CMR
INSTITUTE OF
TECHNOLOGY

USN | 1 | C | | | | | | | | |

Internal Assessment Test II – Nov 2017

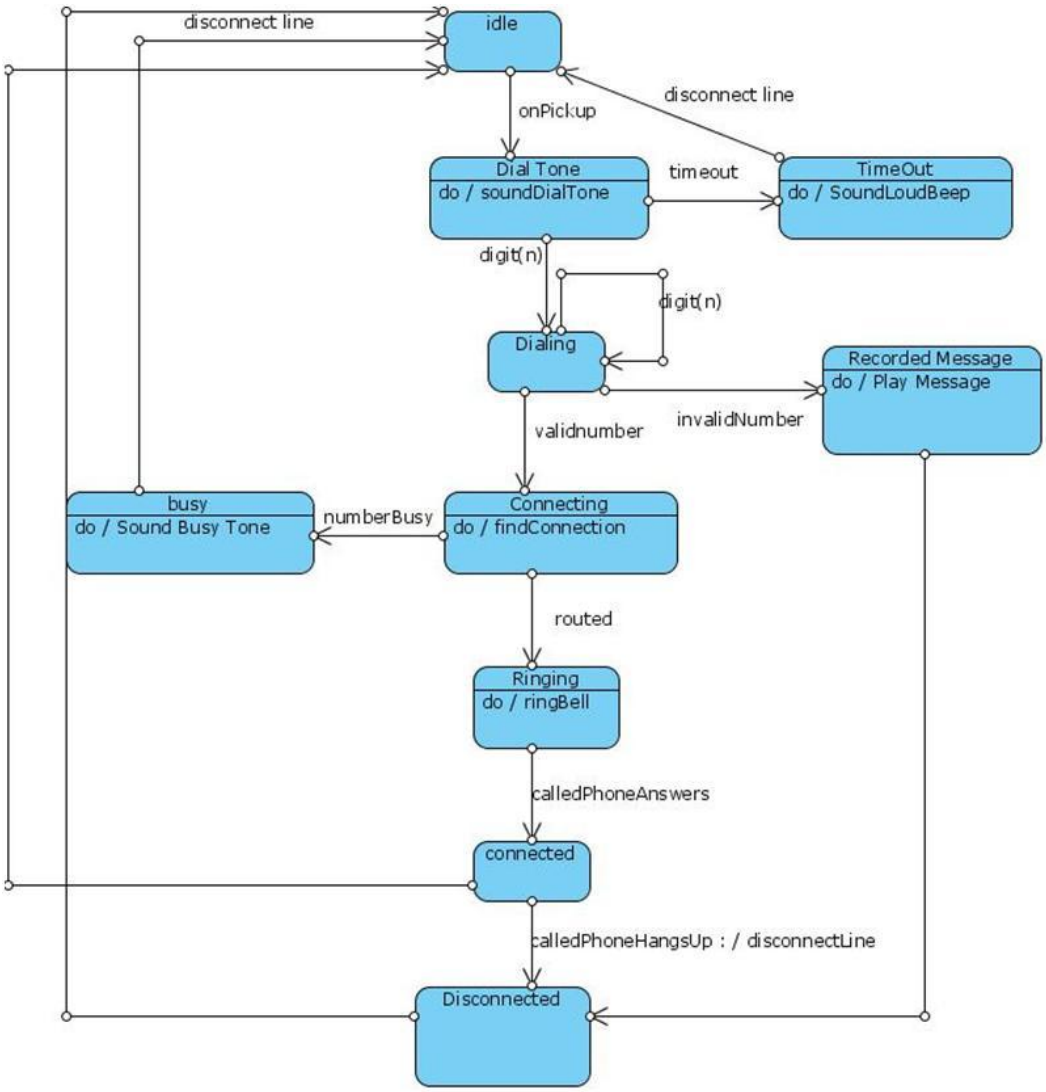
Sub:	Object-Oriented Modeling And Design Patterns							Code:	13MCA51
Date:	07-11-17	Duration:	90 mins	Max Marks:	50	Sem:	VA&B	Branch:	MCA

Answer any 5 questions. All questions carry equal marks.

		Marks
1.	a. Discuss the significance of state diagram and draw the neat state diagram for telephone line system.	[10]
2.	a. Describe the following terms with suitable examples: i. Multiplicity ii. Aggregation iii. Ordering iv. Bags and Sequence v. Association Classes	[5*2=10]
3.	a. Draw the use case diagram for library management system using include relation.	[6]
	b. Describe the guide lines for use case diagram.	[4]
4.	a. Write the guidelines for sequence diagram.	[2]
	b. Draw the sequence diagram for cash withdrawal process in ATM system	[8]
5.	a. Which approach performs the development Life cycle in strict sequence for the entire system. Explain in detail of that approach	[3]
	b. Explain about the well defined stages of software development stages.	[7]
6.	a. In the analysis stage, explain the steps to construct the static structure of the domain model in the real world system.	[10]
7.	a. What are the various stages of Object Oriented Methodology?	[5]
	b. Describe Object Oriented Themes	[5]
8.	a. What do you mean by event? Explain the several kinds of events with examples.	[6]
	b. Explain link, association and n-ary association with suitable examples.	[4]

1a.Discuss the significance of state diagram and draw the neat state diagram for telephone line system.

A state diagram, also called a state machine diagram or state chart diagram, is an illustration of the states an object can attain as well as the transitions between those states in the Unified Modeling Language (UML). In this context, a state defines a stage in the evolution or behavior of an object, which is a specific entity in a program or the unit of code representing that entity. State diagrams are useful in all forms of object-oriented programming (OOP). The concept is more than a decade old but has been refined as OOP modeling paradigms have evolved. A state diagram resembles a flowchart in which the initial state is represented by a large black dot and subsequent states are portrayed as boxes with rounded corners. There may be one or two horizontal lines through a box, dividing it into stacked sections. In that case, the upper section contains the name of the state, the middle section (if any) contains the state variables and the lower section contains the actions performed in that state. If there are no horizontal lines through a box, only the name of the state is written inside it. External straight lines, each with an arrow at one end, connect various pairs of boxes. These lines define the transitions between states. The final state is portrayed as a large black dot with a circle around it.



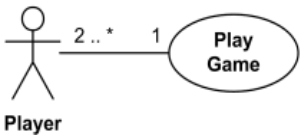
2. a. Describe the following terms with suitable examples: 5*2=10
i. Multiplicity ii. Aggregation iii. Ordering iv. Bags and Sequence
v. Association Classes

i) **Multiplicity** is a definition of **cardinality** - i.e. **number of elements** - of some collection of elements by providing an inclusive interval of non-negative integers to specify the allowable number of instances of described element. Multiplicity interval has some **lower bound** and (possibly infinite) **upper bound**:

multiplicity-range ::= [lower-bound '..'] upper-bound
lower-bound ::= natural-value-specification
upper-bound ::= natural-value-specification | '*'

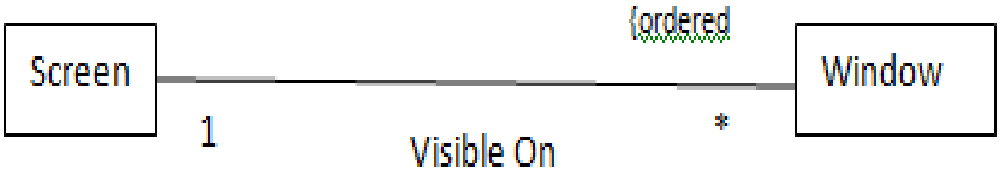
Lower and upper bounds could be natural constants or constant expressions evaluated to natural (non negative) number. Upper bound could be also specified as asterisk '*' which denotes unlimited number of elements. Upper bound should be greater than or equal to the lower bound.

Multiplicity	Option	Cardinality
0..0	0	Collection must be empty
0..1		No instances or one instance
1..1	1	Exactly one instance
0..*	*	Zero or more instances
1..*		At least one instance
5..5	5	Exactly 5 instances
m..n		At least m but no more than n instances



ii) Aggregation is a stronger form of association. An association is a link connecting two classes. In UML, a link is placed between the “whole” and the “parts” classes with a diamond head attached to the “whole” class to indicate that this association is an aggregation .Let's take an example of Department and teacher. A single teacher can not belong to multiple departments, but if we delete the department, the teacher object will not be destroyed. We can think about it as a “has-a” relationship.

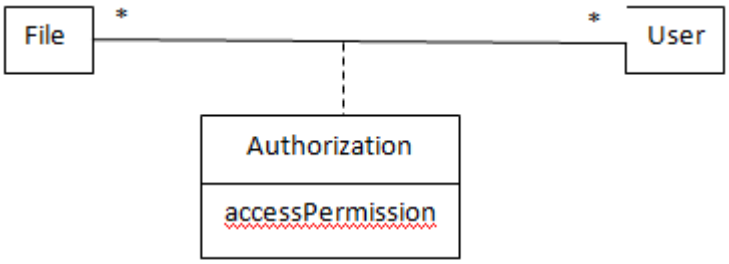
iii. Ordering is the objects on a "many" association end have explicit order and is an inherent part of the association. You can indicate an ordered set of objects by writing" {ordered}" next to the appropriate association end.



iv. Bags and Sequence : A bag is a collection of elements with duplicates allowed. A sequence is an ordered collection of elements with duplicates allowed.

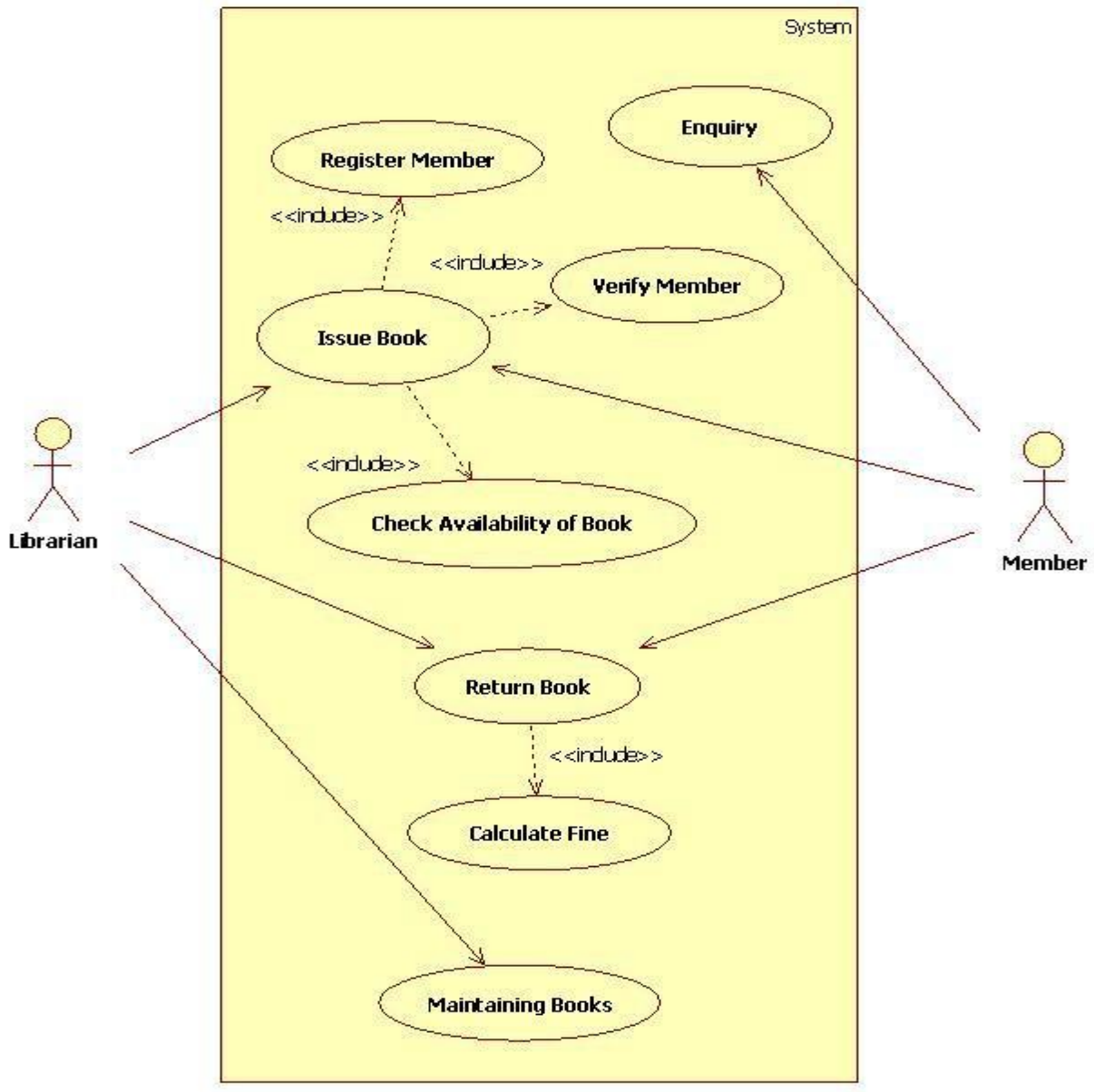
Figure 3.16 An example of a sequence. An itinerary may visit multiple airports, so you should use {sequence} and not {ordered}.

v. Association Classes : An association class is an association that is also a class. Like the links of an association, the instances of an association class derive identity from instances of the constituent classes. Like a class, an association class can have attributes and operations and participate in associations. You can find association classes by looking for adverbs in a problem statement or by abstracting known values.



/etc/termcap	Read	John
/etc/termcap	Read-write	Mary
/usr/doe/.login	Read-write	John

3a. Draw the use case diagram for library management system using include relation. [6]



3b. Describe the guide lines for use case diagram. [4]

First determine the system boundary. It is impossible to identify use cases or actors if the system boundary is unclear.

Ensure that actors are focused. Each actor should have a single, coherent purpose. If a real-world object embodies multiple purposes, capture them with separate actors. For example, the owner of a personal computer may install software, set up a database, and send email. These functions differ greatly in their impact on the computer system and the potential for system damage. They might be broken into three actors: *system administrator, database administrator,* and *computer user*. Remember that an actor is defined with respect to a system, not as a free-standing concept.

Each use case must provide value to users. A use case should represent a complete transaction that provides value to users and should not be defined too narrowly. For example, *dial a telephone number* is not a good use case for a telephone system. It does not represent a complete transaction of value by itself; it is merely part of the use case *make telephone call*. The latter use case involves placing the call, talking, and terminating the call. By dealing with complete use cases, we focus on the purpose of the functionality provided by the system, rather than jumping into implementation decisions. The details come later. Often there is more than one way to implement desired functionality.

Relate use cases and actors. Every use case should have at least one actor, and every actor should participate in at least one use case. A use case may involve several actors, and an actor may participate in several use cases.

Remember that use cases are informal. It is important not to be obsessed by formalism in specifying use cases. They are not intended as a formal mechanism but as a way to identify and organize system functionality from a user-centered point of view. It is acceptable if use cases are a bit loose at first. Detail can come later as use cases are expanded and mapped into implementations.

Use cases can be structured. For many applications, the individual use cases are completely distinct. For large systems, use cases can be built out of smaller fragments using relationships.

4a.Write the guidelines for sequence diagram [2]

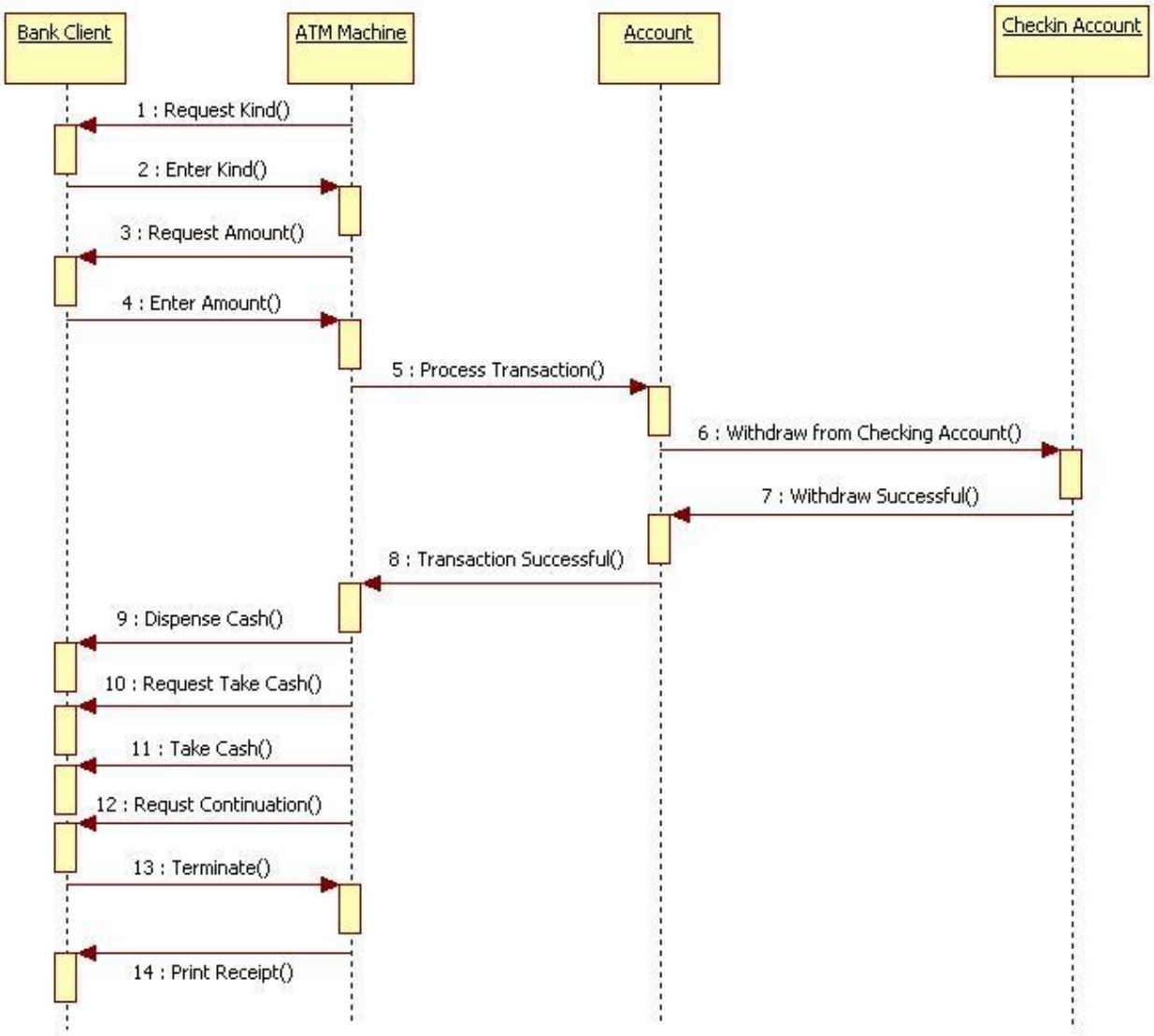
Prepare at least one scenario per use case. The steps in the scenario should be logical commands, not individual button clicks. Later, during implementation, you can specify the exact syntax of input. Start with the simplest mainline interaction—no repetitions, one main activity, and typical values for all parameters. If there are substantially different mainline interactions, write a scenario for each.

Abstract the scenarios into sequence diagrams. The sequence diagrams clearly show the contribution of each actor. It is important to separate the contribution of each actor as a prelude to organizing behavior about objects.

Divide complex interactions. Break large interactions into their constituent tasks and prepare a sequence diagram for each of them.

Prepare a sequence diagram for each error condition. Show the system response to the error condition.

4b.Draw the sequence diagram for cash withdrawal process in ATM system [8]



5a. Which approach performs the development Life cycle in strict sequence for the entire system. Explain in detail of that approach [3]

Waterfall Approach

The waterfall approach dictates that developers perform the software development stages in a rigid linear sequence with no backtracking. Developers first capture requirements, then construct an analysis model, then perform a system design, then prepare a class design, fol-

lowed by implementation, testing, and deployment. Each stage is completed in its entirety before the next stage is begun.

The waterfall approach is suitable for well-understood applications with predictable outputs from analysis and design, but such applications seldom occur. Too many organizations attempt to follow a waterfall when requirements are fluid. This leads to the familiar situation where developers complain about changing requirements, and the business complains about inflexible software development. A waterfall approach also does not deliver a useful system until completion. This makes it difficult to assess progress and correct a project that has gone awry.

5b.Explain about the well defined stages of software development stages. [7]

System conception. Conceive an application and formulate tentative requirements.

Analysis. Deeply understand the requirements by constructing models. The goal of analysis is to specify *what* needs to be done, not *how* it is done. You must understand a problem before attempting a solution.

System design. Devise a high-level strategy—the architecture—for solving the application problem. Establish policies to guide the subsequent class design.

Class design. Augment and adjust the real-world models from analysis so that they are amenable to computer implementation. Determine algorithms for realizing the operations.

Implementation. Translate the design into programming code and database structures.

Testing. Ensure that the application is suitable for actual use and that it truly satisfies the requirements.

Training. Help users master the new application.

Deployment. Place the application in the field and gracefully cut over from legacy applications.

Maintenance. Preserve the long-term viability of the application.

6a In the analysis stage, explain the steps to construct the static structure of the domain model in the real
. world system. [10M]

Find classes
Prepare a data dictionary
Find associations
Find attributes of objects and links
Organize and simplify classes using inheritance
Verify that access paths exists for likely queries
Iterate and refine the model
Reconsider the level of abstraction
Group classes into packages.

Finding Classes:- Classes often correspond to nouns for example . In the statement “a reservation system to sell tickets to performances at various theaters tentative classes would be Reservation , System , Ticket ,Performances and Theater
Keeping the right classes:
We need to discard the unnecessary and incorrect classes:

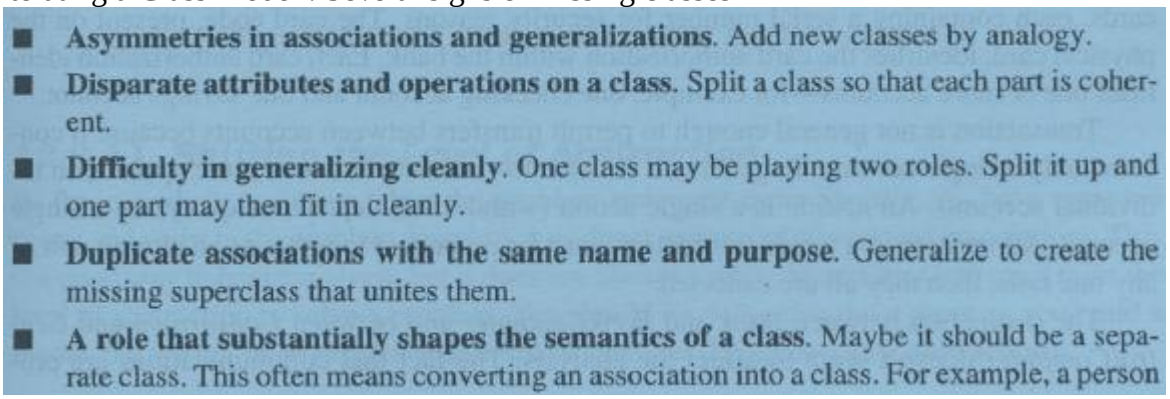
Redundant classes: if two classes express same
Irrelevant classes: It has little or nothing to do with the problem
Vague class: too broad in scope
Attributes: describe individual objects
Operations
Roles
Implementation constructs: CPU, subroutine, process and algorithm
Derived classes: omit class that can be derived from other class

Prepare data dictionary: Information regarding the data is maintained
Finding Associations
Keeping the right associations:
1.Associations between eliminated classes
2.Irrelevant or implementation associations
3.Actions
4.Ternary association
5.Derived Association
6.Misnamed associations
7.Association end names
8.Qualified associations
9.Multiplicity
10.Missing Associations
11.Aggregation

Finding attributes:
Keeping the right attributes:
Refining with inheritance :

1. Bottom-up generalization
2. Top-down generalization
3. Generalization vs enumeration
4. Multiple inheritance
5. Similar associations.
6. Adjusting the inheritance level

Testing Access paths
Iterating a Class Model : Several signs of missing classes



- **Asymmetries in associations and generalizations**. Add new classes by analogy.
- **Disparate attributes and operations on a class**. Split a class so that each part is coherent.
- **Difficulty in generalizing cleanly**. One class may be playing two roles. Split it up and one part may then fit in cleanly.
- **Duplicate associations with the same name and purpose**. Generalize to create the missing superclass that unites them.
- **A role that substantially shapes the semantics of a class**. Maybe it should be a separate class. This often means converting an association into a class. For example, a person

Shifting the level of abstraction
Grouping Classes into packages.

7a. What are the various stages of Object Oriented Methodology? [5M]

• **System conception**. Software development begins with business analysts or users conceiving an application and formulating tentative requirements.

• **Analysis.** The analyst scrutinizes and rigorously restates the requirements from system conception by constructing models. The analyst must work with the requestor to understand the problem, because problem statements are rarely complete or correct. The analysis model is a concise, precise abstraction of what the desired system must do, not how it will be done. The analysis model should not contain implementation decisions.

The analysis model has two parts: the domain model, a description of the real-world objects reflected within the system; and the application model, a description of the parts of the application system itself that are visible to the user.

Application objects might control the execution of trades and present the results. Application experts who are not programmers can understand and criticize a good model.

System design. The development team devise a high-level strategy-the system architecture- for solving the application problem. They also establish policies that will serve as a default for the subsequent, more detailed portions of design. The system designer must decide what performance characteristics to optimize, choose a strategy of attacking the problem, and make tentative resource allocations. For example, the system designer might decide that changes to the workstation screen must be fast and smooth, even when windows are moved or erased, and choose an appropriate communications protocol and memory buffering strategy.

• **Class design.** The class designer adds details to the analysis model in accordance with the system design strategy. The class designer elaborates both domain and application objects using the same OO concepts and notation, although they exist on different conceptual planes. The focus of class design is the data structures and algorithms needed to implement each class. For example, the class designer now determines data structures and algorithms for each of the operations of the Window class.

• **Implementation.** Implementers translate the classes and relationships developed during class design into a particular programming language, database, or hardware. Programming should be straightforward, because all of the hard decisions should have already been made. During implementation, it is important to follow good software engineering practice so that traceability to the design is apparent and so that the system remains flexible and extensible. For example, implementers would code the Window class in a programming language, using calls to the underlying graphics system on the workstation.

7b.Describe Object Oriented Themes [5]

1 Abstraction

Abstraction focus on essential aspects of an application while ignoring details. This means focusing on what an object is and does, before deciding how to implement it. Use of abstraction preserves the freedom to make decisions as long as possible by avoiding premature commitments to details. Most modem languages provide data abstraction, but inheritance and polymorphism add power. The ability to abstract is probably the most important skill required for 00 development.

2 Encapsulation

Encapsulation (also information hiding) separates the external aspects of an object, that are accessible to other objects, from the internal implementation details, that are hidden from other objects. Encapsulation prevents portions of a program from becoming so interdependent that a small change has massive ripple effects. You can change an object's implementation without affecting the applications that use it. You may want to change the implementation of an object to improve performance, fix a bug, consolidate code, or support porting. Encapsulation is not unique to 00 languages, but the ability to combine data structure and behavior in a single entity makes encapsulation cleaner and more powerful things prior languages, such as Fortran, Cobol, and C.

3.Combining Data and Behavior

The caller of an operation need not consider how many implementations exist. Operator polymorphism shifts the burden of deciding what implementation to use from ilk calling code to the class hierarchy. For example, non-OO code to display the contents of a window must distinguish the type of each figure, such as polygon, circle, or text, and call the appropriate procedure to display it. An 00 program would simply invoke the draw operation on

each figure; each object implicitly decides which procedure to use, based on its class. Maintenance is easier, because the calling code need not be modified when a new class is added.

4.Sharing

00 techniques promote sharing at different levels. Inheritance of both data structure and behavior lets subclasses share common code. This sharing via inheritance is one of the main advantages of 00 languages. More important than the savings in code is the conceptual clarity from recognizing that different operations are ail really the same thing. This reduces the number of distinct cases that you must understand and analyze.00 development not only lets you share information within an application, but also offers the prospect of reusing designs and code on future projects. 00 development provides the tools, such as abstraction, encapsulation, and inheritance, to build libraries of reusable components. Unfortunately, reuse has been overemphasized as a justification for 00 technology.

5 Emphasis the Essence of an Object

00 technology stresses what an object is, rather than how it is used. The uses of an object depend on the details of the application and often change during development. As requirements evolve, the features supplied by an object are much more stable than the ways it is used, hence software systems built on object structure are more stable in the long run. 00 development places a greater emphasis on data structure and a lesser emphasis on procedure structure than functional-decomposition methodologies.

6 Synergy

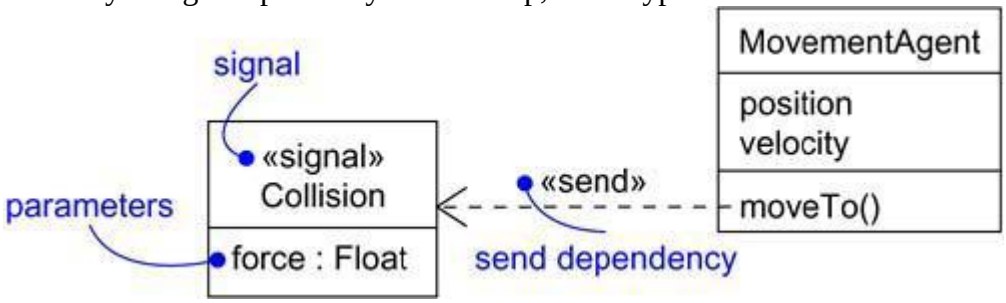
Identity, classification, polymorphism, and inheritance characterize 00 languages. Each of concepts can be used in isolation, but together they complement each other. The benefits of an OQ-approach are greater than they might seem at first. The emphasis on the essential properties of an object forces the developer to think more carefully deeply about what an object IS and does. The resulting system tends to be cleaner, more general, and more robust than it would be if the emphasis were only on the use of data and operations.

8a.What do you mean by event? Explain the several kinds of events with examples. [6]

An **event** is an occurrence at a point in time, such as user depresses left button of mouse. An event happens instantaneously with regard to the time scale of an application. Events cause state changes which is shown in State Diagrams

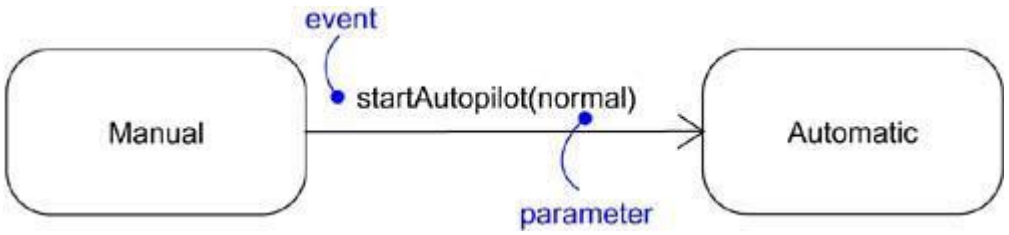
Signal Event

- a signal event represents a named object that is dispatched (thrown) asynchronously by one object and then received (caught) by another. Exceptions are an example of internal signal
- a signal event is an asynchronous event
- signal events may have instances, generalization relationships, attributes and operations. Attributes of a signal serve as its parameters
- A signal event may be sent as the action of a state transition in a state machine or the sending of a message in an interaction
- signals are modeled as stereotyped classes and the relationship between an operation and the events by using a dependency relationship, stereotyped as send



Call Event

- a call event represents the dispatch of an operation
- a call event is a synchronous event



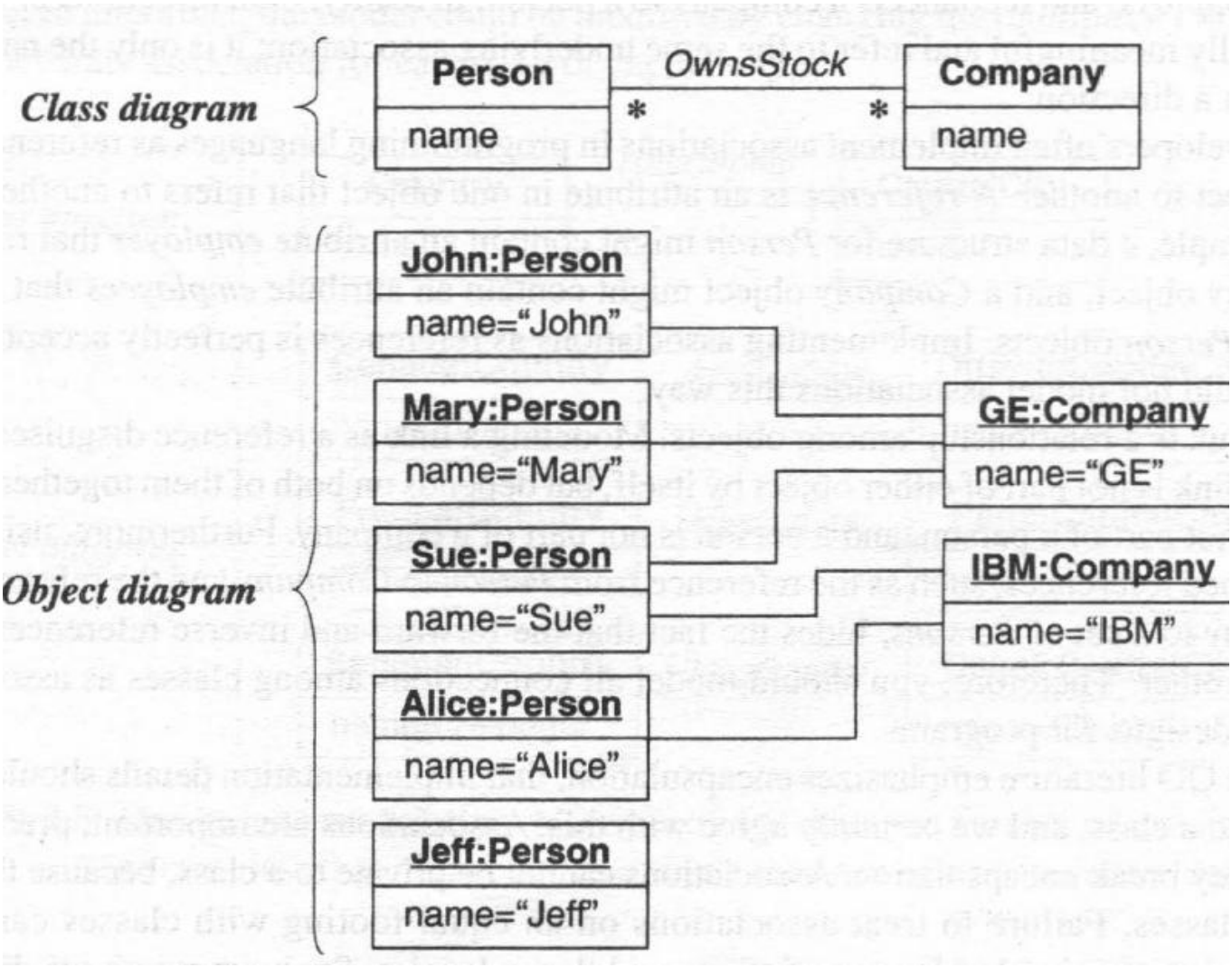
Time and Change Events

- A *time event* is an event that represents the passage of time.
- modeled by using the keyword 'after' followed by some expression that evaluates to a period of time which can be simple or complex.
- A *change event* is an event that represents a change in state or the satisfaction of some condition
- modeled by using the keyword 'when' followed by some Boolean expression

8b.Explain link, association and n-ary association with suitable examples. [4]

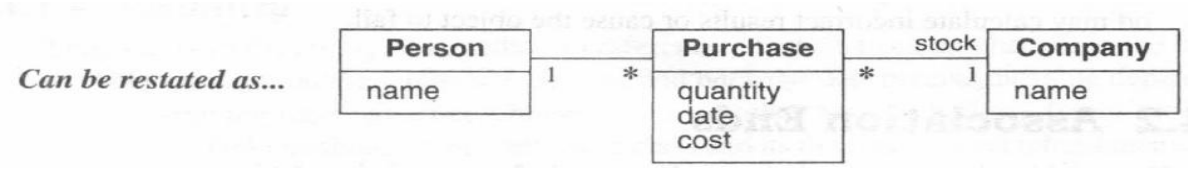
A link is a physical or conceptual connection among objects. For example, Joe Smith Works-For Simplex company. Most links relate two objects, but some links relate three or more objects we define a link as a tuple -that is, a list of objects. A link is an instance of an association.

An association is a description of a group of links with common structure and common semantics. For example, a person Works For a company. The links of an association connect objects from the same classes. An association describes a set of potential links in the same way that a class describes a set of potential objects. Links and associations often appear as verbs in problem statements.



N-ary associations (associations among three or more classes.)

You should try to avoid n-ary associations-most of them can be decomposed into binary associations, with possible qualifiers and attributes. Figure 4.5 shows an association that at first glance might seem to be an n-ary but can readily be restated as binary associations.



Restating an n-ary association. You can decompose most n-ary associations into binary associations....

This n-ary association is an atomic unit and cannot be subdivided into binary associations without losing information. A programmer may know a language and work on a project, but might not use the language on the project. The UML symbol for n-ary associations is a diamond with lines connecting to related classes. If the association has a name, it is written in italics next to the diamond.