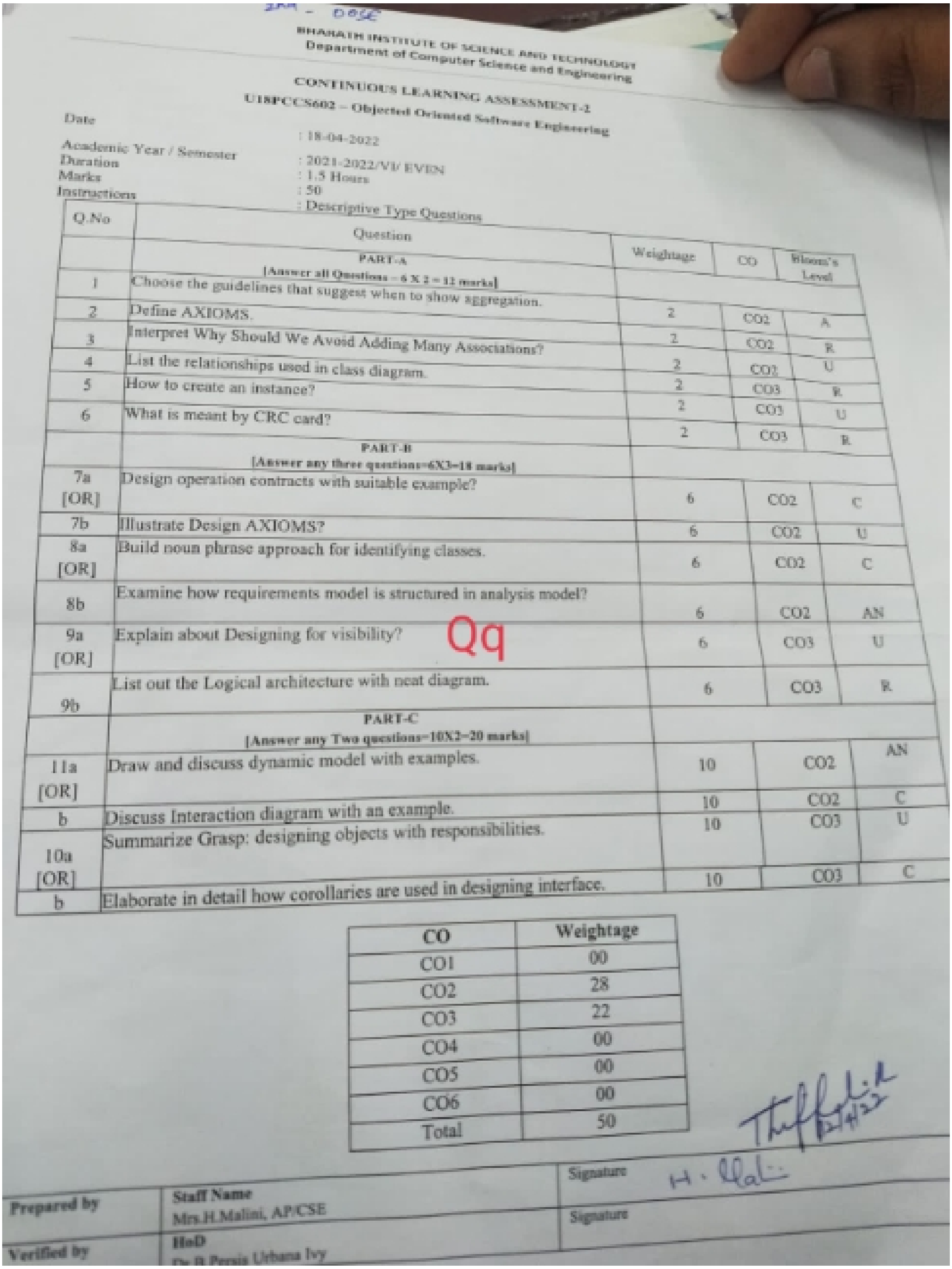


[illegible] - OOSE

BHARATH INSTITUTE OF SCIENCE AND TECHNOLOGY
Department of Computer Science and Engineering

CONTINUOUS LEARNING ASSESSMENT-2

U18PCCS602 – Objected Oriented Software Engineering

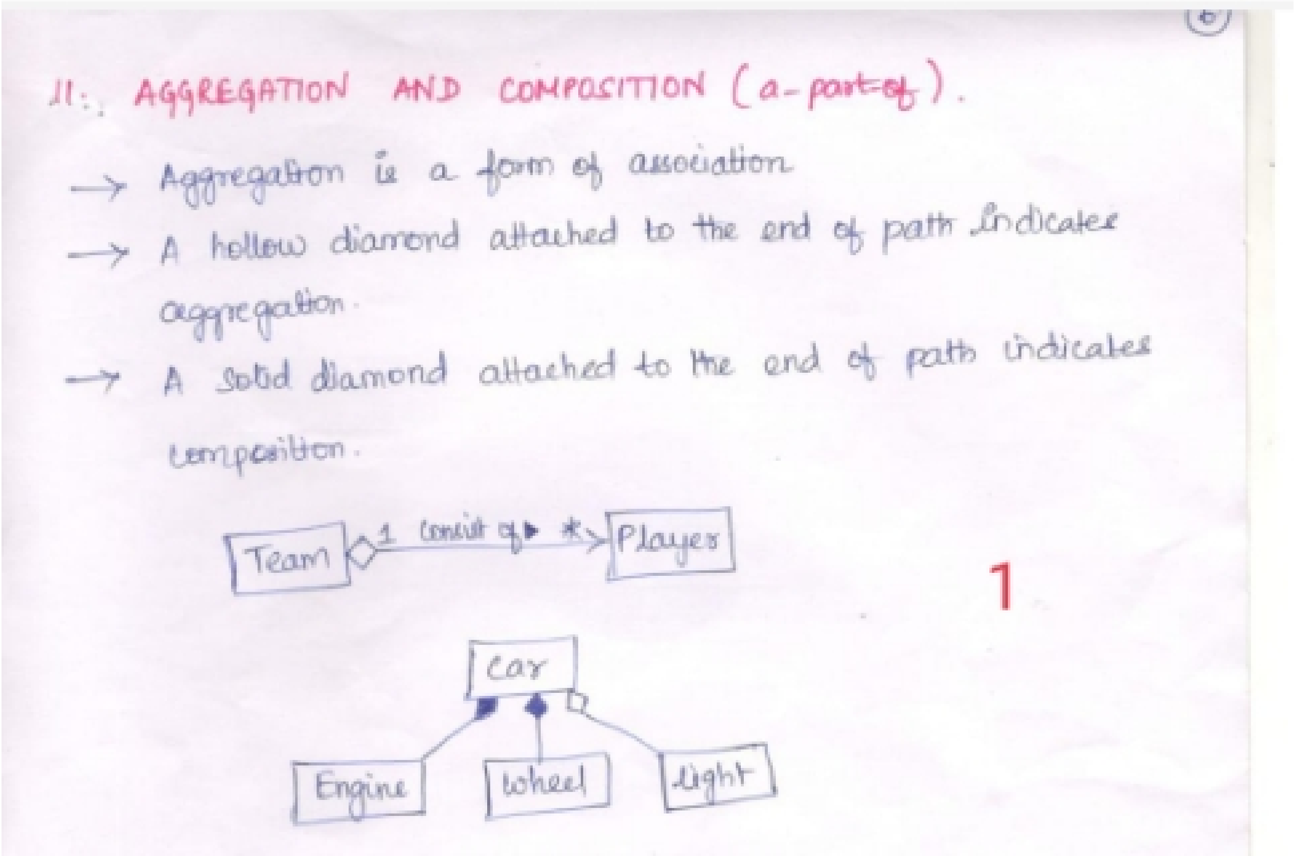
Date : 18-04-2022
Academic Year / Semester : 2021-2022/VI/ EVEN
Duration : 1.5 Hours
Marks : 50
Instructions : Descriptive Type Questions

Q.No	Question	Weightage	CO	Bloom's Level
	PART-A **[Answer all Questions – 6 X 2 = 12 marks]**			
1	Choose the guidelines that suggest when to show aggregation.			
2	Define AXIOMS.	2	CO2	A
3	Interpret Why Should We Avoid Adding Many Associations?	2	CO2	R
4	List the relationships used in class diagram.	2	CO2	U
5	How to create an instance?	2	CO3	R
6	What is meant by CRC card?	2	CO3	U
	PART-B **[Answer any three questions=6X3=18 marks]**	2	CO3	R
7a [OR]	Design operation contracts with suitable example?	6	CO2	C
7b	Illustrate Design AXIOMS?	6	CO2	U
8a [OR]	Build noun phrase approach for identifying classes.	6	CO2	C
8b	Examine how requirements model is structured in analysis model?	6	CO2	AN
9a [OR]	Explain about Designing for visibility?	6	CO3	U
9b	List out the Logical architecture with neat diagram.	6	CO3	R
	PART-C **[Answer any Two questions=10X2=20 marks]**			
11a [OR]	Draw and discuss dynamic model with examples.	10	CO2	AN
b	Discuss Interaction diagram with an example.	10	CO2	C
10a [OR]	Summarize Grasp: designing objects with responsibilities.	10	CO3	U
b	Elaborate in detail how corollaries are used in designing interface.	10	CO3	C

CO	Weightage
CO1	00
CO2	28
CO3	22
CO4	00
CO5	00
CO6	00
Total	50

12/4/22

Prepared by	**Staff Name** Mrs.H.Malini, AP/CSE	Signature H. Mali
Verified by	**HoD** Dr.B.Persia Urbana Ivy	Signature

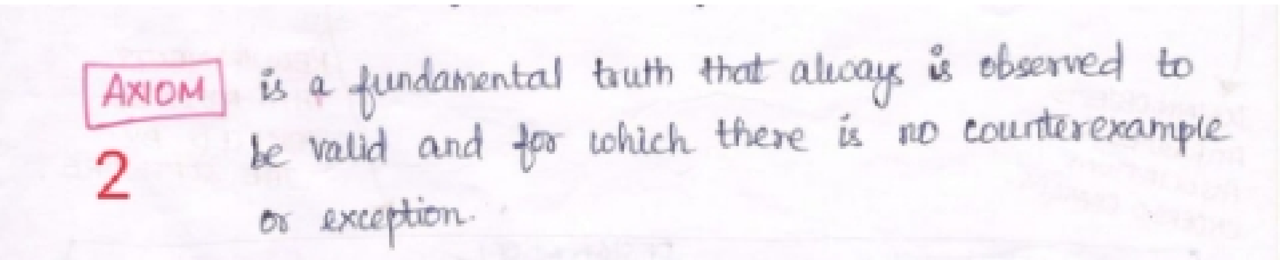


11. AGGREGATION AND COMPOSITION (a-part-of).

- → Aggregation is a form of association
- → A hollow diamond attached to the end of path indicates aggregation.
- → A solid diamond attached to the end of path indicates composition.

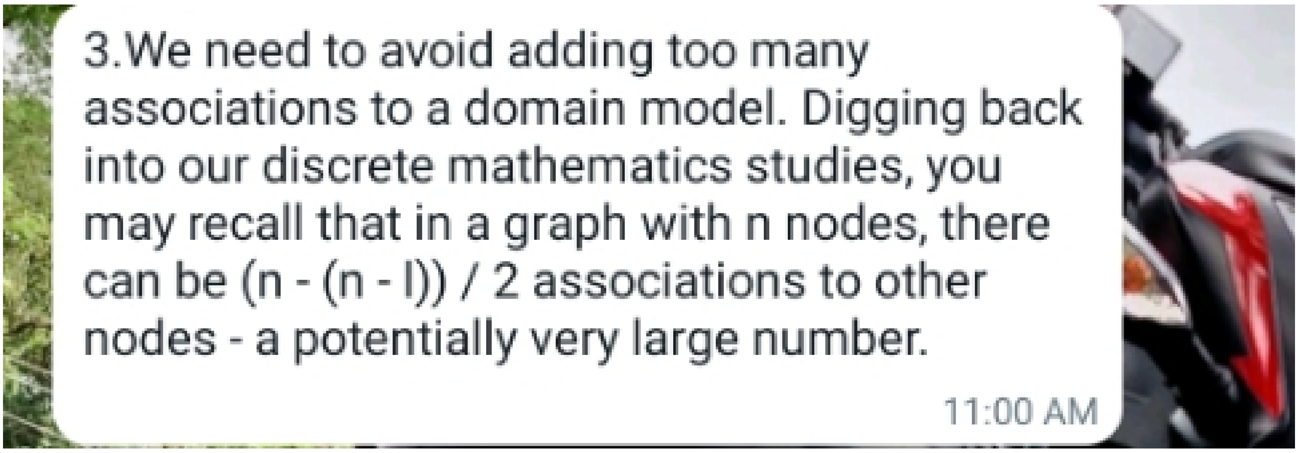
Team
1
consist of ▸
*
Player
Car
Engine
Wheel
light

1



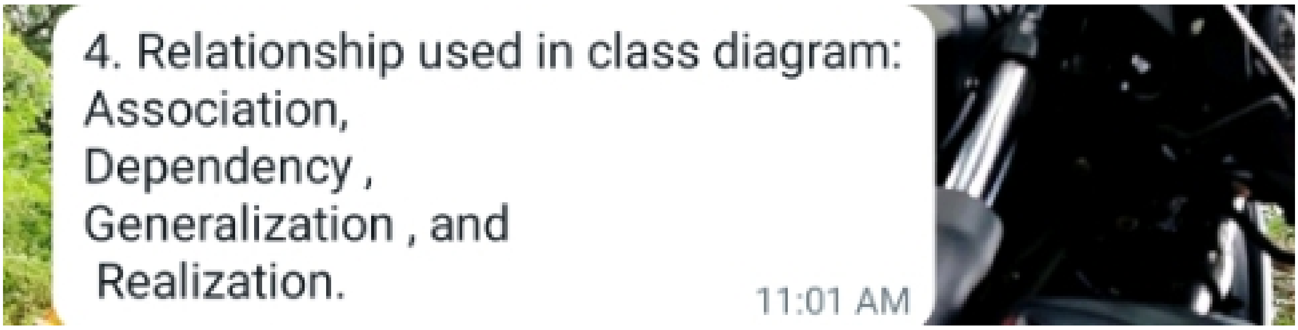
AXIOM is a fundamental truth that always is observed to be valid and for which there is no counterexample or exception.

2



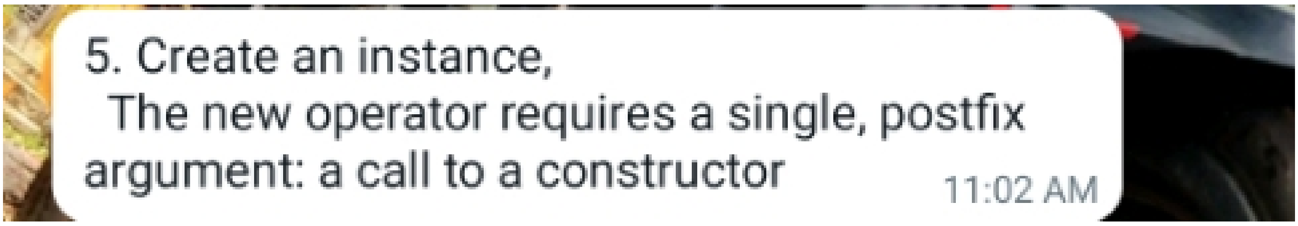
3.We need to avoid adding too many associations to a domain model. Digging back into our discrete mathematics studies, you may recall that in a graph with n nodes, there can be (n - (n - l)) / 2 associations to other nodes - a potentially very large number.

11:00 AM



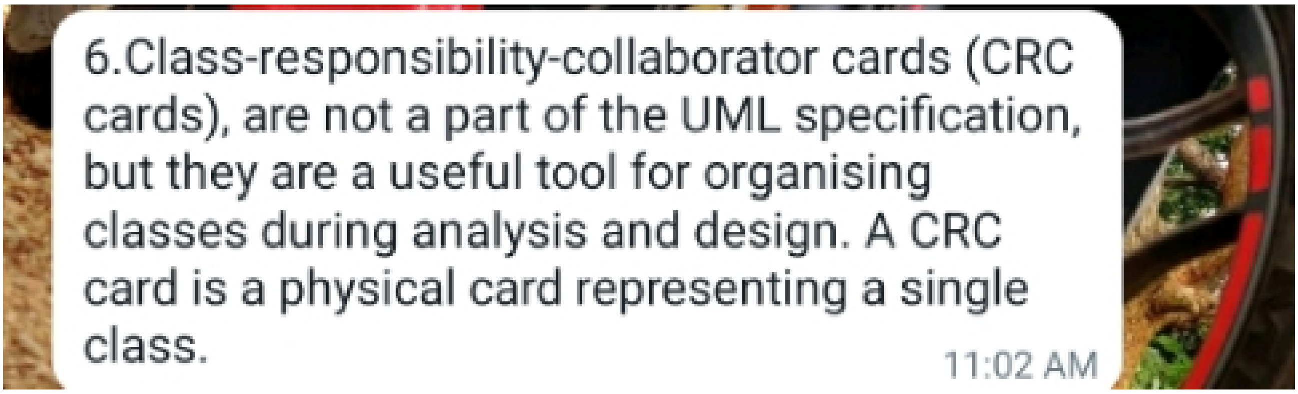
4. Relationship used in class diagram:
Association,
Dependency ,
Generalization , and
 Realization.

11:01 AM



5. Create an instance,
The new operator requires a single, postfix argument: a call to a constructor

11:02 AM



6.Class-responsibility-collaborator cards (CRC cards), are not a part of the UML specification, but they are a useful tool for organising classes during analysis and design. A CRC card is a physical card representing a single class.

11:02 AM

OPERATION CONTRACTS (Pg:9-11)

→ Operation contracts use a pre- and post- condition form to describe detailed changes to objects in a domain model, as the result of a system operation.

→ Operation contracts considered past of UP use case model because they provide more analysis detail on the effect of the system operations implied in the

7.a.1

Sections of a Contract

Operations : Name of Operation and parameters

Cross Reference : Use cases this operation can occur within

Preconditions : Noteworthy assumptions about the state of the system or objects in the domain model before execution of the operation.

Post conditions : The most important section. The state of objects in the domain model after completion of the operation.

Post Conditions → describe changes in the state of objects in the domain model. Domain model state change include

* Instance creation and deletion
* Attribute change of Value
* Associations formed and broken.

→ Pre conditions are not actions to be performed during the operation.

How to write post condition → Express in past tense

How to create & Write Contracts →

1. Identify system operations from SSD's (System sequence diagram)

2. For each system operations that are complex & which is not clear in the use case, construct a contract.

3. To describe the post conditions, use the following Categories:

   * instance creation and deletion
   * Attribute Modification
   * Associations formed and broken.

7.a.2

WRITING CONTRACTS

* Write the postconditions in a declarative, past tense form.

  (eg) A SaleLineItem was created (Better)
  Create a SaleLineItem (Worst)

* Remember to establish an association between existing objects or those newly created.

  (eg) SaleLineItem was associated with the sale. (New Association formed).

NextGen POS Contracts — System Operations of Process Sale Use Case

Contract CO1 : makeNewsale.

7.a.3

Operation : makeNewsale ( )

Cross reference : Use cases : Process sale.

Preconditions : none

Post conditions : - A sale instance s was created (Instance creation) ↑ single occurrene

- s was associated with a Register (association formed)
- Attribute of s were initialized.

Contract CO2 : enterItem ( )

Operation : enterItem (ItemID : ItemID, quantity : integer)

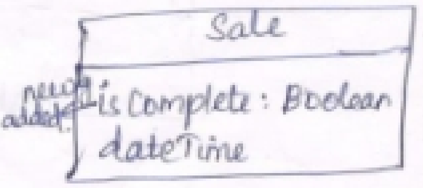
Cross reference : Use cases : Process Sale.

Preconditions : There is a sale underway.

Post conditions : - A SalesLineItem instance 'sli' was created

- 'sli' was associated with current sale. (Associatin formed)
- sli quantity became quatity (attribute modification)
- sli was associated with ProductDescription, based on itemID match (Association formed).

Contract CO3 : endSale.



Operation : endSale ( )

Cross reference : UseCases : Process Sale

Preconditions : There is a sale underway.

Post conditions : Sale.isComplete became true (Attribute Modification)

Operation Contracts Expressed with OCL

(eg) System :: makeNewSale ()

Pre : <Statements in OCL>

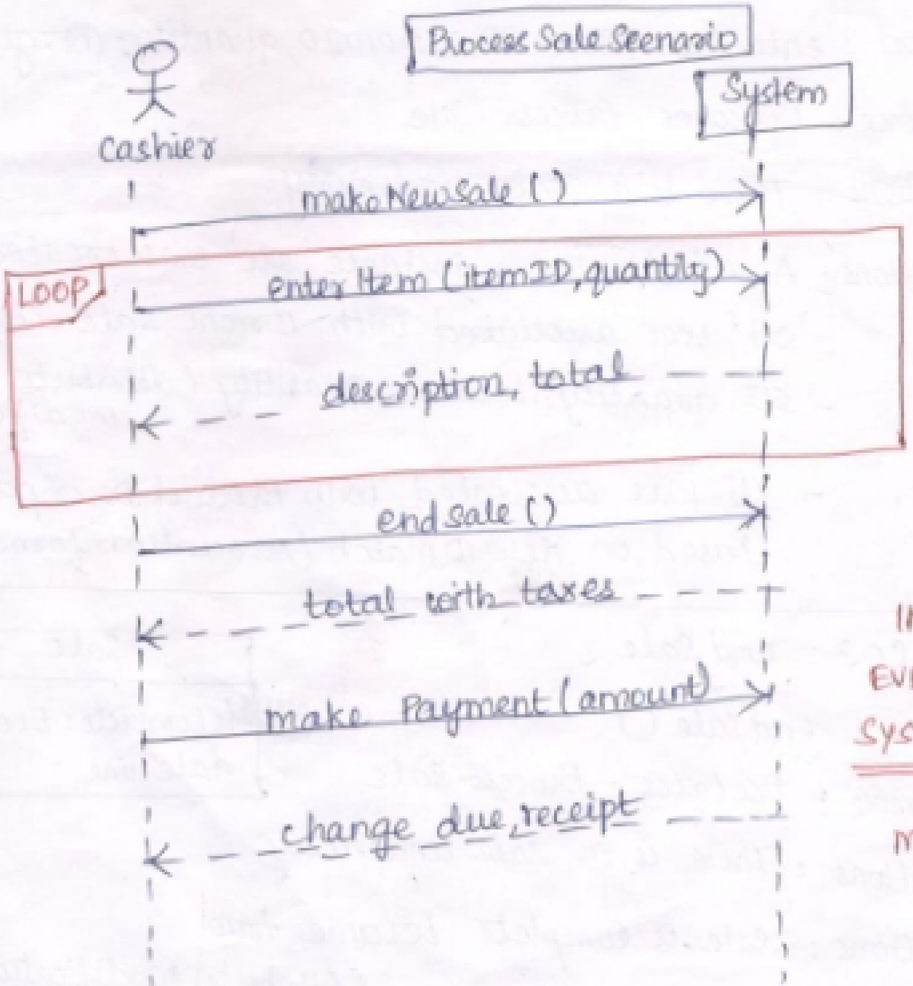
Post : ....

7.a.4

→ In UP, Phases :

Inception → Contracts are not motivated

Elaboration → Most contracts will be written. Write contracts for complex operations.



INPUT SYSTEM EVENTS INVOKE SYSTEM OPERATIONS

message invokes methods()

Noun Phrase approach

- *The noun phrase approach was proposed by Rebecca Wirfs-Brock, Brian Wilker -son, and Lauren Wiener.*
- *In this method, you read through the requirements or use cases looking for noun phrases.*
- *Nouns in the textual description are considered to be classes and verbs to be methods of the classes.*
- *The nouns are listed, and the list divided into three categories: relevant classes, fuzzy classes (The 'fuzzy area," classes we are not sure about), and irrelevant classes.*

1. **Identifying Tentative classes.**
2. **Selecting classes from the Relevant and fuzzy categories.**
3. **The vialNet Bank ATM System: Identifying classes by using Noun phrase Approach.**
4. **Initial List of Noun phrases: candidate classes.**
5. **Reviewing the Redundant classes and Building a common vocabulary.**
6. **Reviewing the classes containing Adjectives.**
7. **Reviewing the possible Attributes.**
8. **Reviewing the class purpose.**

8.a

Identifying Tentative classes: -

- The following are guidelines for selecting classes in an application.
- Look for nouns and noun phrase in the use cases.
- Some classes are implicit or taken from general knowledge.
- All classes must make sense in the application domain; avoid computer Implementation classes- defer them to the design stage. carefully choose and define class names.
- Finding classes is an incremental and iterative process.

Selecting classes from the Relevant and fuzzy categories.

- **Redundant classes**:- DO not keep two classes that express the same information. Choose your vocabulary carefully use the word that is being used by the user of the system.
- **Adjective classes**:- An adjective can suggest a different kind of object, different use of the Same object, or it could be utterly irrelevant
- For example, Adult Members behave differently then youth Members, so the two should be classified as different classes.
- **Attribute classes**:- Tentative objects that are used only as values should be defined or restated as attributes and not as a class.
- **Irrelevant classes**:- Each class must have a purpose and every class should be clearly defined and necessary.

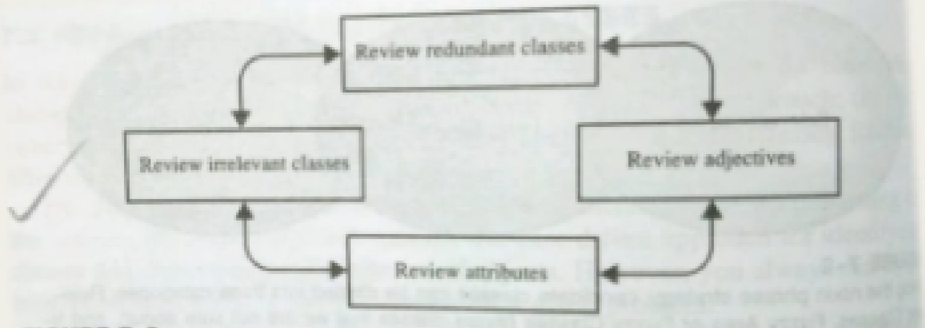


FIGURE 7–3
The process of eliminating the redundant classes and refining the remaining classes is not sequential. You can move back and forth among these steps as often as you like.

DESIGNING FOR VISIBILITY

9.a.1

Visibility → is the ability of an object to "see" or have a reference to another object.

* There are four common ways – visibility can be achieved from object A to Object B.

1. Attribute Visibility → B is an attribute of A.
2. Parameter Visibility → B is a parameter of a method of A.
3. Local Visibility → B is a (non-parameter) local object in a method of A.
4. Global Visibility → B is in some way globally visible.

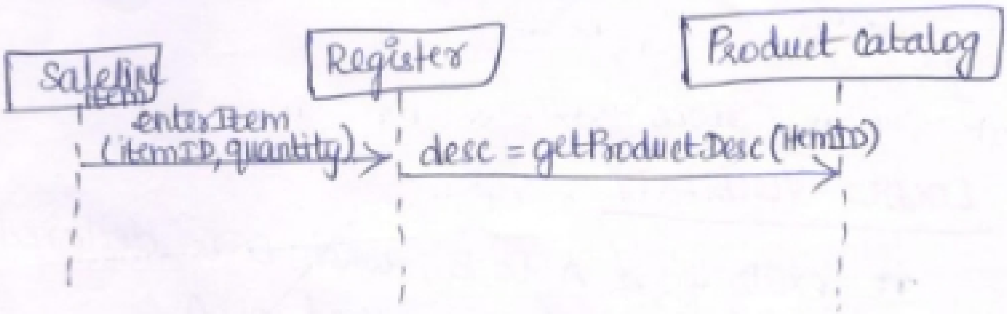
Motivation to consider Visibility

→ For an object A to send a message to an object B, B must be visible to A.

1. ATTRIBUTE VISIBILITY

9.a.2

```
Public class Register
{ ...
  Private ProductCatalog catalog;
  ...
}.
```

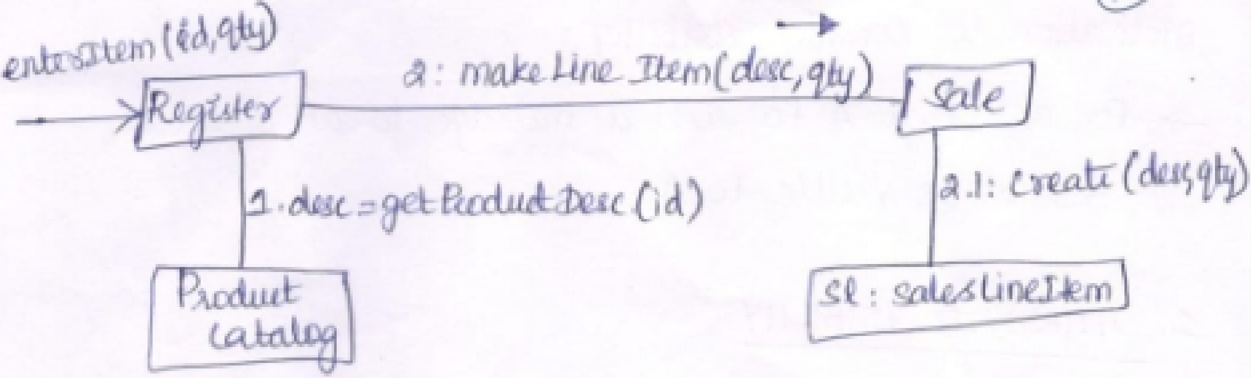


```
class Register
{ ...
 Private ProductCatalog catalog;
 ...
}
```

```
Public Void enteritem(ItemID, qty)
{
  ....
  desc = ~~get~~ catalog.getProductDesc(ItemID)
  ....
}.
```

2) PARAMETER VISIBILITY

Parameter Visibility from A to B exists when B is Passed as a parameter to a method of A.



```
makeLineItem ( ProductDescription desc , int qty)
{
   ...
   sl =  new  SalesLineItem ( desc, qty);
   ...
}.
```

LOCAL VISIBILITY 9.a.3

It exists from A to B, when B is declared as a local object within a method of A.

```
SaleLineItem ( ProductDescription  desc , int qty)
{ ...
   description = desc ;
   ....
 }.
```

Two Common ways to achieve local Visibility.

- Create a new local instance & assign it to a local variable.
- Assign the returning object from a method invocation to a local Variable.

4. GLOBAL VISIBILITY

It exists from A to B, when B is global to A.

eg: local visibility.

9.a.4

```
enterItem (id, qty)
{ ...
  ProductDescription desc = catalog.getProductDesc(id);
  ...
}.
```

DESIGNING OBJECT WITH RESPONSIBILITIES (Pg: till 24)

GRASP : A Learning Aid for OO Design with responsibilities.

* An approach to understand and use design principles based on patterns of assigning responsibilities.

Grasp Patterns →

* Creator
* Information Expert
* Low Coupling
* Controller
* High Cohesion.

CREATOR

Name : creator

Problem : Who creates an A?

Solution : Assign class B the responsibility to create an instance of class A if one of these is true.

* B "contains" or compositely aggregates A.
* B records A.
* B closely uses A.
* B has the initializing data for A

10.a.1

INFORMATION EXPERT

Name : Information Expert

Problem : What is the basic principle by which to assign responsibilities to objects?

Solution : Assign a responsibility to class that has the information needed to fulfill it.

LOW COUPLING

Name : Low coupling.

Problem : How to reduce the impact of change ?

Solution : Assign responsibilities so that (unnecessary) coupling remains low.

CONTROLLER

Name : Controller

Problem : What first object beyond the UI layer receives and coordinates ("controls") a system operation ?

Solution : Assign the responsibility to an object representing one of these choices.

* Represents the overall "system", a "root object", a device that the software is running within, or a major subsystem.

10.à.2

* Represents a usecase scenario within which the system operation occurs.

Consider Options :

1. Represents the overall "system" or "root object" - such as "SALE".
2. Represents a device that the software is running within - specialized hardware devices such as phone / cash machine. (S/W classes).
3. Represents usecase or session.

HIGH COHESION

Name : High Cohesion

Problem : How to keep object focused, understandable and manageable and as a side effects, support low coupling?

Solution : Assign responsibility so that cohesion remains high.

Applying GRASP to Object Design

Grasp → General Responsibility Assignment Software Patterns.

There are nine Grasp patterns.

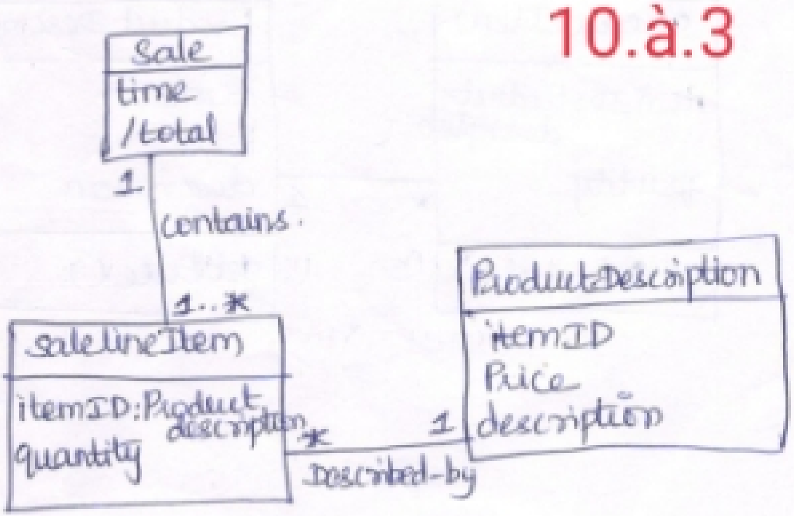
Creator
Information Expert
Low Coupling

Controller
High Cohesion
Polymorphism

Pure Fabrication
Indirection
Protected Variations.

Example : POS Domain

10.à.3



1) ~~Infor~~ Creator

→ Consider the partial domain model,

→ Creator pattern suggests that sale is a good candidate to have the responsibility of creating Sale lineItem

2) Information Expert (Expert).

→ Who should be responsible for knowing the grand total of a sale?

→ Generate design model.

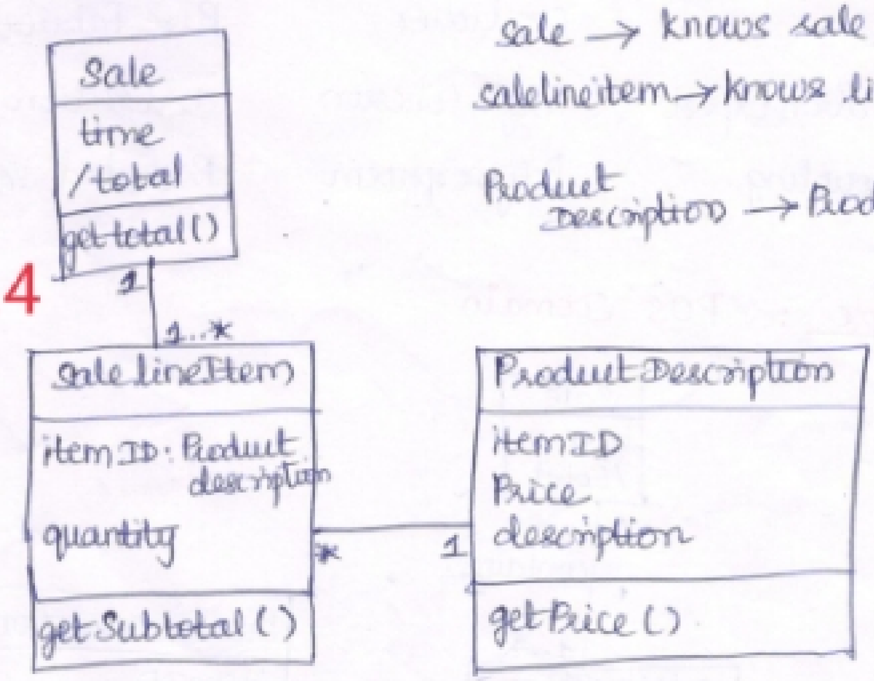
→ Domain Model → Conceptual classes of the domain
Design Model → Software classes. (Methods).

sale → knows sale total

saleline item → knows line item subtotal

Product Description → Product Price

10.a.4



3) LOW COUPLING

coupling → measure of how strongly one element is connected to.

→ An element with low/weak coupling is not dependent on too many elements.

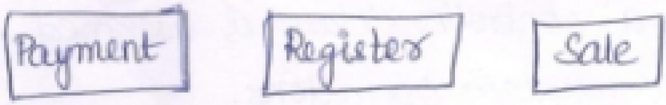
High coupling impacts → • local changes affect the related classes.

• Harder to understand in isolation

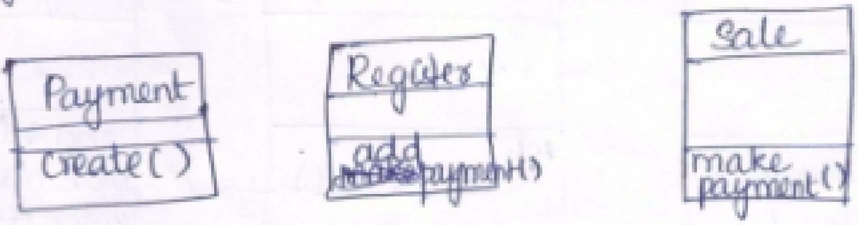
• Harder to reuse.

10.a.5

consider the following partial class diagram.



* Register records payment.



4) Controller

→ What first object beyond the UI layer receives & coordinates a system operations?

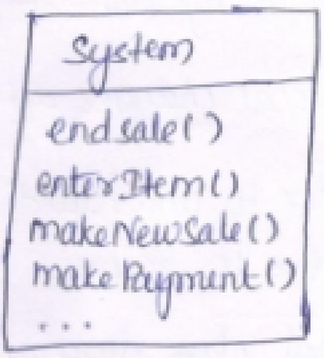
→ System operations → major input events in our system.

POS Domain → endsale button → generating a system event indicating the sale has ended.

→ A Controller is beyond the UI layer that is responsible for receiving or handling a system operation messages.

→ During analysis, system operations may be assigned to the class System.

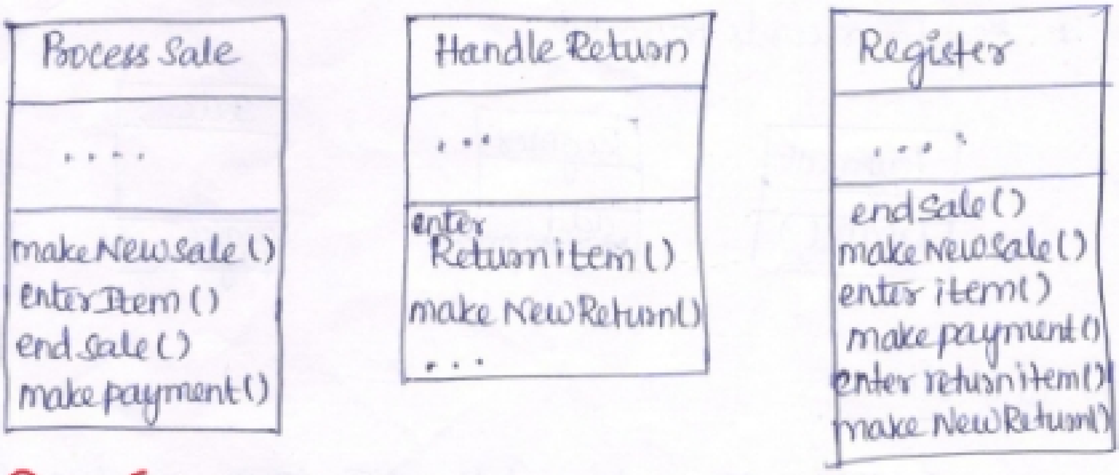
Represents overall system/ root object



→ a device that the s/w is running within..

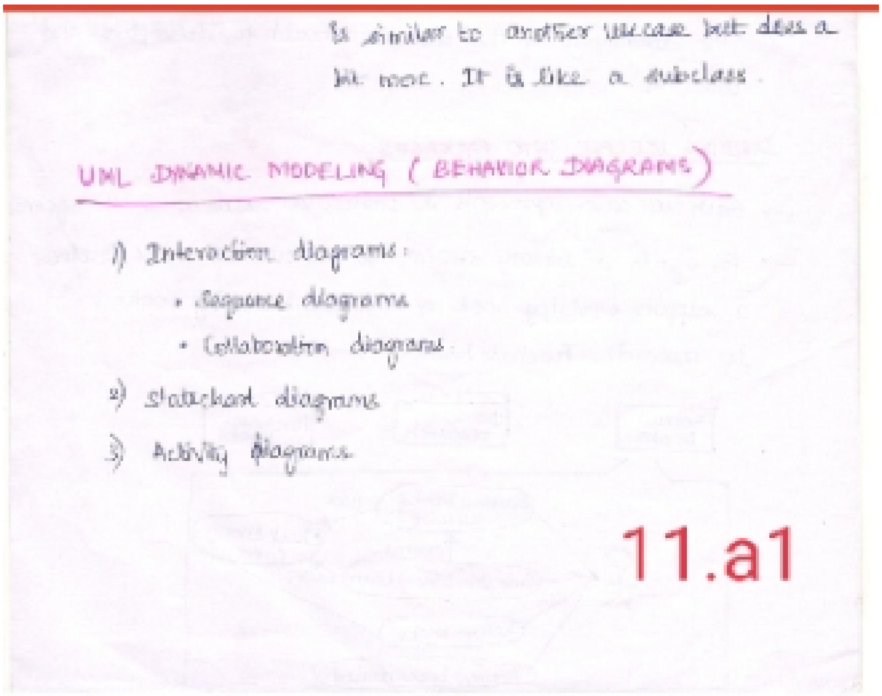
→ Some system Operations of Nextgen POS appln.

→ During design, a controller class is assigned the responsibility for system Operations.



10.a.6

Note: Allocation of System Operations during design, using several use case controllers.

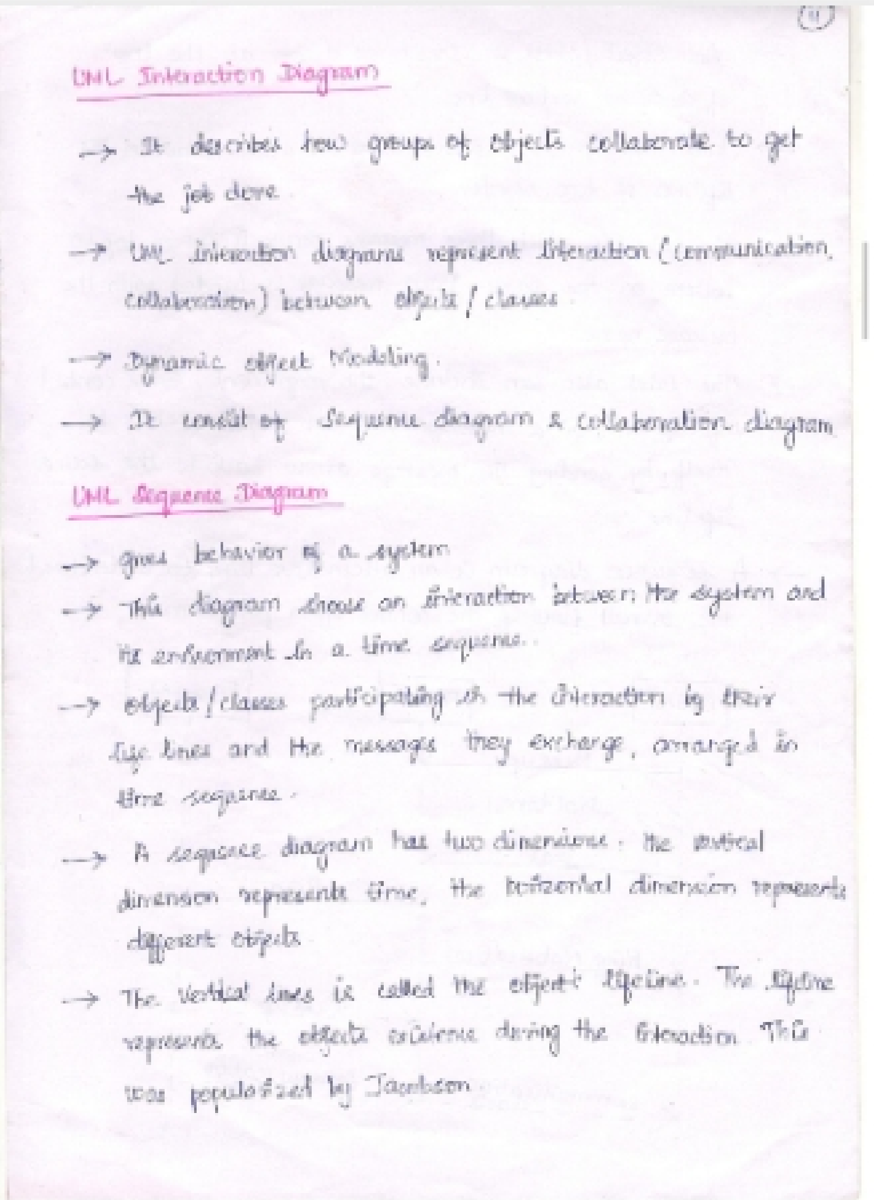


is similar to another usecase but does a bit more. It is like a subclass.

UML DYNAMIC MODELING (BEHAVIOR DIAGRAMS)

1) Interaction diagrams:
   - Sequence diagrams
   - Collaboration diagrams
2) Statechart diagrams
3) Activity diagrams

11.a1



UML Interaction Diagram

→ It describes how groups of objects collaborate to get the job done.

→ UML interaction diagrams represent interaction (communication, collaboration) between objects / classes.

→ Dynamic object Modeling.

→ It consist of Sequence diagram & collaboration diagram

UML Sequence Diagram

→ gives behavior of a system

→ This diagram shows an interaction between the system and its environment in a time sequence.

→ objects/classes participating in the interaction by their life lines and the messages they exchange, arranged in time sequence.

→ A sequence diagram has two dimensions. the vertical dimension represents time, the horizontal dimension represents different objects

→ The vertical lines is called the object's lifeline. The lifeline represents the objects existence during the interaction. This was popularized by Jacobson.

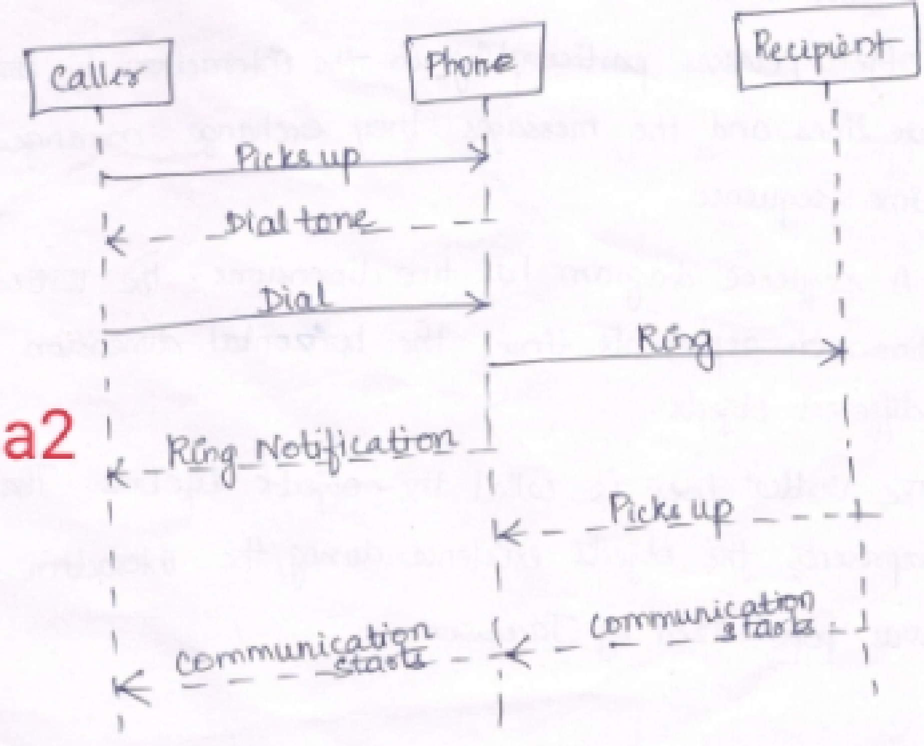
→ An object / class is shown as a box at the top of a dashed vertical line.

→ Each message is represented by an arrow between the lifelines of two objects.

→ The order in which these messages occur is shown top to bottom on the page. Each message is labeled with the message name.

→ The label also can include the argument, some control information, or a message that an object sends to itself, by sending the message arrow back to the same life line.

→ A sequence diagram is an alternative way to understand the overall flow of the control of a program.



11.a2

UML Collaboration Diagram

→ Another type of ~~collo~~ Interaction diagram.

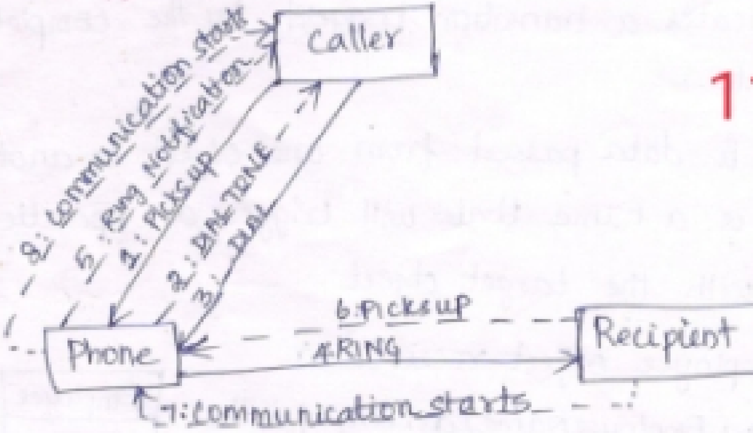
→ A collaboration diagram represents a collaboration, which is a set of objects related in particular context and interaction, which is a set of messages exchanged among the objects within the collaboration to achieve a desired outcome.

→ Sequence of collaboration is indicated by numbering the messages.

→ Numbering the messages make it more difficult to see the sequence than drawing the lines on the page.

→ A collaboration diagram provides several numbering schemes.

→ The simplest numbering is Integer Values.



11.a.3

→ Decimal Numbering scheme can also be used.