605.601 Foundations of Software Engineering
Fall 2020

Module 04: Object Orientation and UML

Dr. Tushar K. Hazra

tkhazra@gmail.com

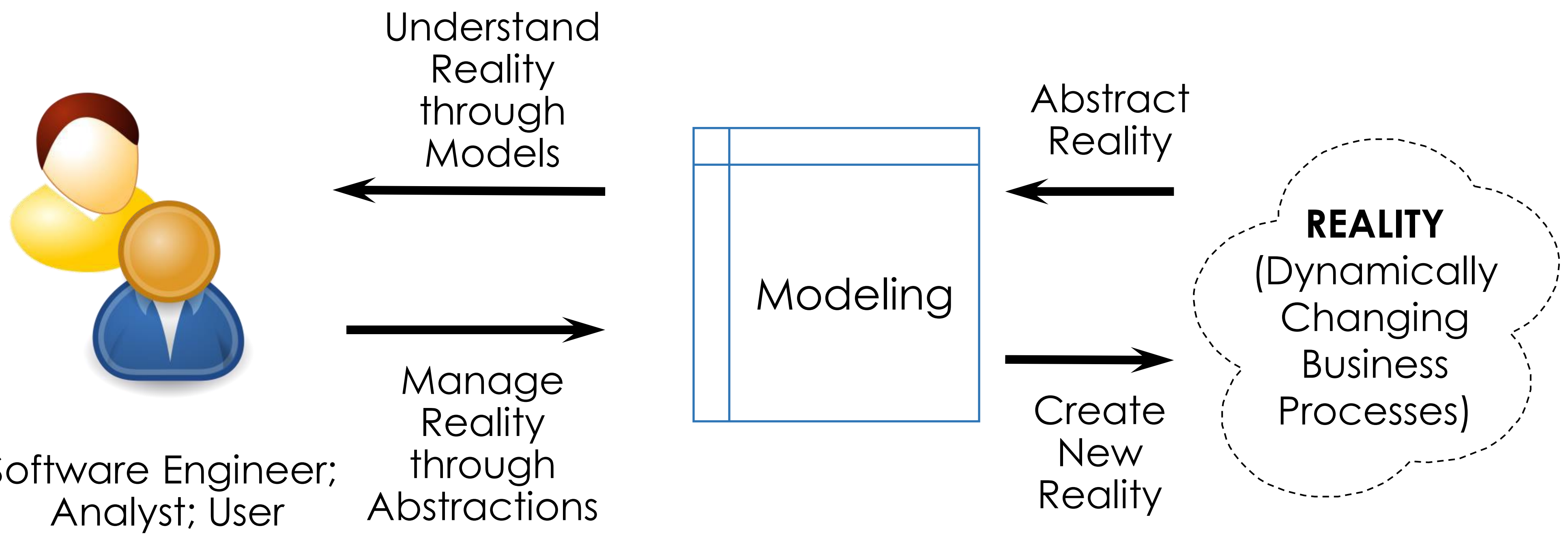
(443)540-2230

605.601 Foundations of Software Engineering
Course Module 04: Object Orientation and UML

- Modeling
- Object Orientation
- Evolution of Modeling
- Unified Modeling Language (UML)
- UML Models and Diagrams
- Using UML Models and Diagrams

Modeling

- Importance of Modeling in Software Engineering


Understand Reality through Models
Software Engineer; Analyst; User
Manage Reality through Abstractions
Modeling
Abstract Reality
Create New Reality
REALITY (Dynamically Changing Business Processes)

Object Orientation

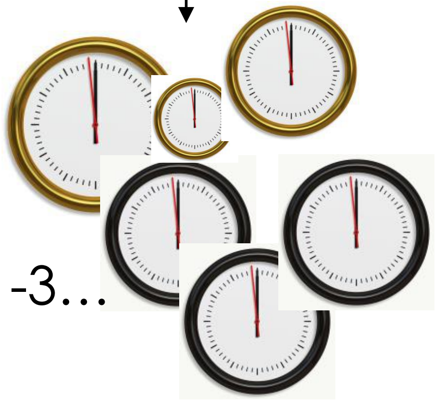
- Program, Class, Object and Data

Program

Data

Class: Clock

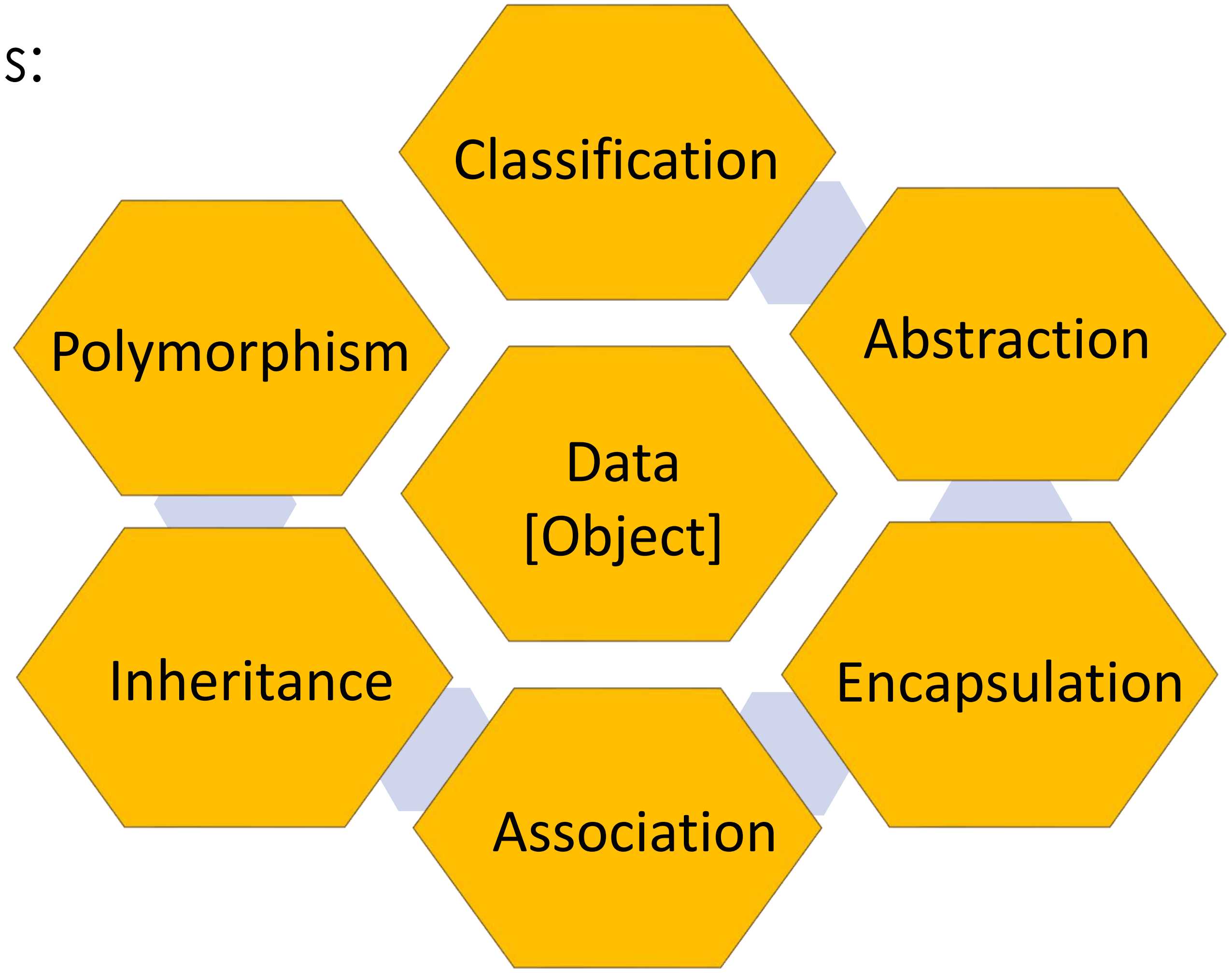
objects: clock-1, -2, -3…



Object Orientation

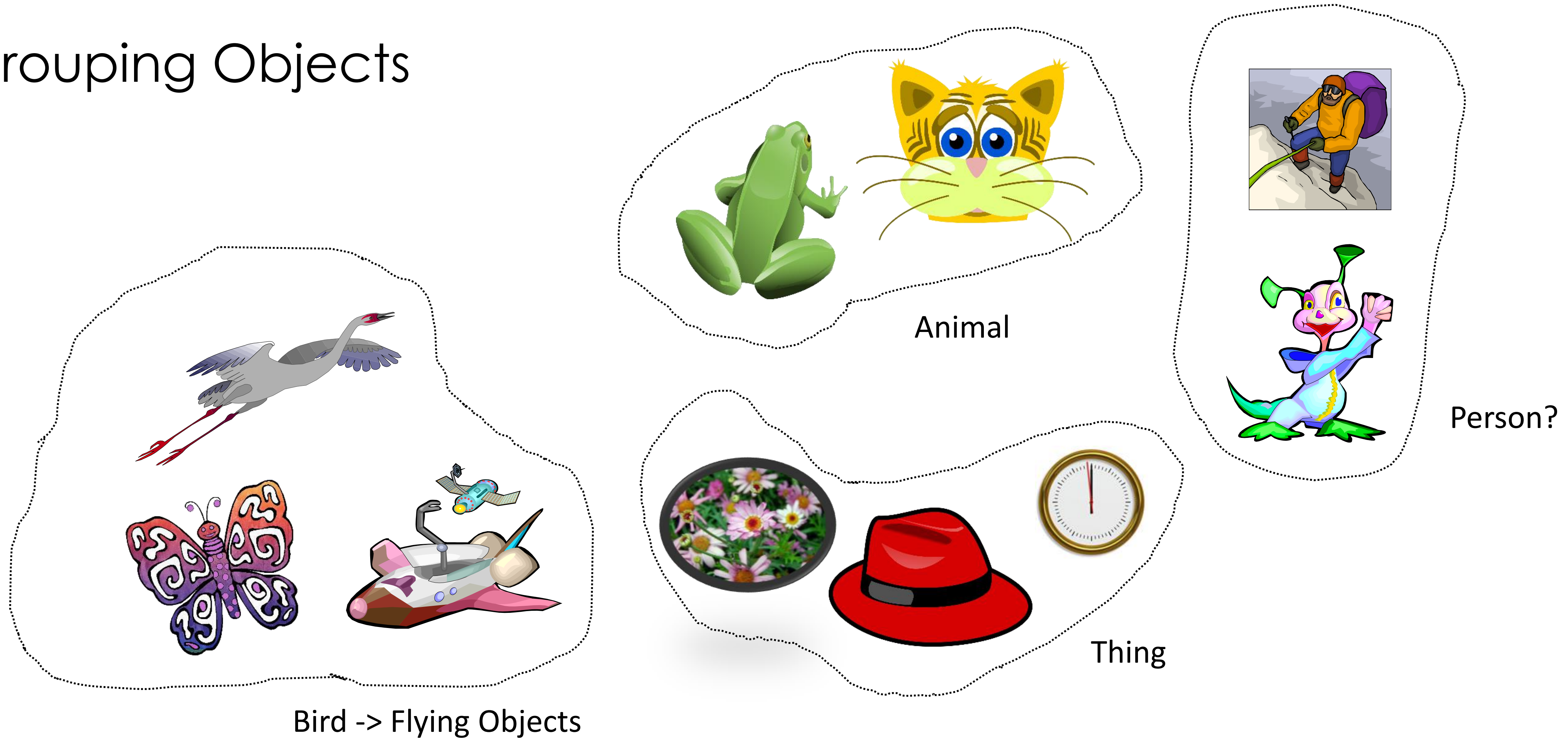
Object Orientation Fundamentals:

- Classification (grouping)
- Abstraction (representing)
- Encapsulation (modularizing)
- Association (relating)
- Inheritance (generalizing)
- Polymorphism (executing)



Classification

- Grouping Objects



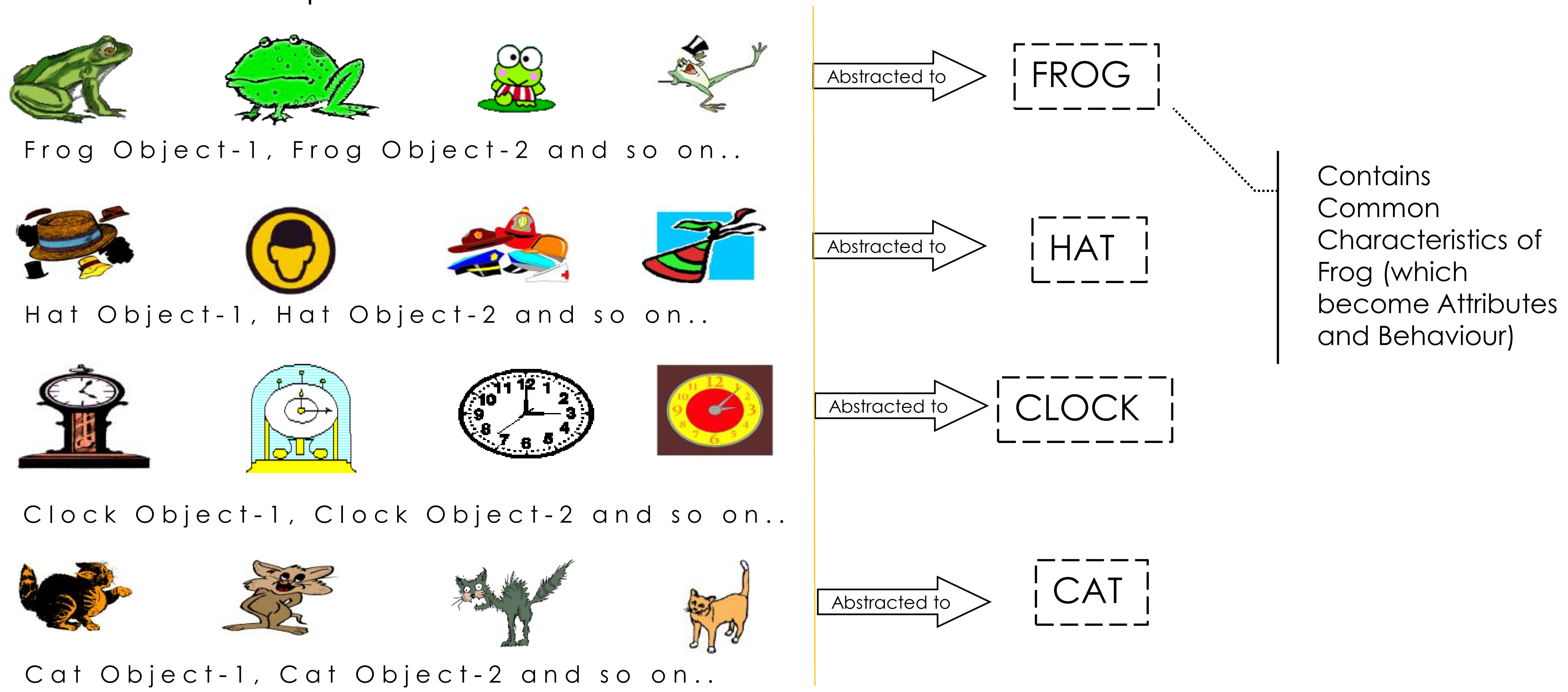
Bird -> Flying Objects

Abstraction

- 1st Level is from Objects to Class

These ones below are real Objects with multiple instances. Each Object has a unique identifier.

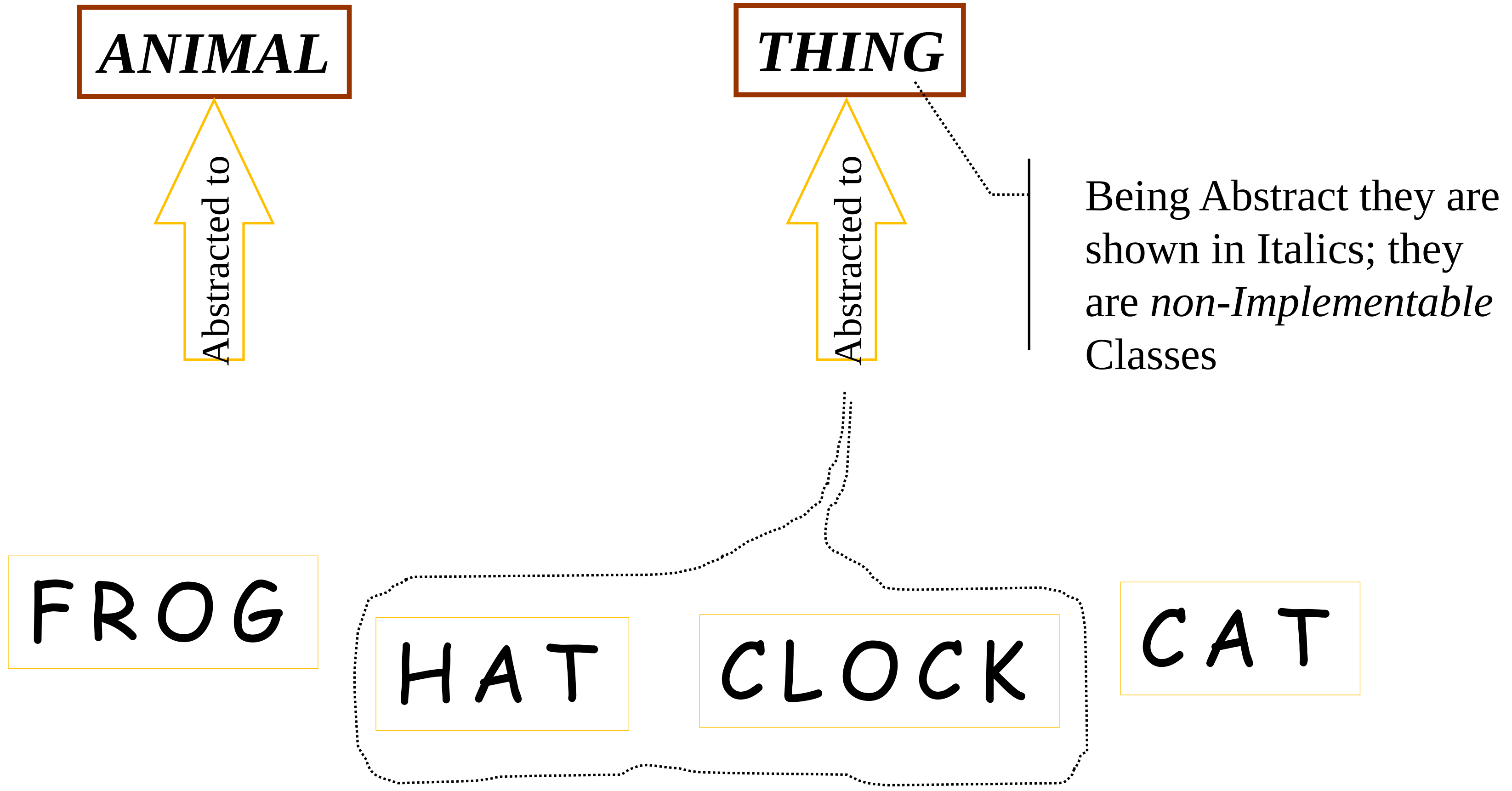
These Names with Boxes around them are *ABSTRACTIONS*. They form the basis for Classes.


Frog Object-1, Frog Object-2 and so on..
Abstracted to
FROG
Hat Object-1, Hat Object-2 and so on..
Abstracted to
HAT
Clock Object-1, Clock Object-2 and so on..
Abstracted to
CLOCK
Cat Object-1, Cat Object-2 and so on..
Abstracted to
CAT
Contains Common Characteristics of Frog (which become Attributes and Behaviour)

Good Classification leads to creation of good Abstractions.

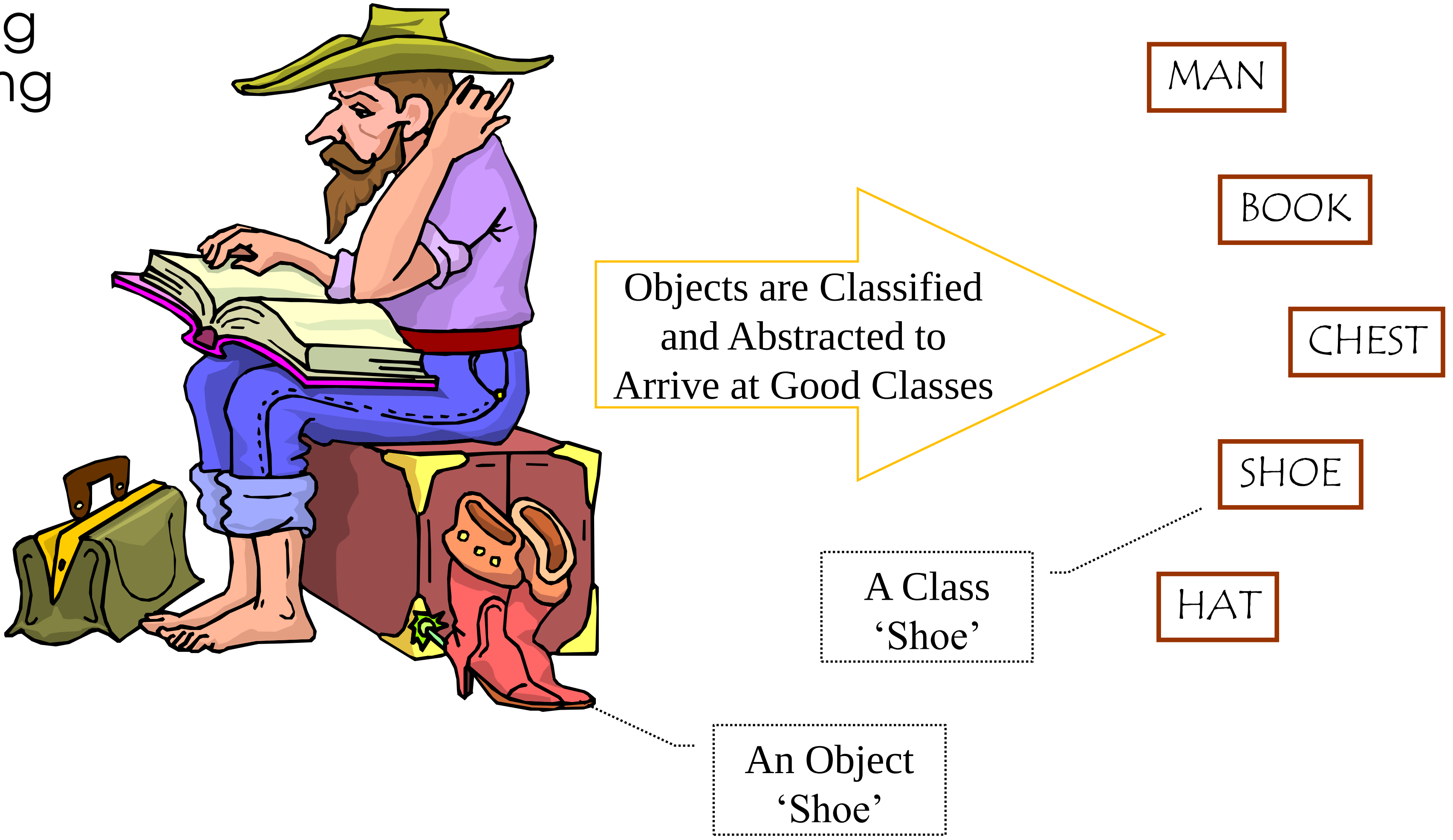
Abstraction

- 2nd Level is from Classes to Classes


ANIMAL
THING
Abstracted to
Abstracted to
Being Abstract they are shown in Italics; they are *non-Implementable* Classes
FROG
HAT
CLOCK
CAT

Classification and Abstraction

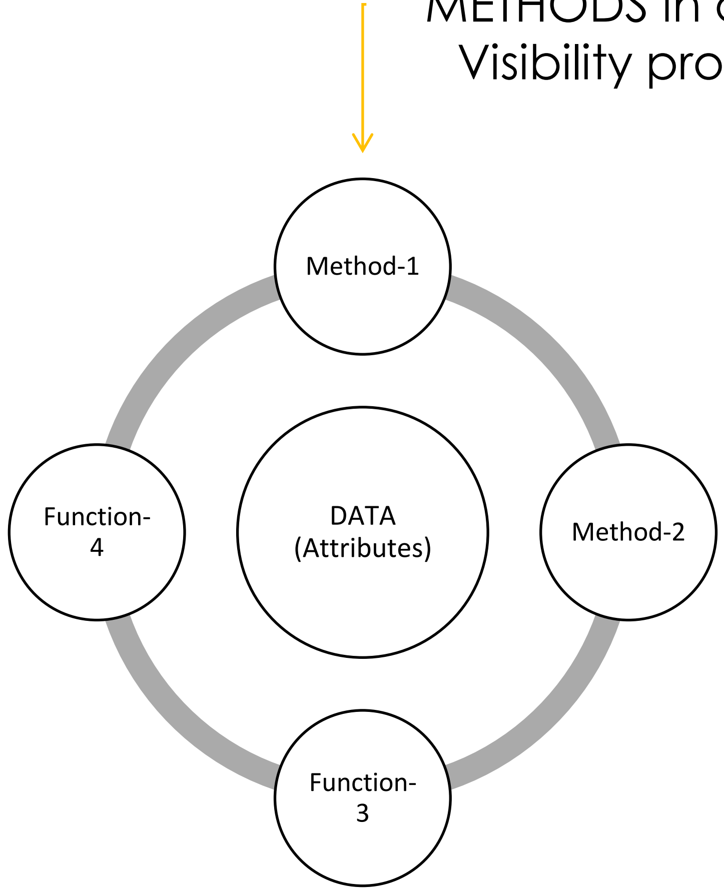
- Representing and grouping objects to classes

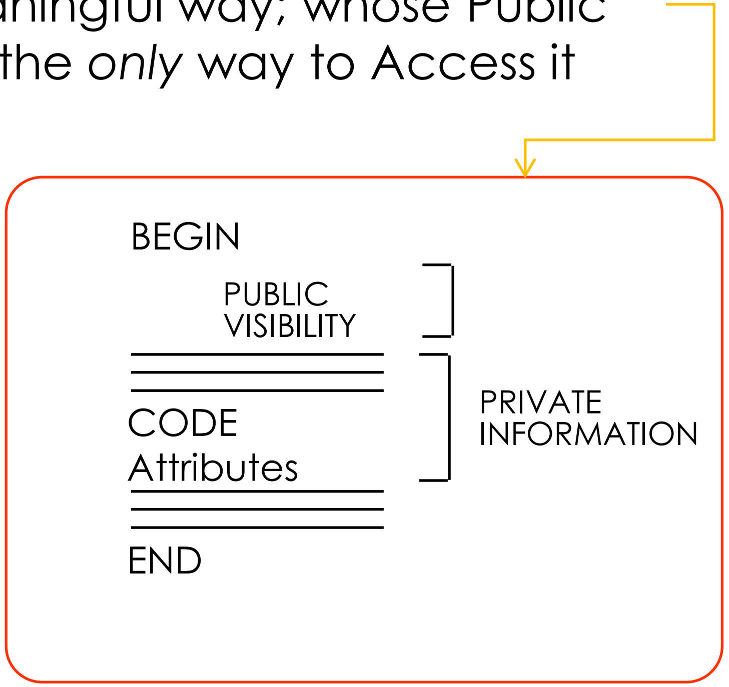

Objects are Classified and Abstracted to Arrive at Good Classes
MAN
BOOK
CHEST
SHOE
HAT
A Class 'Shoe'
An Object 'Shoe'

Encapsulation

- Modularizing the objects

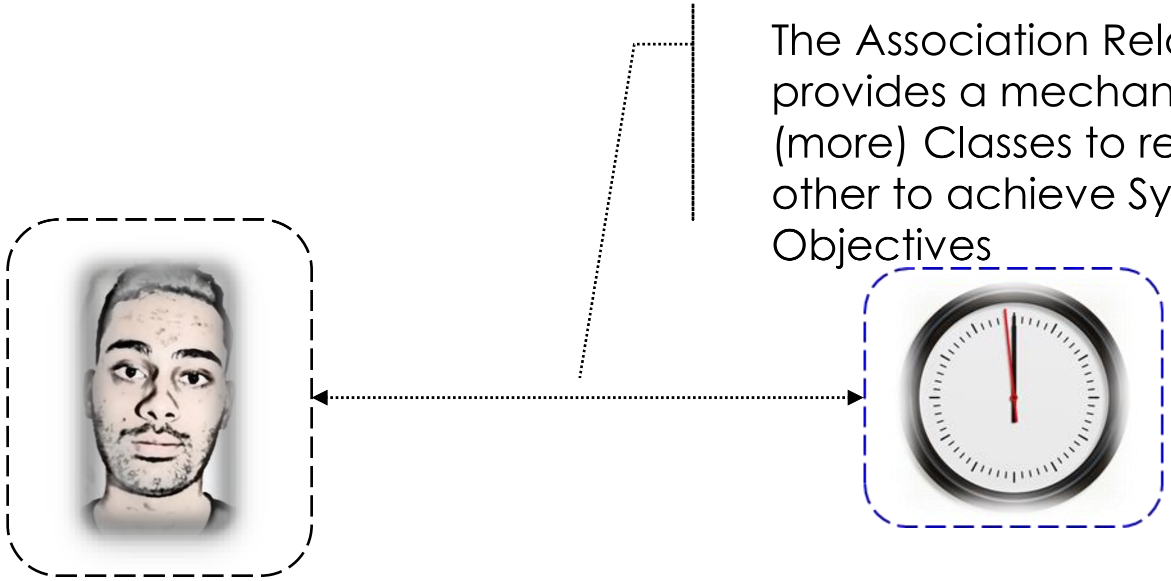
Encapsulation means DATA is 'wrapped' with METHODS in a Meaningful way; whose Public Visibility provides the *only* way to Access it





Association

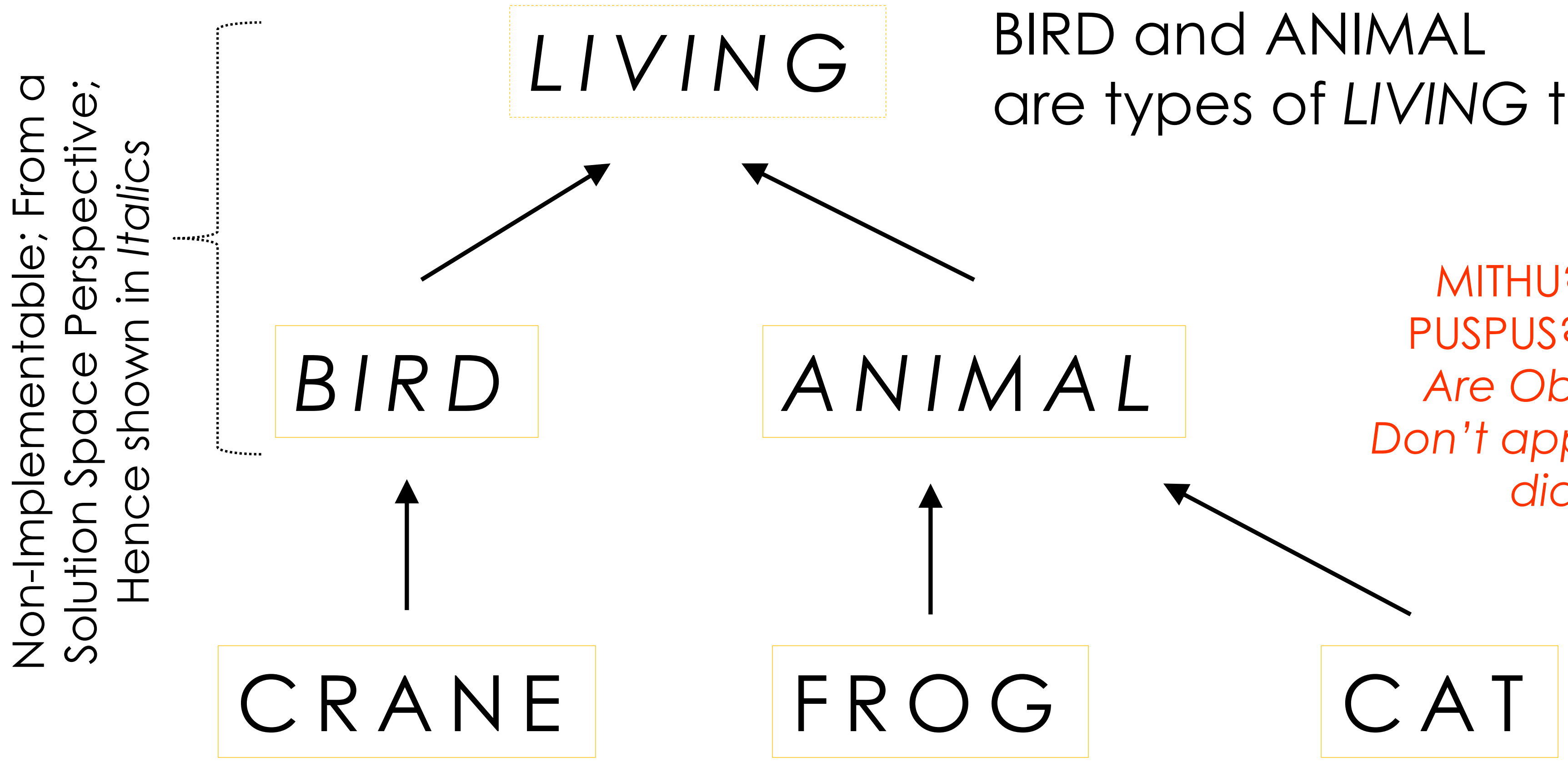
- Relating Classes


The Association Relationship provides a mechanism for two (more) Classes to relate to each other to achieve System Objectives

Class Person *associates with* class Clock

Inheritance

- Creating a Inheritance Hierarchy by Identifying more *general* groupings:
  (Applying Classification and Abstraction to arrive at Inheritance)

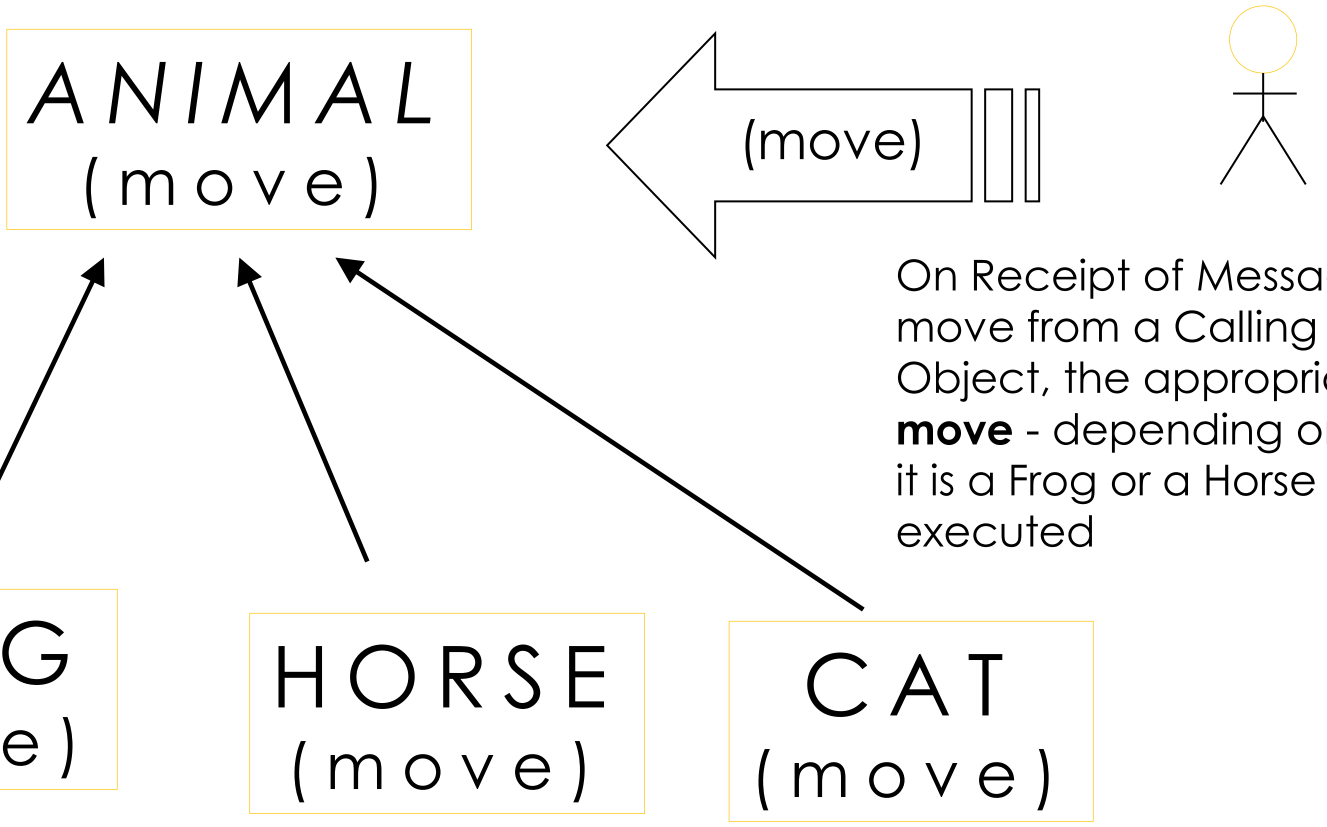

Non-Implementable; From a Solution Space Perspective; Hence shown in *Italics*
LIVING
BIRD
ANIMAL
CRANE
FROG
CAT

BIRD and ANIMAL are types of *LIVING* things

MITHU? (my Bird)
PUSPUS? (your Cat)
Are Objects; They Don't appear on Class diagrams

Polymorphism

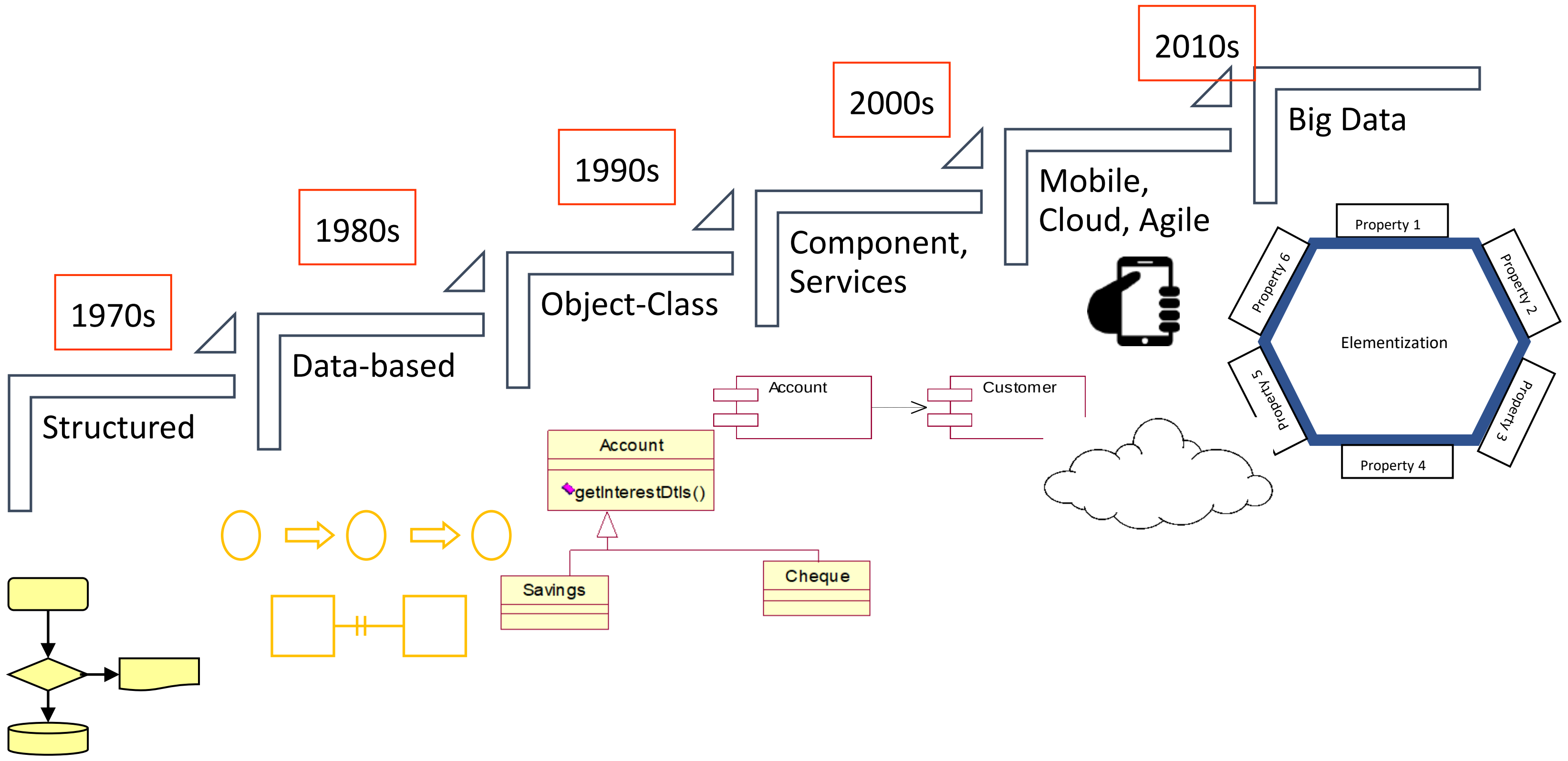
- Executing – “move”



On Receipt of Message move from a Calling Object, the appropriate **move** - depending on whether it is a Frog or a Horse - will be executed

Advantage? CALLING object need not know what is Moved, so, if a *new* CAT object is added, the CALLING class doesn’t change

Evolution of Modeling



Flowcharts
SSADM

Entity-Relationships;
Data Flows;

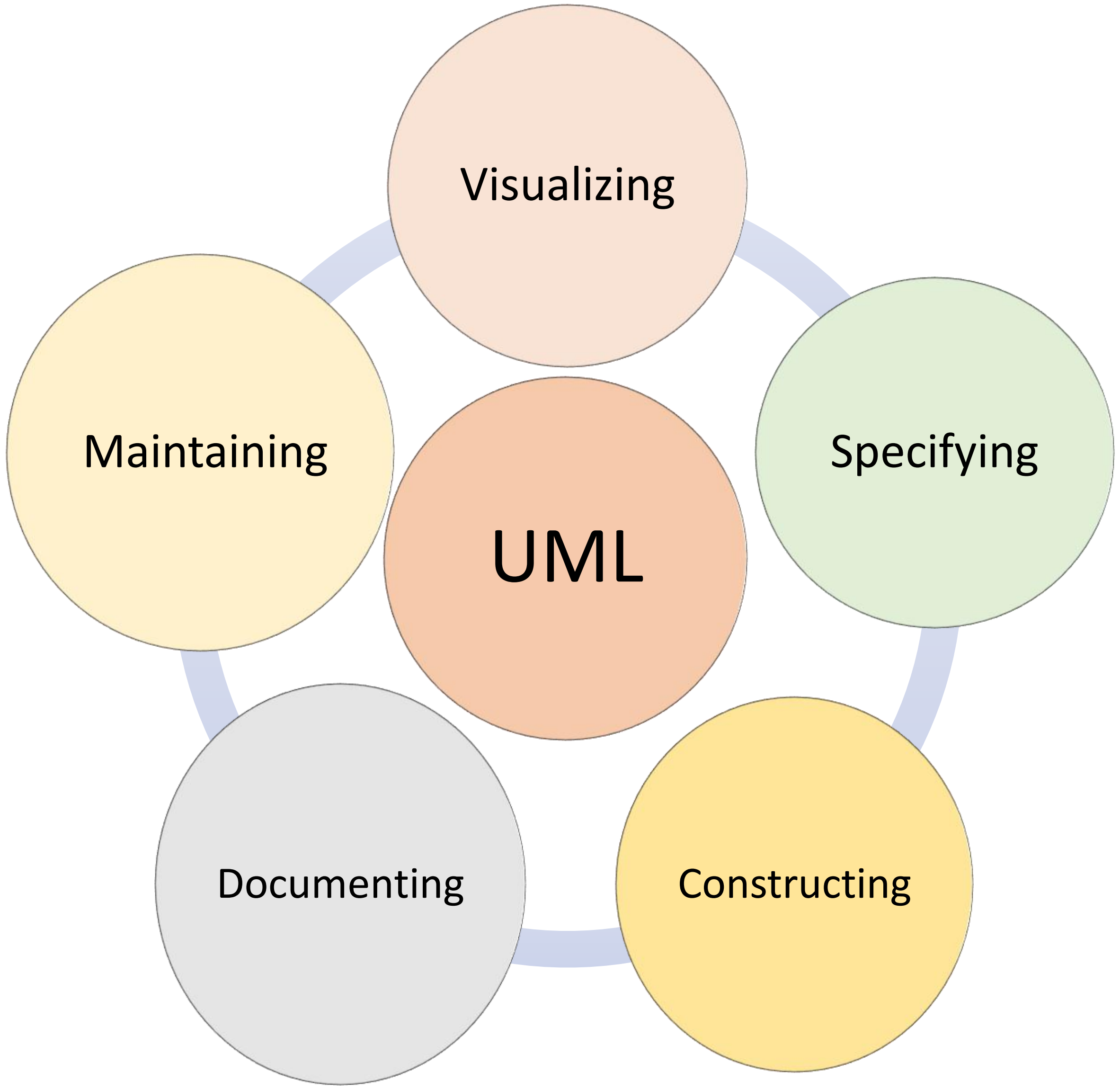
Unified Modelling Language

Mobile Apps & Big Data Modeling (Agile)

UML – The Purpose

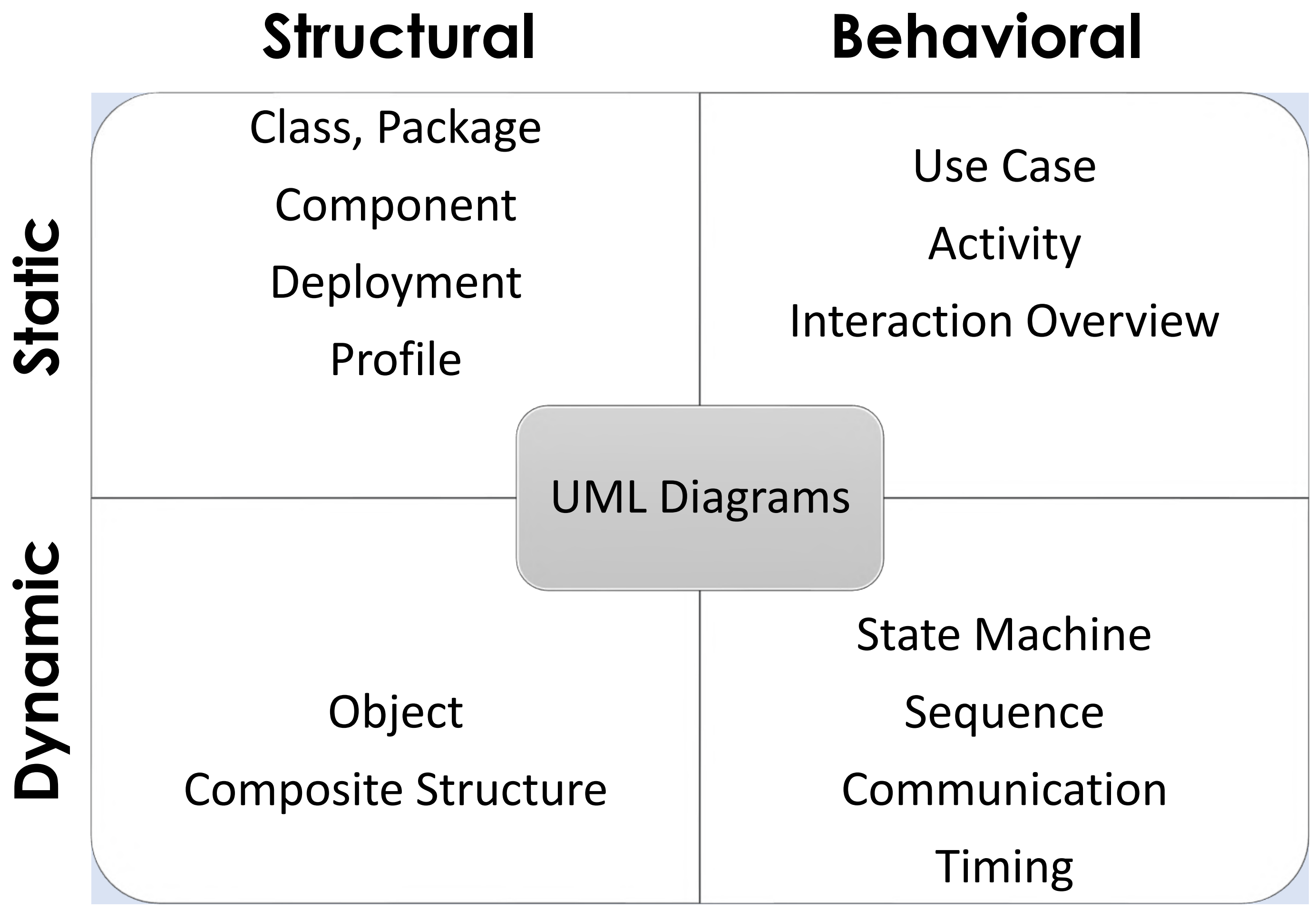
Basic Foundation of UML

- Visualizing
- Specifying
- Constructing
- Documenting
- Maintaining



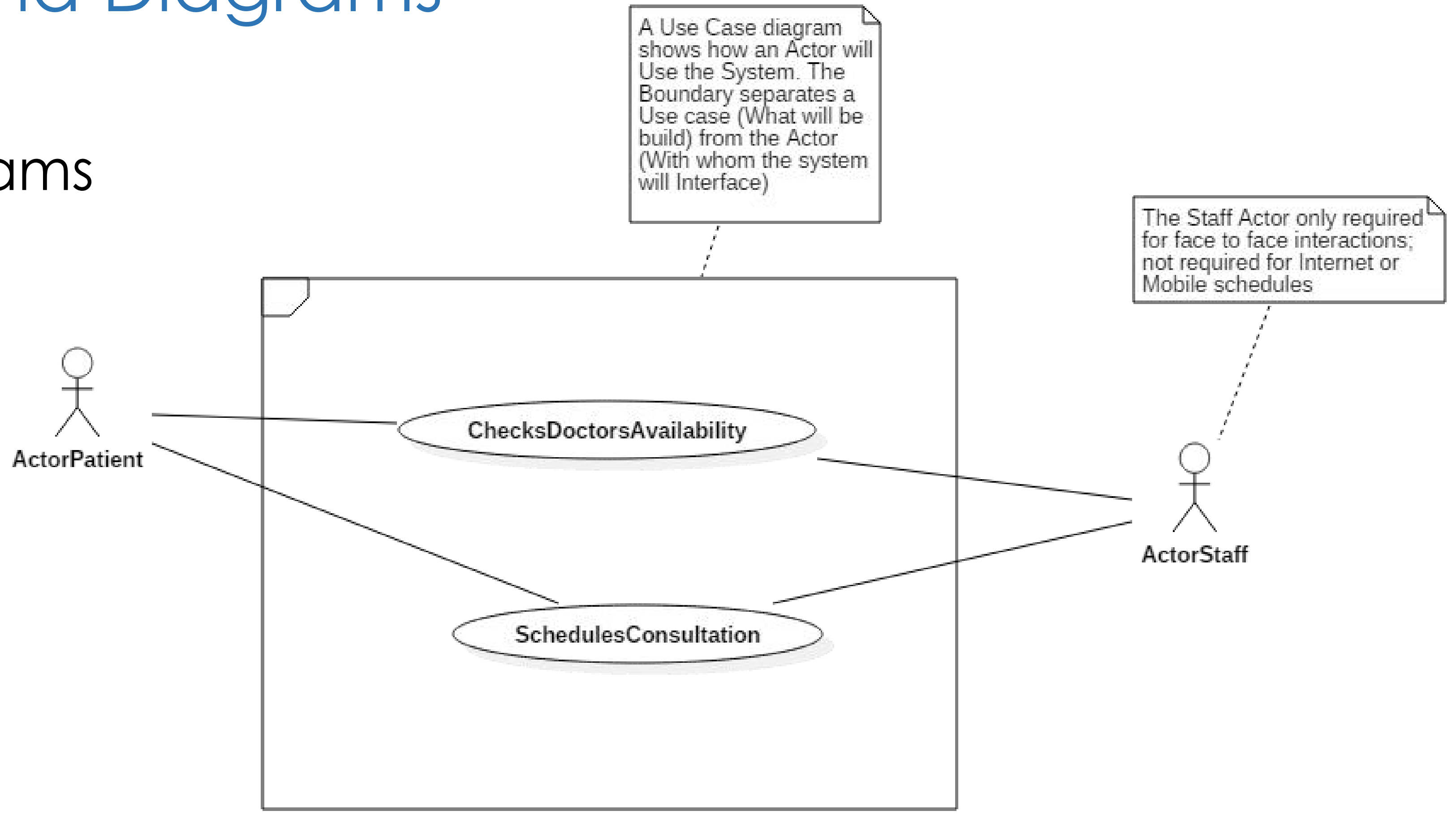
UML – Models and Diagrams

- Understanding UML Diagrams



UML – Models and Diagrams

- Use Case Diagrams

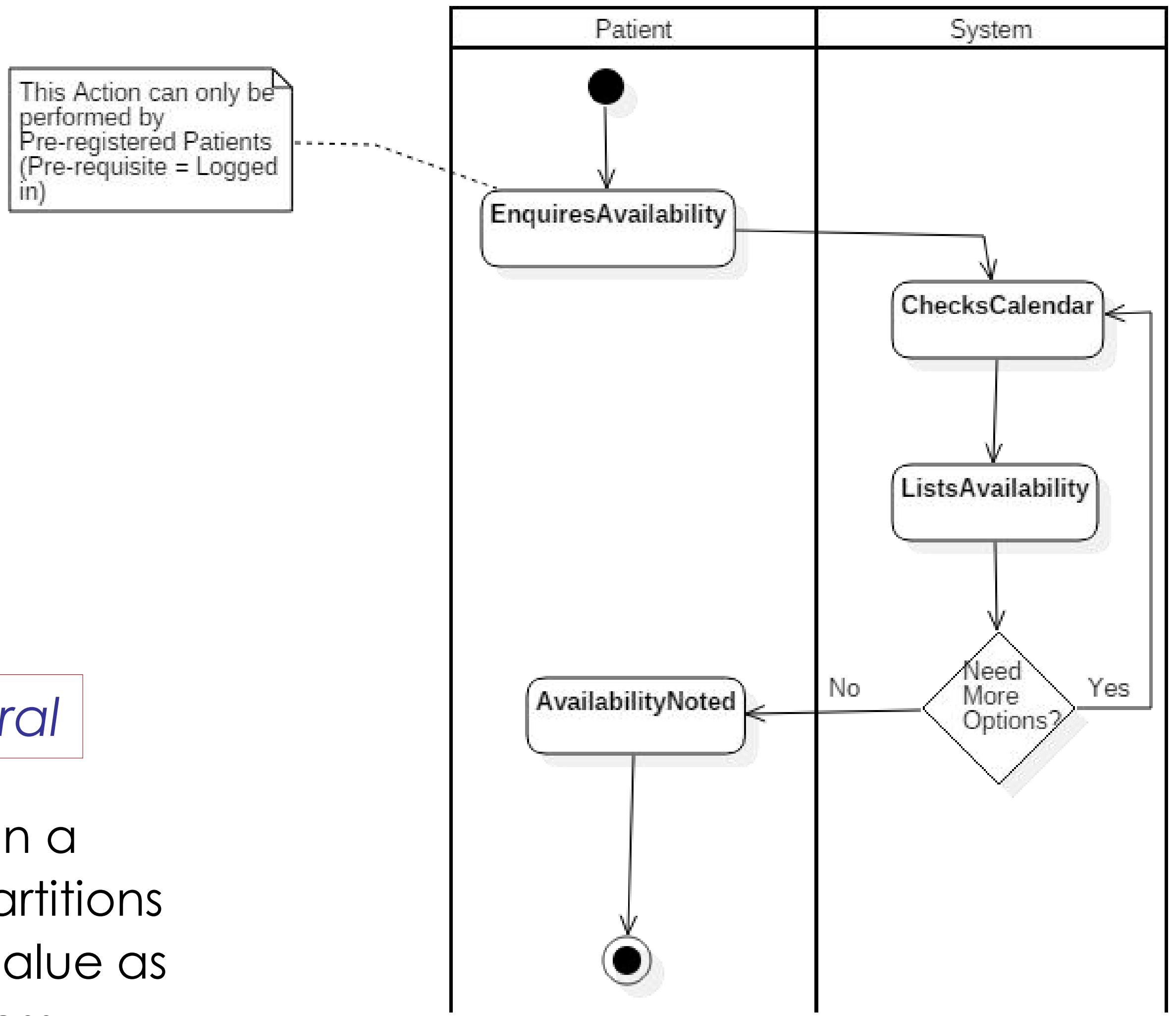

A Use Case diagram shows how an Actor will Use the System. The Boundary separates a Use case (What will be build) from the Actor (With whom the system will Interface)
The Staff Actor only required for face to face interactions; not required for Internet or Mobile schedules
ActorPatient
ChecksDoctorsAvailability
SchedulesConsultation
ActorStaff

Nature: *Static - Behavioral*

Use Case diagrams providing an overview of the Requirements through Actors and Use cases. Internal Documentation of the Use cases contains details of the interactions between Actor and System.

UML – Models and Diagrams

- Activity Diagrams

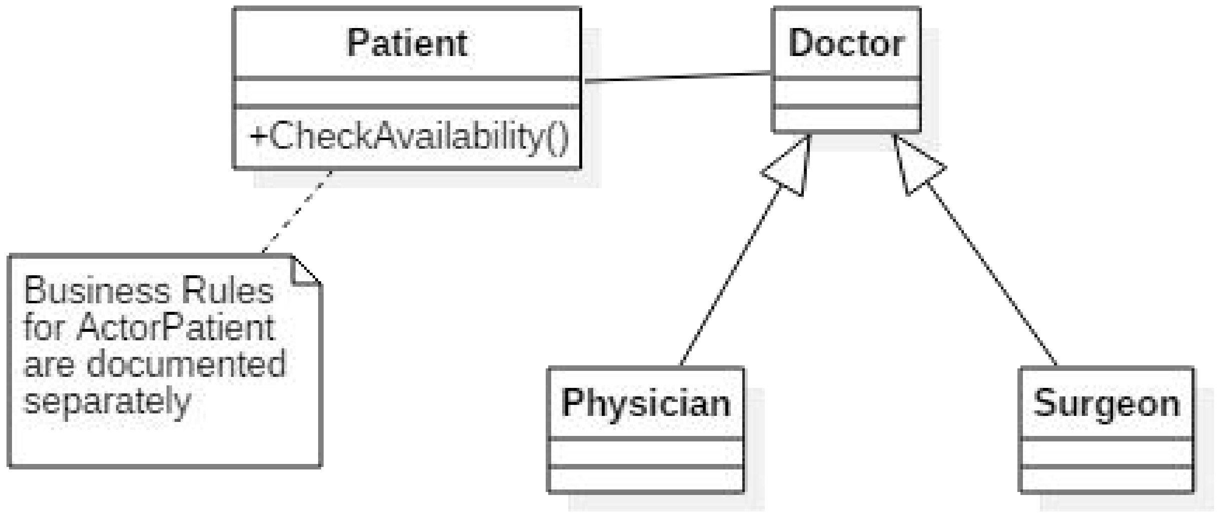


Nature: *Static-Behavioural*

Activity Diagrams represent the Flow within a Use case – primarily its Documentation. Partitions and Multiple threads provide additional value as they also help optimize the business process.

UML – Models and Diagrams

- Class Diagrams

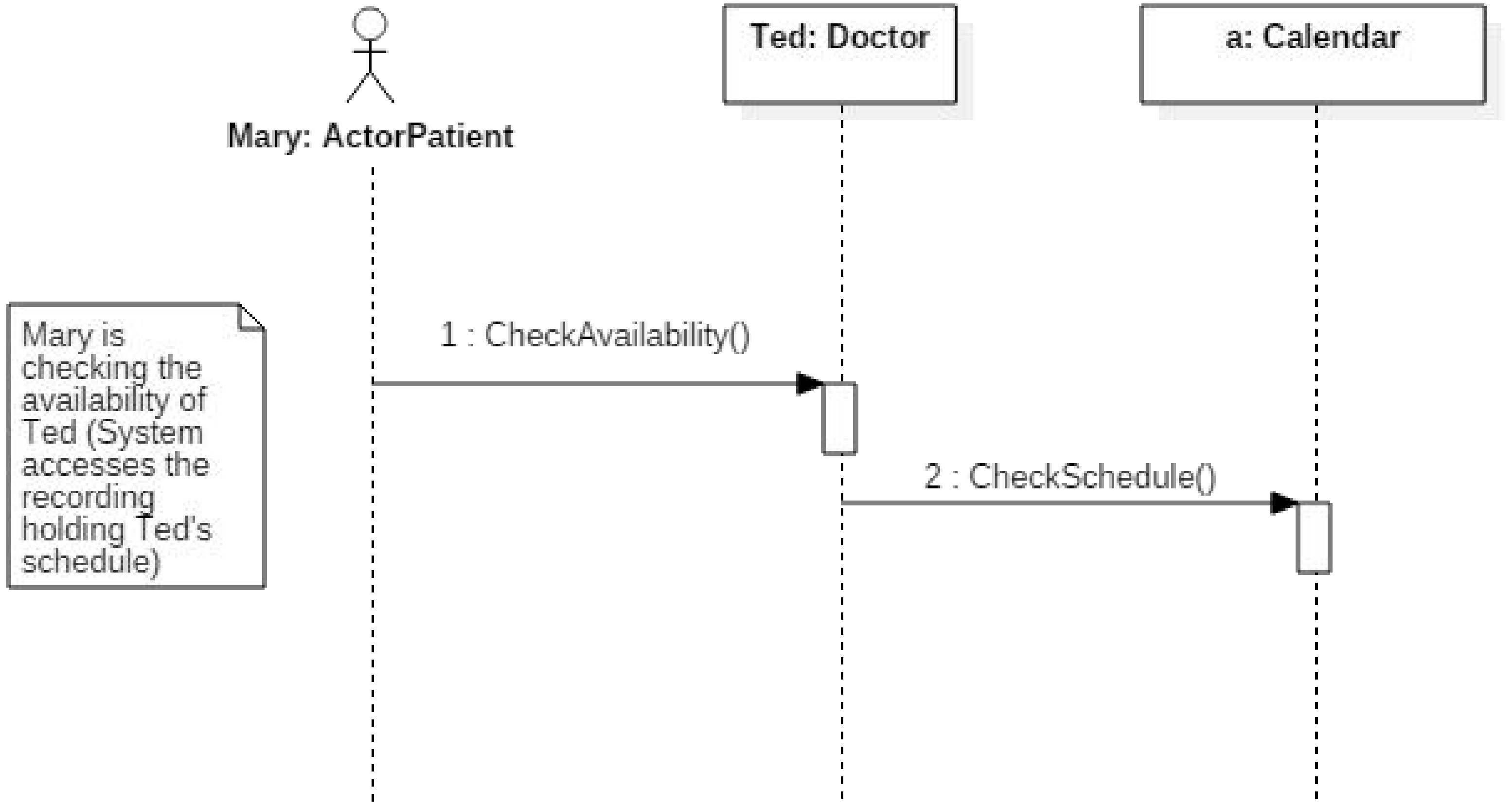


NATURE: *STATIC-STRUCTURAL*

Class diagrams model Entities (i.e. Classes at Business and Technical levels) and their relationships. Classes on these diagrams contain Attributes and Operations (which can be visible or hidden), Relationships (Inheritance, Association) and Multiplicities.

UML – Models and Diagrams

- Sequence Diagrams


Mary: ActorPatient
Ted: Doctor
a: Calendar
Mary is checking the availability of Ted (System accesses the recording holding Ted's schedule)
1 : CheckAvailability()
2 : CheckSchedule()

NATURE: *DYNAMIC-BEHAVIORAL*

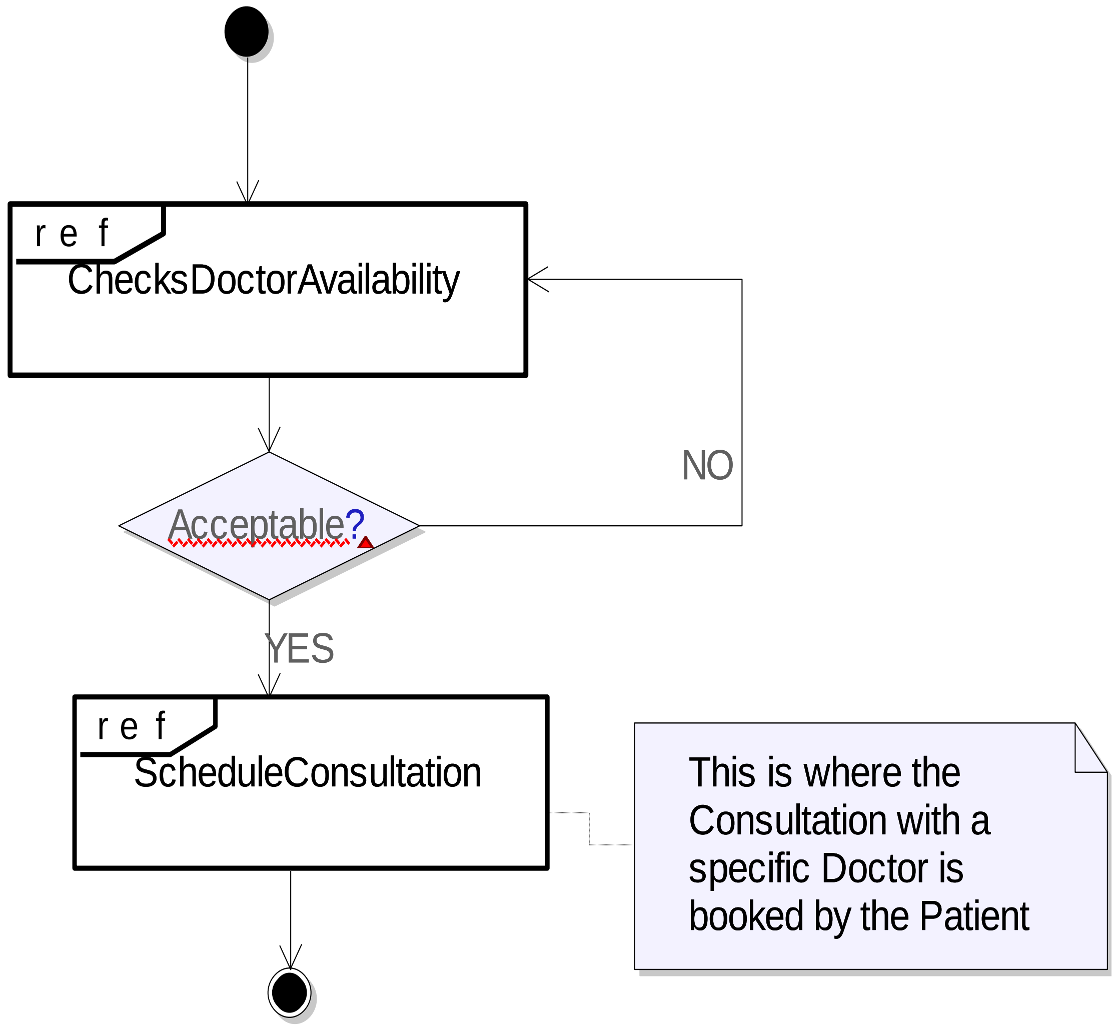
Sequence Diagrams show a single scenario of Interactions between Objects and System (through messages). The sequence of messages is important. These diagrams may contain Actors. Sequence diagrams cannot show conditions (“if-then-else”).

UML – Models and Diagrams

- Interaction Diagrams

NATURE: *STATIC-BEHAVIORAL*

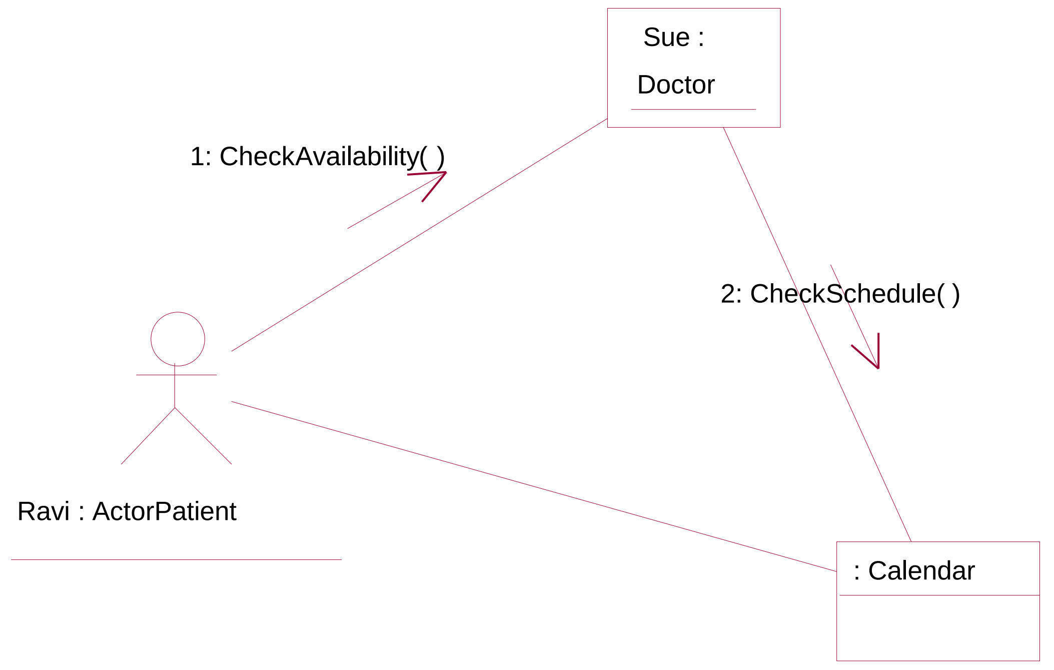
Interaction Overview Diagrams provide a high-level overview of how other diagrams – such as Sequence diagrams or even Use cases – are related to each other; they reference interactions; conceptually, they are similar to activity diagrams (as they have a flow within them)



ref
ChecksDoctorAvailability
NO
Acceptable?
YES
ref
ScheduleConsultation
This is where the Consultation with a specific Doctor is booked by the Patient

UML – Models and Diagrams

- Communication Diagrams

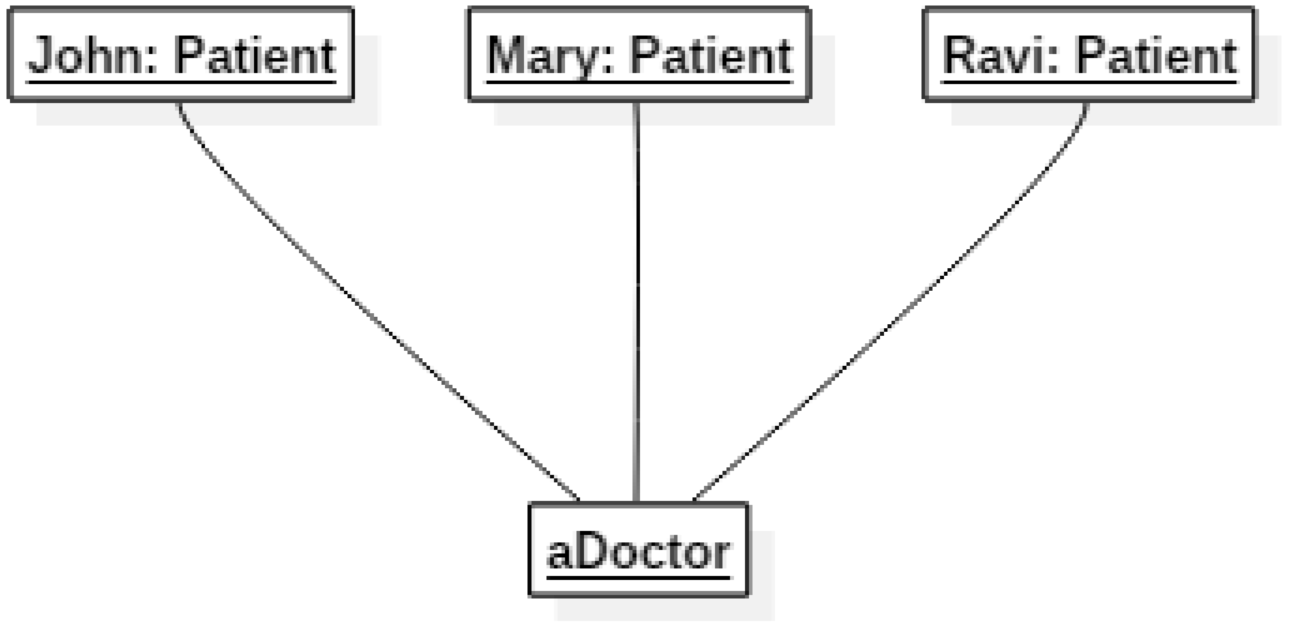


NATURE: *DYNAMIC-BEHAVIOURAL*

Communication Diagrams are an alternative view to the Sequence diagrams. These diagrams model interactions between Objects and their links to each other. The sequencing of messages is depicted by numbers.

UML – Models and Diagrams

- Object Diagram

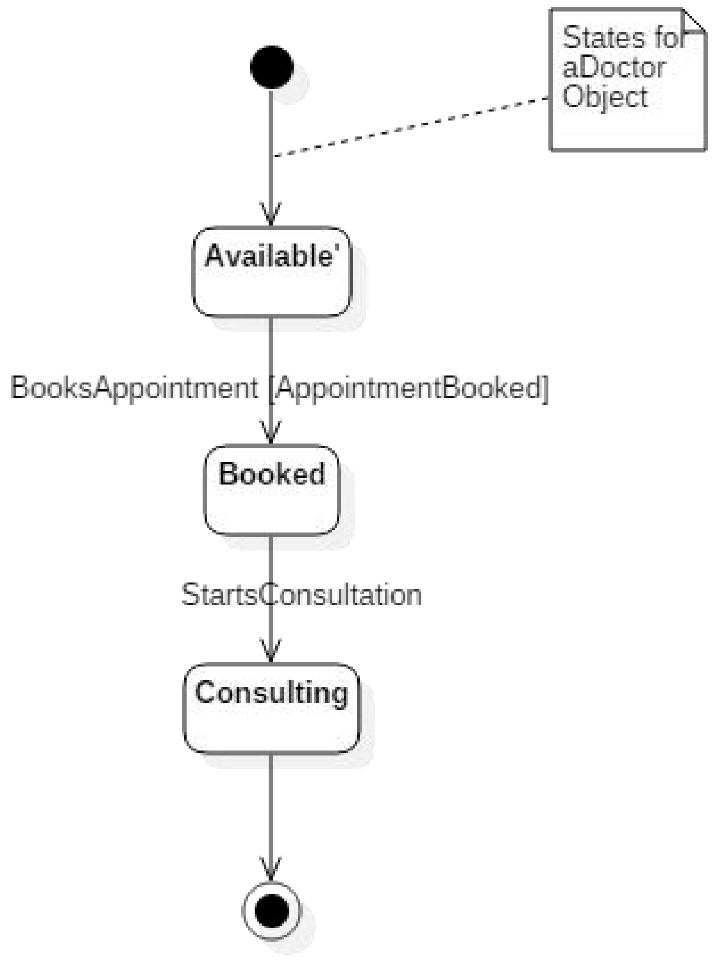


Nature: Dynamic-Structural

Object diagrams describe the various Objects (*instances)* and how they relate to each other. The relationships are *links* in the memory. Being instance level diagrams, they are ideal in depicting the multiplicities between classes.

UML – Models and Diagrams

- State Machine Diagram

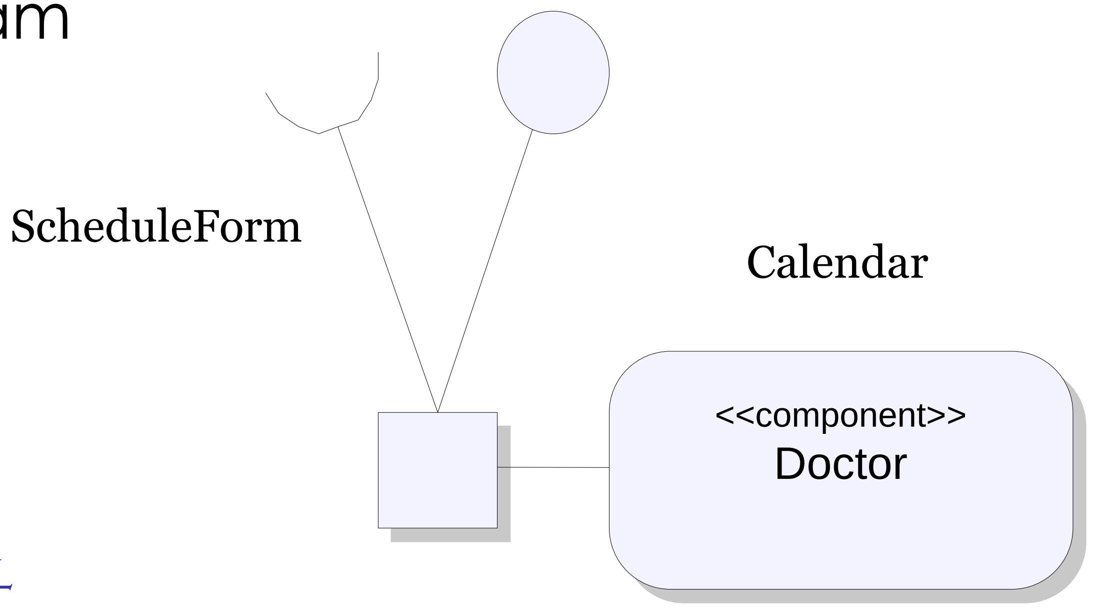

States for aDoctor Object
Available'
BooksAppointment [AppointmentBooked]
Booked
StartsConsultation
Consulting

NATURE:
DYNAMIC-BEHAVIORAL

State machine diagrams show the various States of an Object; they also show the Events and Guard Conditions under which a change in State occurs for an Object. They are usually referred to by their corresponding Class name.

UML – Models and Diagrams

- Composite Structure Diagram

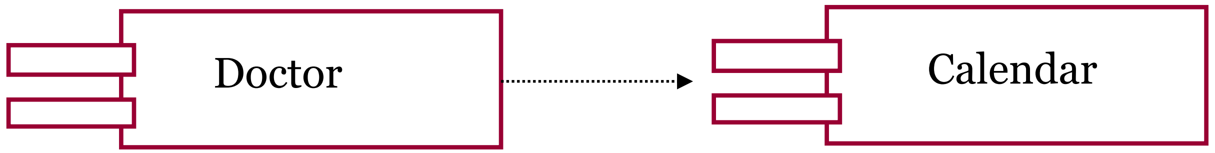


NATURE: *DYNAMIC-STRUCTURAL*

Composite structure diagrams show links and decompositions of components as well as objects at run-time in the memory

UML – Models and Diagrams

- Component Diagrams

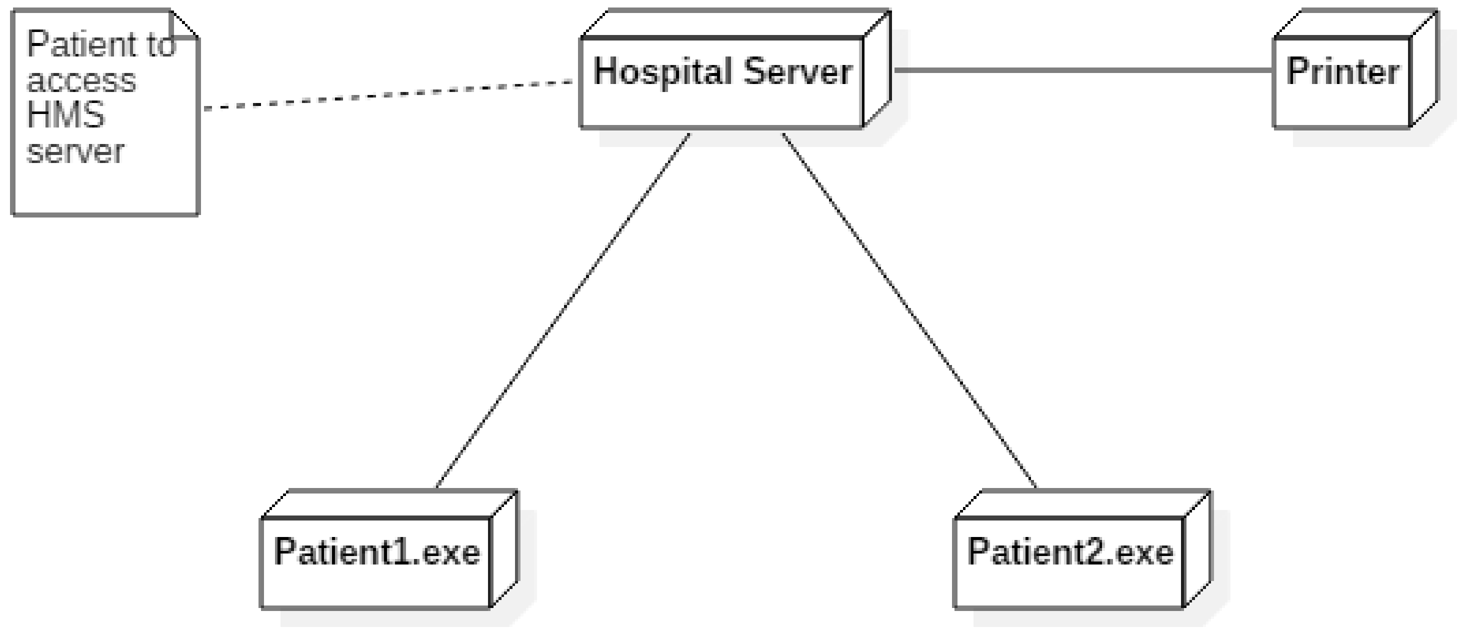


NATURE: *STATIC-STRUCTURAL*

Component diagrams are organizational in nature as they show the composition, organization and dependencies amongst software components. They are not Object-oriented in nature.

UML – Models and Diagrams

- Deployment Diagram

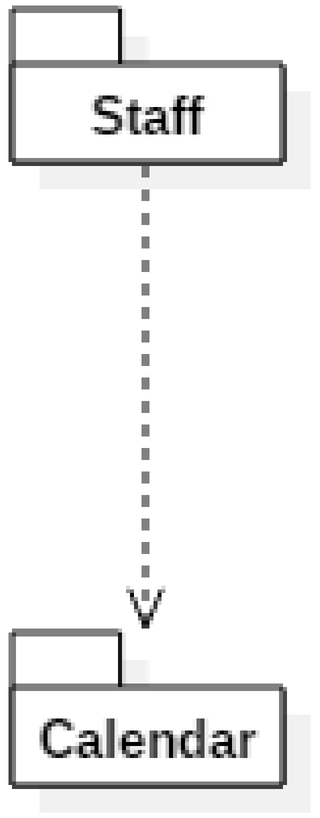


NATURE: *STATIC-STRUCTURAL*

Deployment Diagram shows the manner in which a system will be deployed when in operation. These diagrams show processes and nodes in the physical design of a system. They are the only hardware diagram in the UML

UML – Models and Diagrams

- Package Diagrams

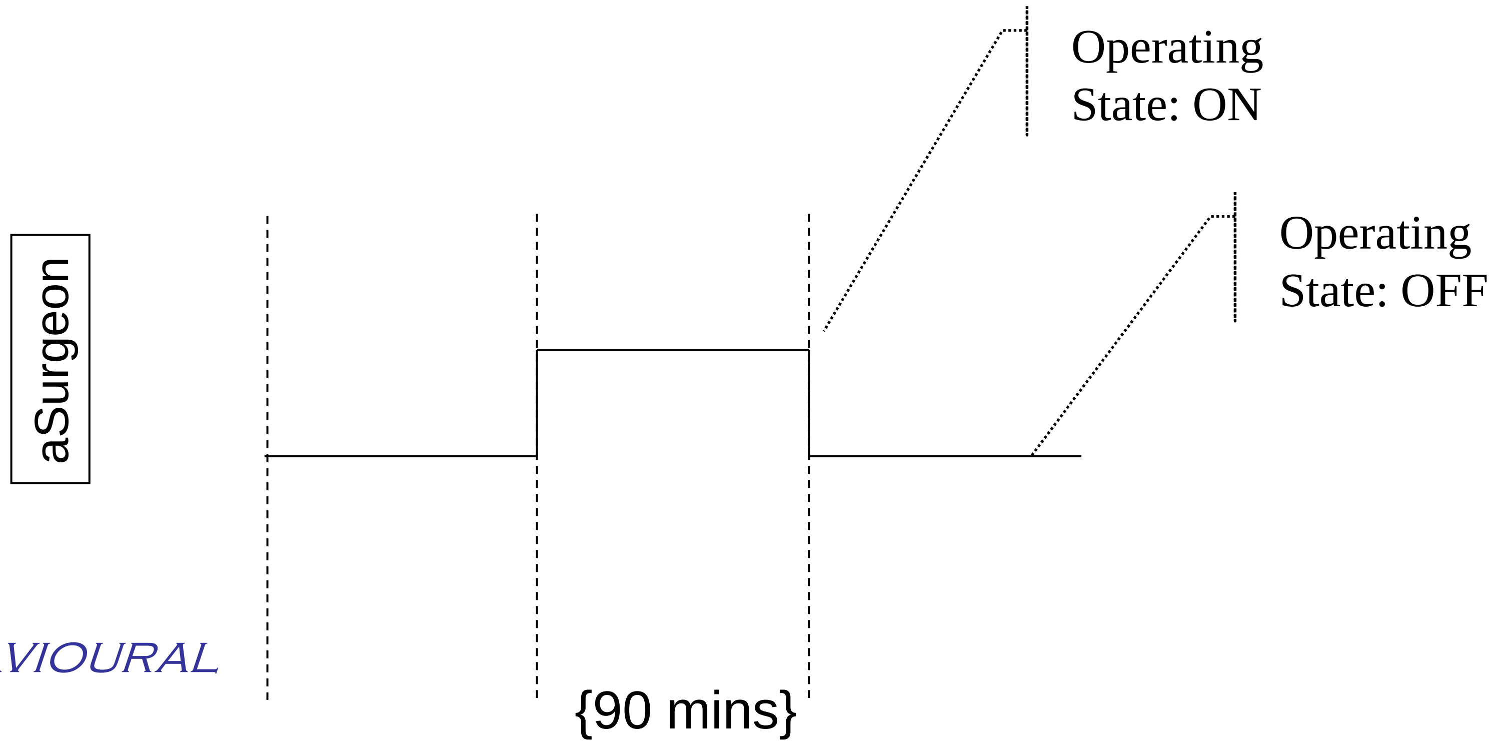


NATURE: *STATIC-STRUCTURAL*

Packages represent sub-systems; They comprise large and cohesive collection of other UML diagrams. Package diagrams are organizational in nature and they show Packages and their dependencies.

UML – Models and Diagrams

- Timing Diagrams

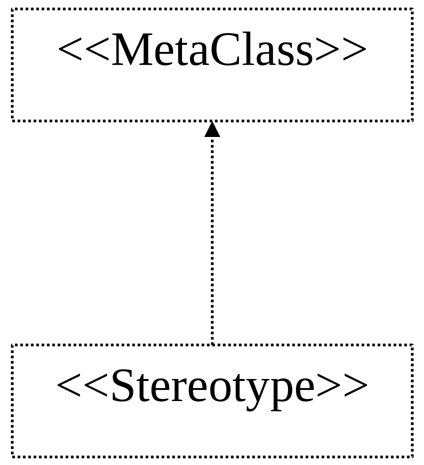


Nature: *Dynamic-Behavioural*

Timing Diagrams show one or more Objects and their states. The time constraints are also shown on this diagram. They help compare states of multiple Objects at points in time.

UML – Models and Diagrams

- Profile Diagram



NATURE: *STATIC-STRUCTURAL*

Profile Diagrams enable extension of UML such as visualization of Stereotypes (e.g. inheriting all the characteristics of a Meta-class as shown in this diagram)

Use of UML Models and Diagrams in Software Projects

- Various Types of Software Projects

New Software Development

- Models of Requirements; Databases; Architectures

Integrating Applications & Services

- Models of Services - External Systems - Interfaces

Package Implementation

- Enterprise Architecture; Process maps;

Mobile App Development

- Storyboarding; Mockups; Algorithms

Business Process Modeling

- Workflow & Activity Modelling;

Cloud-based Service Deployment

- Service Configuration; Analytics Deployment

Small	Medium	Large	Collaborative
Solution Space	Problem & Solution Space	+ Architecture Space	Services, Cloud, IoT

Use of UML Models and Diagrams in Software Projects

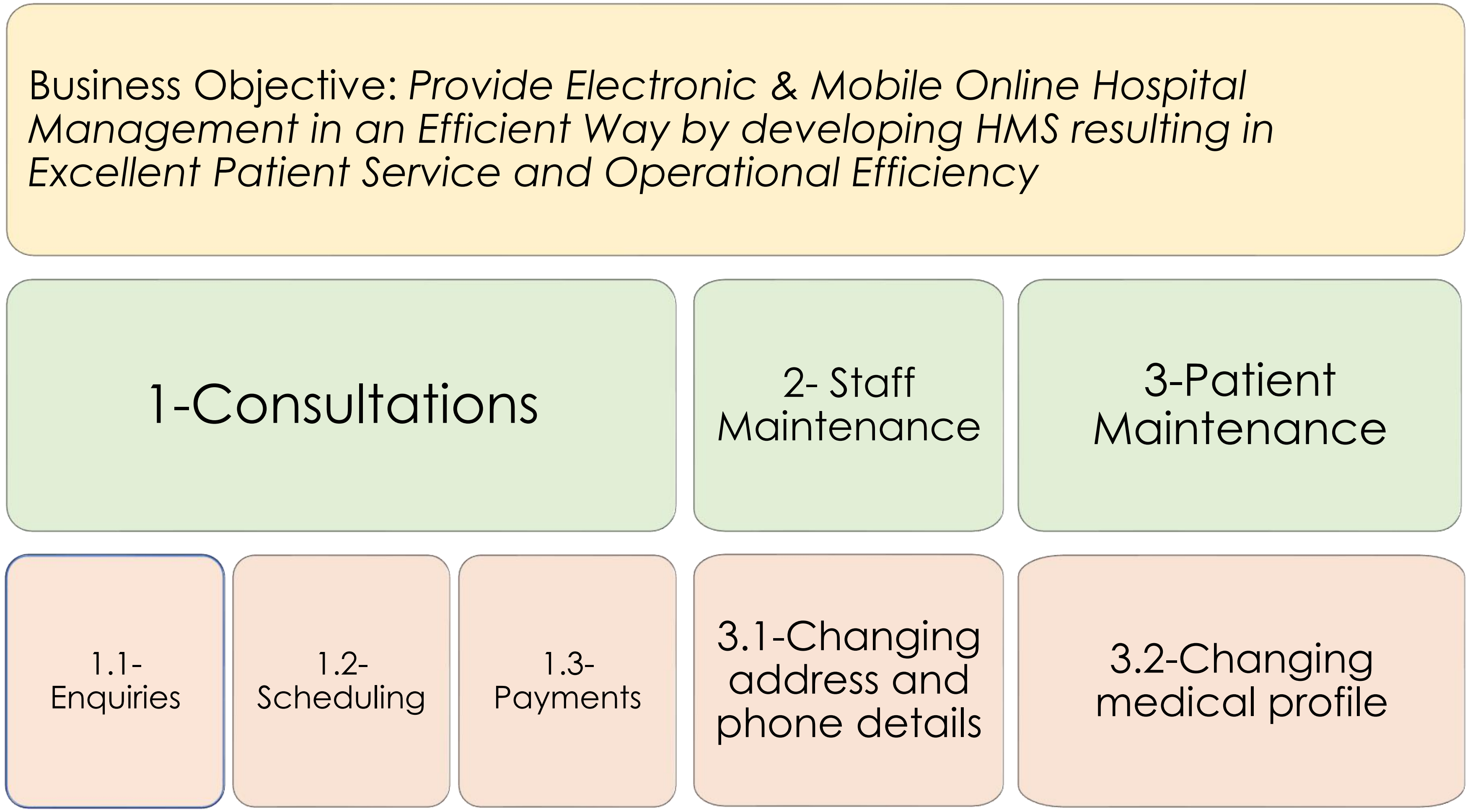
Problem Statement of a Hospital Management System

- To provide electronic and mobile hospital management in an efficient way (WHAT)
- By developing and implementing a hospital management system (HOW)
- Resulting in an excellent patient service and operational efficiency (WHY)

Use of UML Models and Diagrams in Software Projects

- Prioritization of Requirements:

Hospital Management System


Business Objective: *Provide Electronic & Mobile Online Hospital Management in an Efficient Way by developing HMS resulting in Excellent Patient Service and Operational Efficiency*
1-Consultations
2- Staff Maintenance
3-Patient Maintenance
1.1-Enquiries
1.2-Scheduling
1.3-Payments
3.1-Changing address and phone details
3.2-Changing medical profile

Use of UML Models and Diagrams in Software Projects

- Applying Performance Criteria

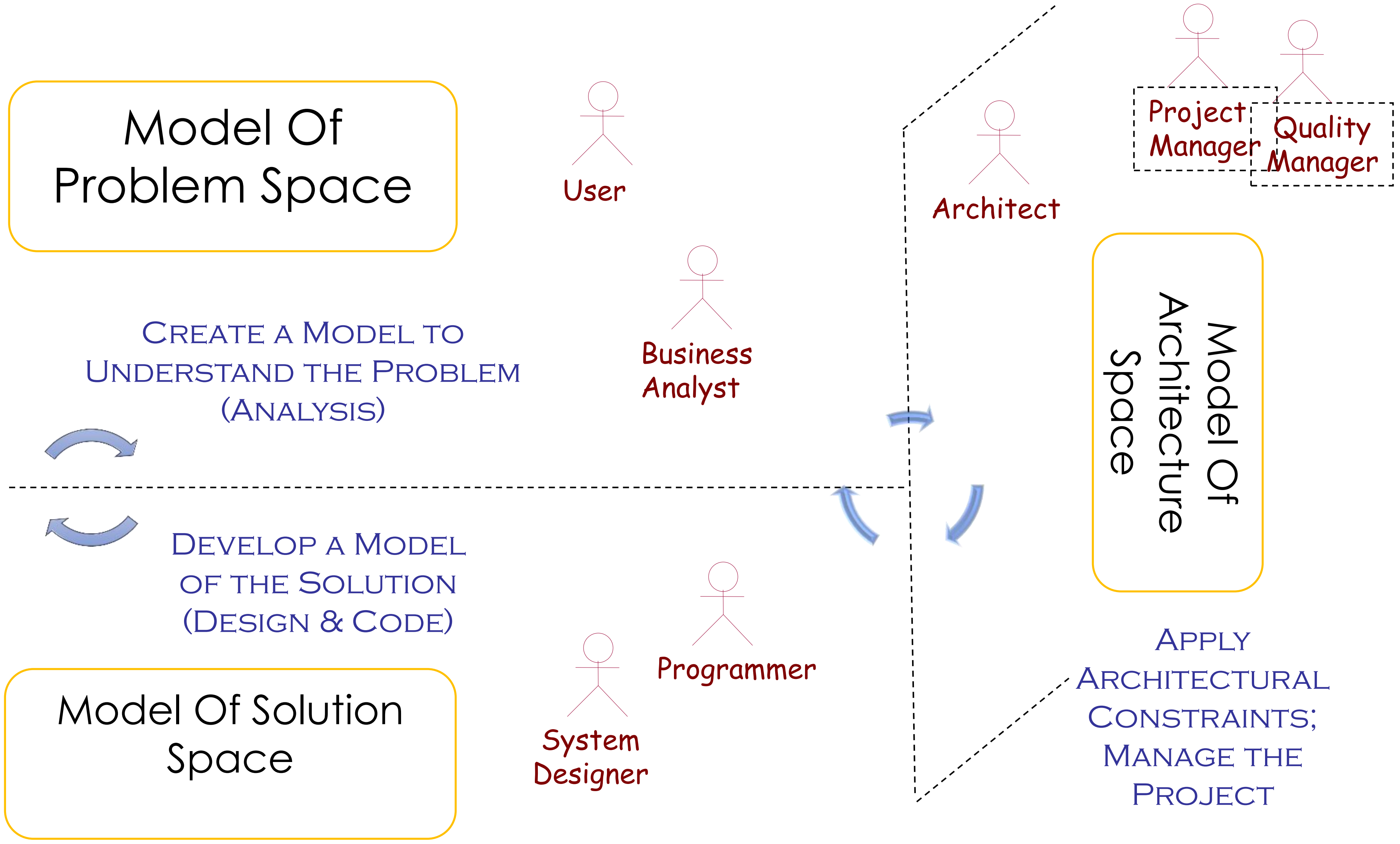
Decomposition	Performance Criteria
Provide efficient response to patient interaction with hospital	• Response time less than 3 seconds • Response accuracy 99% • Response cost – less than 3 cents per transaction
Consultations	• 100% of inquiries
Staff Maintenance	• Manage the details
Patient maintenance	• 99% of address changes

Use of UML Models and Diagrams in Software Projects

Three Modeling Spaces

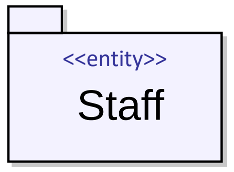
- The Problem Space – Focus on the "WHAT"
- The Solution Space - - Focus on the "HOW"
- The Architecture Space – Focus on the orthogonal to Problem and Solution spaces

Use of UML Models and Diagrams in Software Projects

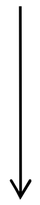

Model Of Problem Space
User
Architect
Project Manager
Quality Manager
Business Analyst
Create a Model to Understand the Problem (Analysis)
Model Of Architecture Space
Develop a Model of the Solution (Design & Code)
Programmer
Model Of Solution Space
System Designer
Apply Architectural Constraints; Manage the Project

Use of UML Models and Diagrams in Software Projects

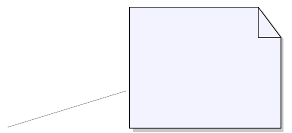
- Creating Package Diagram



PACKAGE –
REPRESENTS A SUBSYSTEM



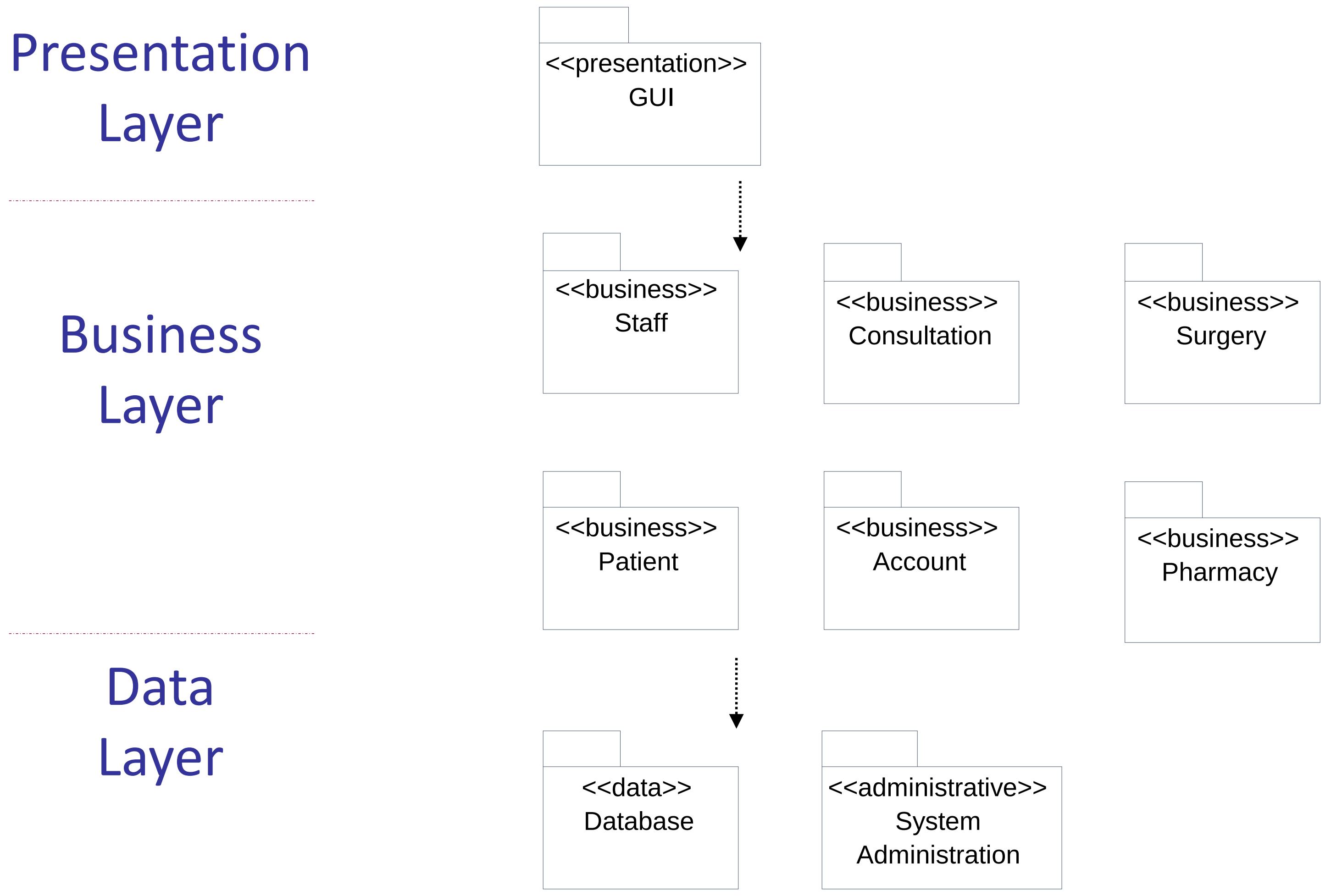
DEPENDENCY –
OPTIONAL



NOTE –
CLARIFIES THE DIAGRAM

Use of UML Models and Diagrams in Software Projects

- Profile Diagram


Presentation Layer
Business Layer
Data Layer
<<presentation>> GUI
<<business>> Staff
<<business>> Consultation
<<business>> Surgery
<<business>> Patient
<<business>> Account
<<business>> Pharmacy
<<data>> Database
<<administrative>> System Administration