

# **Object-Oriented System Analysis and Design**

[ITec 3037](#)

# Objectives of the course

## After the course, students will

- be able to **use** an object-oriented method for analysis and design.
- be able to analyze information systems in **real-world settings** and to conduct methods such as interviews and observations.
- know techniques aimed to achieve the objective and expected results of a systems development process.
- know different types of **prototyping**.
- know how to use **UML** for notation.

# Course Outline

## Chapter one: Introduction

- Definitions and basic concepts
- Structured Vs OO approaches
- Basic Characteristics of Object-Oriented System
- Benefits of Object - Oriented Systems Analysis and Design
- Object Oriented system development methodologies
- Why an object oriented overview of unified approach

## Chapter Two - Modeling using UML

- **An overview of UML**  
Where can the UML be used?
- **Building blocks**
  - **Things**
  - **Relationships**
  - **Diagrams**
    - Use case diagram
    - Class diagram
    - Sequence diagram
    - State diagram
    - Activity diagram
    - Deployment diagram

## **Chapter Three - Requirement elicitation**

- An overview of requirements elicitation.
- Requirements elicitation concepts
- Requirements elicitation activities.
- Managing requirements elicitation

## **Chapter Four - Requirement Analysis**

- An Overview of Analysis.
- Analysis Activities: From Use Cases to Objects
- Analysis Concepts

## **Chapter Five - System and object design**

- An overview of system design.
- System design concepts
- System design activities: From objects to subsystems

## **Chapter Six- Quality assurance/Testing**

- An overview of testing
- Testing concepts
- Testing activities
- Managing testing
- Impact of object oriented testing

# Chapter One

## Introduction

# Definitions and basic concepts

## What is system?

A **system** is an organized set of communicating parts designed for a specific purpose.

**A car:** composed of four wheels, a chassis, a body, and an engine, is designed to transport people.

**Payroll system:** A payroll system, computer, printers, disks, software, and the payroll staff, is designed to issue salary checks for employees of a company.

**A system** can be seen as a computer *programs* and *associated documentation* such as requirements, design models and *user* manuals.

## Example

- Traffic management system
- Automatic library system
- Human resources information system.
- Hotel Management system

# Definitions and basic concepts

System Products–based on **development target**.

**Generic:** developed to be sold to a range of different customers

e.g. Microsoft Excel or Microsoft Word.

**Bespoke** (custom or especially made) - developed for a single customer according to their specification.

**Example**

School/University Systems, Hospital system, Hotel System, POSs, Accounting system, Inventory system, Billing System, etc.

# Attributes of good system

## **Maintainability**

- Software must evolve (progress) to meet changing needs.

## **Dependability**

- Software must be trustworthy (upright).

## **Efficiency**

- Software should not make wasteful (uneconomical) use of system resources

## **Acceptability** (meet users requirement)

- Software must be accepted by the users for which it was designed.
- This means it must be understandable, usable and compatible with other systems.



# Attributes of good system

## Usability

- Users can **learn** it fast and get their job done easily

## Reliability

- It does what it is required to do without failing (no failure)

## Reusability

- Its parts can be used in other projects, so reprogramming is not needed

# Quality software?

## Customer

solves problems at  
an acceptable cost in  
terms of money paid and  
resources used

## User

easy to learn;  
efficient to use;  
helps get work done



## Developer

easy to design;  
easy to maintain;  
easy to reuse its parts

## Development manager

sells more and  
pleases customers  
while costing less  
to develop and maintain

# System Development - SDLC

Any system development has several clearly defined phases - the stable SDLC.

This remain scientific in any system development approaches.

**Planning/R-elicitation** - what does the user want?

**Systems analysis:** Understanding/ documenting requirements

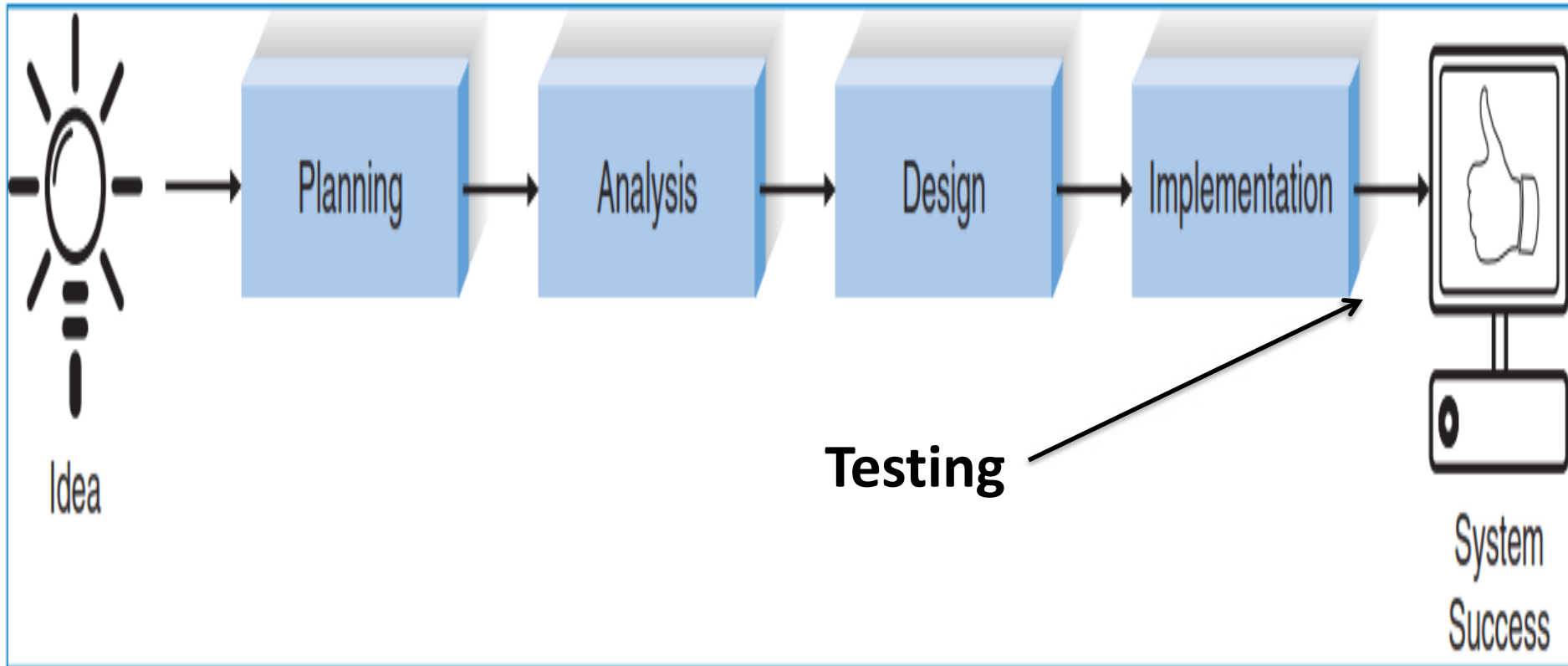
**Design** : Planning a possible solution

**Implementation** : Building a solution

**Testing:** the implementation Ensuring it meets requirements

- The Systems Development Life Cycle (SDLC) provides the **foundation** for the processes used to develop a system.

# SDLC



# System Development Process

- A process used to create a software system consists of:
    - **Steps**  
A sequence of **step-by-step approaches** that help develop the information system
    - **Techniques**  
**Processes** that the analyst follows to ensure thorough, complete and comprehensive analysis and design.
    - **Tools**  
Computer programs that aid in applying techniques.  
The combination of these three forms a methodology.
- Tools + steps + techniques = Methodology**

# What is Methodology?

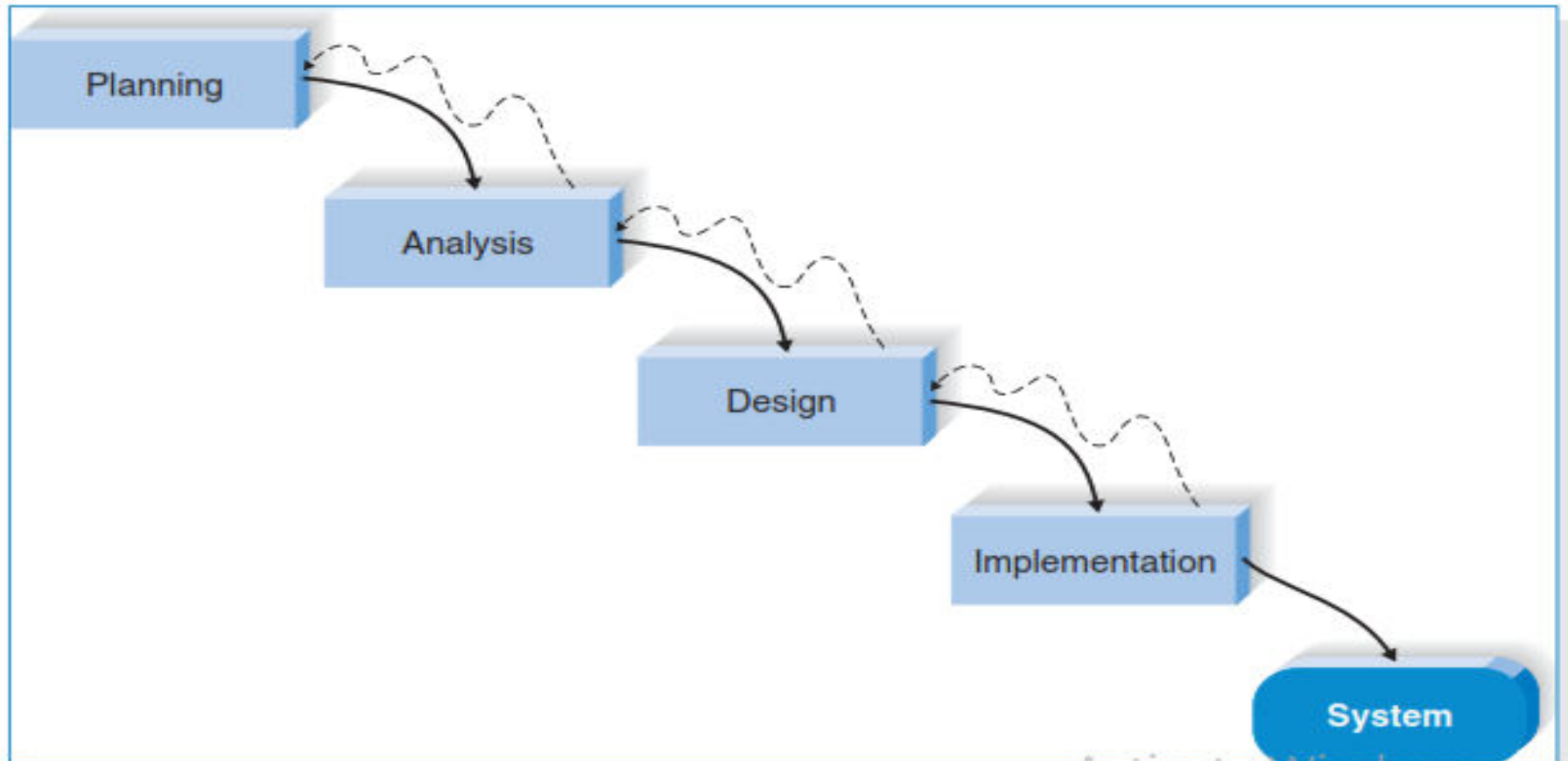
A methodology is a **formalized approach** to implementing the SDLC (i.e., it is a list of steps and deliverables).

Ex.

- Waterfall Development Methodology
- Parallel development methodology
- Agile Software Development Methodology
- Rapid Application Development (RAD)
- V-model development methodology
- Dynamic System Development Model Methodology
- Spiral Model ....

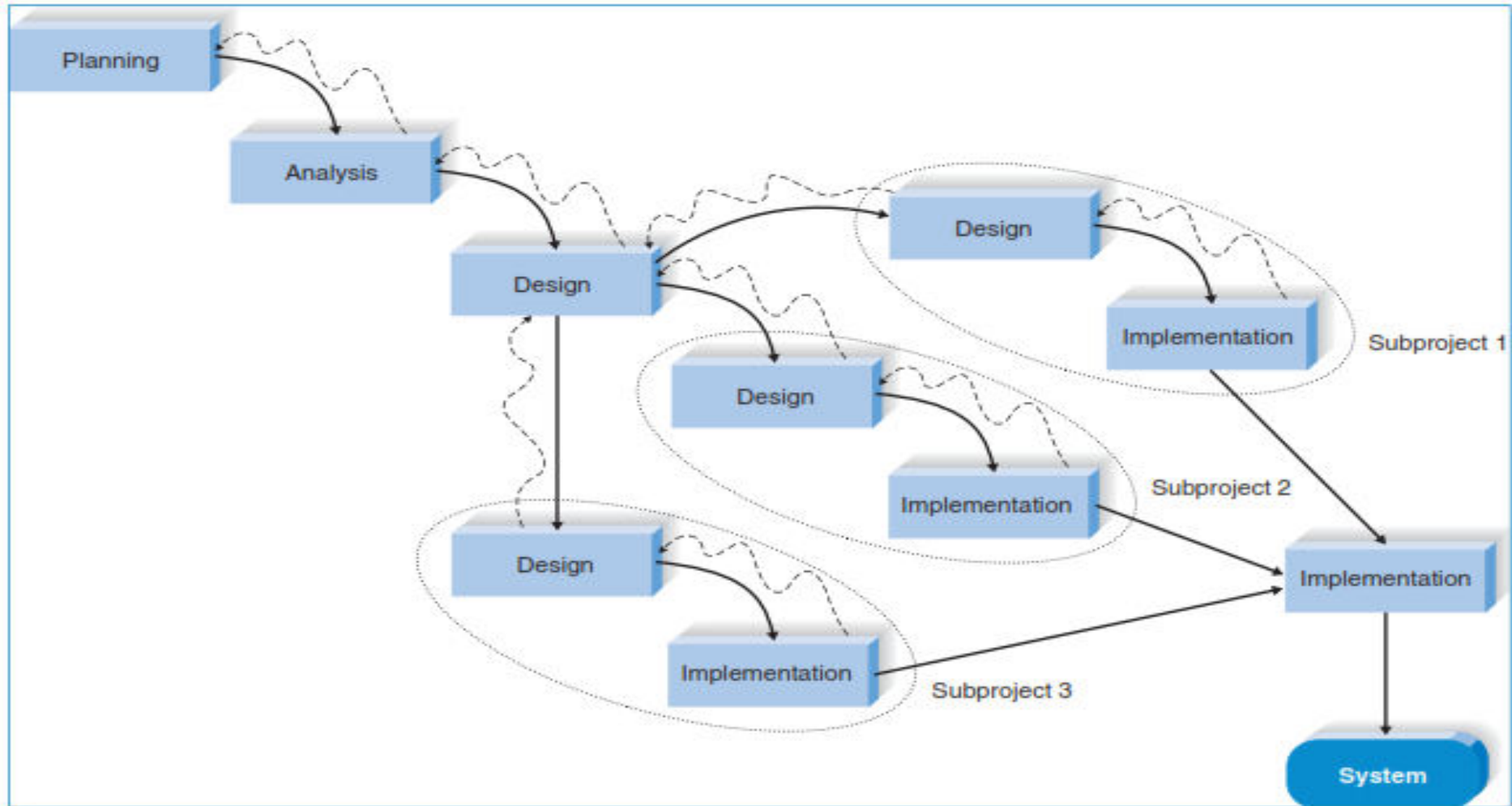
# Waterfall methodology

Is a classical model used in system development life cycle to create a system with a **linear and sequential** approach.



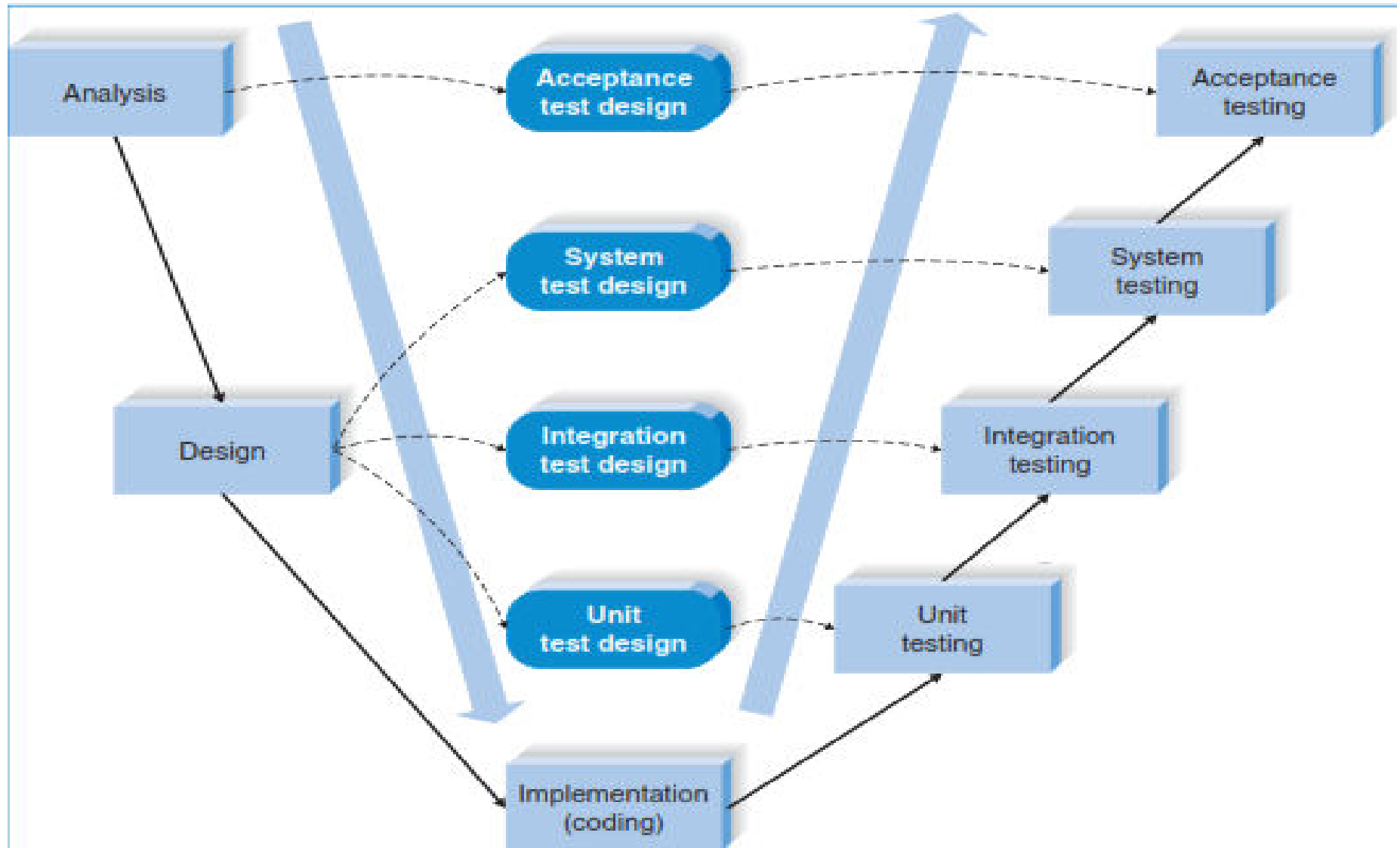
# Parallel development methodology

occurs when multiple developers check out their own working versions from the same object.





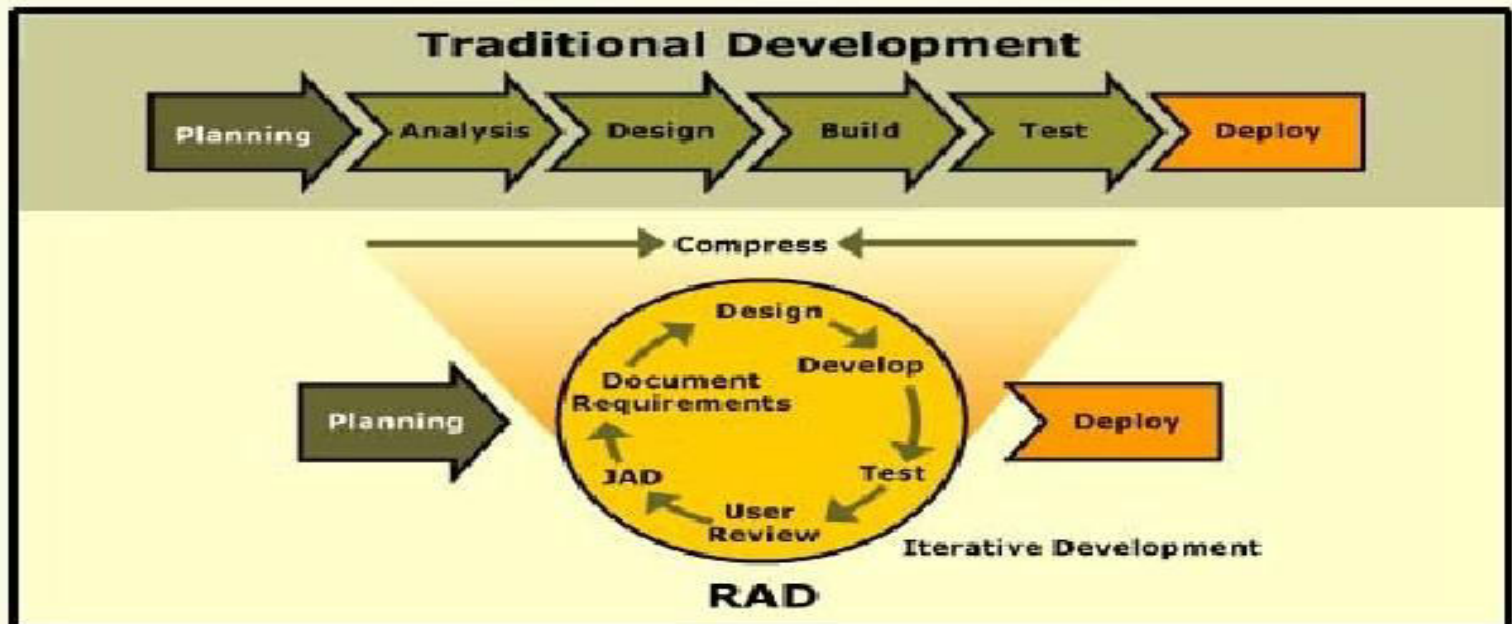
# V-model development methodology



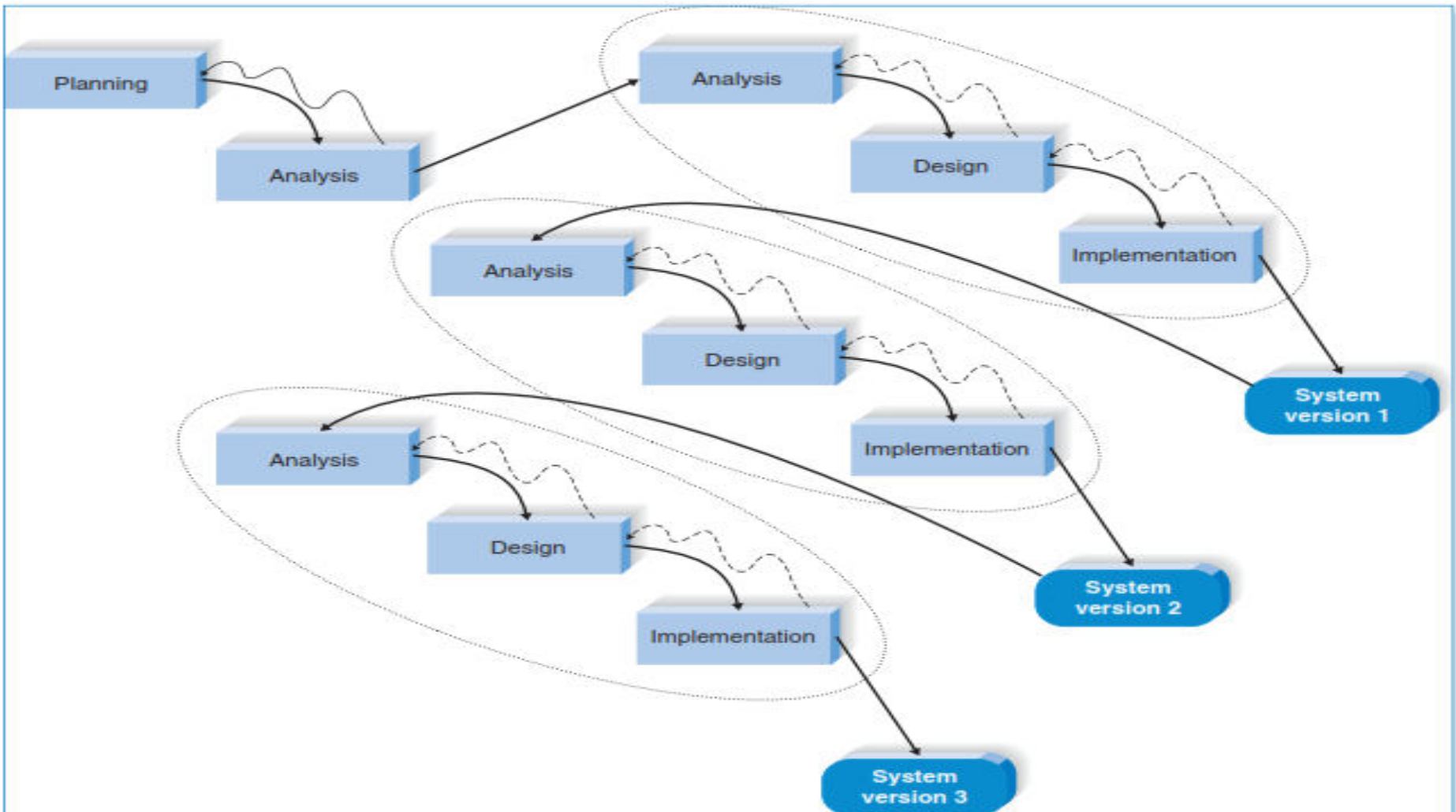
# Rapid application Development

may be conducted in a variety of ways **Iterative development**, **System prototyping** and **Throwaway prototyping** are the majors to be mentioned.

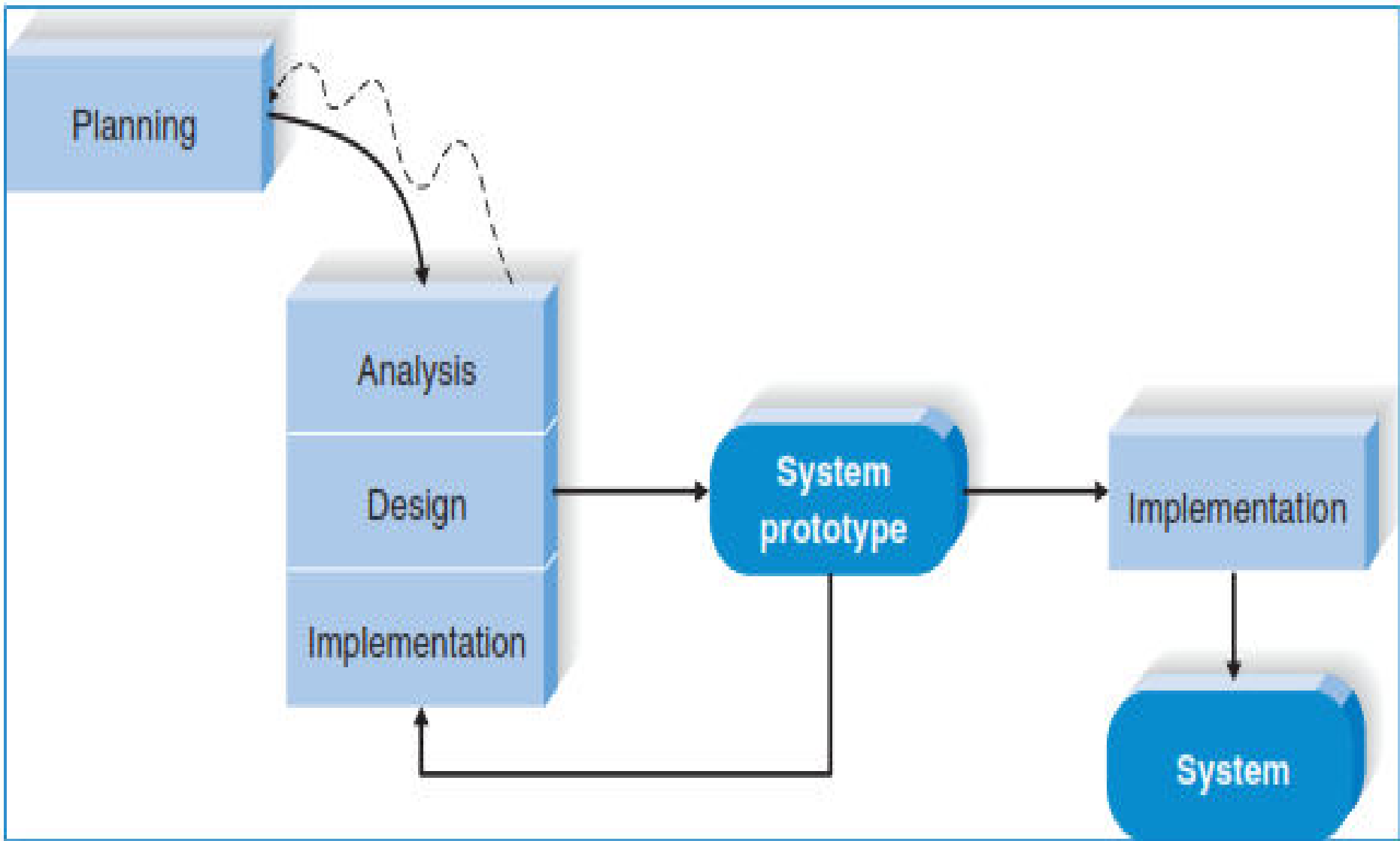
## RAD development cycle



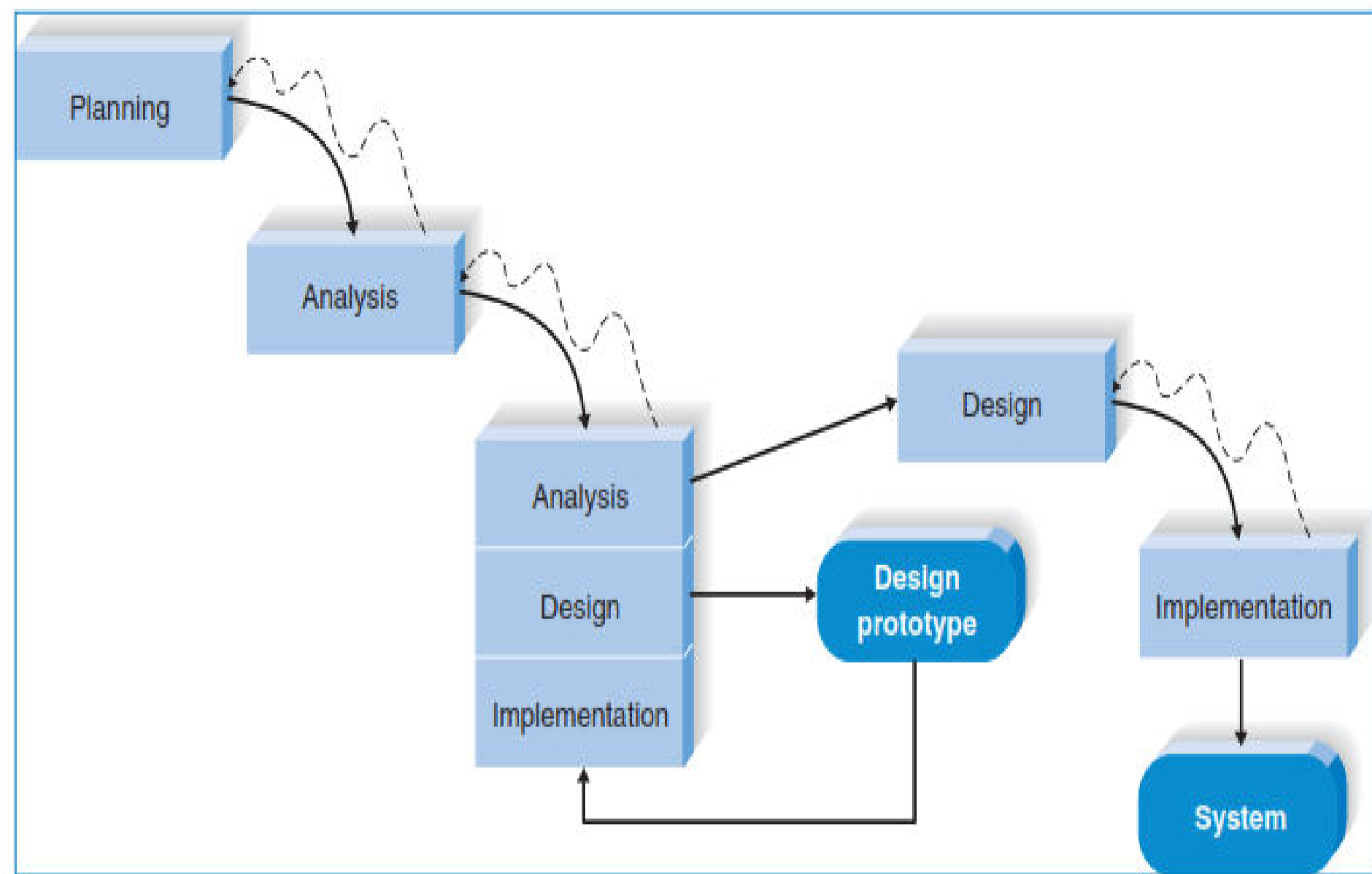
# I. Iterative development



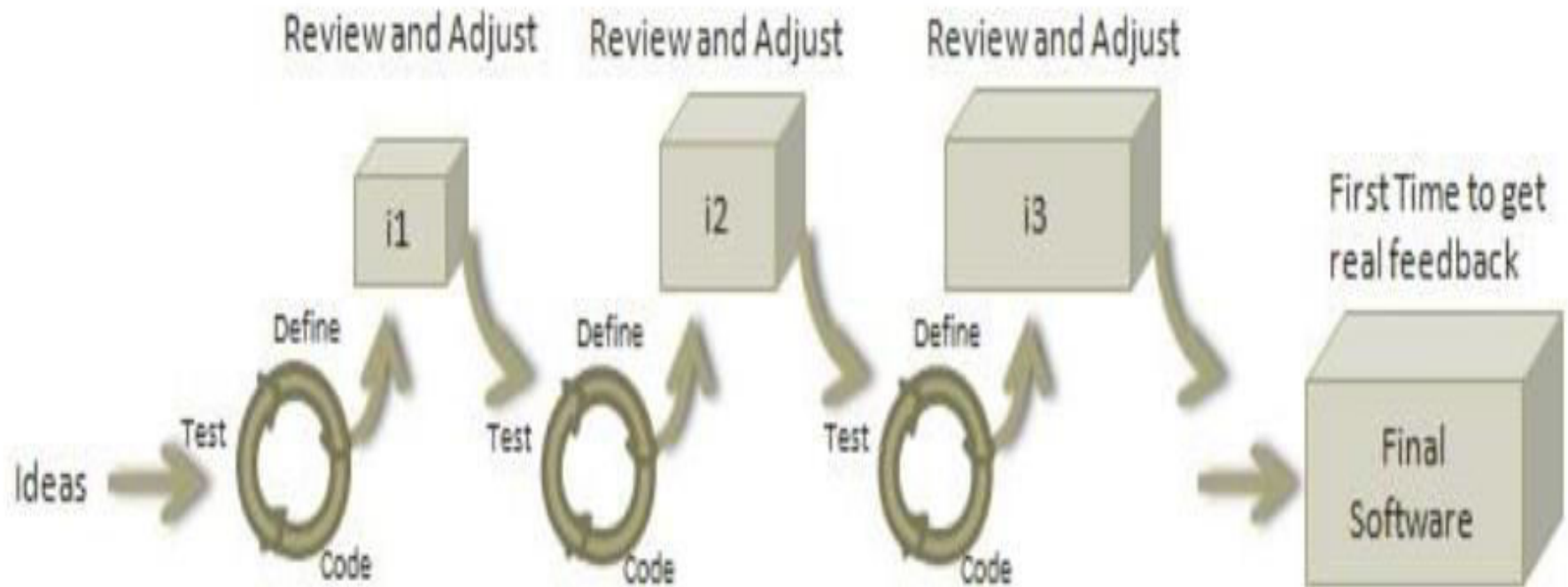
## ii. Prototype Methodology



### iii. Throwaway prototyping



# Agile development methodology



# Assignment I (5 )

1. List and discuss at least five most widely used software development methodologies.
2. Compare these five software development methodologies.
3. If you are going to develop a software which one will you use among the five methodologies. Why?

# Major Approaches to system development

- **Structured approach (classic)**
  - Modeling process and data **separately**
- **Object oriented approach**
  - Is a software development strategy based on the idea of building systems/software from **reusable components** called **objects**
  - Objects are identified having data and function/process together



# Structured Vs O-O approaches

## i. Structured approaches (paradigm)

- Are either data-centric or process-centric
- Modeling process and data separately
- Mostly Suitable for small sized software
- This approach is associated with structured Programming

# Structured programming

- is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program by making extensive use of subroutines, block structures, for and while loops
- (sometimes known as *modular programming*) is a subset of **procedural programming** that enforces a logical structure on the program being written to make it more efficient and easier to understand and modify.

# Structured programming

- frequently employs a top-down design model, in which developers map out the overall program structure into separate subsections. A defined function or set of similar functions is coded in a separate **module or sub module**.
- After a module has been tested individually, it is then integrated **with other modules** into the overall program structure.

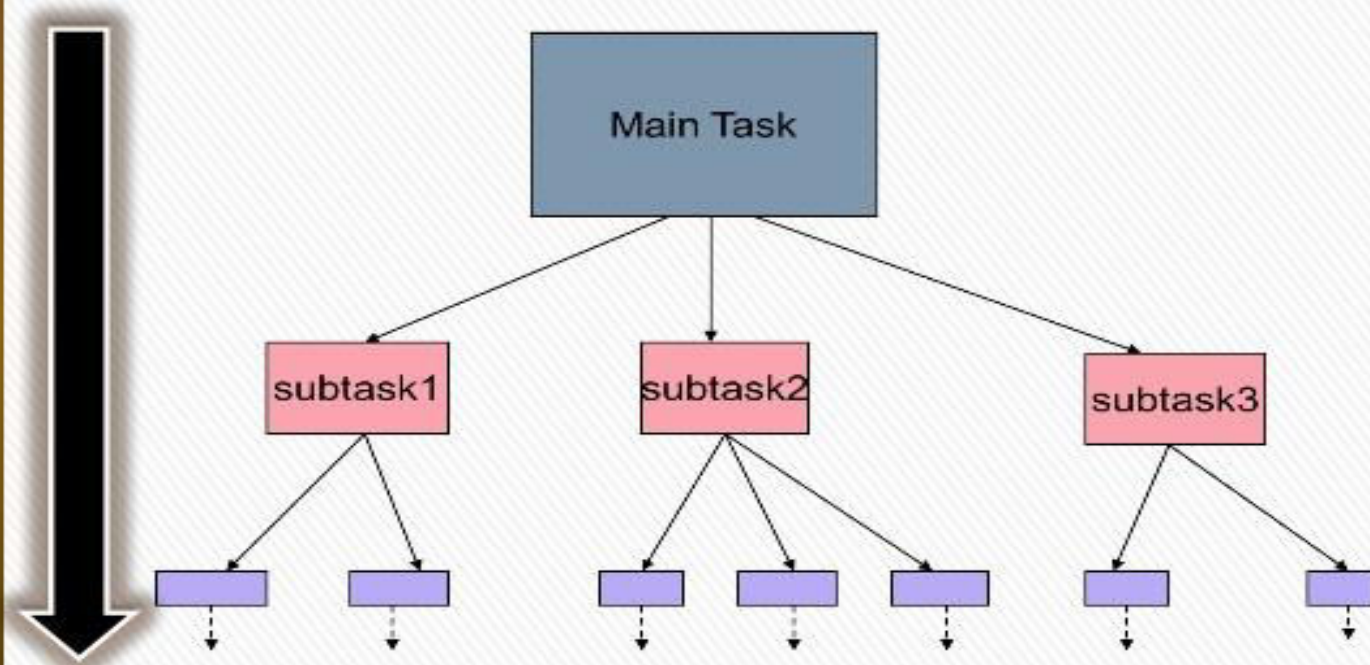
...

## **Example: C language**

C is called a structured programming language because to solve a large problem, C programming language divides the problem into smaller modules called functions or procedures each of which handles a particular responsibility.

# Top Down Design

## Top Down Design



## ... Structured approaches

- Various tools and techniques are used for this system development approach
- **Process Modeling**
  - Data Flow Diagrams (DFD)
  - Data Dictionary
  - Decision Trees
  - Decision Tables
  - Pseudo code
- Entity Relationship Diagrams – ERD (**Data modeling**)

## ii. Object oriented approaches (paradigm)

- The field of system analysis and design still has a lot of room for improvement: Projects are still failing. Thus, the state of the systems analysis and design field is on **constant transition and continuous improvement**.
- Today, an exciting enhancement to systems analysis and design is the application of the *object-oriented approach*.

# Object-Oriented approach

- It views a system as a collection of self-contained objects, including both data and processes or it is an approach - in which all computations are performed in the context of **objects**.
- In object-oriented approach a running program is seen as a collection of objects collaborating to perform a given task, in this approach:
  - Procedures are De - emphasized
  - A system is made up of objects
  - Users can more easily understand objects



# Object-oriented approach

- For system development using this approach the assumption is :
  - Object Oriented Analysis, design ()
  - Object oriented programming (C++, Java, C#, etc.)
  - Object Oriented CASE tools are expected to be used. (*using COOAD (CASE tool for Object-Oriented Analysis and Design) in order to examine the effectiveness of our proposition. COOAD supports object-oriented analysis and design, verification of the analysis and design, and generation of code.*)

# Basic Characteristics of Object-Oriented System

- It is based on **Object-Oriented Theory**
- Object Oriented Theory is built up on a sound engineering foundation whose elements we collectively called the **object model**.
- The Object model encompasses the following principles.

## 1. Abstraction

Denotes essential characteristics of an object that distinguishes it from all other kinds of objects

## 2. Encapsulation

Hiding the inner workings of object's operations from the outside world and from other objects

# .... Characteristics of Object-Oriented

## 3. Modularity

- The property of a system that has been decomposed in to a set of cohesive and loosely coupled modules

## 4. Hierarchy - Is a ranking or ordering of abstractions

- ✓ Inheritance
- ✓ Aggregation - The process of creating a new object from two or more other objects.

## 5. Other Concepts

- Objects, Classes, Polymorphism, Message,

# The major characteristics of object-oriented SAD

## A. Classes, Objects and Attributes

- A *class* is the general template we use to define and create specific *instances*, or objects. Every object is associated with a class.

e.g.

Book, Student, Librarian, Teacher, Staff .....

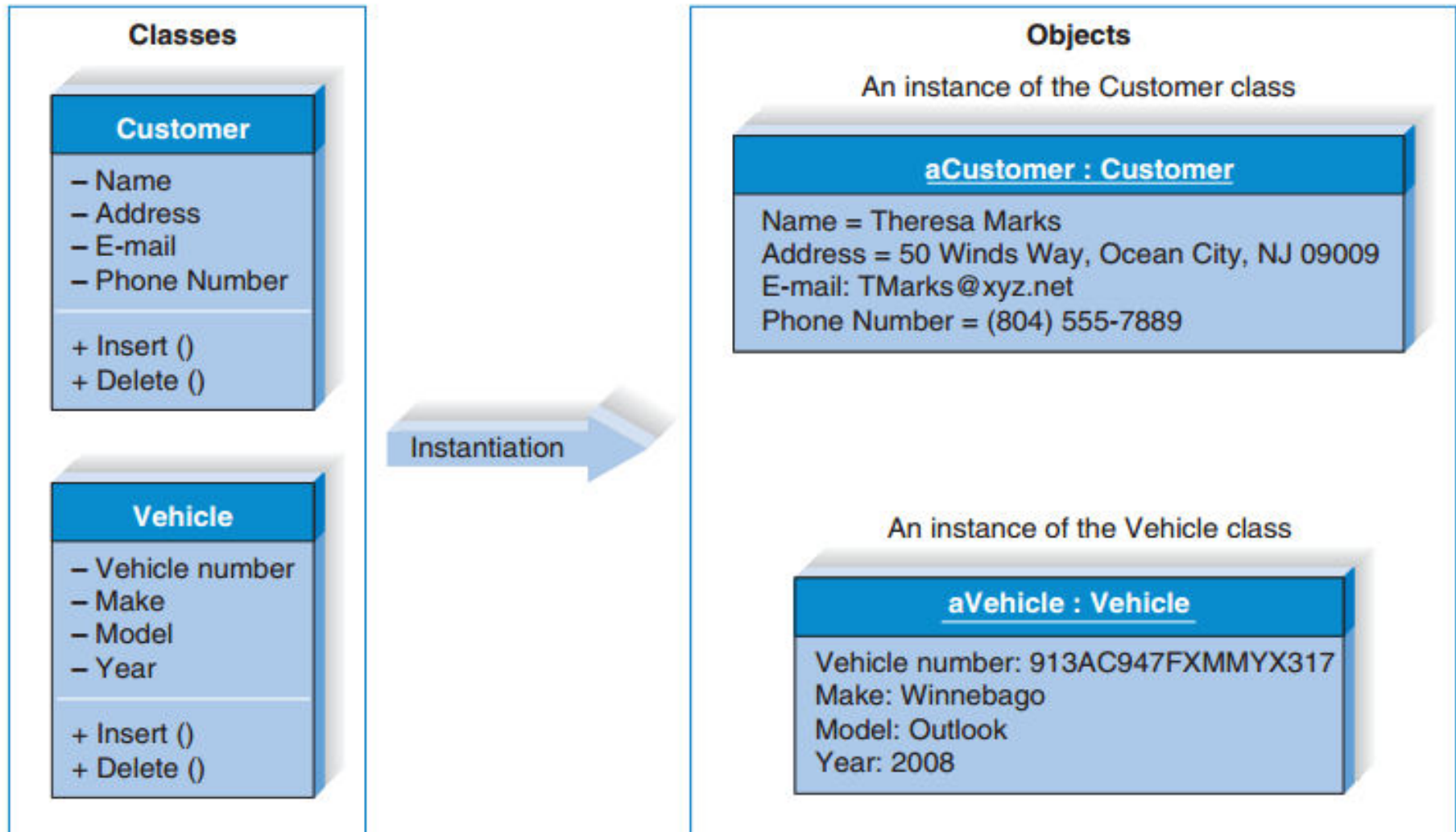
- An *object* is an instantiation of a class. In other words, an object is a person, place, event, or thing about which we want to capture information.

e.g. a specific book, a specific student .....

- Each object has *attributes* that describe information about the object. The state of an object is defined by the value of its attributes and its relationships with other objects at a particular point in time.

e.g. a student is described by his name, address, sex....

# ... major characteristics of O-O SAD



# ... major characteristics of o-o SAD

## b. Methods and Messages

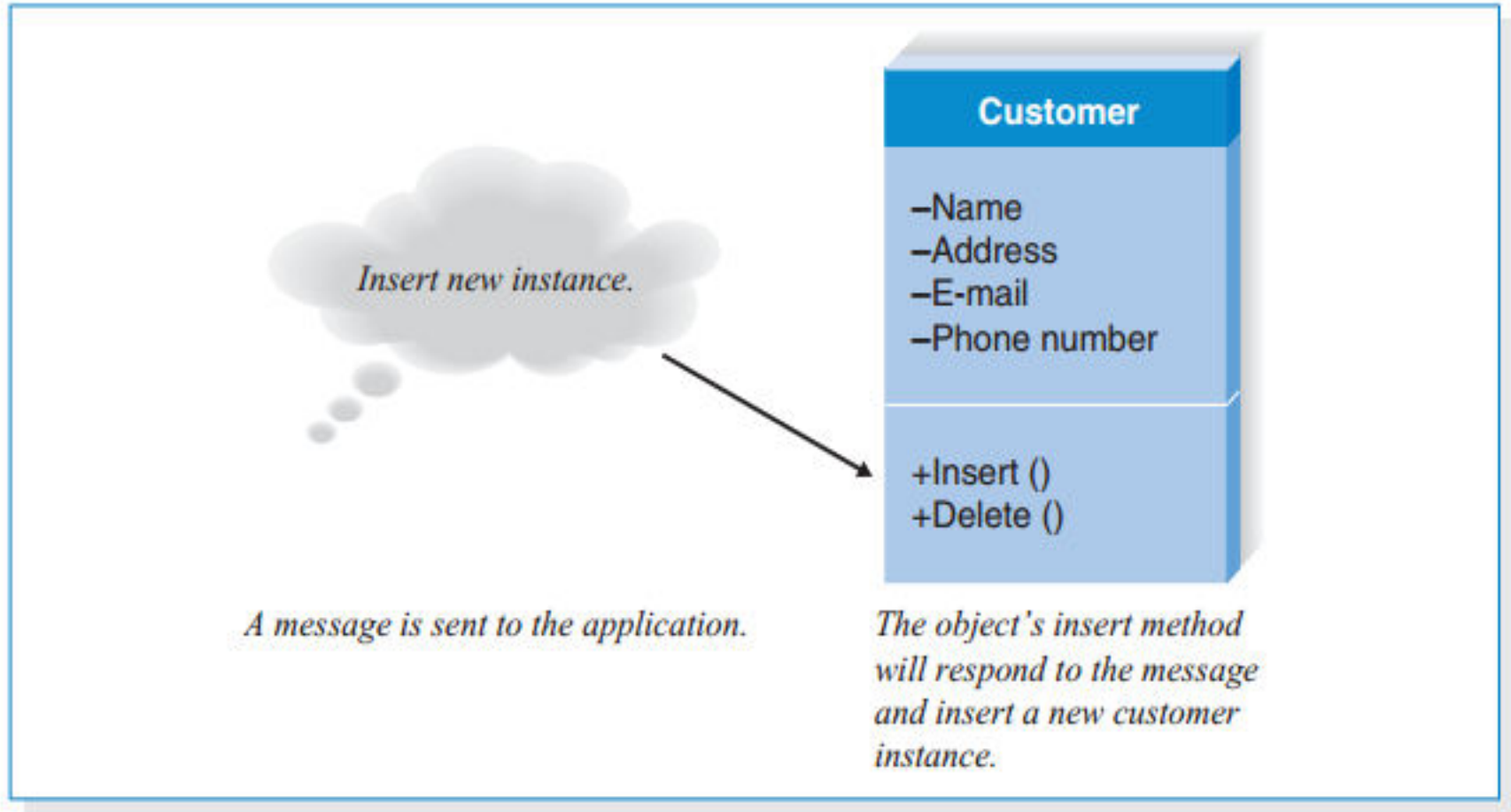
- A **method** is nothing more than an action that an object can perform. Methods are very much like a function or procedure in a traditional programming language.

e.g.

+BorrowBook(), +insert()

- **Messages** are information sent to objects to trigger methods. A message is essentially a function or procedure call from one object to another object.

## ... major characteristics of O-O SAD



## ... major characteristics of O-O SAD

### C. Encapsulation and Information Hiding

- Encapsulation is simply the combining of process and data into a single entity. Object-oriented approaches combine process and data into holistic entities (objects).
- The principle of **information hiding** suggests that only the information required to use a software module be published to the user of the module.



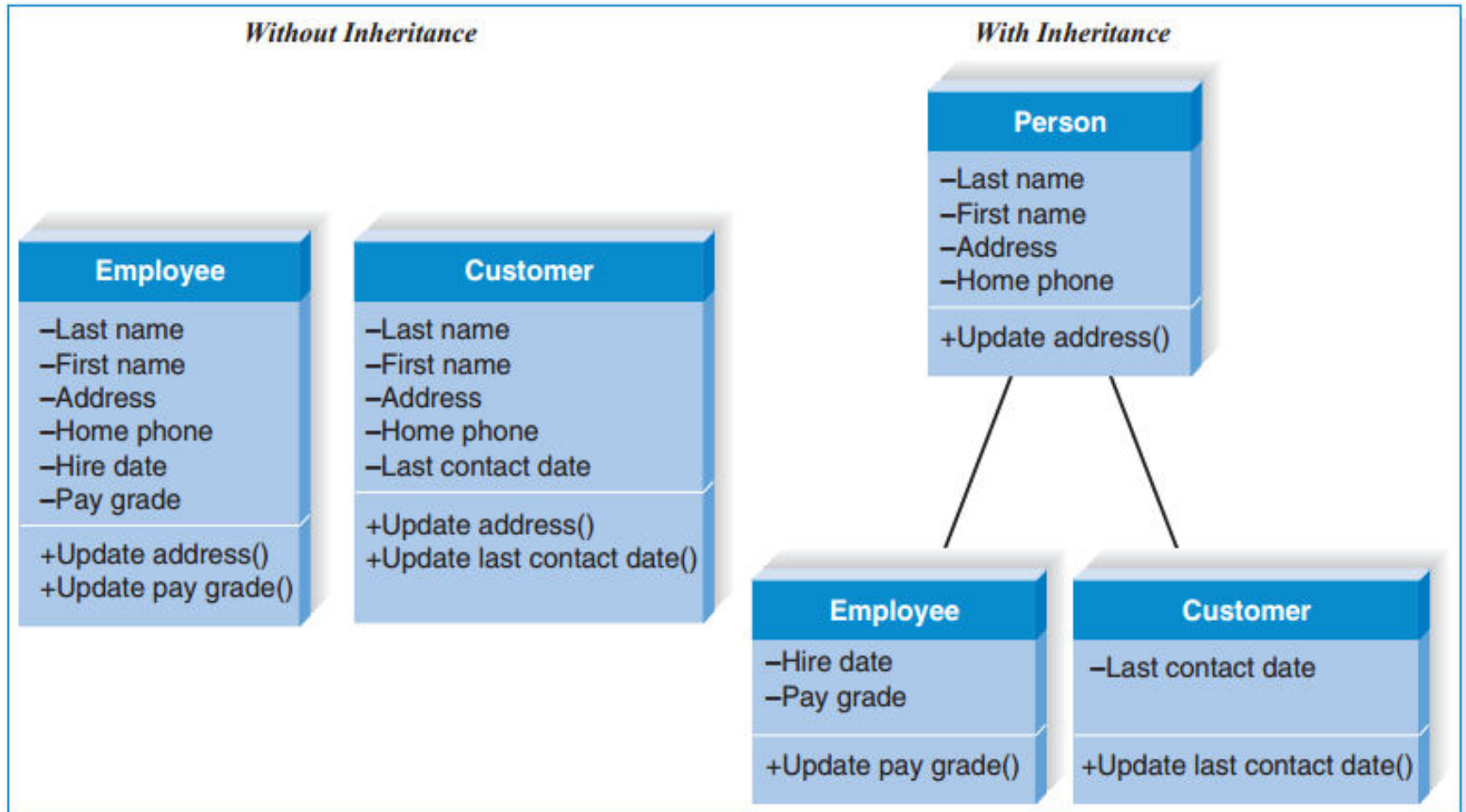
# ... major characteristics of o-o SAD

## D. Inheritance

- The mechanism where features in a hierarchy inherit from super classes to subclasses by identifying higher level, or more general, classes of objects.
- Typically, classes are arranged in a hierarchy where by the super classes, or general classes are at the top, and the *subclasses*, or specific classes, are at the bottom.

# ... major characteristics of o-o SAD

e.g.



## ... major characteristics of OOSAD

### e. Polymorphism and Dynamic Binding

- **Polymorphism** means that the same message can be interpreted differently by different classes of objects.

e.g.

If we sent the message “Draw Image” to a square object, a circle object, and a triangle object, the results would be very different, even though the message is the same.

- Polymorphism is made possible through *dynamic binding*. Dynamic, or late, binding is a technique that delays identifying the type of object until run-time.

# More on O-O SAD

- Object-oriented approaches to developing systems, technically speaking, **can use any of the traditional methodologies** presented in above (waterfall development, parallel development, V-model, iterative development, system prototyping, and throwaway prototyping). The object-oriented approaches are **mostly** associated with a **RAD** methodology.
- In 1995, three industry leaders Grady Booch, Ivar Jacobson, and James Rumbaugh with others create a **single approach - UML** to object-oriented systems development.
- *Unified Modeling Language (UML)* is a set of **diagramming techniques** which is used as a standard approach for OOSAD.

## ..... More on OOSAD

- According to its creators (UML), any object-oriented approach in developing a system must be
  - (1) Use case driven,
  - (2) Architecture centric, and
  - (3) Iterative and incremental.

# Object-oriented approach

## (1) Use Case Driven

- *Use case driven* means that use cases are the primary modeling tool employed to define the behavior of the system.
- A *use case* describes how the user interacts with the system to perform some activity, such as placing an order, making a reservation or searching for information.

# Object-oriented approach

## 2. Architecture Centric

- The underlying architecture of the evolving system drives the specification, construction, and documentation of the system. There are three separate, but interrelated, architectural views of a system: **functional, static, and dynamic**.
- The *functional view* describes the external behavior of the system from the perspective of the user.
- The *static view* describes the structure of the system in terms of attributes, methods, classes, relationships, and messages.
- The *dynamic view* describes the internal behavior of the system in terms of messages passed between objects and state changes within an object.

# Object-oriented approach

## 3. Iterative and Incremental

- Object-oriented approaches emphasize *iterative* and *incremental* development that undergoes **continuous testing** throughout the life of the project. Each iteration of the system brings the system closer and closer to the final needs of the users



# Benefits of Object - Oriented Systems Analysis and Design (Why to Use?)

- Polymorphism, encapsulation, and inheritance taken together allow analysts to break a complex system into smaller, more manageable components.
- To work on the components individually, and to more easily piece the components back together to form a system.
- This modularity makes system development easier to grasp, easier to share among members of a project team, and easier to communicate to users who are needed throughout the SDLC.

# Benefits of OOAD

- Objects are reusable
- Maintenance cost are lowered
- Improved quality and maintainability
- Many people also argue that “object think” is a much more realistic way to think about the real world — so communicating in terms of objects improves the interaction between the user and the analyst or developer.

# Summary of the basic characteristics and benefits of OO approach

Concept	Supports	Leads to
Classes, objects, methods, and messages	<ul style="list-style-type: none"><li>• A more realistic way for people think about their business</li><li>• Highly cohesive units that contain both data and processes</li></ul>	<ul style="list-style-type: none"><li>• Better communication between user and analyst or developer</li><li>• Reusable objects</li><li>• Benefits from having a highly cohesive system (See cohesion in Chapter 10)</li></ul>
Encapsulation and information hiding	<ul style="list-style-type: none"><li>• Loosely coupled units</li></ul>	<ul style="list-style-type: none"><li>• Reusable objects</li><li>• Fewer ripple effects from changes within an object or in the system itself</li><li>• Benefits from having a loosely coupled system design (See coupling in Chapter 10)</li></ul>
Inheritance	<ul style="list-style-type: none"><li>• Allows us to use classes as standard templates from which other classes can be built</li></ul>	<ul style="list-style-type: none"><li>• Less redundancy</li><li>• Faster creation of new classes</li><li>• Standards and consistency within and across development efforts</li><li>• Ease in supporting exceptions</li></ul>
Polymorphism	<ul style="list-style-type: none"><li>• Minimal messaging that is interpreted by objects themselves</li></ul>	<ul style="list-style-type: none"><li>• Simpler programming of events</li><li>• Ease in replacing or changing objects in a system</li><li>• Fewer ripple effects from changes within an object or in the system itself</li></ul>
Use case driven	<ul style="list-style-type: none"><li>• Allows users and analysts to focus on how a user will interact with the system to perform a single activity</li></ul>	<ul style="list-style-type: none"><li>• Better understanding and gathering of user needs</li><li>• Better communication between user and analyst</li></ul>
Architecture centric and functional, static, and dynamic views	<ul style="list-style-type: none"><li>• Viewing the evolving system from multiple points of view</li></ul>	<ul style="list-style-type: none"><li>• Better understanding and modeling of user needs</li><li>• More complete depiction of information system</li></ul>
Iterative and incremental development	<ul style="list-style-type: none"><li>• Continuous testing and refinement of the evolving system</li></ul>	<ul style="list-style-type: none"><li>• Meeting real needs of users</li><li>• Higher quality systems</li></ul>

# **Project Part 1**

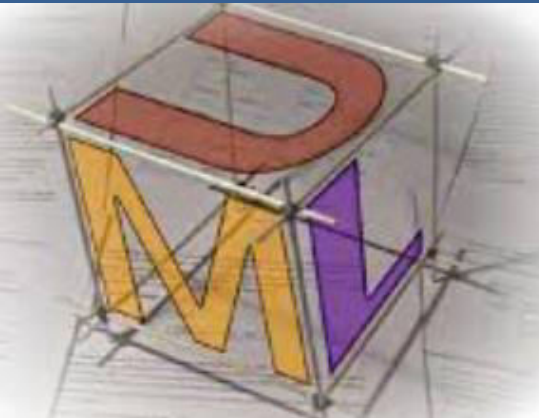
Think about your project proposal

- Title selection
- Contact potential clients

Out Put - Give at least a titles to your teacher

# Chapter Reviews Questions

- What is SDLC? What are the clearly defined phases in SDLC?
- What are software methodologies? Give some example, which one do you prefer to use? Why?
- What is system analyst? List some of skills require from system analyst.
- What are the basic difference between structured and object oriented system analysis and design?
- What are the basic characteristics of OO SAD?
- According to UML creators an object oriented system should be?(hint: mention three criteria)
- What are the reasons of using OO SAD?
- What is UML- Unified Modeling Language?



# Chapter Two

## Modeling Using UML

# Introduction to UML

UML is the language of blueprints for software, which uses a graphical language for

- **Visualizing**
- **Specifying**
- **Constructing**
- **Documenting**

The objective of the Unified Modeling Language is to provide a common vocabulary of object-based terms and diagramming techniques

# What UML is not?

- UML is not a methodology
- UML is independent of any methodology

## What UML is ?

- UML is an approach for OOSAD
- UML is a set of diagramming techniques that is rich enough to model any systems.



# Benefits of using UML

- **Common** language
- **Reduced learning curve**
- Increase domain and design model reuse
- Increase customer involvement  
/understanding

# Building blocks of UML

Includes things, Relationships and Diagrams

- i. Things - simply refers to the objects
- ii. Relationships - the glue that holds things together
  - A. Dependency
  - B. Association
  - C. Generalization
- iii. Diagrams
  - A. Diagrams for Structural Modeling
  - B. Diagrams for Behavioral Modeling
  - C. Diagrams for Architectural Modeling

# Structural Modeling

- Structural modeling captures the static features of a system. They consist of the following –
  - Classes diagrams
  - Objects diagrams
  - Deployment diagrams
  - Package diagrams
  - Composite structure diagram
  - Component diagram
- Structural model represents the framework for the system and this framework is the place where all other components exist. Hence, the class diagram, component diagram and deployment diagrams are part of structural modeling.
- The structural model never describes the dynamic behavior of the system. Class diagram is the most widely used structural diagram.

# Behavioral Modeling

- Behavioral model describes the interaction in the system. It represents the interaction among the structural diagrams. Behavioral modeling shows the dynamic nature of the system. They consist of the following –
  - Activity diagrams
  - Interaction diagrams
  - Use case diagrams
- All the above show the dynamic sequence of flow in a system.

# Architectural Modeling

- Architectural model represents the overall framework of the system. It contains both structural and behavioral elements of the system. Architectural model can be defined as the **blueprint** of the entire system. Package diagram comes under architectural modeling.
  - Component Diagram
  - Deployment Diagram
  - Package Diagram

# The recent version - UML 2.0 defines a set of 14 diagramming techniques.

Diagram Name	Used to	Primary Phase
<b>Structure Diagrams</b>		
Class	Illustrate the relationships between classes modeled in the system.	Analysis, Design
Object	Illustrate the relationships between objects modeled in the system.	
	Function when actual instances of the classes will better communicate the model.	Analysis, Design
Package	Group other UML elements together to form higher level constructs.	Analysis, Design, Implementation
Deployment	Show the physical architecture of the system. Can also be used to show software components being deployed onto the physical architecture.	Physical Design, Implementation
Component	Illustrate the physical relationships among the software components.	Physical Design, Implementation
Composite Structure	Illustrate the internal structure of a class—i.e., the relationships among the parts of a class.	Analysis, Design
<b>Behavioral Diagrams</b>		
Activity	Illustrate business work flows independent of classes, the flow of activities in a use case, or detailed design of a method.	Analysis, Design
Sequence	Model the behavior of objects within a use case. Focuses on the time-based ordering of an activity.	Analysis, Design
Communication	Model the behavior of objects within a use case. Focuses on the communication among a set of collaborating objects of an activity.	Analysis, Design
Interaction Overview	Illustrate an overview of the flow of control of a process.	Analysis, Design
Timing	Illustrate the interaction that takes place among a set of objects and the state changes that they go through along a time axis.	Analysis, Design
Behavioral State Machine	Examine the behavior of one class.	Analysis, Design
Protocol State Machine	Illustrate the dependencies among the different interfaces of a class.	Analysis, Design
Use Case	Capture business requirements for the system and to illustrate the interaction between the system and its environment.	Analysis

# Commonly Used UML Diagrams

**Use case diagram:** describing how the system interacting with external environment. is used the starting point for UML modeling.

**Class diagram,** showing classes and relationships. Sequence diagrams and CRC cards are used to determine classes.

**Sequence diagram,** showing the sequence of activities and class relationships. Each use case may create one or more sequence diagrams.

**Collaboration diagram** is an alternative to a sequence diagram.

**Activity diagram:** Each use case may create one activity diagram.

**Component diagram** Model how components are connected with each other in forming software systems and the interfaces of those components.

Example: third party libraries, files, executable, etc.

# ... Commonly Used UML Diagrams

7. **Deployment diagram:** Model the physical deployment of artifacts with component (e.g. web server), artifact (e.g. jar file) and node (web application, DB).



# Architecture

Architecture is the set of significant decisions about: -

- The organization of a software system
- The selection of the structural elements and their interfaces by which the system is composed
- Their behavior, as specified in the collaborations among those elements
- The composition of these structural and behavioral elements into progressively larger subsystems
- The architectural style that guide this organization: the static and dynamic elements and their interfaces, their collaborations, and their composition.

- In OOSAD Architecture of a system can be described by a **view** - a subset of UML modeling constructs that represent one aspect of the system that involves structural and behavioural models.
- Different stakeholders look at the system in different angles at different times over the project's life span. UML captures these different angles/viewpoints as a set of five interlocking views.

# Views of a system

vocabulary  
functionality

system assembly  
configuration  
management

**Design view**

**Implementation  
view**

Use case view

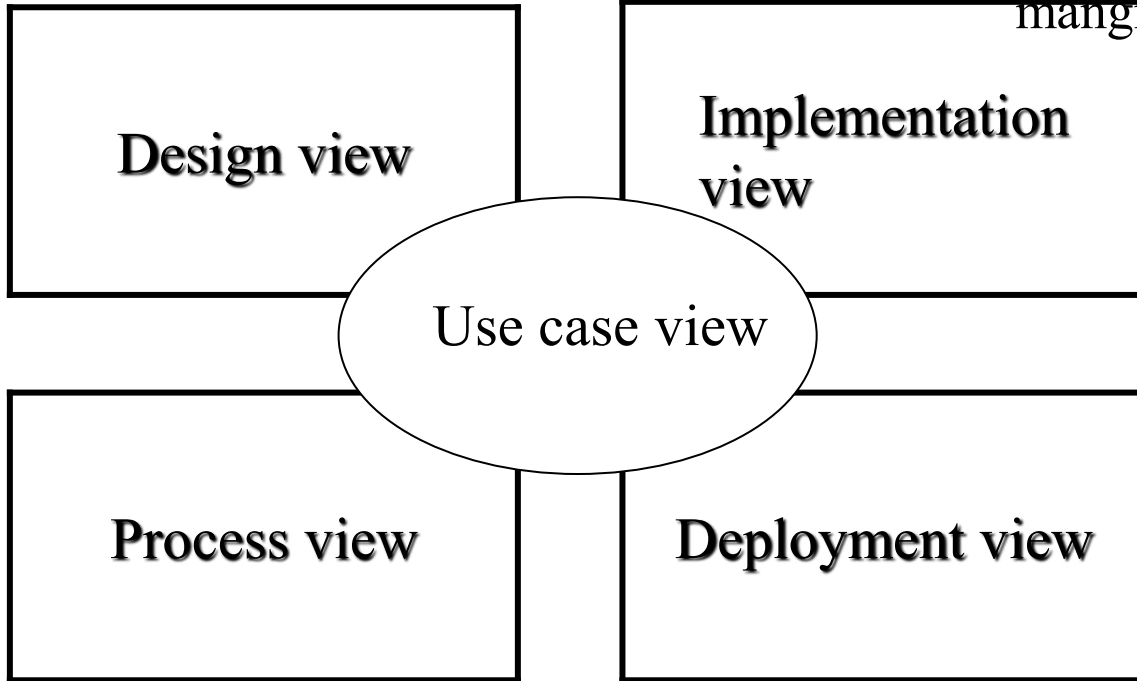
behavior

**Process view**

**Deployment view**

performance  
scalability  
throughput

system  
topology  
distribution  
delivery  
installation




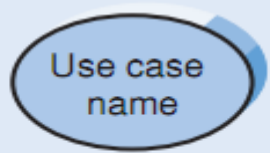
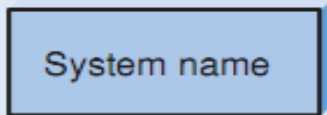
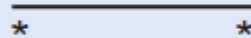
# More on UML Diagrams

- From commonly used diagramming techniques **use case diagram** is the driving one – **Starting Point**.

## Use case diagram

- The use case communicates at a high level what the system needs to do, and each of the UML diagramming techniques build upon it.
- A use case diagram illustrates in a very simple way the **main functions of the system** and the **different kinds of users** who will interact with the system.

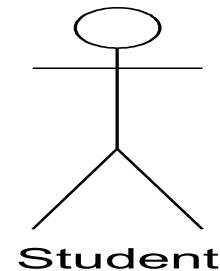
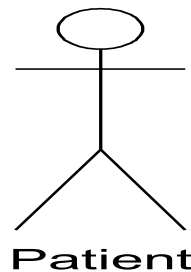
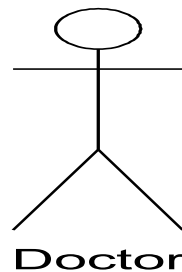
# Components of a Use Case Diagram

Term and Definition	Symbol
<p>An actor</p> <ul style="list-style-type: none"><li>■ Is a person or system that derives benefit from and is external to the system.</li><li>■ Is labeled with its role.</li><li>■ Can be associated with other actors by a specialization/superclass association, denoted by an arrow with a hollow arrowhead.</li><li>■ Is placed outside the system boundary.</li></ul>	 Actor role name
<p>A use case</p> <ul style="list-style-type: none"><li>■ Represents a major piece of system functionality.</li><li>■ Can extend another use case.</li><li>■ Can use another use case.</li><li>■ Is placed inside the system boundary.</li><li>■ Is labeled with a descriptive verb–noun phrase.</li></ul>	 Use case name
<p>A system boundary</p> <ul style="list-style-type: none"><li>■ Includes the name of the system inside or on top.</li><li>■ Represents the scope of the system.</li></ul>	 System name
<p>An association relationship</p> <ul style="list-style-type: none"><li>■ Links an actor with the use case(s) with which it interacts.</li></ul>	

# Actors

- is a person or another system that interacts with and derives value from the system.
- is not a specific user, but a role that a user can play while interacting with the system.

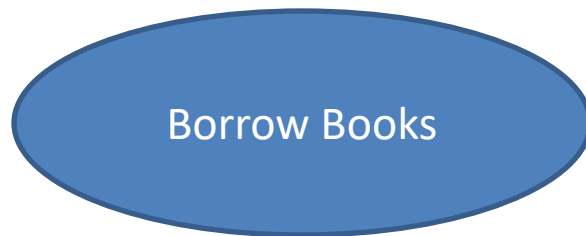
Example of an actor:



# Use Case

- by an oval, is a major process that the system will perform that benefits an actor(s) in some way
- it is labeled by a descriptive verb phrase
- describes ***what*** a system does; not ***how***

***example***



# Association Relationship

- Use cases are connected to actors through association relationships.
- The association typically represents two-way communication between the use case and the actor. The “\*” shown at either end of the association represents **multiplicity**.
- relationships between use cases is stated in terms of includes, extends or generalization.

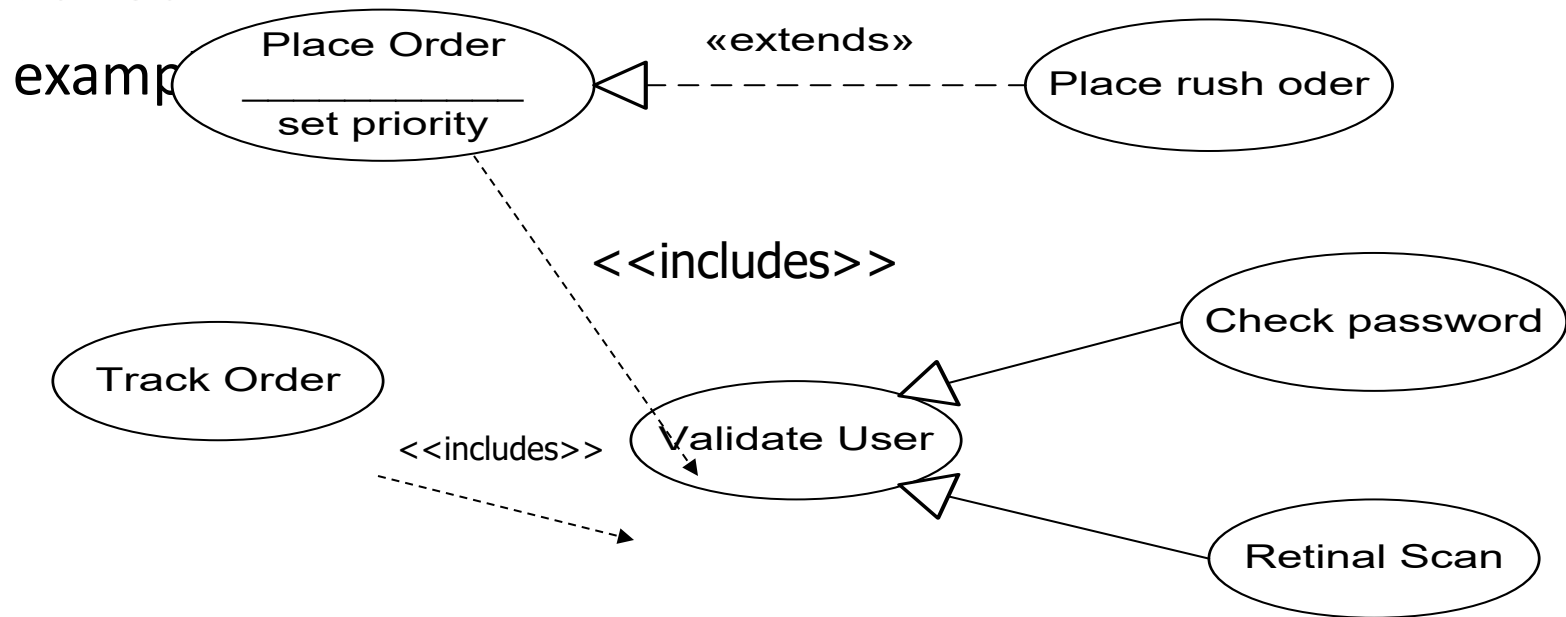


Includes:

Indicates that the base use case will contain the inclusion use case.

Extends :

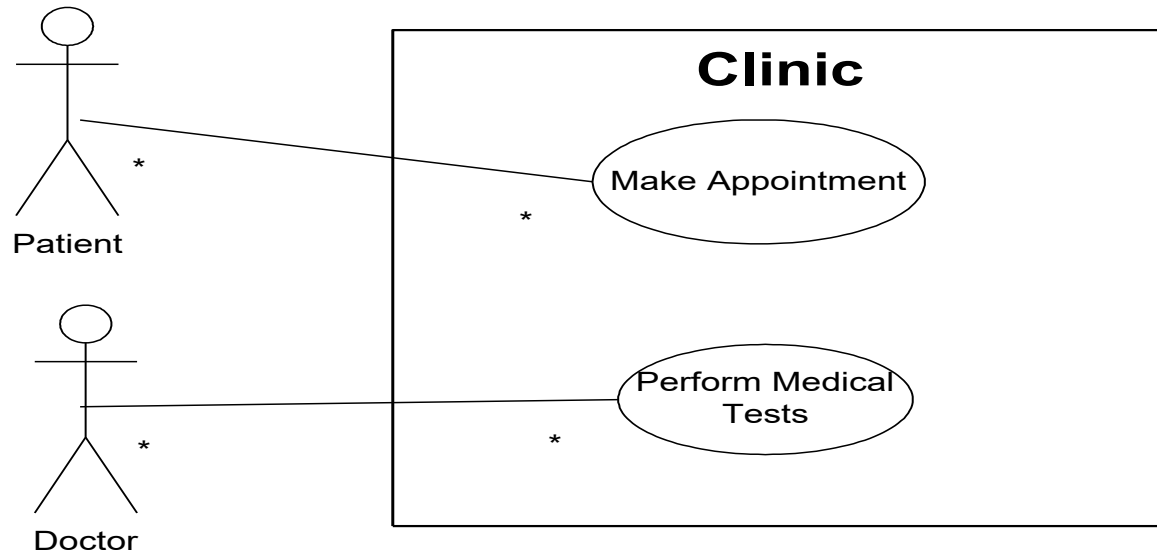
Augment the base use case if an extension condition is satisfied.



# System Boundary

- The use cases are enclosed within a system boundary, which is a box that represents the system.

Example: A use case diagram depicting the system boundary of a clinic application



## ii. Use Case documentation

- The Use Case documentation needs information like:
  - List of Actors
  - List of Business Rules (BR)
  - List of User Interfaces (UI)
- Each use case should be associated with a specific documentation

# ... documentation

Example: a documentation template sample in a library system for Librarian actor of Adding book use case.

Use Case ID	UCID-22	
Use Case Name	Add book	
Participating Actors	Librarian	
Description	Enables to Add book	
Precondition	User is logged in	
Flow of Events	1	The User press on “library” tab from the dashboard.
	2	System displays “library” page with different navigation option
	3	Press “add book” button.
	4	The user fill the information about the new book
	5	The user press “add book” button to save the entered book. (A1)
Alternative flow	A1	If the user enter inappropriate information press “clear” button.
		Resume step 5
Post-condition	The Book information is added!!	

# Project Part II

1. Install one of a UML application (e.g. Visual Paradigm) and prepare a **use case diagram** for the title you selected in project part I.
2. Prepare a documentation for the use case diagram.

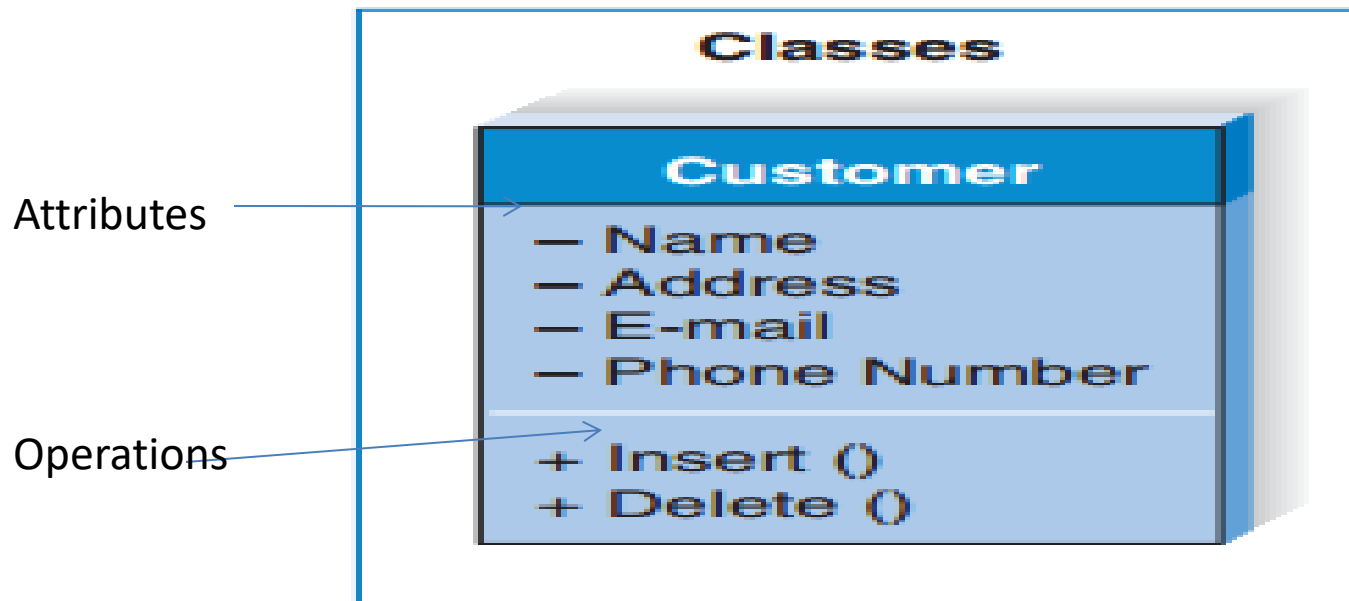
Out Put – Submit the diagram to your teacher.

# Class diagram

- is a **static model** that supports the static view of the evolving system
- It shows the classes and the relationships among the classes that remain constant in the system over time.
- are widely used to describe the types of objects in a system and their **relationships**, this relationship can be **inheritance, aggregation and association**.

# Class diagram

- It also describes the **attributes** and **operations/method** of the class along with the visibility of each.



# Use of class diagram

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

- making a decision about which abstractions are a part of the system
- specify abstractions and their responsibilities

## **2. To model simple collaborations**

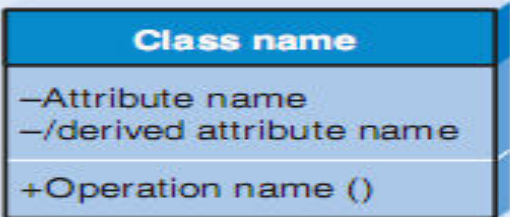
- is a society of classes, interfaces, and other elements that work together

## **3. To model logical database schema**

- to model schemas for databases used to store persistent information.


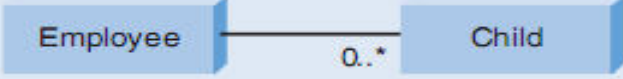
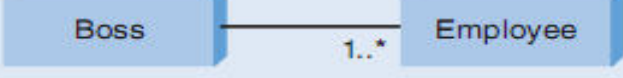
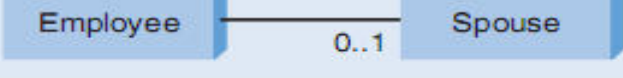




# Components of a Class Diagram

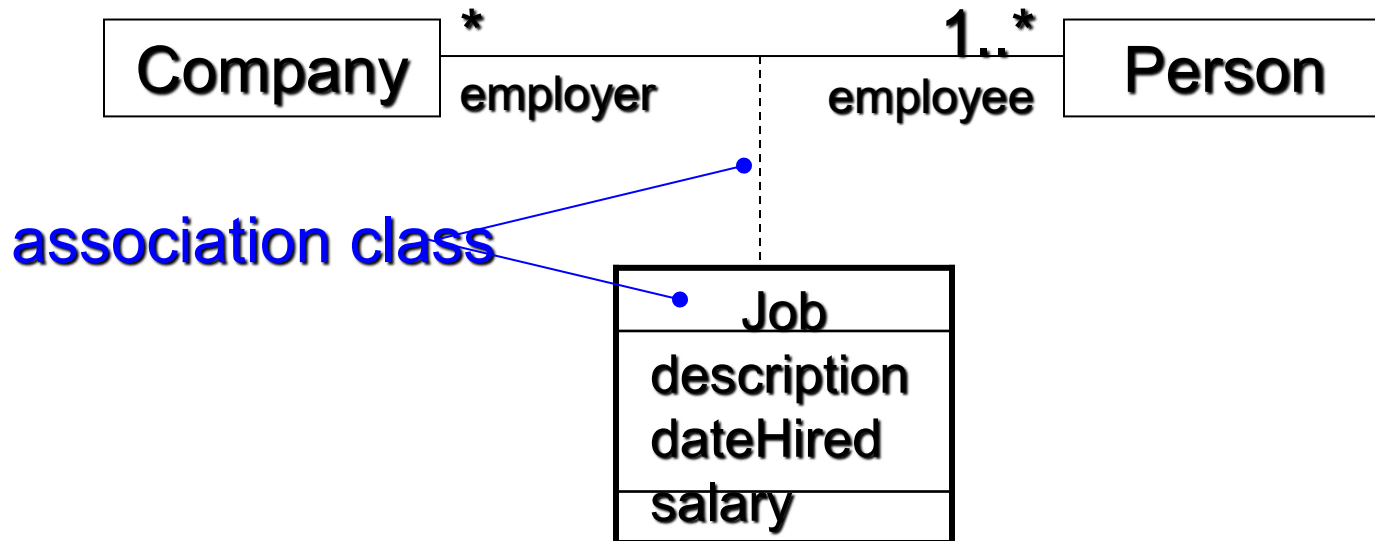
Term and Definition	Symbol
<p>A class</p> <ul style="list-style-type: none"> <li>Represents a kind of person, place, or thing about which the system must capture and store information.</li> <li>Has a name typed in bold and centered in its top compartment.</li> <li>Has a list of attributes in its middle compartment.</li> <li>Has a list of operations in its bottom compartment.</li> <li>Does not explicitly show operations that are available to all classes.</li> </ul>	
<p>An attribute</p> <ul style="list-style-type: none"> <li>Represents properties that describe the state of an object.</li> <li>Can be derived from other attributes, shown by placing a slash before the attribute's name.</li> </ul>	<p>Attribute name /derived attribute name</p>
<p>A method</p> <ul style="list-style-type: none"> <li>Represents the actions or functions that a class can perform.</li> <li>Can be classified as a constructor, query, or update operation.</li> <li>Includes parentheses that may contain special parameters or information needed to perform the operation.</li> </ul>	<p>Operation name ()</p>
<p>An association</p> <ul style="list-style-type: none"> <li>Represents a relationship between multiple classes, or a class and itself.</li> <li>Is labeled by a verb phrase or a role name, whichever better represents the relationship.</li> <li>Can exist between one or more classes.</li> <li>Contains multiplicity symbols, which</li> </ul>	<p>1..*      verb phrase      0..1</p> <hr/>

# multiplicity

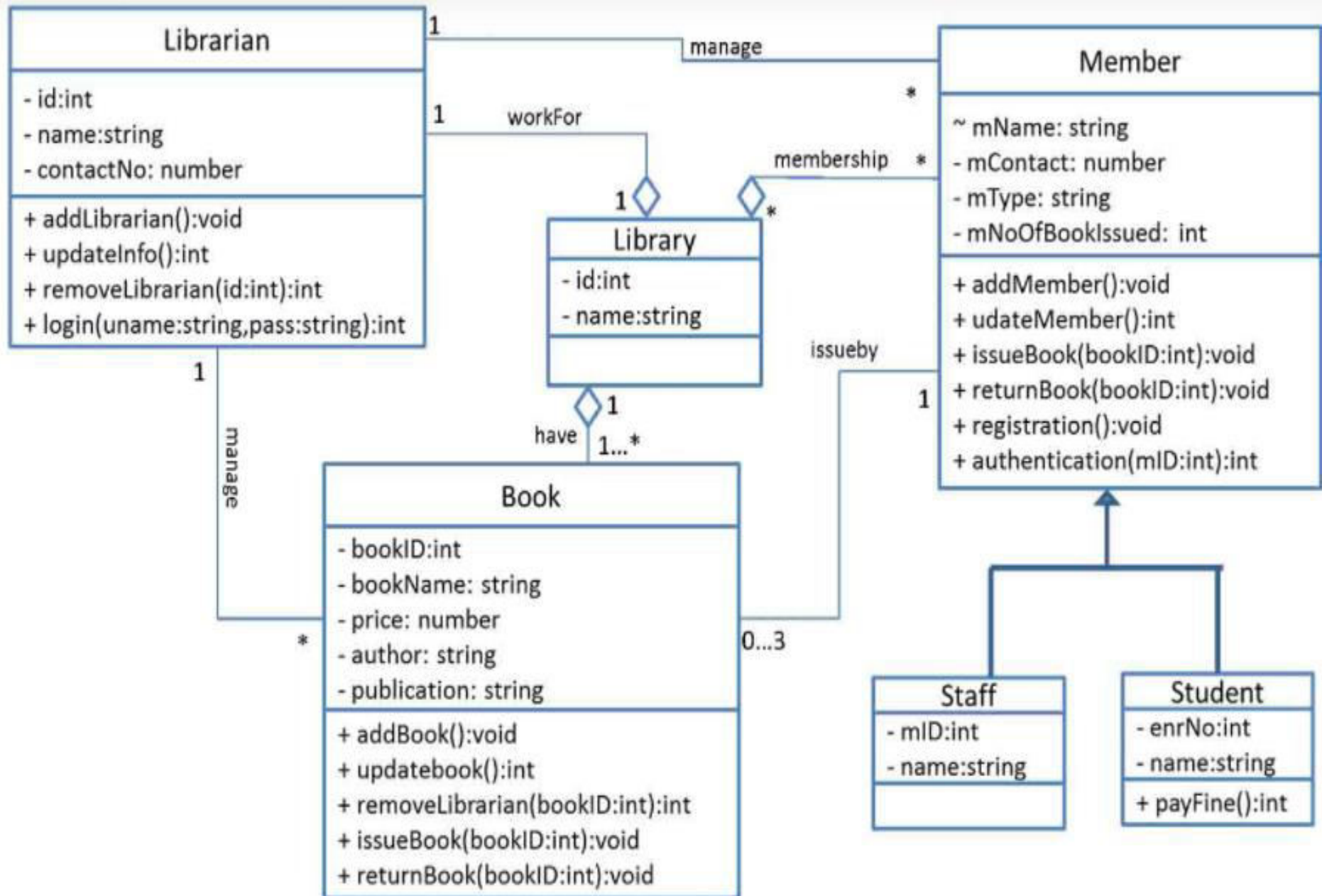
- shows how an instance of an object can be associated with other instances.

Instance(s)	Representation of Instance(s)	Diagram Involving Instance(s)	Explanation of Diagram
Exactly one	1	 <pre> classDiagram     Department "1" -- "1" Boss                     </pre>	A department has one and only one boss.
Zero or more	0..*	 <pre> classDiagram     Employee "1" -- "0..*" Child                     </pre>	An employee has zero to many children.
One or more	1..*	 <pre> classDiagram     Boss "1" -- "1..*" Employee                     </pre>	A boss is responsible for one or more employees.
Zero or one	0..1	 <pre> classDiagram     Employee "1" -- "0..1" Spouse                     </pre>	An employee can be married to zero or one spouse.
Specified range	2..4	 <pre> classDiagram     Employee "1" -- "2..4" Vacation                     </pre>	An employee can take between two to four vacations each year.
Multiple, disjoint ranges	1..3, 5	 <pre> classDiagram     Employee "1" -- "1..3, 5" Committee                     </pre>	An employee is a member of one to three or five committees.

Note: - There are times when an association itself has associated properties, especially when its classes share a many-to-many relationship. In these cases, a class is formed called an association class that has its own attributes and methods. Example:



# Sample class diagram



# Sequence Diagram




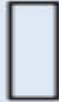
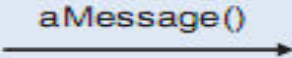

What is sequence diagram?

- A sequence diagram is a dynamic model that illustrates the **objects** that participate in a use case and **the messages** that pass between them over time.
- A sequence diagram is an interaction diagram that emphasizes the time ordering of messages.

# Components of a Sequence Diagram

- A. Actor and Objects appear along the top margin
- B. Lifeline - Each object has a lifeline, which is a dashed line that represent the life and perhaps death of the object
  - Most objects will be in existence for the duration of the interaction
- C. A focus of control, which is a tall thin rectangle that sits on top of an object's lifeline.
- D. Messages show the actions that objects perform on each other and on themselves.



Term and Definition	Symbol
<p>An actor:</p> <ul style="list-style-type: none"> <li>■ Is a person or system that derives benefit from and is external to the system.</li> <li>■ Participates in a sequence by sending and/or receiving messages.</li> <li>■ Is placed across the top of the diagram.</li> </ul>	 <p>anActor</p>
<p>An object:</p> <ul style="list-style-type: none"> <li>■ Participates in a sequence by sending and/or receiving messages.</li> <li>■ Is placed across the top of the diagram.</li> </ul>	
<p>A lifeline:</p> <ul style="list-style-type: none"> <li>■ Denotes the life of an object during a sequence.</li> <li>■ Contains an X at the point at which the class no longer interacts.</li> </ul>	
<p>A focus of control:</p> <ul style="list-style-type: none"> <li>■ Is a long narrow rectangle placed atop a lifeline.</li> <li>■ Denotes when an object is sending or receiving messages.</li> </ul>	
<p>A message:</p> <ul style="list-style-type: none"> <li>■ Conveys information from one object to another one.</li> </ul>	
<p>Object destruction:</p> <ul style="list-style-type: none"> <li>■ An X is placed at the end of an object's lifeline to show that it is going out of existence.</li> </ul>	

# How to create a sequence diagram?

- It is formed by Placing the objects that participate in the interaction at the top of the diagram, along the X-axis.
  - » The object that initiates the interaction is placed on the left most, and the other subordinate objects are placed to the right
- And by Placing the messages that these objects send and receive along the Y-axis, in order of increasing time from top to bottom.



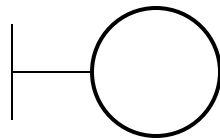
# General steps to Creating a Sequence Diagram

## Step 1. Identify Objects

- The first step is to identify instances of the classes that participate in the sequence being modeled; that is, the objects that interact with each other during the use case sequence.
- i.e objects can be taken from use case diagrams, class diagrams ...
- Objects can be **boundary objects, entity objects or control objects.**

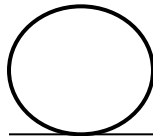
- i. Boundary objects - is an object with which an actor associated interacts.

Example: if the actor is human, the boundary object may be a window, screen, dialog box, or menu.



boundary object

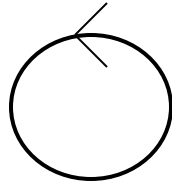
- ii. Entity objects - is an object that contains long-lived information, such as that associated with databases.



entity object

### ... General steps

- iii. Control objects - used as a connecting tissue between boundary objects and entity objects.



control object

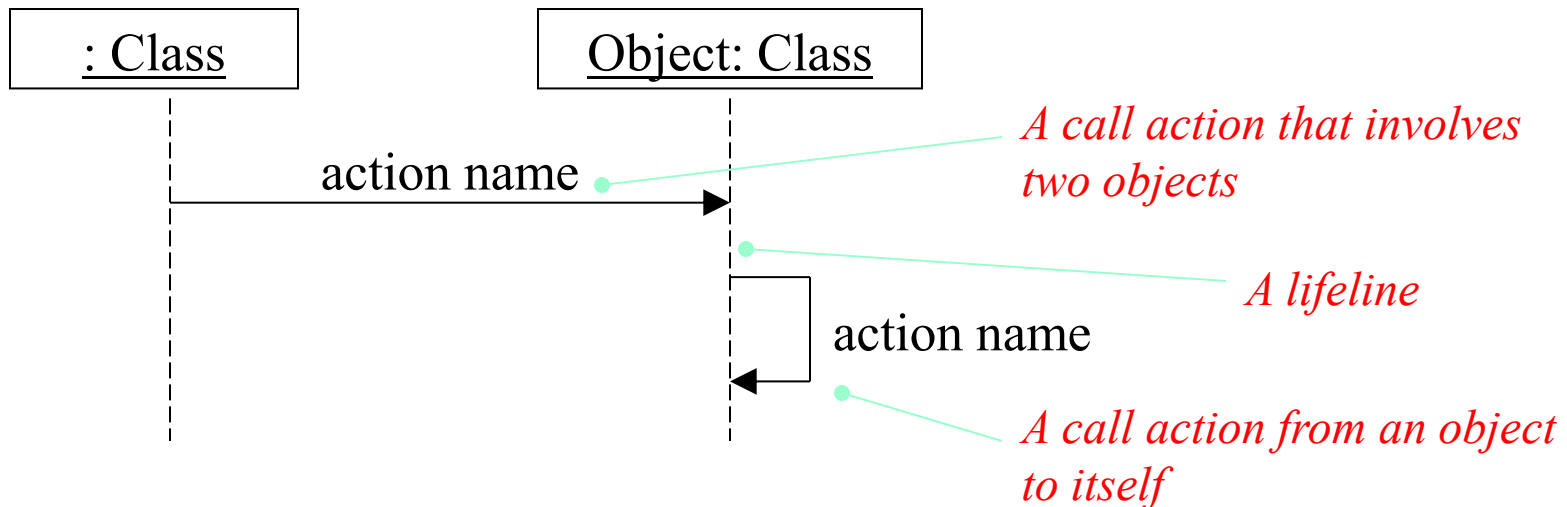
### Step 2. Add Messages

- A message is a communication between two objects, or within an object. Messages support 5 kinds of actions : -

# ...messages

## i. Call and Return

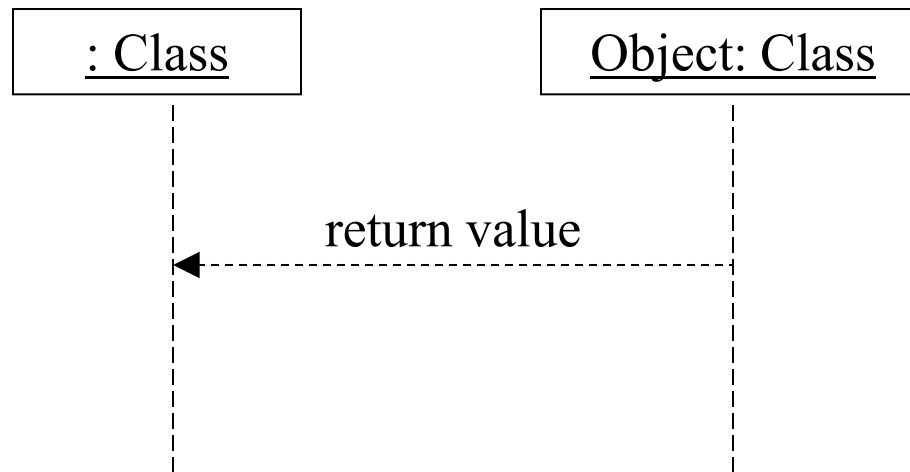
- A call action invokes an operation on an object, it is synchronous, meaning that the sender assumes that the receiver is ready to accept the message, and the sender waits for a response from the receiver before proceeding.



In UML a call action represented as an arrow

## ... message

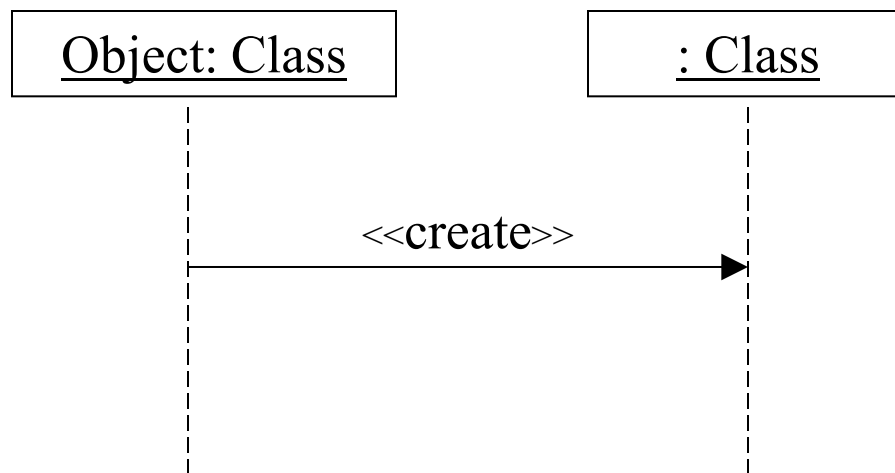
- A return action is the return of a value to the caller, in response to a call action. The UML represents a return action as a dashed arrow.



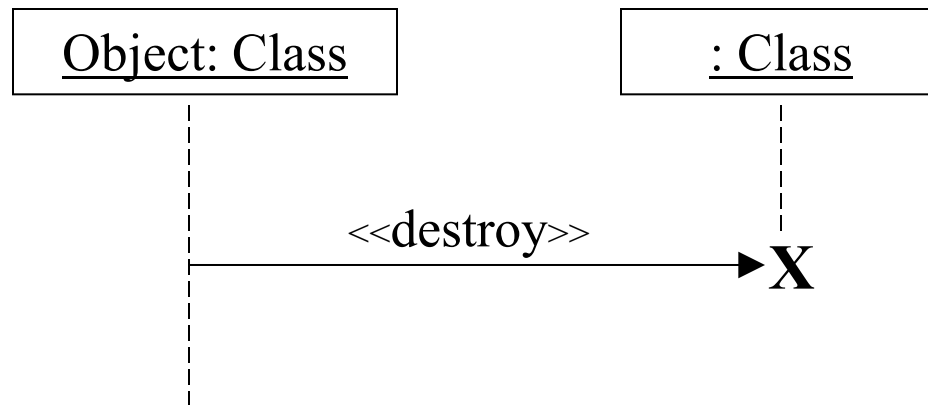
# ... messages

## ii. Create and Destroy

- A create action creates an object, it tells a class to create an instance of itself. In the UML, create action is represented as an arrow with the stereotype <<create>> .



- A destroy action destroys an object; it tells an object to destroy itself. In the UML, a destroy action is represented as an arrow with the stereotype <<destroy>>



# Activity diagrams









- Activity diagram for Library Management System

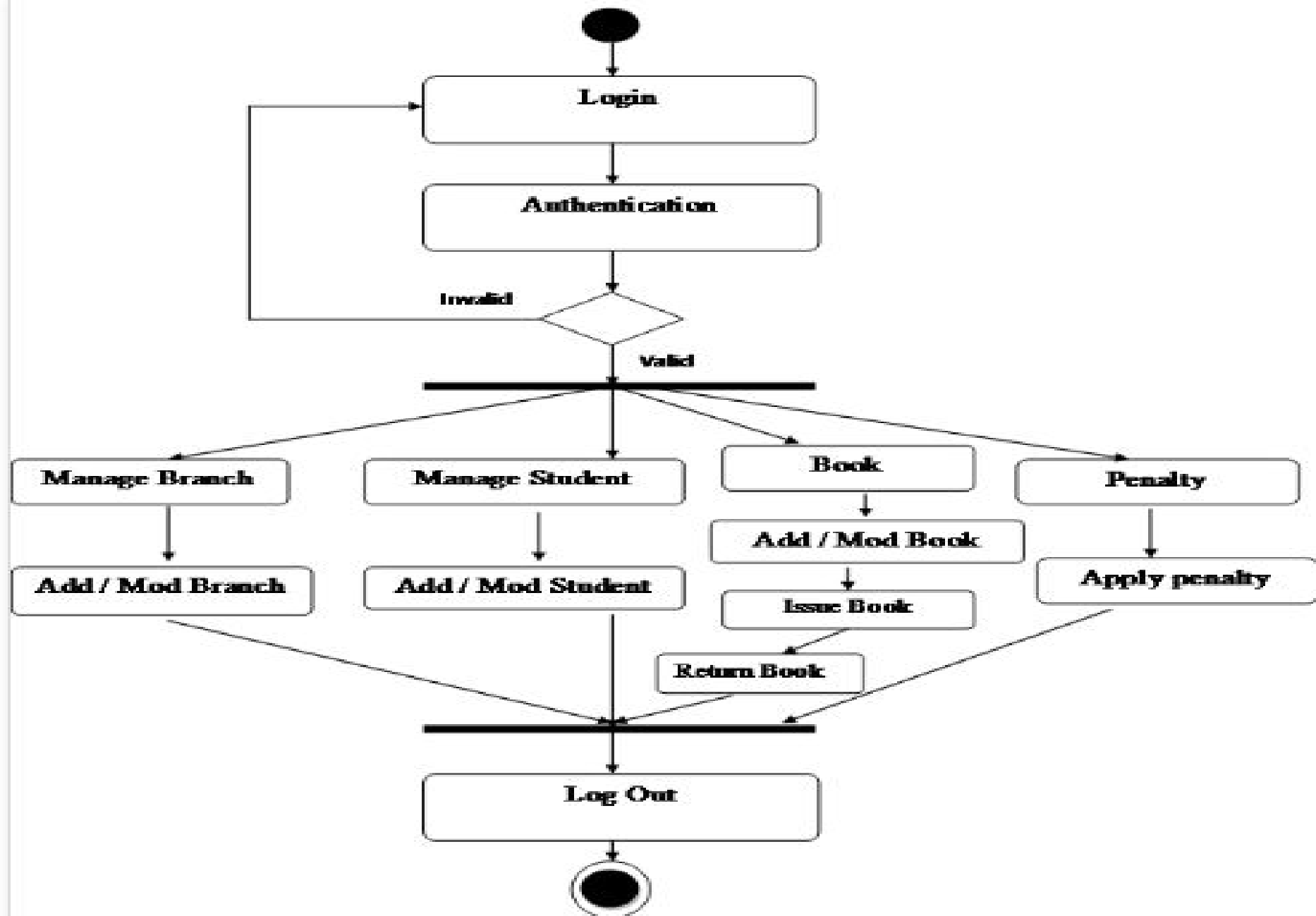
The activity diagram used to describe flow of activity through a series of actions.

Activity diagram is a important diagram to describe the system. The activity described as a action or operation of the system.



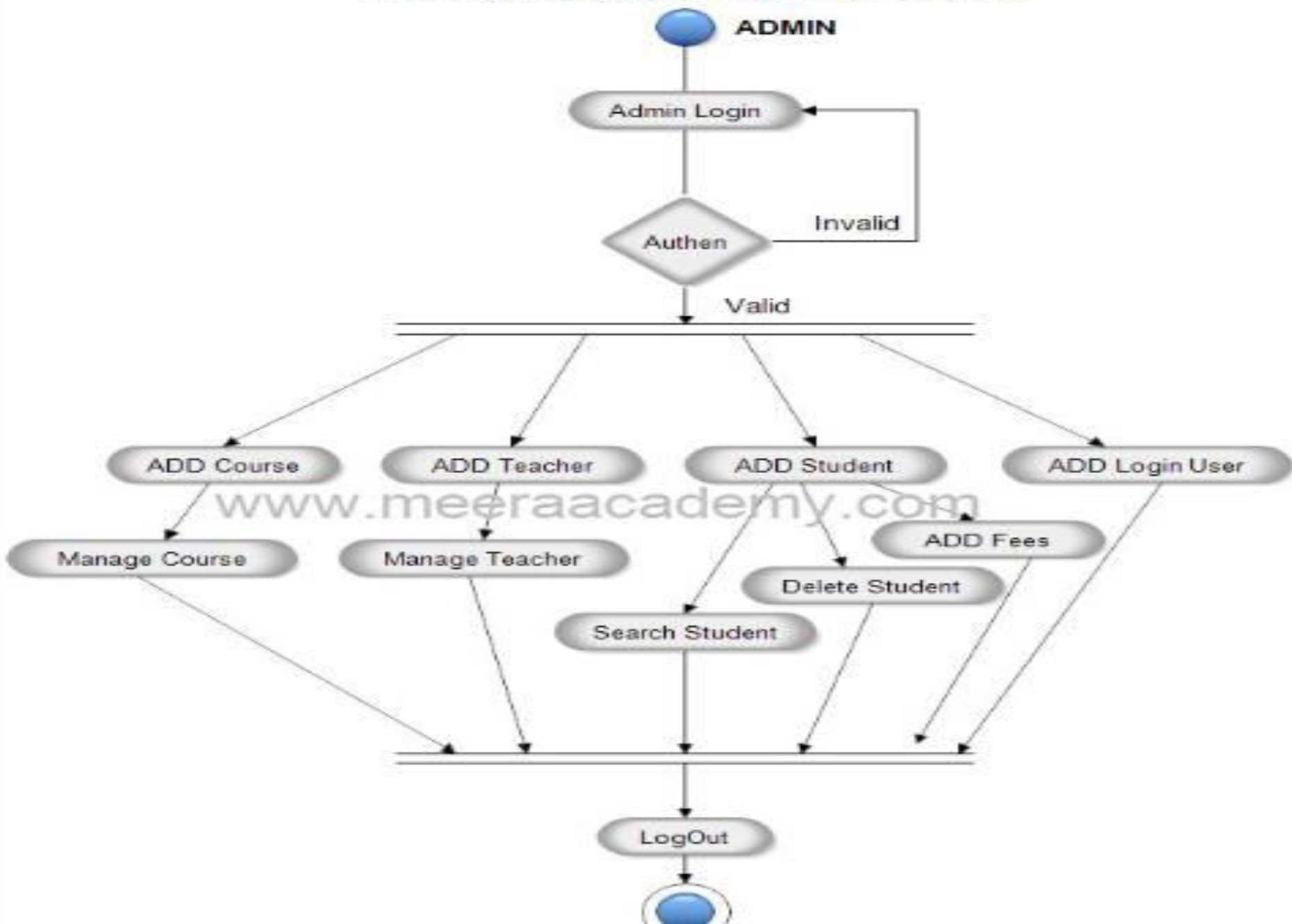
# Activity Diagram Symbols

Symbol	Description
	<b>Solid Circle</b> : Start the process of activity diagram
	<b>Rounded Rectangle</b> : Event and Activity.
	<b>Solid Line</b> : Sequence from one activity to next.
	<b>Dotted Store</b> : Flow of information between events.
	<b>Document</b> : Represent report or document.
	<b>Diamond</b> : Branch.
	<b>Table</b> : A place where data to be stored.
	<b>End of Process.</b>

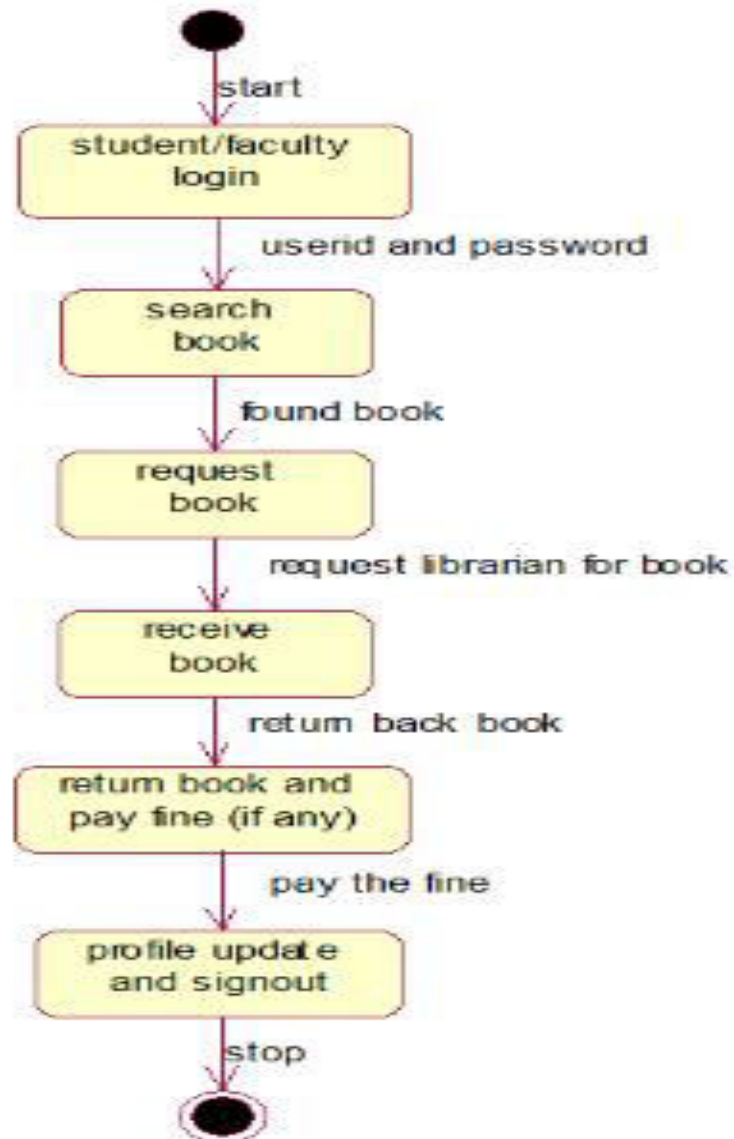


Activity Diagram for Library Management System.

# Activity Diagram Student System



# Statechart diagram



IT 331

# **Chapter Three - Requirements Elicitation**

# Requirements Elicitation?

## What is Requirement ?

- It is a **statement** describing either an aspect of what the proposed system must do or a constraint on the system's development.
- Is a feature that the system **must have** or a **constraint** that it must satisfy to be accepted by the client.
- represents a **negotiated agreement** among the stakeholders, mostly between the system developers and customers.

# Types of Requirements

- Functional requirements
- Non-functional requirements
- Pseudo requirements

# Functional requirements

- Mainly it describes the **interactions** between the system and its environment independent of its implementation.
- Describe:
  - What the system should **do**
  - What **inputs** the system should accept
  - What **outputs** the system should produce
  - What data the system should **store** that other systems might use.
  - What computations the system should perform
  - The timing and synchronization of the above



# Non-functional requirements

- Describe aspects of the system that are **not directly related** with the system.
- Non-functional requirements include qualitative constraints, such as response time (i.e., how fast the system reacts to user commands) or accuracy (i.e., how precise are the system's numerical answers).
- Quality requirements include
  - Response time, Throughput , Resource usage, Reliability, Availability, Recovery from failure , Allowances for maintainability and enhancement, Allowances for reusability

# Pseudo requirements

- Are requirements imposed by the client that **restrict the implementation of the system.**
- Are the implementation language and the platform on which the system is to be implemented.

# What is Requirement elicitation?

- The **process** of requirement **induction** through **domain analysis** using different techniques.
- A **communication** between developers and users for generating a new system using different methods that helps making of domain analysis.
- Includes
  - Focusing on describing the purpose of the system
  - Results in system specification

# What is Domain Analysis?

- The process by which a system analysts **learns** about the domain to better understand the problem.
- The **domain is** *the general field of business or technology in which the clients will requested for system.*
  - e.g. library reception management system
- A person who has a deep knowledge of the domain is known as **domain expert.**

# Benefits of performing domain analysis:

- Faster development
- Better system
- Anticipation of extensions
- Domain analysis is mainly dependent on:
  - Problem
  - Scope

# Problem and Scope in Requirement elicitation

## What is problem?

A problem can be expressed as:

- A **difficulty** the users or customers are facing, or as **an opportunity** that will result in some benefit such as improved productivity or sales.
- The problems which insist a system development that is documented as part of Requirements by a name **problem of statements usually**.

Note:- A good problem statement is short and concise

# What is Scope?

- Involves determining and documenting a list of specific project goals, deliverables, tasks, costs and deadlines.
- It can be restricted both on geographical and idea coverage or on either of them in order to provide a clear focus.

Example: A university registration system

# How to conduct Requirement elicitation?

## Techniques used

- i. Traditional methods
- ii. Essential use case
- iii. CRC model
- iv. Essential user interface (UI prototyping)
- v. Using Supplementary specification
- vi. User Interface flow diagram



# Method 1:- Traditional methods

- Research/Document
- Interviews (interview guide)
- Direct Observation (observation guide)
- Questionnaires and Surveys
- Joint Application Design (JAD) - facilitated workshop.

# Method2: Using essential use case modeling

- Use Cases are used to capture the intended behavior of the system under development (requirements of a system)
- Question you should ask
  - a. Which user groups are supported by the system to perform their task?
  - b. Which user groups execute the systems major function?
  - c. Will the system interact with any external hardware or software?
  - d. What are the tasks that the actor wants the system to perform?

# Method 3: Using Class Responsibility Collaboration (CRC cards)



- CRC are **an index card** on which one records the responsibilities and collaborators of classes, thus the name, which stands for Class-Responsibility-Collaboration.
- Are a brainstorming tool used in the design of object-oriented software.

# e.g. CRC cards

Item	
ItemID	SalesClerk
ItemName	Invoice (UI)
UnitPrice	
Quantity	
AddItem	
CheckReorder	

Sales Clerk <<Actor>>	
SellItem	Item
GenerateReport	Invoice (UI)
CheckAvailability	

Eg: CRC for User Interface Class

Invoice <<UI>>	
*see prototype*	Item
	Sales Clerk

# Method 4.Using Essential UI prototype

- User Interface is portion of software with which user directly interacts with the system.
- An essential UI prototype is a low fidelity model, or prototype of the UI for the actual system. It represents the general ideas behind the UI and not the exact detail.
- It is represented by various rectangles enclosed in a bigger one.

# e.g. Essential UI

Invoice		
<b>Customer Information</b> <div><b><u>CustomerFullName</u></b> <i>Input Field</i></div> <div><b>Customer Address</b> <i>Input Field</i> <i>Includes: <u>Woreda</u>, <u>Kebele</u>, House Number and Tele</i></div>	<b>Company Detail</b> <div><b><u>CompanyTinNo.</u></b> <i>Display Only</i></div> <div><b>Company Address</b> <i>Display Only</i> <i>Includes: <u>Woreda</u>, <u>Kebele</u>, House Number and Tele</i></div>	<b><u>InvoiceNumber</u></b> <i>Display only</i>
		<b>Date</b> <i>Input Field</i>
<b>Purchased Item List</b> <i>Display: includes the item code, item name, unit price, quantity purchased, and total price.</i>		
		<b>Grand Total</b> <i>Display Only</i>
<b>Notice</b> <i>Display Only: includes custodian of receipts</i>		

# Method4: Supplementary Specification

- Business Rules - own internal policies and principles on how to run
- Non-Functional Requirements and
- Constraints - includes the various restrictive conditions that exist in providing/delivering the system
- Change case - changes need to be anticipated before the development of a system