22BCE04 - OBJECT ORIENTED ANALYSIS AND DESIGN

System Development:-

It refers to all the activities used to produce a solution to information system.

Activities:

1.) Analysis 2.) Modelling 3.) Design
4.) Implementation 5.) Testing 6.) Maintanence

Software Development methodology:-

→ it is a list of procus that leads to dwelopment of application.

→ Here we will focus only on how to carry out the work to achieve the goal based on requirement analysis.

→ 2 Orthogonal views of a software
  i) Traditional s/w approach
  ii) object Oriented s/w approach.

i) Traditional s/w approach:
  → Program → Algorithms + Data structures
  → Focus on functions and Procedures.
  → Complexity of the program → length and time.

ii) object Oriented s/w approach:
  → Program → Self-contained modules which can be easily created, modified and replaced.
  → Focus on class and objects.

Advantages of object Oriented approach:-

1.) Easy to update
2.) Easy to maintain
3.) Robust (speed) and error-free
4.) Reusability of code.

Why we need OOAD:

Reasons for OOA

1.) Higher level of Abstraction
   → Traditional ⇒ Functions & Procedures
   → OOA ⇒ objects, class, Data hiding

2.) Seamless transition among diffel. Phases in s/w develo).
   → In traditional approach we use different methodologies for various Phases (Increases Complexity)
   → OOA ⇒ more (or) less similar methodology.

3.) Good Programming Techniques:
   Comparing to traditional approach Programming is easy in OOA, because we are using classes and objects which are independent.

4.) Reusability of code:
   The code once created can be reused in other classes through the concept of Inheritance which is not Possible in traditional approach.

Unified Approach:

→ for s/w development
→ Top developers: Booch, Rambaugh, Jacobson
→ It is the methodology for s/w development.

Some s/w engineers have suggested the best Practice & Procedures & guidelines for development of object oriented System and combines with UI and developed language called UML.

UML is a language consist of seperate notation and conventions to represent the methodologies needed for s/w development.

Basics of object:-

Traditional approch - 2

1) Algorithm centric:

First we will write algorithms based on that we choose data structures.

2) Data centric:

→ First we choose data structures and then we will write algorithms.

→ Not effective objects

object ⇒ combination of data & logic.

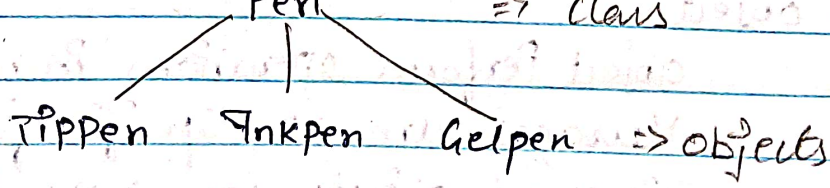
Object / Properties / Procedures
(status) (behaviour)

→ Clauses are used to distinguish one type of object from another.

→ class is a set of objects that share common structure and common behaviour.

→ a single object is simply an Instance of a class.

Ex:



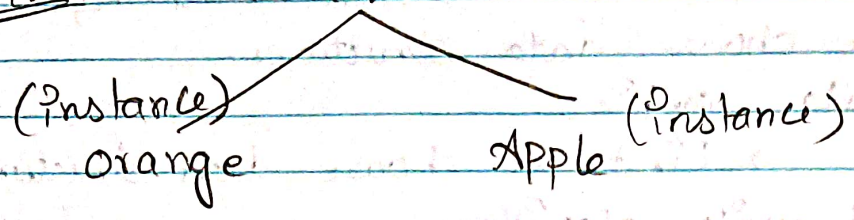
(A class is a set of objects that share common structure & common behaviour).

Class — object — Ex: Person
↳ data — ↳ name, age, gender
↳ logic — ↳ study(), walk(), play()

Instance of a class:-

Each object is an instance of a class.

Ex:



object behavior and methods:-

Behavior denotes the collection of methods that describes what an object is capable of doing.

The object called the receiver on which the method operates.

Methods encapsulate the behavior of the object, hide internal structures and states maintained by the object.

objects respond to messages:

Determined by methods defined for it.

To do an operation a message is sent to an object.

object perform operations in response to messages.

Messages are non-specific function call.

Different objects can respond to the same message in different ways. This is called Polymorphism.

It is receiver's responsibility to respond to the messages on appropriate manner.

Polymorphism is the main difference between message & sub-routine call.

↓

A set of instructions that are used repetedly in a program.

Messages & methods are different
messages => says what to do
methods => says how to do something.

Benefits of OOP:-
Class, object, Polymorphism, Encapsulation, Inheritance

Encapsulation and Information hiding:-
I.s => Principle of hiding the internal data and Procedures of an object and providing an interface to each object.
↳ (means user cannot see the inside of the object "capsule" but can use the object by calling the object-routine

In c++ Public -> accessed from anywhere
Private -> accessed only from with in a class
Protected -> accessed only from subclasses.

=> Rather than allowing an object direct access to another object data, a message is sent to the target object requesting information.
-> so using this ↑ an object's internal format is insulated from other objects.
2 issues:
1.) Per-object: methods can access only the receiver object.

2.) Per-Class: methods can access any objects of the class & not just the receiver.

Data Abstraction:
Incorporates Encapsulation & Polymorphism.

Class Hierarchy:-
→ Superclass (more general)
→ Subclass (most specific)
↳ adds new methods & properties.
→ Code reusability is a benefit.
→ Formal or Abstract classes have no instances but define the common behaviour that can be inherited by more specific classes.

Inheritance:
→ It's a relationship between classes where one class is the parent class of derived class.
→ Allows classes to share & reuse behaviours and attributes.

Dynamic Inheritance:-
It refers to the ability to add/change/delete parents from objects (or classes) at runtime.
(alter the class hierarchy at runtime).

Multiple Inheritance:
Inheriting attributes and methods from more than one Superclass.

Polymorphism:-
→ Poly means "many", morphism means "form".
→ Same operation behaves differently on different classes.

→ It's a relationship of objects of many different classes by some common superclass.
→ Polymorphism allows to write generic, reusable code more easily.

Object Relationship and Associations:-

Association ⇒ Relationship b/w class & object.
⇒ Bi-directional.
Cardinality Constraint ⇒ one, many.
↳ how many instance of one class may relate to a single instance of an associated class.
Consumer-Producer association
↳ one way interaction.

Aggregation and object containment:-

One object refers to other object is called Aggregation. (Relationship blw objects where one object contains another object as a part of its state).
Note:- attributes can be an object itself.

Ex: A 'car' class might have an aggregation relationship with a 'engine' class. The car "has-a." 'Engine" meaning the "Engine" is a part of the "car".

object and Identity:-

Every object has its own unique identity, which is evolved / created when an object is created.
↳ It doesnot depend on object's name, key or location. (all these may change)
Won't change even if all the properties are changed.

→ object identity is implemented through object Identifier (OID) or Unique Identifier (UID)

↓

Responsible for guaranteeing the uniqueness of every identifier

→ OID are never reused.

→ object references directly denote the object to which they refer. → Implemented using UID.

In Object Oriented Systems relationship b/w diff. objects is implemented & maintained by a reference.

Static and Dynamic Binding :-

↓ ↓

Early Binding — Late Binding => determining which function to invoke at runtime

↓

determining which function to invoke at compile time.

Static: => Refers to the process where the compiler resolves the function call during compile-time. (i.e decision on which method to call is made at compile time).

Dynamic: => function call resolved during runtime.

object Persistance:-

→ Lifetime of an object

→ File/DB provide suppor for objects having a longer lifeline. It is called object Persistence.

→ object can exist beyond application session boundaries, during which the object is stored on DB/File.

→ The object can be retrieved in other application session & have relationship to other objects as at the time it was created.

→ Lifetime of the object can be explicitly terminated.

→ Once terminated, its identity is never reused.

Meta class:

→ In object oriented system everything is an object. (like numbers, arrays, records, fields etc.,)

→ A class is also an object

→ Such a class belongs to a class called as Meta-class (or) class of classes.

(All objects are instance of a class and all class are instance of a meta-class)

→ A class contains methods, instances, Parents to perform all the work of being a class. This can be declared in a class called Meta-class.

→ A metaclass is a class and therefore an instance of itself.

→ Meta-class is used by the compiler. Meta-class handle message to class such as constructors "new" and "class variables".

object oriented System Development lifecycle:

→ System Development can be viewed as a process.

→ This process can be divided into small, integrating phases called as Sub-Process.

→ The Sub-process has

1.) Description in terms of how it works

2.) Input required

3.) Output required

S/w is a series of transformations => o/p of one transformation is Input of subsequent transformation.

Transformation 1: User needs in to system requirement.

Transformation 2: Begins with problem statement and ends with detailed design that can be transformed in to an operational system.
→ It includes s/w development activity like how to build the s/w, its development, testing.

Transformation 3: Embedding s/w Product with in the operational Environment.

waterfall model approach:
1.) what 2.) How 3.) Do it 4.) Test 5.) use.

2. ways of testing:
1.) According to how it has been built.
2.) what it should do.

Building High Quality S/w:
1.) Product with minimal or no defect.
2.) ultimate goal of high quality s/w is user satisfaction.

Blum's system evaluation in terms of 4 Quality measures:
1.) Correspondence
2.) Correctness
3.) Verification
4.) Validation

Verification → Am I building the product right?
Validation → Am I building the right product?
Correspondence → Measures how well the delivered system matches the needs of operational environment.
Correctness → Measures the consistency of product requirement with respect to design specification.
Validation → Predicting correspondence
Verification → Determining correctness

Object-Oriented System Development: Use-case driven Approach:

Analysis:

→ Determining system requirement
→ Identify users & actors
→ Scenarios help analyst to understand requirement
→ Identify classes
→ Relationship blw classes.

→ Ivar Jacobson → Use-case model (Scenario to describe user-computer interaction)
→ Interaction among objects role to achieve a given goal is called collaboration.
→ Interaction with customers in a scenario & analyzing it is referred to as use-case modeling.
→ 80:20 rule → 80% of work can be done with 20% of documentation.

Design:

→ Design the classes identified on Analysis phase.
→ we also identify additional objects and classes.

1.) Build object model & their relationship.
2.) Design and refine class, attributes, methods, structures & associations.

Design Guidelines:

1) Reuse existing class
2.) Build / Design large number of simple classes, rather than small complex.
3.) Go back and refine the class.

Prototype:

→ It's a version of s/w product developed
→ Makes use-case modeling much easier.

Types:

1.) Horizontal Prototype:
→ Used during early stages of analysis.
→ gives broad view of application including sample screens, menus, buttons etc., that reflect current requirement.
→ Superficially interactive (means not completed & involves only the most obvious things), so no processing behind them.
→ They may navigate a user to another screen, or show a sample report with dummy data, but the internal functions are not fully implemented.
→ It reflects the breadth of the system without drilling down into too much.

2) Vertical Prototyping:

→ Used on later stages of analysis & design to drill down & elaborate on specific features or functions.
→ more technical.
→ Connect to DB with real data, interface with existing sub-System.

3) Analysis Prototype:

→ Early version of product that is designed with essential functionalities to be tested, measured & observed.
→ Used to analyse the performance & identify potential issues before building the final product.
→ Discarted when it has served its purpose.

4) Domain Prototype:

→ Allows to create a fully working application right after the workshop, which can be discussed & accepted by Stakeholders as iteration artifact.
→ It demonstrates the feasibility of implementation & eventually will evolve in to a deliverable product.
→ prototype dev. time → ~~one~~ few days to Several weeks
→ Involve representation from all user groups.

Purpose of Prototype:

→ To check final specification is appropriate.
→ To collect information about problems.

Implementation:

→ s/w development → CASE tools automate the process by generating automatic coding, but provides only a skeleton of coding & remaining need to be filled by hand.

→ CASE tools begins to support component based development.

→ 2 ideas of CBD is
  ↳ Appl. dvl. are made easy by readymade or pre-fabricated s/w components.
  ↳ large collection of s/w components made available to the developers.

→ CBD appl% are build from readymade components where these components are merged or glued using visual tools or actual code.

Definition: → s/w components are functional units of a program, that provide a collection of reusable services.

→ s/w components interact with each other to complete a task.

OO Development
↓
Analysis
Design
programming

CBD
↓
Implementation &
System Testing

RAD ⇒ Concerned with time reduction to deliver the product.

→ objective is to build the appl. quickly using design and user requirement tools like Delphi, Visualage, Visual Basic or PowerBuilder.

⇒ It (RAD) encourages Incremental devel. approach of "Grow, do not built", s/w.

Reusability:

→ Increased reliability.

→ reduced time & cost for development.

→ Improved consistency.

→ strategy based on Encapsulation etc.,