15UCS502 - OBJECT ORIENTED ANALYSIS AND DESIGN

Presented by,

K.A.MOHAMMED FAIZ,

Asst.prof /CSE Dept ,

SIT.

SYLLABUS

UNIT 1:

Introduction to OOAD – unified process – UML diagram – use case – class diagrams – interaction diagrams – state diagrams – activity diagrams – package ,component and deployment diagrams.

OOAD (OBJECT ORIENTED ANALYSIS AND DESIGN)
INTRODUCTION OF OOAD

What is OOAD?

OOAD Is a Software engineering approach that models a system as a group of interacting objects.

HOW to do OOAD

OO Technology	Process Perspective
OO Prog Languages Smalltalk,C++	Just a Program
OO Design	Design the Program
OO Analysis	Analyse use case first and then Design

OO Design Patterns

Each Design pattern systematically names, explains and evaluates an important and recurring design in Object oriented systems.

Name: Identifies a particular pattern, creating a vocabulary

Problem: Identifies context when pattern should be applied.

Solution: An abstract description of a design problem along with a template object design that solve the problem

Analysis

Understanding, finding and describing concepts in the problem domain.

Design

Understanding and finding software solution/objects that represent the analysis concepts and will eventually be implemented in code.

- Software development is a dynamic and always undergoing major change.
- System Development refers to all activities that go into producing information system solution.
- System Development activities consist of

  -> System analysis

  -> Modeling

  -> Design

  -> Implementation

  -> Testing and maintenance

OBJECT ORIENTED PARADIGM

- Paradigm: It is a way of seeing and doing things.
- OOP: Object Oriented programming

Organizing software as a collection of objects with certain state and behavior.

- OOD: Object Oriented Design

Based on the Identification & Organization of objects.

- OOM: Object Oriented Methodology

Construction Of Models.

The Development of S/W is a Modeling process.

- OOMD: Object Oriented Modeling and Design.

Modeling Objects based on the Real world.

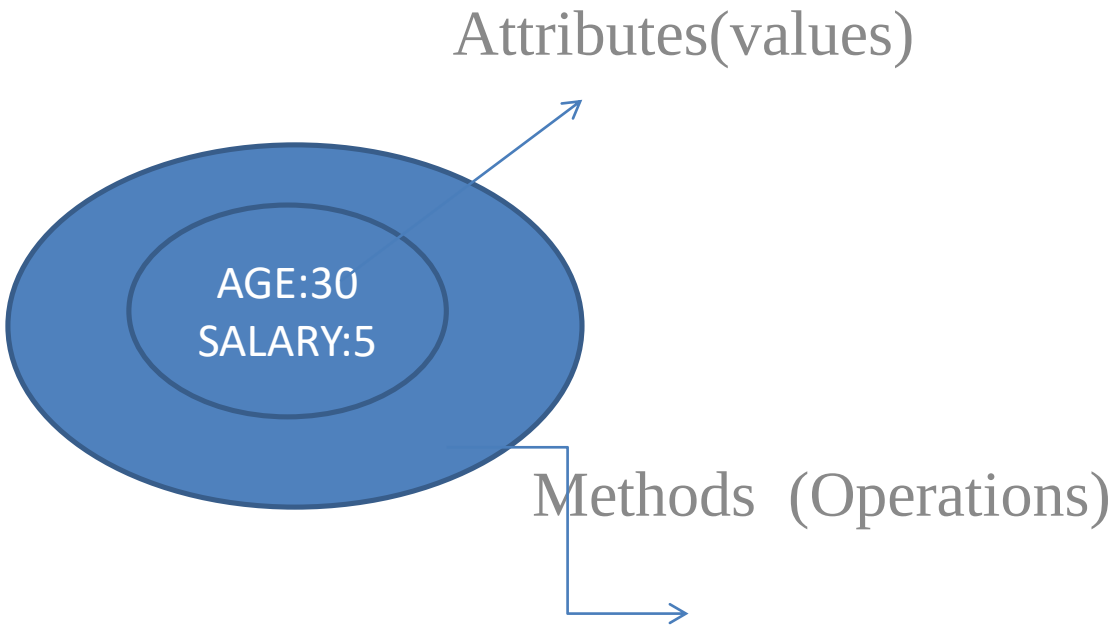
Using these Models to design independently of a programming languages.

OBJECTS:

- Object is a complex data type that has an identity.
- It contains other data type is called attributes.
- The Modules of code called Operations or Methods.
- Attributes and associated values are hidden inside the object.
- Any Object that wants to change or obtain a value associated with other object.
- Sending a message to one of the objects is a invoking method.

OBJECTS

Object: Women



ENCAPSUALTION:

- Each Object methods manage its own attributes is called as hiding.
- An object A can learn about the values of attributes of another object B.
- Example:

Class: Lady

Attributes: Age , salary

Methods: get_age , set_salary

CLASSES:

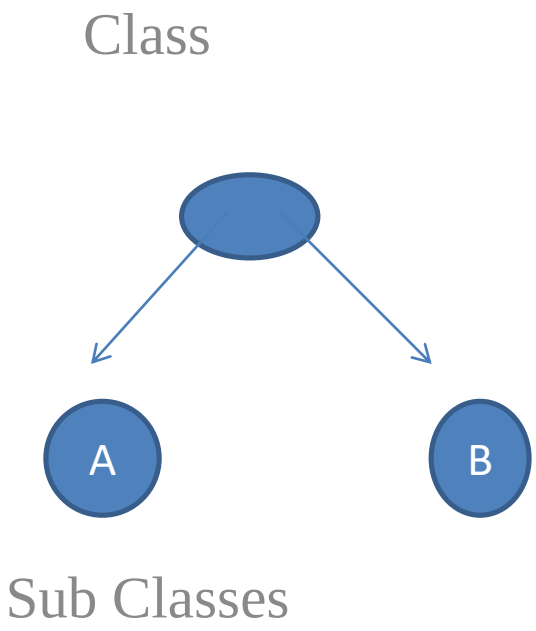
- Classes are templates that have methods and attribute names and type information, but no actual values.
- Objects are generated by these classes and they actually contain values.
- We design an application at the class level.
- When the System is running Objects are created by classes as they are needed to contain state information.

MESSAGE PAASING & AGGREGATION:

- Methods are associated with classes but classes don’t send messages to each other.
- A static diagram(class diagram) shows classes and the logical associations between classes.it doesn’t show the movement of messages.
- An association between two classes means that object of two classes can send messages to each other.
- Aggregation: when an object contains other objects.(a part-whole relationship)

CLASS HIERARCHIES & INHERITANCE:

- Classes can be arranged in hierarchies so that more classes inherit attributes and methods from more abstract classes.



PUBLIC,PRIVATE & PROTECTED:

- Attributes can be public or private:
- Private: It can only be accessed by its own methods.
- Public: It can be modified by methods associated with any class.
- Methods can be public , private or protected:
- Public: Its name is exposed to other objects.
- Private: It can’t be accessed by other objects only internally.
- Protected:(special case) only subclasses that decent directly from a class that

METHOD SIGNATURE:

- It is the method's name and the parameters that must be passed with the message in order for the method to function.

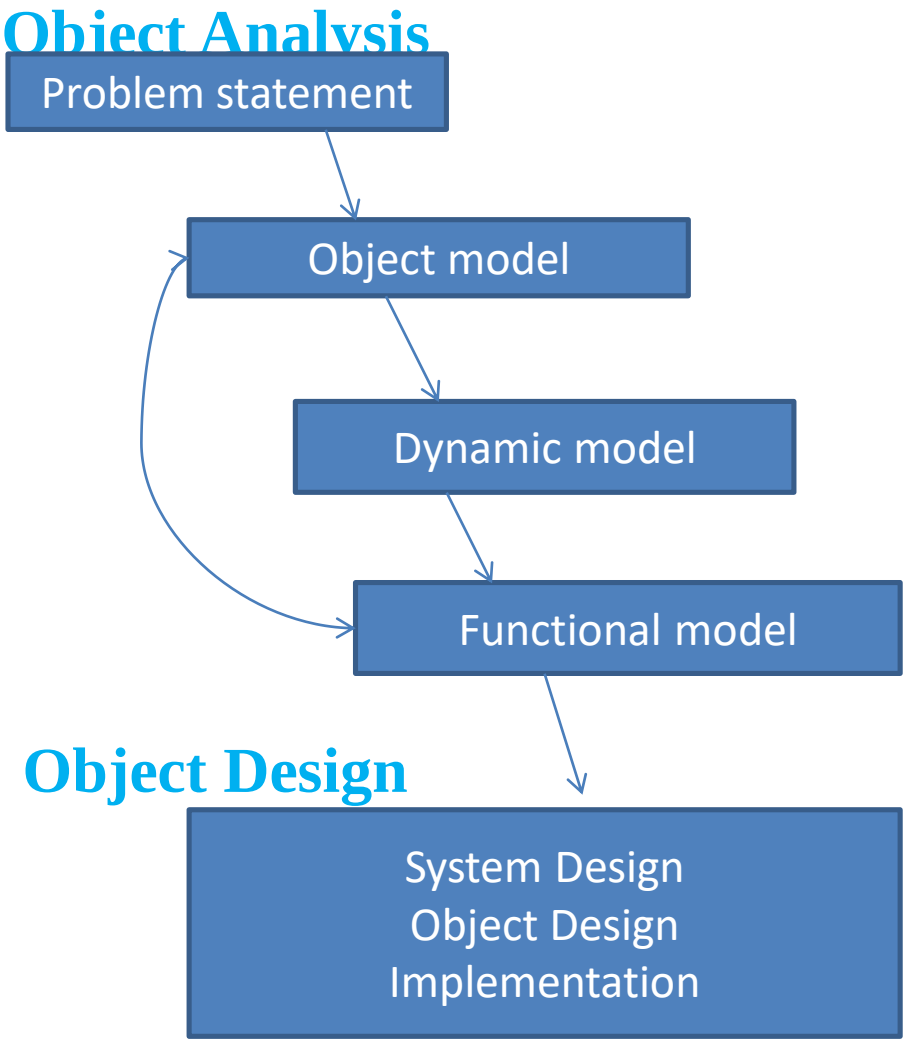
- The parameters are the important because they assure that the method will function properly.

- Compiler or interpreter allow to discriminate between two different methods with the same time.

POLYMORPHISM:

- It means that the same method will behave differently when it applied to the objects of different classes.
- It also means that different methods associated with different classes can interpret the same message in different ways.
- Example: An object can send a message PRINT to several objects, and each one will use it’s own PRINT method to execute the message.

OMT-OBJECT ORIENTED METHODOLOGY



- Object Model:

It describes the static structure of the objects in the system and relationships

->Object diagram

- Dynamic Model:

It describes the interaction among objects in the system->state diagram.

- Functional Model:

It describes the data transformation of the system.

- Analysis:

Model the real world showing the important properties.

- System Design:

Organize into subsystem based on analysis structure.

- Object Design:

Based on analysis model focus on data structure and algorithms to implement each class.

- Implementation:

Translate object classes and relationship into programming languages.

SOFTWARE DEVELOPMENT METHODOLOGY

-> series of processes

-> Can lead to the development of an application.

->Practices, Procedures and rules used to develop software, totally based on system requirements.

ORTHOGONAL VIEWS OF SOFTWARE

- Two Approaches

-> Traditional Approach

-> Object Oriented Approach

- **TRADITIONAL APPROACH**

-> Collection of Programs or Functions.

-> A system is designed for performing certain action.

-> Algorithms+Data structures=Programs.

->Software Development Models(waterfall,Spiral,Incremental)

- **OBJECT ORIENTED APPROACH**

-> Based on Functions and Procedures.

->Software is a collection of discrete object that encapsulate their data as well as fuctionality.

-> Each Object has attributes(properties) and method(Procedures).

-> Objects grouped into the classes and object are responsible for itself.

Difference between Traditional approach and Object oriented approach

TRADITIONAL APPROACH	OBJECT ORIENTED APPROACH
Collection of procedure(Functions)	Combination of data and functionality
Focuses on function and procedures	Focuses on object and Classes modules
Moving from one phase to another phase is complex	Moving from one phase to another phase is easier
Increases duration of project	Decreases duration of project
Increases complexity	Reduces Complexity

UNIFIED PROCESS(UP)

- The Unified Process has emerged as a popular iterative software development process for building object oriented system.
- The UP combines commonly accepted best practices

-> iterative life cycle

-> Risk driven-development.

Key Concepts in UP

- Apply Use cases
- Build Cohesive, Core architecture in early iteration.
- Provides Visual modeling using UML.

UP PHASES

There are 4 Phases in unified Process

- Inception
- Elaboration
- Construction
- Transition

INCEPTION

Inception is the initial stage of the Project. Inception is not a requirements phase but it is a Feasibility phase.

It deals with

-> Approximate vision

-> Business case

-> Scope

ELABORATION

In Elaboration Phase team is expected to capture majority of the system requirements.

It deals with

- Refined vision
- Resolution of high risk
- Identification of more requirements and scope

CONSTRUCTION

Construction phase encompasses on iterative implementation of

- lower risk
- Easier elements
- Preparation of deployment

TRANSITION

Transition phase focus on releasing the final product to the customer for usability.

UP DISCIPLINES

- UP describes work activities such as writing a use case within disciplines a set of activities and related artifacts in one subject within requirement analysis.
- Artifact any work such as code, graphics, text documents, diagrams, models

Several UP Disciplines

- **Business Modeling:**

Domain model artifact to visualize concepts in application domain.

- **Requirements:**

Use case model specification artifact to capture Functional and Non functional requirements.

- **Design:**

All aspects of design including overall

Architecture, objects, database, networks.

UML DIAGRAMS

- UML means Unified Modeling Language
- It is a standard notation for the modeling of real world objects
- UML is a visual language for

- Specifying
- Constructing
- Documenting

Types of UML Diagrams

- Use case diagram
- Class diagram
- Interaction diagram
- Sequence diagram
- Collaboration diagram or communication
- State diagram
- Activity diagram
- Package diagram
- Component diagram
- Deployment diagram

Three ways to apply UML

1. **UML as sketch:**

Informal and incomplete diagrams created to explore difficult parts of the problem.

2. **UML as blueprint:**

Detailed design diagram used for better understanding of code.

3.UML as Programming language:

Complete executable specification of a software system in UML.

Three Perspectives to apply UML

1. Conceptual perspective:

Diagrams describe the things of real world.

2. Specification perspective:

Diagrams describe software abstractions or components with specification and interfaces.

3.Implementation perspective:

Diagrams describe software implementation in a particular technology.

USE CASE DIAGRAM

- USE case diagrams are used to describe a set of actions (use cases) some system or systems should or can perform in collaboration with one or more external uses of the system (actors).

- Each use case should provide some observable and valuable result to the actors or other stakeholders of the system.

Purpose:

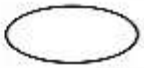
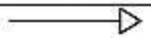
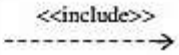
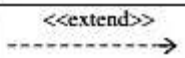
1. 1.Used to gather requirements of a system.
2. 2.Used to get an outside view of a system.
3. 3.Identify external and internal factors influencing the system.
4. 4.Show the interaction among the requirements through actors

USES:

1. Requirwment analysis and high level design
2. Model the context of a system
3. Reverse engineering
4. Forword engineering

NOTATIONS:

Notations:

S.No	Name	Notation	Description
1	**Actor**		Actors are the entities that interact with the system.
2	**System**		The use cases in the system make up the total requirements of the system.
3	**Use Case**		Use Case describes the actions performed by the user.
4	**Generalization**		A generalization relationship is used to represent inheritance relationship between model elements of same type.
5	**Include**		An include relationship specifies how the behavior for the inclusion use case is inserted into the behavior defined for the base use case.
6	**Extend**		An extend relationship specifies how the behavior of the extension use case can be inserted into the behavior defined for the base use case.

SAMPLE EXAMPLE – ATM SYSTEM

Sample Example - ATM System