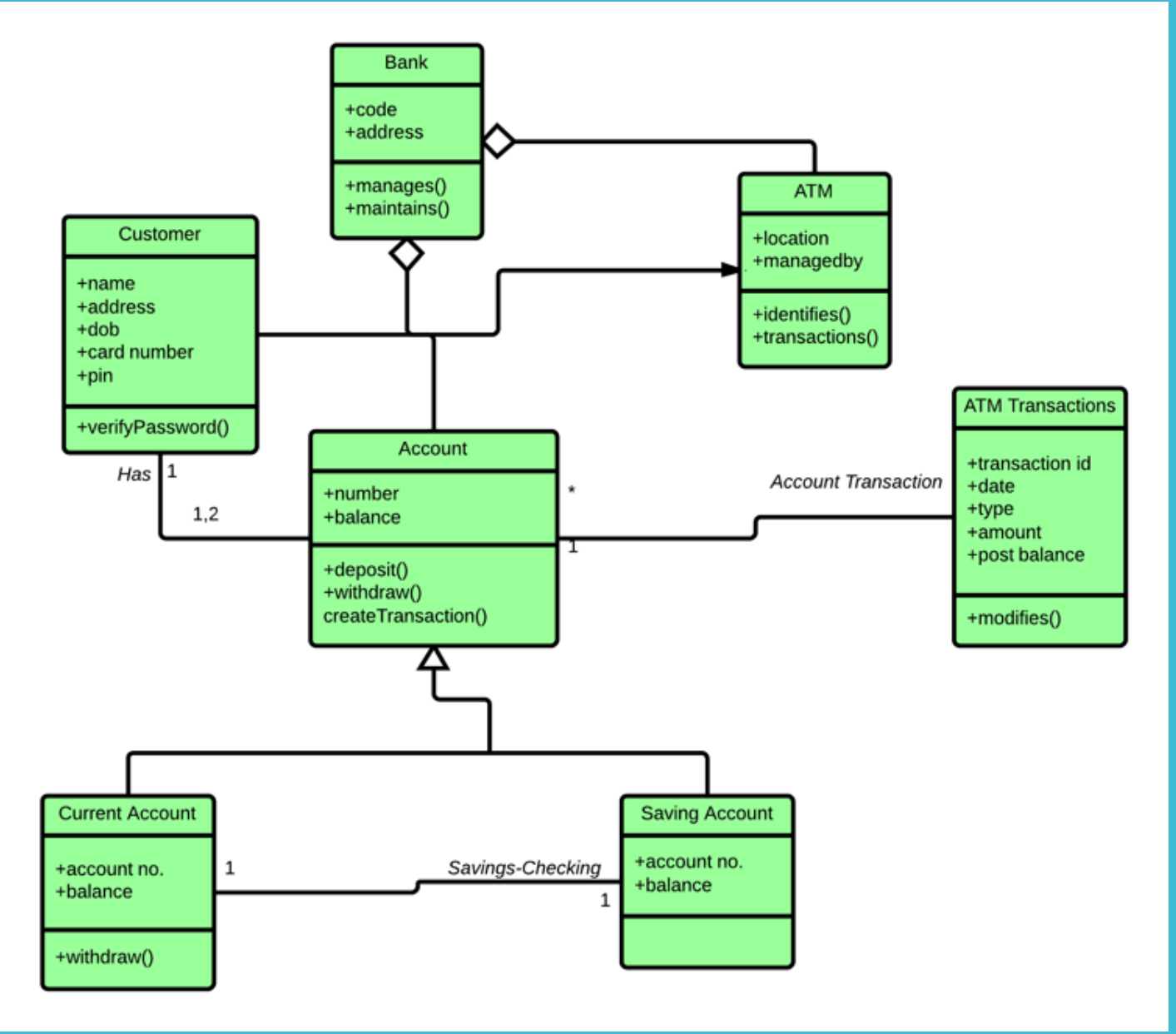


Multiplicity

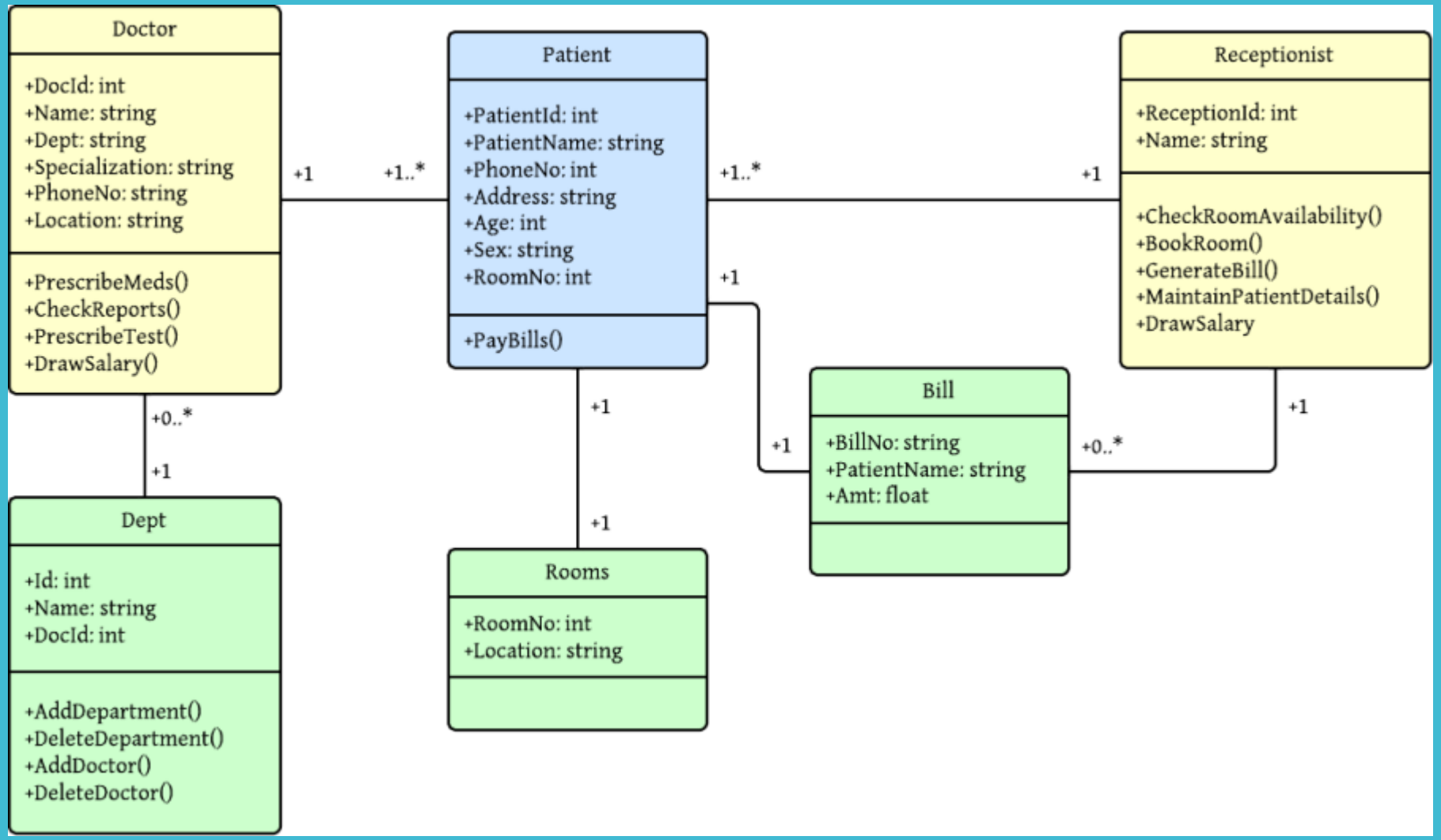
- How many objects of each class take part in the relationships and multiplicity can be expressed as:
- Exactly one - 1
- Zero or one - 0..1
- Many - 0..* or *
- One or more - 1..*
- Exact Number - e.g. 3..4 or 6



Class Diagram of ATM System



Class Diagram of Hospital Management System



The State Model

Event

- An event is an occurrence at a point in time, such as user depresses left button of flight number 123 departs from Ahmedabad.

- There are three different types of Events available

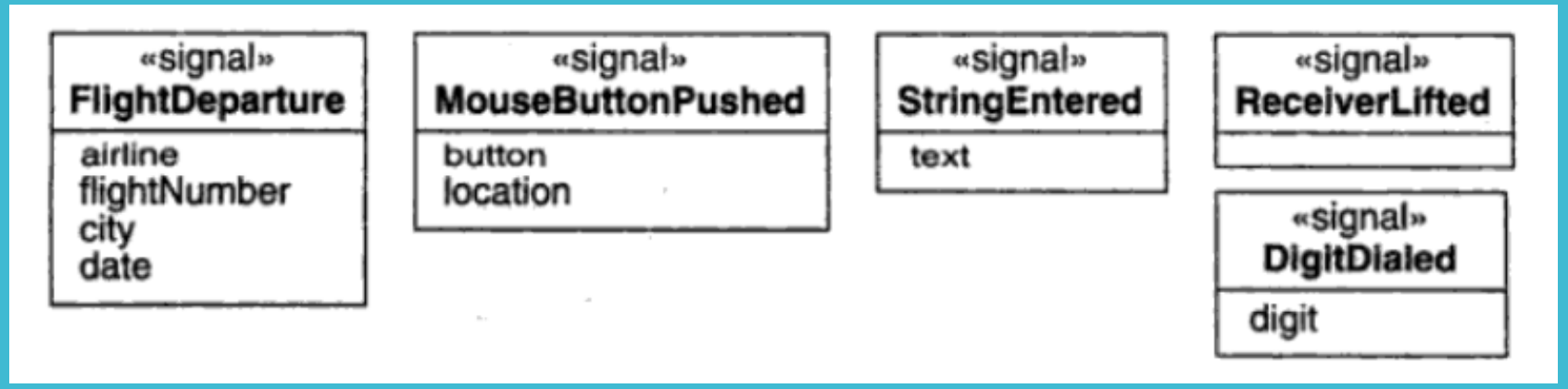
(1) Signal Event

(2) Change Event

(3) Time Event

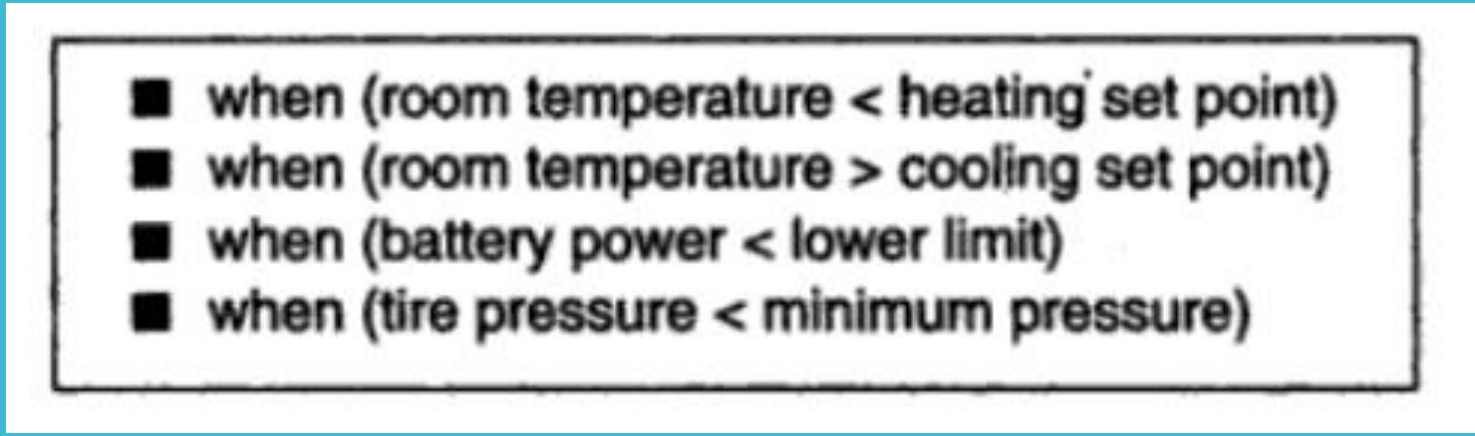
(1) Signal Event

- A signal is an explicit one-way transmission of information from one object to another.
- A signal event is the event of sending or receiving a signal.
- Every signal transmission is a unique occurrence, but we group them into signal classes and give each signal class a name to indicate common structure and behavior.



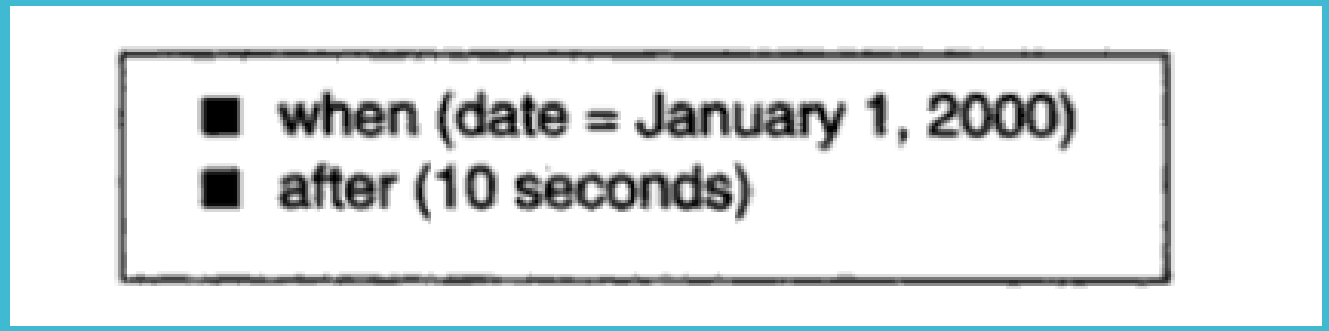
(2) Change Event

- A change event is an event caused by the satisfaction of a Boolean expression.
- The intent of a change event is that the expression is continually tested-whenever the expression changes from false to true, the event happens.



(3) Time Event

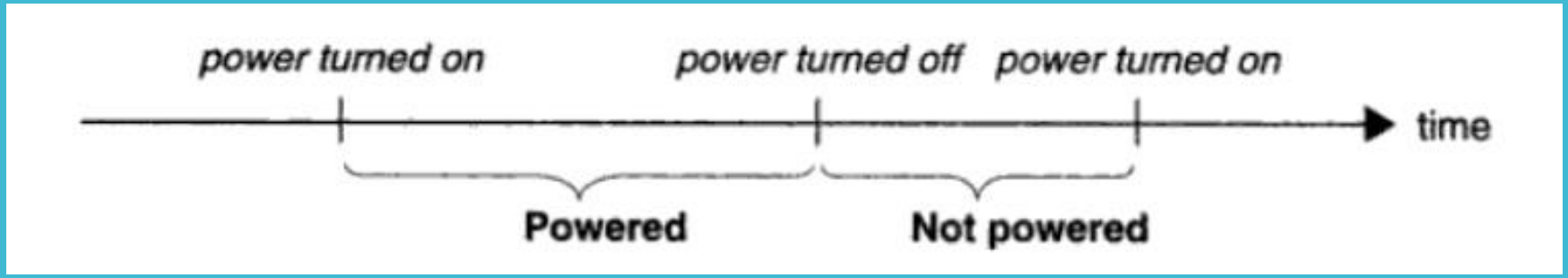
- A time event is an event caused by the occurrence of an absolute time or the elapse of time interval.
- The UML notation for an absolute time is the keyword when followed by a parenthesized expression involving time.



State

- A state is an abstraction of the value and links of an object.
- UML notation for a state is --- a rounded box containing an optional state name.

Event Vs. State



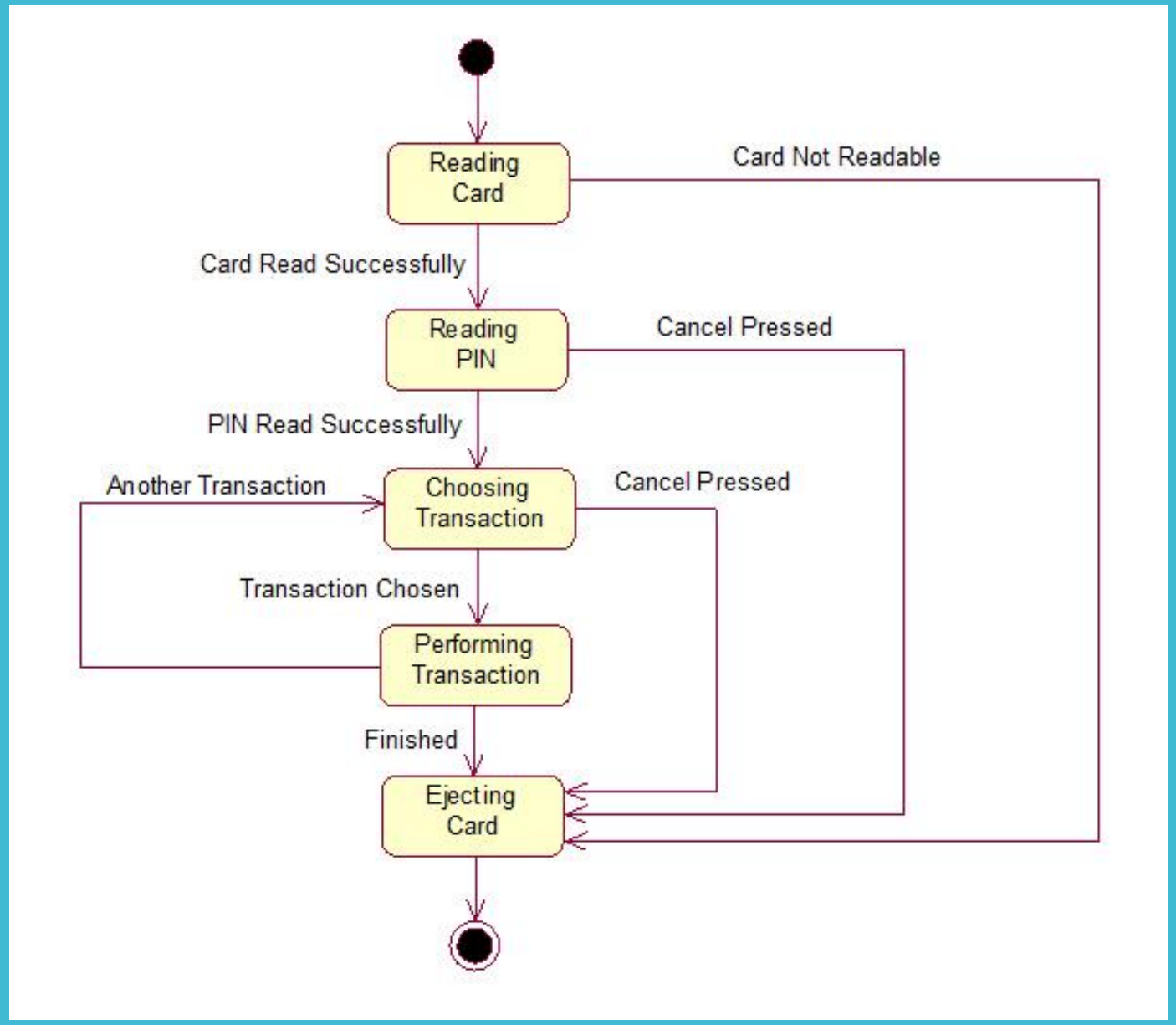
TRANSITION

- A transition is an instantaneous change from one state to another.
- For Example, when a called phone is answered, the phone line transitions from *Ringing* to the *Connected* state.

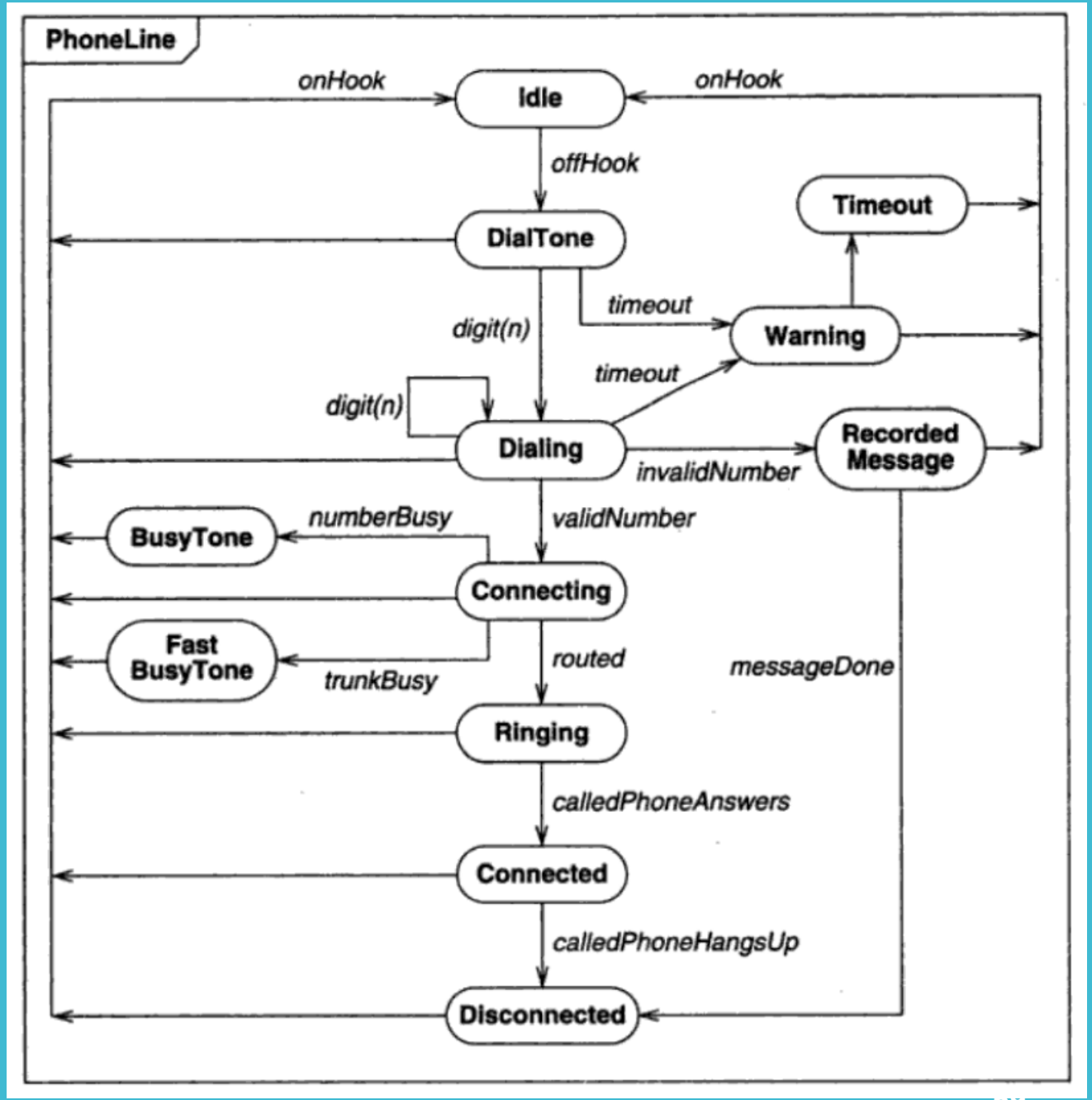
State Diagram

- A state diagram is a graph whose nodes are states and whose directed arcs are transitions between states.
- A state diagram specifies the state sequences caused by an event sequences.
- State names must be unique within the scope of a state diagram.

State Diagram Example



State Diagram Example



Thank You!!!!