

Software Re-Engineering

Lecture: 06



Dr. Syed Muazzam Ali Shah

Department of Software Engineering

National University of Computer &
Emerging Sciences

muazzam.ali@nu.edu.pk

Sequence [Today's Agenda]

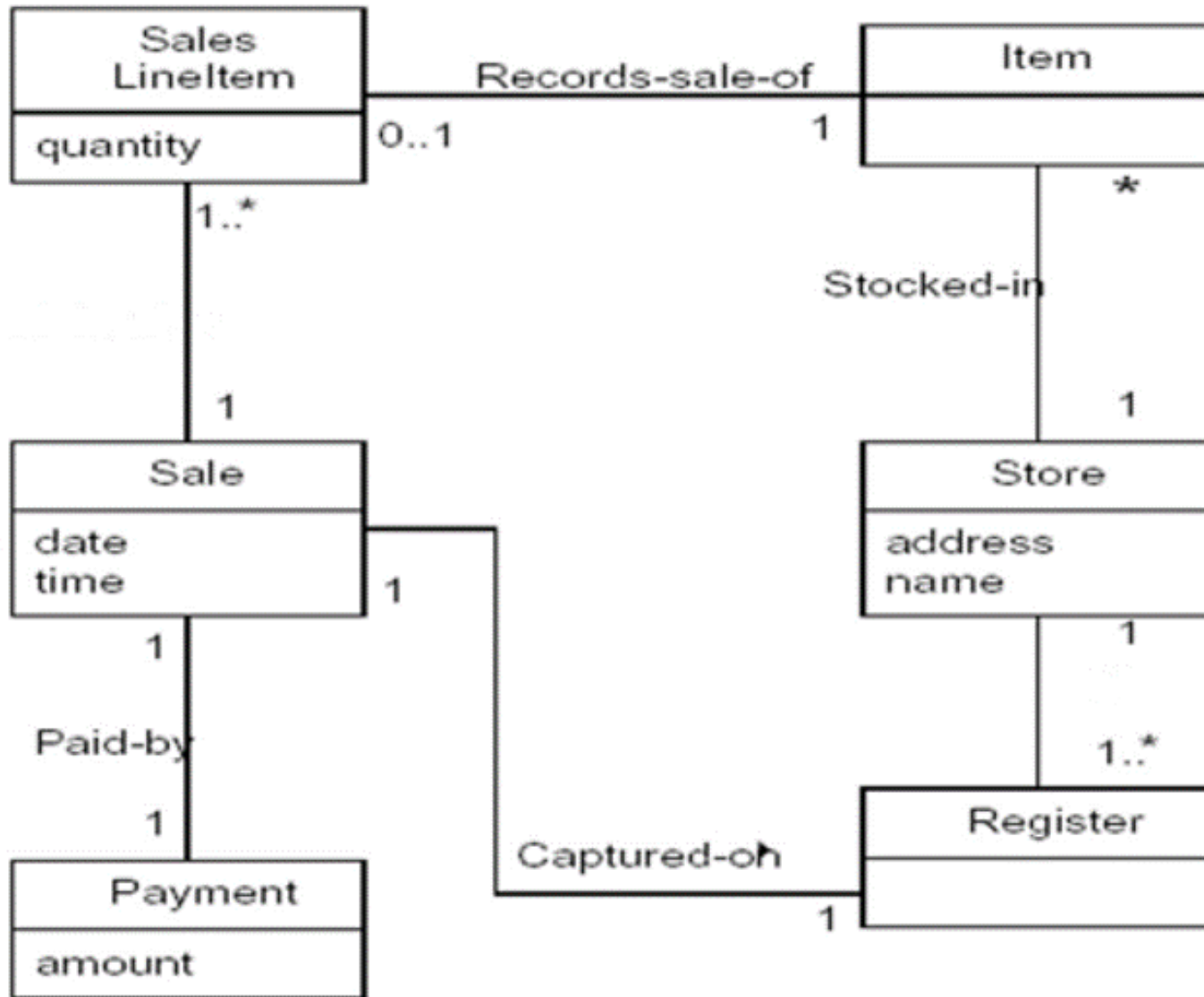
Content of Lecture

- Domain Model
 - How to develop Domain Model
 - Features of Domain Model
 - Example of Domain Model
 - Case Study: Local Hospital Problem
- System Sequence Diagram
 - SSD vs SD
 - Examples
 - Case Study

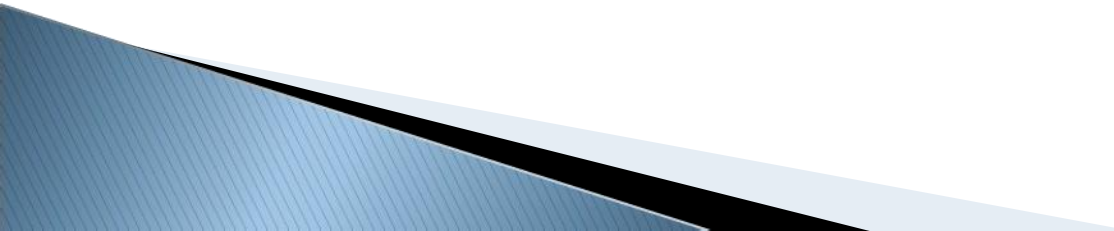
Domain Model

- | Structural model of basic domain concepts and their relationships
- | It may show:
 - | domain objects or conceptual classes
 - | associations between conceptual classes
 - | attributes
- | Also called conceptual models, domain object models, and analysis object models.

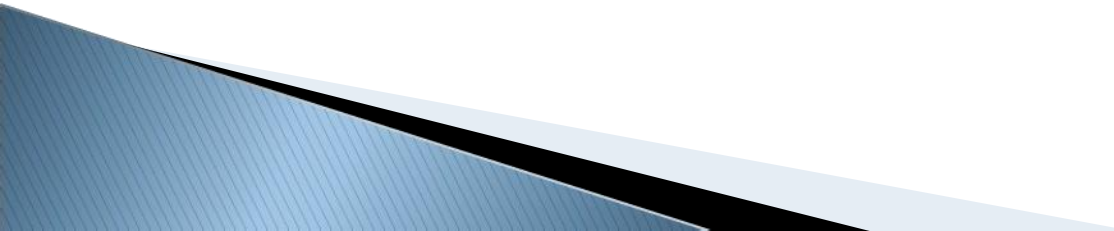
Example



How To Develop Domain Model

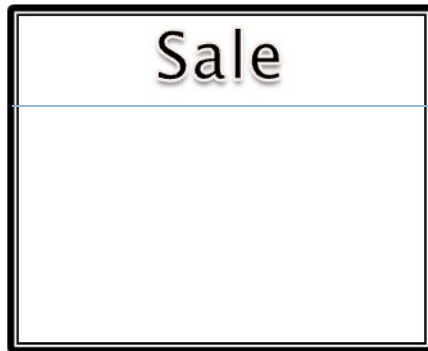
- 1 Identify conceptual classes
 - 2 Draw them as in a UML domain model
 - 3 Add associations necessary to record relationship
 - 4 Add the attributes necessary to fulfill the information requirements
- 

Features Of A Domain Model

- Domain classes
 - Attributes
 - Associations
 - Multiplicity
 - Aggregation
 - Composition
 - Generalization
 - Roles
- 

Domain class

- | Each domain class denotes a type of object

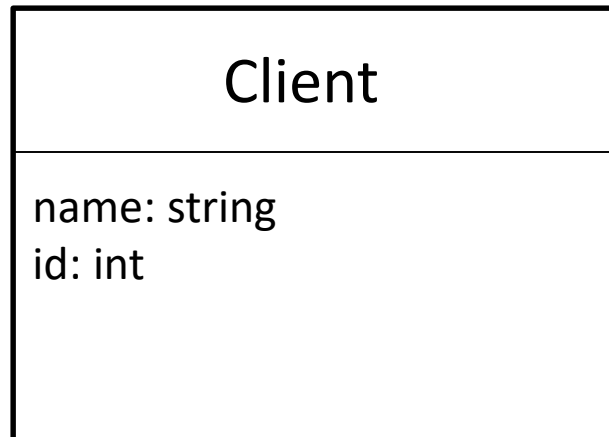


Attributes

- Attributes refer to properties that define the class.

For Example

A class Client will have attributes name and id.



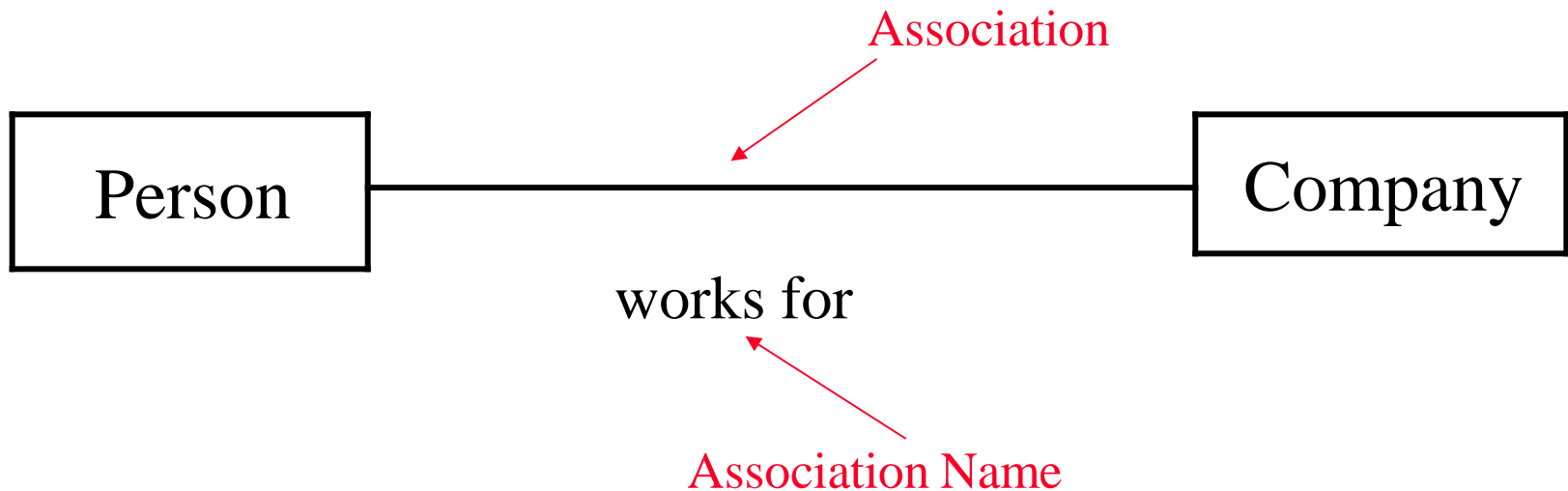
Class Visibility

}	public level	+
}	protected level	#
}	private level	-

Circle	
-	centreX:Int
+	centreY:Int=0

Association

- | A link between two classes
- A **Person** works for a **Company**.



Multiplicity

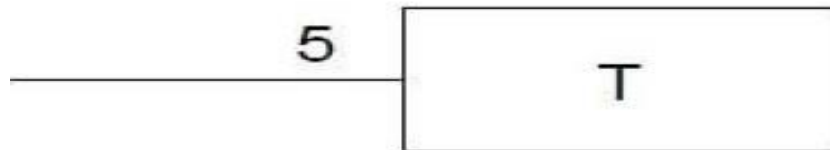
- Describes how many instances of one concept can be associated with one instance of the related concept.



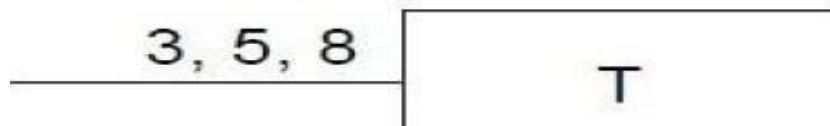
one or more



one to 40



exactly 5



exactly 3, 5, or 8

Multiplicity

- | A **Student** can take up to **five** **Courses**.
Student has to be enrolled in at least **one** course.
- | **Up to 300** students can enroll in a course.
- | A class should have at least 10 students.

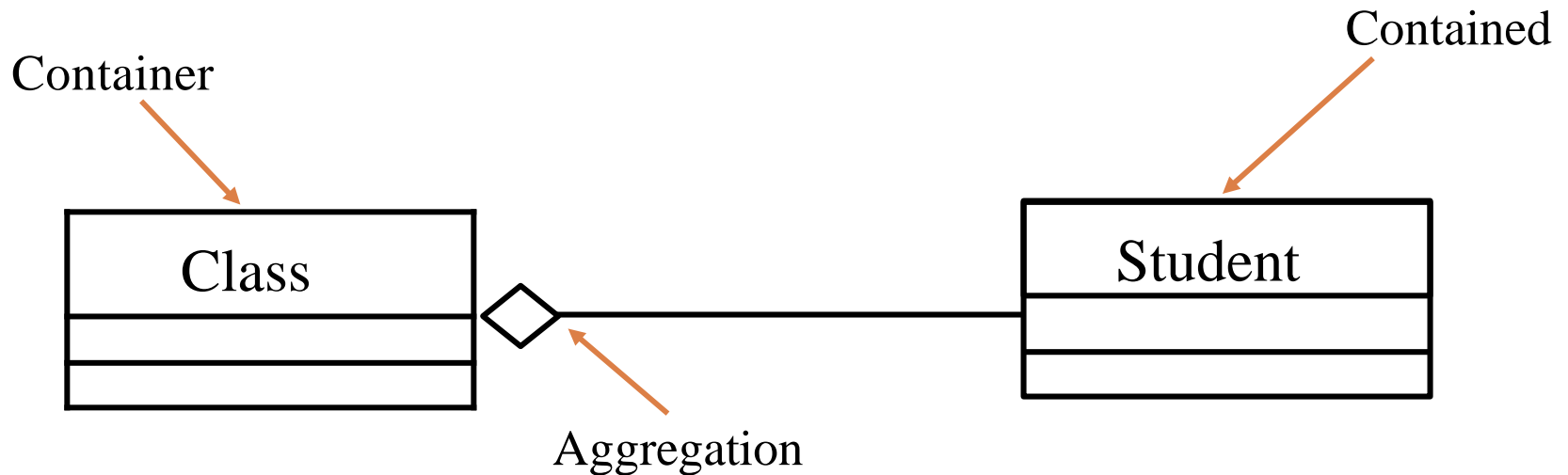


Aggregation

└ “Has a”

A special form of association that models a Container-contained relationship

└ Weak association



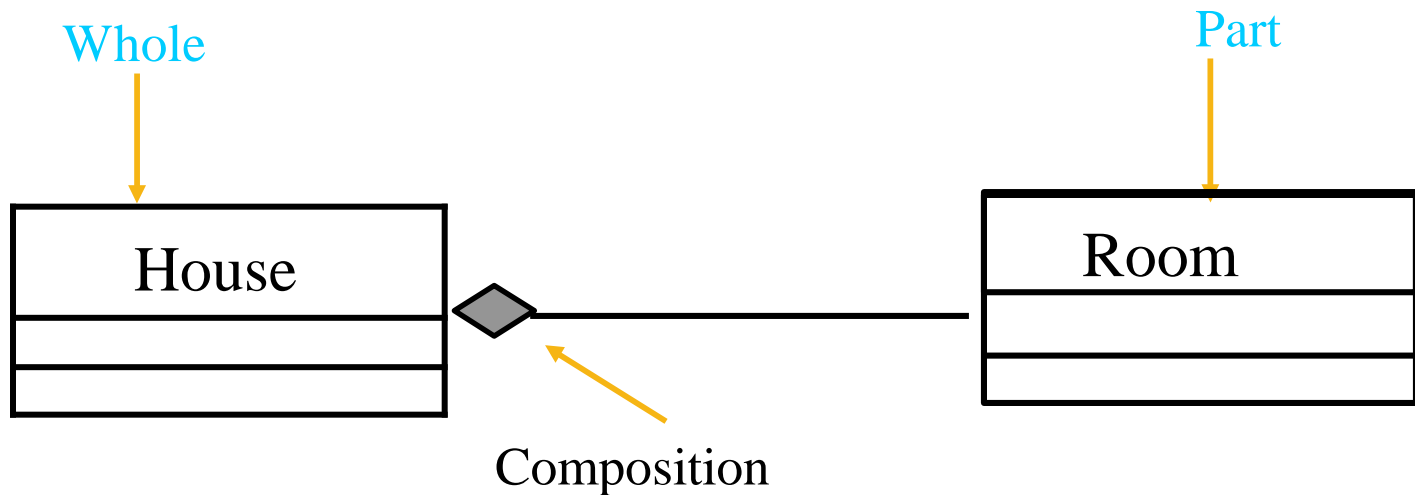
Composition

‣ “Own a”

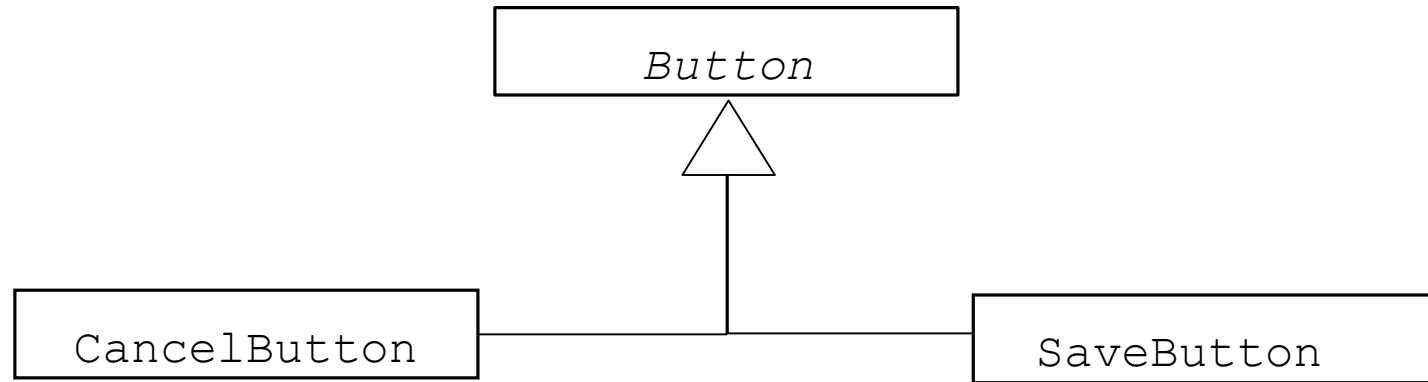
A special form of association that models a Whole-part relationship

A strong form of aggregation where components cannot exist without the aggregate.

‣ The parts cannot survive without the whole/aggregate



Inheritance



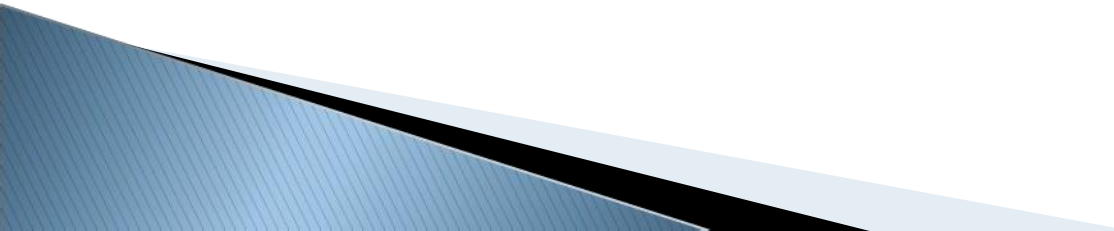
- 1 The **children classes** inherit the attributes of the **parent class**.
- 1 Eliminating redundancy.
- 1 Generalize/specialize

Role

- | Each end of an association is called a role.
- | Roles may have:
 - ♣ Name
 - ♣ multiplicity expression
 - ♣ Navigability

Case Study: Local Hospital Problem

*A Local Hospital consists of many wards, each of which is assigned many patients. Each **patient** is assigned to **one doctor**, who has overall responsibility for the patients in his or her care. Each **patient** is **prescribed drugs** by the **doctor** responsible for that patient. Each **nurse** is assigned to a **ward** and nurses **all the patients** in the **ward**. Each **patient** is assigned **one nurse** in this position of responsibility. Prospective **ward** is look-after by the prospective **doctors**.*



Case Study: Local Hospital Problem

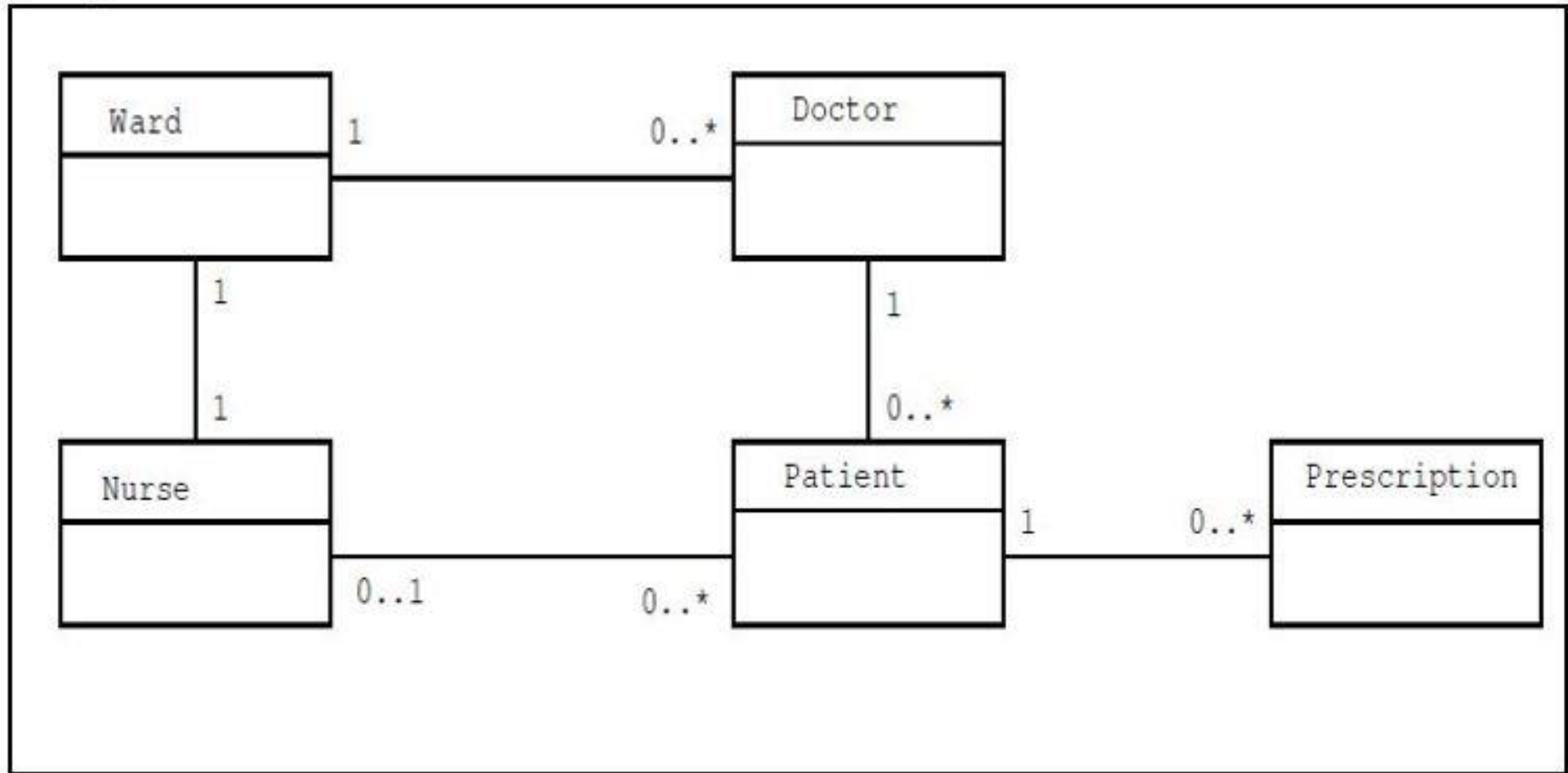


Diagram: Local Hospital Problem LHP

SSD vs. Sequence Diagram



- # A **System Sequence Diagram (SSD)** is an artifact that illustrates input and output events related to the system under discussion
- # System Sequence Diagrams are typically associated with use-case realization in the logical view of system development
- # **Sequence Diagrams** (*Not System Sequence Diagrams*) display object interactions arranged in time sequence

✿ The UML does not define something called a "system" sequence diagram but simply a "sequence diagram".

SSD - System Behavior

- # *System behavior* is a description of what a system does, without explaining how it does it

One part of that description is a System Sequence Diagram

- # System behaves as “**Black Box**”
- # Interior objects are not shown, as they would be on a Sequence Diagram

: System

System Sequence Diagrams



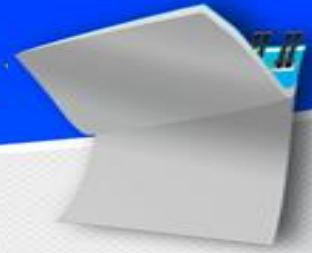
Use cases describe:

- How actors interact with the software system,
- Typical course of events that external actors generate, and
- The order of the events

When an actor interacts with the system, he generates **system events**, usually requesting some **system operation** to handle the event

- For example, when a cashier enters an item's ID, the cashier is requesting the POS system to record that item's sale (the enterItem event). That event initiates an operation upon the system. The use case text implies the enterItem event, and the SSD makes it concrete and explicit

Notation: Object

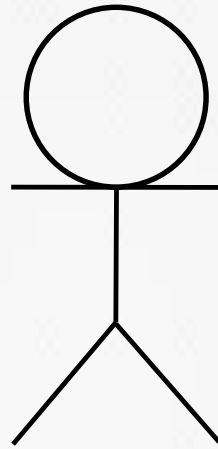


- # Objects are instances of classes
- # Object is represented as a rectangle which contains the name of the object underlined

:Object1

Notation: Actor

- An **Actor** is modeled using the ubiquitous symbol, the stick figure



Notation: Lifeline



- # The **Lifeline** identifies the existence of the object over time
- # The notation for a Lifeline is a vertical dotted line extending from an object



Notation: Message

- # **Messages**, modeled as horizontal arrows between activations, indicate the communications between objects

messageName (argument) →

Example of an SSD



system as black box

the name could be "NextGenPOS" but "System" keeps it simple

the ":" and underline imply an instance, and are explained in a later chapter on sequence diagram notation in the UML

external actor to system

: Cashier

Process Sale Scenario

:System

makeNewSale

a UML loop interaction frame, with a boolean guard expression

loop

[more items]

enterItem(itemID, quantity)

description, total

return value(s) associated with the previous message

an abstraction that ignores presentation and medium

the return line is optional if nothing is returned

endSale

total with taxes

makePayment(amount)

change due, receipt

a message with parameters

it is an abstraction representing the system event of entering the payment data by some mechanism

Relationship between UCs and SSDs



- # An SSD shows system events for *one scenario of a use case*, therefore it is generated from inspection of a use case
 1. Draw a line representing the system as a black box.
 2. Identify each actor that directly operates on the system. Draw a line for each such actor.
 3. From the use case, typical course of events text, identify the system (external) events that each actor generates. They will correspond to an entry in the right hand side of the typical use case. Illustrate them on the diagram.
 4. Optionally, include the use case text to the left of the diagram.

Example



: Cashier

Process Sale Scenario

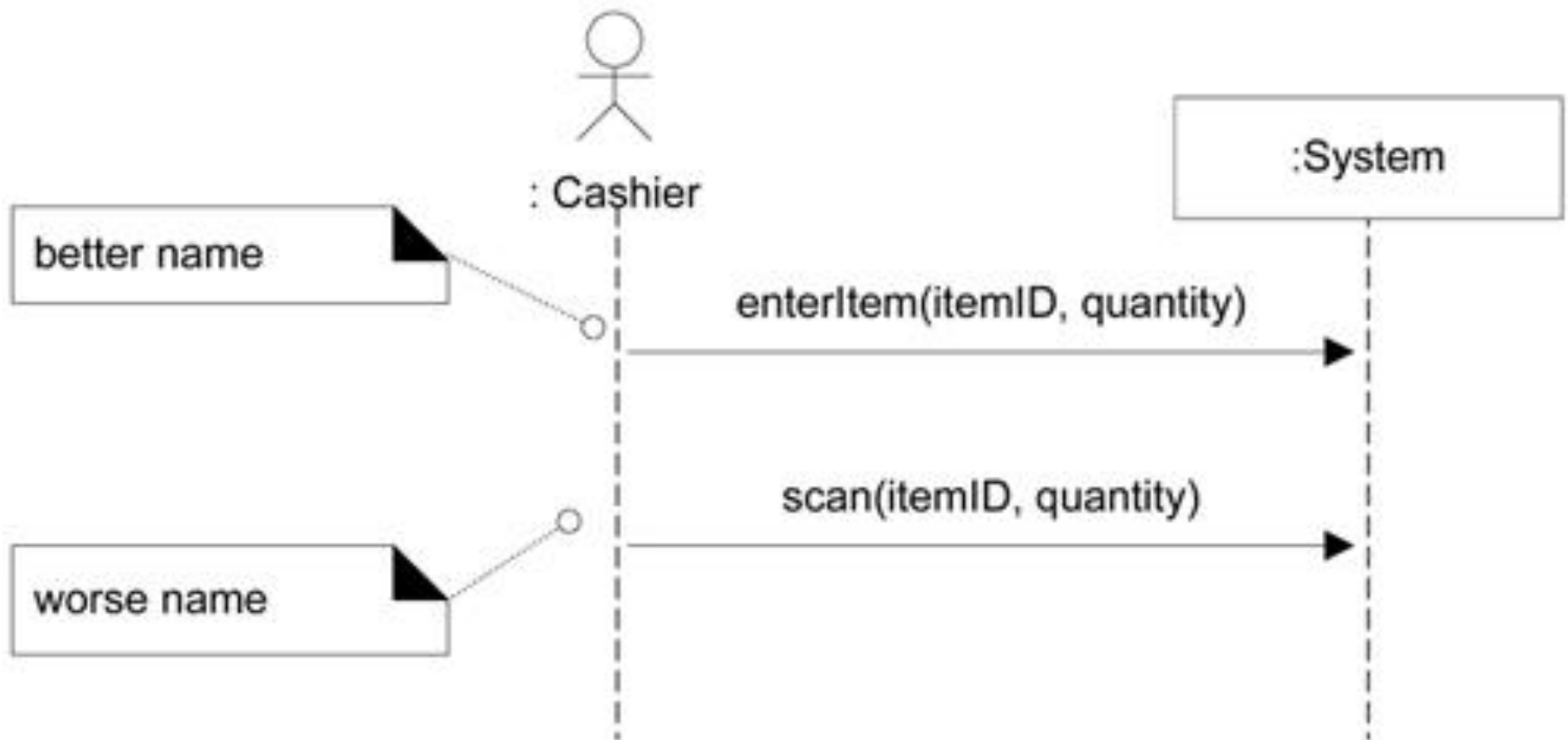
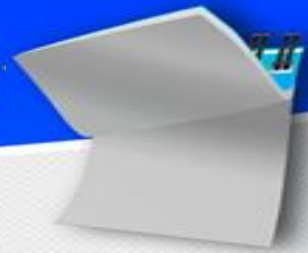
:System



Simple cash-only Process Sale scenario:

1. Customer arrives at a POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total.
Cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
- ...

How to Name ???



Case Study: Cafeteria Management System



Basic Flow:

- # Customer (Actor) request for registration
- # After completion of successful registration, customer gets login to the system
- # After successful login to the system, customer checks the list of meal.
- # After checking the list of meal, customer place order.
- # After confirmation of order, customer makes the payment formality.
- # After payment confirmation, customer gets logout from the system.

Thank You!

