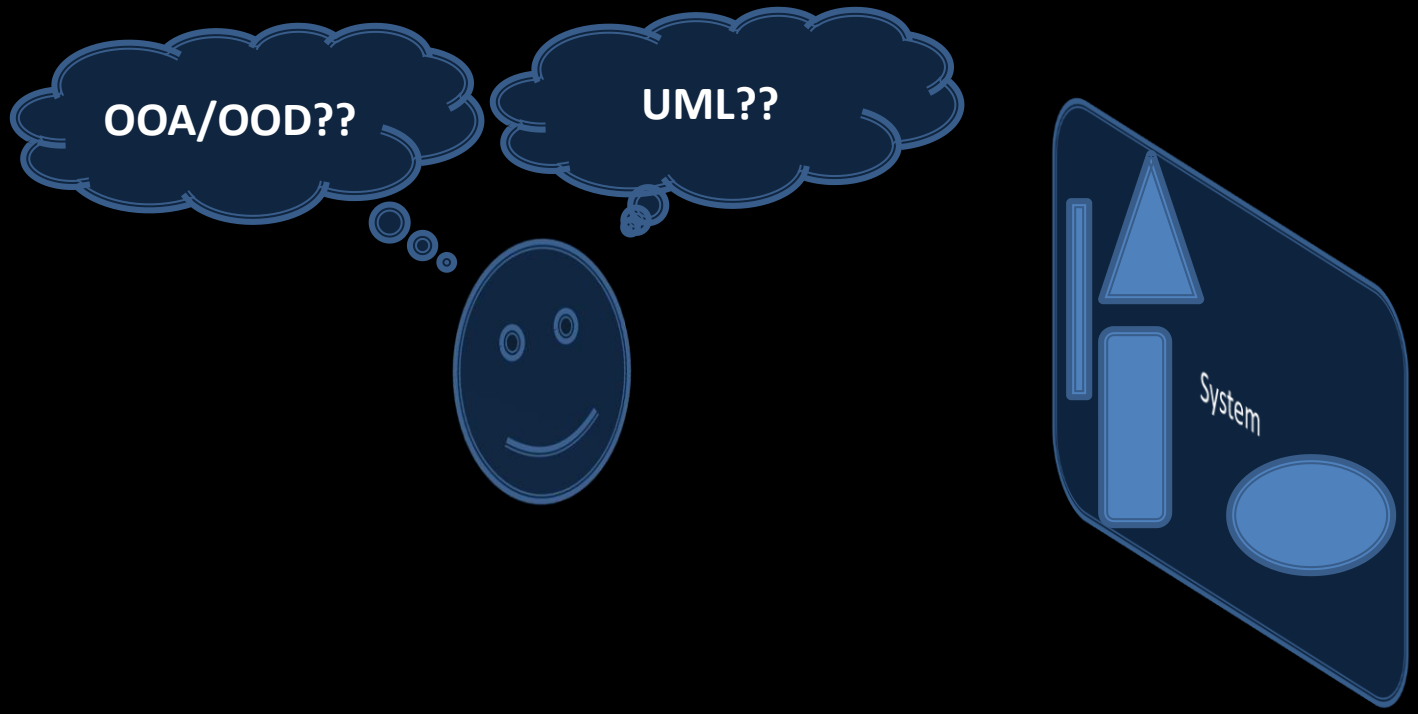
Overview



Specify, Visualize, Construct and Document Software Artifacts!!!

Analysis and Design

Owning a hammer does not make one an architect

Knowing Java is a necessary but insufficient step to create object systems

Knowing how to think in objects is critical

Difference between Analysis and Design

Analysis:

Investigation of the problem domain and requirements rather than a solution

e.g. an Online trading system: how will it be used? What are its functions?

Design:

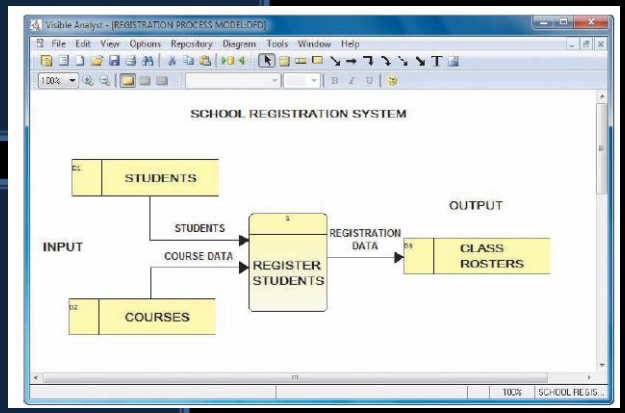
Conceptual solution for fulfilling requirements rather than its implementation

Obvious details are excluded

Structured Analysis and Design

Structured Analysis

- Systems development life cycle (SDLC)
- Predictive approach
- Uses a set of process models to describe a system graphically
- Process-centered technique
- Waterfall model



Structured Design

methods for analyzing and converting business r requirements into specifications

Object-oriented Analysis and Design

UML: a standard **diagramming notations**

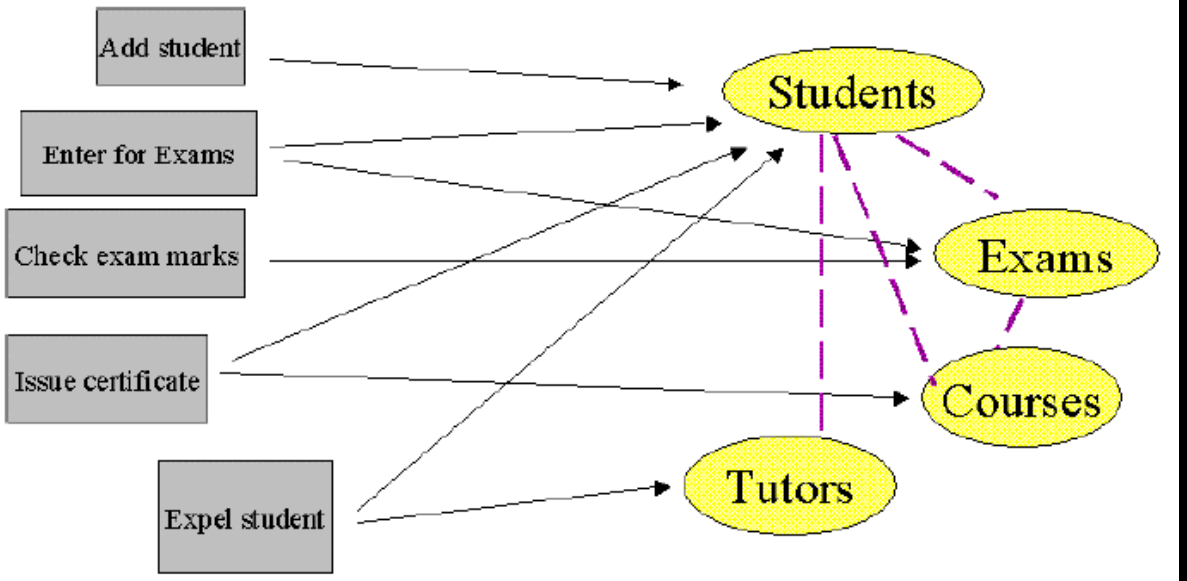
OOA: Emphasis on finding and describing objects or concepts in problem domain. For example, in the case of the library information system, some of the concepts include *Book, Library,* and *Patron.* **WHAT??**

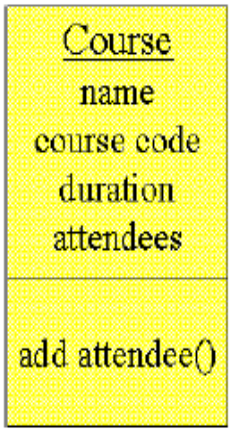
OOD: Emphasis on defining software objects and how they collaborate to fulfill the requirements.
For example, in the library system, a *Book software object may have a title attribute and a getChapter method .* **HOW??**

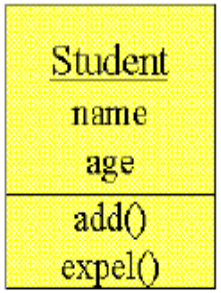
Finally, during implementation or object-oriented programming, **design objects are implemented**, such as a *Book class in Java.*

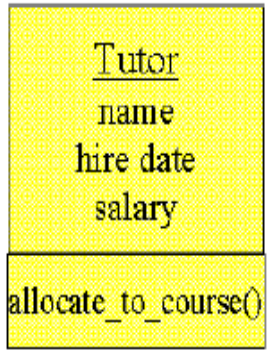
College management system

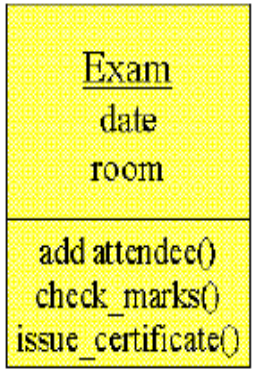
This system holds the details of every student and tutor in the college. In addition, the system also stores information about the courses available at the college, and tracks which student is following which courses.
A possible functional design would be to write the following functions:
add_student
enter_for_exam
check_exam_marks
issue_certificate
expel_student


Add student
Enter for Exams
Check exam marks
Issue certificate
Expel student
Students
Exams
Courses
Tutors


Course
name
course code
duration
attendees
add attendee()


Student
name
age
add()
expel()


Tutor
name
hire date
salary
allocate_to_course()


Exam
date
room
add attendee()
check_marks()
issue_certificate()

Object Oriented System Development

Object Oriented Software Development life cycle consists of three macro processes:
Object Oriented Analysis
Object Oriented Design
Object Oriented Implementation

Object-oriented Analysis and Design

Domain concept

Book

title

Visualization of Domain Concept

Representation in an Object Oriented Programming Language

```
public class Book{

   pivate String title;

    public chapter getChapter(int){

    }

}
```

Describing vs Defining

OOA: deals with describing and finding objects in the problem domain e.g. In case of Flight Information System domain objects are plane, flight, pilot

OOD: deals with defining software objects and how they collaborate to achieve the goal of the system
e.g. In case of Flight Information System plane object has filghtNumber attribute, getFilghtHisory ethod

Define Use Case

Define Domain Model

Define interaction Diagrams

Define Design Class Diagrams

Step 1: Define Use Cases

Requirements analysis may include a **description of related domain processes** written as **use cases.**
Like written stories.

Play a Dice Game use case:
Play a Dice Game: A player picks up and rolls the dice. If the dice face value total seven, they win; otherwise, they lose.

Step 2: Define a Domain Model

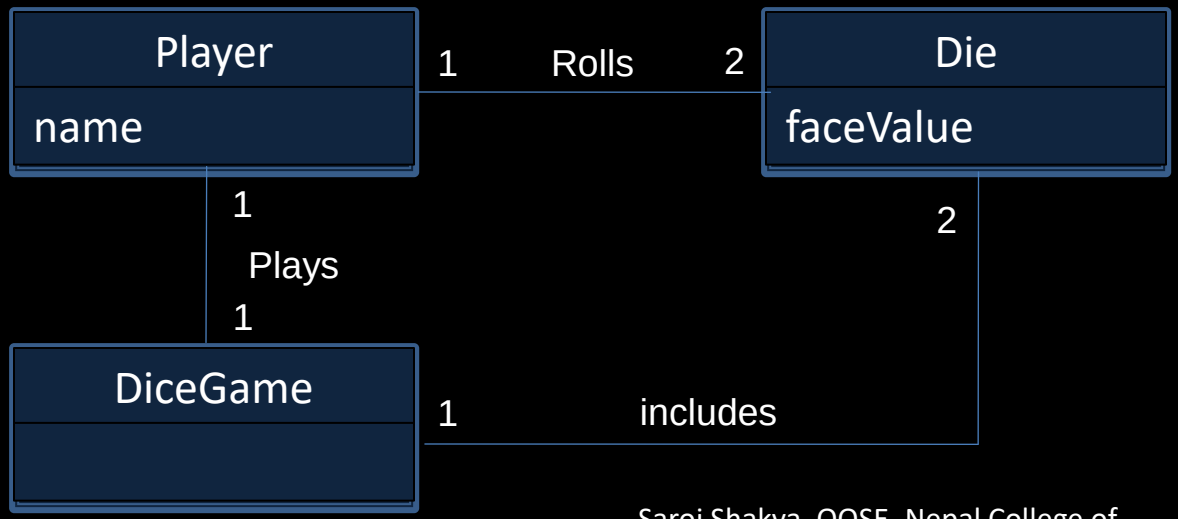
Object-oriented analysis is concerned with **creating a description of the domain** from the perspective of classification by objects.

It is a visualization of concepts in the real-world domain.

A decomposition of the domain involves an **identification of the concepts, attributes, and associations**

A set of diagrams that show domain concepts or objects.

Illustrates *Player, Die, and DiceGame, with* associations and attributes.



Define Use Case

Define Domain Model

Define interaction Diagrams

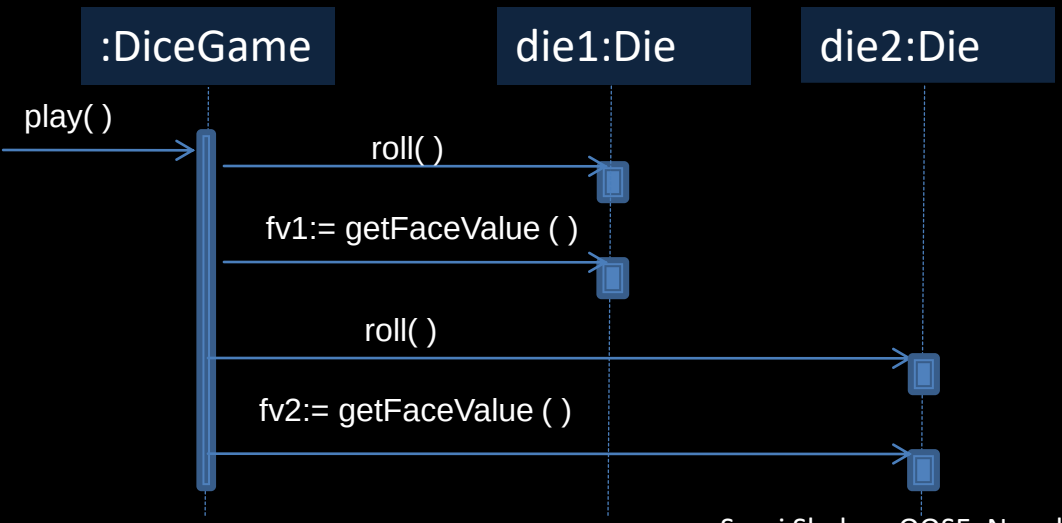
Define Design Class Diagrams

Step 3: Define Interaction Diagrams

Object-oriented design is concerned with defining **software objects and their collaborations.**

Interaction diagram shows the flow of messages between software objects, and thus the invocation of methods.

For example, assume that a software implementation of the dice game is desired. The interaction diagram illustrates the essential step of playing, by sending messages to instances of the *DiceGame and Die classes.*



Define Use Case

Define Domain Model

Define interaction Diagrams

Define Design Class Diagrams

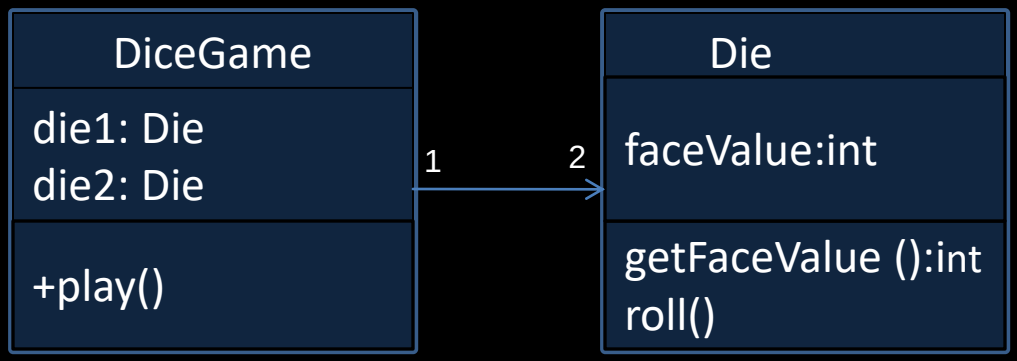
Step 4: Define Design Class Diagrams

Interaction diagrams show *dynamic view of collaborating objects, static view of the class definitions are shown with a* ***design*** **class diagram.**

This illustrates the attributes and methods of the classes.

In the dice game, an inspection of the interaction diagram leads to the partial design class diagram. Since a *play message is* sent to a *DiceGame object, the DiceGame class requires a play method, while* class *Die requires a roll and getFaceValue method.*

Domain model illustrates real world concepts but this diagram shows software classes.



Define Use Case

Define Domain Model

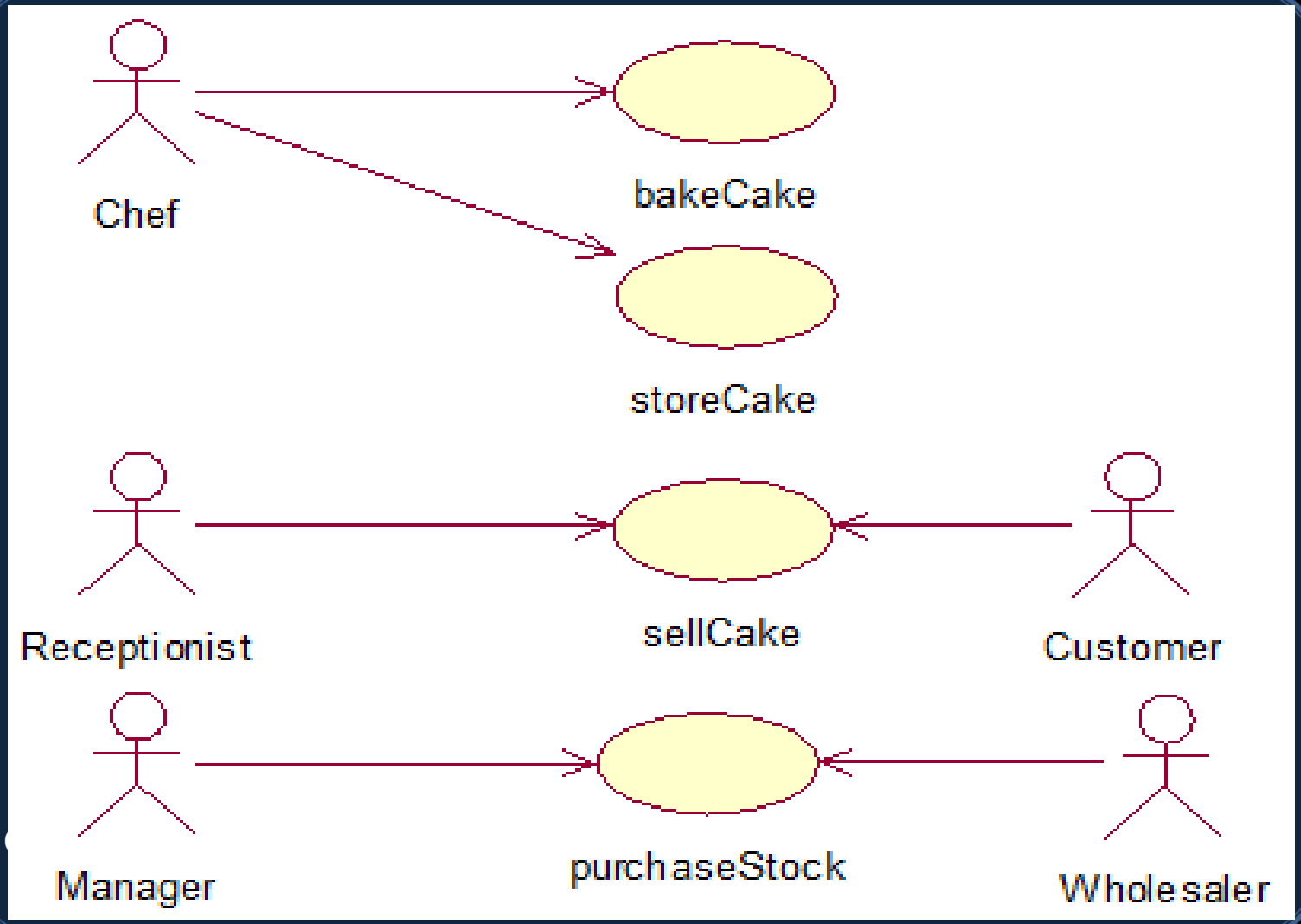
Define interaction Diagrams

Define Design Class Diagrams

Cake Store

What happens in a Cake Store??

Cake Store


Chef
bakeCake
storeCake
Receptionist
sellCake
Customer
Manager
purchaseStock
Wholesaler

ATM MACHINE

We need to build the ATM system to allow a user to withdraw money.
Series of common interactions in this scenario:

- enter card
- enter pin number
- select amount required
- confirm amount required
- remove card
- take receipt

Should each of these steps . for example, .enter pin number. be a use case?

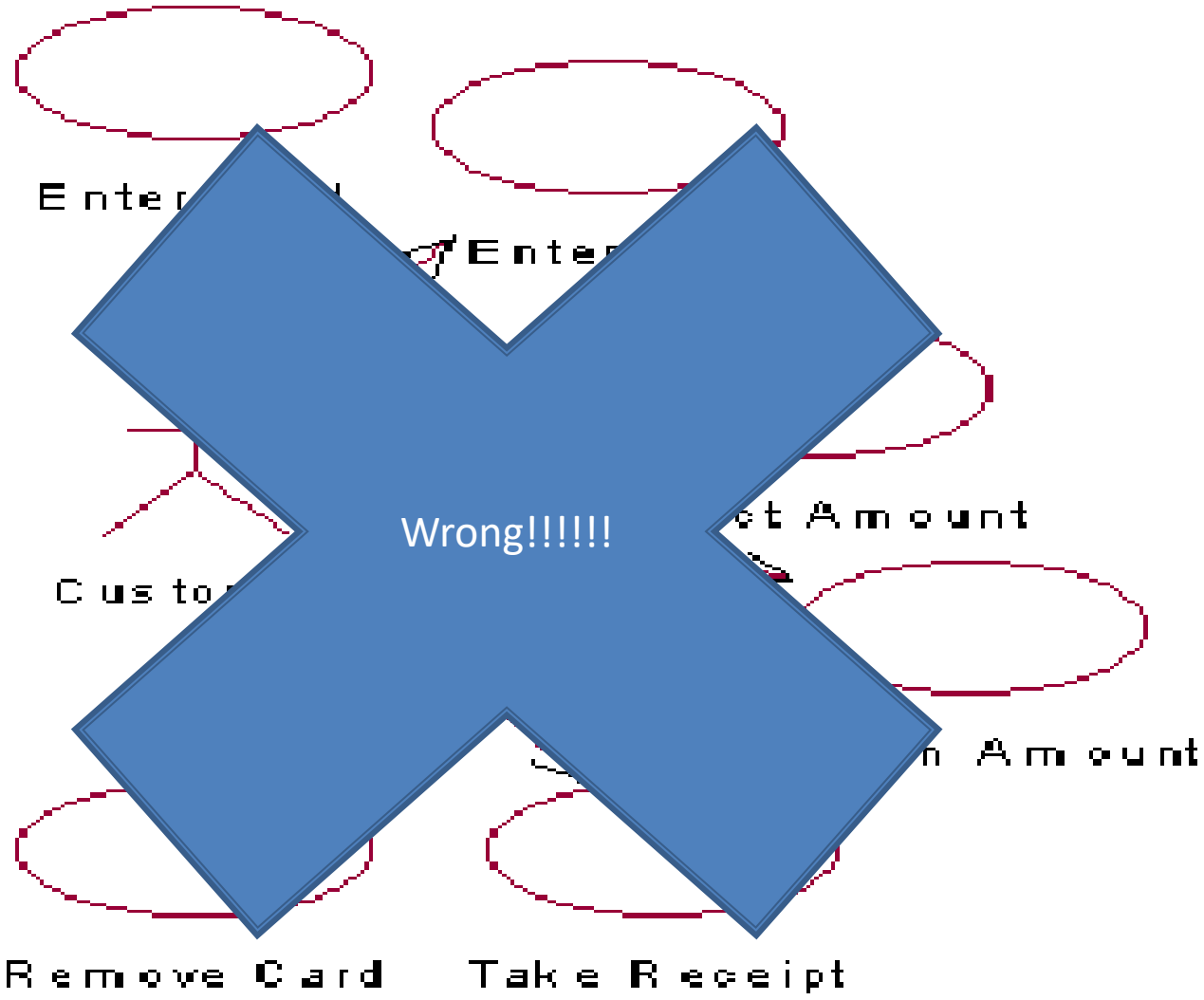
A Use Case should satisfy a goal for the actor

Is **take receipt. the goal for our customer?**

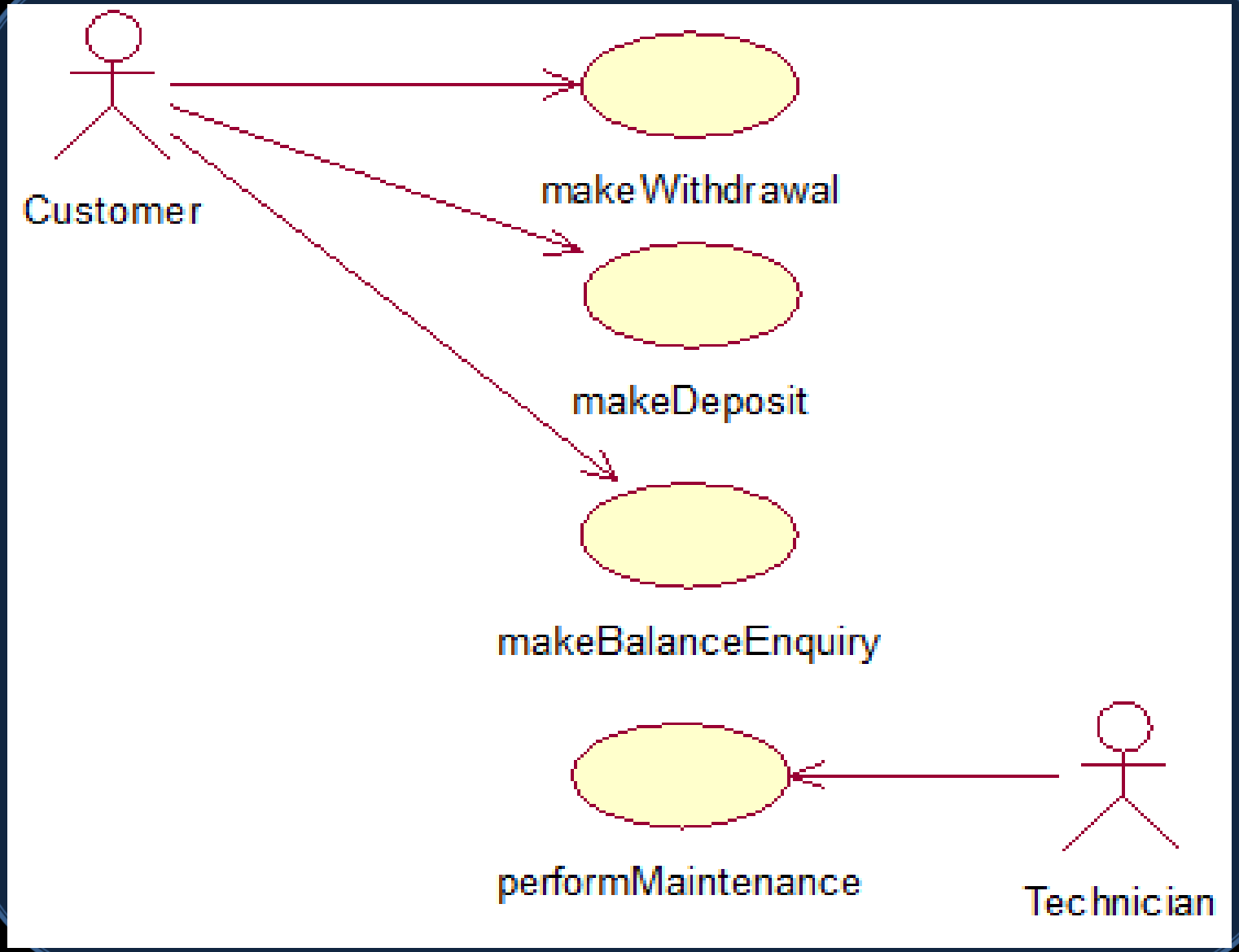
Well, not really. It wouldn't be the end of the world if the receipt wasn.'t dispensed.

Apply the rule to the other Use Cases, and you.ll find that really, none of them describe the goal of the user.
The goal of the user is to **withdraw money, and that** should be the use case!

ATM MACHINE



ATM MACHINE


Customer
makeWithdrawal
makeDeposit
makeBalanceEnquiry
performMaintenance
Technician

Video Shop

Any one who wishes to rent a video from the shop must first be registered as a member. Members may rent out videos provided that they are old enough, for this reason the Sales Assistant needs to record each member's date of birth when they join. It also records each member's address and phone number, so that they can trace them in the event of a video not being returned.

Members can reserve a video provided a copy is available for the day they require it and that they are genuine members. All video reservations are recorded in the company's video database so that the status of each copy can be monitored. Reserved videos must be collected before 7pm.

To rent a video, members must present their Membership Card. Each video attracts a rental charge depending on its popularity and how recently it has been released. All Video rented out must be returned by 6pm the following day otherwise members will be charged an extra day.

Staff need to update stock records with new titles or extra copies when deliveries are received from Suppliers and deleting records of copies which are lost or damaged on return.

Video Shop

Any one who wishes to rent a video from the shop must first **be registered** as a member. Members may **rent out videos** provided that they are old enough, for this reason the Sales Assistant needs to record each member's date of birth when they join. It also records each member's address and phone number, so that they can trace them in the event of a video not being returned.

Members can **reserve a video** provided a copy is available for the day they require it and that they are genuine members. All video reservations are recorded in the company's video database so that the status of each copy can be monitored. Reserved videos must be collected before 7pm.

To **rent a video**, members must present their Membership Card. Each video attracts a rental charge depending on its popularity and how recently it has been released. All Video rented out must be returned by 6pm the following day otherwise members will be charged an extra day.

Staff need to **update stock records** with new titles or extra copies when deliveries are received from Suppliers and deleting records of copies which are lost or damaged on return.

The UML

"UML is a language used to **specify, visualize** and **document** the artifacts of an object-oriented system under development.

It represents the unification of the **Booch, OMT**, and **Objectory** notations, as well as the best ideas from a number of other methodologists

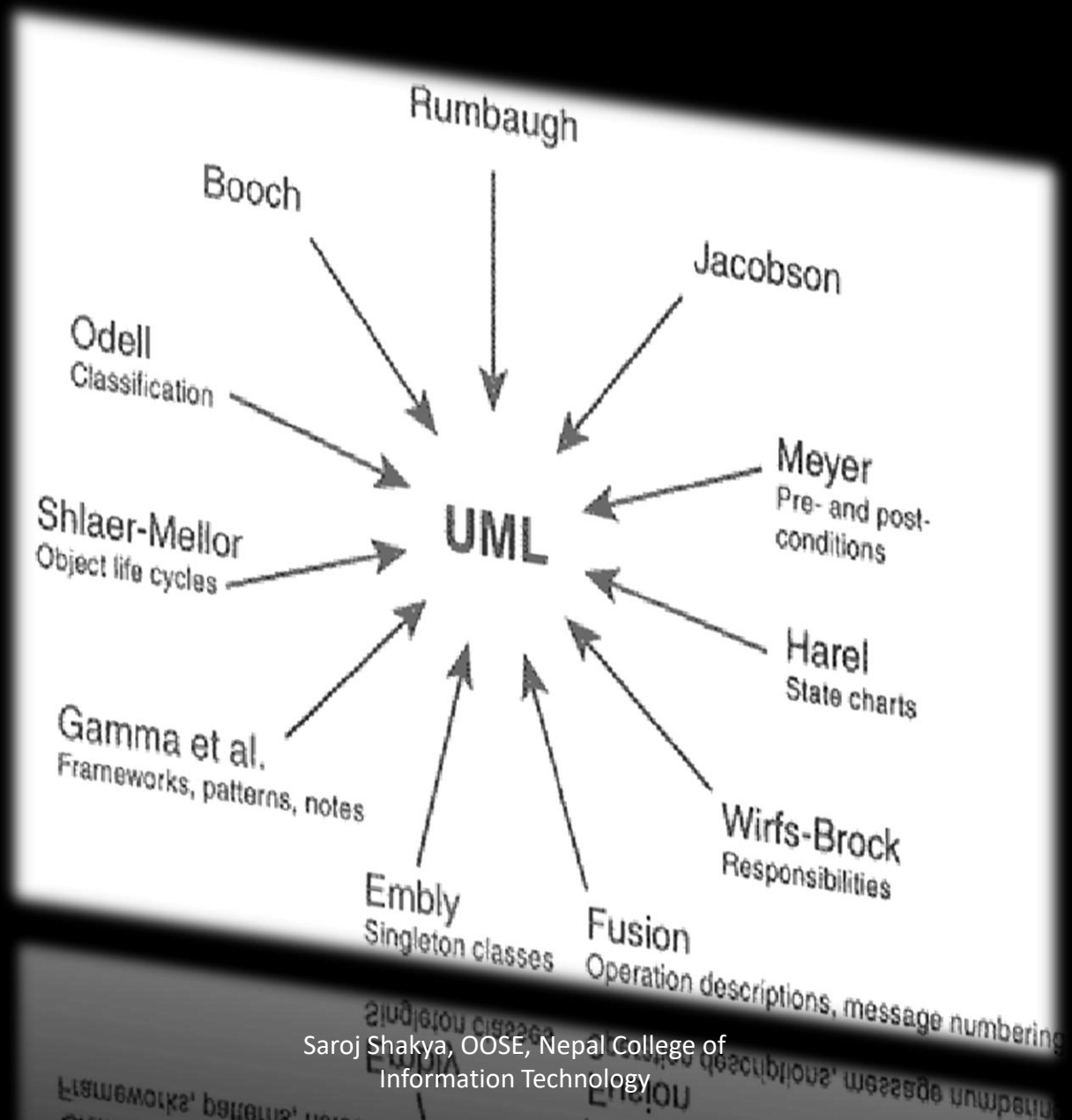
Provides the basis for a **de facto** standard in the domain of object-oriented analysis and design founded on a wide base of user experience.

OOAD

OOA
-Use Case (Essential)
-Domain Modeling (Object Identification)
-System Sequence Diagram (Message Identification)

OOD (Elaboration)
-Use Case (Real)
-Interaction Diagram (Message Design)
-Design Class Diagram (Responsibility Assignment)

The UML


Rumbaugh
Booch
Jacobson
Odell
Classification
Meyer
Pre- and post-conditions
Shlaer-Mellor
Object life cycles
UML
Harel
State charts
Gamma et al.
Frameworks, patterns, notes
Wirfs-Brock
Responsibilities
Embly
Singleton classes
Fusion
Operation descriptions, message numbering

Unified because it …

– **Combines** main preceding OO methods (**Booch by Grady Booch, OMT by Jim Rumbaugh and OOSE by Ivar Jacobson***)*

Modelling because it is …

– Primarily used for **visually modeling** systems. Many system views are supported by appropriate models

Language because …

– It **offers a syntax** through which to express modeled knowledge

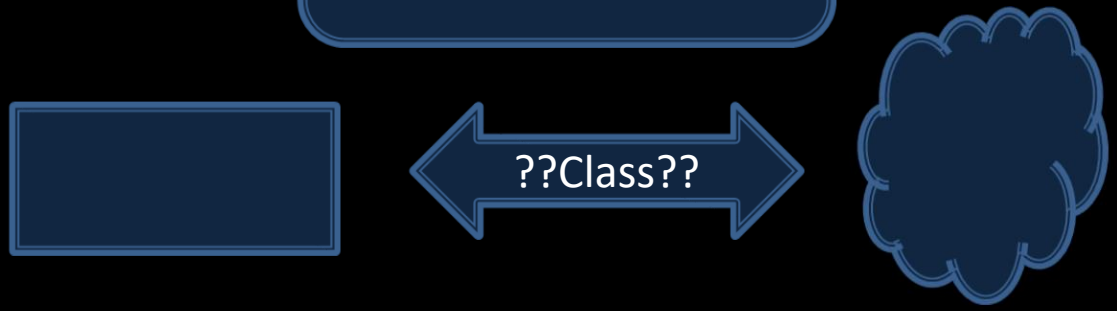
and…What UML is not!

- A **visual programming language** or environment
- A **database specification** tool
- A **development process** (i.e. an SDLC)
- A **panacea**
- A **quality guarantee**

History of the UML

During the 1990s **many different methodologies** their own set of notations

Method War



Multiplicity indicator in OMT

Aggregation symbol in Booch

History of the UML

3 most popular methods: OMT (Rumbaugh), Booch, and OOSE (Jacobson).

OMT
Strong in analysis
Weaker in the design area.

Booch
Strong in design
Weaker in analysis.

Jacobson
Strong in behavior analysis
Weaker in the other areas.

Why UML

Helps to reduce cost and time-to-market.

Helps managing a complex project architecture.

Helps to convey ideas between developers\designers\etc.

The UML

1970 – Object-oriented modeling languages began to appear.

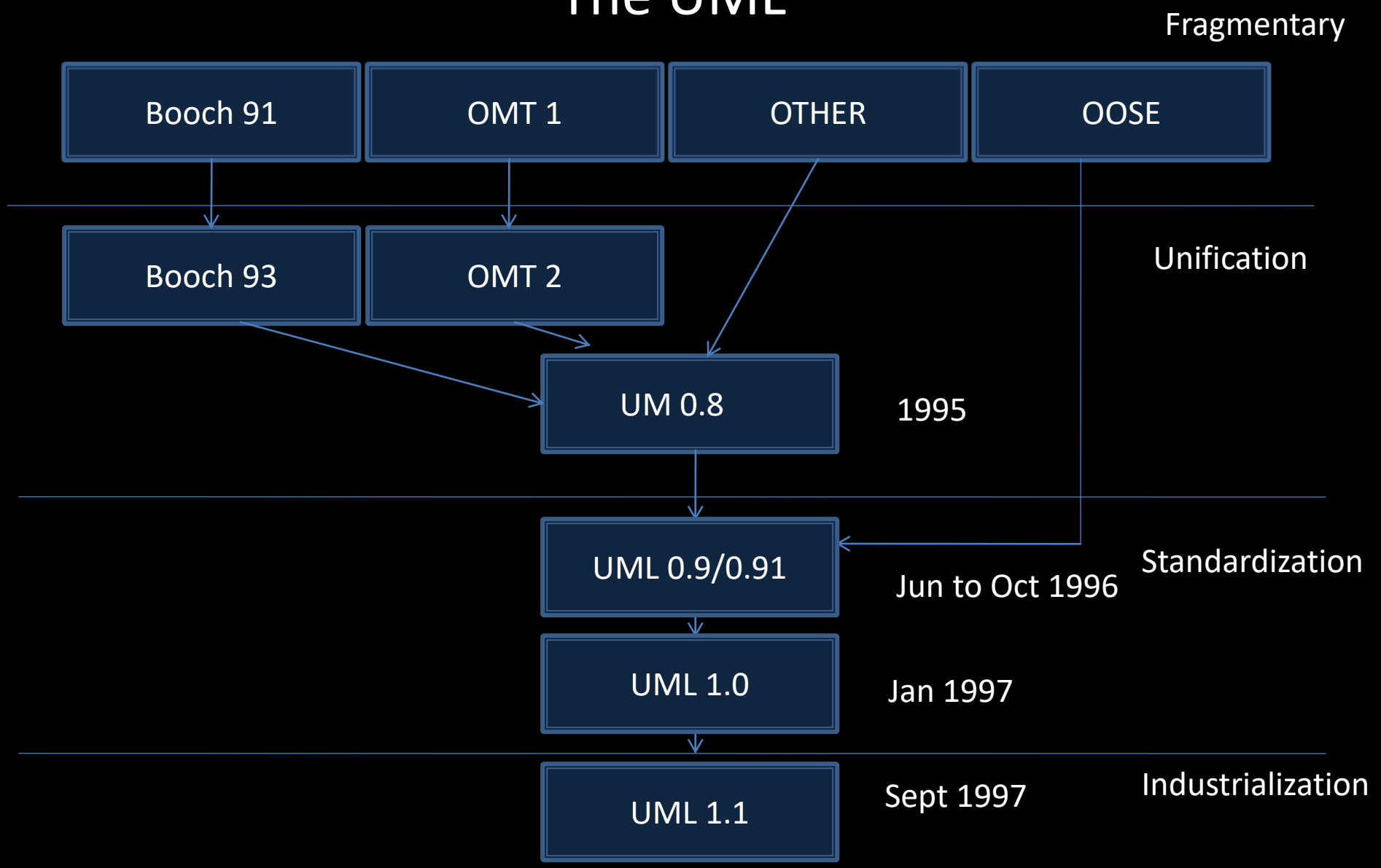
1996 – Release of UML 0.9 by by Grady Booch, Jim Rumbaugh of Rational Software Corporation, Ivar Jacobson of Objectory company.

1996 – Release of UML 1.0 by Digital Equipment, HP, ILogix, IntelliCorp, IBM, ICON, MCI, Microsoft, Oracle, Rational, TI and Unisys.

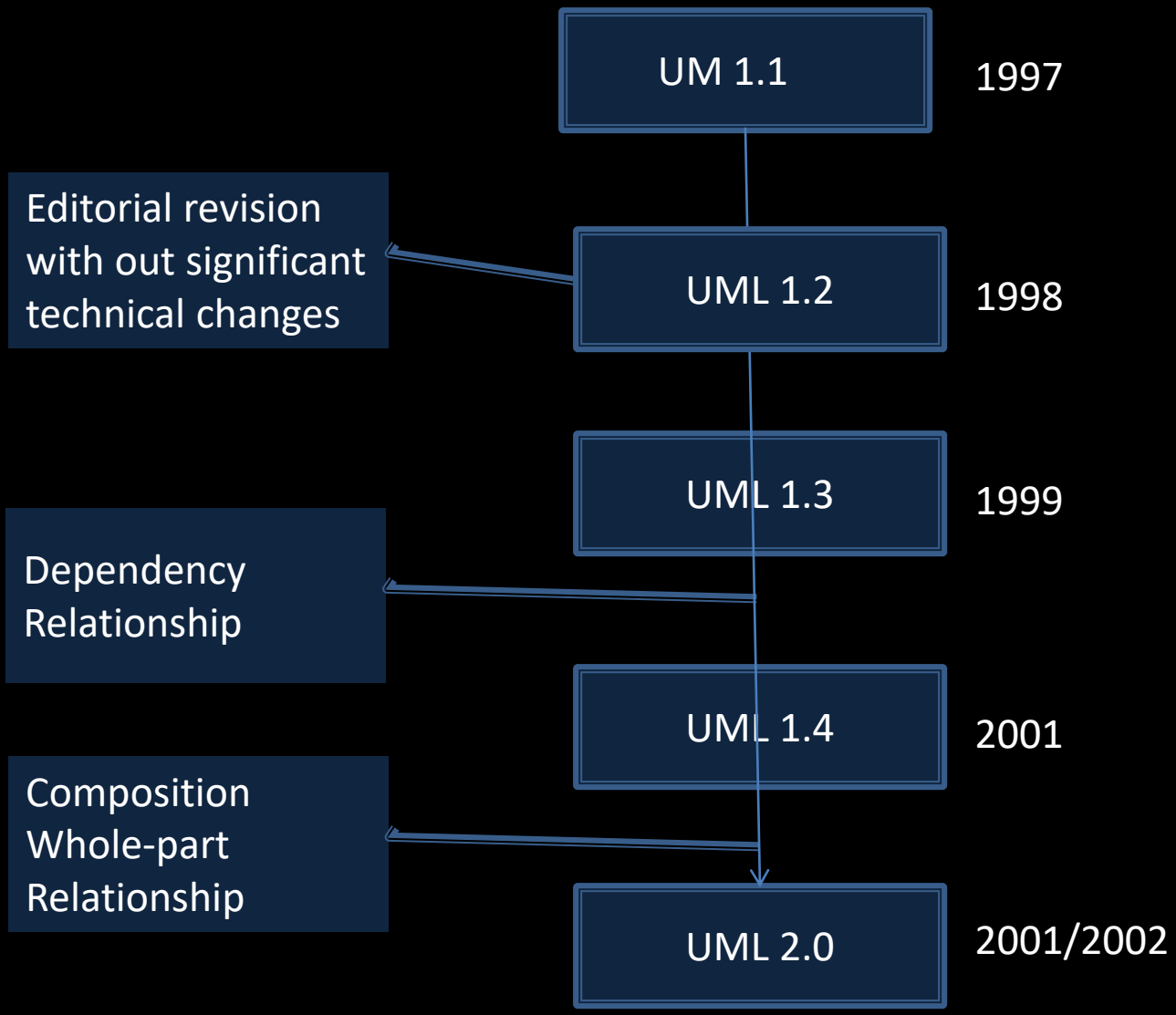
1997 – Release of UML 1.1 by IBM, ObjecTime, Platinum, Ptech, Taskon, Reich and Softeam

2001 – Work on UML 2.0 specifications.

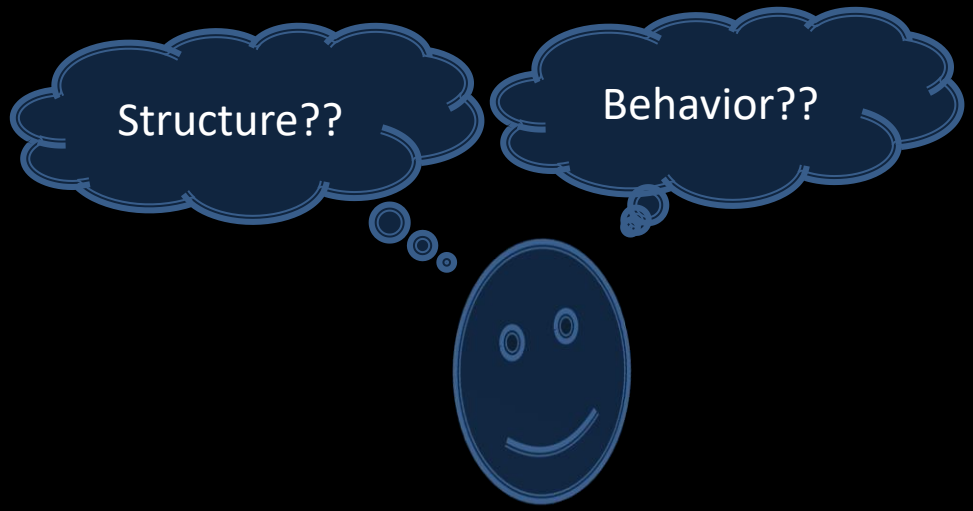
The UML


Fragmentary
Booch 91
OMT 1
OTHER
OOSE
Booch 93
OMT 2
Unification
UM 0.8
1995
UML 0.9/0.91
Jun to Oct 1996
Standardization
UML 1.0
Jan 1997
UML 1.1
Sept 1997
Industrialization

The UML


UM 1.1
1997
Editorial revision with out significant technical changes
UML 1.2
1998
UML 1.3
1999
Dependency Relationship
UML 1.4
2001
Composition Whole-part Relationship
UML 2.0
2001/2002

The UML



Solution:
Create **several diagrams**, each focused on one view.
Each diagram provides a view into the elements that make up the system.

The UML
Terms and Concepts

A **system** is a collection of subsystems organized to accomplish a purpose and described by a set of models, possibly from different viewpoints.

A **subsystem** is a grouping of elements, of which some constitute a specification of the behavior offered by the other contained **elements**.

A **model** is a semantically closed abstraction of a system, meaning that it represents a complete and self-consistent simplification of reality, created in order to better understand the system.

A **view** is a projection into the organization and structure of a system's model, focused on one aspect of that system.

A **diagram** is the graphical presentation of a set of elements, most often rendered as a connected graph of vertices (things) and arcs (relationships).

The UML
Terms and Concepts

Activity 1

Think of some examples of systems.

……

……

……

A subsystem is *a smaller system operating within some larger system and supporting the over all objective of the larger system.*

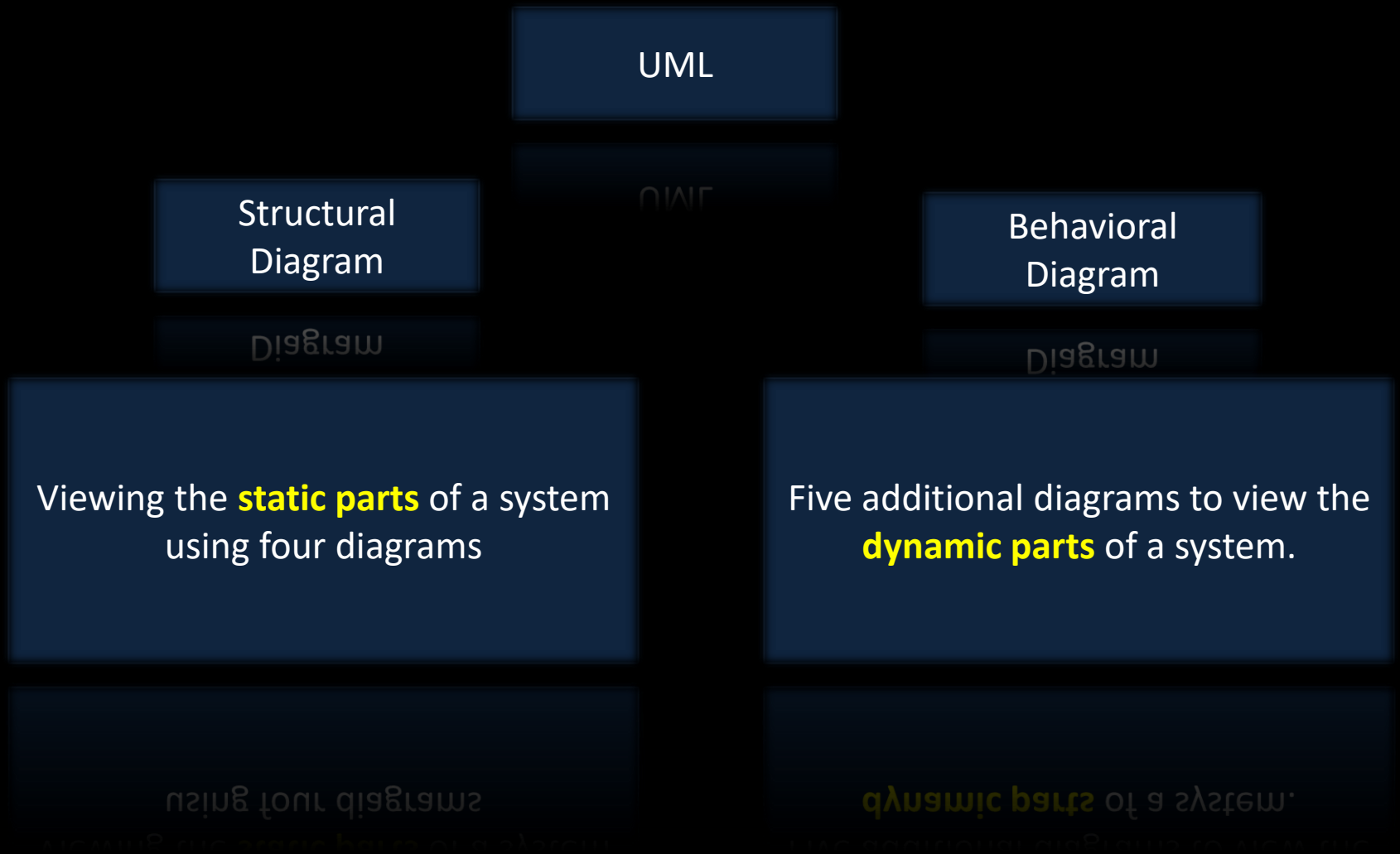
Activity 2

Think of sub systems that operate inside the systems discussed in Exercise 1.

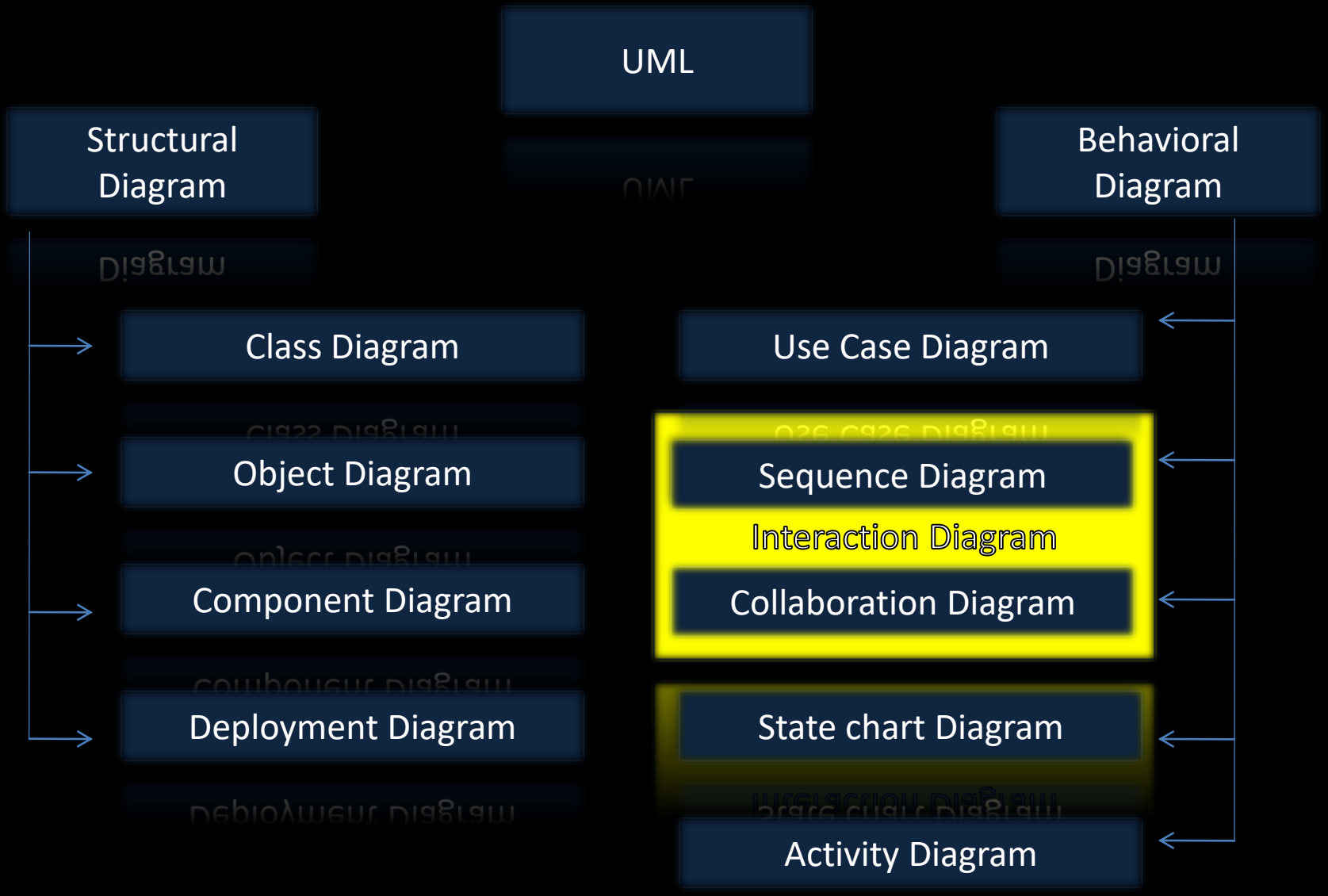
……

……

Diagrams in The UML



Diagrams in The UML



A. Structural Diagrams

Static aspects of a house:
walls, doors, windows, pipes, wires, and vents,

Static aspects of a software system : classes, interfaces, components, and nodes.

Used to describe the **building blocks** of the system
– features **that do not change** with time.

These diagrams answer the question – **Whats there?**

1. Class diagram

Class diagrams commonly contain the following things:

- Classes
- Interfaces
- Collaborations
- Dependency, generalization, and association relationships

Class diagrams are static – display **what interacts** but **not what happens when interaction occurs.**
May contain notes and constraints.

May also contain **packages** or **subsystems**, both of which are used to group elements of your model into larger chunks

1. Class diagram

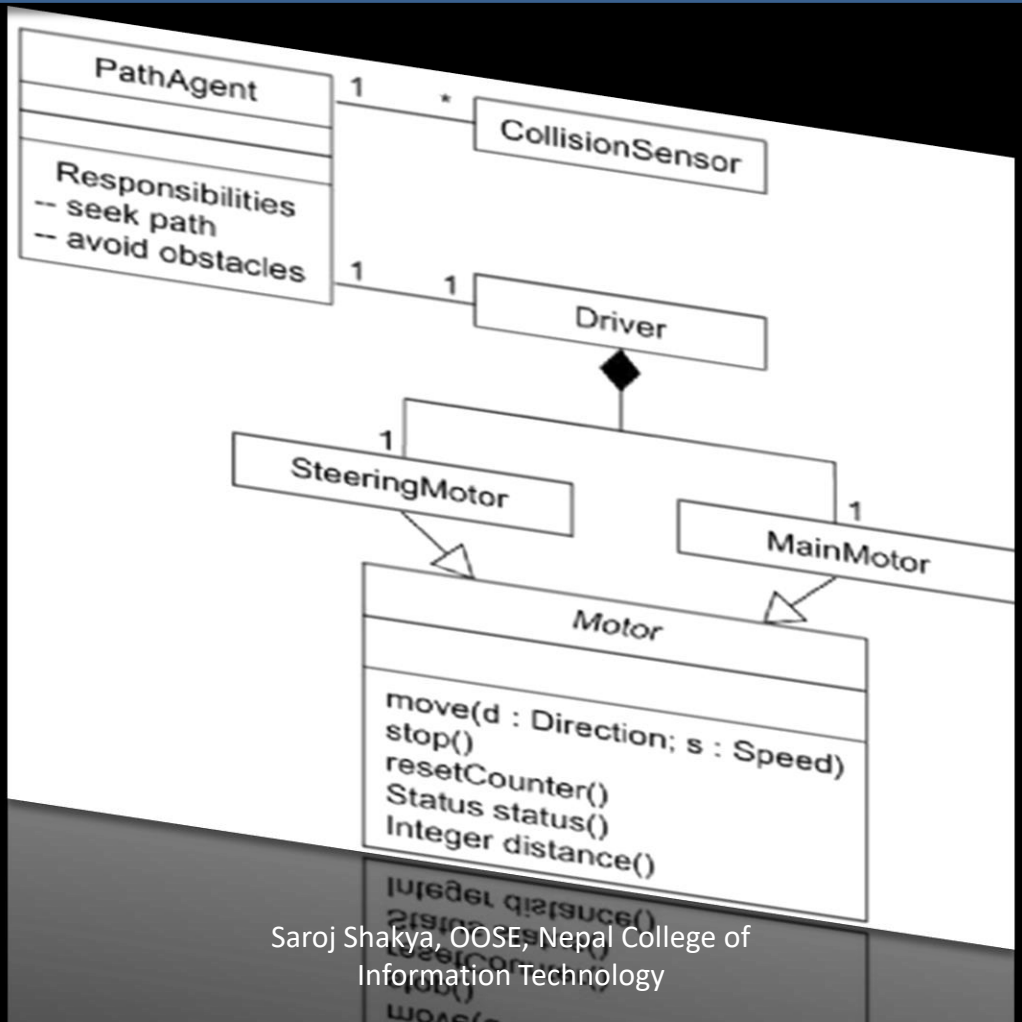
1. To model the **vocabulary of a system**

Making a decision about which abstractions are apart of the system under consideration and which fall outside its boundaries.

Used to specify these abstractions and their responsibilities.

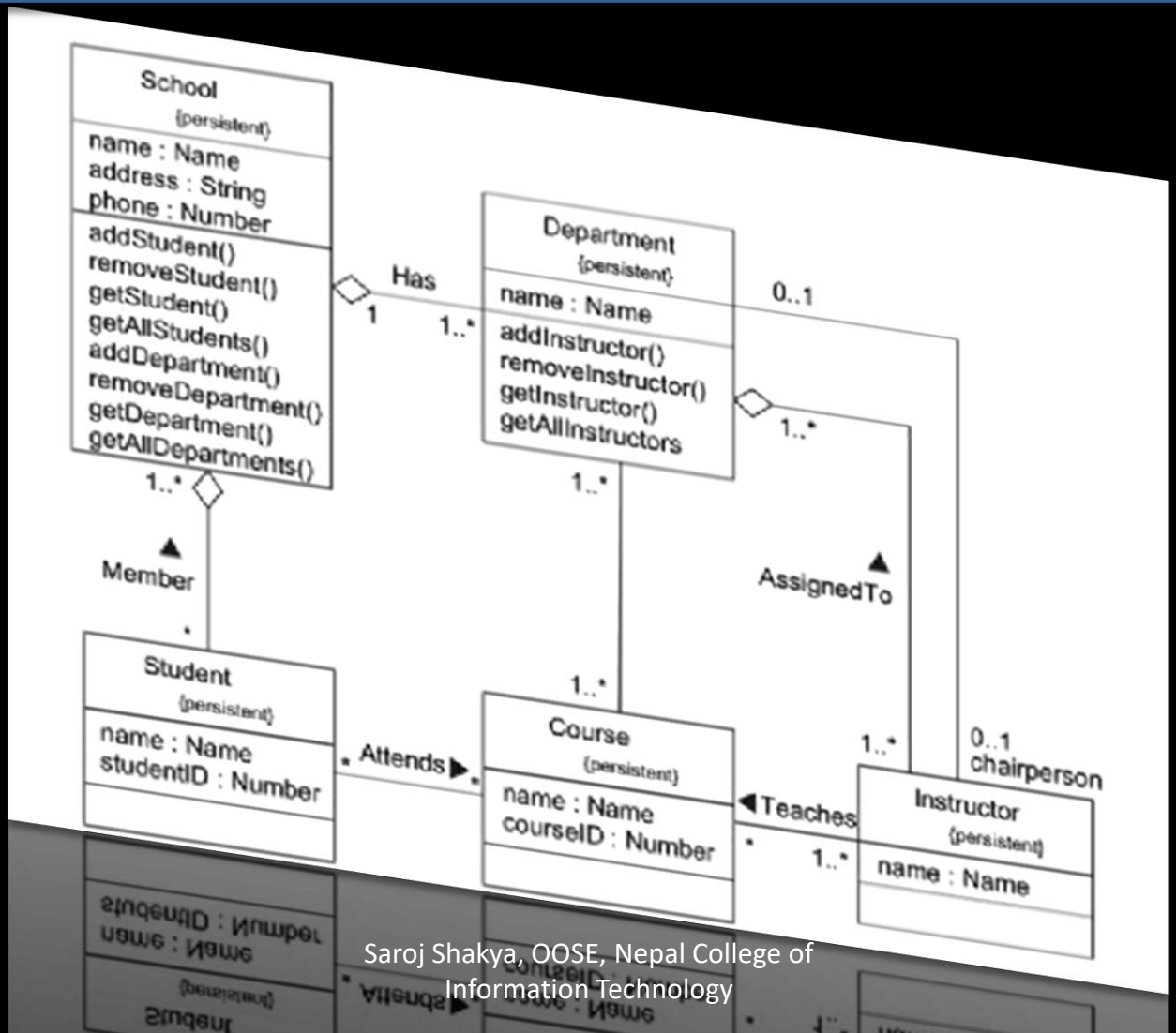
1. Class diagram

2. To model simple **collaborations**
A collaboration is a society of classes, interfaces, and other elements that work together to provide some cooperative behavior



1. Class diagram

3. To model a **logical database schema**
The blueprint for the conceptual design of a database.



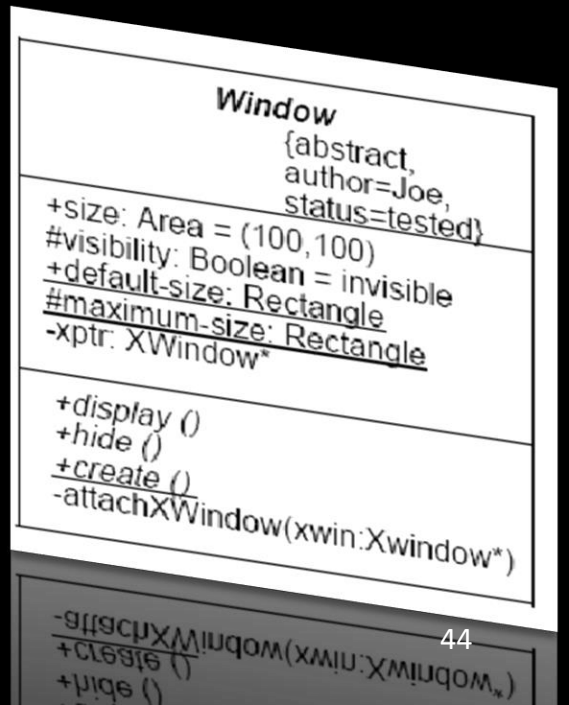
1. Class diagram

Classes are represented by a rectangle divided to three parts: class name, attributes and operations.

Attributes are written as:
visibility name [multiplicity] :type-expression =initial-value

Operations are written as:
visibility name (parameter-list) :return type-expression

Visibility is written as:
+public
#protected
-private



Class Diagram Relationships

Class Diagrams relationships:

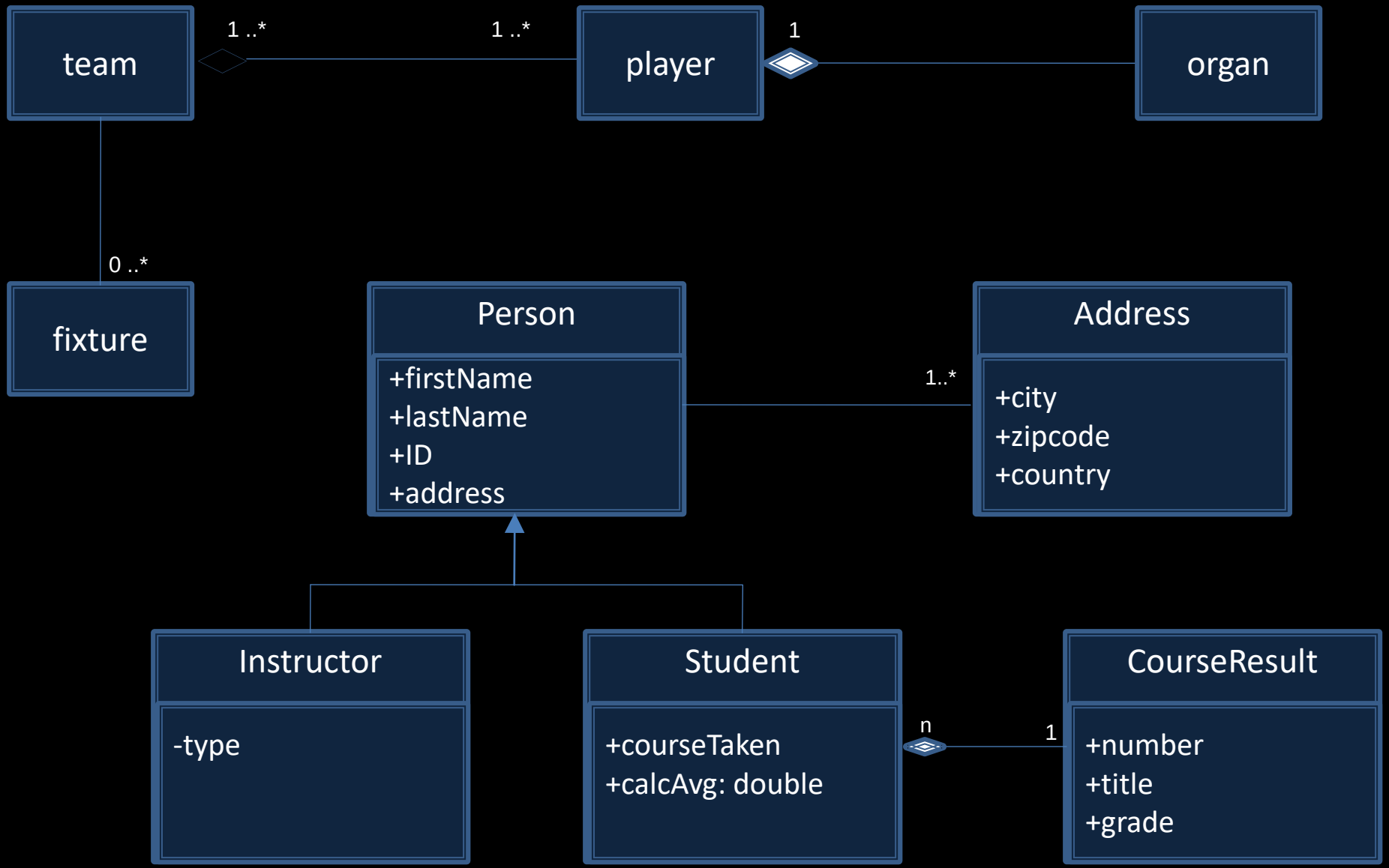
Association –
Two classes are associated if one class has to know about the other. (uses)

Aggregation –
An association in which one class belongs to a collection in the other.(whole part)

Generalization –
An inheritance link indicating one class is a base class of the other.

Dependency –
A labeled dependency between classes (such as friend, classes)

The UML


team
1 ..*
1 ..*
player
1
organ
0 ..*
fixture
Person
+firstName
+lastName
+ID
+address
1..*
Address
+city
+zipcode
+country
Instructor
-type
Student
+courseTaken
+calcAvg: double
n
1
CourseResult
+number
+title
+grade

Case Study

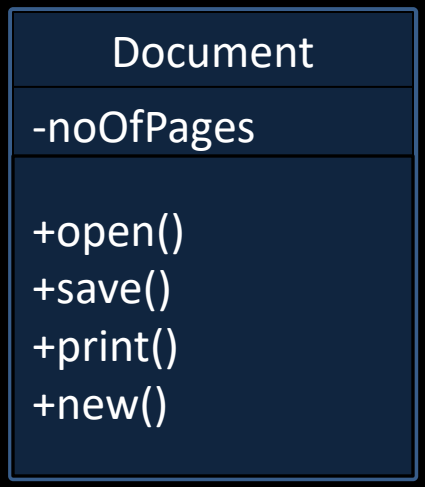
Describing the use of a word processor

A user can *open a new or existing document. Text is entered through a keyboard. A document is made up* of several pages and each page is made up of a header, body and footer. Date, time and page number may *be added to header or footer. Document body is* made up of sentences, which are themselves made up of words and punctuation characters. Words are made up of letters, digits and/or special characters. Pictures and tables may *be inserted into the* document body. Tables are made up of rows and columns. Users can *save or print documents.*

Case Study

Describing the use of a word processor

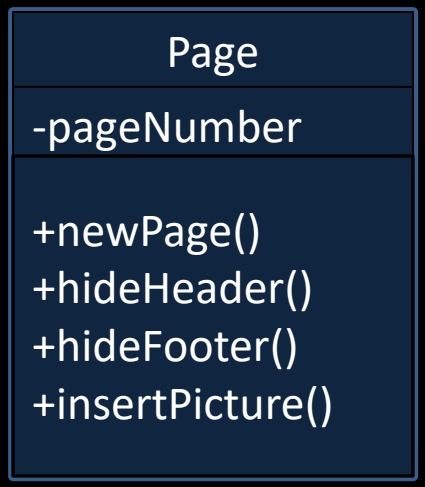
A user can *open a new or existing document*. *Text is entered through a keyboard.* **A document is made up of several pages** and each page is made up of a header, body and footer. Date, time and page number may *be added to header or footer. Document body is* made up of sentences, which are themselves made up of words and punctuation characters. Words are made up of letters, digits and/or special characters. Pictures and tables may *be inserted into the* document body. Tables are made up of rows and columns. **Users can save or print documents.**



Case Study

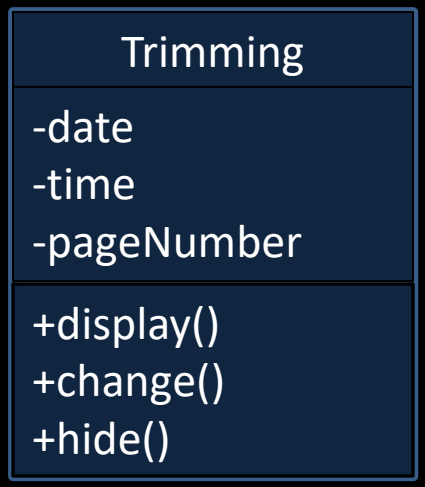
Describing the use of a word processor

A user can *open a new or existing document. Text is entered through a keyboard.* **A document is made up of several pages and each page is made up of a header, body and footer.** Date, time and page number may *be added to header or footer. Document body is* made up of sentences, which are themselves made up of words and punctuation characters. Words are made up of letters, digits and/or special characters. Pictures and tables may *be inserted into the* document body. Tables are made up of rows and columns. Users can *save or print documents.*

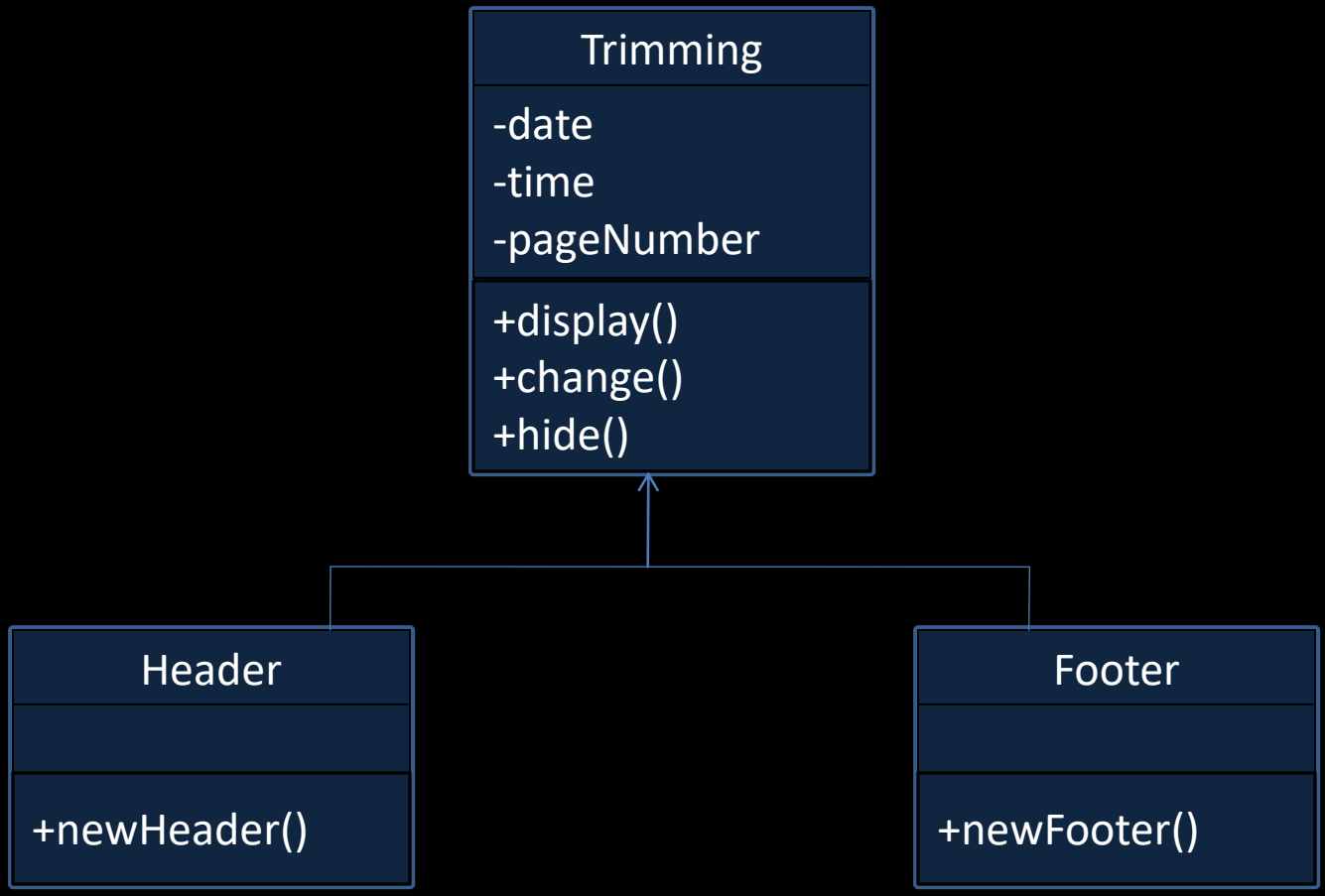


Case Study

Describing the use of a word processor
A user can *open a new or existing document. Text is entered through a keyboard. A document is made up* of several pages and each page is made up of a header, body and footer. **Date, time and page number may be added to header or footer.** *Document body is* made up of sentences, which are themselves made up of words and punctuation characters. Words are made up of letters, digits and/or special characters. Pictures and tables may *be inserted into the* document body. Tables are made up of rows and columns. Users can *save or print documents.*

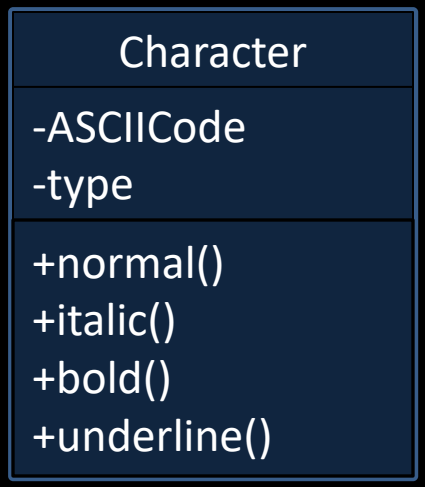


Case Study

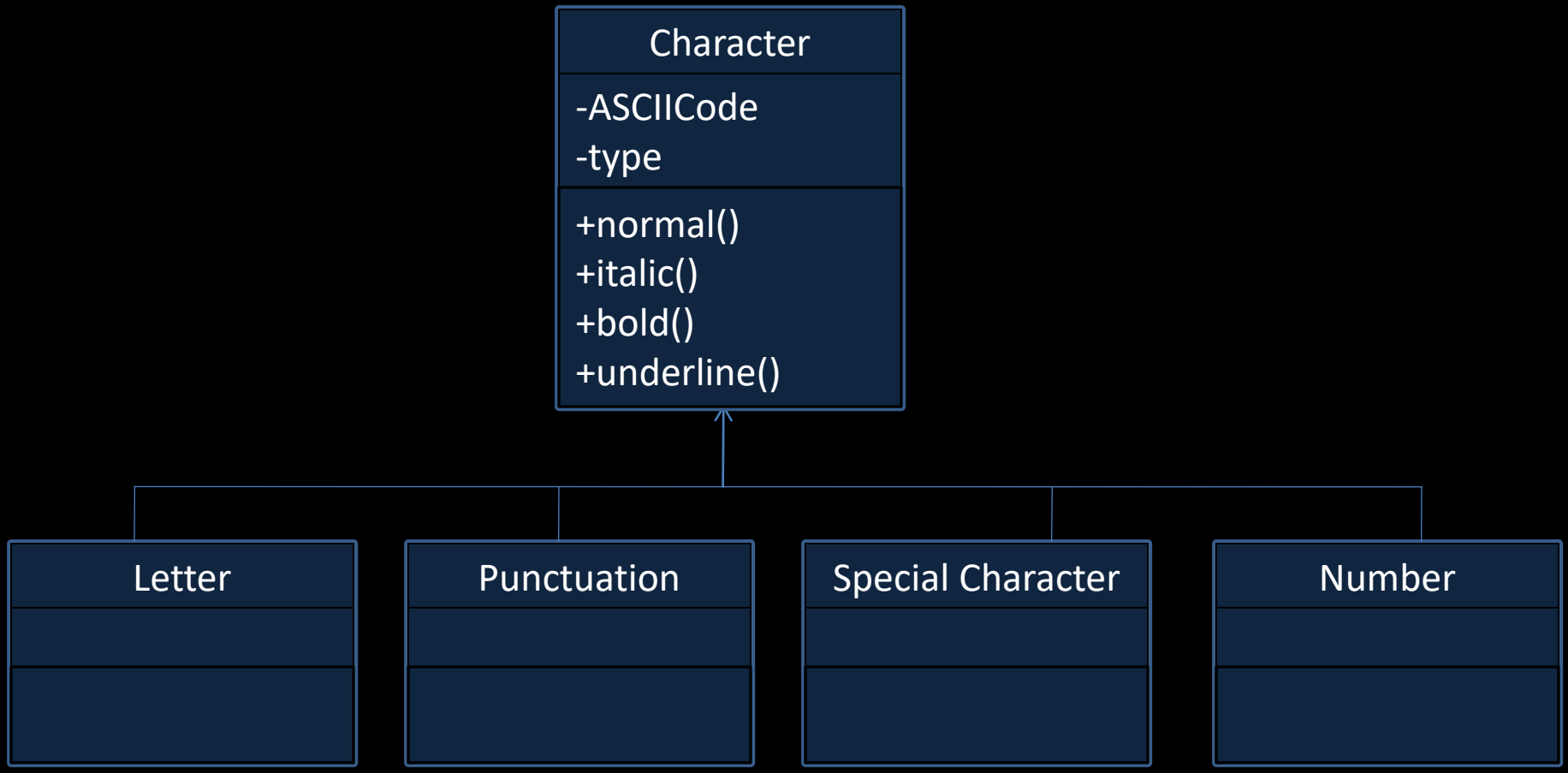

Trimming
-date
-time
-pageNumber
+display()
+change()
+hide()
Header
+newHeader()
Footer
+newFooter()

Case Study

Describing the use of a word processor
A user can *open a new or existing document. Text is entered through a keyboard. A document is made up* of several pages and each page is made up of a header, body and footer. Date, time and page number may *be added to header or footer.* **Document body is made up of sentences, which are themselves made up of words and punctuation characters. Words are made up of letters, digits and/or special characters**. Pictures and tables may *be inserted into the* document body. Tables are made up of rows and columns. Users can *save or print documents.*



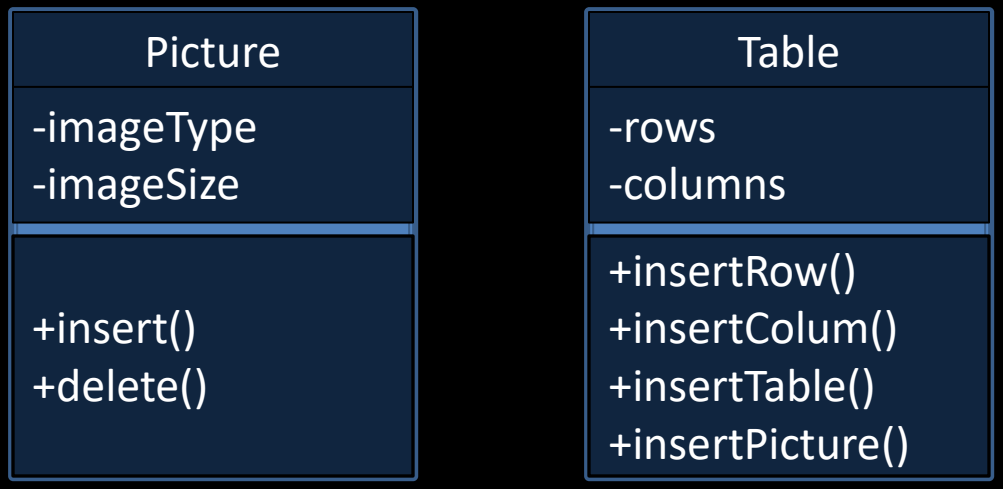
Case Study


Character
-ASCIICode
-type
+normal()
+italic()
+bold()
+underline()
Letter
Punctuation
Special Character
Number

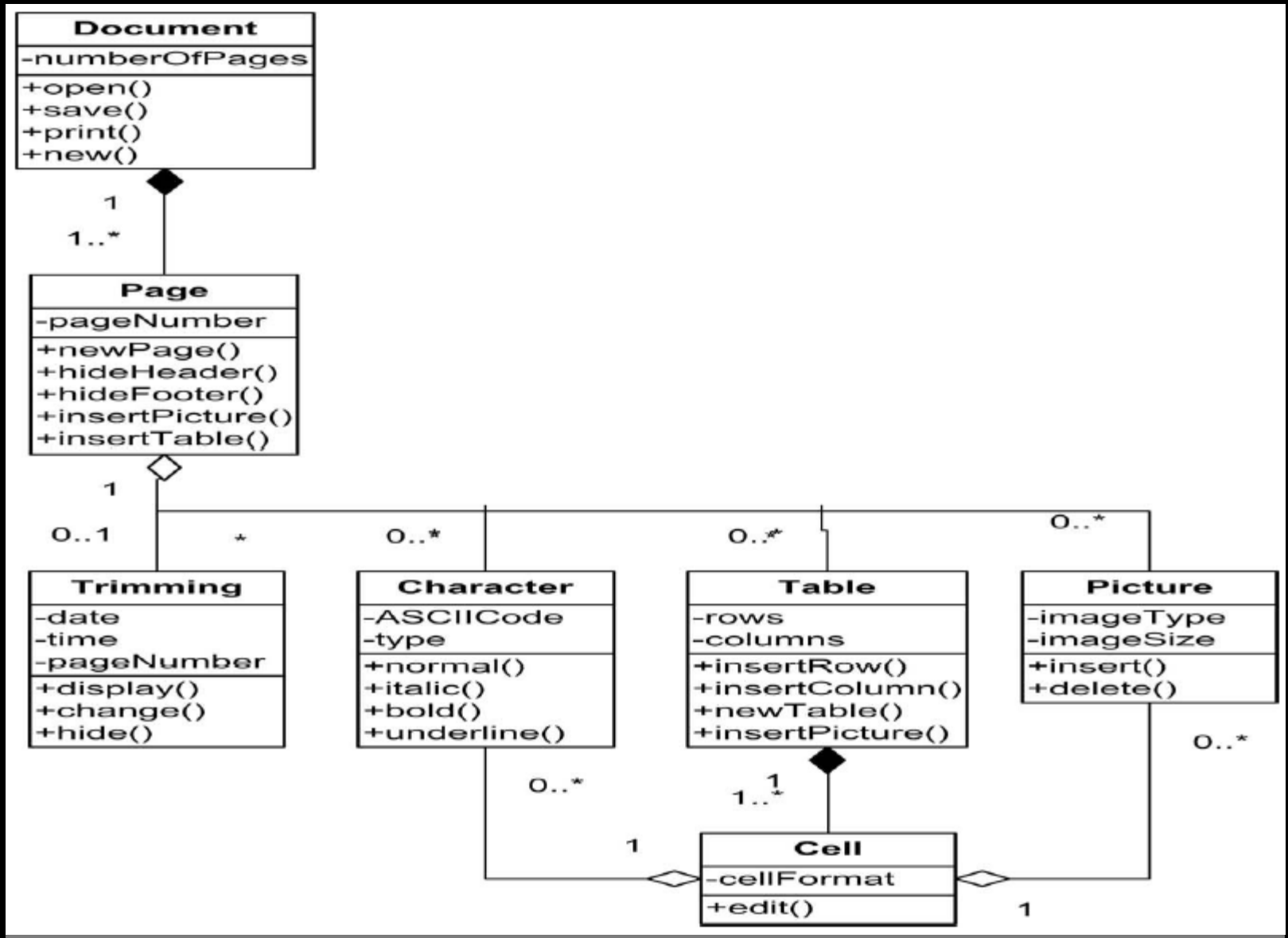
Case Study

Describing the use of a word processor

A user can open a new or existing document. Text is entered through a keyboard. A document is made up of several pages and each page is made up of a header, body and footer. Date, time and page number may be added to header or footer. Document body is made up of sentences, which are themselves made up of words and punctuation characters. Words are made up of letters, digits and/or special characters. **Pictures and tables may be inserted into the document body.** Tables are made up of rows and columns. Users can *save or print documents.*



The UML

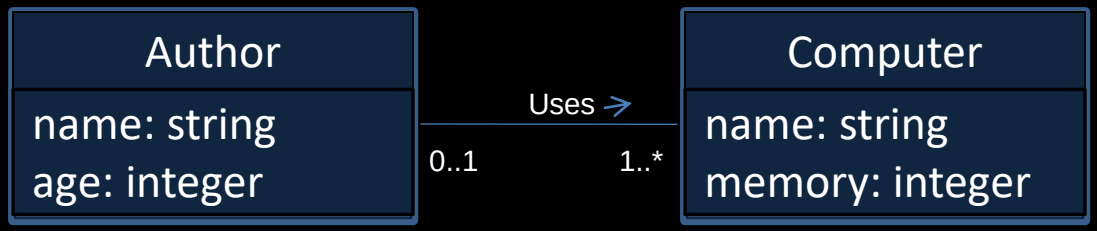

Document
-numberOfPages
+open()
+save()
+print()
+new()
1
1..*
Page
-pageNumber
+newPage()
+hideHeader()
+hideFooter()
+insertPicture()
+insertTable()
1
0..1
*
0..*
0..*
0..*
Trimming
-date
-time
-pageNumber
+display()
+change()
+hide()
Character
-ASCIICode
-type
+normal()
+italic()
+bold()
+underline()
Table
-rows
-columns
+insertRow()
+insertColumn()
+newTable()
+insertPicture()
Picture
-imageType
-imageSize
+insert()
+delete()
0..*
0..*
1..*
1
1
Cell
-cellFormat
+edit()
1

2. Object Diagram

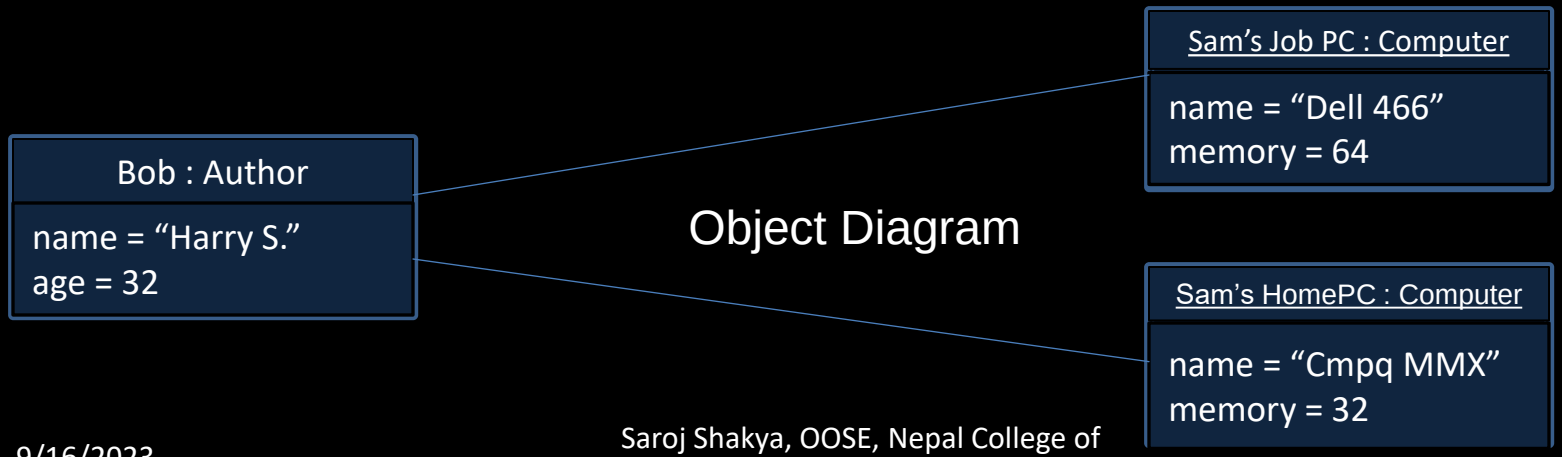
An *object diagram* shows a set of **objects and their relationships**.

Used to illustrate **data structures, the static snapshots of instances** of the things found in class diagrams.

Object diagrams address the static design view or static process view of a system just as do class diagrams, but from the **perspective of real** or **prototypical cases.**



Class Diagram



Object Diagram

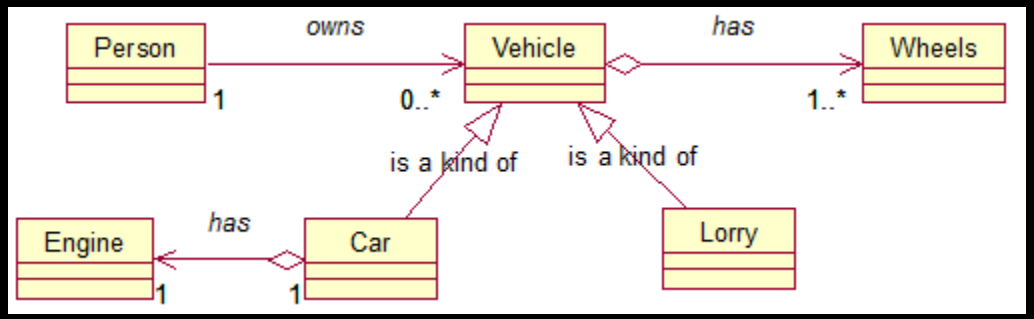
Exercise

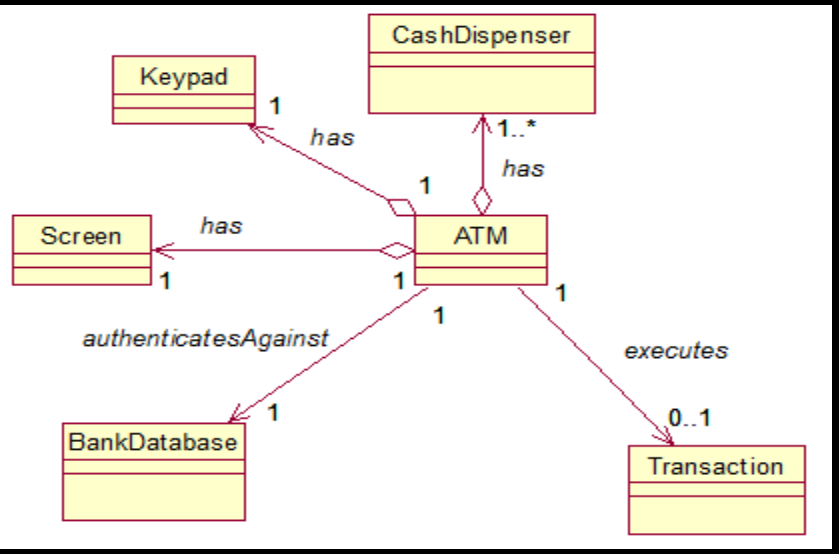
A person owns a vehicle. A vehicle is anything that has round wheels and can move around with load. A car is a kind of vehicle that has an engine and airbags where as a lorry is also a simple kind of vehicle

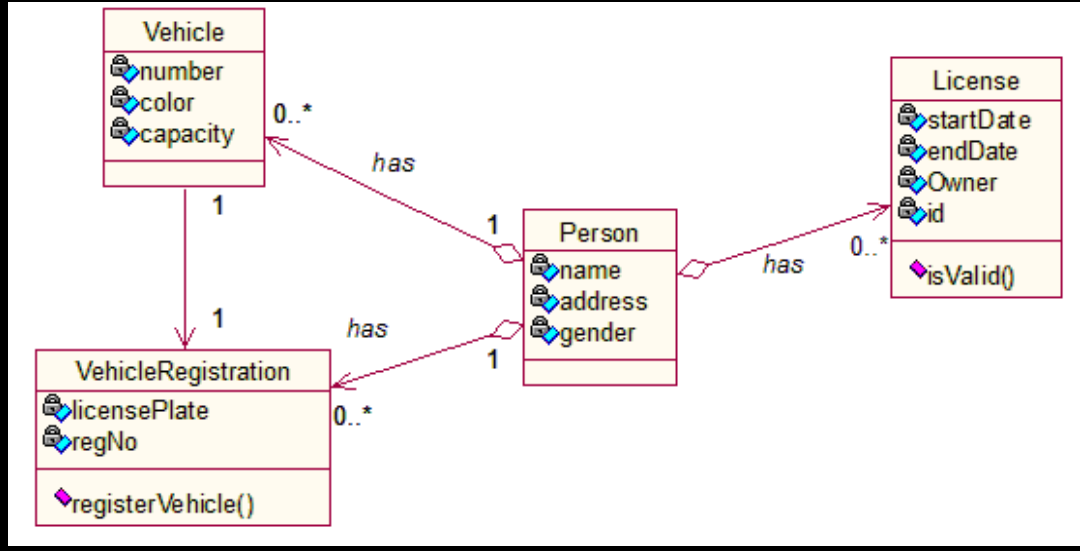
An ATM has a cash dispenser, screen, keypad. It authenticates user against bank database and executes transactions as per the user's choice.

A person can own many vehicles, a vehicle must be registered , one person must have vehicle registrations for each vehicle he owns. Every vehicle owner must have a license to use the vehicle and the license may be valid or invalid.

Exercise


Person
owns
Vehicle
has
Wheels
1
0..*
1..*
is a kind of
is a kind of
Engine
has
Car
Lorry
1
1


CashDispenser
Keypad
1
has
1..*
has
1
Screen
has
ATM
1
1
1
1
authenticatesAgainst
executes
1
0..1
BankDatabase
Transaction

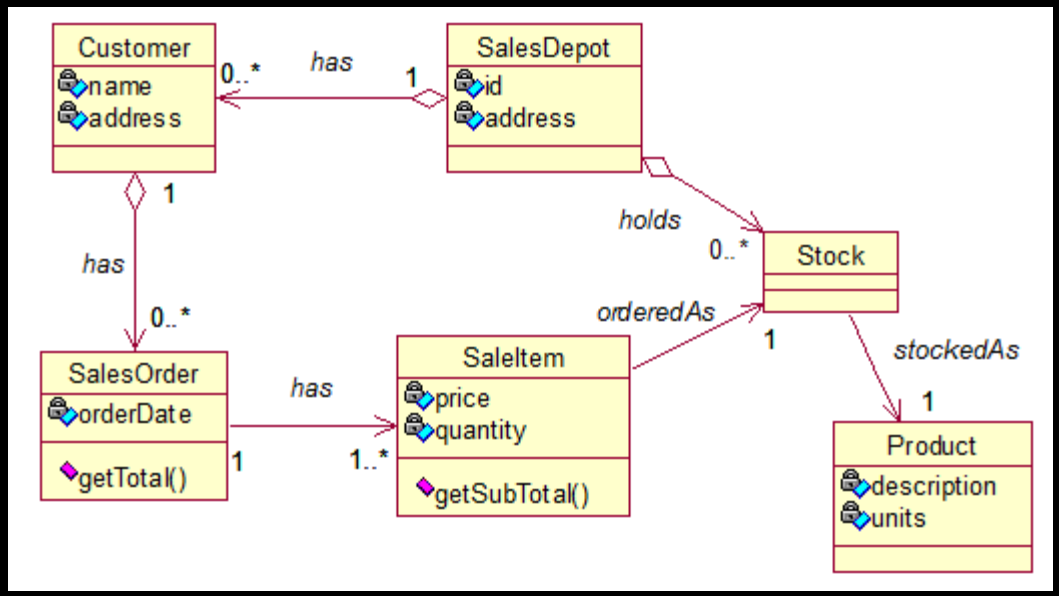

Vehicle
number
color
capacity
0..*
has
1
License
startDate
endDate
Owner
id
isValid()
1
Person
name
address
gender
has
0..*
1
has
1
VehicleRegistration
licensePlate
regNo
registerVehicle()
0..*

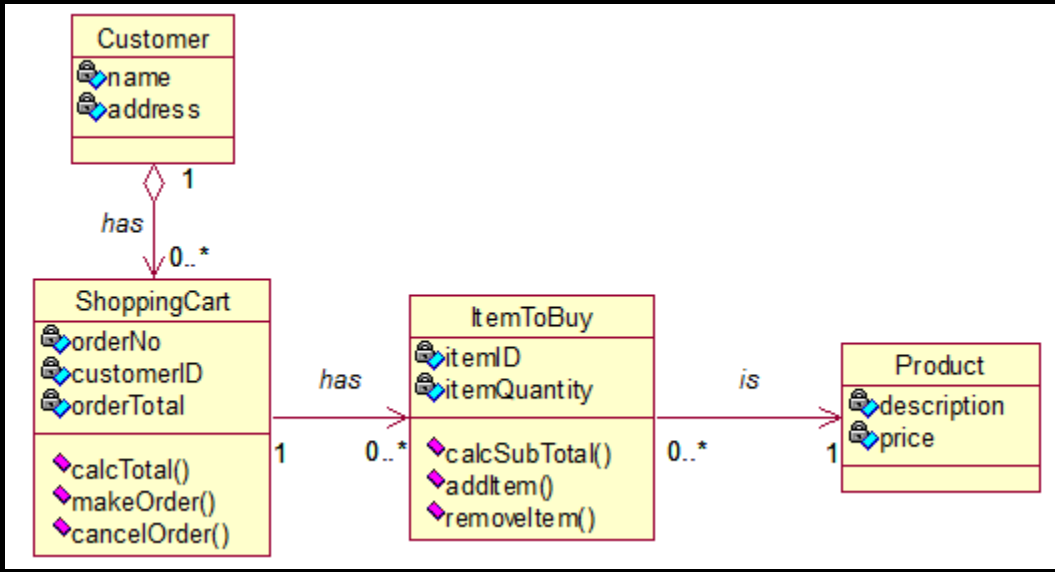
Exercise

A sales depot has customers and it holds stock. The stock is stocked as product with product description, units and are ordered as sale item. Customer places sales order that has order date , one sales order can have may sale items

A customer uses a shopping cart to do shopping online. Shopping cart contains information like order id , customer id, orderTotal, each shopping cart can have many items to buy and has information like order id, item quantity, item id and items can be added and deleted as well. Each item to buy is one specific type of item that has description, id and price.

Exercise


Customer
name
address
0..*
has
1
SalesDepot
id
address
1
has
0..*
holds
0..*
Stock
orderedAs
1
stockedAs
1
SalesOrder
orderDate
getTotal()
has
1
1..*
SaleItem
price
quantity
getSubTotal()
Product
description
units

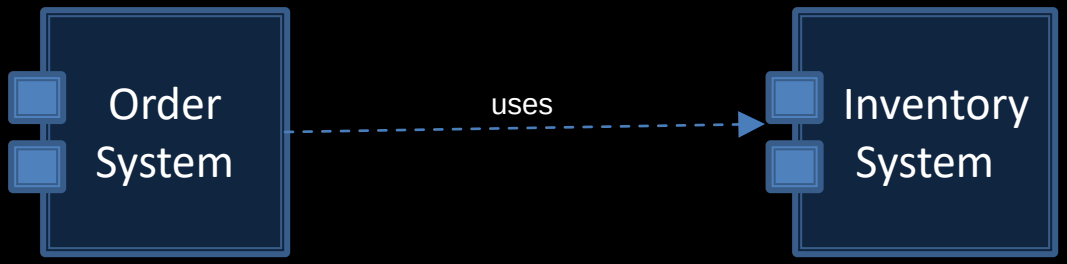

Customer
name
address
1
has
0..*
ShoppingCart
orderNo
customerID
orderTotal
calcTotal()
makeOrder()
cancelOrder()
1
has
0..*
ItemToBuy
itemID
itemQuantity
calcSubTotal()
addItem()
removeItem()
0..*
is
1
Product
description
price

3. Component Diagram

Shows a set of **components and their relationships.**

Used to illustrate the **static implementation view** of a system.

Component diagrams are related to class diagrams in that a component **typically maps to one or more classes, interfaces, or collaborations**.

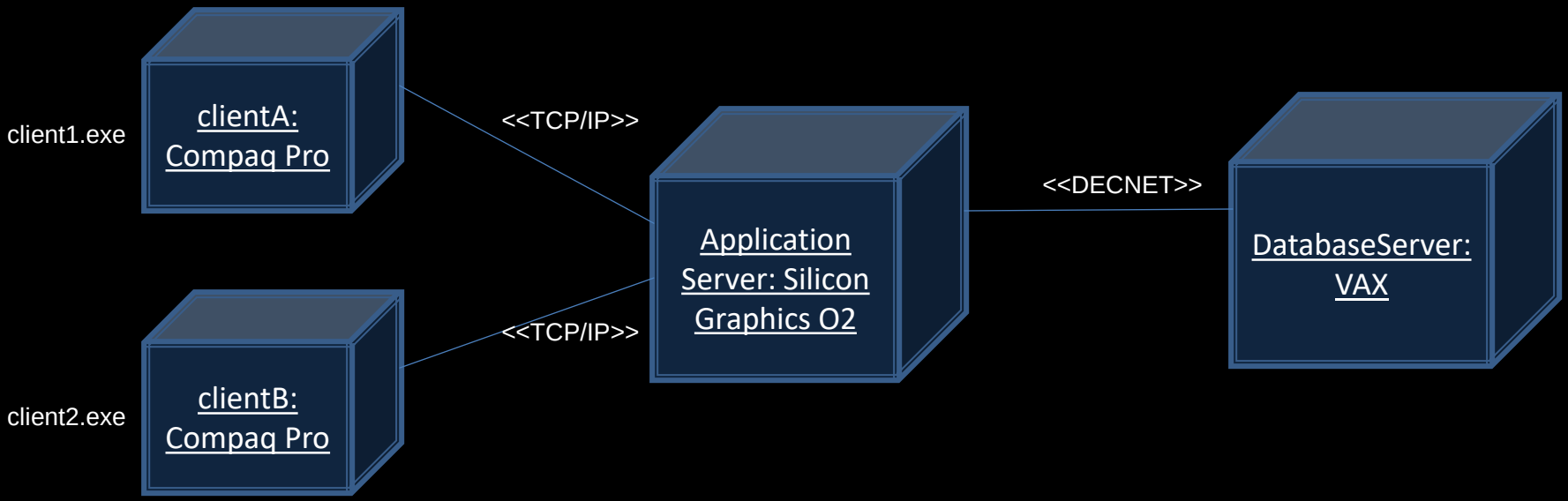


4. Deployment Diagram

A *deployment diagram* shows a **set of nodes and their relationships**.

Used to illustrate the static deployment **view of an architecture**.

Deployment diagrams are **related to component diagrams** in that a node **typically encloses one or more components.**



The UML

Assignment:

Draw a Class Diagram for a patient billing system.
Include only the attributes that would be appropriate for the system's context.

Patient (name, gender, address, ID, tel., DOB, blood type, occupation, pass-times, adverse habits, insurance carrier, dietary preferences)

Doctor (name, category, specialist, warrant No., preferred sport, address, tel., DOB, weekly income, VAT No.)

Insurance carrier (date of establishment, name, registration ID, company staff size, address, tel., contact person name)

B. Behavioral Diagrams

The UML's five behavioral diagrams are used to visualize, specify, construct, and document **the dynamic aspects of a system**.

Dynamic aspects of a system represent **its changing parts.**

Used to show **how** the system evolves over time (responds to requests , events etc)

Dynamic aspects of a house :

Airflow

Traffic through the rooms of a house

Dynamic aspects of a software system:

Flow of messages over time

Physical movement of components across a network.

B. Behavioral Diagrams

The UML's behavioral diagrams are roughly organized around the major ways you can model the dynamics of a system.

1. Use case diagram **Organizes the behaviors** of the system

2. Sequence diagram **Focused on the time ordering** of messages

3. Collaboration diagram **Focused on the structural organization** of objects that send and receive messages

4. Statechart diagram **Focused on the changing state of a system** driven by events

5. Activity diagram **Focused on the flow of control** from activity to activity

5. Use Case Diagram

A *use case diagram* shows **a set of use cases and actors** (a special kind of class) and their relationships.

Usecase diagrams are especially important in organizing and modeling the behaviors of a system.

Describes **what a system does** from the standpoint of an external observer.

Emphasis on what *a system does* **rather then how.**

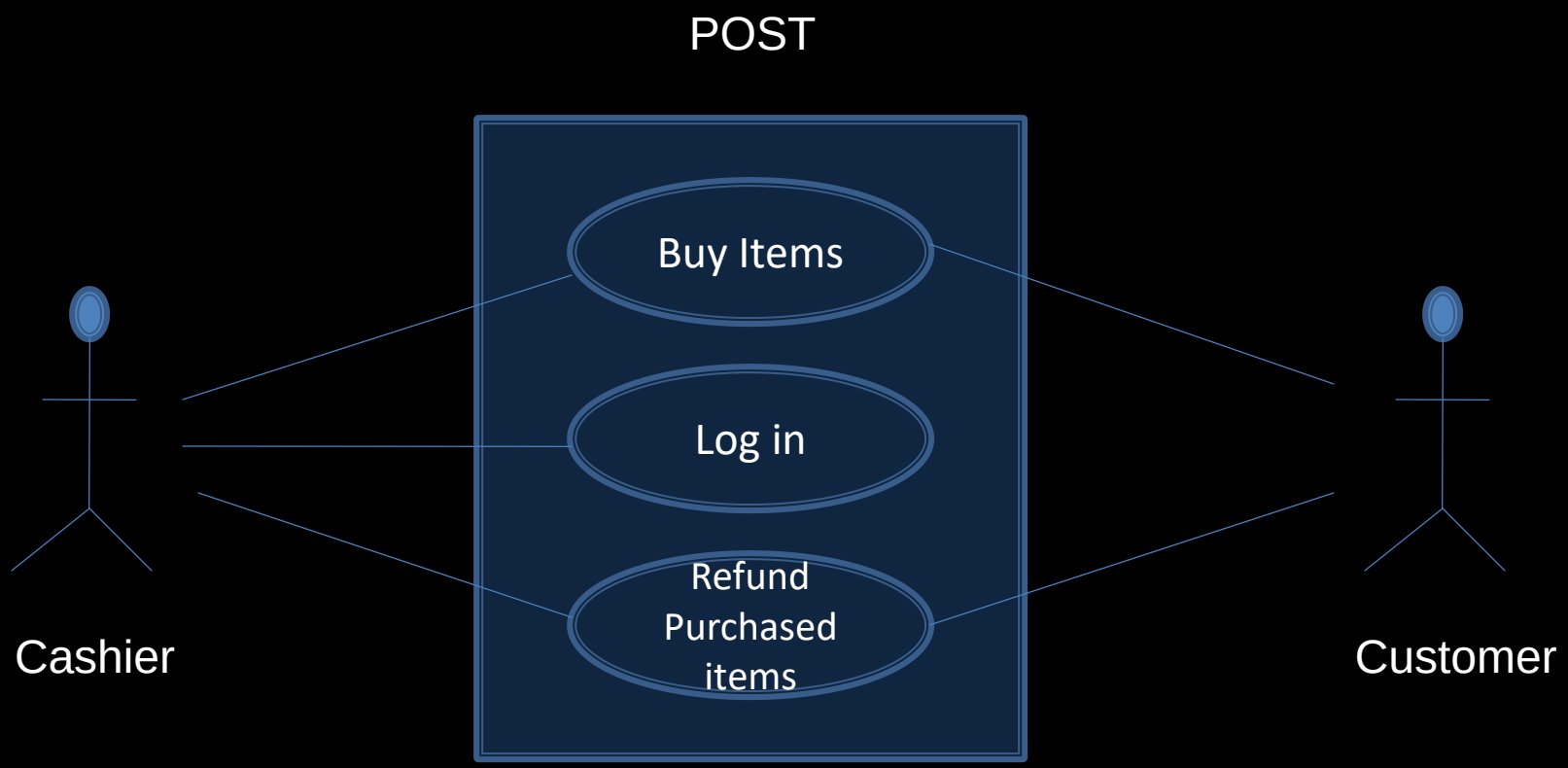
- **Scenario –** Shows what happens **when someone interacts with system.**

- **Actor – A user or another system that interacts with the** modeled system.

A use case diagram describes **relationships between actors and scenarios.**

Provides system requirements **from the user's point of View.**

5. Use Case Diagram



5. Use Case Diagram

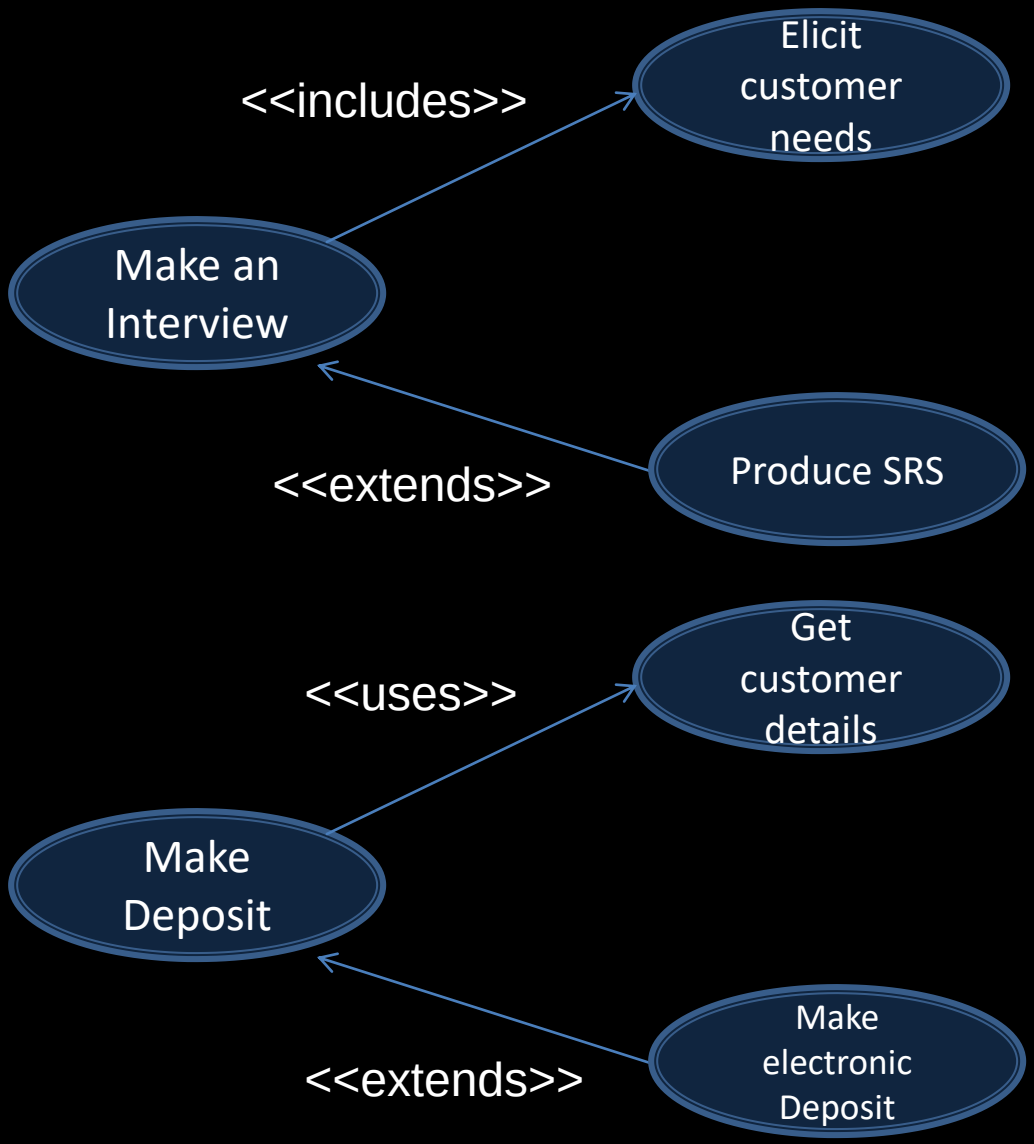
Use case Relationships:

Association – defines a relationship between an actor and a use case.

Extend - defines that instances of a use case may be augmented with some additional behavior defined in an extending *use case.*

Use/Include - drawn as a **dependency relationship** that points from the base use case to the used use case.
- defines that a use case uses a behavior defined in another use case.

5. Use Case Diagram



What a UCD is - and what it isn't

Attention **focused on the part of the business process** that is going to be supported by the IS.

It is the **end-user perspective** model.

It is **goal driven**

Helps to **identify system services.**

Are **not used as DFDs**.

Sequences, branching, loops, rules, etc. cannot (and should not) be directly expressed.

Are often **combined with activity diagrams**, which serve as their refinement

Identifying Use Cases

User Cases are used to specify the functionality of a system from the user's perspective.

A *usecase* is made up of *scenarios*. A scenario describes how a user interacts with the system in certain situations.

For example, from the point of view of the receptionist , a scenario for booking an appointment for a patient with a doctor could be:

Ask the patient which doctor they want to see. Check the diary to see when the next available appointments are. Agree the appointment date and time with the patient. Enter the details on the system.

Identifying business processes

Business processes are usually associated with a major business activity and consequently make reference to a specific *business entity*. A business entity here is anything which can be used to describe the business.

Some examples of business processes are:

- Pay Employee
- Schedule Delivery
- Receive Payment

The structure of the examples above: verb +(singular) object.

One way to find business processes is to consider the life cycle of business entities.

for a bakery we might identify:

- Order Ingredient(s)
- Bake Bread
- Deliver Bread
- Sell Bread

Activity

Working in small groups(2/3), suggest use cases for a car manufacturer to add to the ones identified below:

- DesignVehicle
- Make Vehicle
- SellVehicle

For the business process 'Recruit Staff' – decompose this process by completing the following:

Advertise_____?

_____? Candidate

Appoint_____?

Activity

Complete the use case below representing part of a simple student enrolment system

‘The Administrator at a local college enrols students onto courses as soon as she receives a signed Registration Form from a student. She then needs to update her Student Records with the student’s details before giving the student a Student Card. This completes the enrolment process.’

Activity

From the point of view of the receptionist in a Clinic, a scenario for booking an appointment for a patient with a doctor could be:

Ask the patient which doctor they want to see. Check the diary to see when the next available appointments are. Agree the appointment date and time with the patient. Enter the details on the system.

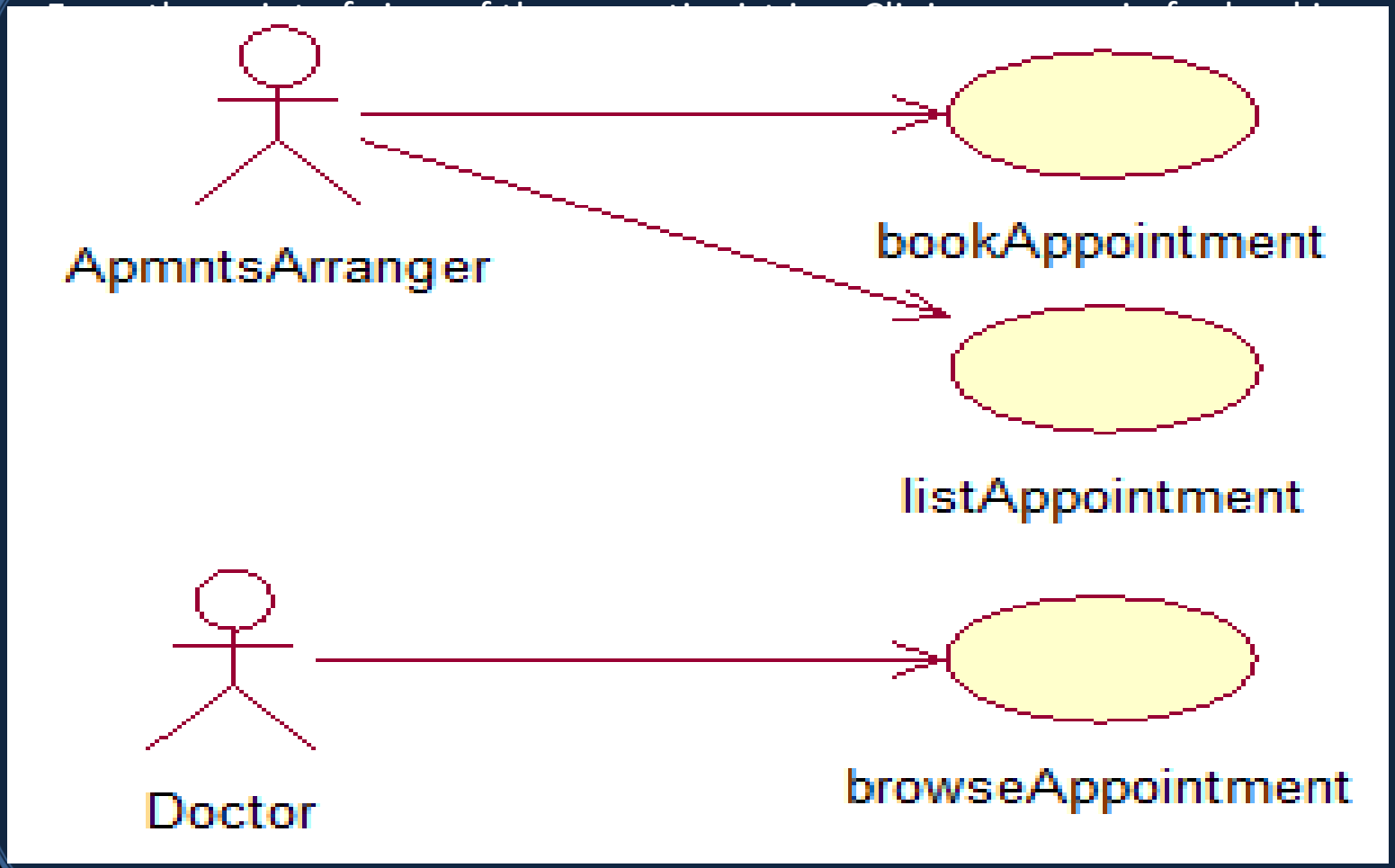
Use Case Name????

Now produce a use case diagram for appointment related activities.

Possible use cases are:

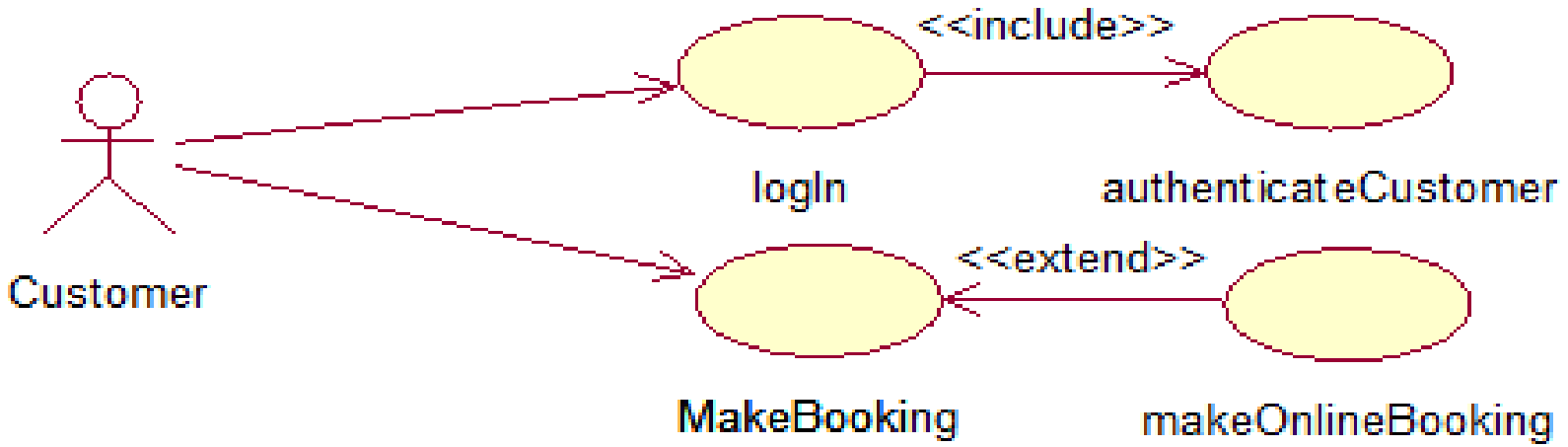
Possible actors(roles)are:

Activity


ApmntsArranger
bookAppointment
listAppointment
Doctor
browseAppointment

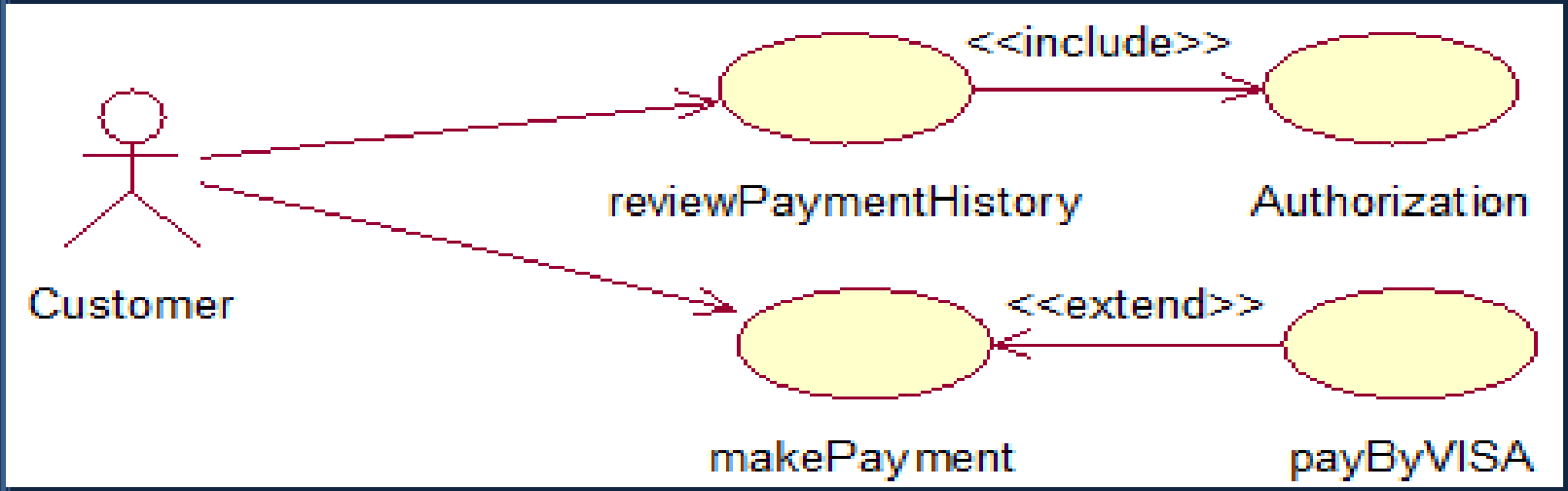
Activity

A customer must log in to the system but the system must authenticate him first. After that the customer can start making booking which can be done online as well.



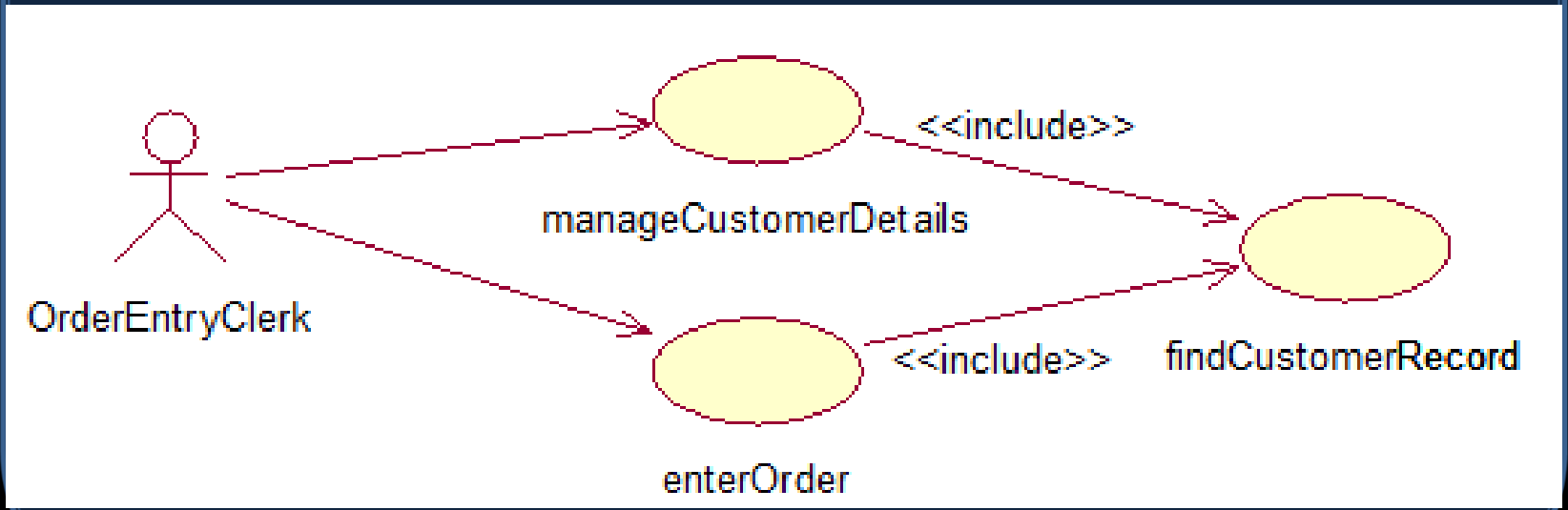
Activity

A customer can review his payment history after getting authorization from the bank. He can make payment for things he buys online , there is also a provision of electronic payment for customers.



Activity

A system allows the clerk who enters orders from customers to manage the customer details and also allows him to enter orders. For these purposes, the customer record must be identified first.

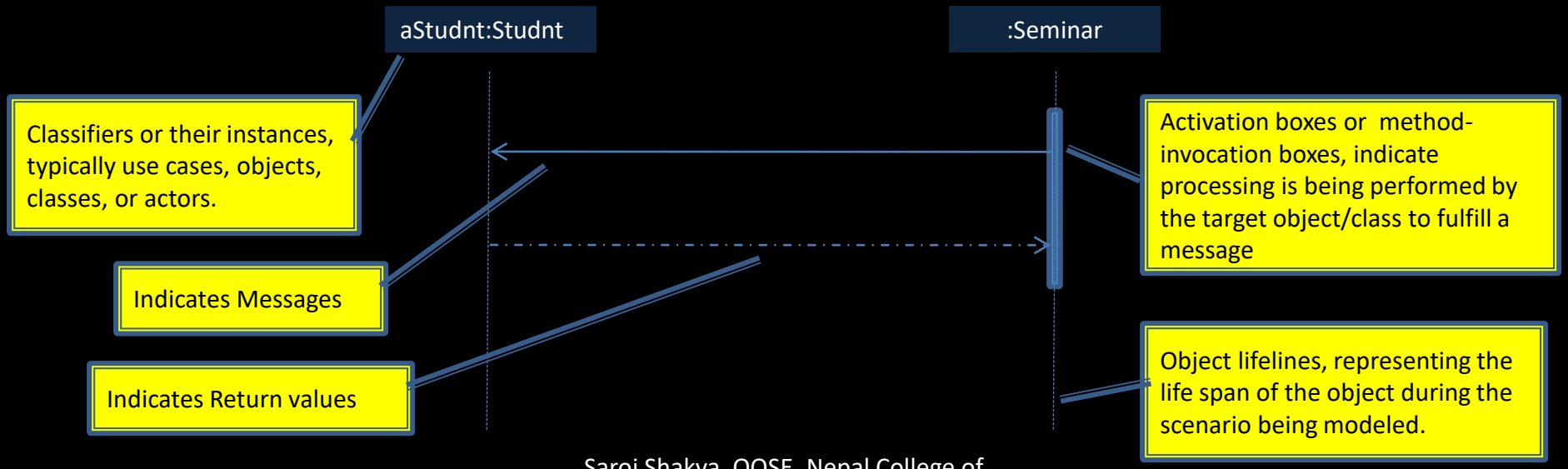


6. Sequence Diagram

A *sequence diagram* is an **interaction diagram** that **emphasizes the time ordering of messages.**

A sequence diagram shows a set of objects and the messages sent and received by those objects.

The objects are typically named or anonymous instances of classes, but may also represent instances of other things, such as collaborations, components, and nodes.

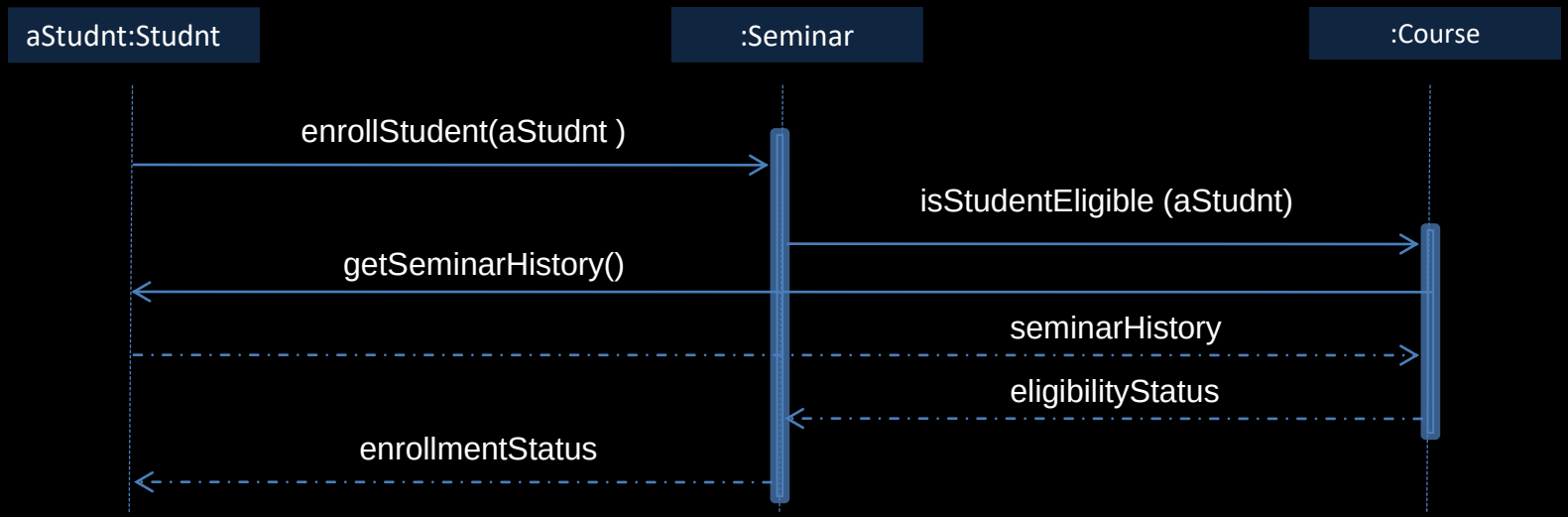


6. Sequence Diagram

A *sequence diagram* is an **interaction diagram** that **emphasizes the time ordering of messages.**

A sequence diagram shows a set of objects and the messages sent and received by those objects.

The objects are typically named or anonymous instances of classes, but may also represent instances of other things, such as collaborations, components, and nodes.



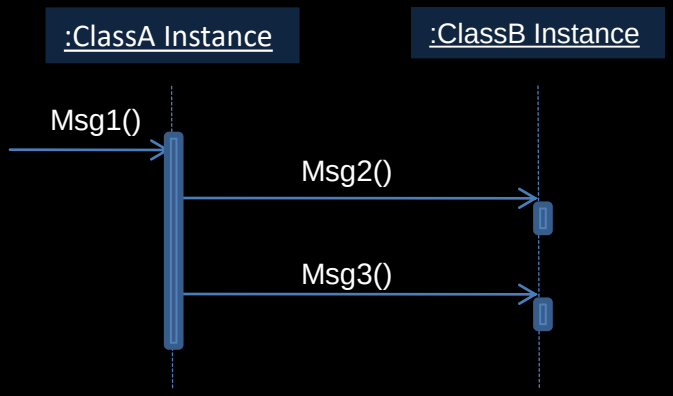
7. Collaboration Diagram

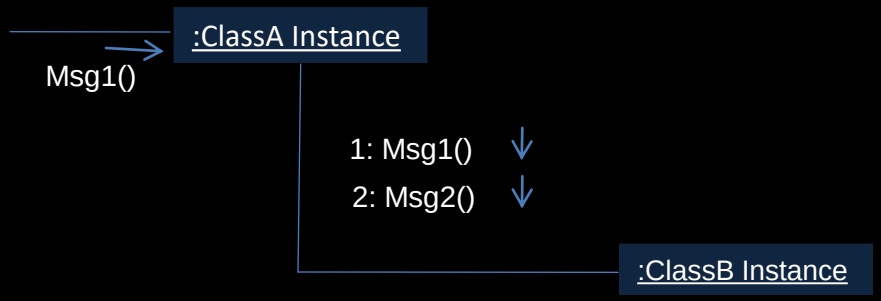
A *collaboration diagram* is an **interaction diagram** that emphasizes the structural organization of the objects that send and receive messages.

A collaboration diagram **shows a set of objects, links among those objects, and messages** sent and received by those objects.

The objects are typically **named or anonymous instances of classes,** but may also represent instances of other things, such as collaborations, components, and nodes.

Sequence and collaboration diagrams are **isomorphic**, meaning that you can convert from one to the other without loss of information.



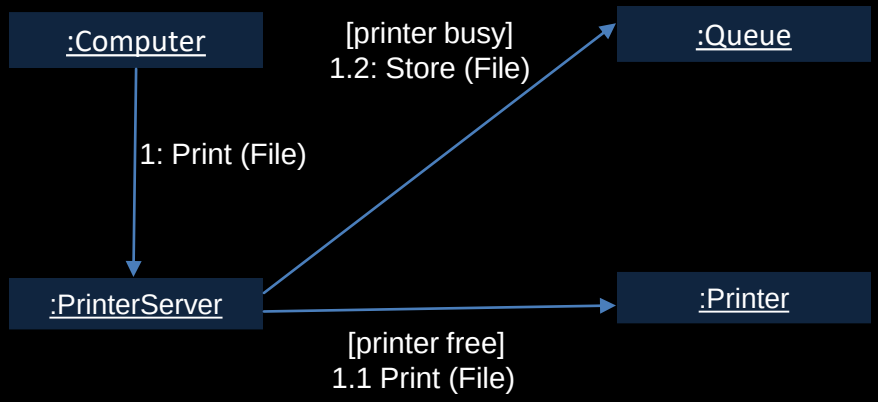


7. Collaboration Diagram

Difference?

A sequence diagram includes **chronological sequence** of messages
A sequence diagram **does not include object relationships**
Important to **visualize timing order** of messages

Collaboration diagram is used when you **are interested in the structural relationships** among the instances in an interaction
Collaboration diagram emphasizes on the **organization structure**

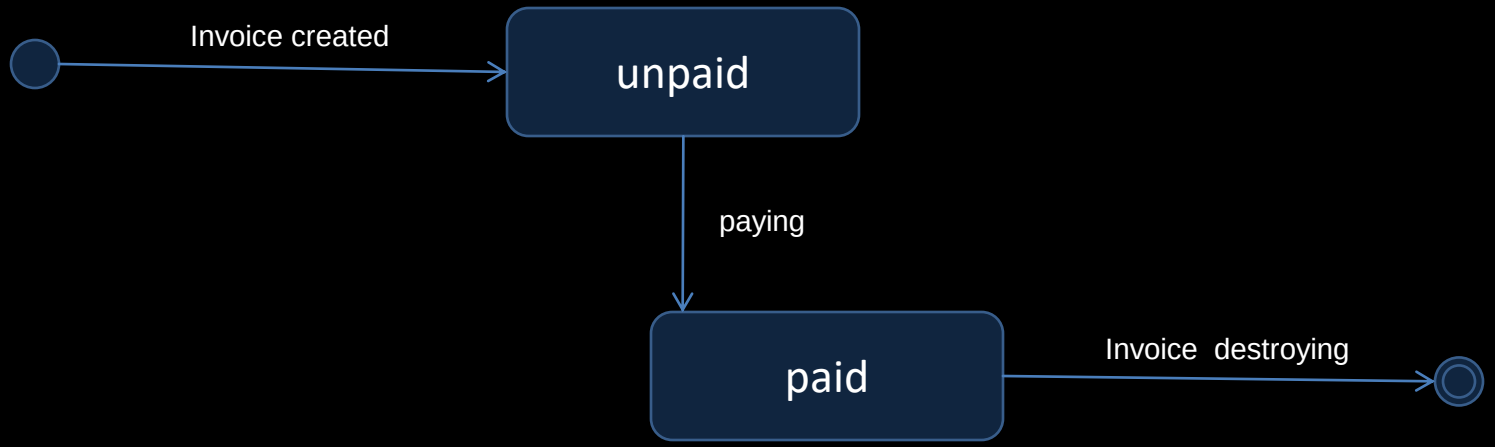


8. Statechart Diagram

A *statechart diagram* shows a **state machine**, consisting of states, transitions, events, and activities.

They are especially important in **modeling the behavior** of an interface, class, or collaboration.

Statechart diagrams emphasize the **event-ordered behavior** of an object, which is especially useful in modeling reactive systems.



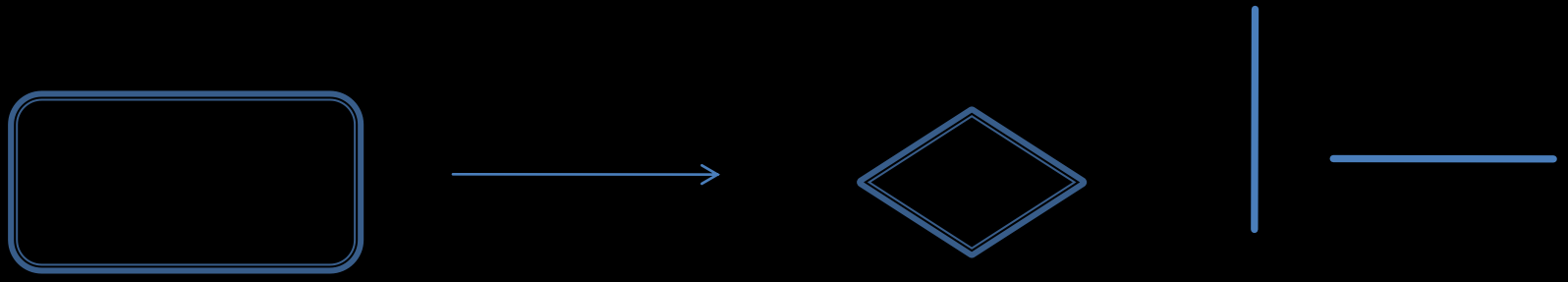
9. Activity Diagram

An *activity diagram* shows the flow from **activity to activity** within a system.

An activity shows a **set of activities, the sequential or branching flow** from activity to activity, and objects that act and are acted upon.

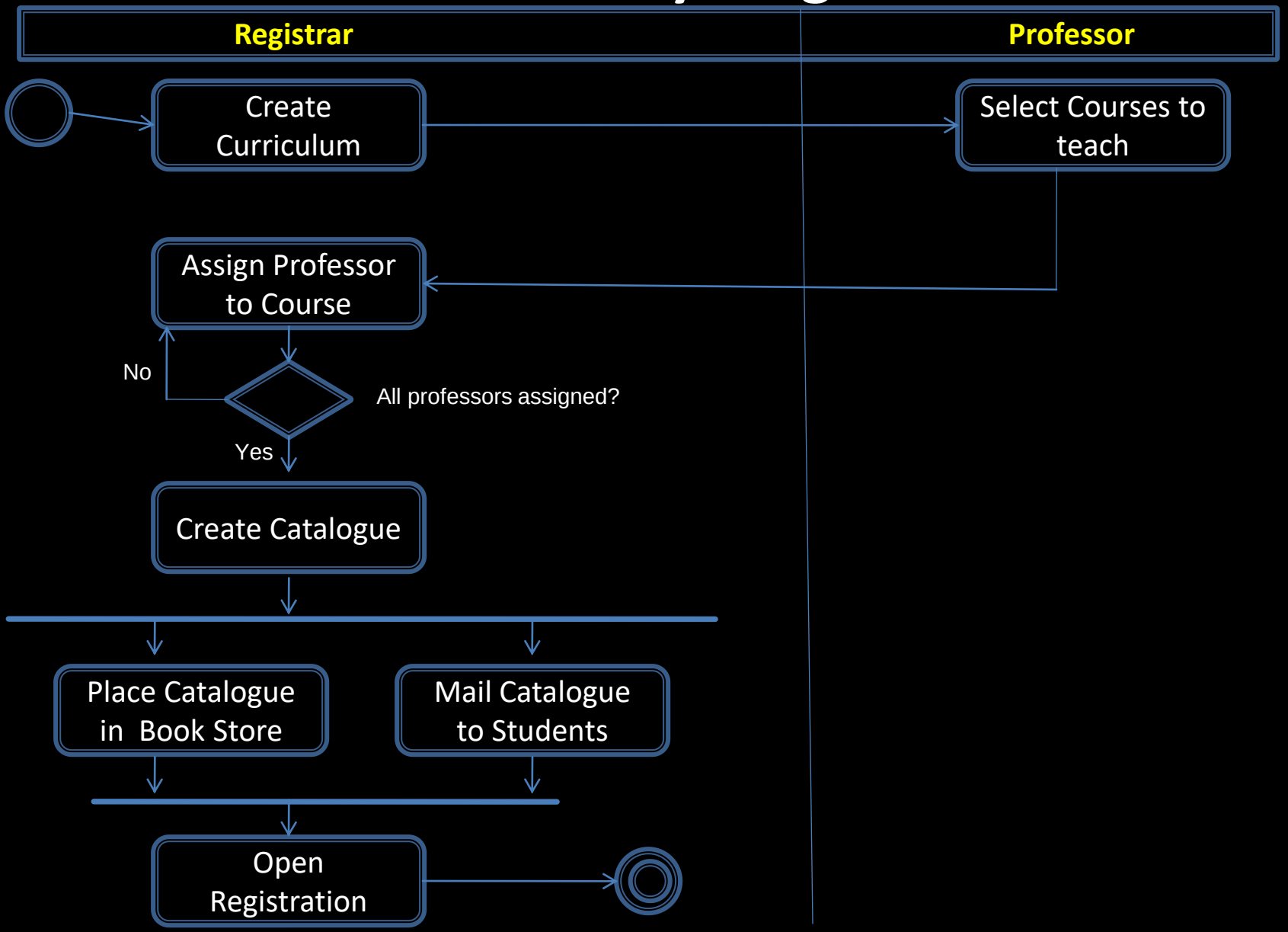
Shows what activities can be done in parallel, and any alternate paths through the flow
Activity diagrams contain **activities, transitions between the activities, decision points, and synchronization bars**

Activity diagrams emphasize the **flow of control** among objects.



Activity Transition Decision Synchronization Bar

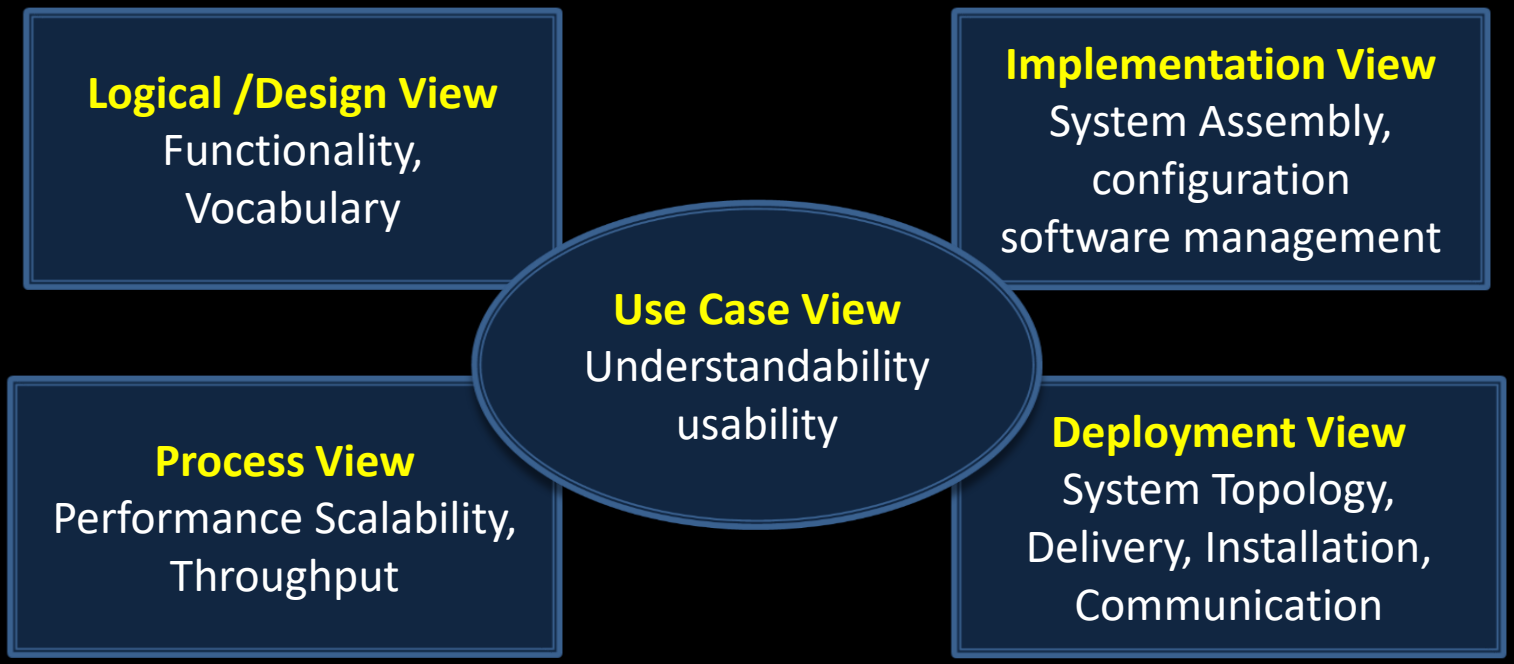
9. Activity Diagram



THE ARCHITECURE

The architecture of a software-intensive system can best be described by the **4+1** interlocking views. Each view is a projection into the organization and structure of the system, **focused on a particular aspect of that system**.

"architecture is **the structure of the system**, comprised of components or building blocks the externally visible properties of those components, and the relationships among them"



The 4+1 View: use case view

The **'plus-one'** view : further describe or consolidate the other views. *encompasses the use cases that describe the behavior of the* system as seen by its end users, analysts, and testers.

This view doesn't really specify the organization of a software system. Rather, it exists to **specify the forces that shape the system's architecture**.

With the UML, the static aspects of this view are captured in **use case diagrams;**

System components:

Record Shop :-
P: sell products (records), order stock, advertise products/shop, maintain stock, manage staff, etc
I: cust queries, cust orders, cust payment, complaints, deliveries, supp invoices, - -
O: cust receipts, supp payments, advertising literature, supp orders, - -
C: stock controls, quality controls, customer credit card checks, sales analysis,- -
E: suppliers, banks, customers, record industry, pop culture, - -

High Street Bank (eg):-
P: process accounts transactions, produce statements, arrange loans, invest capital, advise customers,, - -
I: queries, payments and withdrawals, customer details, transfer requests, -
O: statements, advice, cash, dividends, exchange rate information, - -
C: credit checks, overdraft arrangements, cheque clearance, cash-point controls-
E: stock market, customers, Bank of England, society at large, gov't legislation, -

The 4+1 View: use case view

Bus Company (eg):-
P: Plan routes, run service, maintain vehicles, issue tickets, inspect tickets, -
I: cust complaints, destinations, cash, supp invoices, parts details, - -
O: timetables, tickets, cust advice, supp payments, maintenance reports
C: vehicle servicing, vehicle checking, ticket/service inspection, complaints procedure, -
E: road network, local authority, parts suppliers, gov't legislation, society at large,

4. Business processes:
Student Records - Enrol Student/ Tutor Student/ Assess Student/ Process Result(s)/ Award Qualification/ - - (sensible others can be considered here)
[Students may need help with this – next example a little easier -]

Video shop – Check Membership No/ Note Video selected/ Receive Payment/ Issue Video Copy/ Note Video Returned/ - -

The 4+1 View: use case view

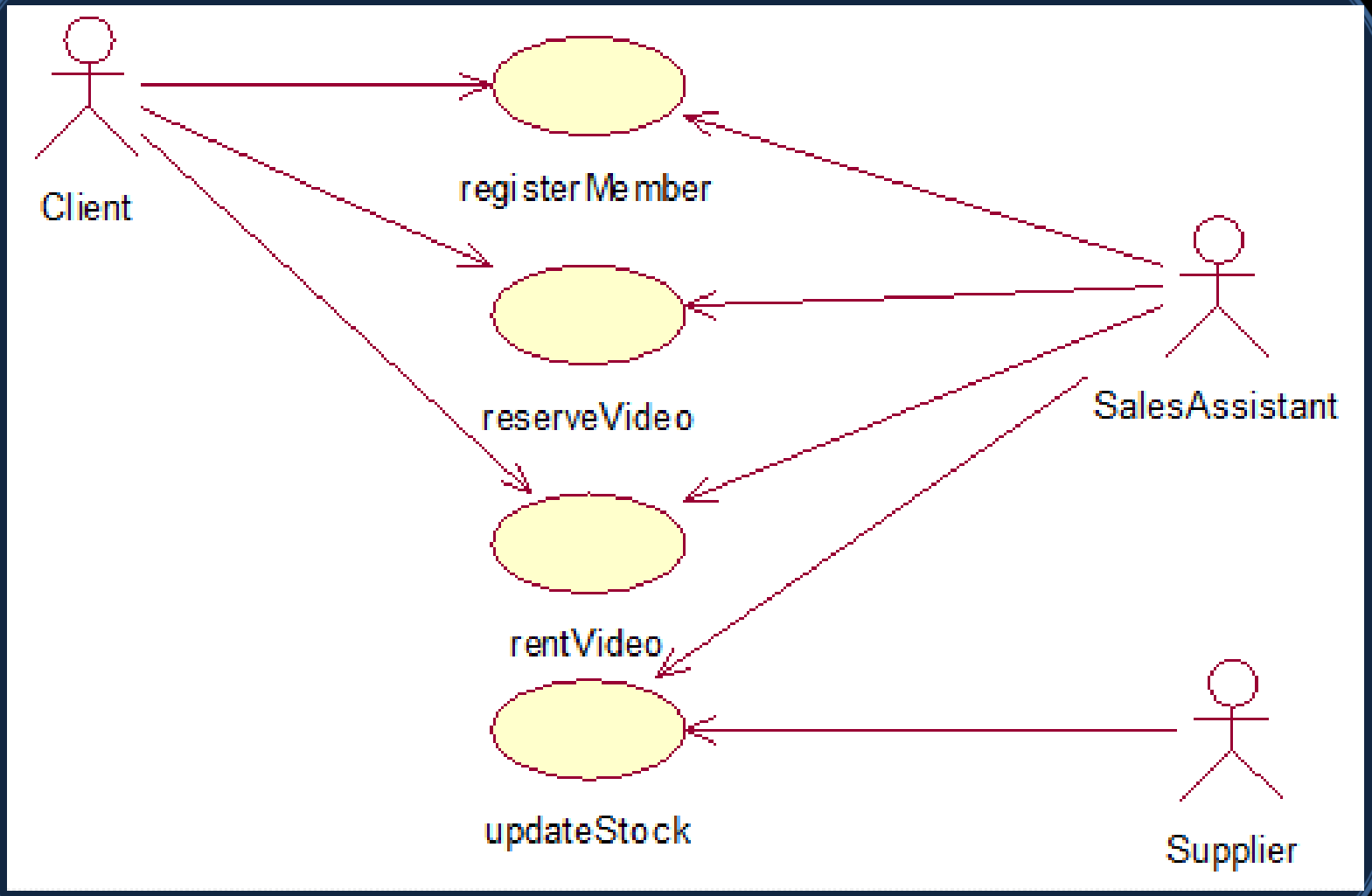
Any one who wishes to rent a video from the shop must first be registered as a member. Members may rent out videos provided that they are old enough, for this reason the Sales Assistant needs to record each member's date of birth when they join. It also records each member's address and phone number, so that they can trace them in the event of a video not being returned.

Members can reserve a video provided a copy is available for the day they require it and that they are genuine members. All video reservations are recorded in the company's video database so that the status of each copy can be monitored. Reserved videos must be collected before 7pm.

To rent a video, members must present their Membership Card. Each video attracts a rental charge depending on its popularity and how recently it has been released. All Video rented out must be returned by 6pm the following day otherwise members will be charged an extra day.

Staff need to update stock records with new titles or extra copies when deliveries are received from Suppliers and deleting records of copies which are lost or damaged on return.

The 4+1 View: use case view



The 4+1 View: The *design view*

encompasses **the classes, interfaces, and collaborations** *that form* the vocabulary of the problem and its solution.

This view primarily supports the functional requirements of the system, meaning the **services** that the system should provide to its end users.

With the UML, the static aspects of this view are captured in **class diagrams** and object diagrams

The 4+1 View: process view

encompasses **the threads and processes** *that form the system's* concurrency and synchronization mechanisms. Deals with the **dynamic aspects** of a system with **activity diagrams**

This view primarily addresses the **performance, scalability, and throughput** of the system.

With the UML, the static and dynamic aspects of this view are captured in the same kinds of diagrams as for the design view, but with a focus on the active classes that represent these threads and processes.

The 4+1 View: implementation view

encompasses the **components** *and files that are used to* assemble and release the physical system. A programmers view

This view primarily addresses the configuration management of the system's releases, made up of somewhat independent components and files that can be assembled in various ways to produce a running system.

With the UML, the static aspects of this view are captured in **component diagrams**

The 4+1 View: The physical view

The *physical view* describes how the **application is installed** and how it executes in a network of computers.

The *deployment view of a system* **encompasses the nodes** *that form the system's hardware* topology on which the system executes.

This view primarily addresses the **distribution, delivery, and installation** of the parts that make up the physical system.

With the UML, the static aspects of this view are captured in **deployment diagrams**

UP and Best Practices and Concepts

The central idea to appreciate and practice in the UP is short time boxed **iterative, adaptive development.**
Another implicit, but core, UP idea is the **use of object technologies**, including OOA/D and object-oriented programming.

UP and Best Practices and Concepts

Some best practices and key concepts in the UP include:

i. Tackle high-risk and high-value issues in early iterations

If the new system is a server application that has to handle 2,000 concurrent clients , do not wait for many months (or years) to design and implement this high risk requirement.

Rather, quickly focus on designing, programming, and proving the essential software components and architecture for this risky issue; **leave the easier work till later** iterations.

The idea is to drive down the high risks in the early iterations, so that the **project does not "fail late,"** which is a characteristic of waterfall projects that defer hard, risky concerns till later in the lifecycle.

Better to "fail early" if at all, by doing the hard things first. Thus, the UP is said to be **risk driven.**

Finally, notice that risk comes in many forms: **lack of skills or resources, technical challenges, usability, politics,** and so on. All these forms influence what is addressed in early iterations.

UP and Best Practices and Concepts

ii. Continuously engage users for evaluation, feedback, and requirements

The majority of failed projects are correlated with lack of user engagement

iii. Build a cohesive, core architecture in early iterations

This is related to tackling the high-risk concerns in early iterations, since **getting the core of the architecture established is usually a risky** or critical element.

UP and Best Practices and Concepts

iv. Continuously verify quality; test early, often, and realistically

Furthermore, iterative development is based on **feedback and adaptation**; therefore, early realistic testing and evaluation are critical activities to obtain meaningful feedback.
This is in contrast to a waterfall project, where the **significant quality assurance step is done near the end of a project**, when response is the most difficult and expensive.
In the UP, quality verification is continuously integrated from the start, so that there are not big surprises near the end of the project

Note that in the UP, quality verification also refers to process quality—each iteration, assessing how well the team is doing.

UP and Best Practices and Concepts

v. Apply use cases

Informally, use cases are written stories of using a system.
They are a mechanism to **explore and record functional requirements,**

The UP recommends applying use cases as the primary form for requirements capture, and as a driving force in planning, designing, testing, and writing end-user documentation.

UP and Best Practices and Concepts

vi. Model software visually (with the UML)

An extraordinary percentage of the human brain is **involved in visual processing**, which is a motivation behind the visual or graphical presentation of information *abstraction* is a useful practice in thinking about and communicating software designs, because this allows us to focus on important aspects, while hiding or ignoring noisy details.

vii. carefully manage requirements

Being skillful in the elicitation, recording, prioritization, tracing, and lifecycle tracking of requirements

UP and Best Practices and Concepts

viii. Practice change request and configuration management

Change request management.
Although an iterative UP project embraces change, it does not embrace chaos.

configuration management.
Configuration and build management tools are used to support frequent (ideally, at least daily) system integration and test, parallel development, separate developer workspaces and configurations, and version control—

The Rational Unified Process

Control for an iterative and incremental life cycle is supported by employing the **Rational Unified Process—an** extensive set of guidelines that address the technical and organizational aspects of software development focusing on requirements analysis and design.

Philippe Kruchten, who also led the development of the RUP, served as chief architect for the project.

The Rational Unified Process is structured along two dimensions:

- **Time**—division of the life cycle into **phases** and **iterations**
- **Process components**—production of a specific set of artifacts with **well-defined activities**

Both dimensions must be taken into account for a project to succeed.

The Rational Unified Process

Structuring a project along the time dimension involves the adoption of the following **time-based phases**:

Inception—specifying the project vision **the seed idea** *for the development is brought up* to the point of being

Elaboration—planning the necessary activities and required resources; specifying the features and designing the architecture *the* **product vision** *and its architecture are* defined , specifying particular **functional or nonfunctional behavior**

Construction—**building the product** as a series of incremental iterations *when the software is brought from an executable* architectural baseline to being ready to be transitioned to the user community

Transition—**supplying the product** to the user community(manufacturing, delivering, and training)

The Rational Unified Process

Structuring the project along the **process component dimension** includes the following activities:

Business Modeling—the identification of desired system capabilities and user needs

Requirements—a narration of the system vision along with a set of functional and nonfunctional requirements

Analysis and Design—a description of how the system will be realized in the implementation phase

Implementation—the production of the code that will result in an executable system

Test—the verification of the entire system

Deployment—the delivery of the system and user training to the customer

The Rational Unified Process